



البرمجة (2)

البرمجة بلغة JAVA





منشورات جامعة دمشق
كلية الهندسة الميكانيكية والكهربائية
قسم هندسة الحواسيب والأتمتة

البرمجة (2) البرمجة بلغة JAVA

الدكتور

أكرم مذکور

مدرس

في قسم هندسة الحواسيب والأتمتة

الدكتور المهندس

مأمون یونس

أستاذ مساعد

في قسم هندسة الحواسيب والأتمتة

١٤٣٩ - ١٤٤٠ هـ
٢٠١٧ - ٢٠١٨ م

جامعة دمشق



الفهرس

الفصل الأول

أساسيات لغة Java

٢٠	١-١- مقدمة
٢١	٢-١- تاريخ تطور لغة Java
٢٣	٣-١- بيئة لغة Java
٢٨	٤-١- الشكل العام لبنية البرنامج بلغة Java
٣٤	٥-١- متواليات الهروب
٣٥	٦-١- المجاري المعيارية في لغة Java
٣٦	٧-١- الحزمة القياسية util
٣٦	٨-١- استيراد صف من الحزمة
٣٨	٩-١- منصة عمل Java
٣٨	١٠-١- رموز لغة Java
٣٩	١١-١- الكلمات المحجوزة في لغة Java
٣٩	١٢-١- المتحولات والثوابت
٣٩	١٣-١- انواع المعطيات الأساسية في لغة Java
٤١	١-١٣-١- المتحولات
٤١	١-١٣-١-١- تسمية المتحولات
٤٢	١-١٣-١-٢- مجال المتحولات
٤٣	١-١٣-١-٣- التصريح عن المتحولات
٥٠	١-١٣-١-٤- التحويل بين المتحولات
٥٤	١-١٤-١- المتحولات الثابتة

٥٥	١٥-١- العوامل الحسابية والمنطقية والعلائقية
٥٥	١-١٥-١- العوامل الحسابية
٥٦	٢-١٥-١- العوامل العلائقية
٥٧	٣-١٥-١- العوامل المنطقية
٥٧	٤-١٥-١- المعامل الشرطي ثلاثي الحدود
٥٨	٥-١٥-١- عوامل الإسناد المركبة
٦٠	٦-١٥-١- عمليات الزيادة والنقصان
٦٣	١٦-١- أولوية تنفيذ العمليات الحسابية والمنطقية والعلائقية
٦٣	١٧-١- العمليات المنطقية على البت
٦٦	١٨-١- واجهات المستخدم المرئية باستخدام JOptionPane
٦٦	١-١٨-١- التابع showMessageDialog
٦٧	٢-١٨-١- التابع showInputDialog
٦٩	٣-١٨-١- التابع exit(0)
٧٠	٤-١٨-١- التابع showConfirmDialog
٧٥	مسائل عامة

الفصل الثاني

بنى التحكم والتكرار

٧٩	١-٢- مقدمة
٧٩	٢-٢- بنى التحكم
٨٠	١-٢-٢- التعليمية if
٨٢	٢-٢-٢- التعليمية if/else
٨٤	٣-٢-٢- التعليمية if-else-if
٨٦	٤-٢-٢- التعليمية if المركبة

٨٩	٥-٢-٢- بنية التعليمة switch
٩٢	٦-٢-٢- بنية تعليمة التحكم (?:)
٩٦	٣-٢- بنى حلقات التكرار
٩٦	١-٣-٢- بنية الحلقة for
١٠٤	٢-٣-٢- بنية حلقة التكرار while
١٠٩	٣-٣-٢- بنية الحلقة do/while
١١٣	٤-٢- تعليمات التفرع
١١٣	١-٤-٢- تعليمة التوقف break
١١٤	٢-٤-٢- تعليمة الاستمرار continue
١١٥	٣-٤-٢- تعليمة return
١١٦	مسائل عامة

الفصل الثالث

التوابع

١٢١	١-٣- مقدمة
١٢١	٢-٣- فوائد استخدام التوابع
١٢٢	٣-٣- نموذج التابع
١٢٥	٤-٣- استدعاء التوابع
١٣١	٥-٣- مكتبة التوابع الرياضية
١٣٤	٦-٣- التوابع بدون وسطاء
١٣٥	٧-٣- التحميل الزائد للتوابع
١٣٨	٨-٣- التوابع العودية
١٤١	٩-٣- توليد الأعداد العشوائية
١٤٤	مسائل عامة

الفصل الرابع المصفوفات

١٤٨	١-٤- مقدمة
١٤٨	٢-٤- تعريف المصفوفة
١٤٩	٣-٤- المصفوفات أحادية البعد
١٥٠	١-٣-٤- التصريح عن المصفوفات أحادية البعد
١٥٣	٢-٣-٤- إعطاء قيم ابتدائية لعناصر أحادية البعد
١٥٦	٣-٣-٤- إدخال وإخراج عناصر المصفوفة أحادية البعد
١٦١	٤-٤- المصفوفات ثنائية البعد
١٦٢	١-٤-٤- التصريح عن المصفوفات ثنائية البعد
١٦٣	٢-٤-٤- إعطاء قيم ابتدائية لعناصر ثنائية البعد
١٦٤	٣-٤-٤- المصفوفة غير المنتظمة
١٦٦	٤-٤-٤- إدخال وإخراج عناصر المصفوفة ثنائية البعد
١٦٩	٥-٤-٤- تمرير المصفوفات كوسطاء للتوابع
١٧٩	مسائل عامة

الفصل الخامس الصفوف في Java

١٨٩	١-٥- مقدمة
١٩٠	٢-٥- بنية الصف في لغة Java
١٩٩	٣-٥- محددات الوصول إلى أعضاء الصف
٢٠١	٤-٥- تعدد البواني
٢٠٢	٥-٥- الباني الافتراضي
٢٠٣	٦-٥- بناء الصف Point

٢٠٧	٧-٥- بناء الصف Time
٢١٦	٨-٥- بناء المكسد Stack
٢١٩	٩-٥- المزيد حول إنشاء الصف
٢٢١	١٠-٥- الكلمة المحجوزة this
٢٢٨	١١-٥- تمرير الأغراض إلى التتابع
٢٣٠	مسائل عامة محلولة
٢٣٩	مسائل عامة

الفصل السادس

السلاسل والملفات

٢٤٦	١-٦- مقدمة
٢٤٦	٢-٦- الصف String
٢٤٧	١-٢-٦- بناء السلسلة المحرفية
٢٥٠	٢-٢-٦- حساب طول السلسلة واستعادة المحارف المنفصلة
٢٥٢	٣-٢-٦- لصق سلاسل المحارف
٢٥٣	٤-٢-٦- استخراج سلسلة فرعية
٢٥٥	٥-٢-٦- مقارنة السلاسل
٢٦١	٦-٢-٦- توابع تحويلات السلاسل
٢٦٤	٧-٢-٦- البحث عن محرف أو نص جزئي في السلسلة
٢٦٧	٨-٢-٦- تحويل المحارف والقيم العددية إلى نصوص
٢٦٩	٩-٢-٦- فحص الكلمات المتناظرة
٢٧٢	٣-٦- الصف Character
٢٧٩	٤-٦- الصف StringBuffer

٢٨٩	StringTokenizer الصف ٥-٦
٢٩٢	٦-٦ معالجة الملفات في Java
٢٩٢	١-٦-٦ مقدمة
٢٩٣	٢-٦-٦ أنواع الملفات
٢٩٣	٣-٦-٦ أنواع القراءة من الملفات
٢٩٤	٤-٦-٦ التعامل مع الملفات File Operations
٢٩٥	٥-٦-٦ عملية القراءة والكتابة في الملفات
١٩٩	٦-٦-٦ التعامل مع ملفات النصوص Text Files
٢٩٩	١-٦-٦-٦ الكتابة في ملفات النصوص
٣٠٣	٢-٦-٦-٦ القراءة من ملفات النصوص
٣٠٦	مسائل عامة

الفصل السابع

الوراثة

٣١٣	١-٧ مقدمة
٣١٣	٢-٧ الصفوف الأساسية والصفوف المشتقة
٣١٤	١-٢-٧ التحكم بالوصول إلى الصف الأساسي
٣١٦	٢-٢-٤ الصفوف المشتقة
٣٢١	٣-٧ بناء الصف Student
٣٢٦	٤-٧ بناء الصف Box (وراثة)
٣٢٦	٥-٧ تجاوز التابع
٣٣١	٦-٧ الوراثة المتعددة

٣٤١	٧-٧- بناء الصف Box (وراثة متعددة)
٣٤٨	مسائل عامة محلولة
٣٦٢	مسائل عامة

الفصل الثامن

تعدد الأشكال والواجهات

٣٧١	٨-١- تعدد الأشكال
٣٧٢	٨-١-١- الصفوف المجردة
٣٨٠	٨-١-٢- المعطيات الأعضاء والتوابع والثوابت الساكنة
٣٨١	٨-٢- الواجهات Interfaces
٣٨١	٨-٢-١- كتابة الواجهات
٣٨٢	٨-٢-٢- تحقيق الواجهة
٣٨٣	٨-٢-٣- التصريح عن واجهة
٣٨٣	٨-٢-٤- جسم الواجهة
٣٩٣	مسائل عامة محلولة
٤٠٢	مسائل عامة

الفصل التاسع

بناء واجهات المستخدم المرئية

٤٠٧	٩-١- مقدمة
٤٠٧	٩-٢- بعض المكونات المرئية البسيطة
٤٠٨	٩-٣- واجهة برمجة تطبيقات Java GUI
٤٠٩	٩-٣-١- الصفوف الحاوية
٤١١	٩-٣-٢- الصفوف المساعدة
٤١٢	٩-٤- عناصر واجهة المستخدم Swing GUI

٤١٥	٩-٤-١- إطارات Frames
٤١٩	٩-٤-٢- إضافة العناصر إلى الإطار
٤٢١	٩-٤-٣- مدراء التنسيق Layout Managers
٤٢٣	٩-٤-٣-١- مدير التنسيق FlowLayout
٤٢٦	٩-٤-٣-٢- مدير التنسيق GridLayout
٤٣١	٩-٤-٣-٣- مدير التنسيق BorderLayout
٤٣٥	٩-٤-٣-٤- مدير التنسيق CardLayout
٤٣٧	٩-٤-٣-٥- مدير التنسيق GridBagLayout
٤٤١	٩-٤-٣-٦- مدير التنسيق BoxLayout
٤٤٣	٩-٥-٥- الرسم البياني Graphics
٤٤٤	٩-٥-١- إعادة رسم المكونات
٤٤٥	٩-٥-٢- رسم الخطوط والرسومات الأخرى ضمن اللوحات
٤٤٨	٩-٥-٢-١- الصف Color
٤٥٥	٩-٥-٢-٢- الصف Font والصف FontMetrics
٤٦٣	٩-٥-٣- رسم الأشكال الهندسية
٤٦٤	٩-٥-٣-١- رسم الخطوط
٤٦٧	٩-٥-٣-٢- رسم المستطيلات
٤٧١	٩-٥-٣-٣- رسم الأشكال البيضاوية
٤٧٤	٩-٥-٣-٤- رسم الأقواس
٤٧٧	٩-٥-٣-٥- رسم المضلعات والخطوط المتعددة
٤٨١	٩-٥-٤- التحكم بمظهر الشكل المراد رسمه
٤٨٦	٩-٥-٤-١- الصف BasicStroke
٤٩٢	٩-٥-٤-٢- التحكم بالرسم من خلال التدوير

٤٩٤		مسائل عامة محلولة
٥٠٢		مسائل عامة

الفصل العاشر

معالجة الأحداث

٥٧٣		١-١٠ البرمجة المقادة بالحدث
٥٧٣		٢-١٠ كتابة معالجة الحدث
٥٧٣		٣-١٠ الصف الداخلي و الصف المجهول
٥٧٤		٤-١٠ كتابة معالج الحدث للعنصر JTextField
٥٧٦		٥-١٠ كتابة معالج الحدث للعنصر JButton
٥٧٩		٦-١٠ معالج حدث لمكونات الاختبار
٥٧٩		٧-١٠ صفوف الـ Adapters للأحداث
٥٨١		٨-١٠ التعامل مع أحداث لوحة المفاتيح والفأرة
٥٨٦		٩-١٠ استخدام العنصر Timer والعنصر Border
٥٨٦		١٠-١٠ استخدام العنصر JcolorChooser
٥٨٦		١١-١٠ استخدام العنصر JTabbedPane
٥٥٢		مسائل عامة

الفصل الحادي عشر

معالجة الاستثناءات

٥٧٣		١-١١ مقدمة
٥٧٣		٢-١١ الاستثناءات وأنماطها
٥٧٣		١-٢-١١ صفوف الاستثناءات
٥٧٤		٢-٢-١١ Checked & Unchecked الاستثناءات
٥٧٦		٣-١١ التعرف على كيفية معالجة الاستثناءات

٥٧٩	١١-٣-١- التصريح عن الاستثناءات
٥٧٩	١١-٣-٢- رمي الاستثناءات
٥٨١	١١-٣-٣- النقاط الاستثناءات
٥٨١	١١-٤- إعادة رمي الاستثناءات
٥٨١	١١-٥- العبارة finally
٥٨١	١١-٦- متى نستخدم الاستثناءات

الفصل الثاني عشر

تعددية المسالك

٥٧٣	١٢-١- مقدمة
٥٧٣	١٢-٢- مفهوم تعددية المسالك
٥٧٣	١٢-٣- إنشاء المسالك من خلال وراثة الصف Thread
٥٧٤	١٢-٤- إنشاء المسالك من خلال تحقيق الواجهة Runnable
٥٧٦	١٢-٥- التحكم بالمسالك والاتصالات بين المسالك
٥٧٩	١٢-٦- حالات المسلك
٥٧٩	١٢-٧- مجموعات المسالك
٥٨١	١١-٨- التزامن Thread Synchronization
٥٨١	١٢-٩- التحكم بالتحريك باستخدام المسالك
٥٨١	١٢-١٠- التحكم بالتحريك باستخدام الصف Timer
٥٥٢	مسائل عامة
٦٣٣	المراجع
٧٣٧	المصطلحات العلمية

مقدمة

إن توسع استخدام النظم الحاسوبية وتطورها في السنوات الأخيرة ، فرض وجود شروط محددة على بنية الحواسيب ، وذلك من خلال المعدات والبرمجيات والتطبيقات الحاسوبية ولغات البرمجة التي تم تصميمها للاستخدام على نطاق واسع ، لتتوسع تطبيقاتها في مختلف مجالات الحياة. حيث إن اتساع مجالات استخدام الحواسيب وانتشارها ، وزيادة تطور علوم صناعة البرمجيات دأى إلى ظهور العديد من لغات البرمجة عالية المستوى ، والتي يستطيع الدارس أن يتعلمها وفقاً لحاجته واختصاصه . فظهرت مجموعة من لغات البرمجة عالية المستوى وقرينة من لغة الإنسان مثل :

BASIC, FORTRAN ,Pascal, PROLOG,C\C++,Java ,C#,...
وتُعد لغة Java من لغات البرمجة عالية المستوى ، كما أنها تعد لغة غرضية التوجه (البرنامج المكتوب عبارة عن صفوف و تستخدم الخواص المتاحة من التغليف و تعددية الأشكال و الوراثة والتركيب...).
وأصبحت لغة Java حالياً اللغة القياسية من أجل التطبيقات المستقلة عن المنصات ، ومن أجل برمجة شبكة الويب العالمية ، وتُعد لغة Java لغة كاملة المواصفات ذات أهداف عامة قادرة على تطوير التطبيقات ذات المهام الحرجة.
لقد انتشرت لغة Java بشكل واسع وسريع ، وهي لا تستخدم اليوم من أجل تطبيقات الويب فقط ، بل تستخدم أيضاً في تطوير تطبيقات قائمة بحد ذاتها ومستقلة عن المنصة ، على المخدمات والزيائن وأجهزة المحمول . ومن النادر حالياً أن تجد شركة لا تعتمد على لغة Java في تطوير وتسهيل تدفق عمل مشاريعها المختلفة.

والهدف الأساسي لهذا الكتاب هو تغطية منهاج مقرر " البرمجة (2) " في قسم هندسة الحواسيب والأتمتة لطلاب السنة الثانية. وقد تم وضع مادة الكتاب بحيث تمكن مختلف القراء من الاستفادة منه في مختلف التطبيقات . ويتكون الكتاب من أحد عشر فصلاً ومسرد للمراجع الأجنبية والعربية والمصطلحات العلمية .

يغطي الفصل الأول أساسيات البرمجة في لغة Java ، وذلك من خلال التعرف على الخطوات اللازمة لكتابة برنامج بلغة Java وتنفيذه والقواعد الواجب إتباعها والتعرف على أنواع المعطيات والعمليات الحسابية والمنطقية . ويخصص الفصل الثاني للتعرف على بنى التحكم وحلقات التكرار وأنواعها وكيفية استخدامها و متى تُستخدم واحدة دون الأخرى.

ويغطي الفصل الثالث التوابع (الطرائق) من حيث التصريح عنها وتعريفها وكيفية استدعائها لتنفيذ بعض المهام ، بالإضافة إلى عملية ربط التوابع مع المصفوفات.

ويتناول الفصل الرابع المصفوفات أحادية البعد والمصفوفات ثنائية البعد وكيفية التصريح عنها والتعامل معها من حيث إدخال قيماً لعناصرها وطباعة هذه القيم وأيضاً كيفية استخدام المصفوفات لإجراء بعض العمليات الحسابية ، وتمرير المصفوفات إلى التوابع كوسطاء لها .

أما الفصل الخامس فيتناول الميزة الأولى في البرمجة غرضية التوجه ، وهي الصفوف ، وباستخدام وسيلة الصف فإن البرامج الكبيرة ستكون واضحة المعالم، حيث تكون جميع البرامج ذات العلاقة ، في مواقع محددة لها ، وهذا الأسلوب يؤدي إلى مفهوم الوحدات المتكاملة ، و يؤدي بالتأكيد إلى الصيانة الكفؤة وتسهيلها .

كما يتناول الفصل السادس السلاسل والملفات حيث يوضح كيف تتم معالجة السلاسل وفتح ملف واغلاقه وقراءة ملف وكتابتة .

وببحث الفصل السابع في الميزة الثانية للبرمجة غرضية التوجه وهي الوراثة والاستفادة من الخصائص المعرفة في برامج أخرى سابقة ، ويطلق مصطلح الوراثة على العملية التي يكتسب غرض ما من خلالها على خصائص غرض آخر .

والفصل الثامن خُصص لموضوع تعدد الأشكال في البرمجة غرضية التوجه والواجهات Interfaces حيث يتم استخدام اسم التابع نفسه لأغراض متعددة ، ويبحث في الصفوف المجردة والصفوف التي تُحقق أكثر من واجهة .

ويُغطي الفصل التاسع واجهات المستخدم المرئية من أجل رسم واجهة المستخدم المناسبة، ومعالجة عناصر الرسم ، ورسم الأشكال الهندسية المناسبة .

أما الفصل العاشر فيغطي معالجة الأحداث وفي الفصل الحادي عشر معالجة الاستثناءات ويتناول الفصل الثاني عشر تعددية المسالك .

وختاماً نأمل أن نكون قد وضعنا للقارئ المبادئ الأساسية للانطلاق في مجال البرمجة غرضية التوجه بلغة Java ، وأسهمنا بتقديم المادة العلمية المتواضعة لكوادرنا الهندسية لتساعدكم في تطوير معلوماتهم ، بحيث تتماشى مع التطور الكمي والنوعي في مجال هندسة الحواسيب والأتمتة .

ولا يفوتنا أن نتقدم بالشكر والعرفان إلى جميع الزملاء الذين أسهموا في إبراز هذا العمل إلى حيز الوجود .

والله ولي التوفيق

المؤلفون

دمشق / آب / 2017



الفصل الأول

أساسيات لغة JAVA

Basics of Java language





أساسيات لغة JAVA

١-١ - مقدمة

يقوم المبرمجون بكتابة برامجهم الحاسوبية بلغات برمجية مختلفة، بعضها يكون مفهوماً مباشرة من قبل الحاسوب وبعضها الآخر يحتاج إلى خطوات وبرامج ترجمة ، ويتوافر حالياً عشرات من لغات البرمجة المستخدمة والتي يمكن تقسيمها إلى نوعين رئيسيين:

١- لغات البرمجة منخفضة المستوى (LLL) .

٢- لغات البرمجة عالية المستوى (HLL).

يستطيع الحاسوب أن يفهم وبشكل مباشر لغة الآلة الخاصة به فهي اللغة الطبيعية بالنسبة له. وتتألف لغة الآلة بشكل عام من سلاسل من الأصفار (0) والواحدات (1) التي تعطي الأوامر للحاسوب من أجل تنفيذ تعليماته الأولية، كل تعليمة على حدة ، ولا بد من الإشارة إلى أن لغة الآلة ترتبط ارتباطاً وثيقاً بالآلة (الحاسوب) ، أي بمعنى آخر إن لكل نوع من أنواع الحواسيب لغة آلة خاصة به، وتعدّ لغة الآلة من اللغات الصعبة والمرهقة للمبرمجين و تستغرق وقتاً طويلاً من أجل كتابة أي برنامج، ولا يستطيع كتابة البرامج بلغة الآلة إلا الشخص الذي لديه إلمام كامل ببنية الحاسوب.

ومع تزايد استخدام الحواسيب ومن أجل تسريع عملية البرمجة وتبسيطها تم تطوير لغات برمجية جديدة سُميت لغات البرمجة عالية المستوى HLL (High Level Languages) ، وسُميت البرامج التي تقوم بتحويل البرامج المكتوبة بلغات البرمجة عالية المستوى إلى لغة الآلة باسم المترجمات compilers وهناك بعض اللغات التي تستخدم نوعاً آخر من البرمجيات تسمى

المفسرات interpreters وهي تقوم بتنفيذ البرامج المكتوبة بلغات برمجية عالية المستوى تعليمية تعليمية دون الحاجة لترجمة كامل البرنامج كما تفعل المترجمات. تساعد لغات البرمجة عالية المستوى المبرمجين على كتابة تعليمات قريبة من لغة الإنسان ، وقد أصبحت لغات البرمجة عالية المستوى أكثر تقبلاً من قبل المبرمجين ، وتعد اللغات C++ و Java و C# من أكثر اللغات البرمجة عالية المستوى قوة وانتشاراً.

١-٢- تطور لغة Java

في عام 1960 صُممت لغة CPL (Combined Programming Language) ، وتم ذلك في جامعة لندن وقد استخدمت فيها بعض تعليمات لغة ALGOL-60، وفي عام 1967 انبثقت لغة BCPL من لغة CPL ، وفي عام 1972 قامت مخابر شركة AT&T الأمريكية باستنباط لغة جديدة من لغة B أخذ منها أحسن تعليماتها وأضاف إليها أوامراً جديدة وأنواعاً جديدة للمعطيات ، وكثيراً من التوابع التي تفيد المبرمج وسُميت هذه اللغة بلغة C . ومنذ ذلك التاريخ أخذت لغة C شهرة واسعة لأنها أصبحت تنتمي إلى اللغات البرمجية العالية المستوى High Level Languages من حيث سهولة الاستخدام من ناحية ومن ناحية ثانية تنتمي إلى اللغات البرمجية المتدنية المستوى من حيث قدرة اللغة على مخاطبة مكونات الحاسوب العادية Hardware. بعد ذلك تم تطوير لغة البرمجة C++ التي تُعد امتداداً للغة C وقد قدمت لغة C++ العديد من الإمكانيات التي تسمو بلغة C وتعلو بها، ولكن الأهم من ذلك كله قدرات البرمجة غرضية التوجه OOP (Object Oriented Programming).

تم تطوير لغة Java من قبل فريق من المهندسين من شركة Sun Microsystems في عام 1991 وكانت تحمل الاسم Oak لتستخدم في تضمين التطبيقات الالكترونية للاستهلاك ، وفي عام 1995 أعيدت تسمية هذه اللغة لتصبح Java كما أعيد تصميمها لتستخدم في تطوير تطبيقات الانترنت ، حيث من الممكن لتطبيقات جافا أن تُضمن في صفحات HTML ثم تُحمل باستخدام مستعرضات الويب لتحقيق عملية التحريك الحية ، والتفاعل المباشر مع مستخدمي الويب ولا تنحصر قوة جافا في تطبيقات الويب بل هي لغة برمجة ذات أهداف عامة ، حيث تمتلك صفات برمجية كاملة ويمكن استخدامها في بناء تطبيقات قائمة بحد ذاتها، وعلى الرغم من أن العديد من اللغات غرضية التوجه بدأت بشكل كامل كلغات إجرائية فإن لغة جافا تم تصميمها من البداية لتكون غرضية التوجه. صممت لغة جافا أساساً كي تكون لغة مستقلة عن محيط التشغيل ، وقد

عبرت عن ذلك شركة Sun Microsystems بمبدأ (Write Once , Run Evreywhere) أي " اكتب لمرة واحدة ، وشغل أينما تريد " . ويتطلب تجسيد هذا المبدأ تعريف وتحقيق ما نسميه بآلة جافا الافتراضية Java Virtual Machine أو اختصاراً JVM وهذه الآلة كما يوحي اسمها ، هي آلة افتراضية تحاكي عمل معالج له تعليماته الخاصة ، بحيث يسمح تحقيق هذه الآلة على منصات عمل متنوعة بنقل برامج جافا (التنفيذية) من محيط تشغيل إلى محيط آخر وتشغيلها دون الحاجة لتعديلها . وقد بادرت شركة Sun Microsystems إلى تحقيقها على أنظمة التشغيل الأخرى ، وسعيًا وراء تحويل هذه اللغة إلى لغة عالمية قياسية . طورت لغة جافا بهدف تحقيق أعلى درجة أمان ممكنة ، كما تُعد من اللغات التي تسهل قراءتها وتعلمها ، وهي في كل الأحوال أبسط وأسهل من لغة C++ ويمكننا أن نكتب بلغة جافا كل ما يُكتب بلغة C++ ، وربما باستخدام أدوات مختلفة لكنها أبسط وليست بالضرورة أضعف أداء .

١-٣- بيئة لغة Java

تُعَدُّ البرمجة غرضية التوجه طريقة قوية للتعامل مع المسائل البرمجية ، فقد ظهرت في كل نقطة من مراحل تطورها طريقة جديدة تساعد المبرمجين على التعامل مع البرامج متزايدة التعقيد . على الرغم من أن البرمجة البنوية كانت تعطي نتائج ممتازة عندما تطبق على برامج متوسطة التعقيد ، إلا أنها كانت تفشل عند بعض النقاط حين يصل البرنامج إلى حد معين . ولكتابة برامج أكثر تعقيداً كانت الحاجة إلى طريقة جديدة في العمل البرمجي وتلبية لهذه الحاجة ظهرت البرمجة غرضية التوجه واختصاراً " OOP " . تأخذ البرمجة OOP أفضل ميزات البرمجة البنوية وتضم إليها مبادئ قوية وجديدة تسمح بتنظيم البرامج بفعالية أكبر ، وتعتمد البرمجة OOP على تحليل المشكلة إلى أجزائها الأساسية بحيث يصبح كل عنصر أو جزء عبارة عن غرض object محتوى ذاتياً يضم تعليماته الخاصة والمعطيات العائدة إليه ، وأصبح المبرمج يستطيع إدارة برامج أكبر . تشترك جميع لغات البرمجة OOP بما فيها C++ و Java بثلاث سمات مميزة مشتركة وهي :

١. التغليف Encapsulation (الصفوف Classes) .

٢. الوراثة Inheritance .

٣. تعدد الأشكال Polymorphism .

تُعَدُّ لغة Java من اللغات القوية جداً في مجال إنشاء التطبيقات المختلفة سواء كانت هذه التطبيقات تعمل منفردة على أجهزة الحاسوب الشخصي أو تطبيقات الإنترنت أو التطبيقات المختلفة للأجهزة المحمولة ، مثل الموبايل والمفكرات الإلكترونية وغيرها . إن لغة Java ليست مجرد مفردات برمجة وتراكيب نحوية ، وإنما هي لغة منتخبة كأفضل أداة برمجة لتطبيق مبادئ هندسة

البرمجيات ، إضافة إلى أنها لغة تطوير متكاملة يمكننا استخدامها لبناء العديد من التطبيقات مثل :

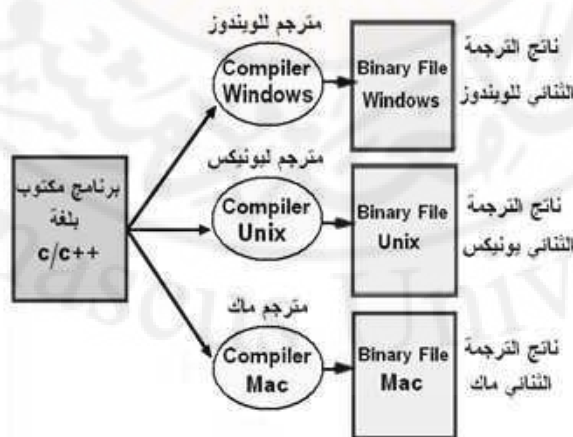
- تطبيقات سطح المكتب Desktop Applications
- التطبيقات الشبكية Client/Server Applications
- تطبيقات الويب Web-Based Applications
- التطبيقات الموزعة Distributed Applications
- تطبيقات المحمول Mobile Applications

ولغة Java هي لغة بسيطة ، غرضية التوجه ، وموزعة ، ومفسرة ، وقوية ، وأمنة ، وذات بنية محايدة ، ومحمولة ، وذات أداء عالٍ ، ومتعددة المسالك وديناميكية . تشبه لغة جافا لغة C++ المبسطة ، ما يقلل من وقت تعلمها ، وإن لغة C++ لغة هجينة (غرضية وبنوية) بينما لغة جافا غرضية التوجه OOP تماماً . وتحقق جافا معظم مبادئ هندسة البرمجيات مثل إعادة الاستخدام وقابلية التعديل ، والفعالية ، والوثوقية ، والتغليف ، وجعلت جافا من انتقال الفيروسات أمراً مستحيلاً ، فهي لا تسمح بإنشاء مؤشرات خارج ترميزها الخاص ، إذ أن المؤشرات الخارجية التي لا تسمح بها جافا هي التي تساعد على عمل الفيروسات ، فالضرر يحدث عندما يخرج الترميز عن مساحة الذاكرة الخاصة به ، حيث إن تطبيقات جافا لا تتفاعل مباشرة مع المعالج أو نظام التشغيل ، فبيئة تشغيل جافا تعالج مسائل الذاكرة ذاتياً بحيث لا يحتاج للقيام بتخصيص الذاكرة ، أو تفريغ الشيفرة منها ، كذلك لا حاجة لاستخدام المؤشر Pointer والذي يعد أهم مصادر الأخطاء في C++ ، فتقدم جافا حلاً جذرياً وتلقائياً لمشكلة الذاكرة التي نصادفها في برامج C++ كما تستبدل جافا التوريث المعقد المتعدد في C++ بالواجهات Interfaces ، ومن أهم ميزاتها :

□ لغة جافا غير مرتبطة بأنظمة التشغيل المختلفة

Java Is Platform Independent

معنى ذلك إنه يمكن نقل البرامج (المكتوبة بلغة جافا) بسهولة من نظام تشغيل إلى آخر، وفي المقابل يمكن القول إنه لا يمكن تشغيل برنامج WORD مثلا والخاص بنظام تشغيل ويندوز (WINDOWS) على جهاز حاسوب آخر يعمل بنظام تشغيل مختلف مثل يونيكس (UNIX) أو نظام تشغيل لينكس (LINUX) أو أي نظام تشغيل آخر غير نظام WINDOWS والمستخدم مع أجهزة الحواسيب المختلفة، ويرجع ذلك لأن برنامج WORD بشكل عام، مكتوب بلغة (C/C++)، والتي تعطي ملف من نوع EXE خلال عملية الترجمة COMPILATION وبذلك يكون الملف الناتج مرتبطاً ارتباطاً كلياً بنظام التشغيل. أما بالنسبة للغة جافا فالوضع مختلف حيث يوجد وسيط بين البرنامج وبين نظام التشغيل وهذا الوسيط يسمى (Byte Code Interpreter) أي الترجمة على مستوى البايت، وكذلك يمكن تسميته بالآلة الافتراضية للجافا (Java Virtual Machine)، ويوضح الشكل (١-١) خطوات تشغيل برنامج مكتوب بلغة (C أو C++).

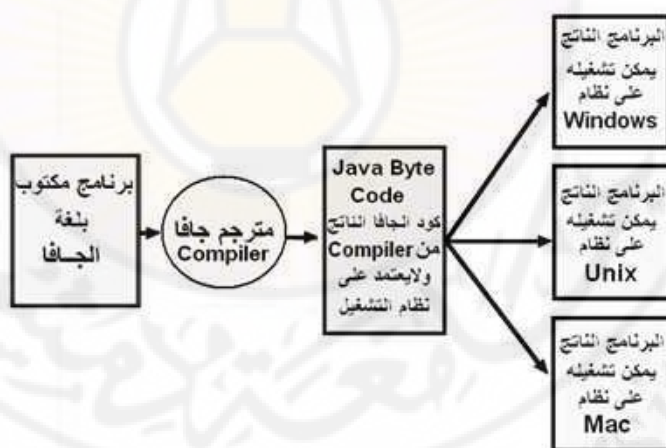


الشكل (١-١) - يبين خطوات تشغيل برنامج مكتوب بلغة (C أو C++)

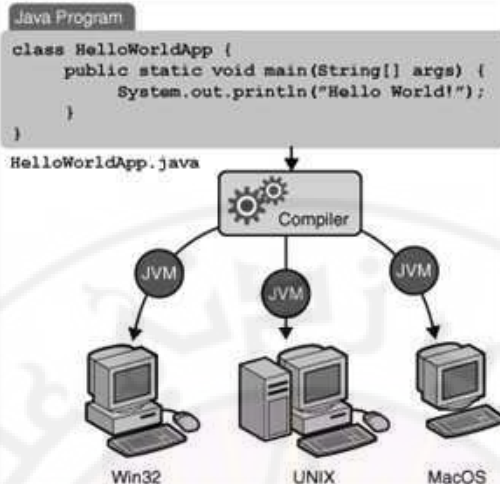
كما يوضح الشكل (٢-١) خطوات تشغيل برنامج مكتوب بلغة جافا . ويوضح الشكل (٣-١) شيفرة البرنامج الناتج من مترجم لغة جافا . ويوضح الشكل (٤-١) أن البرنامج الناتج يمكن تشغيله على نظام win32 و نظام UNIX و نظام Mac .



الشكل (٢-١) - يبين خطوات تشغيل برنامج مكتوب بلغة جافا



ويوضح الشكل (٣-١) - يوضح شيفرة البرنامج الناتج من مترجم جافا



الشكل (١-٤) - يوضح يمكن البرنامج الناتج تشغيله على نظم تشغيل مختلفة

□ إنشاء برامج ذات واجهة مستخدم رسومية (بيانبة)

يُعد بناء واجهة المستخدم الرسومية من الأجزاء المهمة في البرنامج حيث إن هذه الواجهات تعطي البرنامج شكلاً معيناً ، كما أن استخدام مفاهيم وأجزاء موحدة في بناء الواجهات للعديد من البرامج المختلفة يعطي المستخدم قدراً كبيراً من الراحة أثناء استخدام البرامج ، كما أنه يقلل كثيراً من الوقت المستخدم لتعلمها . إن الأجزاء الرسومية الموجودة في لغة جافا مرتبطة مباشرة مع الإمكانيات الرسومية للجهاز الذي يعمل عليه البرنامج . وبذلك فإن الواجهات الرسومية الموجودة في جافا سوف تظهر بأشكال متباينة على الأجهزة المختلفة . أي أننا عندما نقوم بكتابة برنامج يقوم بعمل " زر " على نظام الويندوز فإن هذا الزر يأخذ شكل الزر نفسه المستخدم في نظام الويندوز . ولكن إذا تمت كتابته في نظام تشغيل آخر فإنه يأخذ شكل يتناسب مع نظام التشغيل المستخدم .

□ تصميم برمجيات تستفيد من ميزات الإنترنت Java Applet كلها

هي نوع من التطبيقات التي صممت خصيصاً للإنترنت ، حيث يقوم المطور (Developer) بإعداد برنامج Applet ثم يتم استدعاؤه من خلال ملف

HTML بشرط تحميل برنامج Applet على المخدم (Server) الموجود عليه ملف HTML. فيتم عرض هذا التطبيق من خلال صفحة الانترنت عندما يستعدي المستخدم هذه الصفحة.

لقد قامت شركة Sun (Sun Microsystems) باختراع وتطوير هذه اللغة، وأصبحت شركة Sun مملوكة لشركة أوراكل وبالتالي انتقلت ملكية Java لأوراكل وكان الهدف عند اختراع لغة Java هو عمل لغة قادرة على برمجة نظم التشغيل لجميع الأجهزة من حواسيب عملاقة (mainframes) إلى الأجهزة الصغيرة مثل مشغلات MP3 ولقد اختارت الشركة صورة فنجان القهوة لتمثيل هذه اللغة .

١-٤ - الشكل العام لبنية البرنامج بلغة Java

يتكون أي برنامج مكتوب بلغة Java من أربعة عناصر أساسية هي:

- ١- استدعاء المكتبات المستخدمة في البرنامج.
 - ٢- تعريف المعطيات المستخدمة في البرنامج.
 - ٣- تعليمات (عبارات) البرنامج.
 - ٤- توابع (دوال) المستخدم إن وجدت فهي تُكتب ضمن التابع main .
- الشكل العام لبنية برنامج بلغة Java:

```
public class ClassName
{
    // main method begins execution of Java application
    public static void main( String args[ ] )
    {
        body { Body of main( )
    } // end method main
} // end class Name
```

وتحتوي لغة Java على مجموعة من المكتبات، كل مكتبة فيها مجموعة من الصفوف وكل صف يحتوي على مجموعة من التوابع وكل تابع له وظيفة معينة يؤديها داخل البرنامج. لنكتب أول برنامج بلغة Java والذي يقوم بطباعة العبارة :

Welcome to Java Programming !

1. *// This program will display a message on the screen*
2. **public class Welcom1**
3. **{**
4. *// main method begins execution of Java application*
5. **public static void main(String args[])**
6. **{**
7. **System.out.println("Welcometo Java Programming!");**
8. **} // end method main**
9. **} // end class Welcome1**

وعند تنفيذ البرنامج السابق سنحصل على الخرج التالي:

RESULT

Welcome to Java Programming!

Process completed.

في هذا المثال تم تعريف صف اسمه Welcom1 وهذا الصف عام public ويحمل اسم الملف ، ويبدأ الصف بالقوسين " { " ، " } " وينتهي بهما، لا ينفذ التطبيق دون وجود التابع main() وهو من نوع public static void main(string [] args) ويجب أن يكون التابع main مسبقاً بالكلمة public وإلا لن يتم تنفيذ البرنامج .

شرح البرنامج:

السطر 1 : comment

الذي يبدأ بالإشارة (الشرطة المائلة) // على أن النص المكتوب خلف هذه الإشارة هو عبارة عن تعليق ، حيث يحق للمبرمجين إدراج تعليقاتهم ضمن نص البرنامج وتحسين القدرة على قراءته فيما بعد وخصوصاً من قبل أشخاص آخرين ، وتساعد التعليقات على فهم البرنامج، وهي لا تتطلب من الحاسوب القيام بأي فعل (تعليمية غير تنفيذية) عند تنفيذ البرنامج ، فالمترجم يقوم بتجاهل جميع أسطر التعليقات أثناء عملية الترجمة. تستخدم الشرطة المزدوجة (//) إذا كان التعليق يمتد لسطر واحد فقط single-line comment ، وهناك نوع آخر من التعليقات يتيح لنا كتابة تعليقات تمتد إلى عدة أسطر multi-line comments ، ونستطيع كتابة التعليق السابق كما يلي:

```
/*This program will display a message on the screen */
```

يبدأ الرمز " /* " ثم التعليق وينتهي بالرمز " */ " ، ونجد أن نهاية السطر لا تعني انتهاء التعليق لذا يمكننا كتابة ما نشاء من أسطر التعليقات قبل الانتهاء بالرمز " /* " .

يبدأ البرنامج (السطر 2) بالكلمة class التي تعني تعريف صف جديد ليضاف إلى صفوف java ، وهذا الصف عام public ، يلي هذه الكلمة اسم الصف Welcom1 الذي هو اسم البرنامج (اسم الملف) في الوقت نفسه ، ويجب الأخذ بعين الاعتبار الملاحظات التالية :

- يتضمن كل برنامج صفاً واحداً على الأقل .
- يجب أن يحمل هذا الصف اسم الملف نفسه الذي نحفظ فيه البرنامج .
- تميز لغة جافا بين الأحرف الكبيرة والصغيرة .

- يجب أن يبدأ اسم البرنامج بحرف كبير .

الكلمة المحجوزة **public** : يقوم هذا المحدد بجعل الصف قابلاً للاستخدام من قبل أي صف آخر بغض النظر عن الحزمة التي ينتمي إليها ، وإذا لم نستخدم المحدد **public** لهذا الصف فإنه بشكل افتراضي يمكن استخدام الصف فقط من قبل صفوف أخرى موجودة في الحزمة نفسها .

السطر 5 يمثل التابع **main()** وهو من نوع **void** ، يعيد أي قيمة عند تنفيذه ، ولا يأخذ التابع **main()** أي وسيط ، أما البرنامج الحقيقي فيأخذ وسيطاً اسمه **args** من النمط **string[]** ، وإن التابع **main()** لن يستخدمه ولن نستطيع تنفيذ البرنامج بدونه . وتمثل كل من الكلمتين **public** و **static** صفة من الصفات المميزة للتابع ، فالصفة **public** تعني أنه بإمكان مفسر جافا والصفوف الأخرى استدعاء هذا التابع ، وبدونه لا يستطيع المفسر تنفيذ التابع ، وإن الوظيفة الأساسية للمفسر تنفيذ التابع **main()** ، و يجري غالباً التصريح عن وسيط التابع **main** بالصيغة :

✓ **public static void main(string [] args)**

✓ **public static void main(string args [])**

إن الصيغتين مقبولتان ، وليس ضرورياً أن نطلق الاسم **args** على هذا الوسيط ، ويمكن استخدام أي اسم وسيط آخر شرط ألا يبدأ برقم .

السطر 7 : هو تابع الطباعة **println** الذي يأخذ وسيطاً من النمط **string** لتظهر النتيجة على مجرى الخرج القياسي مسبقاً بمحرف الانتقال إلى سطر جديد ، ويكتب وسيط هذا التابع بين قوسين ، وعندما يكون الوسيط سلسلة محرفية ثابتة يوضع بين علامتي اقتباس مزدوجة (" ") ، كما يلي :

System.out.println("Welcome to Java Programming!")

object reference method name parameter list

والفارق بين `println` و `print` هو أن التابع `print` لا ينتقل إلى سطر جديد ،
ينتمي التابع `println` إلى الصف `out` الذي ينتمي بدوره إلى الصف `system`
فالاسم الكامل للصف هو `java.lang.system` ونجد فيه وصفاً للحقول وقائمة
بالتوابع الخاصة بهذا الصف ونجد الحقل `out` في الصف `system` ، وهو من
النمط `PrintStream` وهو أيضاً يتمتع بالصفة `static` .
هناك صيغتان لطباعة النتائج هما :
١. الصيغة الأولى :

```
System.out.println( " Welcome "+x);
```

حيث `x` : إما `String` أو `int` أو `double` أو الخ

٢. الصيغة الثانية :

```
System.out.printf( " Welcome %s %d %f %c", w ,x , y, z ) ;
```

حيث `s` : تدل على أن `w` هي سلسلة حرفية .

حيث `d` : تدل على أن `x` هو عدد صحيح .

حيث `f` : تدل على أن `y` هو عدد حقيقي .

حيث `c` : تدل على أن `z` هو حرف وحيد .

مثال (1-1)

أكتب برنامجاً بلغة Java يقوم بطباعة العبارة التالية :

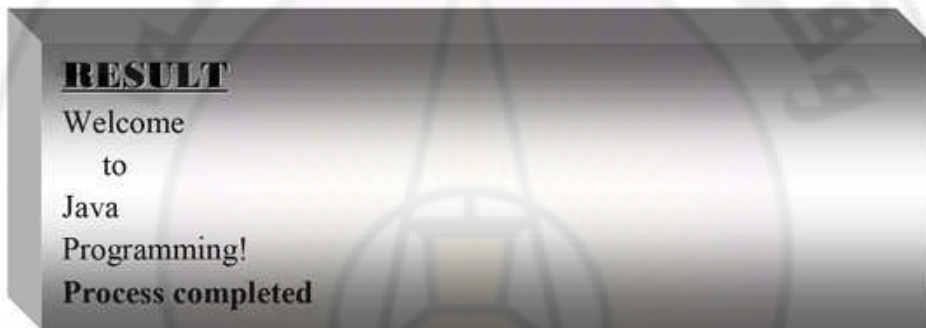
```
"Welcometo Java Programming!"
```

بحيث تكون كل كلمة على سطر .

البرنامج:

1. **public class** Welcom2 {
2. *// main method begins execution of Java application*
3. **public static void** main(**String** args[]) {
4. **System.out.println**(
 " Welcome \n to \n Java \n Programming!");
5. } *// end method main*
6. } *// end class Welcome2*

وعند تنفيذ البرنامج السابق سنحصل على الخرج التالي:



مثال (1-2)

ما هو خرج البرنامج التالي :

1. **public class** Welcom3 {
2. *// main method begins execution of Java application*
3. **public static void** main(**String** args[]) {
4. **System.out.print** ("Welcome to");
5. **System.out.println**("Java Programming!");
6. **System.out.println** ("-----");
7. **System.out.println** ("Welcome to");
8. **System.out.println**("Java Programming!");
9. } *// end method main*
10. } *// end class Welcome3*

وعند تنفيذ البرنامج السابق سنحصل على الخرج التالي:

RESULT

Welcome to Java Programming!

Welcome to
Java Programming!
Process completed.

١-٥- متواليات الهروب (Escape Sequences)

نلاحظ في المثال (1 - 1) أنه لم تتم طباعة \n على الشاشة ، وأن \" تسمى الشرطة الخلفية (Back slash) أو حرف هروب (Escape character) وتسمى الشرطة \ والحرف الذي يليها متواليات الهروب ، ومتواليات الهروب \n يعنى الانتقال إلى سطر جديد حيث يجبر المؤشر على الانتقال إلى بداية السطر التالي ، ويبين الجدول (١-١) بعض متواليات الهروب الشائعة .

الوصف	متوالية الهروب
الانتقال إلى بداية سطر جديد	\n
وضع مسافة أفقية بين قيم الخرج بمقدار 8bits	\t
حرف التراجع back space أرجع المؤشر خطوة إلى الخلف	\b
لطباعة شرطة خلفية	\\
يجبر المؤشر على الانتقال إلى بداية هذا السطر	\r
طباعة مسافة رأسية " "	\v
طباعة علامة استفهام ؟	\?
إصدار صوت alert(deep)	\a
طباعة علامة اقتباس مزدوجة	\"

الجدول (١-١) - يبين بعض متواليات الهروب الشائعة

٦-١- المجاري المعيارية في لغة Java

تستخدم لغة Java ثلاثة مجاري معيارية هي :

١. **System.in** المجرى : هو مجرى الدخل المعياري الذي يُستخدم

لجلب المعطيات من المستخدم إلى البرنامج .

٢. **System.out** المجرى : هو مجرى الخرج المعياري الذي يُستخدم

لإخراج المعطيات من البرنامج إلى المستخدم .

٣. **System.err** المجرى : هو مجرى الخطأ المعياري الذي يُستخدم

لإخراج معطيات الأخطاء من البرنامج إلى المستخدم .

`System.err.print( " Not enough players: " );`

يبين المثال التالي مجاري الدخل /الخرج المعيارية ، حيث يقوم البرنامج بإدخال

عديدين من قبل المستخدم وحساب مجموع العددين وطباعة النتيجة .

مثال (3-1)

```
import java.util.Scanner;

public class Sum {
    public static void main( String args[ ] ) {
        Scanner input = new Scanner( System.in ); // 1
        int number1 , number2,number3;
        System.out.print( "Enter first integer: " )
        number1 = input.nextInt( );
        System.out.print( "Enter second integer: " );
        number2 = input.nextInt( );
        number3 = number1+number2;
        System.out.println( "Cint="+ number3 );
    } // end method main
} // End class Sum
```

السطر 1:

يتم إنشاء غرض من الصف Scanner وحجز مكان لهذا الغرض في الذاكرة .

١-٧- الحزمة القياسية util

تتضمن الحزمة `java.util` العديد من صفوف المعطيات المجردة ADT (Abstract Data Type) بالإضافة إلى واجهتين ، ومن خلال صفوف هذه الحزمة نتمكن من بناء بنى معطيات معقدة وفقاً لمتطلباتنا ، وأن الصف `Scanner` موجود ضمن هذه الحزمة . إذاً هذه الحزمة تتضمن مزيجاً من الصفوف الخدمية العامة كصف التاريخ ، وصف الشعاع ، وما شابه ، أما التصريح `import` يؤمن تضمين صفوف من حزم أخرى .

١-٨- جلب (استيراد) صف من حزمة

يمكن جلب صف من حزمة جافا من أجل استخدامه في الملف الحالي من خلال الكلمة المحجوزة `import` ، ويجب أن تكون عبارة `import` أول عبارة في الملف المصدر (ولكن بعد عبارة `package` في حال وجودها) والعبارة التالية توضح كيفية جلب الصف `Scanner` من الحزمة `util` :

```
import java.util.Scanner;
```

ولإنشاء مثيل من الصف `Scanner` نكتب :

```
Scanner input = new Scanner( System.in );
```

وبالتالي يتم إدخال القيم المطلوبة (عدد صحيح) من المستخدم كما يلي :

```
number1 = input.nextInt( );
```

أما إذا كان المطلوب إدخال عدد حقيقي :

```
number2 = input.nextDouble( );
```

وفي حال كان المطلوب إدخال سلسلة محرفية :

```
ss = input.nextLine( );
```

أما إذا كان المطلوب إدخال سلسلة محرفية مكونة من كلمة واحدة :

```
world = input.next( );
```


وبهذه الطريقة يمكن استخدام الاسم المختصر للصف Scanner وبالتأكيد لا يمكن استخدام أي صف آخر من هذه الحزمة إلا باستخدام اسمه الكامل ، وذلك لأننا لم ن جلب إلا الصف Scanner من الحزمة util ، و لجلب (استيراد) جميع الصفوف والواجهات لهذه الحزمة أو لحزمة ما ، نستخدم إشارة (*) كما يلي :

```
import java.util.*
```

يمكن استخدام أي صف أو واجهة في الحزمة باستخدام الأسماء المختصرة (عند استخدام إشارة *) :

```
// import graphics . Circle ;
```

```
import graphics . * ;
```

```
Circle myCircel = new Circel() ;
```

ولا يمكن استخدام إشارة (*) لجلب مجموعة جزئية من صفوف الحزمة كما يلي :

```
import graphics . A* ; // doesn't work
```

وتُستخدم العبارة import لجلب صف من الحزمة أو جلب جميع صفوف الحزمة. يتم جلب الحزم التالية افتراضياً :

- الحزمة الافتراضية default package .

- والحزمة java.Lang

- والحزمة الحالية current package .

ملاحظة مهمة :

- يجب الملاحظة جيداً أن هناك كلمات لا بد وأن يكتب الحرف الأول منها بحروف كبيرة Capital letter مثل : (System, Scanner , nextInt) .
- يجب أن تكون أسماء المتغيرات واضحة حتى يتم فهم البرنامج جيداً.
- بالنسبة للأرقام المدخلة يجب أن تكون من النوع الصحيح فقط integer و وهذا يكون عن طريق عبارة (nextInt) فالحروف الثلاثة (Int) والتي جاءت بعد كلمة next هي المسؤولة عن إدخال الأرقام الصحيحة من قبل المستخدم ولها عدة حالات كما هم موضح بالجدول (١-٢) .

شكل عبارة next	مثال Examble
Int_Variable = Object_Name.nextInt()	ادخال عدد صحيح int x ; x = input.nextInt()
Double_Variable = Object_Name.nextDouble()	ادخال عدد حقيقي double y ; y = input.nextDouble()
String_Variable = Object_Name.next ()	ادخال كلمة int W ; W = input.next ()
String_Variable = Object_Name.nextLine()	ادخال سلسلة محرفية int S ; S = input.nextLine()

بالتدول (١-٢) - يوضح شكل عبارة next

١-٩- منصة عمل Java

إن مصطلح منصة العمل Platform يعني عتاد الحاسوب Hardware أو بيئة البرمجيات التي تعمل عليها البرامج ، ومعظم منصات العمل يمكن أن توصف على أنها اتحاد بين نظام التشغيل وعتاد الحاسوب ، وإن معظم منصات عمل جافا تختلف عن معظم منصات العمل الأخرى ، حيث إنها منصة عمل برمجية فقط ، وتتكون منصة العمل من :

- ✓ Java VM (Java Virtual Machine)
- ✓ Java API (Java Application Programming Interface)

١-١٠- رموز لغة Java

يبني البرنامج في لغة Java من مجموعة من العناصر التالية :

- الحروف الانكليزية الصغيرة x, y, z , ... a, b, c, d,
- الحروف الانكليزية الكبيرة X, Y, Z , ... A, B, C, D,
- الأرقام العربية 0, 1, 2, 2, ... 8, 9
- رموز خاصة مثل :

< <= > >= + - * / = # % \$
 ? ; : , . << >> ! ' "
 {} [] || & ()

١١-١ - الكلمات المحجوزة في لغة Java

Key Words

تستخدم لغة Java مجموعة من الكلمات المحجوزة بعضها تستخدمها لغة C++ ، وتضيف عدداً من الكلمات الأخرى التي تجسد المبادئ الجديدة في لغة Java ، ونبين في الجدول (٣-١) مجموعة الكلمات المحجوزة .

١٢-١ - المتحولات والثوابت variables and constants

من المعروف أن البرمجة ليست في طباعة بعض الأسطر على الشاشة فقط، بل هي أبعد من ذلك بكثير، لذلك من أجل كتابة برامج تقوم بتنفيذ بعض المهام وحل مسائل متنوعة، فإننا نحتاج إلى التعرف إلى مفهوم المتحولات والثوابت ومعرفة أنواع المعطيات .

١٣-١ - أنواع المعطيات الأساسية في لغة Java

تحتوي لغة Java على نوعين من أنماط المعطيات :

- الأنماط الأساسية Primitive Type

- الأنماط المرجعية References Type

وتستطيع الأنماط الأساسية تخزين قيمة واحدة فقط (رقم ، حرف ، قيمة منطقية ، ...) وتحتاج إلى حجم معين في الذاكرة ، ولا يتغير نمط المعطيات في جافا بتغيير منصة العمل ، كما هو الحال في بعض اللغات الأخرى . ويمكن تلخيصها في الجدول (٤-١) . أما الأنماط المرجعية هي : Arrays و Classes .

الكلمات المحجوزة في اللغة		
Abstract	Finally	Public
Boolean	Float	Return
Break	For	Short
Byte	If	Static
Case	Implements	Super
Catch	Import	Switch
Char	instanceof	Synchronized
class	Int	This
continue	interface	Throw
default	Long	Throws
Do	Native	Transient
double	New	True
Else	Null	Try
extends	package	Void
False	Private	Volatile
Final	Protected	While

الجدول (٣-١) - الكلمات المحجوزة في لغة Java

النوع	الوصف	الحجم
char	محرف (أحرف)	1 byte
short	أعداد صحيحة قصيرة	2 bytes
int	أعداد صحيحة عالية	4 bytes
long	أعداد صحيحة طويلة	8 bytes
float	أعداد حقيقية عالية	4 bytes
double	أعداد حقيقية مضاعفة	8 bytes
long double	أعداد حقيقية ضخمة	10 bytes
bool	قيمة منطقية true/false	1 bit

الجدول (٤-١) - يبين أنواع المعطيات الأساسية

١-١٣-١ المتحولات (المتغيرات) variables

المتحول موضع من الذاكرة مخصص لتخزين قيمة محددة, مثلاً لتعرف على المتغيرات التالية:

$a = 5 ;$ $a = a + 1 ;$
 $b = 2 ;$ $r = a - b ;$

نلاحظ أن المتحول a يأخذ القيمة الصحيحة 5 ، بينما المتحول b يأخذ القيمة الصحيحة 2 ، بعد ذلك يتم تغير قيمة المتحول a من القيمة القديمة (5) إلى القيمة الجديدة (6) ، وفي النهاية يتم إسناد حاصل طرح قيمة المتحول b من قيمة المتحول a ، إلى متحول جديد r .

١-١٣-١-١ تسمية المتحولات Naming of variables

كل متحول يحتاج إلى اسم يميزه عن متحول آخر وعند تسمية أي متحول يجب أن نراعي بعض القواعد تجنباً للوقوع في أخطاء برمجية عند تنفيذ البرامج بلغة Java , ويجب اتباع القواعد التالية عند تسمية المتحولات :

١. يجب أن يبدأ اسم المتحول بحرف من حروف اللغة الانكليزية الصغيرة أو الكبيرة ، ثم تتوالى بعد ذلك الحروف أو الأرقام ويجب ألا يحتوي على أي رمز خاص سوى الشرطة التحتية (_).

٢. من الممكن أن يكون طول اسم المتحول حتى 32 حرفاً وإذا زاد عن ذلك لا يأخذ بعين الاعتبار من قبل مترجم اللغة (Compiler).

٣. لغة Java تفرق بين الأحرف الكبيرة والصغيرة لذلك يجب عدم استخدام الأحرف الكبيرة والصغيرة لاسم المتحول نفسه، لأن المترجم يفرق بينها، مثلاً، المتحول a يختلف عن المتحول A .

٤. يجب أن يكون اسم المتحول فريداً في الكتلة التي عُرف بها .

٥. يجب ألا يكون كلمة محجوزة مثل for ، أو أي قيمة منطقية true, false ، أو الكلمة null أو أي كلمة من الكلمات المحجوزة في لغة Java .

٦. يفضل أن يبدأ اسم المتحول بحرف صغير ، وفي حال كان الاسم مؤلفاً من أكثر من كلمة يفضل جعل الحرف الأول من كل كلمة (ما عدا أول كلمة) حرفاً كبيراً مثل : isGoodColor .

١-٣-١-٢- مجال المتحولات

إن مجال المتحول هو المنطقة التي يمكن فيها استخدام المتحول ، ويحدد مجال المتحول متى يقوم النظام بإنشاء المتحول ومتى يقوم بتحريره من الذاكرة . وإن مكان توضع المتحول البرمجي داخل النص البرمجي هو الذي يحدد مجاله ، وذلك حسب إحدى التصنيفات التالية :

١. المتحولات الأعضاء **Members Variable** : هي معطيات أعضاء من صف ، ويتم التصريح عنها داخل الصف ولكن خارج أي تابع أو بان ، ومجال هذا النوع من المتحولات هو كامل الصف (انظر إلى مقرر الرمجة (1)) .

٢. المتحولات المحلية **Local Variables** : يتم التصريح عنها داخل الكتلة ، ومجالها يمتد من نقطة التصريح عنها ولغاية نهاية الكتلة التي صُرحَت فيها .

٣. الوسطاء لتابع (**Parameters function**) : هي متحولات يتم استخدامها في التتابع أو البواني من أجل تمرير القيم من/إلى التابع أو الباني و أن مجال المتحولات هو كامل التابع .

يمكن تهيئة المتحولات الأعضاء والمتحولات المحلية مباشرة أثناء التصريح عنها :

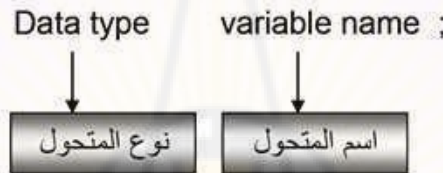
int x=200 ; long y=300 ;

وإن الوسطاء لا يمكن إسناد قيم لها مباشرة , بل يجب تمرير قيمها من خلال النص البرمجي .

١-١٣-١-٣- التصريح عن المتحولات

Declaration of variables

كل متحول يُستخدم في برنامج بلغة Java لا بد من التصريح عنه ويتم ذلك في أي مكان من البرنامج ولكن بشرط أن يكون التصريح قبل العبارات البرمجية التي ستستخدم هذا المتحول . يبين الشكل (١-٥) الشكل العام للتصريح عن المتحولات .



الشكل (١-٥) - يبين الشكل العام للتصريح عن المتحولات

في لغة Java يوجد العديد من المتحولات التي يمكن استخدامها والتصريح عنها وفيما يلي سنذكر أهم هذه المتحولات وكيفية التصريح عنها .

١- النمط boolean & cast

النمط boolean : هو 1Bit من المعطيات , ويأخذ قيمتين true أو false أي (0 , 1) .

Declaration: boolean b;

Assignment: b = true; b = false;

أما المعامل (cast) يستخدم للتحويل من نوع إلى آخر من المعطيات . و يبين المثال (1-4) استخدام هذا النمط .

```
// Demonstrate boolean values.
class BoolTest {
public static void main(String args[ ] ) {
boolean b;
b = false;
System.out.println("b is " + b);
b = true;
System.out.println("b is " + b);
// a boolean value can control the if statement
if(b) System.out.println("This is executed.");
b = false;
if(b) System.out.println("This is not executed.");
// outcome of a relational operator is a boolean value
System.out.println("10 > 9 is " + (10 > 9));
} //end main
} //End class
```

وعند تنفيذ البرنامج السابق سنحصل على الخرج التالي:

RESULT

```
b is false
b is true
This is executed.
10 > 9 is true
Process completed
```

٢- التصريح عن المتحولات المحرفية char

للتصريح عن متحول محرفي تُستخدم الكلمة المحجوزة char ، ويتم تخزين المحارف في متحولات من النوع char ، والعبارة التالية توضح ذلك :

```
char ch ;
```

تنشئ مساحة في الذاكرة لمحرف وتسميه ch وحجمه 1Byte ، ولتخزين محرف ما في هذا المتحول نكتب :

```
ch = 'z'
```

ودائماً تكون المحارف الثابتة كـ 'a' و 'b' محصورة بعلامتي اقتباس مفردة.

مثال:

```
char a ; //a is a character variable
char b ; //b is a character variable
or
char a , b ;
```

يبين المثال (1- 5) هذا النوع من المتحولات . و يمكن استعمال المتحولات من النوع char لتخزين أرقام كاملة بدلاً من محارف فمثلاً يمكننا كتابة:

```
ch = 2 ;
```

لكن نطاق القيم الرقمية التي يمكن تخزينها في النوع char يتراوح بين 128- إلى 127 لذا فإن هذه الطريقة تعمل مع الأرقام الصغيرة فقط.

مثال (1-5)

```
// char variables behave like integers.
```

```
class CharDemo2 {
public static void main(String args[ ] ) {
char ch1;
ch1 = 'X';
System.out.println("ch1 contains " + ch1);
ch1++; // increment ch1
System.out.println("ch1 is now " + ch1);
}
```

```
}//end main
```

```
}//End class
```

وعند تنفيذ البرنامج السابق سنحصل على الخرج التالي:

RESULT

chl contains X

chl is now Y

Process completed

٣- التصريح عن السلاسل المحرفية String

تُعامل السلاسل المحرفية في لغة جافا كأغراض، فهي أمثال من الصف String. لكن طبيعتها البسيطة تضطرننا في بعض الأحيان للحديث عنها كمتحولات من أنماط أولية. فعندما نريد إدخال مجموعة محارف إلى برنامج ما بلغة Java، يجب أن نصرح عن المتحول على شكل سلسلة من المحارف String وذلك كما يلي:

String season ;

أو

String season = " Winter " ;

ملاحظة :

النمط byte : هو عدد صحيح واحد بايت 1byte=8bit (0 - 255) .

٤- التصريح عن المتحولات الصحيحة

تمثل المتحولات الصحيحة أرقاماً كاملة أي قيماً يمكن تعدادها ، كعدد الأشخاص أو الأيام أو عدد الصفحات مثلاً ، ولا يمكن أن تكون الأعداد

الصحيحة أرقاماً ذات فاصلة عشرية ولكنها يمكن أن تكون سالبة، وهناك ثلاثة أنواع من الأعداد الصحيحة في Java ، هي :

short : عدد صحيح قصير .

int : عدد صحيح عادي .

long : عدد صحيح طويل .

وهي تحتل مساحات مختلفة في الذاكرة ، والجدول (١-٥) يبين هذه الأنواع والمساحة التي تأخذها في الذاكرة ومجال الأرقام التي يمكن أن تأخذها.

اسم النوع	الحجم	المجال
char	1byte	128 إلى 127
Short	2byte	-32,768 إلى 32,767
int	4byte	-32768 إلى +32798
Long	8byte	-2,147,483,648 إلى 2,147,483,647

الجدول (٦-٥) - يبين أنواع المتحولات الصحيحة

- للتصريح عن متحول صحيح من النوع **int** .

مثال:

```
int a ; //a is an integer variable
```

```
int b ; //b is an integer variable
```

or

```
int a , b ;
```

- للتصريح عن متحول صحيح من النوع **long** .

مثال:

```
long a ; // a is a long integer variable
```

```
long b ; // b is a long integer variable
```

or
long a, b ;

٥-التصريح عن متحول حقيقي

يتم استعمال المتحولات الحقيقية لتمثيل قيم يمكن قياسها كالأطوال أو الأوزان، ويتم تمثيل الأعداد الحقيقية عادة برقم كامل على اليسار مع فاصلة عشرية وكسر على اليمين، وهناك ثلاثة أنواع من الأعداد الحقيقية في أنظمة التشغيل الشائعة الاستعمال ، وأشهر أنواع الأعداد الحقيقية هي من النوع double والذي يتم استعماله لمعظم توابع C++ الرياضية ، ويتطلب النوع float ذاكرة أقل من النوع double ، و يوضح الجدول (٦-١) هذه الأنواع والحجم الذي تأخذه في الذاكرة.

اسم النوع	الحجم
float	4byte
double	8byte
long double	10byte

الجدول (٦-١) - يبين أنواع المتحولات الحقيقية

- للتصريح عن متحول حقيقي من النوع **float** :

مثال:

float a ; // a is a real variable

float b ; // b is a real variable

Or

float a, b ;

- للتصريح عن متحول حقيقي من النوع **double** :

مثال:

double a ; // a is a big real variable

double b ; // b is a big real variable

or
double a , b ;

ملاحظة

- يجب أن نميز بين المتحولات العامة Global والمتحولات المحلية Local من خلال وضع عملية التعريف داخل البرنامج حيث:
- ١- المتحولات التي يتم تعريفها قبل بداية البرنامج (الدالة main) تمثل متحولات عامة.
 - ٢- المتحولات التي يتم تعريفها داخل الدالة main أو داخل أي تابع تعد متحولات محلية تستخدم فقط داخل الجزء المعرفة فيه.

مثال (1-6)

المثال التالي يحسب السرعة التقريبية للضوء ميل / ثانية باستخدام المتحولات من نمط Long وذلك لـ 1000 يوم .

البرنامج:

```
class Light {  
public static void main(String args[ ] ) {  
int lightspeed;  
long days;  
long seconds;  
long distance;  
// approximate speed of light in miles per second  
lightspeed = 186000;  
days = 1000; // specify number of days here  
days = 1000; // specify number of days here  
seconds = days * 24 * 60 * 60;  
// compute //distance
```

```

distance = lightspeed * seconds;
System.out.print("In " + days);
System.out.print(" days light will travel about\n ");
System.out.println(distance + " miles.");
} //end main
} //End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

In 1000 days light will travel about
16070400000000 miles.

Process completed

ملاحظة

عند إعطاء قيمة أولية لمتغير ما a ، على سبيل المثال (القيمة 0) ، فإننا نصرح عن ذلك كما يلي:

```
int a=0;
```

١-٣-١-٤ - التحويل بين المتحولات

في بعض الأحيان من الضروري دمج مجموعة من أنماط مختلفة في عملية الحساب ، فعند تنفيذ عملية ثنائية تتضمن حدين من نمطين مختلفين فإن لغة Java تقوم بشكل تلقائي بتحويل الحد معتمدة على القواعد التالية :

١. إذا كان أحد الحدود من النمط double عندها يتم تحويل الحد الآخر إلى النمط double (الحد الآخر مثلاً من النمط int أو float).
٢. إذا كان أحد الحدود من النمط float عندها يتم تحويل الحد الآخر إلى النمط float (الحد الآخر مثلاً من النمط int أو byte).

٣. إذا كان أحد الحدود من النمط long عندها يتم تحويل الحد الآخر إلى

النمط long (الحد الآخر مثلاً من النمط int أو byte).

لا يمكن إسناد قيمة إلى متحول نمطه ينتمي إلى مجال أصغر من نمط القيمة

, إلا إذا قمنا بعملية تحويل للنمط (Type Casting) . حيث إن تحويل النمط

هو عبارة عن عملية تقوم بتحويل قيمة تنتمي لنمط معطيات ما إلى قيمة تنتمي

لنمط معطيات آخر .

وتعرف عملية تحويل متحول ينتمي لنمط ذي مجال صغير إلى متحول ينتمي

لنمط ذي مجال كبير بعملية تمديد النمط (Widening a Type)، وتعرف عملية

تحويل متحول ينتمي لنمط ذي مجال كبير إلى متحول ينتمي لنمط ذي مجال

صغير بعملية تضيق النمط (Narrowing a Type) . ومن الممكن إنجاز

عملية تمديد النمط بشكل تلقائي وبدون القيام بعملية تحويل صريحة ، بينما يجب

إنجاز عملية تضيق النمط بشكل صريح .

الصيغة العامة لعملية التحويل هي وضع النمط في أقواس () ثم اتباعه باسم

المتحول أو القيمة التي سيتم تحويلها ، مثلاً :

```
int y = 55 ;
```

```
float x = ( float ) y ;
```

تم تحويل y من النمط int إلى النمط float .

المثال (1 - 7) التالي يقوم بتحويل المتحول i من النمط int إلى النمط byte

وإسناده للمتحول b الذي هو من النمط byte حيث :

```
int i = 256 ;
```

```
byte b = ( byte ) i ;
```


ما هي قيمة المتحول ؟

مثال (7-1)

```
// Demonstrate casts.
class Conversion {
public static void main(String args[ ] ) {
byte b;
int i = 257;
double d = 323.142;
System.out.println("\nConversion of int to byte.");
b = (byte) i;
System.out.println("i and b " + i + " " + b);
System.out.println("\nConversion of double to int.");
i = (int) d;
System.out.println("d and i " + d + " " + i);
System.out.println("\nConversion of double to byte.");
b = (byte) d;
System.out.println("d and b " + d + " " + b);
} //end main
} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Conversion of int to byte.

i and b 257 1

Conversion of double to int.

d and i 323.142 323

Conversion of double to byte.

d and b 323.142 67

Process completed

التمثيل الثنائي للمتحول i هو كالتالي : $2^8 = 256$ (10000000) حيث يمتلك النمط الصحيح 4byte (00000000 00000000 00000000 00000001) والبايت الأول في i هو 00000000 ، والذي تم إسناده إلى المتحول b ، وهكذا أصبحت قيمة b هي 0 . في البرنامج كانت قيمة $i = 257$ عندها قيمة $b = 1$. حيث تم تحويل المتحول $d = 323.142$ إلى متحول من نمط صحيح عندها $i = 323$ حيث تم حذف الجزء العشري . بينما تم تحويل المتحول $d = 323.142$ إلى متحول من نمط byte عندها $b = 67$. ويجب الانتباه عند التحويل من نمط إلى آخر لكي لا نفقد جزءاً من المعطيات .
المثال (1-8) يعرض مجال المتحولات المحلية ضمن الكتل .

مثال (1-8)

```

1. // Demonstrate block scope.
2. class Scope {
3.     public static void main(String args[] ) {
4.         int x; // known to all code within main
5.         x = 10;
6.         if( x == 10) { // start new scope
7.             int y = 20; // known only to this block
8.             // x and y both known here.
9.             System.out.println("x and y: " + x + " " + y);
10.            x = y * 2;
11.        }
12.        // y = 100; // Error! y not known here
13.        // x is still known here.
14.        System.out.println("x is " + x);
15.    } //end main

```

16. `}//End class`

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

x and y: 10 20

x is 40

Process completed

- المتحول x في السطر 4 يتم التصريح عنه ضمن مجال الـ main وبالتالي يكون معرفاً حتى نهاية الـ main وسهل الوصول اليه .
- بينما المتحول y في السطر 7 فهو معرف ضمن كتلة الـ if فقط بينما المتحول x يكون معرفاً في هذه الكتلة مع المتحول y .
- بينما في السطر 12 خارج الكتلة if يكون المتحول x معرفاً أما y فيكون غير معرفاً .

١-٤- المتحولات الثابتة Final Variables

يمكن التصريح عن متحول بحيث تُصبح قيمته ثابتة (أي غير قابلة للتعديل) باستخدام الكلمة المحجوزة `final` وفي لغة Java نميز عدة أنواع من المتحولات الثابتة , منها :

```
final float PI = 3.14159 ;  
final int w = 100 ;  
final char newline = 'a' ;  
final z = 12440 ;
```

ملاحظة

في الحالة التي لا يتم فيها ذكر نوع الثابت فإن المترجم يعد هذا الثابت من النوع **int**.

مثال (6-7)

اكتب برنامجاً بلغة الـ Java يقوم بحساب وطباعة مساحة دائرة نصف قطرها r .

البرنامج:

```
//This program computes the square of a circle
//Compute the area of a circle.

class Area {
public static void main(String args[ ] ) {
double r, a;
r = 10.8; // radius of circle
final pi = 3.1416; // pi, approximately
a = pi * r * r; // compute area
System.out.println("Area of circle is " + a);
} //end main
} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Area of circle is 366.436224

Process completed

١٥-١ - العوامل الحسابية والمنطقية والعلائقية

العامل Operator عبارة عن رمز رياضي يقوم بإجراء عملية معينة على معاملاته ويمكن أن تكون عملية حسابية أو منطقية أو عملية مقارنة ، وفي هذه الفقرة سنتعرف على أنواع العوامل التي توفرها لغة Java .

١٥-١-١ - العوامل الحسابية Arithmetic Operators

تستخدم أكثر البرامج في لغة Java العمليات الحسابية ، مثل : الجمع والطرح وغيرها، ويبين الجدول (٧-١) العمليات الحسابية ورموزها المقابلة في لغة Java .

العملية	الرمز الرياضي للعملية	رمز العملية بلغة Java	مثال
الجمع	+	+	$a + b$
الطرح	-	-	$a - b$
الضرب	x or .	*	$a * b$
القسمة	____ ، ÷	/	a / b
باقي القسمة الصحيح	$a \bmod b$ or %	%	$a \% b$

الجدول (٧-١) يبين العمليات الحسابية ورموزها المقابلة في لغة Java

تتجزئ العوامل الأربعة الأولى أعمالاً مألوفة لدينا، أما عامل باقي القسمة % المسمى أيضاً المعامل modulus، يتم استعماله لحساب باقي قسمة عدد صحيح على عدد آخر، لذلك فالتعبير 3%20 يساوي 2 . تسمى هذه العوامل الحسابية بالعوامل الثنائية لأنها تعمل على قيمتين ، ويمكن استعمال أكثر من عامل في تعبير رياضي واحد. التعبير الحسابي عبارة عن مجموعة من العمليات الحسابية التي تربط الأعداد والثوابت والمتغيرات والتوابع، ويبين الجدول (٨-١) بعض التعبيرات الحسابية.

الشكل الرياضي للتعبير الحسابي	الشكل البرمجي للتعبير الحسابي
$\frac{x+y}{x-y} + c$	$(x+y)/(x-y) + c$
$2 * r - 6.5$	$2 \cdot r - 6.5$
$D = b * b - 4 * a * c$	$D = b^2 - 4ac$
$y = 2 * x * x + 5 * x - \frac{7}{3} + 2 * x * x * x$	$y = 2x^2 + 5x - \frac{7}{3} + 2x^3$

والجدول (٨-١) - يبين بعض التعبيرات الحسابية

١-٥-٢ - العوامل العلائقية (المقارنة)

Relational Operators

تقارن العوامل العلائقية قيمتين، وتكون نتيجة التنفيذ إما صحيحة True أو خاطئة False حسب نتيجة المقارنة مع القيمتين ، ويوضح الجدول (٩-١) العوامل العلائقية (المقارنة) وما يكافئها في لغة Java.

العملية	الرمز الرياضي للعملية	رمز العملية بلغة Java	مثال
المساواة	=	=	a == b ?
أكبر من	>	>	a > b ?
أكبر أو تساوي	≥	>=	a >= b ?
أصغر من	<	<	a < b ?
أصغر أو تساوي	≤	<=	a <= b ?
عدم المساواة	!=	!=	a != b ?

الجدول (٩-١) - يوضح عوامل المقارنة وما يكافئها في لغة Java

١-١٥-٣-العوامل المنطقية Logical Operators

تُستخدم العوامل المنطقية لربط أكثر من تعبير منطقي بسيط مع بعضها

ببعض، ويوضح الجدول (١٠-١) العوامل المنطقية وما يكافئها في لغة Java

العملية	الرمز الرياضي للعملية	رمز العملية بلغة الـ Java	مثال
و	AND	&&	(a>b) && (a>c)
أو	OR		(a>b) (a>c)
التنفي	NOT	!	!(a>b)

الجدول (١٠-١) - يوضح العوامل المنطقية وما يكافئها في لغة Java

١-١٥-٤- المعامل الشرطي الثلاثي الحدود

Conditional Operator (?)

يوجد في لغة جافا معامل خاص يسمح باختبار قيمة الشرط المنطقي واستخدام واحد من تعبيرين حسب قيمة هذا الشرط وهو المعامل (?) وله الصيغة التالية :

expression - if - true : expression - if - else (Condition) ?

يُستخدم المعامل الشرطي (?) من أجل اختبار شرط معين ويُعيد قيمة إذا تحقق الشرط أو يُعيد قيمة أخرى إذا لم يتحقق الشرط , ويمكن كتابته بصيغتين :

1 : (condition) ? result1 : result2

2 : (condition ? result1 : result2)

حيث:

condition - الشرط المنطقي.

result1 - القيمة المعادة إذا تحقق الشرط.

result2 - القيمة المعادة إذا لم يتحقق الشرط.

مثال :

y = (x < 5) ? 44 : 55 ;

y = (x < 5 ? 44 : 55) ;

إذا فرضنا أن x=3 فإن قيمة y = 44 .

١-٥-٥ - عوامل الإسناد المركبة

Compound Assignment Operators

يتوافر في لغة Java عدد من عمليات الإسناد، فبالإضافة لعملية الإسناد

العادية التي تستخدم فيها الإشارة (=) ، يوجد طرق مختصرة أخرى تسمى

عمليات الإسناد المركبة ، ويمكن توضيحها من خلال الجدول (١-١١).

عملية الإسناد المختصرة	عملية الإسناد
a+=b	a=a+b
a-=b	a=a-b
a*=b	a=a*b
a/=b	a=a/b
a%=b	a=a%b

(١-١١) - يوضح عمليات الإسناد المركبة

يوضح المثال (10-1) بعض العمليات الحسابية .

مثال (10-1)

// Demonstrate the basic arithmetic operators.

```
class BasicMath {
```

```
public static void main(String args[ ] ) {
```

```
    // arithmetic using integers
```

```
    System.out.println("Integer Arithmetic");
```

```
    int a = 1 + 1;
```

```
    int b = a * 3;
```

```

int c = b / 4;
int d = c - a;
int e = -d;
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
System.out.println("e = " + e);
        // arithmetic using doubles
System.out.println("\nFloating Point Arithmetic");
double da = 1 + 1;
double db = da * 3;
double dc = db / 4;
double dd = dc - a;
double de = -dd;
System.out.println("da = " + da);
System.out.println("db = " + db);
System.out.println("dc = " + dc);
System.out.println("dd = " + dd);
System.out.println("de = " + de);
        // Demonstrate the % operator.
int x = 42;
double y = 42.25;
System.out.println("x mod 10 = " + x % 10);
System.out.println("y mod 10 = " + y % 10);
} //End main
} //End Class BasicMath

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

i=9

Integer Arithmetic

a = 2

b = 6

c = 1

d = -1

e = 1

Floating Point Arithmetic

da = 2.0

db = 6.0

dc = 1.5

dd = -0.5

de = 0.5

Process completed.

Modulus Operator
x mod 10 = 2
y mod 10 = 2.25

١-٥-٦- عمليات الزيادة والنقصان

Increment and Decrement Operators

إن عامل الزيادة هو (++) ويعني زيادة المتحول بمقدار واحد، بينما عامل النقصان (--) يعني إنقاص قيمة المتحول بمقدار واحد. والزيادة والنقصان يمكن أن يكون من اليسار (الأمام) أو اليمين (الخلف)، أي يمكن وضع ++ أو -- قبل المتحول (الزيادة أو النقصان من اليسار) أو يمكن وضع ++ أو -- بعد المتحول (الزيادة أو النقصان من اليمين). من حيث المبدأ لا يوجد فرق بين الزيادة (النقصان) من اليسار أو الزيادة (النقصان) من اليمين كقيمة وإنما الاختلاف في الاستخدام، فعلى سبيل المثال التعبير ++I يعني إضافة القيمة واحد إلى قيمة المتحول I قبل استخدام المتحول I، في حين أن التعبير I++ يعني استخدام القيمة الحالية للمتحول I ثم إضافة واحد إلى قيمة المتحول I.

□ الزيادة بمقدار واحد (++)

مثال

```
int A ;  
A++ ; // A = A + 1 ;  
System.out.println(A++) ;
```

□ النقصان بمقدار واحد (--)

مثال

```
int B ;  
B-- ; // B = B - 1 ;  
System.out.println(A-- ) ;
```

مثال (11-1)

اكتب برنامجاً بلغة Java يقوم بتعريف ثلاثة متحولات من نوع int كما يلي :

```
int k = 4 , i , j ;
```

- أجرِ عملية الزيادة من اليسار للمتحول k , ثم أجرِ الزيادة من اليمين للمتحول k , ثم اطبع قيمة k .
- اجعل $i = k++$ ثم اطبع قيمة i و k .
- اجعل $i = ++k$ ثم اطبع قيمة i و k .
- ماذا تلاحظ ؟

البرنامج

```

class Increment {
public static void main(String args[ ] ) {
int k=4;
++k ;
k++;
System.out.println("k="+ k );
Int i = k++;
System.out.println("i="+ i + " k = " + k );

Int j = ++k;
System.out.println("j="+ j + " k = " + k );
} //End main
} //End program

```

RESULT

6

6 7

8 8

Process completed.

يوضح البرنامج في المثال (12-1) عمليات الزيادة increment (++) والنقصان decrement (--). حيث نمط المعطيات المطبق هو char أي محرف واحد وليس سلسلة محرفية .

مثال (12-1)

```

/*This program uses increment (++) and decrement (--) */
class CharDemo2 {
public static void main(String args[ ] ) {
char ch1 = 'X', ch2 = 'C';
System.out.println("ch1 contains " + ch1 + "\n ch2 contains " + ch2);
ch1++; // increment ch1
ch2 -- ; // decrement ch2
System.out.println("ch1 is now " + ch1 + "\nch2 is now " + ch2);
} //End main
} //End Class ,

```

وعند تنفيذ البرنامج سيظهر الخرج التالي :

RESULT

ch1 contains X

ch2 contains C

ch1 is now Y

ch2 is now B

Process completed.

١٦-١ - أولوية تنفيذ العمليات الحسابية والمنطقية والعلائقية

إن تنفيذ العمليات الحسابية والمنطقية تتم في مترجم Java حسب قواعد محددة وإن أولويات تنفيذ العمليات الحسابية والمنطقية والمقارنة موضحة في الجدول (١٢-١) . إذا تساوت الأولويات في تعبير حسابي ما، فيتم التنفيذ من اليسار إلى اليمين أو من اليمين إلى اليسار (حسب العمليات).

العملية	رمز العملية	رقم الأولوية	جهة التنفيذ عند التساوي
فك الأقواس	() أو []	1	من اليسار
التوابع	Sin(),.... sqrt(),....	2	من اليسار
الزيادة أو النقصان	++ أو --	3	من اليمين
العمليات الحسابية	* / %	4	من اليسار
العمليات الحسابية	+ -	5	من اليسار
عمليات المقارنة	> >= < <=	6	من اليسار
عمليات المقارنة	== !=	7	من اليسار
العمليات المنطقية	&&	8	من اليسار
العمليات الشرطية	?:	9	من اليمين
عمليات الإسناد	= += -= *= /= %=	10	من اليمين

الجدول (١٢-١) - يوضح أولوية تنفيذ العمليات الحسابية والمنطقية والمقارنة

١٧-١ - العمليات المنطقية على البت Bitwise

تعمل معاملات الحساب بالنظام الثنائي على مستوى البتات بغض النظر عما تمثله المعطيات ، ويعد وجود هذه المعاملات في جافا مثيراً للتساؤل ، ليس لأنها

غير مفيدة , بل أنها ضرورية جداً عند الحاجة لتحسين مستوى الأداء ، إنما الغريب وجود نمط عددي بدون إشارة في لغة جافا . حيث تمثل المعطيات في الذاكرة بسلاسل من البتات تجمع عادة في وحدات من مضاعفات العدد 8 ، والعنصر الأساسي هو البايت Byte . ونجد في جافا نمط معطيات تمثل إحدى بتات العدد إشارته (سالب أو موجب) . ويبين الجدول (١٣-١) أهم المعاملات المنطقية في لغة جافا هي .

AND (&) OUT	OR () OUT
00	00
01	01
10	10
11	11
XOR (^) OUT	NOT (~) OUT
00	0
01	1
10	
11	

الجدول (١٣-١) - يبييت أهم المعاملات المنطقية في لغة Java

يمكن تطبيق العمليات الحسابية الثنائية على الأنماط (char , long , int , short , byte) ويمكن تطبيق المعاملات (& ، | ، ^) على قيم من النمط boolean . ويبين الجدول (١٤-١) العمليات المنطقية والازاحة .


```

System.out.println(" a|b = " + binary[c]);
System.out.println(" a&b = " + binary[d]);
System.out.println(" a^b = " + binary[e]);
System.out.println("~a&b|a&~b = " + binary[f]);
System.out.println(" ~a = " + binary[g]);
} //End main
} //End BitLogic

```

RESULT

```

a = 0011
b = 0110
a | b = 0111
a & b = 0010
a ^ b = 0101
a & b | a & ~b = 0101~
a = 1100~

```

Process completed.

١٨-١ - واجهات المستخدم المرئية باستخدام JOptionPane

Graphical User Interfaces Using JOptionPane

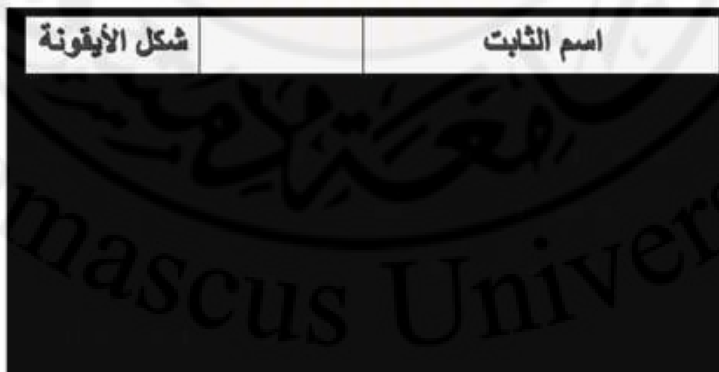
لعرض أي نص ضمن صندوق حوار ، على سبيل المثال الرسالة (message dialog box) ، سنحتاج إلى استخدام التابع showMessageDialog وظيفته إظهار رسالة ما للمستخدم ، هذا التابع موجود ضمن الصف JOptionPane هو أحد صفوف البرمجة المرئية الموجود ضمن الحزمة javax.swing ، ويتضمن هذا الصف أيضاً التابع showInputDialog المسؤول عن إظهار نافذة بسيطة تسمح للمستخدم بإدخال قيمة أو رسالة معينة . ويتم استيراد الصف باستخدام التعليمة import ، لذلك يستطيع المترجم تحديد موقع هذا الصف ، و يوجد الصف System في الحزمة java.lang ، مما لا يستدعي استيراده .

١-١٨-١ - التابع showMessageDialog

هو تابع static ، ويجب استدعاء هذا التابع باستخدام اسم الصف متبوعاً بنقطة (.) ثم التابع مع الوسائط (البارامترات) ، ويُستخدم لعرض أي نص في صندوق الحوار . حيث يأخذ هذا التابع أربعة وسطاء وهي محملة بشكل زائد .

```
JOptionPane.showMessageDialog(null, "Welcome to Java! ",  
"My First Example",  
JOptionPane.INFORMATION_MESSAGE );
```

- الوسيط الأول : يمثل النافذة التي سيظهر فيها صندوق حوار الرسالة , فإذا كان (null) فهذا يعني أن صندوق حوار الرسالة سيظهر في وسط الشاشة .
- الوسيط الثاني: يمثل الرسالة التي ستظهر للمستخدم (Welcome to Java) , ومن الممكن أن تكون سلسلة حرفية تمثل النص الواجب إظهاره .
- الوسيط الثالث (إن وجد) : يمثل الرسالة التي ستظهر على شريط العنوان لصندوق حوار الرسالة ، مثلاً (My First Example) .
- الوسيط الرابع (إن وجد) : فهو يمثل أحد الثوابت التالية المبينة في الجدول (١٥-١) .

شكل الأيقونة	اسم الثابت
	

الجدول (١٥-١) - يبين الثوابت للوسيط الرابع

١٨-٢-١ - التابع `showInputDialog`

هذا التابع موجود ضمن الصف `JOptionPane` الموجود ضمن الحزمة `javax.swing` ، وهو مسؤول عن إظهار نافذة بسيطة تسمح للمستخدم بإدخال قيمة أو رسالة معينة ، وهو تابع `static` ، ويجب استدعاء هذا التابع باستخدام اسم الصف متبوعاً بنقطة (.) ثم التابع مع الوسائط (البارامترات) اللازمة ، ويأخذ هذا التابع أربعة وسطاء وهي محملة بشكل زائد .

```
JOptionPane.showInputDialog(null,  
"Enter a sum Number ", "My SUM Example",  
JOptionPane.QUESTION_MESSAGE);
```

الوسيط الأول : يمثل النافذة التي سيظهر فيها صندوق حوار الرسالة ، فإذا كان (null) فهذا يعني أن صندوق حوار الرسالة سيظهر في وسط الشاشة .

- الوسيط الثاني: يمثل الرسالة التي ستظهر للمستخدم (Enter a sum Number) ، ومن الممكن أن تكون سلسلة حرفية تمثل النص الواجب إدخاله .

- الوسيط الثالث (إن وجد) : يمثل الرسالة التي ستظهر على شريط العنوان لصندوق حوار الرسالة ، مثلاً (My First Example) .

- الوسيط الرابع (إن وجد) : فهو يمثل أحد الثوابت المبينة في الجدول (١-١٥) .

مثال (14-1)

اكتب برنامجاً بلغة Java لطباعة الرسالة "Welcome to Java!" ضمن نافذة.

البرنامج :

```
import javax.swing.JOptionPane;  
public class Welcome{  
    public static void main(String[ ] args) {  
        // Display Welcome to Java! in a message dialog box  
        JOptionPane.showMessageDialog(null,  
            "Welcome to Java!", "My First Example ",
```

```

        JOptionPane.ERROR_MESSAGE );

// Exit the program
    System.exit(0);
} //End main
} //End class

```

وعند تنفيذ البرنامج سنحصل على الخرج بالشكل التالي:



عندما نريد كتابة النص على عدد من الأسطر نستخدم متواليّة الهروب \n :

```

JOptionPane.showMessageDialog(null,
    "Welcome \n to \n Java!", "My First Example ",
    JOptionPane.ERROR_MESSAGE );

```

١-٨-٣- التابع exit(0)

هذا التابع موجود في الصف System الموجود ضمن الحزمة java.lang ، والتابع exit تابع من نمط static ، حيث يؤدي استخدام هذا التابع إلى إنهاء البرنامج ، ويدل البارامتر 0 إلى انتهاء البرنامج بشكل طبيعي . والفائدة من استخدامه هي أنه عندما يبدأ البرنامج يتم صنع مسلك (Thread) من أجل تنفيذ البرنامج ، وعند استدعاء التابع showMessageDialog يتم صنع مسلك

منفصل من أجل تنفيذ هذا التابع ، وفي الواقع لا يتم إنهاء التابع حتى بعد إغلاق صندوق الحوار ، لذلك يجب استدعاء التابع exit من أجل إنهائه .

١-٨-٤- التابع showConfirmDialog

يتضمن التابع showConfirmDialog يت أزراراً مثل (Yes , No , Cancel , . . . etc) , ويعيد التابع عدداً صحيحاً يدل على الزر الذي اختاره المستخدم . أما وسطاء هذا التابع فهي نفسها وسطاء التابع showMessageDialog باستثناء الوسيط الرابع الذي هو أحد الثوابت التالية :

- YES_NO_CANCEL_OPTION : الذي يعطي صندوق حوار مع ثلاثة أزرار yes , no, cancel .

- YES_NO_OPTION : الذي يعطي صندوق حوار مع زرین yes,no

- OK_CANCEL_OPTION : الذي يعطي صندوق حوار مع زرین ok , cancel

وبيين المثال (1-15) هذا التابع .

مثال (1-15)

```
import javax.swing.JOptionPane;
public class Welcome1 {
    public static void main(String[ ] args) {
        // Display Welcome to Java! in a message dialog box
        JOptionPane.showConfirmDialog(null, "Welcome to Java!",
        "My First Example ", JOptionPane.YES_NO_CANCEL_OPTION
        );
        // Exit the program
        System.exit(0);
    } //End main
} //End class
```


وعند تنفيذ البرنامج سنحصل على الخرج بالشكل التالي:



عندما يضغط المستخدم على أحد الأزرار فإن هذا التابع يعيد أحد الثوابت التالية :
OK_OPTION , CANCEL_OPTION , NO_OPTION , YES_OPTION
وذلك وفقاً للزر المضغوط . كما يمكن أن نمرر لهذا التابع وسيطاً خامساً يمثل
أيقونة ، وهو أحد الثوابت التي سبق ذكرها مع التابع showMessageDialog
وفيما يلي مثال على ذلك :

```
int i = JOptionPane.showConfirmDialog ( null , "Do you love Java  
" , JOptionPane.YES_NO_Option , "Java " ,  
JOptionPane.WARING_MESSAGE ) ;
```

المثال (1- 16) يوضح كيف يتم ادخال الرسالة " بسم الله الرحمن الرحيم "
بوساطة نافذة وطباعتها بوساطة نافذة أخرى .

مثال (1-16)

```
import javax.swing.JOptionPane;  
public class Welcome{  
    public static void main(String[ ] args) {  
        // Display in a input dialog box  
        String javaString = JOptionPane.showInputDialog(null,  
        "Enter Your String to " , "My First Example" , JOptionPane.  
        QUESTION_MESSAGE );  
        // Display in a message dialog box
```

```

JOptionPane.showMessageDialog(null, javaString, "My First
Example ", JOptionPane.ERROR_MESSAGE
);
// Exit the program
System.exit(0);
//End main
//End class

```

وعند تنفيذ البرنامج سنحصل على نافذة الدخل والخرج بالشكل التالي:



مثال (17-1)

اكتب برنامجاً يقوم بجمع عددين صحيحين :

العدد الأول number1 و العدد الثاني number2
و المجموع number3

بحيث يتم ادخال العددين بواسطة نافذة وتظهر النتيجة (مجموع العددين) على نافذة .

البرنامج :

```

import javax.swing.JOptionPane;
public class sum{
public static void main(String[ ] args) {
    int number1 , number2,number3;
    // Display sum to Java!
    String javaString =

```

```

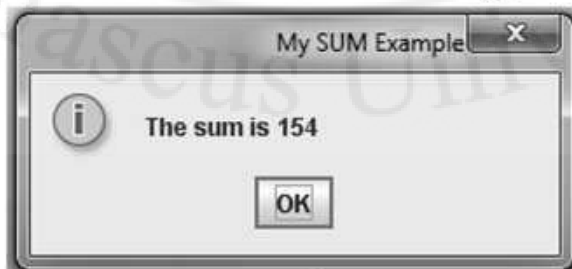
JOptionPane.showInputDialog(null,
    "Enter the first Number ", "My SUM Example",
    JOptionPane.QUESTION_MESSAGE);
number1 = Integer.parseInt(javaString) ;
javaString = JOptionPane.showInputDialog(null,
    "Enter the second Number ", "My SUM Example",
    JOptionPane.QUESTION_MESSAGE);
number2 = Integer.parseInt(javaString) ;
number3 = number1+number2;
// Display Welcome to Java!
JOptionPane.showMessageDialog(null, "The sum is "
+number3 , "My SUM Example ",
JOptionPane.INFORMATION_MESSAGE );
// Exit the program
    System.exit(0);
} //End main
} //End class

```

وعند تنفيذ البرنامج سنحصل على نافذة الدخل والخرج بالشكل التالي:



أما الخرج سيظهر كما يلي :



ملاحظة

يعرض التابع `showInputDialog` صندوق حوار الإدخال (نافذة) المعنون بالنص التالي "My SUM Example" ، وادخل القيمة من النمط `int` ثم اضغط على الزر `OK` لقبول الدخل ، تعاد القيمة الصحيحة كنص يتم إسناده إلى المتحول `javaString` من نمط `String` ، ثم يستخدم التابع `Integer.parseInt(javaString)` لتحويل النص إلى قيمة من النمط `int` ، أما إذا قمنا بإدخال قيمة غير رقمية عندها يحدث خطأ أثناء التنفيذ .

```
=====
=====
=====
```



مسائل عامة

- ١- حدد ما إذا كانت العبارات الآتية صحيحة أم خاطئة:
 - التعليقات تجبر الحاسوب على طباعة النص الذي يلي // على الشاشة عند تنفيذ البرنامج.
 - تتابع الهروب \n يجبر المؤشر على الانتقال إلى سطر جديد.
 - برنامج Java والذي يقوم بطباعة ثلاثة أسطر على الشاشة يجب أن يحتوي على ثلاث عبارات تستعمل (System.out.print).
- ٢- ما الخرج من العبارة الآتية:
`System.out.println( "\n **\n ***\n ****\n" );`
- ٣- اكتب عبارة Java صحيحة تقوم بالآتي:
 - تعريف المتغيرات x ، y ، z و result لتكون من النوع int.
 - الطلب من المستخدم إدخال ثلاثة أرقام صحيحة.
- ٤- حدد ما إذا كانت العبارات الآتية صحيحة أم خاطئة :
 - يجب الإعلان عن المتغيرات قبل استعمالها في البرنامج.
 - يجب تحديد نوع المتغيرات عند الإعلان عنها.
 - لا تفرق Java بين المتغيرات Number و number .
- ٥- ما هو الخرج الناتج عند تنفيذ العبارات البرمجية التالية المكتوبة بلغة Java:
 - 1) $2 * 5 * 5 + 3 * 5 + 7$
 - 2) $(3 * 9 * (3 + (9 * 3 / (3))))$
 - 3) $2 \% 2 + 2 * 2 - 2 / 2$

٦- اكتب برنامجاً بلغة الـ Java يقوم بطباعة مايلي:

Welcome with Java.

This program

Is the third in Java.

٧- اكتب برنامجاً بلغة Java يقوم بحساب وطباعة مساحة مستطيل بعده L و W .

٨ - اكتب برنامجاً بلغة الـ Java يقوم بحساب وطباعة مربع ومكعب أي عدد صحيح a.

```
=====
=====
=====
```



الفصل الثاني

بنى التحكم وحلقات التكرار

Control Structures & Loops





بنى التحكم وحلقات التكرار

٢-١ - مقدمة

نحتاج في التطبيقات العملية لاتخاذ بعض القرارات تبعا لشروط معينة، ومن هنا ظهرت الحاجة لوجود طرق لجعل البرنامج قادراً على تغيير تسلسل تنفيذ التعليمات تبعا للشروط المطلوبة وهو ما يعرف بالجمال الشرطية ، وكثيراً ما نحتاج في البرامج إلى تكرار أمر موجه للحاسوب عدداً من المرات، وتوفر لغة Java عدة وسائل تمكن المبرمج من أداء هذا التكرار، وعادة ما تسمى هذه الوسائل " الحلقات التكرارية " ، ويوجد عدد من الحلقات التكرارية في لغة Java ، وتعد أساليب الشرط والتكرار بمثابة القلب في جسم لغات البرمجة، وبدونها لا يمكن تنظيم أي برنامج. وتوفر لغة Java للمبرمج عدداً من الأساليب والتوابع الفعالة، المتعلقة بهذا الشأن، وتمتاز هذه الأساليب بأنها أساليب بنيوية Structured أي يمكن تنظيم عمليات التحكم والتكرار فيها، بأسلوب ذاتي من بداية العمليات وحتى نهايتها دون تدخل من المبرمج أثناء هذه العمليات، للأشراف على التوجيه والتخطيط لكل خطوة من خطوات البرنامج ، وحيث أن جواب الشرط أما أن يكون صحيحاً true أو خاطئاً false ، فإن لغة Java توفر مرونة كبيرة في استخدام عدد كبير من التوابع ، وفي توجيه البرنامج بطريقة فعالة وفائقة.

٢-٢ - بنى التحكم

Control Structures

إن بنى التحكم عبارة عن بنى تتحكم في تنفيذ خطوات البرنامج من خلال شرط أو مجموعة شروط ، وإن عدم استخدام بنى التحكم يؤدي إلى جعل مترجم لغة Java يقوم بتنفيذ تعليمات البرنامج بنفس الترتيب الذي ظهر في ملف

المصدر ، ويمكن استخدام بنى التحكم لتغيير مسار تنفيذ البرنامج إما بوضع تعليمات شرطية (تنفذ عند تحقيق شرط معين فقط) أو حلقات تكرارية (تكرار مجموعة من التعليمات عدداً من المرات) ، وتزودنا لغة Java بعدد من تعليمات التحكم .

٢-١-٢- if - التعليمية

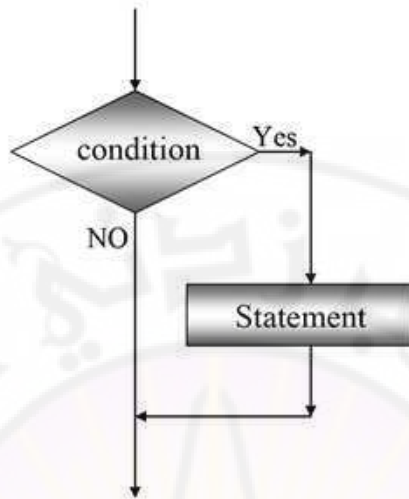
تستخدم التعليمية if في اختبار شرط ما، فإذا تحقق الشرط يتم تنفيذ تعليمية أو مجموعة من التعليمات والشكل التالي يوضح الصيغة العامة لهذه البنية في حالة تنفيذ تعليمية واحدة

```
if ( <condition> )
{
    <statement>;
}
```

حيث يكون condition هو الشرط الواجب تحققه حتى يتم تنفيذ التعليمية statement ، ويمكن أن يكون هذا الشرط على شكل مقارنة بين قيمتين أو متحولين ، وهذه المقارنة يمكن توضيحها من خلال الجدول (١-٢)، ويبين الشكل (١-٢) المخطط التدفقي لبنية التعليمية if .

الشرط	معنى الشرط
if (a == b)	هل قيمة المتغير a تساوي قيمة المتغير b ؟
if (a < b)	هل قيمة المتغير a أصغر من قيمة المتغير b ؟
if (a <= b)	هل قيمة المتغير a أصغر أو تساوي قيمة المتغير b ؟
if (a > b)	هل قيمة المتغير a أكبر من قيمة المتغير b ؟
if (a >= b)	هل قيمة المتغير a أكبر أو تساوي قيمة المتغير b ؟
if (a != b)	هل قيمة المتغير a لا تساوي قيمة المتغير b ؟

الجدول (١-٢) - يبين جدول المقارنة للتعليمية الشرطية



الشكل (١-٢) - يبين المخطط التدفقي للتعليمية if

مثال (1-2)

البرنامج الجزئي التالي يوضح كيفية تنفيذ تعليمية واحدة في حال تحقق الشرط .

```
if (grade >= 90)
```

```
System.out.println(" Congratulations!\n");
```

```
System.out.println(" Your grade is " + grade);
```

وفي حالة تنفيذ مجموعة من التعليمات تكون الصيغة العامة لهذه البنية كما يلي :

```
if(condition)
```

```
{
```

```
statement 1;
```

```
statement 2;
```

```
.....
```

```
statement n;
```

```
}
```

حيث : condition هو الشرط الواجب تحققه حتى يتم تنفيذ التعليمات

. statement 1, statement 2, ..., statement n

مثال (2-2)

البرنامج الجزئي التالي يوضح كيفية تنفيذ مجموعة من التعليمات في حال تحقق الشرط .

```
if( x == 50 )
{
    System.out.print ("x is ");
    System.out.println ("50 " );
}
```

٢-٢ - ٢-٢ - التعليمات if / else

تستخدم التعليمات if / else في اختبار شرط ما ، فإذا تحقق الشرط يتم تنفيذ تعليمة أو مجموعة من التعليمات ، وإذا لم يتحقق الشرط فإنه يتم تنفيذ تعليمة أخرى أو مجموعة من التعليمات ، الصيغة العامة لهذه البنية كما يلي :

```
if(condition)
    statement1;
else
    statement2;
```

مثال (3-2)

البرنامج الجزئي التالي يوضح كيفية تنفيذ تعليمة واحدة في تحقق الشرط

```
if ( x == 10 )
    System.out.println("x is 10 ");
else
    System.out.println("x is not 10 ");
```

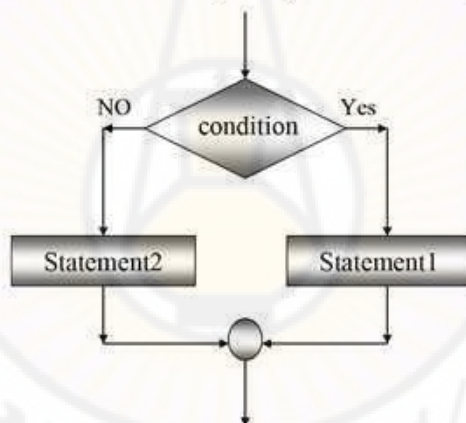
وعند تنفيذ أكثر من تعليمة في حال تحقق الشرط أو عدم تحقق الشرط ، فإن الصيغة العامة لهذه البنية كما يلي :

```

if(condition) {
    statement 1;
    statement 2;
    .....
    statement n;
}
else {
    statement 1;
    statement 2;
    .....
    statement m;
}

```

ويبين الشكل (٢-٢) المخطط التدفقي للتعليمية if/else .



الشكل (٢-٢) - المخطط التدفقي للتعليمية if/else

فإذا تحقق الشرط condition ، يتم تنفيذ التعليمية statement 1 وإذا لم يتحقق الشرط فيتم تنفيذ التعليمية statement 2 .

مثال (4-2)

أكتب برنامجاً بلغة C++ يقوم بإدخال عدد صحيح m ومن ثم يطبع الجملة التالية : "This number m is positive" إذا كان العدد m أكبر أو

يساوي الصفر أو يطبع الجملة : "This number m is negative" إذا
كان العدد m أصغر من الصفر.

البرنامج:

```
import java.util.Scanner;
class Number {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
int d;
System.out.println("Enter the number m ");
d = input.nextInt();
if ( d >= 60 )
System.out.println("The number " + m + " is positive");
else
System.out.println("The number " + m + " is negative ");
} //End main
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Enter the number m: -3

The number -3 is negative

Process completed.

٢-٢-٣ - التعليمة if-else-if

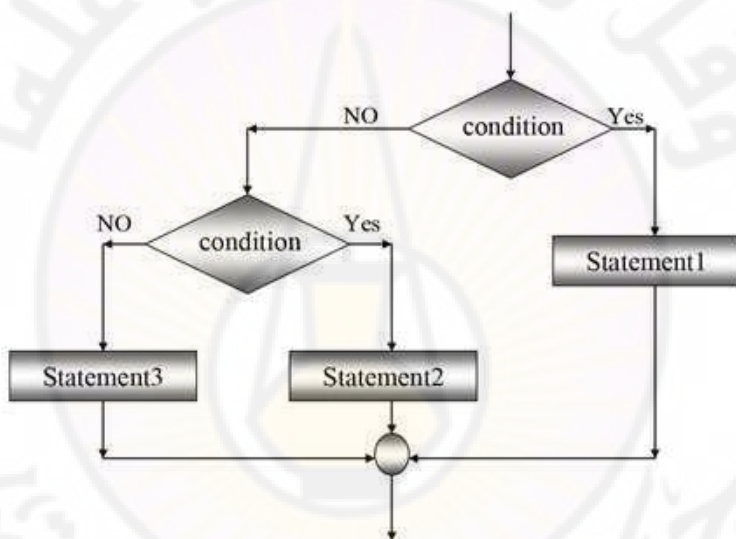
تستخدم التعليمة if-else-if في اختبار الشرط الأول ، فإذا تحقق هذا الشرط فإن التعليمة الأولى (أو مجموعة من التعليمات) تنفذ وإذا لم يتحقق الشرط الأول يتم اختبار الشرط الثاني ، فإذا تحقق الشرط الثاني يتم تنفيذ التعليمة الثانية (أو مجموعة من التعليمات) وإلا يتم تنفيذ التعليمة الثالثة (أو مجموعة من التعليمات) ، والصيغة العامة لهذه البنية هي التالي:

```

if (condition1)
if (condition1)
    statement1;
else if(condition2)
    statement2;
else
    statement3;

```

وبين الشكل (٢ - ٣) المخطط التدفقي للتعليمية if-else-if .



الشكل (٢ - ٣) - يبين المخطط التدفقي لبنية التعليمية if-else-if

مثال (5-2)

اكتب برنامجاً بلغة Java يقوم إدخال رقم الشهر month (1,2,3,...,12) .
 وطباعة الفصل الذي ينتمي إليه الشهر [Winter(12,1,2), Spring(3,4,5), Summer(6,7,8), Autumn(9,10,11)] .

البرنامج :

// Demonstrate if-else-if statements.


```

class IfElse1 {
public static void main(String args[] ) {
int month = 4; // April
String season;
if(month == 12 || month == 1 || month == 2)
season = "Winter";
else if(month == 3 || month == 4 || month == 5)
season = "Spring";
else if (month == 6 || month == 7 || month == 8)
season = "Summer";
else if (month == 9 || month == 10 || month == 11)
season = "Autumn";
else
season = "Bogus Month";
System.out.println("April is in the " + season + ".");
} //End main
} //End Class IfElse

```

وعند تنفيذ البرنامج نحصل على الخرج التالي :

RESULT

Enter your month : 4
 April is in the Spring
 Process completed.

١-٢-٤ - تعليمة if المركبة

يمكن استخدام تعليمات if المتداخلة ، بحيث تحتوي التعليمة if على تعليمة if واحدة أو أكثر ، الصيغة العامة كما يلي :

```

if (expression1) {      // if1
    statement1;
    Statement2;
    if (expression2) {  // if2
        statement1;
        statement2;
    } //end if2
} //end if1

```

مثال (2-6)

البرنامج الجزئي التالي يوضح كيفية استخدام بنية if المتداخلة .

```

if (u > v) {
    a = 1;
    b = 2;

    if ( u > z) {
        x = 11;
        y = 12;
    }
}

```

مثال (2-7)

اكتب برنامجاً بلغة Java يقوم بإدخال درجة الطالب grade ويختبر هل هي A أو B أو C أو D أو F . (الإدخال والإخراج عن طريق الواجهات) .
حيث : $A \geq 90$ ، $B \geq 80$ ، $C \geq 70$ ، $D \geq 60$ ، $F < 60$.

البرنامج :

```

import javax.swing.JOptionPane;
public class IfElse4{
    public static void main(String[] args) {
        double grade;
        String Mark =" ";

```

```

// Display sum to Java!
String javaString = JOptionPane.showInputDialog(null,
    "Enter The mark ", "My Example(4)",
    JOptionPane.QUESTION_MESSAGE );
grade = Double.parseDouble(javaString);
if (grade >= 90)
    Mark = " A";
else if (grade >= 80)
    Mark = " B";
else if (grade >= 70 )
    Mark = " C";
else if (grade >= 60 )
    Mark = " D";
else if (grade < 60)
    Mark = " F  You must take this course again";
// Display Welcome to Java!
JOptionPane.showMessageDialog(null, "The Mark is "+Mark
, "My Example(4) ", JOptionPane.INFORMATION_MESSAGE );
// Exit the program
System.exit(0);
} //end main
} //End class

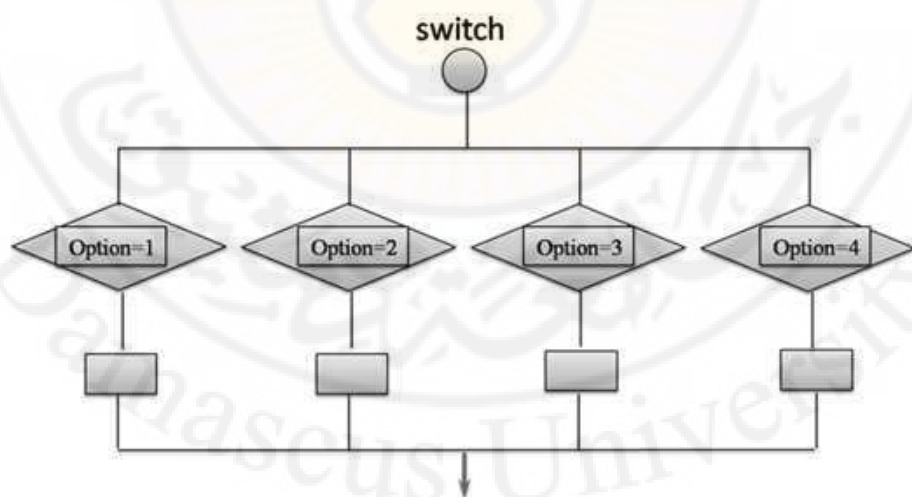
```

وعند تنفيذ البرنامج سنحصل على النوافذ التالية :



٢-٥-٢ - بنية التعليم switch

تستخدم هذه البنية جملة الشرط المغلقة Close Condition أي الشرط الذي يأخذ قيمة محددة (رقمية أو حرفية) ولهذا يمكن القول بأن هذه البنية تتعامل مع متحول واحد يمكن أن يأخذ في كل مرة قيمة محددة من خلال مجموعة من القيم. التعليم switch هي تعليمة تتحكم بتدفق البرنامج ، وتبدأ بتعبير expression وتنقل التحكم إلى إحدى عبارات case بناءً على قيمة ذلك المتحول وتعمل التعليم switch مع الأنماط الأولية: int و short و byte و char وأنماط التعداد enumeration ، وتستخدم عبارة break من أجل الخروج من العبارة switch ، وإذا لم تحتوي عبارة case على الكلمة break فإنه سيتم تنفيذ السطر الموجود بعد نهاية العبارة case ، ويستمر التنفيذ إلى أن يتم الوصول إلى عبارة break أو الوصول إلى نهاية عبارة switch ، ويُسمح بعلامة default واحدة فقط في عبارة switch. ويبين الشكل (٢-٤) المخطط التدفقي للتعليم switch .



الشكل (٢-٤) - يبين المخطط التدفقي للتعليم switch

والشكل العام لبنية التحكم **switch** هو التالي:

```
switch( expression) {           // int expression
    case value 1: statement(s); // int value
        break;
    case value 2: statement(s); // int value
        break;
    :
    case value n: statement(s); // int value
        break;
    default :statement(s);
} //end switch
```

هذا الشكل يمثل الاحتمالات التي يمكن أن يأخذها المتحول variable والذي يمكن أن يكون على شكل تعبير expression، كل احتمال يتم وضعه داخل جملة باستخدام العبارة case ثم بعد ذلك كتابة الحدث الذي يمكن أن يكون تعليمة أو مجموعة تعليمات ولا بد أن ينتهي الحدث بواسطة العبارة break التي تمثل نهاية الجملة، أما الاحتمال default يمثل الحدث الملازم للحالة خارج النطاق ، كما هو موضح في المثال (8-2) .

مثال (8-2)

```
switch(food) {
    case 1:
        System.out.println("Chicken") ;
        break;
    case 2:
        System.out.println("Pizza") ;
        break;
    default:
        System.out.println("Sorry, we are out");
}
```


مثال (9-2)

- اكتب برنامجاً بلغة C++ يقوم بما يلي (استخدم التعليمة switch) :
- إدخال رقم الشهر month (1,2,3,...,12).
 - طباعة الفصل الذي ينتمي إليه الشهر [Winter(12,1,2) , Spring(3,4,5), Summer(6,7,8), Autumn(9,10,11)].

البرنامج :

// An improved version of the season program.

```
public class Switch {  
    public static void main(String args[ ]) {  
        int month = 4;  
        String season;  
        switch (month) {  
            case 12:  
            case 1:  
            case 2:  
                season = "Winter";  
                break;  
            case 3:  
            case 4:  
            case 5:  
                season = "Spring";  
                break;  
            case 6:  
            case 7:  
            case 8:  
                season = "Summer";  
                break;  
            case 9:  
            case 10:  
            case 11:  
                season = "Autumn";  
                break;  
            default:  
                season = "Invalid month";  
                break;  
        }  
        System.out.println("The season is: " + season);  
    }  
}
```

```

break;
default:
season = "Bogus Month";
}
System.out.println("April is in the " + season + ".");
} //End main
} //end Class Switch

```

وعند التنفيذ الأول للبرنامج سنحصل على الخرج التالي:

RESULT

April is in the Spring

Process completed.

٢-٢-٦- بنية تعليمة التحكم (?:)

هذه البنية هي أداة مكافئة للتعليمة if-else ، وقد مر معنا كيفية استخدامها

والشكل العام لبنية التحكم (?:) :

Variable=(condition)?result1:result2;

عندما يكون الشرط condition محققاً (true) فإن Variable =result1 ،

وعندما يكون الشرط condition غير محققاً (false) فإن Variable =result2

مثال (10-2)

اكتب برنامجاً يقوم بما يلي :

- إدخال العددين a و b . - إيجاد العدد الأكبر و طباعته .

البرنامج:

```

import java.util.Scanner;
public class Max1 {
public static void main(String args[ ] ) {
int a,b,y;

```

```

Scanner input = new Scanner( System.in );
System.out.println("Enter the value of a & b: ");
a = input.nextInt( );
b = input.nextInt( );
y = ( a > b ) ? a : b ;

System.out.println("y= " , y) ;
} //End main
} //End Program

```

وعند التنفيذ الأول للبرنامج سنحصل على الخرج التالي:

RESULT
Enter the value of a & b : 99 55
y=99
Process completed.

وعند التنفيذ الثاني للبرنامج سنحصل على الخرج التالي:

RESULT
Enter the value of a & b : 10 88
y=88
Process completed.

مثال (2-11)

مندوب مبيعات يتقاضى راتباً شهرياً أساسياً مقداره (10000 ليرة) ، ويتقاضى عمولة قدرها 4% إذا كانت المبيعات الشهرية لا تتجاوز (200 000) ليرة ، ويتقاضى عمولة قدرها 6% إذا كانت المبيعات الشهرية لا تتجاوز (400 000) ليرة، ويتقاضى عمولة قدرها 7% إذا كانت المبيعات الشهرية أكثر من ذلك. والمطلوب : اكتب برنامجاً بلغة java يقوم بما يلي :

- إدخال الراتب الشهري وقيمة المبيعات الشهرية وطباعتها

- حساب وطباعة الدخل الشهري الكامل للموظف .
- ملاحظة :** افترض salary الراتب الشهري الأساسي ، amount المبيعات الشهرية ، W العمولة ، Allsalary الراتب الشهري الكامل .

البرنامج :

```
import java.util.Scanner;
public class Ssalary {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
int salary , amount ;
double W ,Allsalary ;
System.out.println("inter your salary & amount");
salary = input.nextInt(); amount = input.nextInt() ;
System.out.println("-----");
if( amount <= 200000 )
W = amount*4/100;
else if( amount <= 400000 )
W = amount*6/100;
else
W = amount*7/100;
System.out.println( " W= " + W);
System.out.println("-----");
Allsalary = salary + W ;
System.out.println("Allsalary = " + Allsalary);
System.out.println("-----");
} //End main
} //End Program
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

salary & amount
10000 350000

W=21000 // العمولة

Allsalary = 31000 // الراتب

Process completed.

اكتب برنامجاً بلغة Java يقوم بإدخال عددين صحيحين x ، y ، وإدخال الرمز ch من نوع `char` بحيث إذا كان الرمز المدخل (+) يقوم بجمع العددين ، وإذا كان (-) يقوم بطرح العددين ، وإذا كان (*) يقوم بطرب العددين ، وإذا كان (/) يقوم بقسمة العددين وطباعة النتيجة .

البرنامج :

```
import java.util.Scanner;
public class Arethmatic {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
double x , y , z;
char ch ;
System.out.println("inter yuor x & y & ch ");
x = input.nextDouble( );
y = input.nextDouble( );
String input1=input.next( );
ch = input1.charAt( 0);
System.out.println("-----");
System.out.println( " x = " +x+" y="+y) ;
System.out.println("-----");
if( ch == '+')
System.out.println( x + y );
//-----
else if( ch == '-')
System.out.println( x - y );
//-----
else if( ch == '*')
System.out.println( x * y );
//-----
else if( ch == '/')
System.out.println( x / y );
//-----
```



```

else
    System.out.println( "Sumbol Erorr Input ");
    System.out.println( "-----");
} //End main
} //End Program

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

```

RESULT
x & y & ch
7    8    *
-----
x = 7   y=8
z= 56
-----
Process completed.

```

٢-٣ - بنى حلقات التكرار

Looping Structures

إن استخدام حلقات التكرار يُعد من المواضيع المهمة في البرمجة، حيث إن العديد من المسائل تحتاج إلى تكرار تعليمة أو مجموعة تعليمات أكثر من مرة، أو يتم التكرار لطالما أن شرط ما محقق، وفي لغة C++ هناك ثلاث بنى للتكرار وهي : حلقة for ، حلقة while ، وحلقة do-while .

٢-٣-١ - بنية الحلقة for

تستخدم الحلقة for لتكرار مجموعة من التعليمات عدداً محدداً من المرات، وتحتوي على ثلاثة أقسام هي : التهيئة والتعبير والتحديث .

□ **التهيئة** : هو تعبير يقوم بتهيئة متحول الحلقة initialization , لإعطاء قيمة ابتدائية لعداد (متحول) الحلقة .

□ التعبير termination : يمثل شرط التوقف ويتم اختياره في بداية كل تكرار

ويحدث التكرار فقط في حال كانت قيمة التعبير هي true .

□ التحديث increment : يمثل مقدار الزيادة أو النقصان لعدد (متحول) الحلقة.

الصيغة العامة :

for (initial exp ; condition exp ; incre/decre exp)

<statement>;

Or compound statement (block) { ... }

حيث:

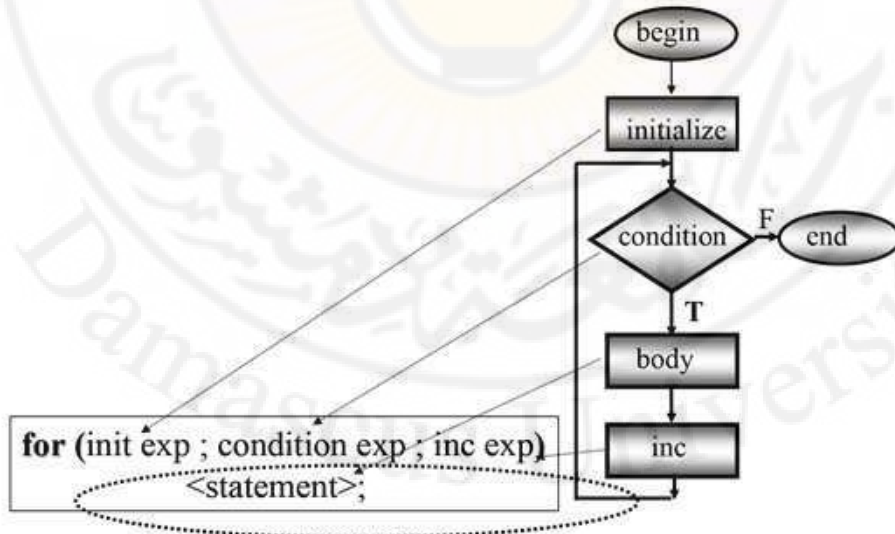
initial- exp - القيمة الابتدائية لعدد حلقة التكرار .

condition exp - شرط حلقة التكرار وغالباً ما يحوي قيمة نهائية.

incre/decre exp - الزيادة المنتظمة لعدد الحلقة .

Statement - تعليمة يتم تكرار تنفيذها لعدد من المرات .

يبين الشكل (٥ - ٢) الحلقة for وما يقابلها من المخطط .



الشكل (٥ - ٢) - يبين الحلقة for وما يقابلها من المخطط

تقوم الحلقة for مبتدئة بقيمة العداد الابتدائية بتنفيذ التعليمة Statement في المرة الأولى، وفي المرة التالية تزداد القيمة الابتدائية للعداد بمقدار الزيادة ثم تنفذ التعليمة Statement مرة ثانية، وهكذا حتى يصبح الشرط condition غير محققاً لإنهاء عملية التكرار والخروج من الحلقة for .

مثال (2-13)

اكتب برنامجاً بلغة C++ يقوم بطباعة الأعداد الصحيحة من 1 حتى 10.

البرنامج:

```
public class Increment {  
    public static void main(String args[ ] ) {  
        for ( int i=1;i<=10 ; i++ ) {  
            System.out.print ( " " +i ) ;  
        } //End for  
        System.out.print ( ) ;  
    } //End main  
} //End Program
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

1 2 3 4 5 6 7 8 9 10

Process completed

ضمن سياق حلقة التكرار for يمكن استخدام في حلقة for النقصان المنتظم في العداد وذلك بدلاً عن الزيادة المنتظمة في العداد والمثال (7-14) يوضح ذلك.

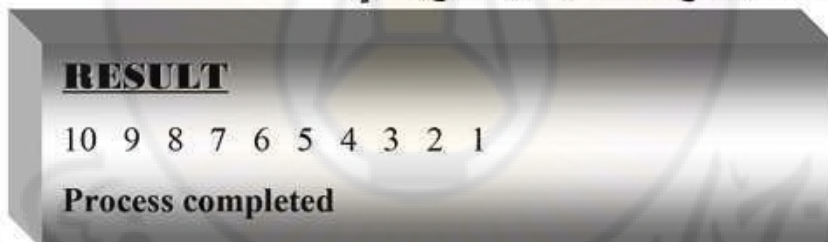
مثال (2-14)

اكتب برنامجاً بلغة Java يقوم بطباعة الأعداد الصحيحة من 10 حتى 1 على سطر واحد .

البرنامج:

```
class Increment {  
public static void main(String args[ ]) {  
for ( int i=10;i<=1 ; i-- ) {  
    System.out.print ( " " +i );  
} //End for  
System.out.print ( ) ;  
} //End main  
} //End Program
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



عندما يحتوي جسم الحلقة على أكثر من تعليمة يجب وضع جسم الحلقة ضمن قوسين { } كما يلي :

```
for (initial exp ; condition exp ; incre exp )  
{  
    Statement 1;  
    .....  
    Statement n;  
}
```

مثال (2-15)

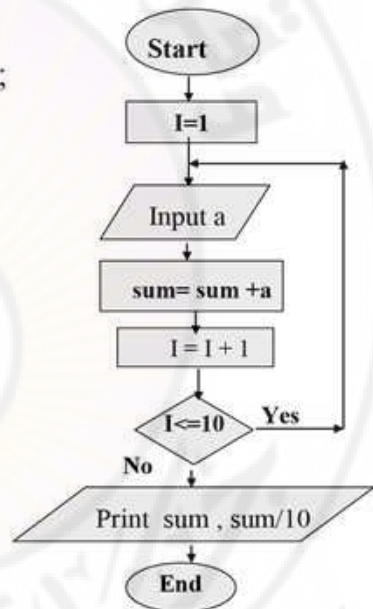
اكتب برنامجاً بلغة Java يقوم بحساب وطباعة المتوسط الحسابي لعشرة أعداد حقيقية يتم إدخالها من لوحة المفاتيح .

البرنامج:

// Add the numbers from 0 to 9 and print the average

```
import java.util.Scanner;
public class average {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
double N,avr , sum = 0;
for (int i = 0; i <= 9; i++){
    N = input.nextDouble( ) ;
    sum = sum +N;
} //End for
avr=sum/10;
```

```
System.out.println(" avre= "+avr ) ;
} //End main
} //End Program
```



وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Enter your numbers :

2.4 5.6 6.7 5.7 7.8 5.5 6.7 4.6 6.0 7.0

average=5.8

Process completed

مثال (2-16)

يقوم البرنامج التالي بجمع سلسلة تبدأ بالقيمة 0.01 وتنتهي بالقيمة 1.0، وستتم زيادة الأعداد في السلسلة بمقدار 0.01 ، وذلك كما يلي :

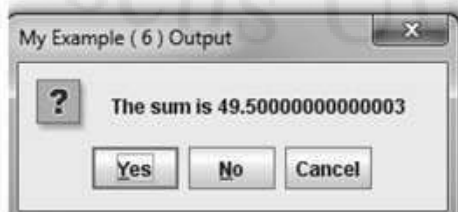
$$0.01 + 0.02 + 0.03 + \dots \text{etc}$$

- نستخدم الحلقة for من أجل الحصول على المجموع sum وتنتهي الحلقة for عندما يتجاوز i القيمة 1.0.
- نستخدم واجهة (صندوق حوار) لطباعة النتيجة .

البرنامج :

```
//TestSum.java: Compute sum = 0.01 + 0.02 + ...
import javax.swing.JOptionPane;
public class SumFor {
    /** Main method */
    public static void main(String[ ] args) {
        // Initialize sum
        double sum = 0;
        // Add 0.01, 0.02, 0.03, ..., 1.0 to sum
        for (double i = 0.01 ; i <= 1.0 ; i = i + 0.01 )
            sum = sum+i;
        // Display result
        JOptionPane.showMessageDialog(null, "The sum is " + sum
            , "My Example ( 6 ) Output",
            JOptionPane.YES_NO_CANCEL_OPTION);
        System.exit(0);
    } //End main
} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



مثال (2-17)

يعمل 100 عامل في معمل نسيج بشكل ساعي , لنفرض أن H هي عدد ساعات العمل وأن R هي أجرة الساعة الواحدة وأن W هو الأجر اليومي . والمطلوب :

اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال وطباعة عدد ساعات العمل وأجرة الساعة .
- حساب وطباعة الأجر اليومي لكل عامل .

البرنامج :

```
import java.util.Scanner;
public class Worker {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
double H , R , W;
for (int i = 1; i <= 100; i++){
System.out.println(" Enter R & H " );
R = input.nextDouble( );
H = input.nextDouble( );
W = R * H ;
System.out.println(" W =" + W ) ;
} //End for
} //End main
} //End Program
```

مثال (2-18)

يقوم البرنامج التالي بحساب وطباعة جدول الضرب على نافذة بحيث :

- نستخدم حلقات for المتداخلة .
- يقوم البرنامج بعرض العنوان في السطر الأول ثم يقوم بإضافة خطوط متقطعة (-) في السطر الثاني من الخرج .
- تعرض حلقة for الأولى الأعداد من 1 حتى 9 في السطر الثالث .

- الحلقة التالية المتداخلة بحيث يتم عرض ناتج ($i * j$) بعدد مرات المتحول j والذي يبدأ من 1 حتى 9 .
- استخدمت التعليمة `if` في الحلقة الداخلية من أجل رصف الناتج بشكل مناسب , حيث إذا كانت قيمة الناتج مؤلفة من رقم وحيد يتم عرضه مع وضع مسافة إضافية قبله .

البرنامج :

```
// TestMulTable.java: Display a multiplication table
import javax.swing.JOptionPane;
public class MulTable {
    /** Main method */
    public static void main(String[ ] args) {
        // Display the table heading
        String output = "جدول الضرب\n Multiplication Table\n";
        output += "-----\n";
        // Display the number title
        output += " |";
        for (int j = 1; j <= 9; j++)
            output += " " + j;
        output += "\n";
        // Print table body
        for (int i = 1; i <= 9; i++) {
            output += i + " |";
            for (int j = 1; j <= 9; j++) {
                // Display the product and align properly
                if (i * j < 10)
                    output += " " + i * j;
                else
                    output += " " + i * j;
            }
            output += "\n";
        }
    }
}
```

وضع فراغ واحد

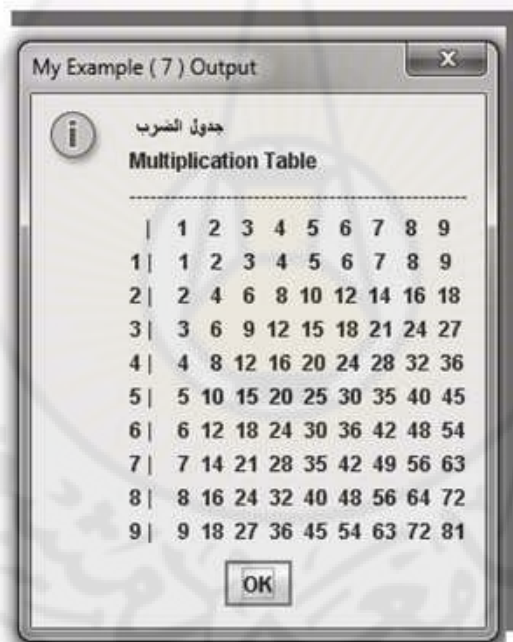
وضع فراغ واحد

```

}
// Display result
JOptionPane.showMessageDialog(null, output,
    "My Example ( 7 ) Output",
    JOptionPane.INFORMATION_MESSAGE);
System.exit(0);
} // End main
} // End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



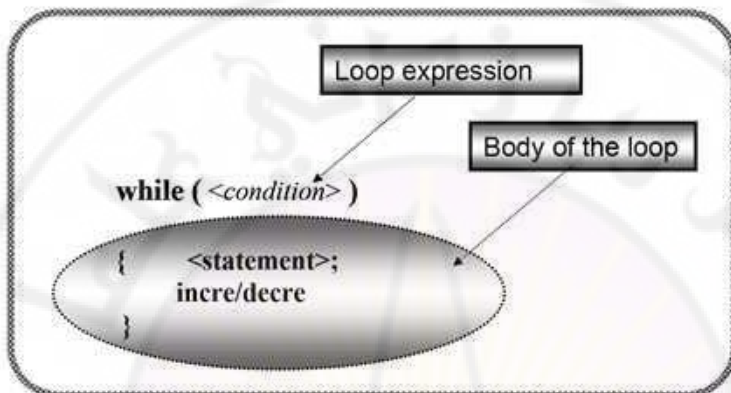
٢-٣ - بنية الحلقة while

تستخدم الحلقة while لتكرار تنفيذ تعليمة أو مجموعة تعليمات عدداً من المرات ويتوقف هذا العدد على شرط موجود في جسم الحلقة ، والشكل العام لبنية الحلقة while هو التالي:

```

while (condition) {
    Statement;
    increment / decrement;
}

```



أو:

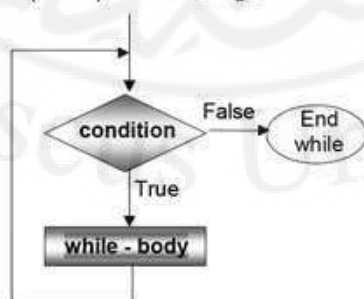
حيث:

condition - شرط حلقة التكرار .

Increment / decrement - الزيادة المنتظمة في العداد أو النقصان .

Statements - مجموعة من التعليمات يتم تكرار تنفيذها عدداً من المرات .

يتم في الحلقة while تكرار تنفيذ التعليمات statements ما دام الشرط condition محققاً أي true , وعندما يصبح الشرط condition غير محقق false يتوقف تنفيذ التعليمات في الحلقة, وينتقل تنفيذ تسلسل تنفيذ البرنامج إلى التعليمة التي تلي الحلقة while , ويبين الشكل (٦-٢) المخطط التدفقي لحلقة التكرار while .



الشكل (٦-٢) - يبين المخطط التدفقي لحلقة التكرار while

مثال (7-19)

اكتب برنامجاً بلغة Java يقوم بحساب مجموع الأعداد الصحيحة من 1 حتى 10 وطباعتها باستخدام الحلقة while .

البرنامج:

// Add the numbers from 0 to 9 and print the result while loop example

```
public class SumNumber {  
    public static void main(String args[ ] ) {  
        int i=0, sum = 0;  
        while(i <= 9) {  
            sum = sum + i;    // sum += i; accumulator  
            i++;             // i = i + 1; counter  
        }  
        System.out.println("Sum = " + sum + ".\n");  
    } //End main  
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Sum = 45.

Process completed

مثال (2-20)

لدينا 300 طالباً يدرسون مقرر البرمجة (2) والمطلوب :

اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال درجة الطالب d وطباعة كلمة ناجح " Passed " إذا كانت درجة الطالب $d \geq 60$ ويطبع كلمة راسب " Failed " إذا كانت غير ذلك .
- استخدم الحلقة while

```

import java.util.Scanner;
public class Student {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
double d;
int i=1; // القيمة الابتدائية
while(i <= 30) // الشرط
{
System.out.println(" Enter degree ");
d = input.nextDouble( );
if ( d >= 60 )
System.out.println(" Passed = "+d);
else
System.out.println(" Failed = "+d);
i++; } // i = i + 1; counter
} //End main
} //End Class

```

الحلقة for هي حلقة تكرار عديدة تعتمد على العدّاد وينتهي التكرار بانتهاء عدد مرات التكرار، أما بنية الحلقة while هي حلقة شرطية أي تعتمد على الشرط الذي يلي الأمر while حيث تتكرر التعليمات التي تليه طالما كان الشرط صحيحاً ، وتنتهي الحلقة عندما يكون الشرط غير محققاً ، وبالتالي الاستخدام الأمثل للحلقة for هو تكرار تعليمة أو مجموعة تعليمات أكثر من مرة بشرط أن يكون عدد مرات التكرار محدداً، أما الاستخدام الأمثل للحلقة while هو التكرار بناء على شرط معين.

مثال (21-2)

يقوم البرنامج التالي بإدخال وحساب مجموع عدد غير محدد من الأرقام الصحيحة

- عند ادخال القيمة 0 (صفر) يشير إلى نهاية البرنامج .
- استخدم الواجهات من أجل إدخال القيم وطباعة المجموع .

البرنامج :

```
// TestWhile.java: Test the while loop
import javax.swing.JOptionPane;
public class TestWhile {
    /** Main method */
    public static void main(String [ ] args) {
        int data , sum = 0;
        // Read an initial data
        String dataString = JOptionPane.showInputDialog(null,
            "Enter an int value, \n the program exits if the input is 0",
            "My Example( 5 ) ", JOptionPane.QUESTION_MESSAGE);
        data = Integer.parseInt(dataString);
        // Keep reading data until the input is 0
        while (data != 0) {
            sum =sum+ data;
            // Read the next data
            dataString = JOptionPane.showInputDialog(null,
                "Enter an int value, \nthe program exits if the input is 0",
                "My Example ( 5)", JOptionPane.QUESTION_MESSAGE );
            data = Integer.parseInt(dataString);
        } // End while
        JOptionPane.showMessageDialog(null, "The sum is " + sum,
            "My Example(5) Output",
            JOptionPane.INFORMATION_MESSAGE );
        System.exit(0);
    } //end main
} //End class
```

وعند تنفيذ البرنامج سنحصل الواجهات التالية:



٣-٢ - ٣-٣ - بنية الحلقة do / while

تُستخدم الحلقة do-while لتكرار تنفيذ تعليمة أو مجموعة من التعليمات أكثر من مرة بناءً على شرط معين كما هو الحال مع الحلقة while ولكن الفرق بينهما هو أن الحلقة while تختبر الشرط أولاً ، فإذا كان الشرط صحيحاً true عندئذ تنفذ التعليمات الموجودة داخل الحلقة ، وإلا ويتم الانتقال إلى تنفيذ التعليمات التي تلي الحلقة، أما الحلقة do-while فتتخذ التعليمات الموجودة ضمن الحلقة ثم تختبر الشرط، فإذا كان الشرط صحيحاً true يتم تكرار تنفيذ الحلقة وإلا تُوقف التكرار أي يتم تنفيذ الحلقة مرة واحدة على الأقل ، والصيغة العامة لبنية الحلقة do-while كما يلي:

```

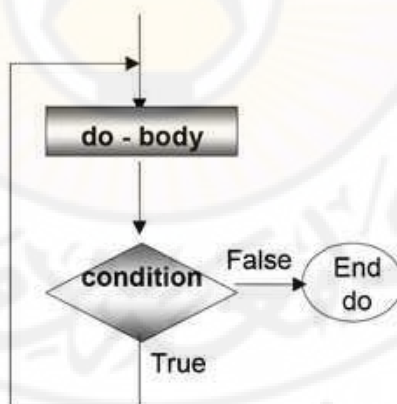
do
{
    Statement1;
    :
    :
    Statement n;
    increment or decrement;
}
while (condition);

```

حيث:

condition - شرط تكرار الحلقة وغالباً ما يحوي قيمة نهائية.
 decrement / increment - الزيادة المنتظمة في العداد أو النقصان.
 Statements - مجموعة التعليمات التي يتم تكرار تنفيذها عدداً محدداً من المرات.

يبين الشكل (٧ - ٢) المخطط التدفقي لبنية الحلقة do- while .



الشكل (٧ - ٢) - يبين المخطط التدفقي لبنية الحلقة do- while

مثال (7-22)

اكتب برنامجاً بلغة Java يقوم بإدخال 30 عدداً حقيقياً وطباعة كلمة "Positive" إذا كان العدد أصغر من الصفر ، ويطبع كلمة "Negative" إذا كان العدد أكبر أو يساوي الصفر ، وذلك باستخدام الحلقة do-while .

البرنامج:

```
import java.util.Scanner;
public class Number {
    public static void main(String args[ ] ) {
        Scanner input = new Scanner( System.in );
        double number;
        int i=1; // القيمة الابتدائية
        do {
            System.out.println(" Enter degree ");
            number = input.nextDouble( );
            if (number >= 60 )
                System.out.println(" Positive = "+ number );
            else
                System.out.println(" Negative = "+ number );
            i++; } // i = i + 1; counter
        while(i <= 30); // الشرط
    } //End main
} //End Class
```

مثال (2-23)

لدى شركة تجارية 30 مندوب مبيعات يتقاضى كل مندوب راتباً شهرياً أساسياً مقداره (10000) ليرة ، ويتقاضى عمولة قدرها 4% إذا كانت المبيعات الشهرية لا تتجاوز (200000) ليرة ، ويتقاضى عمولة قدرها 6% إذا كانت

المبيعات الشهرية لا تتجاوز (400000) ليرة ، ويتقاضى عمولة قدرها 7%
إذا كانت المبيعات الشهرية أكثر من ذلك . والمطلوب :

اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال الراتب الشهري وقيمة المبيعات الشهرية وطباعتها لجميع المندوبين
salary ,W ,Allsalary, amont

- حساب وطباعة الدخل الشهري الكامل للمندوبين .

ملاحظة : افترض salary الراتب الشهري الأساسي ، amont المبيعات الشهرية
، W العمولة ، Allsalary الراتب الشهري الكامل .

البرنامج :

```
import java.util.Scanner;
class Ssalary {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
int salary , amont ;
double W ,Allsalary ;
int i=1 ; i<= 3 ; i++ )
do{
System.out.println("inter yuor salary & amont");
salary = input.nextInt( );
amont = input.nextInt( );
System.out.println("-----");
if( amont <= 200000 )
W=amont*4/100;
else if( amont <= 400000 )
W=amont*6/100;
else
W=amont*7/100;
System.out.println("W= " + W);
System.out.println("-----");
```

```

Allsalary = salary + W ;
System.out.println("Allsalary = " + Allsalary);
System.out.println("-----");
i++;
} while (i <= 30 ) ;           //End for
} //End main
} //End Program

```

٢-٤- تعليمات التفريع

Branching Statement

٢-٤-١- تعليمة التوقف break

تُستخدم تعليمة التوقف break من أجل تغيير مجرى تدفق التحكم ضمن البرنامج وتسبب التعليمة break عند تنفيذها مع الحلقات :
for, while , do- while
الخروج من هذه الحلقات ويتابع البرنامج بعدها التنفيذ مع أول تعليمة تلي الحلقة.

مثال (24-2)

البرنامج التالي يوضح لنا كيفية استخدام التعليمة break مع بنية التكرار for.

البرنامج:

//Using the break statemen in a for structure

```

public class break1 {
public static void main(String args[ ] ) {
int x;
for ( x = 1; x <= 10; x++ )
{
if ( x == 4 )
break ;
System.out.print ("x= " + x+ " ");
}
System.out.println( );
}

```

```
System.out.println("\nBroke out of loop at x == " + x+ "\n");
} //End main
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

```
x= 1    x= 2    x= 3
Broke out of loop at x == 4
Process completed.
```

٢-٤-٢ - تعليمة الاستمرار **continue**

تُستخدم تعليمة الاستمرار **continue** من أجل تغيير مجرى تدفق التحكم ضمن البرنامج ، وتسبب التعليمة **continue** عند تنفيذها مع إحدى الحلقات :
For, while , do / while
تتجاوز تعليمة الاستمرار **continue** ما تبقى من التعليمات في جسم بنية التكرار والمتابعة مع المرور التالي في الحلقة .

مثال (2-25)

البرنامج التالي يوضح لنا كيفية استخدام التعليمة **continue** مع الحلقة **for** .
البرنامج:

```
//Using the continue statemen in a for structure
public class continue1 {
public static void main(String args[ ] ) {
int x;
for ( x = 1; x<= 10; x++ ) {
if ( x == 5 )
```

```

    continue ;
    System.out.print ( " " + x);
}
System.out.println ( );
System.out.println("\nUsed continue to skip printing the value 5 ");
} //End main
} //End Class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

1 2 3 4 6 7 8 9 10
 Used continue to skip printing the value 5
 Process completed.

٢-٤-٣- تعليمة return

تُستخدم التعليمة return للخروج من التابع الحالي , ويُعاد تحكم التنفيذ إلى التعليمة التي تلي تعليمة استدعاء التابع مباشرة , وتملك التعليمة return شكلين:

- الأول : يُعيد قيمة (كما هي في التوابع التي تعيد قيمة) .
- الثاني : لا يعيد قيمة .

لإعادة قيمة يكفي أن نضع القيمة المراد إعادتها بعد العبارة return مباشرة .

مثال :

```
return ++count ;
```

يجب على نمط القيمة المعادة أن يطابق نمط التابع , وعند التصريح عن التابع على أنه void , لا داعي لإعادة أي قيمة ويكفي استخدام العبارة return فقط للخروج من التابع كما في الشكل التالي :

```
return ;
```


مسائل عامة

- ١- اكتب برنامجاً بلغة Java يقوم بحساب مجموع الأعداد الصحيحة الفردية من 10 حتى 20 وطباعتها .
- ٢- اكتب برنامجاً بلغة Java يقوم بحساب مجموع الأعداد الصحيحة الزوجية من 50 حتى 200, وطباعتها , ثم ارسم المخطط التدفقي .
- ٣- اكتب برنامجاً بلغة Java يقوم بحساب مجموع مضاعفات العدد 5 للأعداد الصحيحة من 1 حتى 100, وطباعتها .
- ٤- لدينا 300 طالباً يدرسون مقرر البرمجة والمطلوب :
اكتب برنامجاً بلغة C++ يقوم بما يلي :
 - إدخال درجة الطالب d وطباعة كلمة ناجح " Passed " إذا كانت درجة الطالب $d \geq 60$ ويطبع كلمة راسب " Failed " إذا كانت غير ذلك .
 - استخدم الحلقة while
- ٥- اكتب برنامجاً بلغة Java يقوم بما يلي :
 - إدخال عددين صحيحين a و b .
 - إيجاد وطباعة المضاعف المشترك الأصغر للعددين المُدخلين.
- ٦- اكتب برنامجاً بلغة Java يقوم بما يلي:
 - إدخال عددين صحيحين a و b .

- إدخال قيمة صحيحة للمتحول m بحيث إذا كانت $m=1$ يقوم البرنامج بإيجاد العدد الأكبر من بين العددين المُدخلين ، وإذا كانت $m=2$ يقوم البرنامج بحساب المتوسط الحسابي للعددين المُدخلين ، وإذا كانت $m=3$ يقوم البرنامج بإيجاد المضاعف المشترك الأصغر للعددين المُدخلين .
- استخدم الحلقة switch .

٧- اكتب برنامجاً بلغة Java يقوم بما يلي:

- إدخال عددين صحيحين a و b .
- إدخال قيمة محرفية (* أو / أو + أو - أو أي قيمة أخرى) .
- استخدم التعليمة switch من أجل إجراء العمليات الحسابية الأساسية (الضرب و القسمة و الجمع و الطرح) على العددين المُدخلين a و b .

٨- اكتب برنامجاً بلغة Java يقوم بإيجاد وطباعة درجة الحرارة العظمى خلال شهر تشرين الأول (31 يوم)، باستخدام الحلقة while .

٩- اكتب برنامجاً بلغة Java يقوم بإدخال عدد صحيح m ومن ثم يطبع الجملة التالية : "This number m is positive" إذا كان العدد m أكبر أو يساوي الصفر أو يطبع الجملة : "This number m is negative" إذا كان العدد m أصغر من الصفر .

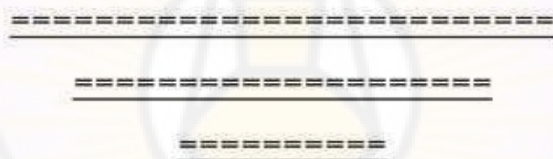
١٠- اكتب برنامجاً بلغة Java يقوم بإدخال a,b,c وحساب وطباعة جذور معادلة من الدرجة الثانية:

$$ax^2+bx+c=0$$

١١- اكتب برنامجاً بلغة Java يقوم بحساب وطباعة قيمة الدالة الرياضية المعرفة كما يلي:

$$f = \begin{cases} x^2 + 3 & \text{if } 5 \leq x \leq 10 \\ x^3 + \frac{3-x}{2} & \text{if } 10 < x \leq 100 \\ \frac{x-1}{25} & \text{otherwise} \end{cases}$$

١٢- اكتب برنامجاً بلغة Java يقوم بإدخال عدد صحيح a ويطبع "zero" إذا كان $a=0$ أو يطبع "one" إذا كان $a=1$ أو يطبع "Two" إذا كان $a=2$ أو يطبع "No thing" في باقي الحالات.



الفصل الثالث

التوابيع

Methods





التوابع (Methods)

٣-١ - مقدمة

تتجلى إحدى الفوائد الأساسية من استخدام التوابع عند كتابة البرامج بتقسيم العمليات الحسابية الطويلة والمعقدة إلى مجموعات من العمليات الصغيرة والبسيطة نسبياً، وينتج عن ذلك إمكانية إعادة استخدام هذه التوابع في العديد من البرامج بدلاً من إعادة كتابة العمليات نفسها في كل مرة، وتسمح التوابع، عند كتابتها بالطريقة المناسبة، بإخفاء التفاصيل الصغيرة للعمليات ما يسهل على المبرمج توضيح البنية العامة لبرامجه ويخفف من صعوبة تعديل هذه البرامج، وقد برهنت التجربة على أن أفضل طريقة لتطوير وصيانة برنامج كبير، تتمثل في عملية تجزئته إلى قطع وأجزاء أصغر يمكن التحكم بها بسهولة أكبر من البرنامج الأصلي. وتوجد في لغة Java مكتبة ضخمة من التوابع تقوم بتنفيذ العمليات الرياضية، والتعامل مع السلاسل والمحارف، والإدخال والإخراج، واكتشاف الأخطاء والعديد من العمليات الأخرى المفيدة ما يسهل مهمة المبرمج الذي يجد في هذه التوابع معيناً كبيراً له في عملية البرمجة. ويمكن للمبرمج كتابة توابع تقوم بأداء عمليات يحتاج لها المبرمج في برامجه .

٣-٢ - فوائد استخدام التوابع

من أهم فوائد التوابع في البرمجة ، هي :

١. تساعد التوابع المخزنة في ذاكرة الحاسوب على اختصار البرنامج إذ يكفي باستدعائها باسمها فقط لتقوم بالعمل المطلوب .
٢. تساعد البرامج المخزنة في ذاكرة الحاسوب أو التي يكتبها المستخدم على تلافي عمليات التكرار في خطوات البرنامج التي تتطلب عملاً مشابهاً لعمل تلك التوابع .
٣. تساعد التوابع الجاهزة في تسهيل عملية البرمجة.
٤. يوفر استعمال التوابع من المساحات المستخدمة في الذاكرة.
٥. كتابة برنامج في لغة Java بشكل واضح المعالم يجعل البرنامج واضحاً لكل من المبرمج والقارئ على حد سواء .
٦. التوابع تمكن المبرمج من تقسيم البرنامج إلى وحدات modules، وكل تابع في البرنامج يمثل وحدة قائمة بذاتها، ولذا نجد أن المتغيرات المعرفة في التابع تكون متحولات محلية (Local) ونعني بذلك أن المتحولات تكون معروفة فقط داخل التابع.
٧. أغلب التوابع تمتلك لائحة من الوسائط (Parameters) والتي هي أيضاً متحولات محلية .

٣-٣ - نموذج التابع

Method Prototype

التابع Method، هو مجموعة من التعليمات التي تنفذ عندما يتم استدعاه من مكان ما في البرنامج ، وفي الواقع، والتابع هو برنامج جزئي يمكن أن ينفذ على معطيات معينة وبعد ذلك يعيد قيمة إلى التابع المستدعي.

وعندما يولد المترجم تعليمات لاستدعاء تابع ما فإنه يحتاج إلى معرفة اسم التابع وعدد وسطائه وأنواعها ونوع قيمة الإعادة ، لذا علينا كتابة نموذج أو تصريح

للتابع ضمن الصف الذي يحتوي على التابع main (قبل التابع main أو بعده)

وعندما نصرح عن تابع ما فإننا نقوم فعلياً بتعريف مهمة هذا التابع و يبلغ المترجم عن اسم التابع وعدد وسطائه وأنواعها ونوع القيمة المعادة بواسطة التابع. فمثلاً في تصريح التابع التالي:

```
public static int sum( int x )
{
    جسم التابع
}
```

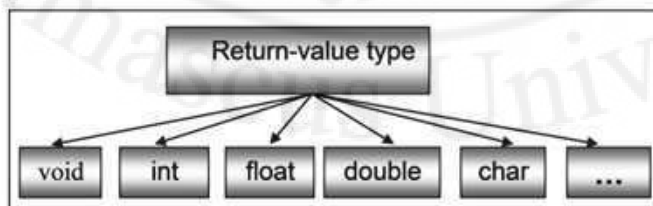
النوع **int** بين القوسين يخبر المترجم بأن الوسيط الذي سيتم تمريره إلى التابع سيكون من النوع **int** و **int** التي تسبق اسم التابع تشير إلى نوع القيمة المعادة بواسطة التابع. ويمكن أن تكون من نمط **double** أو **char** كما يلي :

```
public static double sum1(double x ) { ... }
public static char sum2( char x ) { ... }
```

أو لا يعيد قيمة وهو من نمط **void**

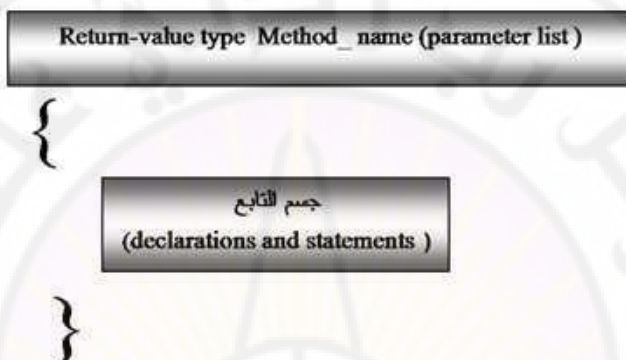
```
public static void sum( int x , double y ) { ... }
```

ونلاحظ أن كل تابع يملك الاسم الخاص به الذي يوضع من قبل المبرمج، وغالباً يكون اسم التابع قريباً من محتواه، وفيما يتعلق بنوع القيمة المعادة ، تحددها طبيعة المسألة المدروسة، فقد تكون من أي نوع من أنواع المعطيات المعروفة في لغة Java ، كما هو موضح الشكل (٣-١) .



الشكل (٣-١) - يبين نوع القيمة المعادة

كل تابع يُستخدم في برنامج بلغة Java يجب أن يكون له تعريف داخل البرنامج (عدا التوابع الرياضية والتوابع الجاهزة الأخرى) .
لا بد من الإشارة إلى أن تعريف التابع يُعطي علماً للمترجم كيف يعمل التابع (مهمة التابع)، ويبين الشكل (٣-٢) الشكل العام لتعريف التابع .



الشكل (٣-٢) الشكل العام لتعريف التابع

حيث:

Return-value type : نوع القيمة المعادة بواسطة التابع والذي يمكن أن يكون أي نوع من أنواع معطيات Java ، وإذا كان التابع لا يرجع أي قيمة يكون نوع إعادته void.

Method_name : اسم التابع والذي يتبع في تسميته قواعد تسمية المعرفات (identifiers).

parameter list : هي لائحة الوسطاء الممرة إلى التابع والتي يمكن أن تكون خالية (void) أو تحتوى على وسيط واحد أو عدة وسطاء تفصل بينها فاصلة ويجب ذكر كل وسيط على حدة.

declarations and statements : تمثل جسم التابع والذي يطلق عليه في بعض الأحيان كتلة (block) ، ويمكن أن تحتوى الكتلة (block)

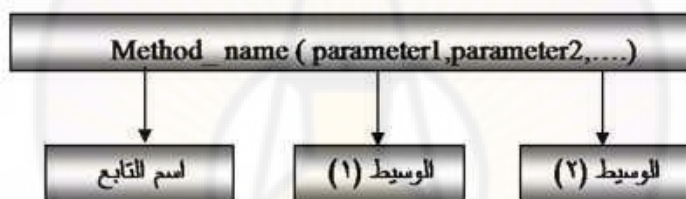
على تصريح عن المتحولات ولكن تحت أي ظرف لا يمكن أن يتم تعريف تابع داخل جسم تابع آخر .

السطر الأول في تعريف التابع يدعى المصريح declarator والذي يحدد اسم التابع ونوع المعطيات الذي يعيدها التابع وأسماء وسطائه وأنواعها .

٣-٤ - استدعاء التتابع

Calling Functions

كل تابع يستخدم في برنامج بلغة Java يجب أن يتم استدعاؤه من التابع الرئيسي main أو من أي تابع آخر، ويبين الشكل (٣-٣) الشكل العام للاستدعاء.



الشكل (٣-٣) - يبين الشكل العام للاستدعاء

يؤدي استدعاء التابع إلى انتقال التنفيذ إلى بداية التابع، ويمكن تمرير بعض الوسطاء إلى التابع عند استدعائه وبعد الانتهاء من تنفيذ التابع يعود التنفيذ للعبارة التي تلي استدعاء التابع في البرنامج الأصلي ، وبإمكان التابع أن يعيد قيمة إلى العبارة التي قامت باستدعائه، وإذا كان التابع لا يعيد شيئاً يجب استعمال الكلمة الأساسية void كنوع إعادة له للإشارة إلى ذلك . يوجد ثلاث طرق يمكن بها إرجاع التحكم إلى النقطة التي تم فيها استدعاء التابع ، وهي :

١. إذا كان التابع لا يرجع قيمة ، يرجع التحكم تلقائياً عند الوصول إلى نهاية التابع .

٢. باستخدام العبارة return ;

٣. إذا كان التابع يرجع قيمة فالعبرة : `return expression` ;
بإرجاع قيمة التعبير `expression` إلى النقطة التي استدعته .

مثلاً : لنصرح عن التابع `sum` يعيد قيمة من النوع `int` ويحتوي على وسيطين من النوع `int` ، ومن أجل استدعاء هذا التابع يجب أن يحتوي برنامج Java الذي يتضمن هذا التابع، على الاستدعاء التالي للتابع:

`sum ( a , b );`

حيث إن:

`sum` - اسم التابع.

`a` - الوسيط الأول (يجب أن يُعطى قيمة محددة أو يتم ادخله من قبل المستخدم).

`b` - الوسيط الثاني (يجب أن يُعطى قيمة محددة أو يتم ادخله من قبل المستخدم).

من غير الضروري أن تكون أسماء الوسطاء في التصريح هي نفسها المستعملة في تعريف التابع ، في الواقع، المترجم يتجاهلها لكنها تكون مفيدة أحياناً للذين يقرؤون البرنامج . ويمكن تعريف التابع `sum` كما يلي :

```
public static int sum ( int x , int y )
```

```
{
```

```
    int s;
```

```
    s=a+b;
```

```
    return s;
```

```
}
```

مثال (1-3)

لنأخذ برنامجاً يستخدم تابعاً يدعى square لحساب مربعات الأعداد من 1 إلى 10.

```
public class SquareNumber {  
    public static void main(String args[ ]) {  
        for (int x = 1; x <= 10; x++){  
            System.out.print( square (x) + " " );  
        }  
        System.out.println( ) ;  
        //End for  
    } //End main  
    //now Method definition  
    public static int square ( int y) {  
        int z;  
        Z= y * y ;  
        return z ;  
    } //end Method  
} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

1 4 9 16 25 36 49 64 81 100

Process completed

كل المتحولات المصرح عنها داخل التابع تُدعى متحولات محلية Local Variables , وتكون معرفة فقط في التابع الحاوي عليها ، بينما

المتحولات المصرح عنها في التابع الرئيسي `main()` تُدعى متحولات عامة `Global Variables` وتكون معرفة فقط في التابع الرئيسي. التابع الرئيسي `main()` يمكن أن يستدعي توابع أخرى، وهذه التوابع بدورها يمكن أن تستدعي نفسها أو توابع أخرى. نعيد كتابة البرنامج السابق باستخدام تابع لا يعيد قيمة .

مثال (2-3)

البرنامج:

```
public class SquareNumber {
public static void main(String args[ ]) {
for (int x = 1; x <= 10; x++){
    square(x) ;
}
    System.out.println( ) ;
//End for
} //End main
//now method definition
public static void square ( int y) {
    int z;
    Z= y * y ;
    System.out.print ( z+" " ) ;
} //end square
} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

1 4 9 16 25 36 49 64 81 100

Process completed

مثال (3-3)

البرنامج التالي يستخدم تابع يدعى maximum والذي يرجع العدد الأكبر بين ثلاثة أعداد صحيحة. يتم تمرير الأعداد كوسائط للتابع الذي يحدد العدد الأكبر بينهم ويرجعه للتابع main باستخدام العبارة return ويتم تعيين القيمة التي تمت إعادتها إلى المتغير largest الذي تتم طباعته.

البرنامج :

```
import java.util.Scanner;
public class Maximum {
    public static void main(String args[ ] ) {
        Scanner input = new Scanner( System.in );
        double a, b, c;
        System.out.println("Enter three integers: " );
        a = input.nextDouble( );
        b = input.nextDouble( );
        c = input.nextDouble( );
        System.out.println(" maximum " + maximum (a, b, c) );
    } //End main
    public static double maximum (double x, double y, double z) {
        double max = x;
        if (y > x)
            max = y;
        if (z > max)
            max = z;
        //Continued
        return max;
    } //End max
} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Enter three integers : 22 85 17

Maximum is: 85

Process completed

نعيد كتابة البرنامج السابق باستخدام الواجهات

```
import javax.swing.JOptionPane;
public class Maximum {
public static void main(String args[ ] ) {
double a, b, c;
// Display Maximum to Java!
    String javaString = JOptionPane.showInputDialog(null,
        "Enter a Number ", "My function Example",
        JOptionPane.QUESTION_MESSAGE);
    a = Double.parseDouble(javaString) ;
    //-----
    javaString = JOptionPane.showInputDialog(null,
        "Enter a Number ", "My function Example",
        JOptionPane.QUESTION_MESSAGE);
    b = Double.parseDouble(javaString) ;
    //-----
    javaString = JOptionPane.showInputDialog(null,
        "Enter a Number ", "My function Example",
        JOptionPane.QUESTION_MESSAGE);
    c = Double.parseDouble(javaString) ;
    JOptionPane.showMessageDialog(null,
        " The maximum is " + maximum (a, b, c),
        "My maximum Example",
        JOptionPane.QUESTION_MESSAGE);
} //End main
```

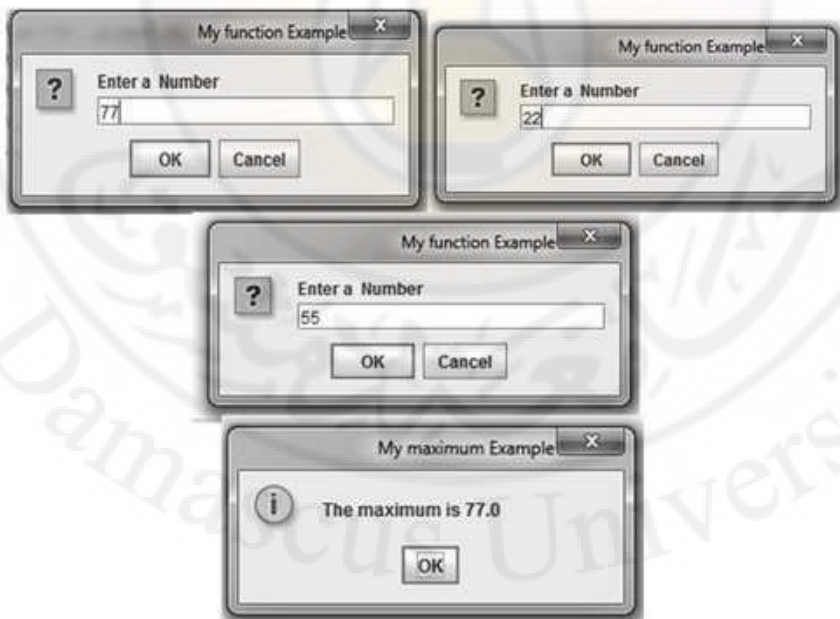


```

public static double maximum (double x, double y,double z) {
    double max = x;
    if (y > x)
        max = y;
    if (z > max)
        max = z;
    //Continued
    return max;
} //End max
} //End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٣ - ٥ - مكتبة التوابع الرياضية

Math Library Methods

تحتوي مكتبة التوابع الرياضية على العديد من التوابع التي تستخدم في تنفيذ العمليات الرياضية الحسابية، فمثلاً المبرمج الذي يرغب في حساب وطباعة الجذر التربيعي للعدد 900 قد يكتب العبارة التالية :

```
number = Math.sqrt ( 900 );  
System.out.println(number) ;
```

عند تنفيذ هذه العبارة يتم استدعاء التابع المكتبي (sqrt) لحساب الجذر التربيعي للعدد بين القوسين (900)، ويسمى العدد بين القوسين وسيط التابع argument وعليه فالعبارة السابقة تقوم بطباعة العدد 30 . ويأخذ التابع (sqrt) وسيط من النوع double وتكون النتيجة قيمة من نفس النوع وينطبق هذا على جميع التوابع الرياضية. ويوضح الجدول (٣-١) أهم التوابع الرياضية التي يمكن استخدامها من خلال مكتبة لغة Java ، ويتضمن الجدول أيضاً على وصف التوابع وأمثلة عليها . وعند استعمال التوابع الرياضية في أي برنامج بلغة Java يجب تضمين الصف Math والذي يحتوي على هذه التوابع .

مثال (3-4)

البرنامج التالي يستخدم التوابع الرياضية ضمن الصف Math ، ويقوم بحساب :
القيمة المطلقة - الجذر التربيعي - الرفع إلى قوة - التوابع المثلثية - التابع الأسّي واللغاريتمي .

البرنامج:

```
public class math1 {  
    public static void main(String args[ ] ) {  
        System.out.println(" The pi number = "+Math.PI);
```

```

System.out.println(" Absolute of -15 is: "+Math.abs(-15));
System.out.println(" Sin(90) = "+Math.sin((Math.PI)/2));
System.out.println(" Cos(90) = "+Math.cos((Math.PI)/2));
System.out.println(" Smallest number = "+Math.ceil(9.3));
System.out.println("Rounding value of 10.5 is "+Math.round(10.5));
System.out.println(" The Root Square of 36 = "+Math.sqrt(36));
System.out.println(" The e number = "+Math.exp(9));
System.out.println(" Convert to Radian =" +Math.toRadians(180));
System.out.println(" 2 Raised to the power of 4 "+Math.pow(2,4));
System.out.println(" log(1000) = "+Math.log10(1000));
} //End main
} //End Class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

```

The pi number = 3.141592653589793
Absolute of -15 is: = 15
Sin(90) = 1.0
Cos(90) = 6.123233995736766E-17
Smallest number( 9.3) = 10.0
Rounding value of 10.5 is = 9
The Root Square of 36 = 6.0
The e number exp(9) = 8103.083927575384
Convert to Radian((180) = 3.141592653589793
2 Raised to the power of 4 = 16.0
log(1000) = 3.0

```

Process completed.

Methods	Description	Example
sqrt(x)	الجذر التربيعي لـ x	sqrt(9.0) is 3
pow(a,b)	الرفع إلى قوة	a^b pow(10,5)
exp(x)	exp(1.0) is 2.718282	e^x
abs(x)	القيمة المطلقة لـ x	if $x > 0$ fabs(x) = x = 0 fabs(x) = 0 < 0 fabs(x) = -x
ceil(x)	تقريب x لأصغر عدد صحيح أكبر من x	ceil(9.2) is 10.0 ceil(-9.8) is - 9.0
floor(x)	تقريب x لأكبر عدد صحيح أصغر من x	floor(9.2) is 9 floor(-9.8) is -10.0
log(x)	اللوغاريتم الطبيعي لـ x	log (2.718282) is 1.0
log10(x)	اللوغاريتم العشري لـ x	log10(1.0) is 0.0
sin(x)	تابع الجيب لـ x	sin(0.0) is 0.0
cos(x)	تابع التمام لـ x	cos(0.0) is 1.0
tan (x)	تابع الظل لـ x	tan(0.0) is 0.0
max(x,y)	إيجاد أكبر قيمة بين العددين x و y	max(5,6)→ 6
min(x,y)	إيجاد أصغر قيمة بين العددين x و y	min(7,10)→ 7

الجدول (٣-١) - التوابيع الرياضية لمكتبة لغة Java

٣ - ٦ - التتابع بدون وسطاء

Functions with Empty Parameter Lists

في لغة Java يُكتب التابع الذي لا يمتلك وسطاء إما بكتابة void بين القوسين الذين يتبعان اسم التابع أو تركهما فارغين ، فمثلاً التصريح عن التابع :

```
public static void print ( ) { .... }
```

يشير إلى أن التابع print لا يأخذ أي وسيط وهو لا يرجع قيمة ، ويبين المثال (3-5) والمثال التالي يبين الطريقة التي تكتب بهما التتابع التي لا تأخذ وسطاء . حيث التابع الأول يقوم بحساب وطباعة جداء عددين أما التابع الثاني يقوم بحساب وطباعة مجموع عددين .

مثال (3-5)

```
public class math1 {  
    public static void f1 ( ) {  
        double x,y ;  
        x=55; y=44;  
        System.out.println( "x*y="+ x*y) ;  
    }  
    public static void f2 ( ) {  
        int x,y ;  
        x=5;y=7;  
        System.out.println("x+y="+ (x+y)) ;  
    }  
    public static void main(String args[ ] ) {  
        f1 ( ) ;  
        f2 ( ) ;  
    } //End main  
} //End Class
```


وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

$x*y = 2420.0$

$x+y = 12$

Process completed.

٣-٧- التحميل الزائد للتوابع

Overloading Functions

التحميل الزائد للتوابع يعنى استخدام الاسم نفسه لعدة توابع، لكن كل تابع يجب أن يكون له تعريف مستقل، وعند استدعاء التابع، يبحث المترجم عن نوع وسطاء التابع وعددها لمعرفة التابع المقصود، ولكي يميز المترجم بين تابع وآخر يحمل الاسم نفسه، يقوم بعملية تعرف بتشويه الأسماء (names mangling)، وتتألف هذه العملية من إنشاء اسم جديد خاص بالمترجم عن طريق دمج اسم التابع مع أنواع وسطائه.

مثال (3-6)

البرنامج التالي يقوم بتحميل التابع square بشكل زائد وذلك من أجل حساب وطباعة الجذر التربيعي لعدد من النوعين int و double على الترتيب .

البرنامج:

```
public class Overloading {  
    public static int square(int x)  
    {  
        return x*x ;  
    }  
    public static double square(double y)
```

```

{
    return y*y;
}
public static void main(String args[ ] ) {
    System.out.println( "The square of integer 6 is"+ square(6) );
    System.out.println( "The square of double 8.5 is "+ square(8.5) );
} //End main
} //End Class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

The square of integer 6 is 36
 The square of double 8.5 is 72.25
 Process completed

مثال (7-3)

البرنامج التالي يقوم بتحميل التابع abs بشكل زائد وذلك من أجل حساب وطباعة القيمة المطلقة لثلاثة أعداد من الأنواع int و double و long على الترتيب.

البرنامج:

```

// abs is overloaded three ways
public class absOverloading {
    public static void main(String args[ ] ) {
        System.out.println( abs (-10) );
        System.out.println( abs(-11.0) );
        System.out.println( abs( -9L ) );
    } //End main
    public static int abs ( int i ) {
        System.out.println("using integer abs( ) ");
    }
}

```

```

    return i < 0 ? -i : i ;
} //End abs int
//Continued -----
public static double abs ( double d )
{
    System.out.println("using double abs( ) ");
    return d < 0.0 ? -d : d ;
} //End abs double
//Continued -----
public static long abs( long l )
{
    System.out.println("using long abs( ) ");
    return l < 0.0 ? -l : l ;
} //End abs Long
} //End Class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

```

RESULT
using integer abs( )
10
using double abs( )
11
using long abs( )
9
Process completed

```

٣-٨ - التوابع العودية

Recursive Functions

من الميزات التي ظهرت في اللغات الحديثة تابع العودية (التكرار والتوليد) ، وتتلخص هذه الميزة بإمكانية أن يستدعي التابع نفسه ، وأن الفائدة التي يجنيها

المبرمج من استدعاء التابع نفسه باستعمال تابع العودية هي أن هذا التابع يستخدم لحل بعض المسائل المرتبة ترتيباً متكرراً أصلاً كمسائل ترتيب الأرقام تصاعدياً أو تنازلياً في برنامج guicksort , كما أن هذا النوع من التوابع يفيد في برمجة بعض مسائل الذكاء الصناعي (Artificial Intelligent) . وعند استخدام تابع العودية نحتاج أن نستعمل التعليمة الشرطية if لإجبار التابع على العودة إلى مكان انطلاقه بعد استدعاء نفسه في البرنامج، وإن لم نستعمل التعليمة if فإن التابع لا يعود .

مثال (3-8)

اكتب برنامجاً بلغة Java , يقوم بما يلي :

- حساب n عاملي (!) باستخدام التابع العودي .

- إيجاد مجموع n عدد صحيح مرتب باستخدام مفهوم العودية .

البرنامج :

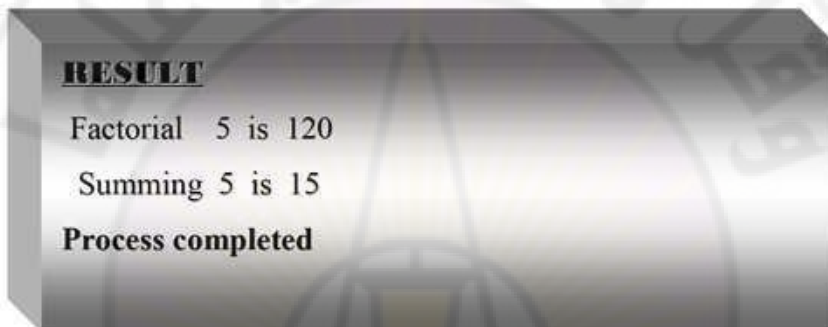
```
public class Recursion {
    public static void main(String args[ ] ) {
        int n = 5 ;
        System.out.println( "Factorial "+n+" is "+fact(n) );
        System.out.println("Summing "+n+" is "+sum(n) );
    } //End main
    public static int fact( int n ) {
        if ( n == 0 || n == 1 )
            return 1;
        else
            return n * fact(n-1);
    } //End fact
    public static int sum( int n ) {
        if ( n == 1 )
```

```

return 1;
else
return n + sum(n-1);
} //End sum
} //End Class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



مثال (3-9)

البرنامج التالي يقوم بإنشاء تابع لحساب العامل Factorial من أجل قيمة معطاة n , ثم كتابة برنامج بلغة java يختبر هذا التابع .

- نستخدم تابع العودية من أجل حساب العامل
- نستخدم واجهة (نافذة) من أجل إدخال قيمة العدد n .
- نستخدم واجهة من أجل طباعة النتيجة .

البرنامج :

```

// Compute factorial of an integer
import javax.swing.JOptionPane;
public class ComputeFactorial {
    /** Main method */

```

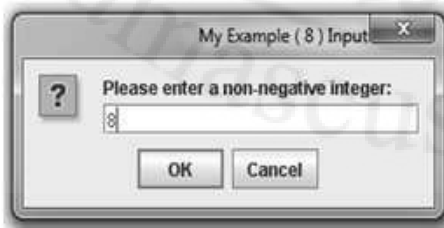


```

public static void main(String[ ] args) {
    // Prompt the user to enter an integer
    String intString = JOptionPane.showInputDialog (
        null,"Please enter a non-negative integer:",
        "My Example ( 8 ) Input", JOptionPane.QUESTION_MESSAGE
    );
    // Convert string into integer
    int n = Integer.parseInt(intString);
    // Display factorial
    JOptionPane.showMessageDialog(null,
        "Factorial of " + n + " is " + factorial(n)+
        "\nحساب العامل باستخدام تابع العودية ", "My Example ( 9 ) Output",
        JOptionPane.INFORMATION_MESSAGE );
} //End main
/** Return the Fibonacci number for a specified index */
public static long factorial(int n) {
    if (n == 0) // Stopping condition
        return 1;
    else
        return n * factorial(n - 1); // Call factorial recursively
} //End factorial
} //End class

```

وعند تنفيذ البرنامج سنحصل على نافذة الدخل والخرج التاليتين:



٣- ٩- توليد الأعداد العشوائية

يوجد في مكتبة Java تابع عشوائي random ، ويولد أعداداً صحيحة من 0 إلى RAND_MAX ، وهو عبارة عن ثابت مُعرف في الصف Math، إذا كان التابع random يقوم فعلياً بتوليد أعداد صحيحة عشوائية فهذا يعني أن الأعداد ما بين 0 و RAND_MAX لها جميعاً نسبة الاحتمال نفسها ، ويتم اختيارها في كل مرة نستدعي هذا التابع ، لكن يمكن لمجال تحول قيم التابع random أن يكون مختلفاً عن ذلك الذي نحتاجه من أجل تطبيق معين . فعلى سبيل المثال في برنامج يحاكي عملية رمي قطعة نقود فنحن نحتاج إلى قيمتين فقط هما : 0 من أجل الطرة head ، و 1 من أجل النقش tail ، أما بالنسبة لبرنامج يحاكي عملية رمي حجر النرد ذي ستة وجوه ، فإننا نحتاج إلى ست قيم ضمن المجال من 1 حتى 6 .

مثال (3-10)

اكتب برنامجاً بلغة الـ Java يقوم بمحاكاة عملية رمي حجر النرد ذي الأوجه الستة ، 20 مرة وبحيث يطبع نتيجة كل عملية من عمليات الرمي.

البرنامج:

```
// program uses class Random
```

```
import java.util.Random;
```

```
public class RandomIntegers {
```

```
public static void main( String args[ ] {
```

```
// random number generator
```

```
Random randomNumbers = new Random( );
```

```

int face; // stores each random integer generated
           // loop 20 times
for ( int counter = 1; counter <= 20; counter++)
{
    // pick random integer from 1 to 6
    face = 1 + randomNumbers.nextInt( 6 );
    System.out.printf( "%d ", face ); // display generated value
    // if counter is divisible by 5, start a new line of output
    if ( counter % 5 == 0 )
        System.out.println( );
    } // end for
} // End main
} // End class RandomIntegers

```

وعند تنفيذ البرنامج (تنفيذ أول) سنحصل على الخرج التالي:

RESULT

6	6	5	5	6
5	1	1	5	3
6	6	4	4	2
6	2	3	4	1

Process completed





مسائل عامة

١- اكتب برنامجاً بلغة Java يقوم بحساب العامل (!) للأعداد من 1 إلى 5.

=====

٢- اكتب برنامجاً بلغة Java يحتوي على تابعين : التابع الأول اسمه Factorial مهمته حساب العامل (!) لأي عدد صحيح موجب ، أما الثاني اسمه SumFactorial مهمته حساب المقدار التالي :

$$sum = n! + \frac{n!}{x!} - m!$$

التابع main يستدعي كلاً من التابعين وطباعة قيمة المقدار Sum (التابع SumFact يستدعي التابع Factorial) .

=====

٣- اكتب برنامجاً بلغة Java يقوم بما يلي : إدخال قيم المتحولات a,b,x وهي من النمط double ويحتوي البرنامج على تابعين :

❖ التابع الأول اسمه FunctionY مهمته حساب المقدار التالي :

$$y = x^2 + 2x + 10$$

❖ أما التابع الثاني اسمه FunctionZ مهمته حساب المقدار Z والمعرف

كما يلي :

$$Z = (y - a)^2 + (y + b)^2$$

❖ التابع main يستدعي كلاً من التابعين لطباعة قيمة Z (التابع
FunctionZ يستدعي التابع FunctionY) .

=====

٤- اكتب برنامجاً بلغة Java يقوم بإدخال أي عدد حقيقي x من قبل المستخدم
. ويحتوي على ثلاثة توابع لحساب قيمة التابع الرياضي التالي :

$$yy = \begin{cases} 5x^5 + 2x + 5.....if ..1 \leq x < 5 \\ \frac{x^6}{2x+1}if ...5 \leq x < 100 \\ 3x + 5.....otherwise \end{cases}$$

. يستدعي التابع main كلاً من التوابع الثلاثة لطباعة قيمة yy .

=====

٥- اكتب برنامجاً بلغة Java يقوم بما يلي:

- ❖ إدخال أي عدد صحيح x.
- ❖ يستخدم تابعاً اسمه fun_prime لإيجاد قواسم العدد المدخل x .
- ❖ يختبر البرنامج إذا كان العدد المدخل x أولياً أو غير أولياً وذلك بناءً على
خرج التابع fun_prime, وطباعة النتيجة في التابع main .

=====

٦- اكتب برنامجاً بلغة Java يقوم بما يلي:

- ❖ إدخال أي عددين صحيحين p و k .
- ❖ يستخدم تابعين :
- ❖ التابع الأول اسمه Factorial لحساب قيمة p! (العاملي)

✚ التابع الثاني اسمه F_permutation لحساب قيمة

المقدار الرياضي التالي:

$$permutation = \frac{p!}{(p-k)!}$$

❖ التابع main يستدعي كلاً من التابعين لطباعة قيمة Permutation

وطباعة النتيجة في التابع main.

٧- أنشئ تابعاً بلغة Java اسمه sumpow يقوم بحساب مجموع السلسلة

$z = \sum_{i=1}^{i=10} i^r$. حيث $r=5$ ثم اكتب برنامجاً يقوم باستدعاء هذا التابع وطباعة النتيجة .

=====

٨- أنشئ تابعاً بلغة Java اسمه Cube يقوم بحساب حجم مكعب طول ضلعه

x . ثم اكتب برنامجاً يقوم باستدعاء هذا التابع وطباعة النتيجة .

=====



الفصل الرابع

المصفوفات

Arrays





المصفوفات

٤-١- مقدمة

تنقسم المعطيات إلى معطيات محرفية (char) و صحيحة (int) و حقيقية (float) ، وتسمى هذه الأنواع بالأنواع الرئيسية للمعطيات، ولكن هناك أنواع أخرى من المعطيات تسمى بالأنواع المشتقة Derived Data Types ، منها المصفوفات Arrays ، ومن أهم فوائد استخدام المصفوفات :

- يمكن استخدام المصفوفات من أجل الترتيب التصاعدي والتنازلي للعناصر والقيم المختلفة (القيم العددية).
- يمكن استخدام المصفوفات في ترتيب الأسماء وفق الترتيب الأبجدي في النصوص الرمزية.
- يمكن استخدام المصفوفات في العمليات الحسابية كضرب المصفوفات. و في إيجاد مقلوب مصفوفة.

٤-٢- تعريف المصفوفة

Array Definition

المصفوفة هي بنية تُستخدم للاحتفاظ بمجموعة من القيم من نفس نمط المعطيات والمصفوفة هي نوع من أنواع بنى المعطيات، لها عدد محدود ومرتب من العناصر التي تكون جميعها من نفس النوع type، فمثلاً يمكن أن تكون جميعها صحيحة int أو حقيقية float أو محرفية char ، ولكن لا يمكن الجمع بين نوعين مختلفين في المصفوفة نفسها، وتنقسم المصفوفات إلى نوعين : مصفوفات أحادية البعد ومصفوفات متعددة الأبعاد (على سبيل المثال مصفوفات ثنائية الأبعاد).

٤ - ٣ - المصفوفات أحادية البعد

One-Dimensional Arrays

المصفوفة التي تتكون من سطر واحد أو عمود واحد ، تسمى مصفوفة أحادية البعد، ويمكن أن تكون المصفوفة أحادية البعد محرفية char أو حقيقية float أو صحيحة int ، أما دليل المصفوفة (index) ، فيجب أن يكون من النوع الصحيح int ، وكل مصفوفة تُستخدم في البرنامج ، لا بد من التصريح عنها، قبل استخدامها . ويبين الشكل (٤-١) مصفوفة a تحتوى على 10 عناصر من النوع int، ويمكن الوصول إلي أي من هذه العناصر بذكر اسم المصفوفة متبوعاً برقم موقع العنصر في المصفوفة محاطاً بالأقواس [] ، أي a[10] ، اسمها a عدد عناصره 10 .



الشكل (٤-١) - يبين مصفوفة أحادية البعد a[10] من النوع int

حيث يرمز لرقم العنصر في المصفوفة بدليل العنصر index ، ودليل العنصر الأول في المصفوفة هو 0 ولهذا يشار إلى العنصر الأول في المصفوفة a بـ a[0] والثاني a[1] والسابع a[6] وعموماً يحمل العنصر i في المصفوفة a الدليل a[i-1] ، وتتبع تسمية المصفوفات قواعد تسمية المتحولات ذاتها.

٤-٣-١ - التصريح عن المصفوفة أحادية البعد

تحتل المصفوفات حجماً في الذاكرة لذا يجب على المبرمج تحديد نوع عناصر المصفوفة وعددها حتى يتسنى للمعرف تخصيص الحجم اللازم من الذاكرة لحفظ المصفوفة، وحتى تخبر المترجم بأن يخصص حجماً محدداً لنوع عناصر المصفوفة، ويتضمن التصريح عن مصفوفة أحادية البعد مرحلتين :

١- التصريح عن متحول يشير إلى مصفوفة

إن التصريح عن مصفوفة يشبه التصريح عن أي متحول آخر ، فالتصريح يحوي قسمين :

- الأول هو نوع المصفوفة

- والثاني هو اسم المصفوفة .

والشكل العام للتصريح عن المصفوفة أحادية البعد هو التالي:

`Type [ ] anArray ;`

حيث `Type` هو نمط المعطيات ، وأما القوسين `[ ]` فيشيران إلى أن هذا المتحول هو مصفوفة وليس قيمة واحدة .

مثال : `String [ ] anArray1 , int [ ] anArray2 , . . .`

إن التصريح عن متحول مصفوفة لن يحجز أي منطقة في الذاكرة حتى يتم فعلياً إسناد قيم إلى المصفوفة ، وإنما يتم فقط صنع مكان تخزين من أجل المرجع على المصفوفة ، وإذا لم يملك المتحول مرجعاً إلى المصفوفة فإن قيمة هذا المتحول هي `null` .

٢ - إنشاء المصفوفة

يتم إنشاء مصفوفة باستخدام المعامل new ، مع الصيغة التالية :

```
anArray = new dataType [ size ] ;
```

تقوم العبارة السابقة بمهمتين : الأولى إنشاء مصفوفة باستخدام new dataType [size] ، والثانية إسناد مرجع للمصفوفة المنشئة حديثاً إلى المتحول anArray .

مثلاً :

```
anArray1 = new int [20] ;
```

- هذه العبارة تقوم بإنشاء مصفوفة وتخصيص ذاكرة محددة لها تتسع لعشرين عدداً صحيحاً ، ثم تسند مرجعاً لهذه المصفوفة إلى المتحول anArray1 .

```
anArray2 = new double [20] ;
```

- إنشاء مصفوفة وتخصيص ذاكرة محددة تتسع لعشرين عدداً حقيقياً ، ثم تسند مرجعاً لهذه المصفوفة إلى المتحول anArray2 .

```
anArray3 = new String [20] ;
```

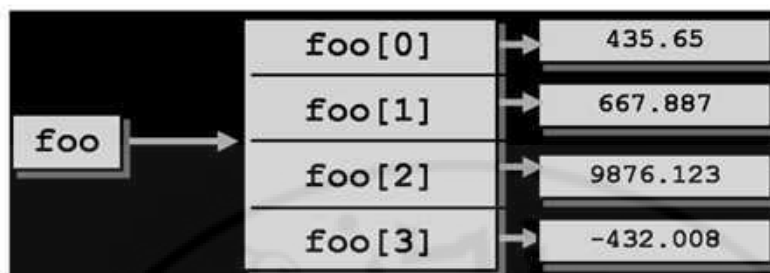
- إنشاء مصفوفة وتخصيص ذاكرة محددة تتسع لعشرين سلسلة محرفية ، ثم تسند مرجعاً لهذه المصفوفة إلى المتحول anArray3 .

مثلاً :

التصريح عن مصفوفة foo من نمط double تتكون من 4 عناصر (إنشاء مصفوفة) .

```
double foo[ ] ;
```

```
foo = new double [4] ;
```



من الممكن دمج عملية التصريح عن متحول المصفوفة وإنشاء المصفوفة وإسناد مرجع للمصفوفة إلى المتحول في عبارة واحدة كما يلي :

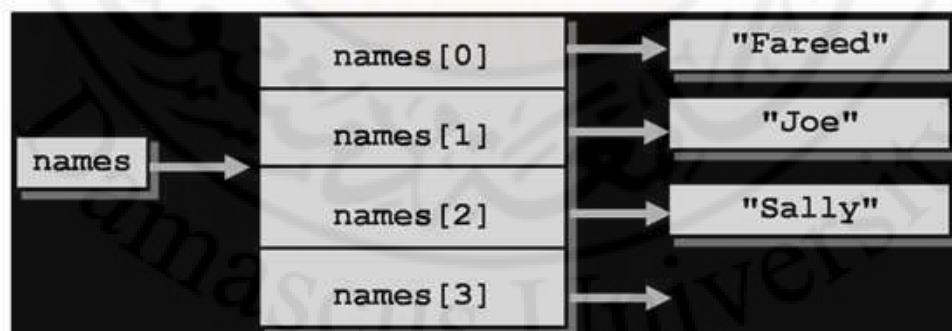
```
dataType [ ] anArray = new dataType [ size ] ;
```

```
dataType anArray[ ] = new dataType [ size ] ;
```

مثلاً :

التصريح عن مصفوفة names من نمط String تتكون من 4 عناصر (إنشاء مصفوفة) .

```
String [ ] names = new String [4];
```



٣- تهيئة ومعالجة المصفوفات

عند إنشاء المصفوفة يتم إسناد القيمة الافتراضية (0) من أجل أنماط المعطيات الأولية العددية والقيمة الافتراضية 'u0000' من أجل النمط char ، ويتم إسناد القيمة الافتراضية false إلى الأنماط المنطقية . للحصول على حجم المصفوفة (عدد عناصرها) نستخدم الصيغة التالية :

Arrayname.length

تعيد العبارة Arrayname.length حجم المصفوفة . ويتم الوصول إلى عناصر المصفوفة من خلال الدليل (index) حيث يتم فهرسة المصفوفة من القيمة 0 إلى القيمة Arrayname.length .

٤-٣-٢- إسناد قيم ابتدائية لعناصر المصفوفة أحادية البعد

يمكن إعطاء قيم ابتدائية لعناصر المصفوفة أحادية البعد باتباع التصريح عن المصفوفة بعلامة المساواة (=) تليها لائحة من القيم المطلوب إسنادها لعناصر المصفوفة عندها، ويتم الفصل بين القيم بفواصل، وتحيط هذه اللائحة الأقواس الحاصرة { } ، على سبيل المثال لنأخذ التصريحات التالية لثلاث مصفوفات من أنواع مختلفة من المعطيات ، وفي الوقت نفسه إعطاء قيماً ابتدائية لعناصر المصفوفة (إجراء عملية تمهيد للعناصر) عند التصريح عنها ، كما يلي:

```
1:int [ ] arry1 = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37};
```

```
2:Sring [ ] arry2 = { "Ahmad", "Omar", "Ali"};
```

```
3:char [ ] arry3 = { 'A', 'B', 'C', 'D', 'E'};
```

مثال (1-4)

يقوم البرنامج التالي بإعطاء قيم ابتدائية لعناصر المصفوفة array1 من النوع int والتي تمثل خمس درجات للطلاب في البرمجة (2) ، والمصفوفة الثانية array2 من النوع String وتمثل أسماء الطلاب ، أما المصفوفة الثالثة array3 من النوع char وتمثل الرمز المكافئ للدرجة .

البرنامج:

```
public class ArrayProg {  
    public static void main(String args[ ]) {  
        // initialize array  
        int [ ]arry1 = {80,66,88,90,95} ;  
        String [ ]arry2 = {"Ahmad", "Omar", "Younes","Waseem","Akram"} ;  
        char array3[ ] = { 'B', 'D', 'B', 'A', 'A'} ;  
        for (int i=0 ; i< 5; i++) // print array  
            System.out.println( i+1+" " +arry1[i]+" " +arry2[i]+" " +arry3[i])  
    } //End main  
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي

RESULT

```
1 80 Ahmad B  
2 66 Omar D  
3 88 Younes B  
4 90 Waseem A  
5 95 Akram A
```

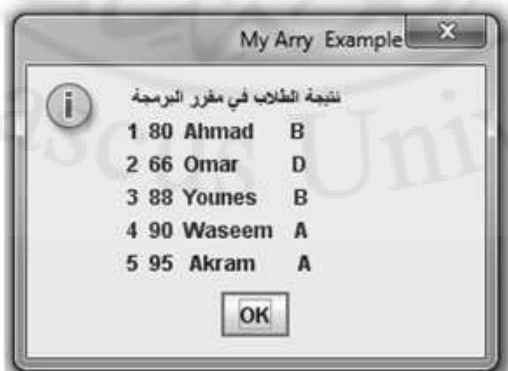
Process completed.

لنعيد كتابة البرنامج باستخدام الواجهات .

البرنامج :

```
import javax.swing.JOptionPane;
public class ArrayProg {
public static void main(String args[ ]) {
    // initialize array
    int [ ]arry1 = {80,66,88,90,95} ;
    String [ ]arry2 = {"Ahmad", "Omar ", "Younes ",
                      "Waseem ", " Akram  " } ;
    char arry3[ ] = { 'B', 'D', 'B', 'A', 'A' } ;
    String output="\n" ;
    for (int i=0 ; i< 5; i++){ // print array
        int j=i+1;
        output = output +j+" " +arry1[i]+" " +arry2[i]+" " +arry3[i]+" \n" ;
    } //end for
    // Display array !
    JOptionPane.showMessageDialog(null,
        "نتيجة الطلاب في مقرر البرمجة "+ output, "My Arry Example ",
        JOptionPane.INFORMATION_MESSAGE );
    } //End main
    } //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



فإذا كان المطلوب إسناد عناصر مصفوفة مهما كان حجمها بأصفار ،
يمكن التصريح عن المصفوفة بالشكل التالي :

```
int anyarray[ ]={0};
```

سيتم إسناد القيمة 0 للعنصر الأول، أما العناصر المتبقية يسند لها القيمة صفر أيضاً كوننا لم نحدد قيمة لها. إذا لم يتم تحديد عناصر المصفوفة من قبل المبرمج عند التصريح عنها، فإن المترجم يقوم بهذه المهمة من خلال عد القيم (العناصر)، الموجودة في الطرف الأيمن وحجز أماكن لعناصر المصفوفة وفق هذا العدد الناتج فعلى سبيل المثال :

```
int array[ ]={10,20,30};
```

هذا التصريح يعني: أن عدد عناصر المصفوفة [array] هو ثلاثة فقط .

٨-٤-٣ - إدخال وإخراج عناصر المصفوفة أحادية البعد

بالإضافة إلى إعطاء قيمة ابتدائية لعناصر المصفوفة عند التصريح عنها، يمكن استخدام أمر الإدخال cin ضمن حلقة التكرار ، ومن أجل طباعة عناصر المصفوفة يُستخدم أمر الإخراج وذلك أيضاً ضمن حلقة تكرار ، عادة قبل استخدام المصفوفة يمكن تصغيرها ، وذلك بإعطاء قيم ابتدائية تساوي الصفر ، ويوضح المثال (4-2) كيفية إدخال وإخراج عناصر المصفوفة احادية البعد.

مثال (4-2)

اكتب برنامجاً بلغة Java يقوم بما يلي :

- تصفير عناصر المصفوفة [5] myArray .
- إدخال 5 أعداد صحيحة وتخزينها في المصفوفة.
- طباعة عناصر المصفوفة المدخلة.

البرنامج:

```
import java.util.Scanner;
public class IOArray {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
int myArray [ ];
myArray = new int [5] ;
    for (int i=0; i< myArray.length ;i++) // initialize array
        myArray [i] = 0;
for (int i=0 ; i< myArray.length ; i++) //Enter new elements
myArray [i] = input.nextInt( );
    for (int i=0 ; i< myArray.length ; i++) // print array
System.out.println("myArray [" +i+"]="+ myArray [i]);
} //End main
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

```
Enter The value of myArray
55 67 88 23 95
myArray[0]=55
myArray[1]=67
myArray[2]=88
myArray[3]=23
The value of myArray[4]=95
Process completed
```

□ نلاحظ في العبارة:

```
final int size = 5 ;  
int [ ] myarray = new int [ size ]
```

استعملنا الكلمة المفتاحية `final` ، يتم استعمالها في تعريف المتحول الذي لا يمكن تغيير قيمته (الثابت) في البرنامج ولذلك يجب إعطاءه قيمة ابتدائية عند تعريفه (في البرنامج السابق تم تحديده بالقيمة 12)

مثال (3-4)

اكتب برنامجاً بلغة Java لحساب وطباعة المتوسط الحسابي لخمس أعداد يتم إدخالها من لوحة المفاتيح وتخزينها في مصفوفة أحادية البعد `myArray [size]`. حيث `size` هو متحول من نوع `final int size = 5`.

البرنامج:

```
import java.util.Scanner;  
public class AvrArray {  
    public static void main(String args[ ]) {  
        Scanner input = new Scanner( System.in );  
        final int size = 5 ;  
        double myArray [ ]= new double [size]  
        for (int i=0 ; i< myArray.length ; i++) //Enter new elements  
            myArray [i] = input.nextDouble( );  
        double sum=0;  
        for (int i=0 ; i< myArray.length ; i++)  
            sum=sum + myArray[i];  
        System.out.println("Average is " + sum/size);    // print sum  
    } //End main  
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

Enter the value of myarry

10.1 11.2 12.3 13.4 14.5

The average is 12.299999999999999

Process completed

مثال (4-4)

يقوم البرنامج بإدخال ستة أرقام صحيحة ثم إيجاد أكبر عدد بين الأعداد المدخلة ،
ثم إيجاد عدد مرات ظهور أكبر عدد .

- يجب التصريح عن مصفوفة numbers لتخزين الأعداد الستة الصحيحة .
- استخدم واجهة (نافذة) من أجل إدخال ستة أعداد صحيحة .
- استخدم واجهة لطباعة عناصر المصفوفة وأكبر عدد وعدد مرات ظهوره .

البرنامج

// Count the occurrences of the largest number

import javax.swing.JOptionPane;

public class MaxArray {

/** Main method */

public static void main(String[] args) {

final int Size = 6;

int [] numbers = **new int**[Size];

// Read all numbers

for (**int** i = 0; i < numbers.length; i++) {

String intString = JOptionPane.showInputDialog(null,

 " Enter Your numbers:\n أدخل الأعداد ",


```

    "My Example  Input", JOptionPane.QUESTION_MESSAGE );
    // Convert string into integer
    numbers[i] = Integer.parseInt(intString);
} //end for
// Find the largest
    int max = numbers[0];
    for ( int i = 1; i < numbers.length; i++) {
        if (max < numbers[i])
            max = numbers[i];
    }
    // Find the occurrence of the largest number
    int count = 0;
    for (int i = 0; i < numbers.length; i++) {
        if (numbers[i] == max) count++;
    }
    // Prepare the result
    String output = "The array is ";
    for (int i = 0; i < numbers.length; i++) {
        output += numbers[i] + " ";
    }
    output += "\nThe largest number is " + max;
    output += "\nThe occurrence count of the largest number "+
        "is " + count;
    // Display the result
    JOptionPane.showMessageDialog(null, output,
        "My Example( 9 ) Output",
        JOptionPane.INFORMATION_MESSAGE );
    System.exit(0);
} //End main
} //End class

```

وعند تنفيذ البرنامج سنحصل على نوافذ الدخل والخرج التالية:



٤ - ٤ - المصفوفات ثنائية البعد

Tow-Dimensional Array

يمكن للمصفوفات في لغة Java أن تكون متعددة الأبعاد ويمكن أن يكون كل بعد بحجم مختلف ، والاستخدام الشائع للمصفوفات متعددة الأبعاد هو تمثيل الجداول Tables ، والجداول (٤-١) يحتوي على معطيات مرتبة في صورة

صفوف وأعمدة، ولتمثيل الجدول نحتاج لبعدين البعد الأول يمثل الأسطر (الصفوف) والبعد الثاني يمثل الأعمدة. وبين الشكل التالي مصفوفة A تحتوى على ثلاثة صفوف وأربعة أعمدة، ويتم تمثيل أي عنصر في المصفوفة A على الصورة $A[i][j]$.

	Column 0	Column1	Column2	Column 3
Row 0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
Row 1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
Row 2	A[2][0]	A[2][1]	A[2][2]	A[2][3]

الجدول (٤-١) - يحتوي على معطيات مرتبة في صورة صفوف وأعمدة

حيث:

A - اسم المصفوفة.

i - رقم السطر الذي ينتمي إليه العنصر.

j - رقم العمود الذي ينتمي إليه العنصر.

نلاحظ أن كل العناصر الموجودة في السطر الأول مثلاً يكون الدليل الأول لها هو 0 وكل العناصر الموجودة في العمود الرابع يكون الدليل الثاني لها هو 3.

٤-٤-١- التصريح عن المصفوفات ثنائية البعد

إن المصفوفات تحتل حجماً في الذاكرة لذا يجب على المبرمج تحديد نوع عناصر المصفوفة وعددها حتى يتسنى للمعرف تخصيص الحجم اللازم من

الذاكرة لحفظ المصفوفة، والشكل العام للتصريح عن المصفوفة ثنائية البعد كما يلي:

Type Array_Name [][] = new Type [size1][size2];

حيث إن:

Type - نوع المعطيات .

Array_Name - اسم المصفوفة، والذي يتم وضعه من قبل المبرمج مع مراعاة لقواعد تسمية المتحولات، وحيث أن كل متحول في المصفوفة يدعى عنصراً وكل عنصر في المصفوفة يجب أن يكون من نوع المصفوفة.

size1 - عدد الأسطر في المصفوفة.

size2 - عدد الأعمدة في المصفوفة.

يتم التصريح عن مصفوفة **a** تحتوى على **x** صف و **y** عمود كما يلي :

int a [][] = new int [x][y] ;

عند استخدام مصفوفة أحادية البعد لا بد من استخدام حلقة تكرار واحدة (**for** أو **while** أو **do-while**) ، بينما عند استخدام مصفوفة ثنائية البعد لا بد من استخدام حلقتي تكرار أو ما يسمى بحلقات التكرار المتداخلة **Nested loops**.

٤-٢- إعطاء قيم ابتدائية لعناصر المصفوفة ثنائية البعد

يمكن التصريح عن المصفوفة ثنائية البعد وإعطائها قيماً ابتدائية (إجراء عملية تمهيد للعناصر)، كما يلي :

int b [][] = { { 1,2 } , { 3,4 } } ;

حيث:

b[1][1]=4 , b[1][0]=3 , b[0][1]=2 , b[0][0]=1

أيضاً نلاحظ في المصفوفة متعددة الأبعاد إذا تم إسناد قيم أولية لا يتوافق عددها مع حجم المصفوفة فإن المترجم سيملاً بقية العناصر أصفاراً. ويوضح البرنامج في المثال (4-5) كيفية إسناد قيم أولية للمصفوفات متعددة الأبعاد عند التصريح عنها.

مثال (4-5)

```
// initializing multidimensional arrays
public class ArrayProg {
    public static void main(String args[ ] ) {
        int a[ ][ ] = { {11, 31, 51}, {44, 74, 24}, {99, 16, 45}, {67, 89, 45} };
        for (int i= 0; i<4 ; i++){
            for (int j= 0; j<3 ; j++){
                System.out.print(" " + a[i][j] );
                System.out.println( ) ;
            }
        }
    } //End main
} //End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT

```
11 31 51
44 74 24
99 16 45
67 89 45
```

Process completed.

٤-٣- المصفوفات غير المنتظمة

كل صف في المصفوفة متعددة الأبعاد هو مصفوفة بحد ذاته ، وهكذا فإنه من الممكن للصفوف أن تملك أطوال مختلفة . تعرف المصفوفة من هذا النوع بالمصفوفة غير المنتظمة ، فيما يلي مثال عن إنشاء مصفوفة غير منتظمة :

```
int [ ][ ] array = { { 1,2,3,4,5}, { 2,3,4,5}, { 3,4,5}, { 4,5}, { 5} } ;
```

لذلك فإن طول `array[0].length` يساوي 5 ، وطول `array[1].length` يساوي 4 ، وطول `array[2].length` يساوي 3 ، وطول `array[3].length` يساوي 2 ، وطول `array[4].length` يساوي 1 . إذا كانت القيم التي ستخزن في المصفوفة غير المنتظمة غير معروفة فإنه يمكن صنع مصفوفة غير منتظمة باستخدام الصيغة التالية :

```
int [ ][ ] array = new int [5][ ] ;
array [0] = new int [5] ;
array [1] = new int [4] ;
array [2] = new int [3] ;
array [3] = new int [2] ;
array [4] = new int [1] ;
```

يمكننا الآن إسناد قيم عشوائية إلى هذه المصفوفة باستخدام الحلقة التالية:

```
for (int i=0; i< array.length ; i++)
    for (int j=0; j< array[ i ].length ; j++)
        array[i][j] = (int) ( Math.random( ) * 1000 ) ;
```

تحتاج الصيغة `new int [5][ ]` إلى تحديد الدليل الأول من أجل إنشاء مصفوفة ، لأن الصيغة التالية خاطئة `new int [ ][ ]` .

مثال (6-4)

```
// initializing multidimensional arrays
class DimArray {
public static void main(String args[ ] ) {
int array1[ ][ ] = { {1, 2, 3}, {4, 5, 6} },
array2[ ][ ] = { {1, 2}, {4} } ;
```

```

        System.out.println("values in array1 by row are : ");
        for (int i=0; i< array1.length ; i++) {
        for (int j=0; j< array1[i].length ; j++)
        System.out.print(array1[i][j] + " ");
        System.out.println( );
        }
//Continued
System.out.println("values in array2 by row are : ");
        for (int i=0; i< array2.length ; i++) {
        for (int j=0; j< array2[ i ].length ; j++)
        System.out.print(array2[i][j] + " ");
        System.out.println( );
        }
} //End main

} //End Class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT
values in array1 by row are :
1 2 3
4 5 6
values in array2 by row are :
1 2
4
Process completed.

٤-٤-٤ - إدخال وإخراج عناصر المصفوفة ثنائية البعد

يمكن إدخال عناصر المصفوفة من قبل المستخدم ، وأيضاً يمكن إخراج (طباعة) عناصر المصفوفة ثنائية البعد بواسطة حلقتي (for ، أو

while ، أو do/while ، أو خليط بينهم) . عادة قبل استخدام المصفوفة يمكن تصغيرها ، وذلك بإعطاء قيم ابتدائية تساوي الصفر ، ويوضح المثال (4 - 7) كيفية إدخال وإخراج عناصر المصفوفة ثنائية البعد.

مثال (4-7)

اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال درجات 10 طلاب في خمس مقررات وتخزينها في مصفوفة ثنائية البعد a[10][5] .
- حساب وطباعة معدل كل طالب في المقررات الخمسة .
- حساب وطباعة المعدل الأكبر .

البرنامج:

```
// multidimensional arrays
import java.util.Scanner;
public class DimArray1 {
public static void main(String args[ ] ) {
Scanner input = new Scanner( System.in );
int Mark[ ][ ] = new int[10][5];
int ormax ;
double sum=0.0 , maxav ;
double avr[ ] =new double [10];
System.out.println("Enter Mrka[ ][ ] values : ");
for (int i=0; i< Mark.length ; i++ )
for (int j=0; j< Mark.length - 5 ; j++ ){
int x = input.nextInt( );
Mark[i][j]=x;
}
}
```

//Continued

```
for (int k=0; k<10; k++) {  
    sum=0.0 ;  
    for (int j=0; j<5; j++) {  
        sum=sum + Mark[ k ] [ j ] ;  
    }  
    avr[ k ]=sum / 5 ;  
}  
maxav = avr[0];  
ormax =1;  
for (int i2=0; i2<10; i2++) {  
    if( avr[ i2 ] > maxav ) {  
        maxav = avr[ i2 ] ;  
        ormax = i2 ;  
    }  
}  
for (int i1=0; i1<10; i1++) {  
    System.out.println(" avr [" +i1+ "]=" +avr[i1] ); }  
System.out.println(" The greatest average is " + maxav );  
System.out.println(" The greatest number is " + ormax );  
}//End main  
}//End Class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Enter Marks [] values : 66 77 88 22 77 77 77 77 77 77 88 88 88 88 88 55 77 88 55 55 88 66 44 66 66 99 88 88 88 88 55 66 66 66 66 77 77 77 66 55 55 66 99 66 77 55 66 88 99 77 Process completed	avr [0]=66.0 avr [1]=77.0 avr [2]=88.0 avr [3]=66.0 avr [4]=66.0 avr [5]=90.2 avr [6]=63.8 avr [7]=70.4 avr [8]=72.6 avr [9]=77.0 The greatest average is 90.2 The greatest number is 5
---	--

٤ - ٥ - تمرير المصفوفات كوسطاء للتابع

عند تمرير مصفوفة أحادية البعد كوسيط لتابع عند استدعائه في التابع `main()` يجب استخدام اسمها فقط بدون أقواس ، فعلى سبيل المثال ، إذا تم التصريح عن المصفوفة `a` على الشكل التالي :

```
int [ ] a = new int[ 20 ] ;
```

فإن الاستدعاء التالي للتابع `sumArray()` كما يلي :

```
sumArray( a , 20 ) ;
```

يأخذ التابع اسم المصفوفة وحجمها كوسيطين له ، وعند تمرير مصفوفة إلى تابع فإنه يتم تمرير حجمها أيضاً حتى يتعامل مع العدد المحدد من عناصر المصفوفة الممررة . إن استقبال المصفوفة المرسله عند استدعاء التابع يجب أن تحتوي قائمة وسطاء التابع عند تعريفه على اسم المصفوفة مع الأقواس `[ a ]` ، ويجب كتابة الجزء الرأسي للتابع `sumArray()` على الشكل التالي :

```
public static void sumArray( int a[ ] , int size )  
{  
    Body  
}
```

للدلالة على أن التابع `sumArray()` يتوقع استقبال مصفوفة من الأعداد الصحيحة حسب الوسيط `a` وهي تحتوي على `size` عنصراً . ليس من الضروري وضع حجم المصفوفة بين القوسين عند كتابة الوسيط `[ 20 ] a` ، وفي حال القيام بذلك فإن المترجم يقوم بتجاهله ، وعلى اعتبار أن المصفوفات يتم تمريرها بالعنوان فإن التابع المُستدعى يقوم باستخدام اسم المصفوفة `a` من أجل الرجوع إلى المصفوفة الأصلية الموجودة لدى التابع الذي تم الاستدعاء ضمنه المصفوفة ، ويوضح المثال (4 - 8) كيفية تمرير مصفوفة كوسيط لتابع .


```
// compute the sum of the elements of the array
import java.util.Scanner;
public class PassArray {
    public static void main(String args[ ] ) {
        Scanner input = new Scanner( System.in );
        int arry[ ] = new int [10] ;
        for ( int i=0 ; i< 10 ; i++ ) {
            System.out.println(" Enter Array ");
            int x = input.nextInt( );
            arry[i]=x;
        } // End for
        sumArray ( arry, 10 ) ;
    } //End main
    public static void sumArray ( int a [ ] , int n ) {
        int total = 0;
        for (int i= 0; i< n ; i++)
            total = total+ a[i];
        System.out.println(" total of array element values is "+ total ) ;
    } //End Method
} //End Class
```

وعند تنفيذ البرنامج نحصل على الخرج التالي:

RESULT

```
11 22 33 44 55 66 77 88 99 100
total of array element values is 595
Process completed
```

يوجد العديد من الحالات التي لا نرغب فيها بتغيير قيم عناصر المصفوفة ، وعلى اعتبار أن عملية تمرير مصفوفة تتم بالعنوان فإنه من الصعب التحكم بعملية التغيير . لكن يتوفر في Java التعليمية `final` لمنع القيام بأي تغيير على قيم عناصر المصفوفة الممررة عند استخدام التعليمية `final` من قبل أي مصفوفة معطاة كوسيط فإن عناصرها تصبح عبارة عن قيم ثابتة ضمن جسم التابع ، ولا يمكن تغيير قيمها وأي محاولة لذلك ستؤدي إلى حدوث خطأ أثناء ترجمة البرنامج، ما يساعد المبرمج على تصحيح برنامجه بشكل لا يسمح بالقيام بعمليات التغيير غير المطلوبة ، نعدل جزء من البرنامج في المثال (4-8) باستخدام التعليمية `final` .

```
public class PassArray {
    public static void main(String args[ ] ) {
        Scanner input = new Scanner( System.in );
        final int size=10 ;
        int array[ ] = new int [size] ;
        for ( int i=0 ; i< size ; i++ ) {
            System.out.println(" Enter Array ");
            int x = input.nextInt( );
            array[i]=x;
        } // End for
        sumArray ( array, size ) ;
    } //End main
}
```

عند تمرير مصفوفة ثنائية البعد كوسيط لتابع عند استدعائه في التابع `main( )` يجب استخدام اسمها فقط بدون أقواس ، فعلى سبيل المثال ، إذا تم التصريح عن المصفوفة `array` على الشكل التالي :

```
int array [ ][ ] = int [2][3] ;
```

فإن الاستدعاء التالي للتابع `printArray( )` كما يلي :

printArray(array) ;

وعلى اعتبار أن المصفوفات يتم تمريرها بالعنوان فإن التابع المُستدعى يقوم باستخدام اسم المصفوفة array من أجل الرجوع إلى المصفوفة الأصلية الموجودة لدى التابع الذي تم استدعاء المصفوفة ضمنه، ويوضح المثال (4 - 9) كيفية تمرير مصفوفة ثنائية البعد كوسيط لتابع .

مثال (4-9)

يوضح البرنامج التالي كيفية إسناد قيم أولية للمصفوفات متعددة الأبعاد عند التصريح عنها، وكيفية تمرير المصفوفات كوسائط للتوابع وطباعتها ، ثم استخدام التابع () printArray لطباعة عناصر المصفوفة .

```
// initializing multidimensional arrays
public class DimArray {
    public static void main(String args[ ] ) {
        int array1[ ][ ] = { { 1, 2, 3}, {4, 5, 6} },
            array2[ ][ ] = { { 1, 2}, {4} } ;
        System.out.println("values in array1 by row are : ");
        printArray( array1 ) ;

        //Continued
        System.out.println("values in array2 by row are : ");
        printArray(array2);
    } //End main

    public static void printArray(int array[ ][ ] ) {
        for (int i=0; i< array.length ; i++) {
            for (int j=0; j< array[ i ].length ; j++)
                System.out.print(array[i][j] + " ");
            System.out.println( );
        }
    } //End function
} //End Class
```

وعند تنفيذ البرنامج نحصل على الخرج التالي:

RESULT

values in array1 by row are :

1 2 3

4 5 6

values in array3 by row are :

1 2

4

Process completed

مثال (4-10)

اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال 12 عدداً صحيحاً ، ويتم تخزينها في مصفوفة ثنائية البعد a[3][4]
- طباعة عناصر المصفوفة a .
- حساب المتوسط الحسابي لعناصر المصفوفة وطباعته .
- استخدم التتابع في البرنامج كما يلي :
- التابع () printArray لطباعة عناصر المصفوفة .
- التابع () printArray1 لحساب مجموع عناصر المصفوفة وحساب المتوسط الحسابي .

// multidimensional arrays

import java.util.Scanner;

public class DimArray2 {

public static void main(String args[]) {

Scanner input = **new** Scanner(System.in);

int Array[][] = **new int**[3][4] ;

System.out.println("Enter Array [][] values : ");

for (**int** i=0; i< Array.length; i++)

```

    for (int j=0; j< Array[ i ].length; j++) {
        int x = input.nextInt( );
        Array[i][j]=x;
    }
    printArray(Array);    printArray1(Array);
} //End main
//Continued
public static void printArray ( int a[ ][ ] ) {
    for (int i=0; i< a.length; i++) {
        for (int j=0; j< a[ i ].length; j++)
            System.out.print( a[ i ][ j ] + " " );
        System.out.println();
    } //end for
} //End printArray
public static void printArray1(int a[ ][ ] ) {
    int sum=0;
    for (int i=0; i< a.length; i++)
        for (int j=0; j< a[ i ].length; j++)
            sum = sum + a[ i ][ j ];
    double avr = sum/12;
    System.out.println(" avr = "+ avr );
} //End printArray1
} //End Class

```

وعند تنفيذ البرنامج نحصل على الخرج التالي:

RESULT

```

Enter a[ ][ ] values :
1 2 3 4 5 6 7 8 2 4 6 8
Print Array
1 2 3 4
5 6 7 8
2 4 6 8
avr = 4.0
Process completed

```


نعيد كتابة البرنامج باستخدام الواجهات .

```
// multidimensional arrays
import javax.swing.JOptionPane;
public class DimArray2 {
public static void main(String args[ ] ) {
int Array[ ][ ] = new int[3][4] ;
    for (int i=0; i<3; i++ )
        for (int j=0; j<4; j++ ){
            String intString = JOptionPane.showInputDialog(null,
                "Enter Array numbers:\n أدخل الأعداد ", "My Example Array Input",
                JOptionPane.QUESTION_MESSAGE);
// Convert string into integer
            Array[i][j] = Integer.parseInt(intString);
        }
String output = printArray(Array)+"\n"+printArray1(Array);
// Display the result
    JOptionPane.showMessageDialog(null, output,
        "My Example Array Output",
        JOptionPane.INFORMATION_MESSAGE);
    System.exit(0);
} //End main
//Continued
public static String printArray ( int a[ ][ ] ) {
    String output= "\n";
    for (int i=0; i<3; i++){
        for (int j=0; j<4; j++){
            output= output+a[i][j]+" ";
        }
        output= output+"\n";
    }
return output;
} //End printArray
public static String printArray1(int a[ ][ ] ) {
```

```

int sum=0;
for (int i=0; i<3; i++)
for (int j=0; j<4; j++)
    sum = sum + a[ i ][ j ];
double avr = sum/12;
String output ="avr = "+avr;
return output;
} //End printArray1
} //End Class

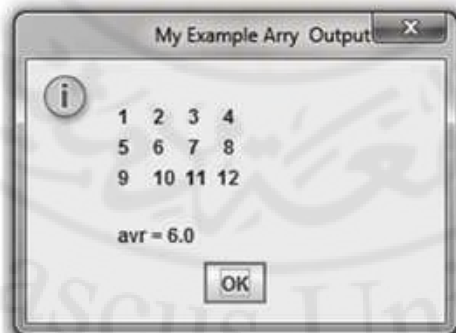
```

وعند تنفيذ البرنامج نحصل على الخرج التالي:



الخرج بعد إدخال عناصر
المصفوفة من (1 to 12)

تكرر العملية حسب حجم المصفوفة
(في حالتنا 3x4)



مثال (4-11)

اكتب برنامجاً بلغة C++ يقوم بما يلي :

- إدخال عناصر المصفوفة $a[3][3]$ وهي مصفوفة ثنائية البعد ذات أعداد حقيقية .



2.00	2.01	2.02
2.10	2.11	2.12
2.20	2.21	2.22

- طباعة عناصر المصفوفة a .
- حساب المتوسط الحسابي للأعداد التي تقع فوق القطر الرئيسي للمصفوفة وطباعته .

استخدم التتابع في كتابة البرنامج .

البرنامج :

```
// multidimensional arrays
import java.util.Scanner;
public class DimArray3 {
    public static void main(String args[ ] ) {
        Scanner input = new Scanner( System.in );
        int A[ ][ ];
        A = new int[3][3];
        System.out.println("Enter A[ ][ ] values : ");
        for (int i=0; i<3; i++)
            for (int j=0; j<3; j++){
                int x = input.nextInt();
                A[i][j]=x;
            }
        printArray1(A);
        printArray(A);
    } //End main
    public static void printArray ( int a[ ][ ] )
    {
```

```

int sum=0 ;
    int n = 0 ;
    for (int i=0; i<3; i++) {
        for (int j=0; j<3; j++)
            if( i<= j ) {
                sum=sum + a[ i ][ j ];
                n = n+1 ; }
    }
    double avr = sum / n ;
    System.out.println("avr = "+ avr );
} //End printArray1
public static void printArray1(int a[ ][ ]) {
    System.out.println( " Print Array");
    for (int i=0; i<3; i++){
        for (int j=0; j<3; j++)
            System.out.print( a[ i ][ j ] +" ");
        System.out.println();
    }
} //End printArray
} //End Class

```

وعند تنفيذ البرنامج نحصل على الخرج التالي:

RESULT

Enter a[][] values :
 11 22 33 55 66 77 99 12 13

 11 22 33
 55 66 77
 99 12 13

Avr = 44
Process completed



مسائل عامة

١- اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال 12 عدداً صحيحاً وتخزينها في مصفوفة ثنائية البعد $a[4][3]$.
- إدخال 6 أعداد صحيحة وتخزينها في مصفوفة ثنائية البعد $b[3][2]$.
- إجراء عملية ضرب المصفوفتين $a[4][3]$ و $b[3][2]$ وتخزين النتيجة في المصفوفة $c[4][2]$.
- طباعة عناصر المصفوفات الثلاثة $a[4][3]$ و $b[3][2]$ و $c[4][2]$.

توضيح رياضي: لنفرض أن:

$$a[4][3] = \begin{bmatrix} 3 & 4 & 5 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 5 & 6 & 7 \end{bmatrix}$$

عندئذ:

$$a[4][3] \times b[3][2] = \begin{bmatrix} 3 & 4 & 5 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 5 & 6 & 7 \end{bmatrix} \times \begin{bmatrix} 2 & 2 \\ 3 & 7 \\ 9 & 2 \end{bmatrix}$$

ومنه نجد:

$$a[4][3] \times b[3][2] = \begin{bmatrix} 3*2+4*3+5*9 & 3*2+4*7+5*2 \\ 1*2+1*3+1*9 & 1*2+1*7+1*2 \\ 2*2+2*3+2*9 & 2*2+2*7+2*2 \\ 5*2+6*3+7*9 & 5*2+6*7+7*2 \end{bmatrix}$$

وبالتالي:

$$a[4][3] \times b[3][2] = \begin{bmatrix} 63 & 44 \\ 14 & 11 \\ 28 & 22 \\ 91 & 66 \end{bmatrix}$$

وحيث إن دستور ضرب المصفوفتين بشكل رياضي هو التالي:

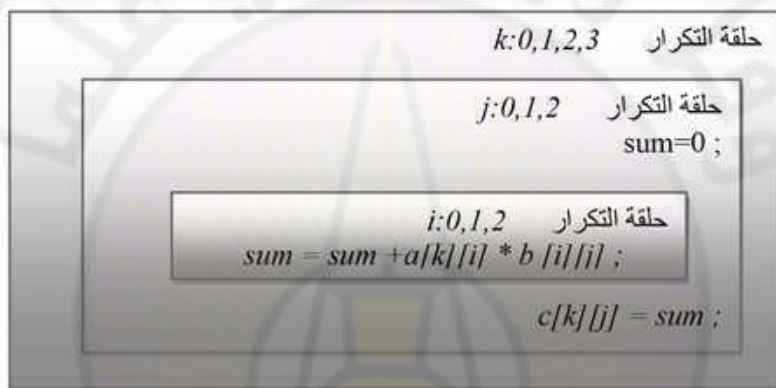
$$c[k][j] = \sum_{i=0}^2 a[k][i] \times b[i][j]; ; k = 0,1,2,3; ; j = 0,1$$

والعبارات البرمجية المقابلة لهذا الشكل الرياضي هي التالية:

$sum = sum + a[k][i] * b[i][j];$

$c[k][j] = sum$

وحيث إن هذه العبارات البرمجية بحاجة إلى ثلاث حلقات تكرار متداخلة هي:



=====

٢- اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال 9 أعداد صحيحة وتخزينها في مصفوفة ثنائية البعد $a[3][3]$.
- حساب مجموع ومتوسط الأعداد التي تقع على وفوق القطر الرئيسي للمصفوفة.

- طباعة عناصر المصفوفة المدخلة على شكل مصفوفة مربعة.

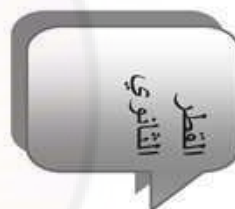
- طباعة مجموع ومتوسط الأعداد الذين تم إيجادهما .

توضيح رياضي:لنأخذ الشكل الرياضي للمصفوفة ثنائية البعد :

$$a[3][3] = \begin{bmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{bmatrix}$$

من خلال هذا الشكل الرياضي ، نلاحظ أن عنصر المصفوفة $a[i][j]$ يقع على القطر الرئيسي إذا كان رقم السطر i مساوياً لرقم العمود j ويقع عنصر المصفوفة $a[i][j]$ فوق القطر الرئيسي إذا كان رقم السطر i أصغر من رقم العمود j ، بينما يقع عنصر المصفوفة $a[i][j]$ تحت القطر الرئيسي إذا كان رقم السطر i أكبر من رقم العمود j .

لنأخذ الشكل الرياضي للمصفوفة المربعة الثنائية البعد $a[5][5]$:



$$a[5][5] = \begin{bmatrix} a[0][0] & a[0][1] & a[0][2] & a[0][3] & a[0][4] \\ a[1][0] & a[1][1] & a[1][2] & a[1][3] & a[1][4] \\ a[2][0] & a[2][1] & a[2][2] & a[2][3] & a[2][4] \\ a[3][0] & a[3][1] & a[3][2] & a[3][3] & a[3][4] \\ a[4][0] & a[4][1] & a[4][2] & a[4][3] & a[4][4] \end{bmatrix}$$

من خلال هذا الشكل الرياضي للمصفوفة المربعة (عدد الأسطر يساوي عدد الأعمدة) ، سنقدم الجدول الذي من خلاله سنتعرف على موقع كل عنصر $a[i][j]$ من عناصر المصفوفة المربعة الثنائية البعد وذلك بالنسبة لكلا القطرين الرئيسي والثانوي.

القطر	على	فوق	تحت
الرئيسي	$i=j$	$i<j$	$i>j$
الثانوي	$i+j=size-1$	$i+j<size-1$	$i+j>size-1$

حيث size هو بعد المصفوفة (وفي مثالنا: size = 5 ، عدد الأسطر يساوي عدد الأعمدة ويساوي ٥).

=====

٣- اكتب برنامجاً بلغة Java يقوم بما يلي:

- إدخال 5 أعداد صحيحة وتخزينها في مصفوفة أحادية البعد $n[5]$.
- حساب مربعات ومكعبات الأعداد المدخلة وتخزينها في المصفوفتين $s[5]$ و $c[5]$ على الترتيب.
- طباعة عناصر المصفوفات الثلاثة على أسطر مستقلة.

=====

٤- اكتب برنامجاً بلغة Java يقوم بما يلي :

- إدخال 25 عدداً صحيحاً من لوحة المفاتيح وتخزينها في مصفوفة ثنائية البعد $y[5][5]$.
- إيجاد العدد الأصغر من بين جميع الأعداد الواقعة على وفوق القطر الرئيسي.
- طباعة عناصر المصفوفة المدخلة على شكل مصفوفة مربعة.
- طباعة قيمة العدد الأصغر الذي تم إيجاده.

=====

- ٥- اكتب برنامجاً بلغة Java يقوم بما يلي :
- إدخال 25 عدداً حقيقياً من لوحة المفاتيح وتخزينها في مصفوفة ثنائية البعد $x[5][5]$.
 - حساب المتوسط الحسابي لجميع الأعداد الواقعة تحت القطر الثانوي.
 - طباعة عناصر المصفوفة المدخلة على شكل مصفوفة مربعة.
 - طباعة قيمة المتوسط الحسابي الناتج.

=====

- ٦- اكتب برنامجاً بلغة Java يقوم بما يلي :
- إدخال 16 عدداً حقيقياً من لوحة المفاتيح وتخزينها في مصفوفة ثنائية البعد $x[4][4]$.
 - حساب المتوسط الحسابي لعناصر المصفوفة الواقعة في السطر الأول.
 - إيجاد العنصر الأكبر من بين عناصر المصفوفة الواقعة في العمود الأول.
 - طباعة عناصر المصفوفة المدخلة على شكل مصفوفة مربعة.
 - طباعة النتائج التي تم التوصل إليها من خلال الطلبين الثاني والثالث.

=====

- ٧- اكتب برنامجاً بلغة Java يقوم بما يلي :
- ❖ يدخل 20 عدداً حقيقياً ويخزنها في مصفوفة أحادية البعد $A[20]$.
 - ❖ ويحتوي البرنامج على تابعين:
 - التابع الأول اسمه $SumAve()$ مهمته حساب المتوسط الحسابي لعناصر المصفوفة.
 - أما التابع الثاني اسمه $EvenArray$ مهمته حساب مجموع الأعداد الزوجية في المصفوفة .

❖ يستدعي التابع main التابعين ويطبع المتوسط ومجموع الأعداد الزوجية .

=====

٨- اكتب برنامجاً بلغة Java يقوم بما يلي :

❖ يدخل 20 عدداً حقيقياً ويخزنها في مصفوفة أحادية البعد A[20].

❖ ويحتوي البرنامج على تابعين:

- التابع الأول اسمه (MaxArry) مهمته حساب العنصر الأكبر للمصفوفة .

- أما التابع الثاني اسمه OddArry مهمته حساب مجموع الأعداد الفردية في المصفوفة .

❖ يستدعي التابع main التابعين ويطبع العنصر الأكبر ومجموع الأعداد الفردية .

=====

٩- اكتب برنامجاً بلغة Java يقوم بما يلي :

❖ يدخل 20 عدداً حقيقياً ويخزنها في مصفوفة أحادية البعد A[20].

❖ ويحتوي البرنامج على تابعين:

- التابع الأول اسمه (SumAve) مهمته حساب المتوسط الحسابي لعناصر المصفوفة.

- أما التابع الثاني اسمه EvenArry مهمته حساب مجموع الأعداد الزوجية في المصفوفة .

❖ التابع main يستدعي التابعين ويطبع المتوسط ومجموع الأعداد الزوجية

=====

١٠- اكتب برنامجاً بلغة Java يقوم بما يلي :

- ❖ يدخل 20 عدداً حقيقياً ويخزنها في مصفوفة أحادية البعد [20].A
- ❖ ويحتوي البرنامج على تابعين:
- التابع الأول اسمه (MaxArry) مهمته حساب العنصر الأكبر للمصفوفة .
- أما التابع الثاني اسمه OddArry مهمته حساب مجموع الأعداد الفردية في المصفوفة .
- ❖ التابع main يستدعي التابعين ويطبع العنصر الأكبر ومجموع الأعداد الفردية .

=====

١١- أكتب برنامجاً بلغة Java يقوم بما يلي:

- ❖ إدخال 10 أعداد صحيحة وتخزينها في مصفوفة أحادية البعد [10].a
- ❖ يستخدم تابع من النوع void اسمه sort لترتيب الأعداد المدخلة بشكل تصاعدي.
- ❖ التابع main يستدعي التابع sort ويطبع طباعة الأعداد المدخلة بعد عملية الترتيب .

=====

١٢- اكتب برنامجاً بلغة Java يقوم بما يلي:

- ❖ إدخال 9 أعداد صحيحة وتخزينها في مصفوفة ثنائية البعد [3][3]. x
- ❖ طباعة عناصر المصفوفة [3][3]. x على شكل مصفوفة مربعة .
- ❖ يستخدم تابع من النوع void لإيجاد كافة الأعداد الأولية من بين عناصر المصفوفة المدخلة ، ومن ثم طباعة هذه الأعداد وأيضاً طباعة عددها .

❖ يستدعي التابع main التابع لطباعة النتائج .

=====

١٣- اكتب برنامجاً بلغة Java يقوم بما يلي:

- يُدخل أي عدد صحيح n .
- يستخدم تابع اسمه factiral ، مهمته حساب العامل (!) لأي عدد صحيح .
- يقوم بحساب قيمة المقدار الرياضي التالي:
$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$$
- يطبع النتيجة في التابع الرئيسي main

=====



الفصل الخامس

الصفوف في Java

Classes In Java





الصفوف في Java

٥-١ - مقدمة

تُعَد البرمجة غرضية التوجه OPP طريقة قوية للتعامل مع المسائل البرمجية ، فقد ظهرت في كل نقطة من مراحل تطورها طريقة جديدة تساعد المبرمجين على التعامل مع البرامج المتزايدة التعقيد . ورغم أن البرمجة البنيوية كانت تعطي نتائج ممتازة عندما تطبق على برامج متوسطة التعقيد ، إلا أنها كانت تفشل عند بعض النقاط حين يصل البرنامج إلى حد معين ، ولكتابة برامج أكثر تعقيداً كانت الحاجة إلى طريقة جديدة في العمل البرمجي ، وتلبية لهذه الحاجة لذلك ظهرت البرمجة غرضية التوجه واختصاراً " OOP " . ويوجد عدد غير محدود من التعريفات والتفسيرات لمصطلح البرمجة غرضية التوجه ، ولكن يمكن وصف البرمجة غرضية التوجه بأنها البرمجة المتعلقة بالمعطيات والطرائق (التوابع) التي تستخدم تلك المعطيات، حيث يتم تسمية المعطيات والطرائق باسم معين هو الغرض Object ويكون هذا الغرض مكتفياً ذاتياً ، إذ يشكل وحدة برمجية متكاملة .

وحسب هذا التعريف فإن كثيراً من البرامج التي تقوم بمهمة معينة وتحتوي على المعطيات التي تحتاجها لأداء عملها تسمى الغرض Object . فمثلاً قيامنا برسم شكل أو صورة على شاشة الحاسوب هذا غرض ، وبالتالي نستطيع وصف تلك الصورة أو الشكل باللون والحجم وكثافة التظليل ومواصفات أخرى . إن لغة Java توفر وسيلة تسمى وسيلة التصنيف أو الصفوف ، وهذه الوسيلة تشبه عملية بنى المعطيات Data Structures ولكنها تتصف بصفات أخرى لا تتصف بها عملية بنى المعطيات حيث تحتوي هذه الوسيلة على برامج بحد ذاتها ، وتعتمد هذه البرامج على بنى المعطيات .

إن تحديد حالة الصف class بحد ذاتها تسمى غرضاً Object، وتوفر وسيلة الصفوف مجموعة من الميزات في أداء البرمجة حيث يمكن للمبرمج تعريف متحولات تابعة للصف ، ويطلق عليها متحولات خاصة لا يمكن الوصول إليها من خارج الصف نفسه . وهذا الأسلوب يساعد على حماية المعلومات من البرامج غير المسموح لها بالاستفادة من معطيات معينة .

وباستخدام وسيلة الصف فإن البرامج الكبيرة ستكون واضحة المعالم حيث تكون جميع البرامج ذات العلاقة في مواقع محددة لها ، وهذا الأسلوب يؤدي إلى مفهوم الوحدات المتكاملة و بالتأكيد يؤدي إلى الصيانة الكفاءة وتسهيلها .

٥-٢ - بنية الصف في لغة Java

يصرح عن الصف بلغة Java باستخدام الكلمة المحجوزة **class** ، ويشبه التصريح عن الصف بالتصريح عن البنية structure ، والبنية العامة للصف لها الشكل التالي :

```
class class_name {           // 1
    type instance-variable1;  // 2
    type instance-variable2;
    // ...
    type instance-variableN;
    type methodname1(parameter-list) { //3
// body of method
    }
    type methodname2(parameter-list) {
// body of method
```

```

    }
    // ...
    type methodnameN(parameter-list) {
    // body of method
    }
} //End class

```

يبدأ الصف باسم الصف *class_name* (السطر 1) ثم القوسين " { } " (السطر 2) الذي يدلان على بداية جسم الصف ونهايته ، ثم المتحولات (السطر 3) والتتابع أو الطرائق (السطر 3) المصرح عنها داخل الصف وتدعى بأعضاء الصف *class members* ، وتكون جميع المتحولات والتتابع المصرح عنها داخل الصف خاصة بهذا الصف ، ويتكون الصف *class_name* من قسمين : القسم الأول : يتضمن أعضاء المعطيات *Data Members* للصف أو ما يسمى بالصفات *Properties* :

```

type instance-variable I;
// ...
type instance-variable N

```

القسم الثاني : يتضمن طرائق الصف *Class Methods* (التتابع) أو ما يسمى بالأعمال :

```

type methodname I(parameter-list) {
// body of method
}
// ...

```

يدعى التابع الذي يعرف ضمن الصف تابعاً عضواً (member method) ، وتدعى المعطيات المعرفة ضمن الصف بأعضاء المعطيات (Data members)

إن الصف class هو عبارة عن بنية تحتوي على الأسماء التعريفية والمعطيات بأنواعها **int** , **char** وأيضاً **float** , **double** ، وتحتوي أيضاً على تعاريف التوابع العامة بأنواعها .

يوضح المثال (5-1) بناء صف اسمه Box الذي يحسب حجم متوازي مستطيلات أبعاده : width ، height ، depth . ويحتوي الصف على تابع طباعة لحساب الحجم ، وحتى يتم تنفيذ تطبيق مكتوب بلغة Java لا بد من أن يتضمن هذا التطبيق صفاً يحوي الطريقة (التابع) (main) ، وحتى نتمكن من استخدام الصف Box سنقوم بإنشاء صف الاختبار التطبيقي BoxTest للحصول على النتائج .

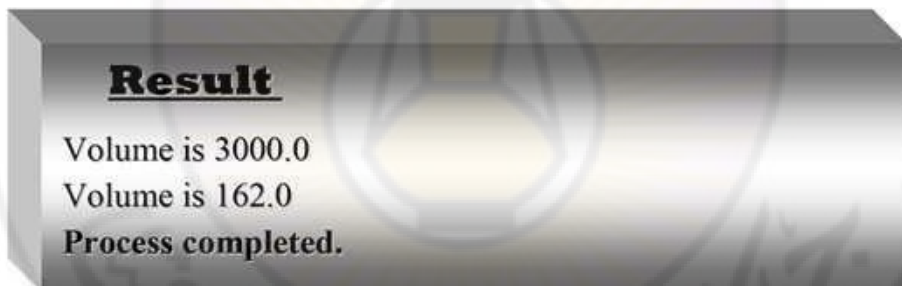
مثال (5-1)

```
public class Box {  
    double width;  
    double height;  
    double depth;  
    // display volume of a box  
    public void volume() {  
        System.out.print("Volume is ");  
        System.out.println(width * height * depth);  
    } // end volume  
} // end class Box  
  
public class BoxTest {  
    public static void main(String args[] ) {  
        Box mybox1 = new Box();  
        Box mybox2 = new Box();
```



```
// assign values to mybox1's instance variables
mybox1.width = 10;   mybox1.height = 20;
mybox1.depth = 15;
/* assign different values to mybox2's instance variables */
mybox2.width = 3;    mybox2.height = 6;
mybox2.depth = 9;
// display volume of first box
mybox1.volume( );
// display volume of second box
mybox2.volume( );
} //end main
} // end class BoxTest
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



إن متحولات الأعضاء width ، height ، depth في الصف Box ، تُعد متحولات من نوع package لأننا لم نحدد محدد الوصول للمعطيات ، لذلك تكون المعطيات مرئية من قبل الصفوف الموجودة في الحزمة ذاتها . لذلك أسندنا قيم للمتحولات عند إنشاء الأغراض في التابع main في الصف Box ، لأن التابع main يستطيع الوصول إلى هذه المعطيات لأنه في الحزمة نفسها . ويستخدم المعامل new لإنشاء غرض جديد أو مصفوفة جديدة

Box mybox1 = new Box();

أي يتم إنشاء غرض من الصف Box وحجز مكان لهذا الغرض في الذاكرة .
ويمكن أن يحتوي البرنامج على بانٍ لتمرير المعطيات والعملية تشبه تماماً كيفية
إنشاء بانٍ عادي أو افتراضي في لغة C++ ، ويبين البرنامج في المثال (2-5)
كيفية إنشاء بانٍ في صف بينما التابع Volume من نوع يعيد قيمة .

مثال (2-5)

```
class Box1 {
    double width , height , depth ;
    public Box1 ( )
    {
        width = 0 ;height = 0 ;depth = 0 ;
    }
    public Box1 ( double w ,double h ,double d )
    {
        width = w ;height = h ;depth = d ;
    }
    // display volume of a box1

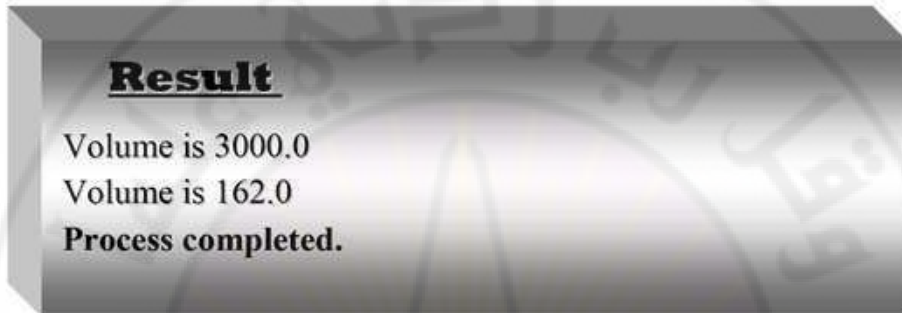
    public void volume( ) {
        System.out.print("Volume is ");
        System.out.println(width * height * depth);
    }
} //End class Box1

class Box1Test {
    public static void main(String args[ ] ) {

        Box1 mybox1 = new Box1( 10,20,15);
        Box1 mybox2 = new Box1( 3,6,9);
        // display volume of first box
        mybox1.volume( );
```

```
// display volume of second box
mybox2.volume( );
} //end main
} // End class Box1Test
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



لندرس بناء المعطيات في الصف Box1

- يبدأ الصف بالتعليمة class ثم اسم الصف Box1 ، ثم المتحولات من نوع package والتوابع العامة public ، وذلك ضمن القوسين { } اللذين يسميان بجسم الصف .
- تدعى المحددات package و public بمحددات طريقة الوصول إلى الأعضاء Member Access Specifies . ويفيد التصريح عن المعطيات الأعضاء والتوابع الأعضاء بعد التعليمة public في جعل هذه المعطيات أو التوابع متاحة للاستخدام من أي مكان ضمن البرنامج وتسمى بالمعطيات التوابع العامة أو واجهة الصف Interface of class ، حيث يمكن الوصول إلى أي غرض من الصف Box1 .
- يمكن استدعاء التوابع الأعضاء () volume و () Box1 و Box1 (double w ,double h ,double d) من أي جزء من البرنامج لأنها أعضاء عامة في الصف Box1 .

- أما متحولات الأعضاء `depth` ، `height` ، `width` في الصف `Box1` فهي من نوع `package` لأننا لم نصرح عنها وبالتالي يمكن الوصول إليها من قبل التوابع والصفوف الموجودة في الحزمة نفسها .
- نسمي التابع العضو () `Box1` الذي يحمل اسم الصف بالتابع الباني `Constructor` المرتبط بذلك الصف ، و يقوم التابع الباني بإعطاء قيم ابتدائية وتمرير القيم إلى المعطيات الأعضاء لكل غرض من أغراض الصف حيث يتم استدعاؤه آلياً عند إنشاء الغرض .
- أما تعريفات المعطيات بعد التعليلة `private` وهي تدل على أنه لا يمكن الوصول إلى المعطيات الأعضاء الخاصة إلا من خلال التوابع الأعضاء العامة.
- عادة تُذكر المعطيات الأعضاء الخاصة بعد المحدد الخاص `private` وتذكر التوابع الأعضاء العامة بعد المحدد العام `public` .
- إن المعطيات الخاصة بعد محدد الوصول `private` هي معطيات خاصة بالصف ، ولا يمكن الوصول إليها من خارجه وتهدف إلى حجب المعطيات الخاصة عن المستخدمين له ، فهي تعد بنية مخفية عن زبائنه ، وهي تساعد على زيادة القدرة على تعديل البرنامج وعلى تبسيط تعامل الزبائن مع الصف .
- ويقوم التابع الباني بإسناد قيم إلى معطيات الأعضاء ، أي تمرير المعطيات، ويسمى بتابع تمرير المعطيات، ويمكن استخدام تابع غير التابع الباني في تمرير المعطيات ويسمى بتابع تعديل المعطيات للقيام بذلك .
- إن التوابع الأعضاء مثل التابع () `volume` تُعرف بشكل ضمني وأنه يقوم بحساب وطباعة حجم متوازي المستطيلات للغرض الذي يرتبط بالصف `Box1` ، ما يساعد على جعل التابع العضو أكثر ارتباطاً بالمعطيات .

إن التابع () Box1 الذي يحمل اسم الصف هو عبارة عن تابع بانى والذي يستدعى عند تكوين الصف Box1 ، ويتم استدعاء التابع البانى عند إنشاء أي غرض object من الصف بشكل آلي . لا يمكن إعطاء قيم ابتدائية للمعطيات الأعضاء ضمن تعريف الصف وإنما يتم بواسطة التابع البانى المرتبط بالصف .

□ لنعيد كتابة البرنامج في المثال (5-1) من أجل إدخال وإخراج المعطيات عن طريق الواجهات . وبالتالي التعديلات التي سوف تحدث على البرنامج هي كما يلي :

١. نضيف السطر `import javax.swing.JOptionPane ;` كأول سطر في بداية البرنامج قبل الصف `class Box1` من أجل استخدام الواجهات .

٢. الصف `class Box1` يبقى كما هو بدون تعديل ما عدا التابع `volume ( )` حيث يصبح كما يلي :

```
double volume( ) {  
    return (width * height * depth);  
}
```

٣. أما الصف `class Box1Test` الذي يحوي التابع () `main` يصبح كما يلي :

```
class Box1Test {  
    public static void main(String args[ ] ) {  
        double w1,h1,d1;  
        String javaString = JOptionPane.showInputDialog(null,  
            "Enter a w1 Number ", "My Volume Example",  
            JOptionPane.QUESTION_MESSAGE);  
        w1 = Double.parseDouble(javaString) ;  
        javaString = JOptionPane.showInputDialog(null,
```



```

        "Enter a h1 Number ", "My Volume Example",
        JOptionPane.QUESTION_MESSAGE);
h1 = Double.parseDouble(javaString) ;
javaString = JOptionPane.showInputDialog(null,
        "Enter a d1 Number ", "My Volume Example",
        JOptionPane.QUESTION_MESSAGE);
d1 = Double.parseDouble(javaString) ;
double w2,h2,d2;
        javaString = JOptionPane.showInputDialog(null,
        "Enter a w2 Number ", "My Volume Example",
        JOptionPane.QUESTION_MESSAGE);
w2 = Double.parseDouble(javaString) ;
        javaString = JOptionPane.showInputDialog(null,
        "Enter a h2 Number ", "My Volume Example",
        JOptionPane.QUESTION_MESSAGE);
h2 = Double.parseDouble(javaString) ;
        javaString = JOptionPane.showInputDialog(null,
        "Enter a d2 Number ", "My Volume Example",
        JOptionPane.QUESTION_MESSAGE);
d2 = Double.parseDouble(javaString) ;

Box1 mybox1 = new Box1( w1,h1,d1);
Box1 mybox2 = new Box1( w2,h2,d2);
// display volume of first box
        String ouput =" Volume1 is " + mybox1.volume( )+ "\n";
// display volume of second box
        ouput = ouput +" Volume2 is " + mybox2.volume( );
        JOptionPane.showMessageDialog(null, ouput "My First Class",
JOptionPane.ERROR_MESSAGE );
    } //end main
} // End class Box1Test

```




٥-٣- محددات الوصول إلى أعضاء الصف

إن مستوى الوصول هو الذي يسمح لنا بأن نتحكم بوصول الصفوف الأخرى إلى المتحول العضوي من خلال واحد من أربعة مستويات وصول هي :
`private` , `protected` , `public` , `package`
 إن إحدى فوائد الصفوف أنها تستطيع حماية المتحولات الأعضاء وكذلك أعضائها من أن تصل إليها الأغراض الأخرى . وجميع محددات الوصول في لغة Java تشبه محددات الوصول في لغة C++ ، ما عدا محدد الوصول `package` .

١. محدد الوصول private :

يشبه محدد الوصول في لغة ++C ، و يُعَدّ جميع المتحولات خاصة في هذا الصف ويمكن الوصول إليها من أي مكان داخل الصف ، ولا يمكن الوصول إليها من خارج هذا الصف .

٢. محدد الوصول protected :

يشبه محدد الوصول في لغة ++C ، يستطيع الصف الذي يحتوي هذه المتحولات الوصول إليها وكذلك الصفوف المشتقة (الوراثة) من هذا الصف التي هي في نفس الحزمة .

٣. محدد الوصول public :

إن محدد الوصول public هو الأسهل من بين جميع محددات الوصول . إن أي صف لأي حزمة يستطيع الوصول إلى هذه الأعضاء ، وهذه الأعضاء مرئية داخل هذا الصف وجميع الصفوف من خارج هذا الصف .

٤. محدد الوصول package :

إن محدد الوصول package هو مستوى الوصول الذي نحصل عليه إذا لم نحدد صراحة نوع الوصول الذي هو واحداً من محددات (public ، private ، protected) . وإن مستوى الوصول هذا يسمح للصفوف في نفس الحزمة أن تصل إلى المتحولات الأعضاء (الصف نفسه والصفوف المشتقة والصفوف من خارج هذا الصف) ويفترض أن الصفوف التي في نفس الحزمة هم أصدقاء . إذا جعلنا المتحولات الأعضاء من نوع private في الصف Box1 عندها لا نستطيع الوصول إليها من خارج هذا الصف ، نعيد كتابة الصف Box1 في المثال (5 - 2) على اعتبار أنها متحولات خاصة .

```
public class Box1 {  
    private double width;
```

مثال (3-5)

```

private double height;
private double depth;
public Box ( ) {
width = 0 ;height = 0 ;depth = 0 ;
}
public Box ( double w ,double h ,double d ) {
width = w ;height = h ;depth = d ;
}
public void volume( ) {
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
} //End class Box1
public class Box1Test {
public static void main(String args[ ] ) {
Box1 mybox1 = new Box1( 10,20,15);
Box1 mybox2 = new Box1( 3,6,9);
// display volume of first box
mybox1.volume( );
// display volume of second box
mybox2.volume( );
} //end main
} // End class Box1Test

```

٥-٤- تعدد البواني (التحميل الزائد للبواني)

إن تمرير المتحولات إلى التابع الباني عملية سهلة وبسيطة وذلك بإضافة الوسيط المناسب إلى تصريح تعريف التابع الباني ، ثم تعيين قيمة الوسيط عند التصريح عن الغرض من الصف . فتعدد البواني هو إعادة استخدام اسم التابع الباني نفسه ضمن سياقات مختلفة أحدهما بدون وسيط أو بوسيط واحد أو اثنين أو ثلاثة... الخ .

لنأخذ الصف Box على سبيل المثال

```
public class Box {  
    public Box ( ) { .... } // باني افتراضي بدون وسيط  
    public Box (double w ) { .... } // باني بوسيط واحد  
    public box (double w , double h ) { .... } // باني بوسيطين  
    public Box (double w ; double h , double d ) { .... }  
    // باني بثلاث وسطاء  
} // End Class .
```

فإن المترجم يعرف من خلال السياق أيّاً من البواني سيطبق .
بالنسبة للصف Box من الضروري فقط وجود بانين أحدهما افتراضي بدون
وسيط والثاني يأخذ ثلاثة وسطاء (في مثالنا) ، ولا يمكن أن تحدد التوابع البانية
أية أنماط أو قيم معادة .

٥-٥ - الباني الافتراضي

Default Constructor

هو بان يحمل اسم الصف ، وهو بدون وسطاء . إذا لم نكتب أي بان فإن
المترجم يقوم تلقائياً بتقديم بان افتراضي ، أما إذا كان هناك أي بان فإن المترجم
لا يقدم أي بان افتراضي ، وعند إنشاء غرض يمكن لأعضائه أن تأخذ قيماً
ابتدائية بواسطة الباني الافتراضي ، ويمكن تعديل هذه القيم لاحقاً بعد إنشاء
الغرض ، مثلاً إذا عرفنا الصف Box بدون بان كما يلي :

```
public class Box1 {  
    private double width , double height , double depth;  
    public void setBox ( double w ,double h ,double d )  
    {  
        width = w ;height = h ;depth = d ;  
    }  
}
```



```

public void volume( ) {
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
} //End class Box1

```

نظراً لأننا كتبنا الصف Box بدون بان فإن المترجم يقوم بتقديم بان افتراضي تلقائياً وبدون وسيط ، وبسبب وجود الباني الافتراضي يمكننا استخدام الصف للتصريح عن أغراض ، أي :

```

Box mybox1 = new Box( );
Box mybox2 = new Box( );

```

إذا لم نكتب أي بان فإن المترجم يقوم بتزويدنا ببان افتراضي بحيث يمكن لمستخدم الصف أن يصرح عن أغراض . أما إذا احتوى الصف على بان لا يمكن التصريح عن أغراض بدون وسطاء إلا إذا احتوى الصف على بان افتراضي بالإضافة إلى الباني العادي .

والأفضل أن نكتب بان افتراضي خاصاً بنا بدلاً من الاعتماد على المترجم . ويمكن أن يكون الباني الافتراضي بسيطاً بالقدر الذي نريده ، ويمكن أن يكون خالياً من التعليمات .

إن سلوك الباني الافتراضي المزود من قبل المترجم هو إسناد جميع أعضاء المعطيات إلى أصفار لضمان قيام الصف بالعمل الصحيح .

٥-٦- بناء الصف Point

الصف Point يحتوي على ثلاثة متحولات من نوع private من النمط int وهما: x, y, z . ويحتوي الصف على بان افتراضي لإعطاء قيم ابتدائية للأغراض عند إنشائها ويحتوي على بان عادي لتحرير المعطيات وعلى تابع طباعة printpoint لطباعة إحداثية النقطة الموجبة حيث الصف يعرف دوماً

نقطة موجبة.

أنشئ صفاً آخر يتضمن التابع main اسمه PointTest لإنشاء أغراضاً لاستدعاء التوابع اللازمة والحصول على النتائج .

مثال (4-5)

```
public class Point {
    private int x, y, z;
    public Point () { x=0; y=0 ;z=0;}
    public Point(int new_x , int new_y , int new_z) {
        if ( new_x <0 )
            new_x *= -1;
        if ( new_y <0 )
            new_y *= -1;
        if ( new_z <0 )
            new_z *= -1;
        x=new_x ;
        y=new_y;
        z=new_z ;
    }
    public void printPoint() {
        System.out.print("Point is ");
        System.out.println( "(" +x+ " , " +y +"," +z+ ")" );
    }
} //End class Point

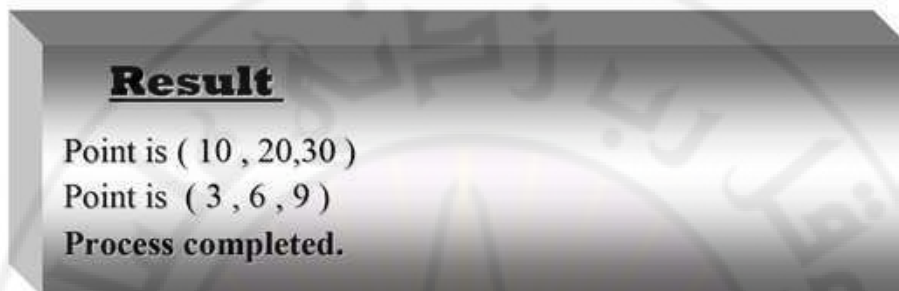
public class PointTest {
    public static void main(String args[ ]) {
        Point mypoint1 = new Point(10,-20,30);
        Point mypoint2 = new Point( -3,6,-9);
        // display first Point
        mypoint1.printPoint( );
        // display second Point
```

```

mypoint2.printPoint( );
} //End main
} //End class PointTest

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



□ نعيد كتابة البرنامج من أجل الإدخال والإخراج عن طريق واجهة .

التعديلات التي سوف تحدث على البرنامج هي كما يلي :

- نضيف السطر `import javax.swing.JOptionPane ;` كأول سطر في بداية البرنامج قبل الصف `class Point`.

- الصف `class Point` يبقى كما هو بدون تعديل ما عدا التابع `printPoint( )` حيث يصبح كما يلي :

```

public String printPoint( ) {
    String output = "Point is ";
    output=output+ ( "(" +x+ " , " +y +"," +z+ ")" );
    return output ; }

```

أما الصف `class PointTest` الذي يحوي التابع `main( )` يصبح كما يلي

```

public class PointTest {
    public static void main(String args[ ]) {
        int x1 ,x2, y1,y2,z1,z2 ;// 10, 20,30 ---, -3,6,-9
        String javaString = JOptionPane.showInputDialog(null,
            "Enter a x1 Number ", "My Point Example",
            JOptionPane.QUESTION_MESSAGE);
    }
}

```

```

x1 = Integer.parseInt(javaString) ;
javaString = JOptionPane.showInputDialog(null,
    "Enter a y1 Number ", "My Point Example",
    JOptionPane.QUESTION_MESSAGE);
y1 = Integer.parseInt(javaString) ;
javaString = JOptionPane.showInputDialog(null,
    "Enter a z1 Number ", "My Point Example",
    JOptionPane.QUESTION_MESSAGE);
z1 = Integer.parseInt(javaString) ;
//-----
javaString = JOptionPane.showInputDialog(null,
    " Enter a x2 Number ", "My Point Example",
    JOptionPane.QUESTION_MESSAGE);
x2 = Integer.parseInt(javaString) ;
javaString = JOptionPane.showInputDialog(null,
    " Enter a y2 Number ", "My Point Example",
    JOptionPane.QUESTION_MESSAGE);
y2 = Integer.parseInt(javaString) ;
javaString = JOptionPane.showInputDialog(null,
    " Enter a z2 Number ", "My Point Example",
    JOptionPane.QUESTION_MESSAGE);
z2 = Integer.parseInt(javaString) ;
//-----
Point mypoint1 = new Point(x1,y1,z1);
Point mypoint2 = new Point( x2,y2,z2);
// display the first point
String output = "\n";
output = mypoint1.printPoint()+"\n";
// display the second point
output = output + mypoint2.printPoint()+"\n";
JOptionPane.showMessageDialog(null, output,

```

```
"My POINT Class ", JOptionPane.ERROR_MESSAGE );
} //End main
} //End class PointTest
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٥ - ٧ - بناء الصف Time

يحتوي الصف Time على ثلاثة متحولات (hour , minute , second) وهي متحولات خاصة من نمط int ، ويتم طباعة الوقت بالشكل القياسي standard والشكل العام أي العسكري . بحيث يتم تعيين قيم ابتدائية للمتحولات الثلاثة بالشكل القياسي (AM 0 : 0 : 12) وبالشكل العام (0 : 0 : 0) ،

وعند الإدخال يجب أن يطبع (0) إذا كانت hour و minute و second

ذات قيم غير صحيحة . و يتضمن الصف Time :

- بأن افتراضي يسند القيمة صفر للمتحويلات الخاصة .
- بأن عادي يمرر المعطيات إلى الأغراض عند إنشائها .
- بأن وسيطه غرض .
- بأن بوسيط واحد يسند القيمة نفسها إلى المتحويلات الخاصة .
- بأن بوسيطين يقوم بإسناد قيمة للساعات وقيمة أخرى للدقائق والثواني .
- تابع طباعة بالشكل القياسي .
- تابع طباعة بالشكل العسكري .

ويتضمن البرنامج صفاً آخر يحتوي على التابع main اسمه TimeTest لإنشاء أغراض واستدعاء التوابع اللازمة والحصول على النتائج .

مثال (5-5)

```
public class Time {  
    private int hour , minute , second;  
    public Time( ) { hour=0;minute=0;second=0 ; }  
    public Time ( int h ,int m ,int s ) {  
        hour=(h>=0 && h<24)? h:0;  
        minute=(m>=0 && m<60)? m:0;  
        second=(s>=0 && s<60)? s:0;  
    }  
    public Time( Time ob ) {  
        hour=ob.hour;  
        minute=ob.minute;  
        second=ob.second;  
    }  
    public Time( int h , int ms ) {  
        hour=h ;minute= second=ms;  
    }  
}
```



```

}
public Time( int val ){
    hour=minute=second=val;
}
public void printS( ) {
    System.out.printf( "%d : %02d : %02d %s \n",
        ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
        minute, second, ( hour < 12 ? "AM" : "PM" ) );
}
public void prinU ( ) {
    System.out.printf( “ %d %d %d \n ",
        hour , minute,second) ;
}
} //end class time

```

```

Public class TimeTest {
public static void main(String args[ ]) {
    Time  myTime0 = new Time( );
    Time  myTime1 = new Time( 8,15,40);
    Time  myTime2 = new Time( 18,45,35 );
    Time  myTime3 = new Time(myTime1 );
    Time  myTime4 = new Time(18 );
    Time  myTime5 = new Time(10,44 );
    // display initial Time
    System.out.println(" initial Time U  ");
    myTime0.printu( );
    System.out.println(" initial Time S  ");
    myTime0.prints( );
    // display first Time
    System.out.println(“ first Time U  ");
    myTime1.printu( );

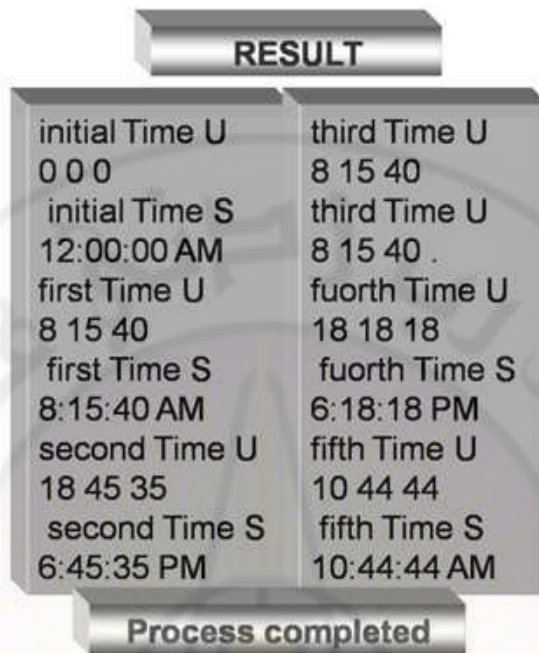
```

```

System.out.println(" first Time S ");
myTime1.prints( );
// display second Time
System.out.println(" second Time U ");
myTime2.printu( );
System.out.println(" second Time S ");
myTime2.prints( );
// display third Time
System.out.println("third Time U ");
myTime3.printU( );
System.out.println("third Time S ");
myTime3.printS( );
// display fuorth Time
System.out.println("fuorth Time U ");
myTime4.printU( );
System.out.println("fuorth Time S ");
myTime4.printS( );
// display fifth Time
System.out.println(" fifth Time U ");
    myTime5.printU( );
System.out.println(" fifth Time U ");
    myTime5.printS( );
} //end main
} // End Class TimeTest

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



□ نعيد كتابة البرنامج من أجل إدخال المعطيات وطباعة النتيجة على واجهة

```
import javax.swing.JOptionPane ;
public class TimeGUI {
private int hour, minute, second;
public TimeGUI() {
    hour=0;minute=0;second=0 ;
}
public TimeGUI ( int h,int m ,int s ) {
    hour=(h>=0 && h<24)? h:0;
    minute=(m>=0 && m<60)? m:0;
    second=(s>=0 && s<60)? s:0;
}
```

```

public TimeGUI( TimeGUI ob ) {
    hour=ob.hour;
    minute=ob.minute;
    second=ob.second;
}
public String printS( )
{
    String output = " ";
    output=output+ " "+( ( hour == 0 || hour == 12 ) ? 12 :
    hour % 12 ) + ":" + minute+ ":" +second+ ":" +
    ( hour < 12 ? "AM" : "PM" );
    return output ;
}
public String printU( )
{
    String output = " ";
    output=output+ " "+hour+ " "+minute+ " "+second ;
    return output ;
}
} //end class TimeGUI

```

```

public class TimeTestNewGUI {
public static void main(String args[ ]) {
    int h1,m1,s1;// 8,15,40
    String javaString = JOptionPane.showInputDialog(null,
    "Enter a h1 hour ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    h1 = Integer.parseInt(javaString) ;
    javaString = JOptionPane.showInputDialog(null,
    "Enter a m1 minute ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);

```

```

    m1 = Integer.parseInt(javaString) ;
    javaString = JOptionPane.showInputDialog(null,
    "Enter a s1 second ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    s1 = Integer.parseInt(javaString) ;
    //-----
int h11,m11,s11;// 18,45,35
    javaString = JOptionPane.showInputDialog(null,
    "Enter a h11 hour ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    h11 = Integer.parseInt(javaString) ;
    javaString = JOptionPane.showInputDialog(null,
    "Enter a m11 minute ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    m11 = Integer.parseInt(javaString) ;
    javaString = JOptionPane.showInputDialog(null,
    "Enter a s11 second ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    s11 = Integer.parseInt(javaString) ;
    //-----
int value;
    javaString = JOptionPane.showInputDialog(null,
    "Enter a value ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    value = Integer.parseInt(javaString) ;
    //-----
    int h,ms;
    javaString = JOptionPane.showInputDialog(null,
    "Enter a value hour ", "My time Example",
    JOptionPane.QUESTION_MESSAGE);
    h=Integer.parseInt(javaString) ;

```



```

javaString = JOptionPane.showInputDialog(null,
"Enter a value minute and second ", "My time Example",
JOptionPane.QUESTION_MESSAGE);
ms=Integer.parseInt(javaString) ;
//-----
TimeGUI myTime0 = new TimeGUI( );
TimeGUI myTime1 = new TimeGUI( h1,m1,s1);
TimeGUI myTime2 = new TimeGUI( h11,m11,s11);
TimeGUI myTime3 = new TimeGUI(myTime1 );
TimeGUI myTime4 = new TimeGUI(value );
TimeGUI myTime5 = new TimeGUI(h,ms );
// display initial Time
//-----
String ouput =" initial Time U \n" + myTime0.printU()+ "\n";
    ouput = ouput +"initial Time S \n"+myTime0.printS( )+"\n";
    ouput = ouput +"first Time U \n"+myTime1.printU( )+"\n";
    ouput = ouput +"first Time S \n"+ myTime1.printS( )+"\n";
    ouput = ouput +"second Time U\n"+ myTime2.printU( )+"\n";
    ouput = ouput +"second Time S \n"+ myTime2.printS( )+"\n";
    ouput = ouput +"third Time U \n"+ myTime3.printU( )+"\n";
    ouput = ouput +"third Time S \n"+ myTime3.printS( )+"\n";
    ouput = ouput +"fuorth Time U \n"+ myTime4.printU( )+"\n";
    ouput = ouput +"fuorth Time S \n"+ myTime4.printS( )+"\n";
    ouput = ouput +"fifth Time U \n"+ myTime5.printU( )+"\n";
    ouput = ouput +"fifth Time S \n"+ myTime5.printS( ) +"\n";
JOptionPane.showMessageDialog(null, ouput,
    "My time Class ", JOptionPane.ERROR_MESSAGE );
} //end main
} // End Class TimeTest4

```

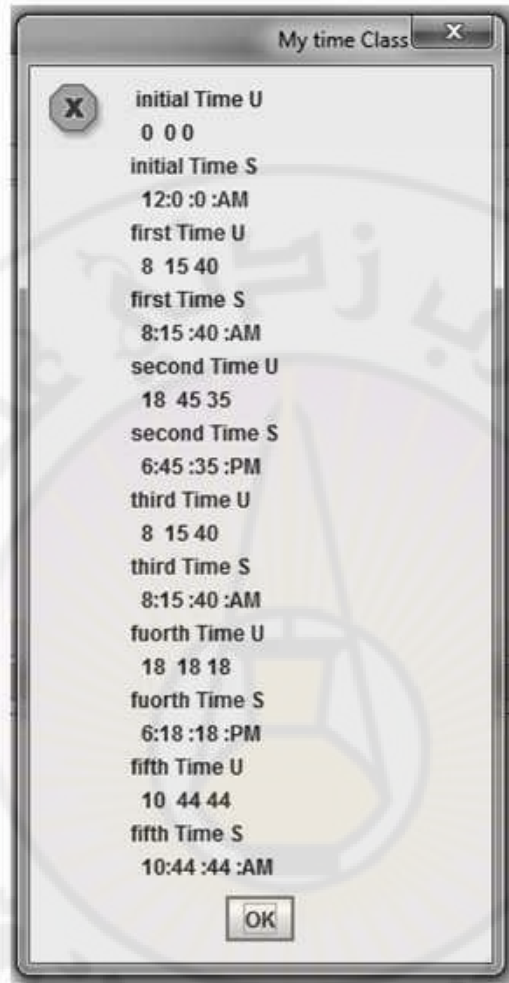
يتم إدخال المعطيات عن طريق والواجهات التالية :



The image displays eight dialog boxes, each titled "My time Example" with a close button (X) in the top right corner. Each dialog box contains a question mark icon, a text prompt, an input field, and "OK" and "Cancel" buttons. The prompts and values are as follows:

- Top-left: "Enter a s1 second" with value "40".
- Top-right: "Enter a m1 minute" with value "15".
- Second row, left: "Enter a h1 hour" with value "8".
- Second row, right: "Enter a s11 second" with value "35".
- Third row, left: "Enter a m11 minute" with value "45".
- Third row, right: "Enter a h11 hour" with value "18".
- Bottom-left: "Enter a value minute and second" with value "44".
- Bottom-right: "Enter a value hour" with value "10".

وإظهار النتيجة على الواجهة التالية :



٥-٨- بناء المكس Stack

المثال (5 - 6) التالي يظهر قوة الأغراض ، حيث ينشئ هذا البرنامج صفاً يدعى stack ، وهو مثال لتخزين الرموز ، وهذا الصف يحتوي على متحولين خاصين هما stack و tos ، وتقوم المصفوفة stack بالاحتفاظ بالرموز المدفوعة إلى المكس ويحتوي المتحول tos على دليل قمة المكس . هذا الصف يحتوي على ثلاثة توابع عامة هي :

- التابع () init لتهيئة المكس .
 - التابع () push لدفع قيمة إلى المكس .
 - التابع () pop لسحب قيمة من المكس .
- يقوم التابع () main بإنشاء مكسيتين mystack1 و mystack2 ، ثم يدفع عشر أعداد إلى المكس . إن المكسيتين mystack1 و mystack2 منفصلين عن بعضهما بحيث لا تؤثر القيم المدفوعة إلى mystack1 على القيم المدفوعة إلى mystack2 ، أي أن كلاً من الغرضين يملك نسخة خاصة من المتحولين stck و tos . يُعد هذا المبدأ أساساً لفهم الأغراض ، وعلى الرغم من أن جميع أغراض هذا الصف تشترك بالتوابع الأعضاء نفسها ، إلا أن كل غرض ينشئ ويحتفظ بأعضاء معطياته الخاصة .

مثال (5-6)

// This class defines an integer stack that can hold 10 values.

```
public class Stack {
private int stck[ ] = new int[10];
private int tos;
// Initialize top-of-stack
Stack( ) { tos = -1 ;}
// Push an item onto the stack
void push(int item) {
    if(tos==9)
        System.out.println("Stack is full.");
    else
        stck[++tos] = item;
}
// Pop an item from the stack
```

```

int pop( ) {
if(tos < 0) {
    System.out.println("Stack underflow.");
    return 0;
}
else
return stck[tos--];
}
} //End Class stack

```

```

public class TestStack {
public static void main(String args[ ] ) {
    Stack mystack1 = new Stack( );
    Stack mystack2 = new Stack( );
    // push some numbers onto the stack
    for(int i=0; i<10; i++)
        mystack1.push(i);
    for(int i=10; i<20; i++)
        mystack2.push(i);
    // pop those numbers off the stack
    System.out.println("Stack in mystack1:");
    for(int i=0; i<10; i++)
        System.out.println( mystack1.pop( ) );
    System.out.println("Stack in mystack2:");
    for(int i=0; i<10; i++)
        System.out.println( mystack2.pop( ) );
    }//end main
} // end class TestStack

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٥-٩- المزيد حول إنشاء الصف

بعد أن تعرفنا على كيفية القيام بإنشاء أغراض من الصف وكيفية استخدامها يتم تنظيف الأغراض (تحرير فضاء الذاكرة) من خلال جامع النفايات. يقوم جامع النفايات Garbage Collector بتحرير فضاء الذاكرة المستخدم من قبل الأغراض والتي أصبحت غير مطلوبة وذلك خلال فترات زمنية منتظمة. ويقوم جامع النفايات بفحص مناطق الذاكرة الديناميكية بحثاً عن الأغراض، ويُعلن الأغراض التي لها مرجع referenced بعد أن تُكتشف جميع التوابع الممكنة والمؤدية إلى الأغراض، فإن الأغراض غير المعنونة تصبح معروفة وتُجمع ليتم التخلص منها.

□ الشكل العام للصف في لغة Java هو كما يلي :

```
[ public ][ abstract ][ final ] class Name of Class [ extends
Super ] [ implement interface ]
{
    [ Class Body ]
}
```

البنود المطلوبة هي الكلمة المحجوزة **class** ، واسم الصف *Name of Class* والقوسين { } . حيث جميع البنود الأخرى اختيارية ، وقد تم وضعها ضمن قوسين [] ، أما الكلمات المكتوبة بخط مائل تشير إلى محدد مثل اسم صف أو واجهة ، وإذا لم نقم بالتصريح عن البنود الاختيارية **public** و **abstract** و **final** ، فإن مترجم Java يفترض افتراضات معينة مثل : **nonpublic** , **nonabstract** ,

سنقوم بتعريف أولي لبنود الصف :

١. **المحدد public** : إذا لم نقم باستخدام المحدد **public** فإنه بشكل افتراضي يمكن استخدام الصف فقط من قبل صفوف أخرى موجودة في نفس الحزمة .

٢. **المحدد abstract** : يستخدم هذا المحدد لتعريف صف لا يمكن إنشاء أغراض منه .

٣. **المحدد final** : يستخدم هذا المحدد لتعريف صف لا يمكن أن يورث ، وتُعدّ المعطيات النهائية والمتحولات النهائية ثوابت ولا يمكن تجاوزالتوابع النهائية في الصفوف المشتقة .

٤. **الكلمة class** : تُخبر المترجم بأن هذا صف واسمه *Name* .

٥. **المحدد extends** : يحدد أن الصف *Super* هو صف أساسي Superclass وأن الصف *subclass* هو صف مشتق .

٦. المحدد **implement** : للإشارة إلى أن الصف يحقق الواجهة
. Interface

٥-١٠- الكلمة المحجوزة **this**

حتى نشير إلى عضو ما من أعضاء الغرض نقوم بذكر اسم الغرض متبوعاً بنقطة ثم اسم العضو المراد ذكره :

```
int getX( ) { return x ; }  
ob.getX( )
```

ويستدعي الغرض ob التابع () **getX** فينتقل التنفيذ إلى التابع () **getX** الذي سيجد عضو المعطيات x دون اسم الغرض ، عندها يكون الغرض ob هو الذي قام باستدعاء التابع () **getX** وهو مالك هذا العضو . لكن في بعض الأحيان نحتاج لأن نميز بين عضو معطيات (متحولات أعضاء الصف) ووسيط لتابع لهما الاسم نفسه وخاصة التابع الباني وتابع تعديل المعطيات ، كما هو موضح في المثال التالي :

```
public class Point {  
    int x , y;  
  
    public Point(int x, int y) {  
        x = x;  
        y = y;  
    }  
  
    . . . . .  
}
```

}//End Class Point

في التابع الباني أو تابع تعديل المعطيات هل أسند عضو المعطيات إلى وسيط التابع أم أسند الوسيط إلى عضو المعطيات ؟

لكي نحل هذا الغموض يمكننا استخدام الكلمة المحجوزة `this` والتي تشير إلى الغرض الذي قام بالاستدعاء ، وسنعيد كتابة المثال السابق كما يلي :

```
public class Point {
    int x , y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    . . . . .
} //End Class Point
```

بهذا الشكل نستطيع أن نقول بأن أعضاء المعطيات (وهي المسبوقة بـ `this`) هي التي تم إسناد القيم إليها . بشكل عام يفضل المبرمجون القيام باستخدام الكلمة المحجوزة `this` سواء أكان هناك تماثل بين أسماء الأعضاء وأسماء الوسطاء أو لم يكن ، لأن ذلك يجعل النص البرمجي أكثر وضوحاً . ويبين المثال (5 - 7) التالي الصف `SimpleTime` كيفية استخدام الكلمة المحجوزة `this` .

مثال (5-7)

```
// class SimpleTime demonstrates the "this" reference
public class SimpleTime {
    private int hour;
    private int minute;
    private int second;
    /* if the constructor uses parameter names identical to instance
    variable names the "this" reference is required to distinguish
    between names .*/
    SimpleTime(int hour ,int minute ,int second ){
```

```

    this.hour = hour;    // set "this" object's hour
    this.minute = minute; // set "this" object's minute
    this.second = second; // set "this" object's second
} // end SimpleTime constructor
public void printu ( ) {
    System.out.printf("%d %d %d \n ", this.hour,this.minute ,
    this.second) ;
}
} //end class SimpleTime

public class SimpleTimeTest {
public static void main(String args[ ] ) {
SimpleTime  myTime1 = new SimpleTime(22,30,40);
SimpleTime  myTime2 = new SimpleTime( 4,6,13);
// display first Time
System.out.println("first SimpleTime u ");
    myTime1.printu( );
// display second Time
System.out.println("second SimpleTime u ");
    myTime2.printu( );
} // end main
} // end class SimpleTimeTest

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```

first SimpleTime u
22 30 40
second SimpleTime u
4 6 13
Process completed.

```


يحتوي بعض الأحيان الصف بوان كثيرة ، وكل بان يسمح للمستدعي أن يعطي قيمة ابتدائية لمتحولات مثيلة instance مختلفة للغرض الجديد . فعلى سبيل المثال الصف Time له ثلاثة بوان هي :

Time () ;

Time (int hour , int minute) ;

Time (int hour , int minute , int second) ;

الباني الأول لا يحتوي على وسطاء وهو بان افتراضي (وإذا لم نكتب في الصف أي بان فإن المترجم يولد بان افتراضي عند إنشاء الأغراض) لا يسمح بإعطاء أي قيم ابتدائية لأي شيء .

البانيان الآخران يسمحان للمستدعي أن يضبط قيمة ابتدائية للوقت ، ومع ذلك فإن الوقت يجب أن تتم تهيئته وذلك لجميع المتحولات المثيلة .

غالباً يقوم بان واحد بكامل هذا العمل وتقوم البواني الأخرى باستدعاء ذلك الباني وتزويده (إعطائه) إما بالقيم من خلال وسطائها (وسطاء الباني الممررة) أو قيم افتراضية . كمثال ، لنفرض أن hour , minute , second هي متحولات مثيلة يراد تهيئتها :

Time ()

```
{  
    this ( 0 , 0 , 0 ) ;  
}
```

Time (int hour)

```
{  
    this ( hour , 0 , 0 ) ;  
}
```

Time (int hour , int minute)

```
{  
    this ( hour , minute , 0 ) ;  
}
```

Time (**int** hour , **int** minute , **int** second)

```
{  
    this (hour , minute , second ) ;  
}
```

نلاحظ أن كلاً من الباني الأول والثاني والثالث قام باستدعاء الباني الرابع ، حيث قام كل من الباني الأول والثاني والثالث باستدعاء الباني الرابع مع تمرير الوسطاء اللازمة ، وبهذا الشكل نتجنب إعادة كتابة نص برمجي متشابه عدة مرات .

إن استخدام البواني هو الأفضل للأسباب التالية :

- جميع عمليات التهيئة تكون في مكان واحد .
- القيم الافتراضية تعالج بشكل صحيح .
- تُعدّ البواني مفهومة على نطاق واسع من مبرمجي Java .

يوضح المثال (5 - 8) التالي التحميل الزائد للبواني مع الكلمة المحجوزة **this** لاستدعاء بانٍ من خلال بانٍ آخر .

مثال (5-8)

// Time2 class declaration with overloaded constructors

```
Public class Time2 {
```

```
    private int hour , minute , second ;
```

// Time2 no-argument constructor: initializes each instance variable

// to zero; ensures that Time2 objects start in a consistent //state

```
public Time2( ) {
```

```
    this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
```

```
} // end Time2 no-argument constructor
```

// Time2 constructor: hour supplied, minute and second defaulted to 0

```
    public Time2( int h ){
```

```
        this ( h, 0, 0 ); // invoke Time2 constructor with three arguments
```

```
    } // end Time2 one-argument constructor
```

// Time2 constructor: hour and minute supplied, second defaulted to 0

```

public Time2( int h, int m ) {
    this( h, m, 0 ); // invoke Time2 constructor with three arguments
} // end Time2 two-argument constructor

public Time2(int h,int m,int s ){
    SetTime ( h , m , s );
} // end SimpleTime constructor

public void prints( ) {
    System.out.printf( "%d:%02d:%02d %s\n",
        ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),
        minute, second, ( hour < 12 ? "AM" : "PM" ) );
}

public void printu ( ) {
    System.out.printf("%d %d %d \n ", hour ,minute ,second) ;
}

public void SetTime ( int h ,int m ,int s ) {
    hour=(h>=0 && h<24)? h:0;
    minute=(m>=0 && m<60)? m:0;
    second=(s>=0 && s<60)? s:0;
}

// Time2 constructor: another Time2 object supplied
public Time2 ( Time2 time )
{
    // invoke Time2 three-argument constructor
    this ( time.hour, time.minute, time.second );
}

// end Time2 constructor with a Time2 object argument
} //end class time2

public class Time2Test {
public static void main(String args[ ] ) {
    Time2 t1 = new Time2( ); // 00:00:00

```

```

Time2 t2 = new Time2( 2 );           // 02:00:00
Time2 t3 = new Time2( 21, 34 );      // 21:34:00
Time2 t4 = new Time2( 12, 25, 42 );  // 12:25:42
Time2 t5 = new Time2( 27, 74, 99 );  // 00:00:00
Time2 t6 = new Time2( t4 );          // 12:25:42

// display first Time
System.out.println("first Time u t1 ");
t1.printu( );
System.out.println("first Time s t1 ");
t1.prints( );

// display second Time
System.out.println("second Time u t2 ");
t2.printu( );
System.out.println("second Time s t2 ");
t2.prints( );

// display 3'ed Time
System.out.println("second Time u t3 ");
t3.printu( );
System.out.println("second Time s t3 ");
t3.prints( );

// display 4'th Time
System.out.println("second Time u t4 ");
t4.printu( );
System.out.println("second Time s t4 ");
t4.prints( );

// display 5'th Time
System.out.println("second Time u t5 ");
t5.printu( );
System.out.println("second Time s t5 ");
t5.prints( );

// display 6'th Time

```



```

System.out.println("second Time u t6 ") ;
t6.printu( );
System.out.println("second Time s t6 ") ;
t6.prints( );
} // end main
} // end class Time2est

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

RESULT	
second Time u t4 12: 5: 42	first Time u t1 0 : 0 : 0
second Time s t4 12:25:42 PM	first Time s t1 12:00:00 AM
second Time u t5 0: 0 : 0	second Time u t2 2: 0 : 0
second Time s t5 12:00:00 AM	second Time s t2 2:00:00 AM
second Time u t6 12: 25 : 42	second Time u t3 21 : 34 : 0
second Time s t6 12:25:42 PM	second Time s t3 9:34:00 PM
Process completed	

١١-٥ - تمرير الأغراض إلى التوابع

يمكن تمرير الأغراض إلى التوابع بنفس أسلوب تمرير باقي أنواع المعطيات الأخرى ، علينا أن نصرح عن وسائط التابع على أنها من نوع صف ، ثم نستخدم أغراض ذلك الصف كمتحولات عند استدعاء التابع ، حيث تمرر الأغراض إلى التوابع بالقيمة ، يوضح المثال (5 - 9) التالي تمرير الأغراض للتوابع .

// Objects may be passed to methods.

public class Test {

مثال (5-9)


```

private int a, b;
public Test ( int i, int j ) {
    a = i;    b = j;
}
// return true if o is equal to the invoking object
public boolean equals (Test ob) {
    if (ob.a == a && ob.b == b)
        return true;
    else
        return false;
}
} // End class Test

public class PassOb {
    public static void main(String args[ ] ) {
        Test obj1 = new Test(100, 22);
        Test obj2 = new Test(100, 22);
        Test obj3 = new Test(-1, -1);
        System.out.println("obj1 == obj2: " + obj1.equals(obj2));
        System.out.println("obj1 == obj3: " + obj1.equals(obj3));
    }
} // End class passOb

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

obj1 == obj2: true

obj1 == obj3: false

Process completed.

مسائل عامة محلولة

- ١ - اكتب برنامجاً بلغة جافا يحتوي على ثلاثة صفوف أساسية هي :
- الصف الأساسي الأول يدعى Building بحيث يخزن عدد الطوابق Floors في البناء وعدد الغرف Rooms وهي من نمط int ومساحة الغرف Sqmeter بالمتر المربع وهي من نمط double وجميعها متغيرات خاصة . ويحتوي الصف على بان افتراضي وبان عادي وتابع طباعة .
 - الصف الأساسي الثاني اسمه House ، ويخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي من نمط int وهي متغيرات خاصة. يحتوي الصف على بان افتراضي وبان عادي وتابع طباعة .
 - الصف الأساسي الثالث اسمه Office يخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة . ويحتوي الصف على بان افتراضي وبان عادي وتابع طباعة .
 - أنشئ صفاً آخر يتضمن التابع main اسمه BuildingTest لإنشاء أغراضاً للصفوف الثلاثة لاستدعاء التوابع اللازمة والحصول على النتائج .

البرنامج

```
public class Building {  
    private int Floors, Rooms;  
    private double Sqmeter;  
    public Building() { Floors=0;Rooms=0;Sqmeter=0;}  
    public Building ( int f,int r ,double m ) {  
        Floors =f;  
        Rooms=r;  
        Sqmeter=m;  
    }  
}
```

```

public void Printbuilding( ) {
    System.out.println(" Rooms is " + Rooms);
    System.out.println(" Floors is " + Floors );
    System.out.println(" Sqmeter is " + Sqmeter);
}
} //End Class
public class House {
    private int Bedrooms , Bathrooms;
public House ( ) {
        Bedrooms = 0 ; Bathrooms = 0 ; }
public House( int br, int bth) {
    Bedrooms = br; Bathrooms = bth;
}
public void PrintHouse( ) {
    System.out.println(" bedrooms is " + Bedrooms);
    System.out.println(" bathrooms " + Bathrooms);
}
} //End class House
public class Office {
    private int Phones , Extinguishers;
public Office( int p, int ext) {
    Phones = p; Extinguishers = ext;
}
public void PrintOffice( ) {
    System.out.println(" Telephones is " + Phones);
    System.out.println(" Fire Extinguishers is " + Extinguishers);
}
} //End Class Office
public class BuildingTest {
public static void main(String args[ ] ) {
    Building myBuilding1 = new Building(10, 200, 1500);

```

```

myBuilding1.Printbuilding( );
House myHouse1 = new House(100, 150);
myHouse1.PrintHouse( );
Office myOffice1 = new Office(150, 250);
myOffice1.PrintOffice( );
} //End main
} // End class SumTest

```

٢- اكتب برنامجاً بلغة Java للصف StudentMark حيث الصف studentMark يقوم بإدخال ثلاثة أسماء طلاب name1, name2, name3 كثلاثة متحولات من النمط String ، وإدخال ثلاث درجات m1,m2,m3 كثلاثة متحولات من النمط int . يحتوي الصف على باني عادي وعلى تابع طباعة max() لحساب وطباعة أعلى درجة وطباعة اسم الطالب الذي حصل عليها ، وعلى تابع طباعة آخر min() لحساب وطباعة أدنى درجة وطباعة اسم الطالب الذي حصل عليها .

أنشئ صفاً اختبارياً يتضمن التابع main اسمه Test لإنشاء أغراض واستدعاء التتابع اللازمة والحصول على النتائج .

البرنامج :

```

public class StudentMark {
private int m1,m2,m3;
private String name1 ,name2,name3;
StudentMark (String n1 ,String n2,String n3,int x, int y,int z) {
m1 = x; m2 = y; m3 = z ;
name1 = n1 ; name2 = n2 ; name3 = n3 ;
}
public void max( ) {
int max1 = m1 ;
String max2 = name1 ;

```

```

if ( m2 > max1 ){
    max1 = m2 ;
    max2 = name2 ;
}
else if ( m3 > max1 ){
    max1 = m3 ;
    max2 = name3 ;
}
System.out.println(max1 + max2);
public void min( ) {
    int min1 = m1 ;
    String min2 = name1 ;
    if ( m2 < min1 ){
        min1 = m2 ;
        min2 = name2 ;
    }
    else if ( m3 < min1 ){
        min1 = m3 ;
        min2 = name3 ;
    }
    System.out.println(min1 + min2);
}
}
public class Test {
    public static void main(String args[ ]) {
        StudentMark Ob = new StudentMark (" Younes ", "Akram "
                                           , " Waseem " ,80, 65,75);

        Ob.max();
        Ob.min();
    } //End main
} // End class Test

```


٣- أكتب برنامجاً بلغة Java يتضمن ما يلي :

□ صفاً يمثل درجات الحرارة وليكن اسمه Temp يحتوي على وسيطين minT

maxT , من النوع int ويحتوي على التوابع (الطرائق) العامة التالية:

- بانٍ افتراضي يسند القيمة صفر للمعطيات الخاصة.

- بانٍ عادي ، وسيطه مصفوفة ، مهمته إيجاد العنصر الأكبر لمصفوفة من الأعداد ومن ثم إيجاد العنصر الأصغر لنفس المصفوفة ، بعد ذلك يُسند النتائج للمعطيات الخاصة maxT و minT على الترتيب .

- تابع عضو لطباعة قيم المعطيات الخاصة وليكن اسمه print .

- تابع عضو وليكن اسمه diff ، وسيطه غرض من الصف Temp ويُعيد قيمة، مهمته حساب القيمة المطلقة للفرق بين قيمتي العنصرين الأكبر والأصغر .

- تابع عضو من النوع boolean للمقارنة بين غرضين من الصف Temp .

□ صفاً للاختبار يحوي على التابع الرئيسي main ، من أجل إدخال درجات

الحرارة لشهري أيار وحزيران (كل شهر 30 يوماً) وتخزينهما في المصفوفتين

TM[30] و TJ[30] بعد ذلك صرح عن الأغراض اللازمة من الصف

Temp من أجل حساب و طباعة درجتَي الحرارة العظمى والصغرى لشهري

أيار و حزيران وكذلك حساب وطباعة الفرق بينهما ونتيجة المقارنة.

البرنامج :

```
import java.util.Scanner;
public class Temp {
int maxT,minT;
public Temp( ){maxT=0;minT=0;}
public Temp(int x[ ],int n) {
int i,mx=x[0],mn=x[0];
for(i=0;i<n;i++) {
if(x[i]>mx)
```

```

        mx=x[i];
    }
    for(i=0;i<n;i++) {
        if(x[i]<mn)
            mn=x[i];
    }
    maxT=mx;
    minT=mn;
}
public void print( ) {
    System.out.println("\nmaxT= "+maxT);
    System.out.println("\nminT= "+minT);
}
boolean equals (Temp ob) {
    if (ob.maxT == maxT && ob.minT == minT)
        return true;
    else
        return false;
}
public double Fr(Temp ob ) {
    double d1,d;
    d1=(double) (ob.maxT-ob.minT);
    d=Math.abs(d1);
    return d;
}
} // End class
// test class
public class FindingMaxMinTemp {
public static void main(String args[ ]) {
    Scanner input=new Scanner(System.in);
    int TA[ ],TJ[ ];

```

```

TA=new int[30];
TJ=new int[30];
System.out.println("Enter the Tempretures of may \n");
for(int i=0;i<30;i++)
    TA[i]=input.nextInt( );
System.out.println("Enter the Tempreture of june \n");
for(int i=0;i<30;i++)
    TJ[i]=input.nextInt( );
Temp ob =new Temp( );
Temp ob1 =new Temp(TA,30);
Temp ob2 =new Temp(TJ,30);
ob1.print( );
ob2.print( );
System.out.println("ob1 == ob2: " +ob1.equals(ob2));
System.out.println("\n The difference is "+ob1.Fr(ob1));
System.out.println("\n The difference is "+ob2.Fr(ob2));
} //End main
} //End Class

```

- ٤- أكتب برنامجاً بلغة Java يتضمن ما يلي :
- صفاً اسمه Numbers يحتوي على وسيطين sd , se كأعضاء خاصة ويحتوي على التوابع (الطرائق) العامة التالية:
- باني افتراضي يسند القيمة صفر للمعطيات الخاصة.
 - باني عادي وسيطه مصفوفة ويقوم بحساب مجموع الأعداد الفردية ومجموع الأعداد الزوجية لمصفوفة من الأعداد مكونة من عشرة عناصر ومن ثم يُسند النتائج للمعطيات الخاصة sd و se على الترتيب.
 - تابع عضو عام لطباعة قيم المعطيات الخاصة وليكن اسمه print ومن النوع void.

- تابع عضو عام وليكن اسمه fr ، وسيطه غرض من الصف Numbers ويُعيد قيمة، مهمته حساب القيمة المطلقة للفرق بين مجموعي الأعداد الفردية والأعداد الزوجية المُدخلة.

□ صفاً للاختبار وليكن اسمه SumEvenOddNumbers يحوي على التابع الرئيسي main ، يقوم بإدخال 10 أعداد صحيحة وتخزينها في مصفوفة أحادية البعد x[10] ومن ثم طباعة عناصر المصفوفة . بعد ذلك يتم إنشاء الأغراض المناسبة من الصف Numbers من أجل استدعاء التوابع اللازمة لحساب وطباعة النتائج المطلوبة (حساب المجاميع والفرق بينهما).

البرنامج :

```
//This program computes the sum of even and odd numbers
import java.util.Scanner;
public class SumEvenOddNumbers {
public static void main(String args[ ] ) {
Scanner input=new Scanner(System.in);
int x[ ];
int i;
x=new int[10];
System.out.println("Enter the numbers \n");
for( i=0;i<10;i++)
x[i]=0;
for( i=0;i<10;i++)
x[i]=input.nextInt();
for( i=0;i<10;i++)
System.out.print(x[i]+" ");
Numbers ob =new Numbers( );
Numbers ob1 =new Numbers(x);
ob1.print( );
System.out.println("\n The difference is "+ob1.fr(ob1));
```

```
}//End main  
} //End Class
```

```
public class Numbers{  
    private int sd,se;  
    public Numbers( ){sd=0;se=0;}  
public Numbers(int x[ ]) {  
    int i,s1=0,s2=0;  
    for(i=0;i<10;i++) {  
        if(x[i]%2==0)  
            s1=s1+x[i];  
        else  
            s2=s2+x[i];  
    }  
    sd=s1;  
    se=s2;  
}  
public void print( ) {  
    System.out.println("\nsd= "+sd);  
    System.out.println("\nse= "+se);  
}  
public double fr(Numbers ob ) {  
    double d1,d;  
    d1=(double) (ob.se-ob.sd);  
    d=Math.abs(d1);  
    return d;  
}  
} // End class
```


مسائل عامة

١- اكتب برنامجاً بلغة Java للصف Date يحتوي على ثلاثة متحولات خاصة ويحتوي على باني افتراضي وباني عادي وتابع طباعة () print() ويقوم بطباعة التاريخ بالشكل " 15 may 2006 " . بحيث يتم تعيين قيمة ابتدائية تعتبر (01 January 2006) ، وعند الإدخال يجب أن يطبع (0) إذا كانت day أصغر أو أكبر من (31) ، وأن يطبع (0) إذا كانت year أصغر من 2006 ، حيث المتحول day و المتحول year من نوع int ، أما المتحول month من نوع String .

أنشئ صفاً آخر يتضمن التابع main اسمه DateTest لإنشاء أغراضاً لاستدعاء التوابع اللازمة والحصول على النتائج .

٢- اكتب برنامجاً بلغة Java للصف Student حيث الصف student يقوم بإدخال الاسم الثلاثي first, mid, last كثلاثة متحولات من النوع String وطباعتها . ثم إدخال ثلاث مواد دراسية course1, course2 , course3 من النوع int وحساب المعدل الوسطي للطالب .

أنشئ صفاً آخر يتضمن التابع main اسمه StudentTest لإنشاء أغراضاً لاستدعاء التوابع اللازمة والحصول على النتائج .

٣- اكتب برنامجاً بلغة Java يحتوي على صفاً اسمه myclass يحتوي على وسيطين خاصين sumEven ، sumOdd من النمط int ، ويحتوي الصف على الأعضاء العامة التالية :

- بان افتراضي يسند القيمة صفر للمعطيات الخاصة .

- بان عادي وسيطه مصفوفة ويقوم بحساب مجموع الأعداد الفردية والزوجية لمصفوفة من الأعداد الصحيحة مكونة من 10 عناصر، ثم يسند النتائج للمعطيات الخاصة sumEven ، sumOdd على الترتيب.
- تابع عضو لطباعة قيم المعطيات الخاصة اسمه print .

أنشئ صفاً آخر يتضمن التابع main اسمه myclassTest ، ثم أدخل في التابع main عشرة أعداد صحيحة وخزنها في مصفوفة [10] x ، ثم صرح عن غرض ob من الصف myclass لاستدعاء التوابع اللازمة لحساب وطباعة النتائج .

- =====
- ٤- اكتب برنامجاً يحتوي على صف اسمه Myclass1 يحتوي على ثلاثة متحولات n,m,x من النمط int ، يحتوي الصف على بان عادي لتمرير المعطيات إلى الأغراض عند إنشائها ، وعلى تابع بوسيط واحد اسمه Factorial لحساب العامل ، ويعيد قيمة من النوع int ، وعلى تابع آخر اسمه SumFact من النوع void يقوم بحساب وطباعة المقدار التالي : $s = n! + \frac{m!}{x!}$
- أنشئ صفاً آخر يتضمن التابع main اسمه myclass1Test ثم صرح عن أغراض في التابع main() لاستدعاء توابع الأعضاء للصف Myclass1 وطباعة النتائج .

- =====
- ٥- اكتب برنامجاً يحتوي على صف اسمه myclass1 يحتوي على ثلاثة متحولات a,b,x من النمط double ، يحتوي الصف على بان عادي لتمرير المعطيات إلى الأغراض عند إنشائها ، وعلى تابع اسمه functY من النوع double يقوم بحساب المقدار التالي : $y = x^2 + 2x + 10$ ، وعلى تابع آخر اسمه functZ من النوع void يقوم بحساب وطباعة المقدار التالي :
- $$Z = (y - a)^2 + (y + b)^2$$

- أنشئ أنشئ صفاً آخر يتضمن التابع main اسمه myclass2Test ثم صرح عن أغراض في التابع () main لاستدعاء توابع الأعضاء للصف Myclass2 وطباعة النتائج .

=====

٦- أكتب برنامجاً بلغة الـ java يتضمن الصفوف التالية :

□ صفاً وليكن اسمه Students معطياته الخاصة (private) هي maxAvg (المعدل الأكبر) و ImaxAvg (الرقم التسلسلي للمعدل الأكبر) ويحتوي على التوابع (الطرائق) الأعضاء العامة التالية: بانيفتراضي ، باني عادي يُسند قيم للمعطيات الخاصة، توابع وصول للمعطيات الخاصة، تابع عضو وليكن اسمه ComputeAvg وسيطه مصفوفة ثنائية البعد مهمته حساب معدل كل طالب ومن ثم يقوم بطباعة كل المعدلات عن طريق واجهة GUI ، بالإضافة لذلك يقوم هذا التابع باستدعاء التابع الذي يقوم بإيجاد المعدل الأكبر، تابع عضو وليكن اسمه ComputeMax وسيطه مصفوفة أحادية البعد مهمته إيجاد المعدل الأكبر من بين المعدلات ومعرفة الرقم التسلسلي لهذا المعدل ومن ثم إسناد النتائج إلى المعطيات الخاصة.

□ صفاً للاختبار اسمه AverageStudents يحتوي على التابع الرئيسي main ، والذي بدوره يتضمن إدخال درجات 50 طالباً وطالبة في 10 مقررات من لوحة المفاتيح وتخزينها في مصفوفة ثنائية البعد Mark[50][10]، وإنشاء مايلزم من أغراض من الصف Students من أجل حساب معدلات الطلاب وإيجاد وطباعة المعدل الأكبر مع رقمه التسلسلي وحيث أن الطباعة يجب أن تكون باستخدام واجهة GUI.

ملاحظة مهمة: الإدخال عن طريق لوحة المفاتيح أما الإخراج يجب أن يكون باستخدام الواجهات GUI .

- ٧- أكتب برنامجاً بلغة java يتضمن الصفوف التالية :
- صفاً اسمه Engineer يحتوي على وسيط mark من النوع int كعضو خاص ويحتوي على التوابع (الطرائق) العامة التالية:
 - بان افتراضي يسند القيمة صفر للمعطيات الخاصة.
 - بان عادي وسيطه من نوع مصفوفة ويقوم بحساب عدد عناصر المصفوفة التي قيمها أكبر أو تساوي 60 ويسند النتيجة للمعطى الخاص.
 - تابع وصول للمعطى الخاص وليكن اسمه getmark.
 - تابع عضو عام وليكن اسمه counter وسيطه غرض من الصف Engineer و يقوم بإسناد القيمة المعادة من التابع getmark إلى متغير t ومن ثم يعيد القيمة t .
 - صفاً للاختبار وليكن اسمه GoodEngineer يحوي على التابع الرئيسي main ، والذي يتضمن إدخال درجات طالب ما في 15 مقرر وتخزينها في مصفوفة أحادية البعد m[15] . بعد ذلك يتم إنشاء الأغراض المناسبة من الصف Engineer من أجل استدعاء التوابع اللازمة لحساب وطباعة النتائج المطلوبة (حساب وطباعة عدد المقررات التي نجح فيها الطالب، على اعتبار أن درجة النجاح في المقرر هي 60).

ملاحظة مهمة: الإدخال الإخراج يجب أن يكون باستخدام الواجهات GUI .

- ٨- صفاً اسمه Students معطياته الخاصة (private) هي maxAvg (المعدل الأكبر) و ImaxAvg (الرقم التسلسلي للمعدل الأكبر)

ويحتوي على التوابع (الطرائق) الأعضاء العامة التالية: بان افتراضي ، باني عادي يُسند قيم للمعطيات الخاصة، توابع وصول للمعطيات الخاصة، تابع عضو اسمه ComputeAvg وسيطه مصفوفة ثنائية البعد مهمته حساب معدل كل طالب ومن ثم يقوم بطباعة كل المعدلات عن طريق واجهة GUI ، بالإضافة لذلك يقوم هذا التابع باستدعاء التابع الذي يقوم بإيجاد المعدل الأكبر، تابع عضو اسمه ComputeMax وسيطه مصفوفة أحادية البعد مهمته إيجاد المعدل الأكبر من بين المعدلات ومعرفة الرقم التسلسلي لهذا المعدل ومن ثم إسناد النتائج إلى المعطيات الخاصة.

صفاً للاختبار اسمه AverageStudents يحتوي على التابع الرئيسي main ، والذي بدوره يتضمن إدخال درجات 50 طالباً وطالبة في 10 مقررات من لوحة المفاتيح وتخزينها في مصفوفة ثنائية البعد [10][50]Mark، وإنشاء ما يلزم من أغراض من الصف Students من أجل حساب معدلات الطلاب وإيجاد وطباعة المعدل الأكبر مع رقمه التسلسلي وحيث أن الطباعة يجب أن تكون باستخدام واجهة GUI .

- ما هي وظيفة الكلمة المحجوزة this .

- ما هو الصف Object

٩- أكتب برنامجاً بلغة java يتضمن :

□ صفاً باسم StudentMark معطياته الخاصة name1,name2,name3 كمتحولات من النمط String والتي تمثل أسماء ثلاثة طلاب وأيضاً معطيات خاصة m1,m2,m3 من النمط int والتي تمثل درجات الطلاب في مقرر البرمجة (2) على الترتيب ، ويحتوي الصف على التوابع العامة التالية:

- بانٍ عادي ، و تابع عضو وليكن اسمه max من النوع String مهمته إيجاد أعلى درجة من بين الدرجات الثلاثة ومعرفة من حصل عيها.

- تابع عضو وليكن اسمه min من النوع String مهمته إيجاد أدنى درجة من بين الدرجات الثلاثة ومعرفة من حصل عليها.

□ صفاً للاختبار وليكن اسمه TestMark يتضمن إنشاء أغراضاً من أجل إدخال قيمٍ للمعطيات الخاصة عن طريق لوحة المفاتيح ومن ثم استدعاء التتابع اللازمة للحصول على النتائج المطلوبة وإظهارها عن طريق واجهة GUI .

ملاحظة: الإدخال عن طريق لوحة المفاتيح أما الإخراج يجب أن يكون باستخدام الواجهات GUI .



الفصل السادس

السلاسل والملفات في Java

Strings & Files In Java





السلاسل والملفات

١-٦ - مقدمة

يمكن تعريف السلاسل strings بأنها مجموعة أو سلسلة من الحروف والأرقام والرموز التي تُعامل معاملة الحروف . وهي تُشفّر في لغة Java بتقنية unicode أي تشفير 16bit ، وفيه يتم تمثيل الحرف الواحد في 16bit ، وتحتوي على جميع الحروف العربية والإنجليزية وغيرها . وقديماً كان يستخدم تشفير ASCII code ولكنه كان تشفيراً محدوداً بـ 8bit حيث يحتوي هذا النوع على الحروف الإنجليزية فقط بالإضافة إلى بعض الرموز . ويختلف تعريف المحارف في لغة Java عن تعريف الأرقام التي من نوع (int , float,....) .

٢-٦ - الصف String

يمكن أن نقول إنه في لغة Java يمكن عدّ كل رمز محاط بعلامتي اقتباس مزدوجة " " سلسلة string . فالسلسلة في لغة Java عبارة عن صف class ويمكن أن ننشئ منه أغراضاً objects ، ولإنشاء أغراض object من الصف String نستخدم الكلمة المحجوزة new .

ويُنمذج الصف java.lang.String سلسلة المحارف كسلسلة نصية ، ويملك الصف String أحد عشر بانياً وأكثر من أربعين تابعاً ، ومن أجل فحص المحارف المنفصلة في السلسلة ومقارنة السلاسل المحرفية ، والبحث عن السلاسل المحرفية الفرعية ، واستخراج السلاسل المحرفية الفرعية ، وصنع نسخ من السلسلة المحرفية ، ثم تحويل جميع المحارف الموجودة في السلسلة المحرفية إلى شكلها الكبير ، أو الصغير .

٦-٢-١ - بناء السلسلة المحرفية

يمكن إنشاء سلسلة محرفية باستخدام قيمة محرفية أو باستخدام مصفوفة من المحارف وذلك باستخدام الصيغة التالية :

```
String S1 = new String ( stringLiteral ) ;
```

إن المتحول stringLiteral هو سلسلة من المحارف المغلفة ضمن علامتي اقتباس مزدوجة ، فعلى سبيل المثال :

تُنشئ العبارة التالية الغرض المحرفي S1 من أجل السلسلة المحرفية التالية :

```
" welcome to java "
```

```
String S1 = new String ( " welcome to java " )
```

بما أن السلاسل المحرفية تُستخدم بشكل متكرر وبكثرة في البرمجة لذلك قدمت Java ترميزاً مختصراً من أجل إنشاء السلسلة المحرفية ، كما يلي :

```
String S1 = " welcome to java " ;
```

وتعرف السلسلة المحرفية التي تم إنشاؤها باستخدام الترميز المختصر بالنص القانوني (canonical) . وإذا تم إنشاء سلسلتين متطابقتين باستخدام الترميز المختصر فإن هاتين السلسلتين تتشاركان في النص القانوني نفسه . ونستطيع استخدام التابع intern الموجود في الصف String من أجل استعادة النص القانوني ، والذي يمثل النص الذي تم إنشاؤه باستخدام الترميز المختصر .

□ لنأخذ على سبيل المثال العبارات التالية :

```
String s = " welcome to java " ;
```

```
String s1 = new String ( " welcome to java " ) ;
```

```
String s2 = s1.intern() ; //Return a canonical string
```

```
String s3 = " welcome to java " ;
```

```
System.out.println( " s1== s is " + ( s1 == s ) ) ;
```

```
System.out.println( " s2== s is " + ( s2 == s ) ) ;
```

```
System.out.println( " s == s3 is " + ( s == s3 ) ) ;
```


يبين المثال (1-6) بعض العمليات على السلاسل .

مثال (1-6)

```
// String class constructors.  
public class String1 {  
  public static void main( String args[ ] ) {  
    // use String constructors  
    String s = " welcome to java " ;  
    String s1 = new String ( " welcome to java " ) ;  
    String s2 = s1.intern( ) ; //Return a canonical string  
    String s3 = " welcome to java " ;  
    // display strings  
    System.out.println( " s1== s is " + ( s1 == s ) ) ;  
    System.out.println( " s2== s is " + ( s2 == s ) ) ;  
    System.out.println( " s == s3 is " + ( s == s3 ) ) ;  
  } // end main  
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

S1 == s is false

S2 == s is true

S1 == s is true

Process completed.

في العبارات السابقة يشير كل من s و s2 و s3 إلى النص القانوني نفسه " welcome to java " لذلك فإن قيمة s == s2 و s == s3 هي true .
على أي حال فإن s == s1 هي false وذلك لأن s و s1 هما غرضان محرفيان مختلفان على الرغم من أن كليهما يملك المحتوى نفسه .

نستطيع أيضاً إنشاء سلسلة محرفية باستخدام مصفوفة من المحارف ، فعلى سبيل المثال ، يتم إنشاء السلسلة المحرفية Good day كما يلي :

```
char[ ] charArray = { 'G', 'o', 'o', 'd', 'd', 'a', 'y' };  
String s = String ( charArray );
```

وبين المثال (2-6) كيفية إنشاء وطباعة السلسلة المحرفية أو جزءاً منها .

مثال (2-6)

// String class constructors.

```
public class String2 {  
    public static void main( String args[ ] ) {  
        char charArray[ ] = { 'G', 'o', 'o', 'd', ' ', 'd', 'a', 'y' };  
        String s = new String( "hello" );
```

// use String constructors

```
String s1 = new String( );  
String s2 = new String( s );  
String s3 = new String( charArray );  
String s4 = new String( charArray, 5, 3);
```

// display strings

```
System.out.printf("s1 = %s\n s2 = %s\n s3 = %s\n s4 = %s\n",  
s1, s2, s3, s4 );
```

```
} // end main
```

```
} // end class
```

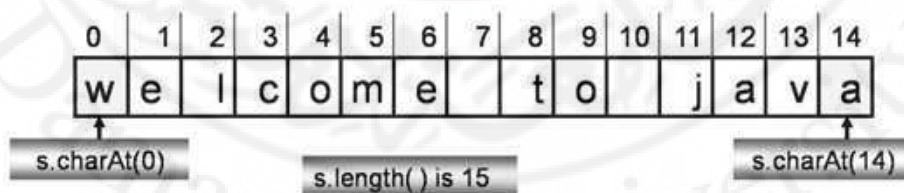
وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```
s1 =  
s2 = hello  
s3 = Good day  
s4 = day  
Process completed.
```

٦-٢-٢ - حساب طول السلسلة واستعادة المحارف المنفصلة

يمكن حساب طول السلسلة المحرفية من خلال استدعاء التابع `length()` ، حيث يعيد التابع `s.length()` طول السلسلة المحرفية `s`. ويُستخدم التابع `s.charAt(index)` لاستعادة محرف محدد من السلسلة المحرفية `s` بحيث يقع الدليل (`index`) بين القيمة 0 والقيمة `s.length() - 1` ، على سبيل المثال ، تعيد العبارة `s.charAt(0)` المحرف `w` وتعيد العبارة `s.charAt(14)` المحرف `a` كما هو مبين في الشكل (٦-١) التالي . ويبين المثال (٦-3) كيفية حساب طول السلسلة واستعادة محرف منها .



الشكل (٦-١) - يبين كيفية حساب طول السلسلة واستعادة المحارف

مثال (3-6)

```
/* This application demonstrates the length,  
charAt and getChars methods of the String class. */  
public class String3 {  
    public static void main( String args[ ] ) {  
        String s1 = "Welcome to Java";  
        char charArray[ ] = new char[ 7 ];  
        System.out.printf( "s1: %s", s1 );  
        // test length method  
        System.out.printf( "\nLength of s1: %d", s1.length() );  
        // loop through characters in s1 with charAt and display reversed  
        System.out.print( "\nThe string reversed is: " );  
        for ( int count = s1.length( ) - 1; count >= 0; count-- )  
            System.out.printf( "%s ", s1.charAt( count ) );  
        // copy characters from string into charArray  
        s1.getChars( 0, 7, charArray, 0 );  
        System.out.print( "\nThe character array is: " );  
        for ( char character : charArray )  
            System.out.print( character );  
        System.out.println( );  
    } // end main  
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```
s1: Welcome to Java  
Length of s1: 15  
The string reversed is: a v a J o t e m o c l e W  
The character array is: Welcome  
Process completed.
```


٦-٢-٣- لصق سلاسل المحارف

يُستخدم التابع () concat لدمج (لصق) سلسلتين محرفيتين ، فعلى سبيل المثال تقوم العبارة التالية بلصق السلسلتين s1 و s2 وإسنادها إلى السلسلة s3 .
String s3 = s1.concat(s2) ;
بما أن دمج السلسلتين يُستخدم بكثرة في البرمجة لذلك قدمت لغة Java طريقة مناسبة لتحقيق هذه الغاية ، وهي استخدام علامة الجمع (+) لدمج سلسلتين أو أكثر ، وتقوم العبارة التالية بجمع السلاسل message و "and" و "HTML" إلى سلسلة واحدة mystring :

String mystring = message + "and" + " HTML" ;
ويبين المثال (4-6) كيفية استخدام التابع () concat واستخدام علامة الجمع (+) لدمج سلسلتين أو أكثر .

مثال (4-6)

```
// String concat method.  
public class String4 {  
    public static void main( String args[ ] ) {  
        String s1 = new String( "Happy " );  
        String s2 = new String( "Birthday" );  
        System.out.printf( "s1 = %s\ns2 = %s\n\n", s1, s2 );  
        String s3 = s1.concat( s2 );  
        System.out.printf( "Result of s1.concat( s2 ) = %s\n", s3 );  
        System.out.printf( "s1 after concatenation = %s\n", s1 );  
        String message = " Welcome to Java " ;  
        String mystring = message + " and " + " HTML " ;  
        System.out.printf( "mystring = %s\n", mystring );  
    } // end main  
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

s1 = Happy

s2 = Birthday

Result of s1.concat(s2) = Happy Birthday

s1 after concatenation = Happy

mystring = Welcome to Java and HTML

Process completed.

٦-٢-٤ - استخراج سلسلة فرعية (جزئية)

إن الصف String هو صف ثابت ، لا يمكن تعديل قيمة السلسلة المحرفية المنشأة باستخدام هذا الصف . أي لا يمكن تغيير السلسلة "java" الموجودة في المتحول message (" Welcome to Java ") إلى السلسلة "HTML" ، عند الحاجة إلى تعديل السلسلة المحرفية الموجودة في المتحول message يجب علينا إسناد سلسلة محرفية جديدة إلى المتحول message كما في العبارة التالية :

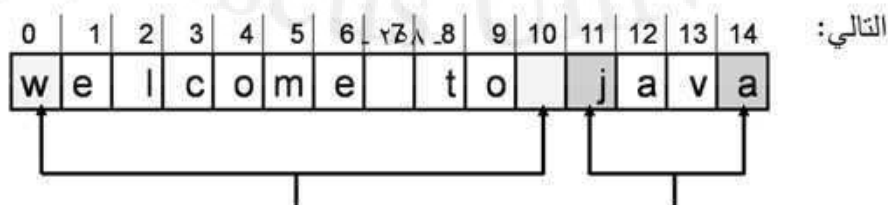
message = " welcome to HTML " ;

ويمكن استخدام التابع () substring كبديل للطريقة السابقة ، حيث يقوم هذا التابع باستخراج سلسلة جزئية من السلسلة الأصلية . ويوجد إصداران للتابع () substring :

الإصدار الأول :

public String substring (int beginIndex , int endIndex)

يعيد هذا التابع سلسلة محرفية جزئية ، تبدأ من الدليل beginIndex وتنتهي بالمحرف المقابل $endIndex - 1$ للسلسلة الأصلية . كما هو مبين بالشكل (٦-٢)



الشكل (٦-٢) - يبين كيفية استخراج سلسلة جزئية من السلسلة الأصلية

وبالتالي فإن طول النص الجزئي المستخرج هو :

$endIndex - beginIndex$

الإصدار الثاني :

public String substring (int beginIndex) ;

يعيد هذا التابع سلسلة محرفية جزئية تبدأ من المحرف المقابل للدليل *beginIndex* ويمتد إلى نهاية السلسلة المحرفية الأصلية ، كما هو مبين في الشكل السابق .

لنأخذ العبارة التالية كمثال على تعديل السلسلة باستخدام التابع *substring* :

String message = " welcome to Java ".substring(0,11)+ " HTML";
عندئذ يُسند للسلسلة *message* السلسلة التالية :

" welcome to HTML "

ويبين المثال (6-5) كيفية استخدام التابع (*substring*) .

مثال (6-5)

// String class substring methods.

public class string5 {

public static void main(**String** args[]) {

String message = "Welcome to Java";

// test substring methods

System.out.printf("Substring from index 11 to end is \"%s\"\\n",
message.substring(11));

```

System.out.printf( "%s \\\n",
    "Substring from index 3 up to, but not including 10 is",
    message.substring( 3, 10 ) );
message = message.substring(0,11)+ "HTML" ;
System.out.printf( "%s \\\n",
"Substring from index 0 up to, but not including 11 is", message );
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```

s1 = Happy Substring from index 11 to end is "Java"
Substring from index 3 up to, but not including 10 is "come to"
Substring from index 0 up to, but not including 11 is
    "Welcome to HTML"
Process completed.

```

٦-٢-٥ - مقارنة السلاسل

عندما نقارن بين محتوى سلسلتين محرفيتين نستخدم المعامل (==) لتحقيق هذه العملية وذلك كما يلي :

```

if( string1 == string2 )
    System.out.println( " string1 and string2 are the same object " );
else
    System.out.println( " string1 and string2 are different object " );

```

يفحص المعامل (==) فيما إذا كانت السلسلتان string1 و string2 تؤشران إلى الغرض نفسه أم لا ، وبالتالي لا نعرف فيما إذا كانت السلسلتان تحتويان على النص المحرفي نفسه عندما يؤشران إلى غرضين مختلفين لذلك لا يمكننا استخدام

المعامل (==) لمعرفة فيما إذا كانت السلسلتان المحرّفتان تحتويان على النص المحرّفي نفسه أم لا ، ولتحقيق ذلك يجب استخدام التابع (equals) الذي يفيد في مقارنة تساوي لمحتويات الأغراض . على سبيل المثال ، يمكن استخدام العبارة التالية للمقارنة بين سلسلتين :

```
if( string1.equals( string2) )
    System.out.println( " string1 and string2 are the same object " ) ;
else
    System.out.println( " string1 and string2 are different object " ) ;
```

يختلف المتحولان s1 و s2 عن بعضهما على الرغم من أن محتوياتهما متطابقة :

```
String s0 = " Java " ;
String s1 = " Welcome to " + s0 ;
String s2 = " Welcome to Java " ;
System.out.println" s1 == s2 is " + ( s1 ==s2 ) ) ;
System.out.println( " s1.equals(s2) is " +(s1.equals(s2)) ) ;
```

في هذه الحالة قيمة s1 == s2 هي false لأنهما كلاهما يؤشران إلى غرضين مختلفين . أما قيمة s1.equals(s2) هي true لأن الغرضين يملكان المحتويات نفسه . ويبين المثال (6-6) كيفية مقارنة السلاسل باستخدام المعامل (= =) وكيفية التحقق منها .

// String methods .

```
public class String6 {
    public static void main( String args[ ] ) {
        String s0 = "Java";
        String s1 = "Welcome to " + s0 ;
        String s2 = "Welcome to Java";
        System.out.println(" s1 == s2 is " + ( s1 ==s2 ) ) ;
        System.out.println( " s1.equals(s2) is " +(s1.equals(s2)) ) ;
```

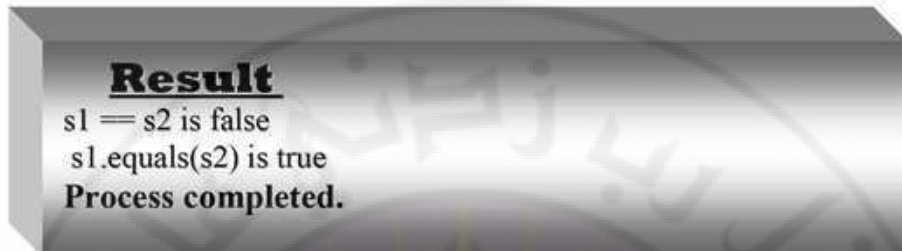
مثال (6-6)


```

    } // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



يمكن استخدام التابع (compareTo) من أجل المقارنة بين سلسلتين ، كما يلي :

```
s1.compareTo(s2) ;
```

لدينا السلاسل s1 و s2 و s3 :

```

s1 = "hello"
s2 = "goodbye"
s3 = "Happy Birthday"
s4 = "happy birthday"

```

يعيد هذا التابع القيمة 0 إذا كانت السلسلة s1 تساوي السلسلة s2 ، ويعيد قيمة أصغر من (0) إذا كانت السلسلة s1 أصغر من السلسلة s2 وفق الترتيب الأبجدي، ويعيد قيمة أكبر من (0) إذا كانت السلسلة s1 أكبر من السلسلة s2 وفق الترتيب الأبجدي . وتعتمد القيمة الفعلية المعادة من التابع compareTo على الفرق بين أول حرفين مختلفين من s1 و s2 ، وذلك بالمقارنة من اليسار إلى اليمين . وبيّن المثال (6-7) كيفية استخدام التوابع : equals و compareTo .

مثال (6-7)

```

/* String methods equals, equalsIgnoreCase, compareTo and
regionMatches. */

```

```

public class String6 {
    public static void main( String args[ ] ) {
        String s1 = new String( "hello" ); // s1 is a copy of "hello"
    }
}

```



```

String s2 = "goodbye";
String s3 = "Happy Birthday";
String s4 = "happy birthday";
System.out.printf(
    "s1 = %s\ns2 = %s\ns3 = %s\ns4 = %s\n\n", s1, s2, s3, s4 );

// test for equality
if ( s1.equals( "hello" ) ) // true
    System.out.println( "s1 equals \"hello\"" );
else
    System.out.println( "s1 does not equal \"hello\"" );

// test for equality with ==
if ( s1 == "hello" ) // false; they are not the same object
    System.out.println( "s1 is the same object as \"hello\"" );
else
    System.out.println( "s1 is not the same object as \"hello\"" );

// test for equality (ignore case)
if ( s3.equalsIgnoreCase( s4 ) ) // true
    System.out.printf( "%s equals %s with case ignored\n", s3, s4 );
else
    System.out.println( "s3 does not equal s4" );

// test compareTo
System.out.printf( "\ns1.compareTo( s2 ) is %d", s1.compareTo( s2 ) );
System.out.printf( "\ns2.compareTo( s1 ) is %d", s2.compareTo( s1 ) );
System.out.printf( "\ns1.compareTo( s1 ) is %d", s1.compareTo( s1 ) );
System.out.printf( "\ns3.compareTo( s4 ) is %d", s3.compareTo( s4 ) );
System.out.printf(
    "\ns4.compareTo( s3 ) is %d\n\n", s4.compareTo( s3 ) );

// test regionMatches (case sensitive)
if ( s3.regionMatches( 0, s4, 0, 5 ) )
    System.out.println( "First 5 characters of s3 and s4 match" );

```

```

else
    System.out.println(
        "First 5 characters of s3 and s4 do not match" );

// test regionMatches (ignore case)
if ( s3. regionMatches( true, 0, s4, 0, 5 ) )
    System.out.println( "First 5 characters of s3 and s4 match" );
else
    System.out.println(
        "First 5 characters of s3 and s4 do not match" );
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```

s1 = hello
s2 = goodbye
s3 = Happy Birthday
s4 = happy birthday
s1. equals "hello"
s1 is not the same object as "hello"
Happy Birthday equals happy birthday with case ignored
s1.compareTo( s2 ) is 1
s2.compareTo( s1 ) is -1
s1.compareTo( s1 ) is 0
s3.compareTo( s4 ) is -32
s4.compareTo( s3 ) is 32
First 5 characters of s3 and s4 do not match
First 5 characters of s3 and s4 match
Process completed.

```

□ يقدم الصف String أيضاً التابعين :

regionMatches()

equalsIgnoreCase()

ويقارن التابع regionMatches() أجزاء من السلسلتين من أجل فحص التساوي.
حيث يتجاهل التابع equalsIgnoreCase() حالة الأحرف ، عندما يقرر فيما إذا كانت السلسلتان متساويتين أم لا .

- يمكن استخدام التابع str.startsWith(*prefix*) لفحص فيما إذا كانت السلسلة str تبدأ بسابقة محددة (الموجودة في بداية السلسلة) .

- يمكن استخدام التابع str.endsWith(*suffix*) لفحص فيما إذا كانت السلسلة str تنتهي بلاحقة محددة (الموجودة في نهاية السلسلة) .

وبين المثال (6-8) كيفية استخدام التابعين :

str.startsWith(*prefix*) , str.endsWith(*suffix*)

مثال (6-8)

// String methods startsWith and endsWith.

public class string8 {

public static void main(String args[]) {

String strings[] = { "started", "starting", "ended", "ending" };

// test method startsWith

for (String string : strings) {

if (string.startsWith("st"))

System.out.printf("\"%s\" starts with \"st\"\n", string);

} // end for

System.out.println();

// test method startsWith starting from position 2 of string

for (String string : strings) {

if (string.startsWith("art", 2))

System.out.printf(

"\"%s\" starts with \"art\" at position 2\n", string);

```

    } // end for
    System.out.println( );
// test method endsWith
    for ( String string : strings ) {
        if ( string.endsWith( "ed" ) )
            System.out.printf( "\"%s\" ends with \"ed\\n\"", string );
    } // end for
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

started" starts with "art" at position 2
 starting" starts with "art" at position 2"
 started" ends with "ed"
 ended" ends with "ed"
 Process completed.

٦-٢-٦ - توابع تحويلات السلاسل

- لا يمكن تغيير محتوى النص في السلسلة بعد أن يتم إنشاؤها ، لكن يمكننا تحويل النص في السلسلة إلى نص جديد باستخدام التوابع التالية :
- يُستخدم التابع `replace( oldChar , newChar )` من أجل استبدال محرف من النص القديم في السلسلة بمحرف جديد .
 - يُستخدم التابع `toLowerCase( )` من أجل تحويل جميع محارف النص في السلسلة إلى شكلها الصغير .

- يُستخدم التابع `toUpperCase()` من أجل تحويل جميع محارف النص في السلسلة إلى شكلها الكبير .
- يُستخدم التابع `trim()` من أجل حذف محارف الفراغات من بداية ونهاية النص في السلسلة.
- يُستخدم التابع `toArray()` لتحويل النص في السلسلة إلى مصفوفة ، وكذلك العكس .

مثلاً يمكننا أن نحول النص "hello" إلى مصفوفة محرفية كما يلي :

```
String s1 = new String( "hello" );
char charArray[ ] = s1.toCharArray( );
```

وبالتالي فإن قيمة `charArray[0]` هي h و `charArray[1]` هي e و `charArray[2]` هي l و `charArray[3]` هي l و `charArray[4]` هي o .

يبين المثال (6- 9) كيفية استخدام التوابع السابقة .

□ التابع التالي :

`getChars( int srcBegin ,int srsEnd, char[ ] dst , int dstBegin )` يستخدم من أجل نسخ نص جزئي من النص الأصلي بحيث يبدأ من الدليل `srcBegin` وينتهي عند الدليل `srsEnd-1` إلى مصفوفة محرفية `dst` والتي يبدأ التخزين فيها انطلاقاً من الدليل `dstBegin` ، والمثال (6- 9) يبين كيفية الحصول على مصفوفة `charArray[7]` من النص الأصلي في السلسلة `s1 = "Welcome to Java"` كما يلي :

```
s1.getChars( 0, 7, charArray, 0 );
```


مثال (9-6)

```

/* String methods replace, toLowerCase, toUpperCase, trim and
toCharArray. */
public class string9 {
    public static void main( String args[ ]) {
        String s1 = new String( "hello" );
        String s2 = new String( "GOODBYE" );
        String s3 = new String( " Welcom to Java " );
        System.out.printf( "s1 = %s\ns2 = %s\ns3 = %s\n\n", s1, s2, s3 );
        // test method replace
        System.out.printf(
            "Replace 'l' with 'L' in s1: %s\n\n", s1.replace( 'l', 'L' ) );
        // test toLowerCase and toUpperCase
        System.out.printf( "s1.toUpperCase( ) = %s\n", s1.toUpperCase( ) );
        System.out.printf( "s2.toLowerCase( ) = %s\n\n", s2.toLowerCase());
        // test trim method
        System.out.printf( "s3 after trim = \"%s\"\n\n", s3.trim() );
        // test toCharArray method
        char charArray[ ] = s1.toCharArray( );
        System.out.print( "s1 as a character array = " );
        for ( char character : charArray )
            System.out.print( character );
        System.out.println( );
    } // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```
replace 'l' with 'L' in s1: heLlO  
s1.toUpperCase( ) = HELLO  
s2.toLowerCase( ) = goodbye  
s3 after trim = "Welcom to Java"  
s1 as a character array = hello  
Process completed.
```

٦-٢-٧ - البحث عن محرف أو نص جزئي في السلسلة

يمكن استخدام التابعين `indexOf( )` و `lastIndexOf( )` للبحث عن محرف أو نص جزئي ضمن السلسلة ، ويوجد في الصف `String` أربعة توابع ذات تحميل زائد للتابع `indexOf( )` وكذلك للتابع `lastIndexOf( )` .

١. يُعيد التابع `public int indexOf( int ch)` دليل أول محرف موجود في النص الأصلي في السلسلة المحرّفة يتطابق مع المحرف `ch` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الأصلي في السلسلة .

٢. يُعيد التابع `public int lastIndexOf( int ch)` دليل آخر محرف موجود في النص الأصلي في السلسلة المحرّفة يتطابق مع المحرف `ch` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الأصلي في السلسلة .

٣. يُعيد التابع `public int indexOf( int ch, int fromIndex)` دليل أول محرف موجود في النص الأصلي في السلسلة المحرّفة يتطابق مع المحرف `ch` ويبدأ من المحدد ، `fromIndex` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الجزئي في السلسلة .

٤. يُعيد التابع `public int lastIndexOf( int ch, int endIndex)`

دليلاً آخر محرف موجود في النص الأصلي في السلسلة المحرفية، يتطابق مع المحرف `ch` وينتهي بالمحدد `endIndex` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الجزئي في السلسلة .

٥. يُعيد التابع `public int indexOf( String str)` دليل أول محرف من

النص الجزئي الموجود في النص الأصلي في السلسلة المحرفية ، والذي يتطابق مع المحرف `str` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الجزئي في السلسلة .

٦. يُعيد التابع `public int lastIndexOf( String str)` دليلاً آخر

محرف من النص الجزئي الموجود في النص الأصلي، في السلسلة المحرفية ، والذي يتطابق مع المحرف `str` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الجزئي في السلسلة .

٧. يُعيد التابع `public int indexOf( String str, int fromIndex)`

دليل أول محرف موجود في النص الجزئي الموجود في النص الأصلي في السلسلة المحرفية يتطابق مع المحرف `str` ، ويبدأ من المحدد `fromIndex` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الجزئي في السلسلة .

٨. التابع `public int lastIndexOf( String ch, int endIndex)`

يعيد دليلاً آخر محرف موجود في النص الجزئي الموجود في النص الأصلي في السلسلة المحرفية يتطابق مع المحرف `str` ، وينتهي بالمحدد `endIndex` ، ويعيد القيمة (-1) إذا لم يجد المحرف المحدد في النص الجزئي في السلسلة .

يبين المثال (6- 10) كيفية استخدام التوابع السابقة .

// String searching methods indexOf and lastIndexOf.

```
public class string10 {
```

```
    public static void main( String args[ ] ) {
```

```
        String letters = "abcdefghijklmabcdefghijklm";
```

// test indexOf to locate a character in a string

```
        System.out.printf(
```

```
            "'c' is located at index %d\n", letters.indexOf( 'c' ) );
```

```
        System.out.printf(
```

```
            "'a' is located at index %d\n", letters.indexOf( 'a', 1 ) );
```

```
        System.out.printf(
```

```
            "'$' is located at index %d\n\n", letters.indexOf( '$' ) );
```

//test lastIndexOf to find a character in a string

```
        System.out.printf( "Last 'c' is located at index %d\n",
```

```
            letters.lastIndexOf( 'c' ) );
```

```
        System.out.printf( "Last 'a' is located at index %d\n",
```

```
            letters.lastIndexOf( 'a', 25 ) );
```

```
        System.out.printf( "Last '$' is located at index %d\n\n",
```

```
            letters.lastIndexOf( '$' ) );
```

// test indexOf to locate a substring in a string

```
        System.out.printf( "\"def\" is located at index %d\n",
```

```
            letters.indexOf( "def" ) );
```

```
        System.out.printf( "\"def\" is located at index %d\n",
```

```
            letters.indexOf( "def", 7 ) );
```

```
        System.out.printf( "\"hello\" is located at index %d\n\n",
```

```
            letters.indexOf( "hello" ) );
```

// test lastIndexOf to find a substring in a string

```
        System.out.printf( "Last \"def\" is located at index %d\n",
```

```
            letters.lastIndexOf( "def" ) );
```



```

System.out.printf( "Last \"def\" is located at index %d\n",
    letters.lastIndexOf( "def", 25 ) );
System.out.printf( "Last \"hello\" is located at index %d\n",
    letters.lastIndexOf( "hello" ) );
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```

'c' is located at index 2
'a' is located at index 13
' ' is located at index -1
'$' is located at index -1
Last 'c' is located at index 15
Last 'a' is located at index 13
Last '$' is located at index -1
def" is located at index 3"
def" is located at index 16"
hello" is located at index -1"
Last "def" is located at index 16
Last "def" is located at index 16
Last "hello" is located at index -1
Process completed.

```

٦-٢-٨- تحويل المحارف والقيم العددية إلى نصوص

من الممكن استخدام التابع (`valueOf()`) من أجل تحويل مصفوفة من المحارف إلى نص . ويوجد عدة توابع ذات تحميل زائد يمكن استخدامها من أجل

تحويل المحارف والقيم العددية إلى نصوص ، حيث تقبل تلك التوابع أنماط مختلفة من المتحولات المممة مثل : char, double, long, int, float . ويمكن استخدام التابع (valueOf) من أجل تحويل القيمة 5.44 من النمط double إلى نص ، حيث يعيد التابع ; String.valueOf(5.44) قيمة نصية تتألف من المحارف التالية : '4' , '4' , '.' , '5' . ويمكن استخدام التابع (valueOf(char []) من أجل تحويل مصفوفة من المحارف إلى نص . ويبين المثال (6- 11) كيفية استخدام التوابع السابقة .

مثال (6-11)

// String valueOf methods.

```
public class string11 {
    public static void main( String args[ ] ) {
        char charArray[ ] = { 'a', 'b', 'c', 'd', 'e', 'f' };
        boolean booleanValue = true;
        char characterValue = 'Z';
        int integerValue = 7;
        long longValue = 10000000000L; // L suffix indicates long
        float floatValue = 2.5f; // f indicates that 2.5 is a float
        double doubleValue = 5.44; // no suffix, double is default
        Object objectRef = "hello"; // assign string to an Object reference
        System.out.printf(
            "char array = %s\n", String.valueOf( charArray ) );
        System.out.printf( "part of char array = %s\n",
            String.valueOf( charArray, 3, 3 ) );
        System.out.printf(
            "boolean = %s\n", String.valueOf( booleanValue ) );
        System.out.printf(
            "char = %s\n", String.valueOf( characterValue ) );
        System.out.printf( "int = %s\n", String.valueOf( integerValue ) );
```

```
System.out.printf( "long = %s\n", String.valueOf( longValue ) );
```

```
System.out.printf( "float = %s\n", String.valueOf( floatValue ) );
```

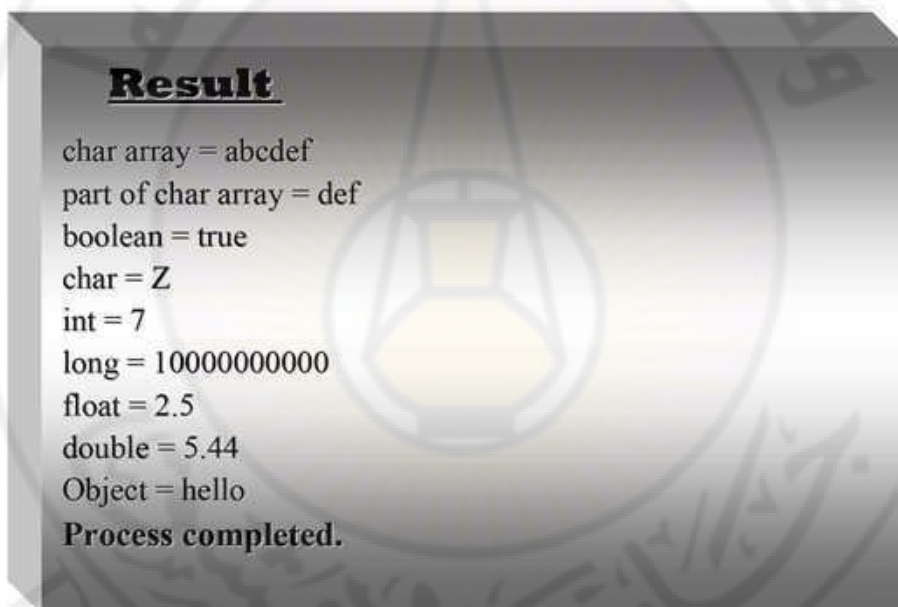
```
System.out.printf( "double = %s\n", String.valueOf( doubleValue ) );
```

```
System.out.printf( "Object = %s", String.valueOf( objectRef ) );
```

```
    } // end main
```

```
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٦-٢-٩ - فحص الكلمات المتناظرة

هو كتابة برنامج يفحص فيما إذا كان النص المُمرَّر يُمثل كلمة متناظرة أو لا ، أي يفحص فيما إذا كان من الممكن قراءة الكلمة من اليسار إلى اليمين أو من اليمين إلى اليسار دون تغيير المعنى . مثل كلمة dad أو غيرها .

مبدأ عمل الخوارزمية التي تحدد فيما إذا كان النص يمثل كلمة متناظرة أم لا هو كما يلي :

- استخدام حلقة لفحص فيما إذا كان الحرف الأول في النص هو نفسه آخر حرف ، فإذا كان الجواب نعم ، ننتقل إلى الحرف الثاني ، ثم نفحص هذا الحرف مع الحرف ما قبل الأخير في النص ، ونستمر في هذه العملية حتى نصل إلى حالة عدم توافق ، أو يتم فحص جميع المحارف في النص ما عدا المحرف المتوسط في السلسلة ، إذا كان عدد المحارف فردياً في النص) .

ويوجه البرنامج المستخدم لإدخال النص ، ثم يبين ما إذا كان هذا النص يمثل كلمة متناظرة أم لا ، ويبين المثال (6-12) ذلك . يستخدم التابع isPalindrome(s) الحلقة while ليقارن المحارف ، ويقرر فيما إذا كان النص يمثل كلمة متناظرة أم لا . ويوجد طريقة أخرى لفحص الكلمات المتناظرة ، ومبدأ هذه الطريقة هو الحصول معكوس الكلمة المتناظرة ، ثم القيام بعملية فحص للكلمة الأصلية مع الكلمة المعكوسة ، لمعرفة ما إذا كانت الكلمتان تحتويان على القيمة نفسها (متطابقتان) . وإذا كانت النتيجة (نعم) فهذا يعني أن الكلمة متناظرة ، أما إذا كانت النتيجة (لا) فهذا يعني أن الكلمة غير متناظرة . لا يوجد في الصف String تابع يقوم بعكس الكلمة ، لذلك يجب استخدام التابع reverse() الموجود في الصف StringBuffer لفحص فيما إذا كانت الكلمة متناظرة أم لا .

مثال (6-12)

```
import javax.swing.JOptionPane;
public class string12 {
    /** Main method */
    public static void main(String[ ] args) {
        // Prompt the user to enter a string
        String s = JOptionPane.showInputDialog(null,
            "Enter a string:", "Example 12 Input",
```

```

JOptionPane.QUESTION_MESSAGE);
//Declare and initialize output string
String output = "";
if (isPalindrome(s))
    output += s + " is a palindrome";
else
    output += s + " is not a palindrome";
// Display the result
JOptionPane.showMessageDialog(null, output,
    "Example 7.1 Output",
    JOptionPane.INFORMATION_MESSAGE);
System.exit(0);
} //end main
/** Check if a string is a palindrome */
public static boolean isPalindrome(String s) {
    // The index of the first character in the string
    int low = 0;
    // The index of the last character in the string
    int high = s.length() - 1;
    while (low < high) {
        if (s.charAt(low) != s.charAt(high))
            return false; // Not a palindrome
        low++;
        high--;
    } //end while
    return true; // The string is a palindrome
} //end isPalindrome
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٦-٣- Character الصف

تقدم Java صفاً مغلفاً من أجل كل نمط معطيات أولي ، وهذه الصفوف هي :

- | | |
|----------------|------------------|
| - الصف Long | - الصف Character |
| - الصف Integer | - الصف Boolean |
| - الصف Double | - الصف Byte |
| | - الصف Short |

وذلك من أجل كل الأنماط التالية :

char , boolean , byte , short , int , long , float , double

وتوجد جميع هذه الصفوف في الحزمة java.lang . , يمكننا هذه الصفوف من

التعامل مع قيم المعطيات الأولية على أنها أغراض ، بحيث نستطيع استخدام

العديد من التوابع المفيدة في معالجة تلك القيم . ويحتوي الصف Character

على بان واحد و أكثر من 30 تابعا للتعامل مع المحرف ، أهمها :

- يبني التابع ('c') Character غرضاً محرفياً كما يلي :
Character ch = new Character ('c') ;
- يُعيد التابع () charValue القيمة المحرفية المغلفة في الغرض Character .
- يقارن التابع () compareTo هذا المحرف مع آخر ويعيد قيمة صحيحة تدل على الفرق بين الترميز Unicode التابع لهذا المحرف والترميز Unicode التابع للمحرف الآخر .
- يعيد التابع () equal القيمة true ، إذا كان هذا المحرف يعادل (يساوي) محرف آخر .
- إن معظم التوابع التابعة للصف Character هي توابع ساكنة (static) :
- يعيد التابع () isDigit القيمة true إذا كان المحرف رقماً .
- يعيد التابع () isLetter القيمة true إذا كان المحرف حرفاً .
- يعيد التابع () isLetterOrDigit القيمة true إذا كان المحرف رقماً أو حرفاً .
- يعيد التابع () isLowerCase القيمة true إذا كان المحرف هو حرف في شكله الصغير .
- يعيد التابع () isUpperCase القيمة true إذا كان المحرف هو حرف في شكله الكبير .
- يعيد التابع () isLower القيمة true إذا كان المحرف هو حرف في شكله الصغير .
- يعيد التابع () toLowerCase الحرف الموجود في المحرف ch في شكله الصغير .
- يعيد التابع () toUpperCase الحرف الموجود في المحرف ch في شكله الكبير .

- يعيد التابع () isDefined القيمة true إذا كان المحرف c معرف في مجموعة ترميز Unicode ، وإلا يعيد False .
- يستخدم التابع () isJavaIdentifierStart لتحديد فيما إذا كان c هو المحرف الذي يمكن أن يكون الحرف الأول في معامل java ، كحرف أو علامة (_) underscore أو علامة (\$) عندها يعيد القيمة true .
- ويوضح المثال (6-13) بعض التتابع الساكنة التي تختبر المحارف لتحديد فيما إذا كانت محارف خاصة أو تتابع ساكنة لتقوم بتحويلات المحارف . ويمكننا إدخال أي محرف وتطبيق التتابع على هذه المحارف .

مثال (6-13)

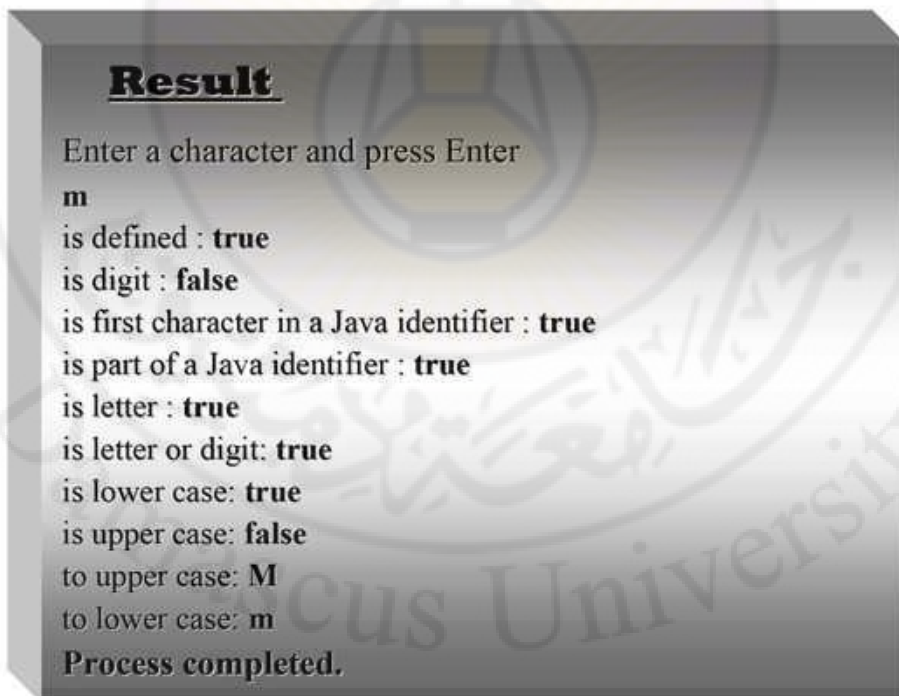
```
// Static Character testing methods and case conversion methods.
import java.util.Scanner;
public class string13 {      //StaticCharMethods
    public static void main( String args[ ] ) {
// create scanner
Scanner scanner = new Scanner( System.in );
System.out.println( "Enter a character and press Enter" );
String input = scanner.next( );
char c = input.charAt( 0 ); // get input character
// display character info
System.out.printf( "is defined: %b\n", Character.isDefined( c ) );
System.out.printf( "is digit: %b\n", Character.isDigit( c ) );
System.out.printf( "is first character in a Java identifier: %b\n",
                    Character.isJavaIdentifierStart( c ) );
System.out.printf( "is part of a Java identifier: %b\n",
                    Character.isJavaIdentifierPart( c ) );
System.out.printf( "is letter: %b\n", Character.isLetter( c ) );
```

```

System.out.printf(
    "is letter or digit: %b\n", Character.isLetterOrDigit( c ) );
System.out.printf(
    "is lower case: %b\n", Character.isLowerCase( c ) );
System.out.printf(
    "is upper case: %b\n", Character.isUpperCase( c ) );
System.out.printf(
    "to upper case: %s\n", Character.toUpperCase( c ) );
System.out.printf(
    "to lower case: %s\n", Character.toLowerCase( c ) );
} // end main
} // end class

```

١- إدخال محرف على شكل حرف



٢- إدخال محرف على شكل رقم

Result

Enter a character and press Enter

5

is defined: **true**

is digit: **true**

is first character in a Java identifier: **false**

is part of a Java identifier: **true**

is letter: **false**

is letter or digit: **true**

is lower case: **false**

is upper case: **false**

to upper case: 5

to lower case: 5

Process completed.

۳- إدخال محرف علی شکل رمز

Result

Enter a character and press Enter

@

is defined: **true**

is digit: **false**

is first character in a Java identifier: **false**

is part of a Java identifier: **false**

is letter: **false**

is letter or digit: **false**

is lower case: **false**

is upper case: **false**

to upper case: @

to lower case: @

Process completed.

في هذا المثال (6-14) يوضح استخدام التوابع الساكنة المحرّفية مثل :

- يقوم التابع digit بتحويل المحارف إلى أرقام .
 - يقوم التابع forDigit بتحويل الأرقام إلى محارف .
- هذه التوابع تقوم بتحويل المحارف إلى أرقام والأرقام إلى محارف ويشمل التحويل نظام العد العشري والثماني والسادسي عشر والثنائي .
- في السطر 1 التابع (digit, radix) Character.forDigit يُستخدم لتحويل الرقم الصحيح إلى حرف في نظام العد المحدد بواسطة المتحول radix ، فعلى سبيل المثال العدد العشري 13 يُمثل في نظام العد السداسي عشر بالمحرف (d) .
 - في السطر 2 التابع (character, radix) Character.digit يُستخدم لتحويل المحرف إلى رقم في نظام العد المحدد بواسطة المتحول radix ، فعلى سبيل المثال المحرف (A) يمثل في نظام العد العشري بالقيمة (10) .

مثال (6-14)

```
// Static Character conversion methods.
import java.util.Scanner;
public class string14 { // StaticCharMethods2
// create StaticCharMethods2 object execute application
    public static void main( String args[ ] ) {
        Scanner scanner = new Scanner( System.in );
// get radix
        System.out.println( "Please enter a radix:" );
        int radix = scanner.nextInt();
// get user choice
        System.out.printf( "Please choose one:\n1 -- %s\n2 -- %s\n",
            "Convert digit to character", "Convert character to digit");
        int choice = scanner.nextInt();
// process request
```



```

switch ( choice ) {
    case 1 : // convert digit to character
        System.out.println( "Enter a digit:" );
        int digit = scanner.nextInt();
        System.out.printf( "Convert digit to character: %s\n",
            Character.forDigit( digit, radix ) );// 1
        break;
    case 2 : // convert character to digit
        System.out.println( "Enter a character:" );
        char character = scanner.next().charAt( 0 );
        System.out.printf( "Convert character to digit: %s\n",
            Character.digit( character, radix ) );//2
        break;
} // end switch
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result
 Please enter a radix:
 23
 Please choose one:
 1 -- Convert digit to character
 2 -- Convert character to digit
 1
 Enter a digit:
 18
 Convert digit to character: i
 Process completed.

Result
 Please enter a radix:
 23
 Please choose one:
 1 -- Convert digit to character
 2 -- Convert character to digit
 2
 Enter a character:
 g
 Convert character to digit: 16
 Process completed.

في المثال (6-15) يوضح عدد من التوابع غير الساكنة مثل : `charValue()` ، `equal()` ، `toString()`

مثال (6-15)

// Non-static Character methods.

```
public class string15 { //OtherCharMethods
    public static void main( String args[ ] ) {
        Character c1 = 'M';
        Character c2 = 'm';
        System.out.printf(
            "c1 = %s\nc2 = %s\n\n", c1.charValue( ), c2.toString( ) );
        if ( c1.equals( c2 ) )
            System.out.println( "c1 and c2 are equal\n" );
        else
            System.out.println( "c1 and c2 are not equal\n" );
    } // end main
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```
c1 = M
c2 = m
c1 and c2 are not equal
Process completed.
```

٦-٤ - الصف StringBuffer

يُعد الصف StringBuffer بديلاً للصف String ، و بشكل عام ، من الممكن استخدام الصف StringBuffer في أي مكان يُستخدم فيه الصف String ، ويتميز الصف StringBuffer عن الصف String بأنه أكثر مرونة حيث نستطيع إضافة محتويات جديدة إلى غرض الـ StringBuffer في حين لا يمكن تغيير قيمة غرض String بعد أن يتم إنشاؤه .

يملك الصف `StringBuffer` ثلاثة بواني وأكثر من 30 تابعاً من أجل إدارة الوسيط `Buffer` وتعديل السلاسل المحرفية (النصوص) المخزنة فيه .

١- بواني `StringBuffer`

يقدم الصف `StringBuffer` ثلاثة بواني هي :

- الباني `public StringBuffer( )` وهو بان بدون وسطاء ، يبني وسيطاً نصياً بدون محارف ومع سعة ابتدائية حجمها 16 محرفاً .
 - الباني `public StringBuffer( int length )` وهو بان بوسيط واحد من نوع `int`، يبني وسيطاً نصياً بدون محارف ، ومع سعة ابتدائية حجمها يساوي قيمة المحدد `length` .
 - الباني `public StringBuffer( String str )` وهو بان بوسيط واحد من نوع `String` ، يبني وسيطاً نصياً من أجل الوسيط المحرفي `str` مع سعة ابتدائية حجمها يساوي طول الوسيط المحرفي مع العدد 16 .
- يبين المثال (6-16) بواني الصف `StringBuffer` .

مثال (6-16)

// StringBuffer constructors.

```
public class string16 {  
    public static void main( String args[ ] ) {  
        StringBuffer buffer1 = new StringBuffer( );  
        StringBuffer buffer2 = new StringBuffer( 10 );  
        StringBuffer buffer3 = new StringBuffer( " Welcome " );  
        System.out.printf( "buffer1 = \"%s\"\\n", buffer1.toString( ) );  
        System.out.printf( "buffer2 = \"%s\"\\n", buffer2.toString( ) );  
        System.out.printf( "buffer3 = \"%s\"\\n", buffer3.toString( ) );  
    } // end main  
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```
buffer1 = " "  
buffer2 = " "  
buffer3 = " Welcome "  
Process completed.
```

٢- توابع الصف StringBuffer

يقدم الصف StringBuffer عدة توابع من أجل التعامل مع الوسيط النصي . ويوضح المثال (6-17) هذه التوابع ، ومن أهمها :

- يعيد التابع (toString) النص من الوسيط النصي (السلسلة المحرفية) .
- يعيد التابع (capacity) السعة الحالية للوسيط النصي (السعة هي عدد المحارف الممكن تخزينها) .
- يعيد التابع (length) العدد الفعلي للمحارف المخزنة في الوسيط النصي .
- يقوم التابع (setLength(newLength) بتحديد طول السلسلة المحرفية ، وإذا كان طول المحدد newLength أصغر من الطول الحالي للوسيط النصي عندها يتم قص هذا الوسيط بحيث يحتوي تماماً على عدد المحارف المحدد في البارامتر newLength ، أما إذا كان newLength أكبر أو يساوي الطول الحالي عندها تتم إضافة عدد كاف من المحرف null ('\u0000') على الوسيط النصي بحيث يصبح طوله يساوي قيمة المحدد newLength .
- يضمن التابع (ensureCapacity(أن الصف StringBuffer لديه على الأقل السعة الكافية .

مثال (6-17)

```
/* StringBuffer length, setLength, capacity  
and ensureCapacity methods. */  
public class string17 {  
    public static void main( String args[ ] ) {  
        StringBuffer buffer = new StringBuffer( "Hello, how are you?" );  
        System.out.printf( "buffer = %s\nlength = %d\ncapacity = %d\n\n",  
            buffer.toString(), buffer.length(), buffer.capacity( ) );  
        buffer.ensureCapacity( 75 );  
        System.out.printf( "New capacity = %d\n\n", buffer.capacity( ) );  
        buffer.setLength( 10 );  
        System.out.printf( "New length = %d\nbuf = %s\n",  
            buffer.length(), buffer.toString() );  
    } // end main  
} // end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```
buffer = Hello, how are you?  
length = 19  
capacity = 35  
New capacity = 75  
New length = 10  
buf = Hello, how  
Process completed.
```

٣- التوابيع (reverse, charAt, setCharAt, getChars)

يقدم الصف `StringBuffer` عدة توابيع أخرى من التعامل مع السلاسل المحرفية ، ومن أهمها :

- يعيد التابع `charAt(index)` المحرف الموجود ضمن الوسيط المحرفي والذي دليله يساوي المحدد `index` ، ويجب ان يكون المحدد `index` أكبر أو يساوي الصفر (0) وأصغر تماماً من طول السلسلة المحرفية .
- والتابع (`getChars( int srcBegin ,int srcEnd, char[ ] dst , int dstBegin )` يُستخدم من أجل نسخ سلسلة جزئية من السلسلة الأصلية بحيث تبدأ من الدليل `srcBegin` وتنتهي عند الدليل `srcEnd-1` إلى مصفوفة محرفية `dst` والتي يبدأ التخزين فيها انطلاقاً من الدليل `dstBegin` .
- يستبدل التابع (`setCharAt(int,char)` المحرف الذي دليله `int` إلى حرف صغير أو كبير أو إلى أي محرف آخر .
- يعكس التابع (`reverse()` السلسلة المحرفية في الصف `StringBuffer` .
- ويبين المثال (6-18) كيفية استخدام هذا التتابع في الصف `StringBuffer` .

مثال (6-18)

```

/* StringBuffer methods
   charAt, setCharAt, getChars and reverse. */
public class string18 {
    public static void main( String args[ ] ) {
        StringBuffer buffer = new StringBuffer( "welcome to you" );
        System.out.printf( "buffer = %s\n", buffer.toString() );
        System.out.printf( "Character at 0: %s\nCharacter at 4: %s\n\n",
                           buffer.charAt( 0 ), buffer.charAt( 4 ) );
        char charArray[ ] = new char[ buffer.length( ) ];
        buffer.getChars( 0, buffer.length( ), charArray, 0 );
        System.out.print( "The characters are: " );
        for ( char character : charArray )
            System.out.print( character );
        buffer.setCharAt( 0, 'W' );
        buffer.setCharAt( 7, 'T' );
    }
}

```

```

buffer.setCharAt( 10, 'Y' );
System.out.printf( "\n\nbuf = %s", buffer.toString( ) );
buffer.reverse( );
System.out.printf( "\n\nbuf = %s\n", buffer.toString( ) );
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

```

buffer = welcome to you
Character at 0: w
Character at 4: o
The characters are : welcome to you
buf = Welcome To You
buf = uoY oT emocleW
Process completed.

```

٤ - التتابع (append ,insert, deleteCharAt, Methods)

يحتوي الصف StringBuffer على توابع من أجل إضافة محتويات جديدة إلى نهاية السلسلة المحرفية أو حشر محتويات جديدة في مكان محدد ضمنه أو حذف بعض المحارف منه أو استبدالها . ويقدم الصف StringBuffer عدة توابع ذات تحميل زائد من أجل إضافة قيم من جميع الأنماط :

(String , boolean, int,long,float, double,char,...)

منها :

- التابع (append(string)
- التابع (insert(int,char)
- التابع (deleteCharAt(int)

يبين المثال (6-19) كيفية استخدام هذه التتابع .

// StringBuffer append methods.

```
public class string19 {
    public static void main( String args[ ] ) {
        Object objectRef = "hello";
        String string = "goodbye";
        char charArray[ ] = { 'a', 'b', 'c', 'd', 'e', 'f' };
        boolean booleanValue = true;
        char characterValue = 'Z';
        int integerValue = 7;
        long longValue = 100000000000L;
        float floatValue = 2.5f; // f suffix indicates 2.5 is a float
        double doubleValue = 33.333;

        StringBuffer lastBuffer = new StringBuffer( "last StringBuffer" );
        StringBuffer buffer = new StringBuffer( );
        StringBuffer buffer1 = new StringBuffer( );

        // append Methods
        buffer.append( objectRef );
        buffer.append( "\n" ); // each of these contains new line
        buffer.append( string );
        buffer.append( "\n" );
        buffer.append( charArray );
        buffer.append( "\n" );
        buffer.append( charArray, 0, 3 );
        buffer.append( "\n" );
        buffer.append( booleanValue );
        buffer.append( "\n" );
        buffer.append( characterValue );
        buffer.append( "\n" );
        buffer.append( integerValue );
        buffer.append( "\n" );
    }
}
```

```

buffer.append( longValue );    buffer.append( "\n" );
buffer.append( floatValue );   buffer.append( "\n" );
buffer.append( doubleValue );  buffer.append( "\n" );
buffer.append( lastBuffer);

System.out.printf( "buffer contains %s\n", buffer.toString() );
System.out.printf( "-----buffer isert----- \n");
    buffer1.append( string );
    buffer1.append( "\n" );
    buffer1.insert( 0," Mr Mamoun " );
    System.out.printf( "buffer isert  %s\n", buffer1.toString( ) );
    System.out.printf( "-----buffer delete----- \n");
    buffer1.deleteCharAt(2);
    System.out.printf( "buffer delete  %s\n", buffer1.toString() );
} // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على، الخرج التالي:

Result

```

buffer contains  hello
goodbye
abcdef
abc
true
Z
7
100000000000
2.5
33.333
last StringBuffer
-----buffer isert-----
buffer isert  Mr Mamoun goodbye
-----buffer delete-----
buffer delete  M Mamoun goodbye
Process completed.

```


مثال (6-20)

هذا المثال هو تعديل على المثال (6-12) الذي يفحص الكلمات المتناظرة .
يفحص المثال فيما إذا كانت الكلمة (السلسلة) متناظرة أم لا . ويقوم هذا
البرنامج بتجاهل المحارف غير الأبجدية أو الرقمية أثناء فحص الكلمة المتناظرة .
ومن الممكن تحقيق ذلك بإضافة كل محرف (أبجدي - رقمي) موجود في
السلسلة إلى الوسيط المحرفي ، تتم استعادة السلسلة من الوسيط المحرفي .
نستطيع استخدام التابع (ch) isLetterOrDigit الموجود في الصف
Character من أجل فحص المحرف ch لمعرفة فيما إذا كان حرفاً أو رقماً .

البرنامج :

```
import javax.swing.JOptionPane;
public class string21 {
    /** Main method */
    public static void main(String[ ] args) {
        // Prompt the user to enter a string
        String s = JOptionPane.showInputDialog(null,
            "Enter a string:", "Example 21 Input",
            JOptionPane.QUESTION_MESSAGE);
        // Declare and initialize output string
        String output = "Ignoring non-alphanumeric characters, \nis "
            + s + " a palindrome? " + isPalindrome(s);
        // Display the result
        JOptionPane.showMessageDialog(null, output,
            "Example 21 Output",
            JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }
    /** Return true if a string is a palindrome */
    public static boolean isPalindrome(String s) {
```

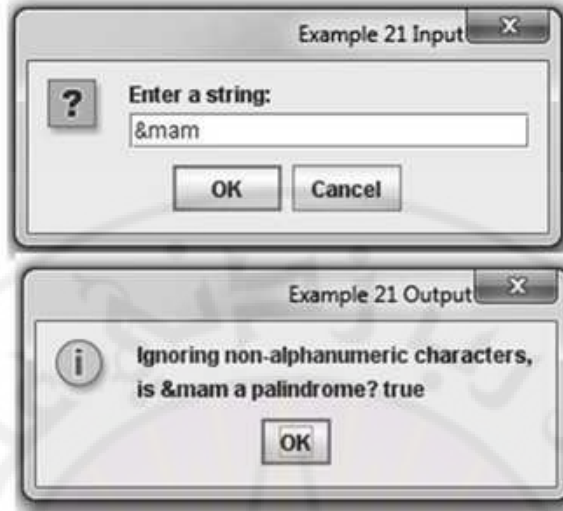


```

// Create a new string by eliminating non-alphanumeric chars
String s1 = filter(s);
// Create a new string that is the reversal of s1
String s2 = reverse(s1);
// Compare if the reversal is the same as the original string
return s2.equals(s1);
}
/** Create a new string by eliminating non-alphanumeric chars */
public static String filter(String s) {
// Create a string buffer
StringBuffer strBuf = new StringBuffer( );
// Examine each char in the string to skip alphanumeric char
for (int i = 0; i < s.length(); i++) {
if (Character.isLetterOrDigit(s.charAt(i))) {
strBuf.append(s.charAt(i));
}
}
// Return a new filtered string
return strBuf.toString();
}
/** Create a new string by reversing a specified string */
public static String reverse(String s) {
StringBuffer s1 = new StringBuffer(s);
s1.reverse( ); // Use the reverse method for StringBuffer object
return s1.toString( );
}
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٦-٥ - الصف StringTokenizer

يُستخدم الصف StringTokenizer من أجل تقسيم السلسلة المحرفية إلى وحدات tokens ، وتتألف كل وحدة من عدة محارف ، يفصل بينها محارف فواصل مثل الفراغ والفاصلة والنقطة وغيرها من الفواصل .

مثلاً من أجل الحصول على جميع الكلمات التي تتألف منها السلسلة المحرفية (I am Learning Java Now) ، يمكننا إنشاء غرضٍ من الصف StringTokenizer لهذه السلسلة ، تتم استعادة الكلمات المنفصلة والموجودة

في هذه السلسلة باستخدام التوابع التابعة لهذا الصف ، وللصف ثلاثة بواني :

- الباني StringTokenizer(**Strnig** s) يبيني غرضاً جديداً من أجل السلسلة المحرفية s ، ويحدد الفواصل الافتراضية "\r\n\t" ، ولا تُعدّ الفواصل هنا جزء من السلسلة ،

- الباني StringTokenizer(**Strnig** s, **Strnig** d) يبيني غرضاً من أجل السلسلة s ويحدد الفواصل d ، ولا تُعدّ الفواصل هنا جزءاً من السلسلة .

- الباني StringTokenizer(**Strnig s, Strnig d , bool r**) يبني
غرضاً من الصف من أجل السلسلة المحرفية s ويحدد الفواصل d ، وتُعدّ
كل محرف من المحدد d فاصلاً. إذا كانت قيمة المحدد r هي true عندها
تُعدّ الفواصل جزءاً من السلسلة .
- يحتوي الصف StringTokenizer على عدد من التوابع لمعالجة الأجزاء :
- يعيد التابع hasMoreTokens () القيمة true إذا كان لا يزال يوجد أجزاء
من السلسلة المحرفية .
- يعيد التابع nextToken () الجزء التالي من السلسلة المحرفية .
- يعيد التابع nextToken (String d) الجزء التالي من السلسلة المحرفية
بعد تبديل الفواصل بمجموعة الفواصل المحددة في المتحول d .
- يعيد التابع countTokens () عدد الأجزاء المتبقية من السلسلة المحرفية .
- يبين المثال (21-6) كيفية استخدام هذه التوابع .

مثال (21-6)

```
// StringTokenizer class.
import java.util.Scanner;
import java.util.StringTokenizer;
public class string20 {
// execute application
public static void main( String args[ ] ) {
// get sentence
Scanner scanner = new Scanner( System.in );
System.out.println( "Enter a sentence and press Enter" );
String sentence = scanner.nextLine();
// process user sentence
StringTokenizer tokens = new StringTokenizer( sentence );
```

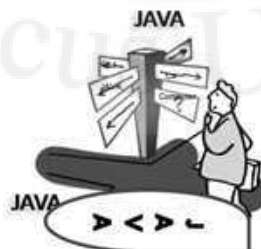
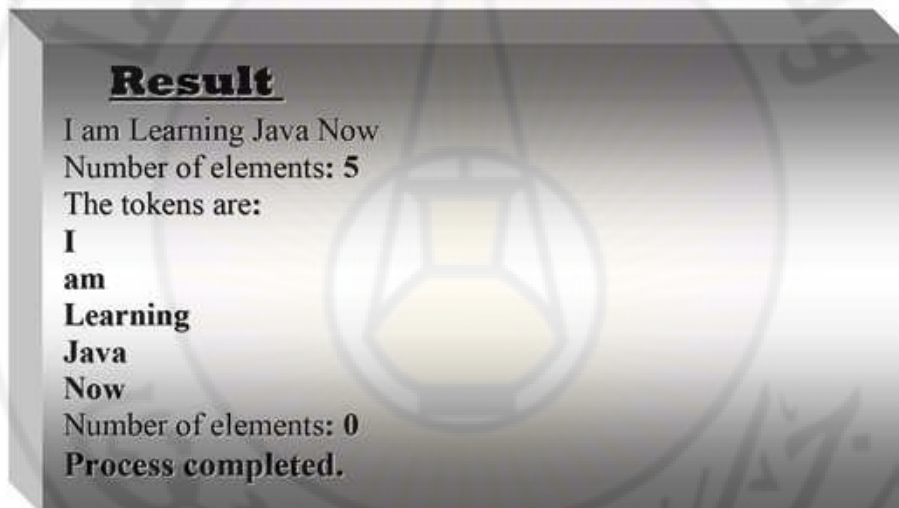
```

System.out.printf( "Number of elements: %d\nThe tokens are:\n",
    tokens.countTokens( ) );
    while ( tokens.hasMoreTokens( ) )
System.out.println( tokens.nextToken( ) );
System.out.printf( "Number of elements: %d\n",
    tokens.countTokens( ) );

    } // end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٦-٦-٦ معالجة الملفات في Java

٦-٦-٦-١ مقدمة

إن البرامج التي تمت دراستها في الفصول السابقة تحتاج إلى المعطيات فورية لعملية الإدخال والإخراج ، وينتهي دورها بمجرد إظهار النتائج على الشاشة. لكن هناك العديد من التطبيقات تحتاج إلى معالجة كمية كبيرة من المعطيات تكون موجودة قبل تنفيذ البرنامج ، ويتم تخزينها بعد تنفيذ البرنامج . وهنا يجب أن نتمكن مستخدم البرنامج من الوصول إلى المعلومات التي قام بتخزينها وقراءتها وتعديلها وحفظها.

وسنأخذ مثلاً على ذلك أي " text editor " ، وهو برنامج لتحرير النصوص ، وليكن برنامج "word" أو "notepad" ، لنفرض أننا نريد أن نكتب نص شعري فأول ما سنقوم به هو فتح ملف "word" جديد باستخدام الأمر "New" وإعطائه اسماً ما ، يمكننا من الاستدلال عليه في المستقبل ، ومن ثم نقوم بالكتابة على الملف وبعد الكتابة نقوم بعمل "Save" لتخزين الملف . وكذلك هو الحال في الملفات في لغة Java ، فلو قمنا بتصميم برنامج لحفظ أسماء الموظفين ورواتبهم وأرقامهم الوظيفية ، فإن ذلك يستلزم أن تكون في ملف في قاعدة المعطيات. ولو وضعناها كالعادة في مصفوفة Array سنجد أننا بعد الكتابة في المصفوفة Array وإغلاق المترجم compiler لن نستطيع أن نجد المعلومات مرة أخرى ، على افتراض أننا ندخل معلوماتهم باستخدام الصف Scanner. فهنا الملفات قامت بحل المشكلة و بعد معرفتنا للملفات نستطيع أن نكتب ما شئنا من معلومات ومعطيات ونخزنها في ملف على حاسوبنا ، وفي أي مكان نريد ونستطيع التعديل عليها وقراءتها . وهناك أكثر من 80 % من

الشركات والمؤسسات الكبرى تعتمد اعتماداً كلياً في عملها على فكرة الملفات .
ومن هنا ظهرت الحاجة إلى استخدام الملفات .

٦-٦-٢- أنواع الملفات

هناك نوعان من أنواع الملفات :

١- الملفات الثنائية Binary File

وهي ملفات يتم تخزينها على شكل أعداد ثنائية (0 و 1) ونستطيع باستخدام هذه الملفات أن نخزن بها غرض object كاملاً . مثلاً لو أردنا تخزين بيانات موظف لديه اسم ورقم وظيفي وراتب في شكل Object واحد ، فسوف يُمكننا الملف الثنائي Binary File من تخزينه والوصول إليه . وسيكون مخزناً على شكل Stream أي مسار ، ولا نستطيع أن نقرأ من الملف مباشرة . ويتم عادة استخدام اللاحقة data. مثل: ("myFile.data").

٢- ملفات النصوص Text File

وفيها يتم تخزين البيانات على هيئة نصوص ، وهي تشبه تماماً ملف Word . ويتم تخزين المدخلات فيها على شكل String أي نصوص كما هو الحال في ال Word حتى لو كتبنا رقم سيخزن على أنه String . وتخزن عادة هذه الملفات على شكل ("MyFile.txt") .

٦-٦-٣- أنواع القراءة من الملفات

يوجد ثلاثة أنواع :

١- الملفات المتتابعة Sequence Files

ويتم تخزين الملفات والقراءة منها على شكل سلسلة مرتبة، الأول فالتالي . أي أننا لا نتمكن من قراءة مجرى Stream قبل مجرى Stream تم تخزينه أولاً . ويكون

الوصول للغرض object المخزن في صورة stream داخل ملف " قراءة أو كتابة متسلسله بالترتيب " Sequential-Access File .

٢ - الملفات العشوائية Files Random

وتكون القراءة من هذه الملفات عشوائياً ، أي لا يشترط أن يكون مخزناً أولاً حتى يُقرأ أولاً. وبذلك تتم القراءة من الآخر ومن الأول ومن المنتصف بشكل غير مرتب .

٣ - ملفات الوصول المباشر Direct Access

في هذه الحالة يصبح الملف كالمصفوفة Array وهنا نستطيع الوصول للمجرى Stream الذي نريده مباشرةً ونعني هنا بالمجرى Stream (الغرض Object المخزن في الملف) ولا يجب أن نمر على السلسلة من الأول لنصل لغرض Object معين .

□ تعريف المجرى Stream :

هو الغرض object الذي يسمح بتدفق (نقل) المعطيات بين البرنامج وبعض وحدات الإدخال والإخراج أو الملفات . وهناك نوعان من المجاري مجرى للإدخال (I/P Stream) ومجرى للإخراج (O/P Stream) .

٦-٤ - التعامل مع الملفات File Operations

يوجد ثلاث خطوات للتعامل مع الملفات من أي نوع :

١- فتح الملف open

في هذه الحالة نقوم بفتح الملف وإعطائه اسم باستخدام الصف File ويكون الأمر كالتالي :

```
File myFile= new File("newFile.");
```

٢- القيام بعملية القراءة أو الكتابة فيه (Process)

ويتم ذلك بعبارات الإدخال والإخراج في لغة Java .

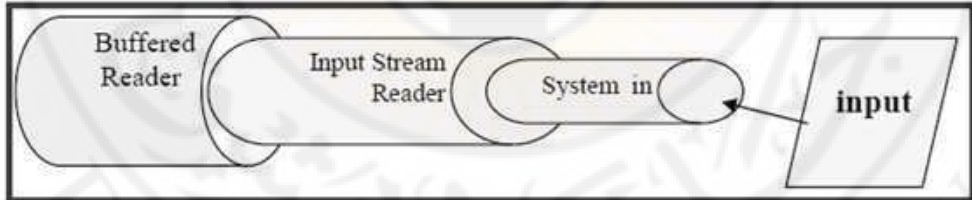
٣- إنهاء الملف وتخزينه (Save)

ويتم ذلك باستخدام الأمر (Close()) .

كما هو معروف لكي نتعامل مع أي ملف لابد أن يكون هناك إما مدخلات وإما مخرجات وسوف نتعرف على كيفية الإدخال وأنواعه وعملية الإخراج في الملفات النصية .

٦-٥-٦ - عملية القراءة والكتابة في الملفات

تتم عملية الإدخال والإخراج عموماً في لغة Java كما يلي :
أدوات الإدخال والإخراج في لغة Java تسمى مجارى (Stream) الدخل والخرج .
حيث يتم استقبال المعلومات المدخلة عن طريق المجرى stream كما هو موضح في الشكل (٦-٣) :



الشكل (٦-٣) - يوضح عملية القراءة في الملفات

أولاً - عملية القراءة

نحتاج في عملية القراءة إلى ثلاثة مجاري (Stream) كما هو مبين في الشكل (٦-١) وهذه المجاري هي :

❖ المجرى System in

وهو المجرى الأول ويعمل على قراءة بايت واحد في كل مرة .

❖ المجرى Input Stream Reader

وهو المجرى الثاني و يقوم بتحويل كل 2 بايت إلى حرف أو رمز .

❖ المجرى Buffered Reader

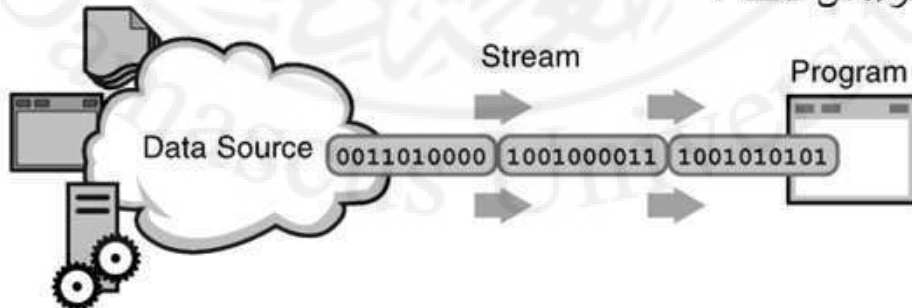
وهو المجرى الثالث ويعمل على تجميع هذه الحروف أو الرموز في ذاكرة مؤقتة لعمل سلسلة منها (String) . والصيغة العامة له كما يلي:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

هنا الغرض (القارئ) in من الصف BufferedReader والصف InputStreamReader له المعامل InputStreamReader(parameter) والصف InputStreamReader له المعامل System.in وهو صف لقراءة بايت واحد. وتكوّن مع بعضها كما في الصيغة السابقة مجرى لقراءة حرف أو سلسلة من الأحرف ونلاحظ هنا أن الغرض (in) في BufferedReader هو مجرد اسم يمكن تغييره .

- القراءة من ملف

في هذه الحالة يقوم البرنامج بقراءة واستدعاء معطيات موجودة مسبقا في ملف ليقوم البرنامج بالتعامل معها . والشكل (٦-٤) يوضح كيف يقوم البرنامج بعملية القراءة من الملف .



الشكل (٦-٤) - يوضح كيف يقوم البرنامج بعملية القراءة من الملف

عبارة Try And Catch

عند عمليات القراءة والكتابة في لغة Java تحصر أوامر القراءة والكتابة بين الكلمتين try و catch ، وتعني جرب القراءة أو الكتابة ، وإذا لم تنجح فأمسك الخطأ الذي تسبب في عدم إنجاح العملية ، وهذه الأخطاء تسمى في لغة Java الاستثناءات (Exception) ، وهناك أنواع عديدة منها مثل IOException وتعني خطأ في الإدخال والإخراج . ولكي نفهم هذه العبارة جيداً نجري المثال التالي :

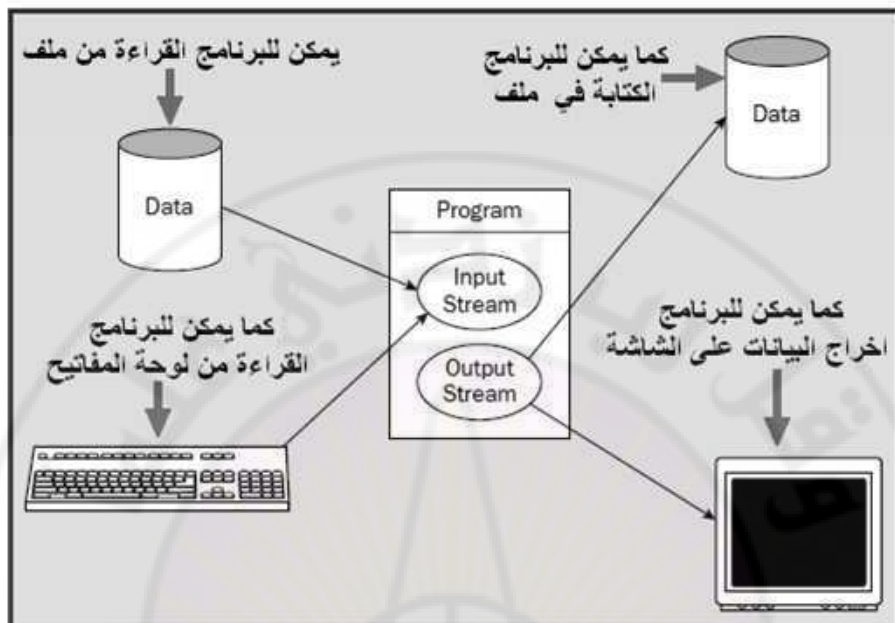
مثال :

```
try{
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String s = in.readLine(); // اقرأ سطر من لوحة المفاتيح
} catch(Exception e){
    .....ماذا نريد أن يفعل إذا كان هناك خطأ في العملية.....
}
```

حيث (e) هو خطأ عام أي لم نحدد أي نوع من الأخطاء التي يجب أن يمسك بها البرنامج . وطبعا الحرف (e) يمكن تغييره بحروف أخرى تعبر كذلك عن نوعية أخطاء أخرى . ويمكن طباعة الخطأ على الشاشة كما يلي :

```
try{
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String s = in.readLine();
} catch(Exception e){
    System.out.print(e); // طبع لنا الخطأ بالضبط
}
```

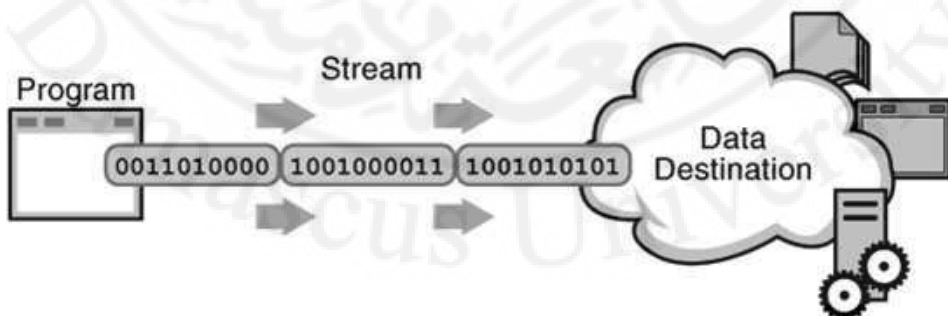
ويوضح الشكل (٦-٥) يوضح مفهوم المجاري في عملية القراءة والكتابة في الملف .



الشكل (٦-٥) - يوضح مفهوم المجاري في عملية القراءة والكتابة في الملف

ثانياً - الكتابة في ملف

وهي عبارة عن كتابة (خزن) مخرجات البرنامج في جهاز الحاسوب في ملف موجود . والشكل (٦-٦) يبين كيفية الكتابة في ملف .



الشكل (٦-٦) - يبين كيفية الكتابة في ملف

٦-٦-٦- التعامل مع ملفات النصوص Text Files

الملفات النصية : وهي الملفات التي تحتوي على رموز ونصوص فمثلا الملف الذي يحتوي على برنامج Java يُعدّ ملف نصي وملفات وورد "Word" أو نوت باد "notepad" جميع هذه الملفات ستُكتب على شكل سلسلة String من المحارف. وسوف ندرس هنا كيفية الكتابة والقراءة من ملفات النصوص . وهناك عدة طرائق لكيفية الكتابة والقراءة في الملفات وسنتناول إحدى هذه الطرائق .

٦-٦-٦-١- الكتابة في ملفات النصوص

Writing To a Text File

- لكي تتم عملية الكتابة في الملف النصي نقوم بثلاث خطوات كما يلي :
١. فتح ملف موجود للكتابة أو إنشاء ملف جديد ومنحه اسم name للتعامل معه في بيئة Java .
 ٢. استخدام Stream Class للتعامل مع الملف والكتابة فيه .
 ٣. إنشاء غرض Object يمكننا من الكتابة في الملف واستخدامه في كل مرة .
- نطبق الخطوات السابقة كما يلي :

□ استخدام الصف (PrintWriter)

ينتمي هذا الصف إلى فئة الصفوف ذات المجاري Stream والتي تقوم بعملية الكتابة في ملفات النصوص . وهو يشبه استخدام تابع الطباعة System.out.println المستخدم في طباعة المخرجات على الشاشة . وعند استخدام الصف (PrintWriter) سوف تذهب مخرجات البرنامج إلى ملف نصوص . وترتبط صفوف الإدخال والإخراج بالمكتبة (java . io) ولذلك يجب استدعاء هذه المكتبة في بداية البرنامج بالعبار (import) ، فتصبح العبارة كاملة . import java.io

والصيغة العامة لاستخدام هذا الصف وتنفيذ النقاط الثلاثة للكتابة في الملف تكون كما يلي :

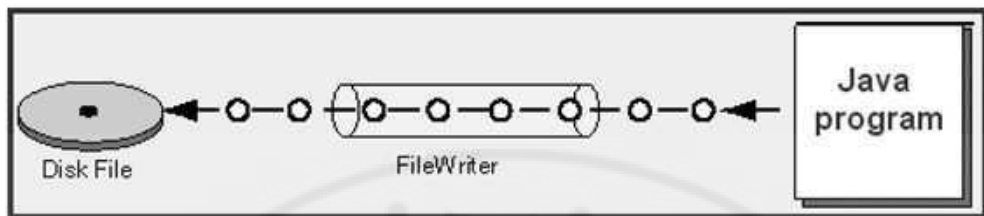
```
PrintWriter ← الإعلان عن متغير من نوع
1 PrintWriter Output_Stream_Name;
2 Output_Stream_Name =
    new PrintWriter(new FileOutputStream(File_Name));
```

اسم الملف المخزن على الحاسب

في السطر الأول تم الإعلان عن متغير من النوع `PrintWriter` . وفي السطر الثاني تم إنشاء غرض `object` وعمل مجرى `Stream` للتعامل مع الملف والكتابة فيه . كما هو موضح في المثال التالي :

```
1 PrintWriter outputStream = null;
2 outputStream =
    new PrintWriter(new FileOutputStream("stuff.txt"));
```

في السطر الأول تم التصريح عن متحول اسمه (`outputStream`) وتم إسناد القيمة (`null`) له للتأكد من أن هذا المتحول لا يحتوي على شيء . وفي السطر الثاني تم إنشاء غرض `object` ، وتحديد مجرى `Stream` للكتابة والتخزين في ملف (`stuff.txt`) ، ويجب تحديد مجرى هذا الملف على القرص المغناطيسي . يوضح الشكل (٦-٧) طريقة الكتابة في الملف .



الشكل (٦-٧) - يوضح طريقة الكتابة في الملف

وبهذا الشكل يقوم برنامج Java بالكتابة على القرص الصلب عن طريق
المجرى الذي تم تحديده .

مثال (٦-٢٢)

```

1. import java.io.PrintWriter;
2. import java.io.FileOutputStream;
3. import java.io.FileNotFoundException;
4. class TextFile {
5. public static void main( String [ ]args) {
6. PrintWriter output = null ;
7. try {
8. output = new PrintWriter ( new FileOutputStream( "D:\\
stuff.txt")); ;
9. }
10. catch( FileNotFoundException e) {
11. System.out.println(" Errorr the Opening the file stuff.txt ");
12. System.exit(0);
13. }
14. System.out.println(" writing to file ");
15. output.println(" The quick brown fox ");
16. output.println("jumped over the lazy dog. ");
17. output.close( );
18. output.println(" end of program ");
19. } // end main

```


20. } //end class

Result

The quick brown fox
jumped over the lazy dog.

Process completed.

الأسطر من (1 إلى 3) استخدام الصفوف الموجودة في الحزمة (Java.io) .
والتي تحتوي على صفوف الإدخال والإخراج .

تستخدم العبارة (try-catch) في السطر 7 والسطر 10 لحصر الأخطاء
داخل البرنامج ، ونلاحظ أن كلاً من العبارة (try) لها قوسين بداية ونهاية ،
وتوضع داخلها عبارة فتح الملف ، ومعناه (حاول فتح الملف فإذا وجدت أي خطأ
قم بمسكه catch) . ونفذ ما بين قوسي العبارة catch وهنا طباعة العبارة :

(Error opening the file stuff.txt)

أي حدث خطأ عند فتح الملف (stuff.txt) . ويقوم بطباعة العبارة (Writing
to file) في حالة عدم وجود أخطاء (السطر 14) . أما تابع الطباعة println
الموجود في الغرض output يقوم بطباعة النصوص الموجودة داخل الأقواس في
الملف stuff.txt. العبارة (); outputStream.close() وتقوم هذه العبارة بإغلاق
المجاري المفتوحة Stream لهذا الملف . فإذا لم يتم كتابة هذه العبارة تقوم لغة
Java بإغلاق الملف . ولكنه من الأمن برمجياً أن نقوم بكتابة هذه العبارة .

السطر 18: output.println(" end of program "); يقوم بطباعة العبارة
(End of program) وذلك عند الانتهاء من الكتابة في الملف بعد ذلك يتم إنهاء
البرنامج .

ولابد من تحديد المسار، الذي سنقوم بحفظ هذا الملف عليه في القرص الصلب ويتم تحديد ذلك كما هو مبين في السطر 8 ، ونفصل بين المسارات بشرطتين مائلتين ، ولقراءة الملف ، الذي تم عمله بواسطة البرنامج السابق نذهب إلى المسار الذي حددناه لحفظ الملف ، وسنجد إنه تم عمل ملف باسم (stuff.txt) يمكننا فتحه بأي محرر نصوص ، مثل برنامج word أو برنامج المفكرة ، وسنجد داخل هذا الملف أنه تم كتابة مخرجات البرنامج . لإضافة معطيات جديدة دون مسح المعطيات في الملف القديم نستخدم التعبير التالي :

output =

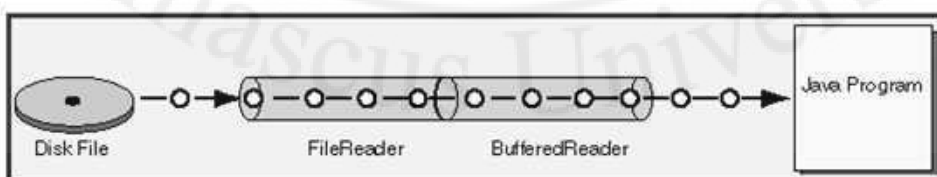
```
new PrintWriter ( new FileOutputStream( "D:\\ stuff.txt", mod));
```

هذه العبارة الموجودة في السطر (8) ينقصها معامل آخر وهو (mode) . فإذا كان هذا المعامل (true) حينئذ يمكننا إضافة المعطيات مع الحفاظ على المعطيات السابقة . أما إذا كانت (false) فإنه يتم مسح المعطيات في الملف القديم واستبداله بمعطيات جديدة .

٦-٦-٢ - القراءة من ملفات النصوص

Reading From a Text File

المقصود بعملية القراءة (أن يقوم البرنامج بقراءة محتويات ملف نصي سبق تخزينه) وإن عملية القراءة عكس عملية الكتابة والشكل (٦-٨) يوضح هذه العملية .



الشكل (٦-٨) - يوضح عملية القراءة

يوجد طريقتان شائعتان يتم استخدامهما للقراءة من الملف .

❑ الطريقة الأولى باستخدام الصف Scanner .

وسبق أن تعرفنا على الصف Scanner في عملية الإدخال من لوحة المفاتيح .

□ الطريقة الثانية باستخدام الصف `BufferedReader`.

وتُستخدم هذه الطريقة للقراءة من الملف النصي ، والصيغة العامة للعبارة
BufferedReader هي :

```
BufferedReader Stream_Object =  
    new BufferedReader(new FileReader(File_Name));
```

اسم ومسار الملف
الذي ستتم القراءة منه

مثلاً :

```
BufferedReader input =  
    new BufferedReader(new FileReader("stuff.txt"));
```

ويوضح المثال (6 - 23) قراءة محتويات الملف السابق الذي تمت كتابته باسم (stuff.txt) .

مثال (23-6)

1. **import** java.io.*;
2. **class** TextFilein {
3. **public static void** main(**String** []args) {
4. **try**{
5. BufferedReader input = **new** BufferedReader(
 new FileReader("D:\\stuff.txt"));
6. **String** Line = input.readLine();
7. System.out.println(" The First Line Read From the File is: ");
8. System.out.println(Line);

```

9. Line = input.readLine( );
10. System.out.println(" The Second Line Read From the File is: ");
11. System.out.println(Line);
12. input.close( );
13. }
14. catch( IOException e) {
15. System.out.println(" Error Reading From stuff.txt ");
16. System.exit(0);
17. }
18. } //end main
19. } //end class

```

السطر 8 تم استخدام الصف `BufferedReader` يقوم بالقراءة من الملف `stuff.txt`.

السطر 6 تم استخدام العبارة `input.readLine( )` لقراءة سطر واحد من الملف `stuff.txt`. وتم تخزين القيمة المقروءة في متحول `(Line)` من نوع `String` حيث تتبع العبارة `readLine()` للصف `BufferedReader` حيث تم إنشاء `object` هو `input`.

السطر 7 يقوم بطباعة رسالة تُشير إلى السطر الأول الذي تمت قراءته.

السطر 8 يطبع القيمة المخزنة في المتحول `Line` وهو السطر الأول في الملف.

السطر 12 يتم إغلاق مجرى الإدخال للملف.

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

The First Line Read From the File is :

welcome to you

The Second Line Read From the File is :

Hello Mr. Mamoun

Process completed.

مسائل عامة

١- ما هو نتيجة تنفيذ البرنامج الآتي :

```
import java.io.*;

public class magdy{

    public static void main(String[] args) throws IOException
    {
        String s="magdymohamed";

        char x = s.charAt(4);
        System.out.println( x);
    }
}
```

(y) -D

(d) -C

(a) -B

(4) -A

٢- أي العبارات الآتية تشير إلى إنشاء متحول من نوع String

- ☐ A. String s = "Welcome to Java";
- ☐ B. String s = new String("Welcome to Java");
- ☐ C. String s; s = "Welcome to Java";
- ☐ D. String s; s = new String("Welcome to Java");

٣- ما هي مخرجات البرنامج التالي :

```
public class Test {
    public static void main(String[] args) {
        String s1 = "Welcome to Java!";
        String s2 = s1;

        if (s1 == s2)
            System.out.println("s1 and s2 reference to the same String object");
        else
            System.out.println("s1 and s2 reference to different String objects");
    }
}
```

- كلا من s1 و s2 تشير الى نفس الشيء
- كلا من s1 و s2 تشير الى شيئين مختلفين

٤- ما هي مخرجات البرنامج التالي :

```
public class Test {
    public static void main(String[] args) {
        String s1 = "Welcome to Java!";
        String s2 = "Welcome to Java!";

        if (s1 == s2)
            System.out.println("s1 and s2 reference to the same String object");
        else
            System.out.println("s1 and s2 reference to different String objects");
    }
}
```

- A. s1 and s2 reference to the same String object
- B. s1 and s2 reference to different String objects

٥- ما هي ناتج طباعة البرنامج التالي :

```
public class Test {
    public static void main(String[] args) {
        String s1 = new String("Welcome to Java!");
        String s2 = new String("Welcome to Java!");

        if (s1 == s2)
            System.out.println("s1 and s2 reference to the same String object");
        else
            System.out.println("s1 and s2 reference to different String objects");
    }
}
```

- A. s1 and s2 reference to the same String object
- B. s1 and s2 reference to different String objects

٦- لنفرض أن (s1) و (s2) متحولين من نوع String ، ما هي نتيجة العملية الآتية :

s1.equals(s2) == s2.equals(s1)

- ☐ A. true
- ☐ B. false

٧- ما هي مخرجات البرنامج التالي :

```
public class Test {  
    public static void main(String[] args) {  
        String s1 = new String("Welcome to Java!");  
        String s2 = new String("Welcome to Java!");  
  
        if (s1.equals(s2))  
            System.out.println("s1 and s2 have the same contents");  
        else  
            System.out.println("s1 and s2 have different contents");  
    }  
}
```

- ☐ A. s1 and s2 have the same contents
- ☐ B. s1 and s2 have different contents

٨- ما هي مخرجات البرنامج التالي :

```
public class Test {  
    public static void main(String[] args) {  
        String s1 = new String("Welcome to Java!");  
        String s2 = s1.toUpperCase();  
  
        if (s1 == s2)  
            System.out.println("s1 and s2 reference to the same String object");  
        else if (s1.equals(s2))  
            System.out.println("s1 and s2 have the same contents");  
        else  
            System.out.println("s1 and s2 have different contents");  
    }  
}
```

- ☐ A. s1 and s2 reference to the same String object
- ☐ B. s1 and s2 have the same contents
- ☐ C. s1 and s2 have different contents

٩- لنفرض أن (s1) و (s2) متحولان من نوع String ، أي العبارات التالية صحيحة (يمكن أن يكون أكثر من اختيار) :

- ☐ A. String s = new String("new string");
- ☐ B. String s3 = s1 + s2
- ☐ C. s1 >= s2
- ☐ D. int i = s1.length
- ☐ E. s1.charAt(0) = '5'

١٠- ما هي نتيجة المقارنة التالية :

"abc".compareTo("aba")

- ☐ A. 1
- ☐ B. 2
- ☐ C. -1
- ☐ D. -2
- ☐ E. 0

١١- ما هي نتيجة المقارنة التالية :

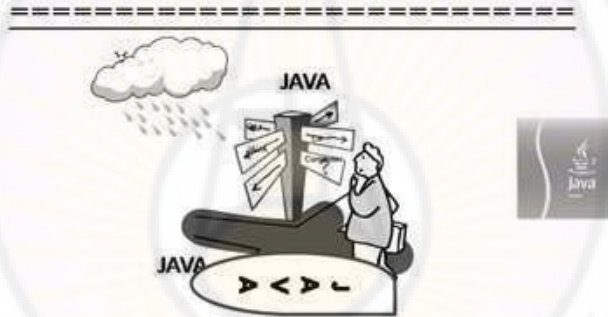
"AbA".compareToIgnoreCase("abC")

- ☐ A. 1
- ☐ B. 2
- ☐ C. -1
- ☐ D. -2
- ☐ E. 0

١٢- ما هي نتيجة تنفيذ مجموعة التعليمات التالية :

```
String s = "University";  
s.replace("i", "ABC");  
System.out.println(s);
```

- ☐ A. UnABCversity
- ☐ B. UnABCversABCTy
- ☐ C. UniversABCTy
- ☐ D. University





الفصل السابع

الوراثة

Inheritance





الوراثة

Inheritance

٧-١- مقدمة

تُعدّ الوراثة إحدى أشكال إعادة استخدام النصوص البرمجية ، حيث يتم إنشاء صفوف جديدة انطلاقاً من صفوف موجودة مسبقاً ، وتأخذ الصفوف الجديدة صفات وتصرفات الصفوف البرمجية القديمة ، وتضيف إليها إمكانيات جديدة تحتاجها . وتساعد عملية إعادة استخدام النصوص البرمجية على توفير الوقت اللازم لتطوير البرامج ، وتشجع على استخدام برمجيات موثوقة وفعالة ، كما أنها تقلل من الإشكاليات الناتجة عن وضع البرمجيات قيد الاستخدام الفعلي .

وتسمى الصفوف الموجودة مسبقاً بالصفوف الأساسية Super Class وتسمى الصفوف المشتقة عنها بالصفوف المشتقة Sub Class . وتمثل الصفوف المشتقة أغراضاً أكثر تخصصاً من أغراض الصف الأساسي ، كما أنها تتضمن توابع وإجرائيات مخصصة لها ، ونسمي الصف الأساسي الذي يجري اشتقاق صف عنه بالصف الأساسي المباشر Direct Suber Class ، أما إذا كان لدينا أكثر من مستوى للوراثة بين الصف الأساسي والصف المشتق فإننا نسمي الصف الأساسي غير المباشر Indirect Suber Class .

تدعم لغة Java الوراثة المتعددة غير المعقدة Multiple Inheritance التي يقوم فيها الصف المشتق بالوراثة من صف أساسي واحد .

٧-٢- الصفوف الأساسية والصفوف المشتقة

نعرف الوراثة " هي الآلية التي بواسطتها يمكن لصف ما أن يرث خصائص صف آخر " ، وتسمح الوراثة ببناء بنية هرمية من الصفوف ابتداءً من الصف الأكثر

عمومية وانتهاءً بالصف الأكثر خصوصية . يجب تعريف مصطلحين مستخدمين بكثرة في الوراثة:

□ يدعى الصف الذي يورث بالصف الأساسي Super Class .

□ ويدعى الصف الذي يرث بالصف المشتق Sub Class .

ويعرّف الصف الأساسي (السابر) Suber Class جميع الخواص المشتركة لأي من الصفوف المشتقة أي يقدم وصفاً عاماً لمجموعة من الخصائص ، ويرث الصف المشتق (الفرعي) Sub Class الخصائص العامة من الصف الأساسي ويضيف إليها خصائص خاصة به . و توفر لغة Java وسيلة لتقليل حجم البرنامج حيث يستفاد من بعض التوابع في بعض الصفوف ، وهذا يعني اشتقاق صفوف عديدة من صفوف محددة ثم إضافة الموصفات الإضافية الخاصة بالصف المشتق ، وبهذه الطريقة لا نحتاج إلى إعادة كتابة التوابع مرة أخرى . ومن الممكن معاملة أغراض الصف الأساسي وأغراض الصف المشتق بشكل متشابه ، حيث يتوفر الجزء المشترك بينهما ضمن تعريف أعضاء الصف الأساسي ، كما يمكن اعتبار أغراض كافة الصفوف المشتقة كأنها أغراض تابعة للصف الأساسي .

٧-٢-١ - التحكم بالوصول إلى الصف الأساسي

Suber Class Access Control

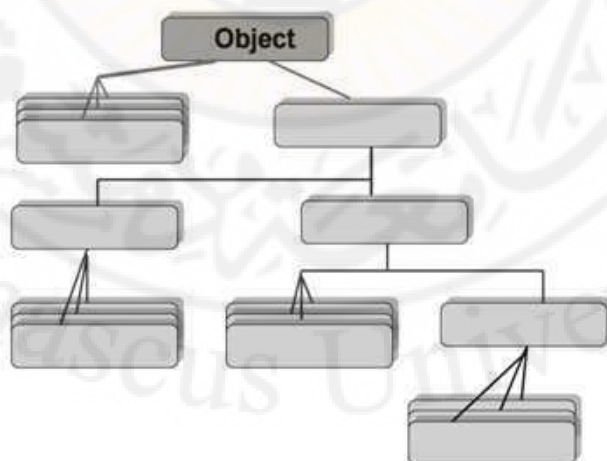
عند وراثة صف من صف آخر نستخدم الشكل العام التالي من أجل تعريف الصف المشتق وربطه مع الصف الأساسي :

```
class derived_Class_Name : access Base_Class_Name {  
    .....class Body .....  
}; // End derived Class
```

إذا كان الصف لا يمتلك محدد الوصول extends فإن لهذا الصف صف أساسي (superclass) هو Object ، لذلك نقول بأن كل صف في Java له على الأقل صف أساسي واحد مباشر .

يوجد في جافا صف يُعدّ هو الصف الأساسي الأعلى (أعلى صف في الشجرة الهرمية) نسميه الصف Object ، وكل صف آخر يُعدّ صف مشتق له بشكل مباشر أو غير مباشر . إن أي متحول من نمط الصف Object يمكن أن يحوي مرجعاً على أي غرض آخر ، مثل غرض من صف أو مصفوفة .

الصف Object يوفر مجموعة من التتابع Methods اللازمة لجميع الأغراض التي تعمل على آلة جافا الافتراضية java virtual machine ، فمثلاً جميع الصفوف ترث التابع toString التابعة للصف Object والتي تعيد سلسلة String تمثل شكل مطبوع للغرض . إذاً الصف Object هو الصف الأعلى في بنية الشجرة الهرمية ، هذا الصف تُشتق منه جميع الصفوف الأخرى وهذا الصف موجود في الحزمة java.lang ، ويوضح الشكل (٧-١) هذا المفهوم .



الشكل (٧-١) - يوضح الصف الأساسي الأعلى Object

إن الصف Object يُعرف ويحقق مجموعة من التوابع (Methods) التي يحتاجها كل صف في نظام java . ويمكننا القول كلما اتجهنا من أعلى إلى أسفل شجرة الصفوف السابقة فإننا سنحصل على صفوف أكثر تخصيصاً في سلوكها . الحزمة java.lang تتضمن الصفوف الأكثر استخداماً في java ، وهذه الحزمة مهمة جداً لدرجة أنه يتم استيرادها بشكل افتراضي حتى ولو لم تجلبها بنفسك .

٧-٢-٢ - الصفوف المشتقة

Sub Classes

يمكن في لغة Java أن يرث صف مشتق subclass الصف الأساسي superclass بواسطة المحدد extends . ويرث الصف المشتق جميع المعطيات الأعضاء والتوابع الأعضاء الموجودة في الصف الأساسي والتي يمكن الوصول إليها من خلال الصف المشتق ، ولا يمكن للصف المشتق الوصول إلى الأعضاء الخاصة private في الصف الأساسي فهي غير مرئية له . يستطيع الصف المشتق الوصول إلى الأعضاء العامة public والمحمية protected والتي من نوع package . يمكن تصنيف الأعضاء التي يمكن وراثتها كما يلي :

١. ترث الصفوف المشتقة من الصف الأساسي الأعضاء التي لها المحدد

public و protected .

٢. ترث الصفوف المشتقة من الصف الأساسي الأعضاء التي لم يعين لها

نوع محدد الوصول (package) طالما أن الصفوف المشتقة في الحزمة نفسها .

٣. لا ترث الصفوف المشتقة من الصف الأساسي التوابع البانية لأنها

ليست أعضاء .

٤. لا تترث الصفوف المشتقة عضواً من الصف الأساسي إذا صرح الصف المشتق عن عضو له الاسم نفسه . ففي حالة المتحولات العضوية ، فإن المتحول العضوي في الصف المشتق سيخفي المتحول العضوي الذي له الاسم نفسه في الصف الأساسي ، وفي حالة التتابع فإن التابع في الصف المشتق يعيد صياغة - كتابة- التابع الذي له الاسم نفسه في الصف الأساسي .
يوضح المثال (7 - 1) البسيط التالي عملية الوراثة .

مثال (7 - 1)

// A simple example of inheritance. Create a superclass.

```
public class ClassA {  
    int i, j;  
    void showij() {  
        System.out.println("i and j: " + i + " " + j);  
    }  
} //End ClassA
```

// Create a subclass by extending class A.

```
public class ClassB extends ClassA {  
    int k;  
    void showk() {  
        System.out.println("k: " + k);  
    }  
    void sum() {  
        System.out.println("i+j+k: " + (i+j+k));  
    }  
} //End ClassB
```

```
public class SimpleInheritance {  
    public static void main(String args[] ) {
```

```

ClassA  superOb = new ClassA( );
ClassB  subOb = new ClassB( );
// The superclass may be used by itself.
superOb.i = 10;
superOb.j = 20;
System.out.println("Contents of superOb: ");
superOb.showij( );
// The subclass has access to all public members of its superclass.
subOb.i = 7;
subOb.j = 8;
subOb.k = 9;
System.out.println("Contents of subOb: ");
subOb.showij( );
subOb.showk( );
System.out.println( );
System.out.println("Sum of i, j and k in subOb:");
subOb.sum( );
} // End main
} // End SimpleInheritance

```

وعند تنفيذ البرنامج يظهر الخرج التالي :

Result

Contents of superOb:

i and j: 10 20

Contents of subOb:

i and j: 7 8 k: 9

Sum of i, j and k in subOb:

i+j+k: 24

Process completed.

نلاحظ من المثال السابق أن الصف الأساسي ClassA لديه متحولي أعضاء هما j , i لم يحدد نوع محدد الوصول فيُعدّ محدد الوصول من نوع package (أي افتراضي) ويستطيع الصف المشتق ClassB الوصول إلى هذه المعطيات ، كما هو مبين في البرنامج السابق . أما في الصف SimpleInheritance يستطيع التابع main أن يسند قيم إلى المتحولات الأعضاء في الصف الأساسي والصف المشتق لأن المتحولات i , j , k هي من نوع package ، وبالتالي يستطيع أي صف ضمن الحزمة الوصول إلى هذه المعطيات . وتدعم الحزمة package حقوق الوصول الافتراضية التي تطبق على الصفوف والواجهات والمتحولات والتوابع .

ماذا يحصل إذا حددنا في الصف الأساسي ClassA المتحول z كمتحول خاص تابع تعديل معطيات ؟
نعيد كتابة البرنامج كما هو موضح في المثال (2-7) .

مثال (2-7)

// A simple example of inheritance. Create a superclass.

```
public class ClassA {
    private int j ; // private to ClassA
    void setij(int x, int y) {
        i = x;
        j = y;
    }
    void showij() {
        System.out.println("i and j: " + i + " " + j);
    }
} //End ClassA
```

// ClassA's j is not accessible here.

```
public class ClassB extends ClassA {  
private int k;  
void set1( int x , int y ,int z )  
{  
    setij( x,y);  
    k =z;  
}  
void showk( ) {  
    System.out.println("k: " + k);  
}  
void sum( ) {  
    System.out.println("i+j+k: " + (i+j+k));  
} // ERROR, j is not accessible here  
}//End ClassB  
public class SimpleInheritance {  
public static void main(String args[] ) {  
    ClassA superOb = new ClassA( );  
    ClassB subOb = new ClassB( );  
  
    subOb.set1(10, 12,8);  
    System.out.println("Contents of superOb: ");  
    superOb.showk( );  
}//End main  
}//End SimpleInheritance
```

□ في هذا البرنامج يحاول الصف المشتق ClassB الوصول إلى العضو الخاص ز في الصف الأساسي . وهذا خطأ لأن العضو الخاص في الصف الأساسي يبقى خاصاً به مهما كان الشكل الذي تمت به عملية الوراثة .

و يوضح هذا المثال بأننا لا نستطيع الوصول إلى المعطيات الخاصة وسوف يعطينا خطأ ERROR .

٧-٣- بناء الصف Student

لدينا الصف الأساسي student يحتوي على المتحولات الخاصة التالية :

- ❖ First الاسم الأول ، mid الاسم الوسط ، last الاسم الأخير ، id الرقم الجامعي .
- ❖ ويحتوي الصف على بانٍ عادي وتابع طباعة .

و لدينا الصف المشتق Course الذي يرث الصف الأساسي Student ويحتوي على المتحولات الخاصة التالية :

- ❖ cid القسم ، description اسم المقرر ، grade النتيجة في المقرر .
- ❖ ويحتوي الصف على بانٍ عادي وتابع طباعة .

أما البرنامج له الشكل التالي .

مثال (3-7)

```
public class Student {  
    private String first ,mid, last,id;  
    // constructor  
    public Student( String f, String m, String l, String i ) {  
        first = f;  
        mid = m ;  
        last = l;  
        id = i ;  
    } // end constructor  
    public void printstudent( ) { // print()  
        System.out.printf("%s %s %s %s \n", first ,mid ,last,id );  
    } // end function print  
} // end class student
```



```

public class Course extends Student {
private String cid , description, grade ;
public Course ( String f ,String m ,String l,String i ,
String ci ,String cd ,String g)
{
    super( f, m, l, i ) ;
    cid = ci ;
    description = cd;
    grade = g;
} //end constructor
public void showgrade( ) // print( )
{
    printstudent( );           // super. print( )
    System.out.printf("%s %s %s " ,cid ,description ,grade ) ;
}
} // End class course

public class StudentTest {
public static void main(String args[ ] ) {
    Course mystudent1 = new Course (" Ahmad", "Saleem", "Ali",
        "012345678", "CIS 25c","Pogramming in JAVA "," A+");
    mystudent1.showgrade( ) ;
} // end main
} // end StudentTest

```

وعند تنفيذ البرنامج يظهر الخرج التالي :

Result

Ahmad Saleem Ali 012345678

CIS 25c Pogramming in JAVA A+

Process completed.

الاستخدام الأول للكلمة المحجوزة super

يتم استخدام الكلمة المحجوزة super لاستدعاء باني الصف الأساسي من الصف المشتق وتميرير الوسطاء له . إذا لم يكن هناك استدعاء صريح لباني الصف الأساسي فإنه يتم القيام باستدعاء آلي لباني الصف الأساسي الذي ليس له وسطاء

نعيد كتابة البرنامج من أجل إدخال المعطيات عن طريق واجهة وطباعة النتيجة أيضاً على واجهة .

```
import javax.swing.JOptionPane;
public class Student {
    private String first ,mid, last,id;
    // constructor
    public Student( String f, String m, String l, String i ) {
        first = f;
        mid = m ;
        last = l;
        id = i ;
    } // end constructor
    public String printstudent( ) {
        String output =" ";
        output = output +first+ " " +mid + " " +last+ " " +id+ "\n" ;
        return output ;
    } // end function print
} // end class student

public class Course extends Student {
    private String cid , description, grade ;
    public Course ( String f ,String m ,String l
                    ,String i ,String ci ,String cd ,String g)
{
```

```

    super( f, m, l, i );
    cid = ci ;
    description = cd;
    grade = g;
} //end constructor
public String showgrade( ) {
    String output= printstudent( );
    output= output + cid+ " " +description + " " + grade+ "\n";
return output;
}
} // End class course

public class StudentTest {
public static void main(String args[ ] ) {
    String first = " ", mid=" ", last= " ", id = " "+cid = " ",
                                description=" ", grade= " ";
    String javaString = JOptionPane.showInputDialog(null,
        "Enter a first Student ", "My Student Example",
        JOptionPane.QUESTION_MESSAGE);
    first=javaString;
    javaString = JOptionPane.showInputDialog(null,
        "Enter a mid Student ", "My Student Example",
        JOptionPane.QUESTION_MESSAGE);
    mid=javaString;
    javaString = JOptionPane.showInputDialog(null,
        "Enter a last Student ", "My Student Example",
        JOptionPane.QUESTION_MESSAGE);
    last=javaString;
    javaString = JOptionPane.showInputDialog(null,
        "Enter a id Student ", "My Student Example",
        JOptionPane.QUESTION_MESSAGE);

```

```

id=javaString;
javaString = JOptionPane.showInputDialog(null,
    "Enter a cid Student ", "My Student Example",
    JOptionPane.QUESTION_MESSAGE);
cid=javaString;
javaString = JOptionPane.showInputDialog(null,
    "Enter a description Student ", "My Student Example",
    JOptionPane.QUESTION_MESSAGE);
description=javaString;
javaString = JOptionPane.showInputDialog(null,
    "Enter a grade Student ", "My Student Example",
    JOptionPane.QUESTION_MESSAGE);
grade=javaString;
Course mystudent1 = new Course( first ,mid,last,
                                id,cid , description, grade) ;

//display My Student Class
String output = mystudent1.showgrade( );
JOptionPane.showMessageDialog(null, output, "My Student
Class ", JOptionPane.ERROR_MESSAGE );
} // end main
} // end StudentTest

```

و عند تنفيذ البرنامج يظهر الخرج التالي :

سوف نكتفي بواجهة الإخراج .



٧-٤ - بناء الصف Box (وراثه)

يحتوي البرنامج التالي على الصفوف التالية :

١. يحتوي على صف أساسي superclass اسمه Box يقوم بحساب حجم الصندوق الذي هو عبارة عن متوازي مستطيلات أبعاده , height , width , depth وهي متحولات خاصة private من نمط double ، يحتوي الصف على بانٍ افتراضي لإعطاء القيم الابتدائية وعلى بانٍ عادي لإسناد المعطيات للأعضاء الخاصة وعلى بانٍ بوسيط واحد لإعطاء قيمة واحدة للمتحولات ليتحول الشكل إلى مكعب ، وعلى بانٍ يستعمل الأغراض كوسطاء له ، وعلى تابع volume يقوم بحساب حجم متوازي المستطيلات (حجم الصندوق) .
٢. يحتوي البرنامج على صف مشتق subclass اسمه BoxWeight يرث الصف الأساسي superclass ، ويحتوي على متحول واحد هو وزن الصندوق Weight وهو من نمط double ، ويحتوي على بانٍ افتراضي وعلى بانٍ عادي وعلى بانٍ وسيطه غرض ، وعلى بانٍ يستخدم عندما يصبح الصندوق مكعباً .
٣. يحتوي البرنامج على صف اختبار اسمه DemoSuper يحتوي على التابع main لإنشاء الأغراض واستدعاء التوابع اللازمة والحصول على النتائج .

مثال (4-7)

```
// A complete implementation of BoxWeight.  
public class Box {  
  private double width;  
  private double height;  
  private double depth;  
  // construct clone of an object  
  public Box(Box ob) { // pass object to constructor
```



```

width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
public Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
// constructor used when no dimensions specified
public Box() {
width = 1; // use -1 to indicate an un initialized box
height = 1;
depth = 1;
}
// constructor used when cube is created
public Box(double len) {
width = height = depth = len;
}
// compute and return volume
double volume() {
return width * height * depth;
}
} //End Class Box

// BoxWeight now fully implements all constructors.
public class BoxWeight extends Box {
double weight; // weight of box
// construct clone of an object
public BoxWeight(BoxWeight ob) { // pass object to constructor

```

```

super(ob);
weight = ob.weight;
}
// constructor when all parameters are specified
public BoxWeight( double w, double h, double d, double m) {
super(w, h, d); // call superclass constructor
weight = m;
} // default constructor
public BoxWeight( ) {
super( );
weight = 1;
}
// constructor used when cube is created
public BoxWeight(double len, double m) {
super(len);
weight = m;
}
} //End BoxWeight

public class DemoSuper {
public static void main(String args[ ] ) {
BoxWeight mybox1 = new BoxWeight(10,20,15,34.3);
BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
BoxWeight mybox3 = new BoxWeight( ); // default
BoxWeight mycube = new BoxWeight(3, 2); //cube
BoxWeight myclone = new BoxWeight(mybox1);
double vol;
vol = mybox1.volume( );
System.out.println("Volume of mybox1 is " + vol);
System.out.println("Weight of mybox1 is " + mybox1.weight);
System.out.println( );
vol = mybox2.volume( );

```

```
System.out.println("Volume of mybox2 is " + vol);
System.out.println("Weight of mybox2 is " + mybox2.weight);
System.out.println( );
vol = mybox3.volume( );
System.out.println("Volume of mybox3 is " + vol);
System.out.println("Weight of mybox3 is " + mybox3.weight);
System.out.println( );
vol = myclone.volume( );
System.out.println("Volume of myclone is " + vol);
System.out.println("Weight of myclone is " + myclone.weight);
System.out.println( );
vol = mycube.volume( );
System.out.println("Volume of mycube is " + vol);
System.out.println("Weight of mycube is " + mycube.weight);
System.out.println( );
} //End main
} //End DemoSuper
```

و عند تنفيذ البرنامج يظهر الخرج التالي :

Result

Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3
Volume of mybox2 is 24.0
Weight of mybox2 is 0.076
Volume of mybox3 is 1.0
Weight of mybox3 is 1.0
Volume of myclone is 3000.0
Weight of myclone is 34.3
Volume of mycube is 27.0
Weight of mycube is 2.0

Process completed.

السؤال : ما لتعديلات التي يمكن أجراءها على الصف المشتق BoxWeight عندما نصرح عن المتحول weight كمتحول private .

٧-٥- تجاوز التابع

Method overriding

يتم استخدام الكلمة المحجوزة super من أجل الوصول إلى التوابع الموجودة في الصف الأساسي والمعاد تعريفها من قبل الصفوف المشتقة . وكما ذكرنا سابقاً ، المتحولات الأعضاء المعرفة في الصف المشتق تُخفي المتحولات الأعضاء في الصف الأساسي والتي لها الاسم نفسه . ويستطيع الصف المشتق الوصول إلى المتحولات المخفية أو التوابع المخفية في الصف الأساسي باستخدام الكلمة المحجوزة super (أي تجاوز التابع) وهو الاستخدام الثاني للكلمة المحجوزة super ، كما هو موضح في المثال (5-7) التالي .

مثال (5-7)

// Using super to overcome name hiding.

```
public class ClassA {
    int x ;
    void show( ) {
        System.out.printf(" show( ) in superclass : %d\n" , x* x);
    }
} //End ClassA

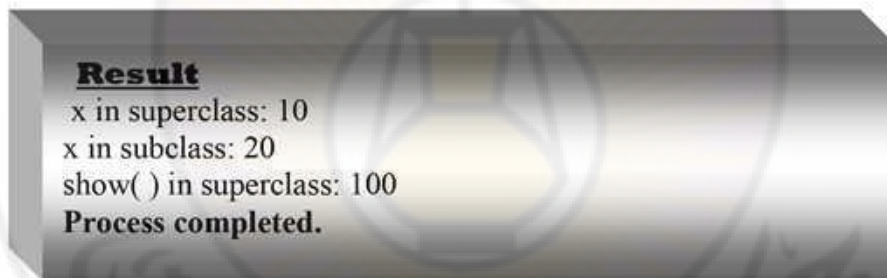
// Create a subclass by extending class A.
public class ClassB extends ClassA {
    int x ; // this x hides the i in ClassA
    public ClassB(int a, int b) {
        super.x = a; // x in ClassA
        x = b; // x in ClassB
    }
}
```

```

void show( ) {
System.out.println("x in superclass: " + super.x);
System.out.println("x in subclass: " + x );
super.show( );
}
} //end ClassB
public class UseSuper {
public static void main(String args[ ] ) {
ClassB subOb = new ClassB(10, 20);
subOb.show( );
}
} //End Class UseSuper

```

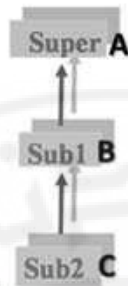
وعند تنفيذ البرنامج يظهر الخرج التالي :



٧-٦- الوراثة المتعددة

من الممكن أن يكون الصف المشتق صفاً أساسياً لصف مشتق آخر مشكلاً بذلك مراتب متعددة المستويات من الصفوف . ويصبح الصف الأساسي الأصلي صفاً غير مباشر للصف المشتق التالي . كما هو مبين في الشكل (٧-٢) . ويبين البرنامج التالي في المثال (6-7) ثلاثة صفوف كما يلي :

- الصف الأساس A
- الصف المشتق الأول B
- الصف المشتق الثاني C



الشكل (٧-٢) - يبين كيفية الوراثة المتعددة

مثال (٦-٧)

// Create a superclass.

```

public class A {
  private int i , j; // private to A
  public void setij(int x, int y) {
    i = x;
    j = y;
  }
  public void sumA( ) {
    int v=i+j ;
    System.out.println("i + j is " + v);
  }
} //End Class A
  
```

// Create a subclass by extending class A.

```

public class B extends A {
  private int total;
  public void setl( int x , int y ,int z ) {
    setij( x,y);
    total=z;
  }
  public void sum( ) {
    sumA( );
  }
}
  
```

```

System.out.println("TotalB is " + total);
}
} //End Class B
// Create a subclass by extending class B .
public class C extends B {
private int totalC;
public void setC( int x , int y ,int z,int t )
{
    set1( x,y,z);
    totalC=t;
}
public void sumC( ) {
    sum( );
    System.out.println("TotalC is " + totalC);
}
} //End Class C
//Test Class
public class Access {
public static void main(String args[ ] ) {
    C subOb = new C();
    subOb.setC(10, 12,8,33);
    subOb.sumC( );
} //End main
} //end Class Access

```

وعند تنفيذ البرنامج يظهر الخرج التالي :

Result

i + j is 22

TotalB is 8

TotalC is 33

Process completed.

تمرين (1-7)

اكتب برنامجاً بلغة جافا يحتوي على :

- الصف على الأساسي Building بحيث يخزن عدد الطوابق Floors في البناء وعدد الغرف Rooms ومساحة الغرف Sqmeter بالمتر المربع وهي متغيرات خاصة private. ويحتوي الصف على بان افتراضي وبان عادي وتابع طباعة ، وعلى بان وسيطه غرض ob.
- الصف المشتق الأول اسمه House ، يرث الصف Building ، ويخزن عدد غرف النوم Bedrooms وعدد الحمامات Bathrooms وهي متغيرات خاصة private. ويحتوي الصف على بان افتراضي وبان عادي وتابع طباعة ، وعلى بان وسيطه غرض ob.
- الصف المشتق الثاني اسمه Office ، يرث الصف House ، ويخزن عدد أجهزة الإطفاء Extinguishers وعدد الهواتف Phones وهي متغيرات خاصة private. ويحتوي على بان افتراضي وبان عادي وتابع طباعة ، وعلى بان وسيطه غرض ob.
- أنشئ صفاً آخر يتضمن التابع main اسمه BuildingTest لإنشاء أغراضاً في الصف المشتق Office ، لاستدعاء التوابع اللازمة والحصول على النتائج .

البرنامج :

```
class Building {  
    private int Floors, Rooms;  
    private double Sqmeter;  
    Building ( ) { Floors =0;Rooms=0;Sqmeter=0;}  
    Building ( int f,int r ,double m )  
    { Floors =f;
```

```

        Rooms=r;
        Sqmeter=m;
    }
    Building (Building ob) { // pass object to constructor
        Floors = ob.Floors;
        Rooms = ob.Rooms;
        Sqmeter = ob.Sqmeter;
    }
    void Printbuilding( )
    {
        System.out.println(" Rooms is " + Rooms);
        System.out.println(" Floors is " + Floors );
        System.out.println(" Sqmeter is " + Sqmeter);
    }
} //End Class

class House extends Building {
    private int Bedrooms , Bathrooms;
    House ( ) {super( ); // call superclass constructor
        Bedrooms = 0 ; Bathrooms = 0 ;
    }
    House( int f ,int r ,double m , int br, int bth) {
        super(f, r, m ); // call superclass constructor
        Bedrooms = br; Bathrooms = bth;
    }
    House( House ob) { // pass object to constructor
        super(ob ); // call superclass constructor
        Bedrooms = ob.Bedrooms;
        Bathrooms = ob.Bathrooms;
    }
    void PrintHouse( ) {
        Printbuilding( );
    }
}

```

```

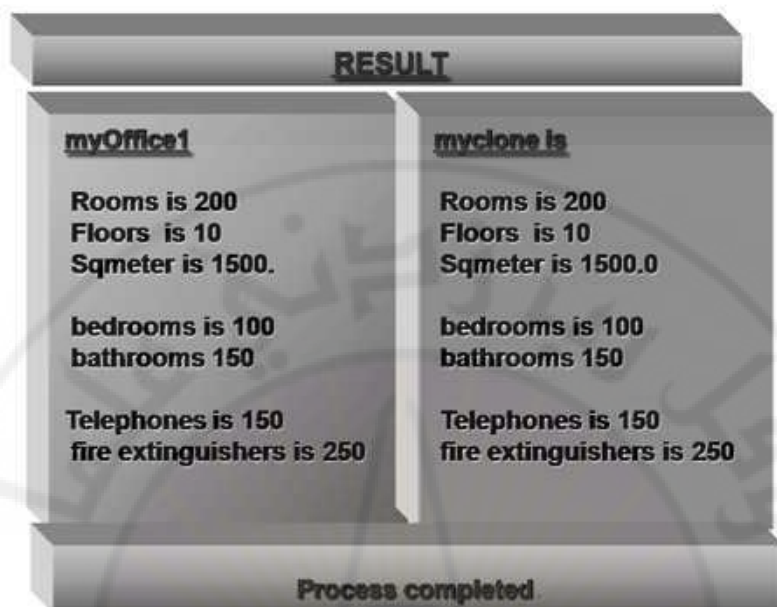
        System.out.println(" bedrooms is " + Bedrooms);
        System.out.println(" bathrooms " + Bathrooms);
    }
} //End class House

class Office extends House {
    private int Phones , Extinguishers;
    Office() {super() ; Phones = 0; Extinguishers = 0; }
    Office( int f,int r,double m , int br,int bth,int p, int ext) {
        super(f, r, m ,br ,bth ) ; // call superclass constructor
        Phones = p; Extinguishers = ext;
    }
    Office( Office ob) { // pass object to constructor
        super(ob) ; // call superclass constructor
        Phones = ob.Phones;
        Extinguishers = ob.Extinguishers;
    }
    void PrintOffice() {
        PrintHouse() ;
        System.out.println(" Telephones is " + Phones);
        System.out.println(" Fire Extinguishers is " + Extinguishers);
    }
} //End Class Office

public class BuildingTest {
    public static void main(String args[] ) {
        Office myOffice1 = new Office(10, 200, 15000 ,100,150 ,150, 250 );
        myOffice1.PrintOffice() ;
        System.out.println("\n myclone is\n ");
        Office myOffice2 = new Office(myOffice1 ) ;
        myOffice2.PrintOffice() ;
    } //End main
} // End class SumTest

```

وعند تنفيذ البرنامج يظهر الخرج التالي :



نعيد كتابة البرنامج من أجل إدخال وطباعة النتيجة على واجهة .

البرنامج :

```
import javax.swing.JOptionPane;
public class Building {
    private int Floors, Rooms;
    private double Sqmeter;
    Building() { Floors = 0; Rooms = 0; Sqmeter = 0; }
    Building ( int f ,int r ,double m ) {
        Floors =f;
        Rooms=r;
        Sqmeter=m;
    }
    Building (Building ob) { // pass object to constructor
        Floors = ob.Floors;
        Rooms = ob.Rooms;
        Sqmeter = ob.Sqmeter;
    }
}
```

```

String Printbuilding( ) {
    String output=" Rooms is " + Rooms+"\n"+ " Floors is " +
        Floors+"\n"+ " Sqmeter is " + Sqmeter+"\n";
    return output;
}
} //End Class

public class House extends Building {
    private int Bedrooms , Bathrooms;
    House ( ) { // call superclass constructor
        super( );
        Bedrooms = 0 ; Bathrooms = 0 ;
    }
    House( int f ,int r ,double m , int br, int bth) {
        super(f, r, m ); // call superclass constructor
        Bedrooms = br; Bathrooms = bth;
    }
    House( House ob) { // pass object to constructor
        super(ob ); // call superclass constructor
        Bedrooms = ob.Bedrooms;
        Bathrooms = ob.Bathrooms;
    }
    String PrintHouse( ) {
        String output=Printbuilding( )+ " bedrooms is " + Bedrooms+
            "\n"+ " bathrooms " + Bathrooms+"\n";
        return output;
    }
} //End class House

public class Office extends House {
    private int Phones , Extinguishers;
    Office() {super( ) ; Phones = 0; Extinguishers = 0; }
}

```

```

Office( int f ,int r ,double m , int br, int bth ,int p, int ext) {
    super(f, r, m ,br ,bth ) ; // call superclass constructor
    Phones = p; Extinguishers = ext;
}
Office( Office ob) { // pass object to constructor
    super(ob ); // call superclass constructor
    Phones = ob.Phones;
    Extinguishers = ob.Extinguishers;
String PrintOffice( ) {
    String output=PrintHouse()+" Telephones is " + Phones+"\n"+
        " fire extinguishers is " +Extinguishers+"\n";
    return output;
}
}//End Class Office

```

```

public class BuildingTest {
public static void main(String args[ ] ) {
int Floors=0, Rooms=0,bedrooms=0,Bathrooms=0,Phones=0,
    Extinguishers=0;
double Sqmeter=0 ;
String javaString = JOptionPane.showInputDialog(null,
    "Enter a Floors Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);
    Floors=Integer.parseInt(javaString);
javaString = JOptionPane.showInputDialog(null,
    "Enter a Rooms Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);
    Rooms=Integer.parseInt(javaString);
javaString = JOptionPane.showInputDialog(null,
    "Enter a Sqmeter Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);

```

```

Sqmeter=Double.parseDouble(javaString);
javaString = JOptionPane.showInputDialog(null,
    "Enter a bedrooms Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);
bedrooms=Integer.parseInt(javaString);
javaString = JOptionPane.showInputDialog(null,
    "Enter a Bathrooms Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);
Bathrooms =Integer.parseInt(javaString);
javaString = JOptionPane.showInputDialog(null,
    "Enter a Phones Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);
Phones=Integer.parseInt(javaString);
javaString = JOptionPane.showInputDialog(null,
    "Enter a Extinguishers Building ", "My Building Example",
    JOptionPane.QUESTION_MESSAGE);
Extinguishers =Integer.parseInt(javaString);

Office myOffice1 = new Office( Floors, Rooms ,Sqmeter,
    bedrooms,Bathrooms,Phones,Extinguishers);
Office myOffice2 = new Office( myOffice1 );
// display Building
String output =" " ;
output =output+myOffice1.PrintOffice() +"\n myclone is\n"
    +myOffice2.PrintOffice());
JOptionPane.showMessageDialog(null, output, "My Building
Class ", JOptionPane.ERROR_MESSAGE );
} //End main
} // End class SumTest

```

سوف نكتفي باظهار واجهة الخرج لطباعة النتائج في الواجهة .



٧-٧- بناء الصف Box (وراثة متعددة)

يحتوي البرنامج على الصفوف التالية :

١. يحتوي على صف أساسي superclass اسمه Box يقوم بحساب حجم الصندوق الذي هو عبارة عن متوازي مستطيلات أبعاده , width , height , depth وهي متحولات خاصة private من نمط double ، ويحتوي الصف على بان افتراضي لإعطاء القيم الابتدائية و بان عادي و بان بوسيط واحد لإعطاء قيمة واحدة للمتحولات ليتحول الشكل إلى مكعب ، وعلى بان يستعمل الأغراض كوسطاء له ، وعلى تابع volume يقوم بحساب حجم متوازي المستطيلات (حجم الصندوق) .

٢. يحتوي البرنامج على صف مشتق subclass اسمه BoxWeight يرث الصف الأساسي superclass ، يحتوي على متحول واحد هو وزن الصندوق

Weight وهو من نمط double ، يحتوي على بان افتراضي و بان عادي و باني وسيطه غرض ، و بان يستخدم عندما يصبح الصندوق مكعباً .

٣. يحتوي البرنامج على صف اسمه Shipment وهو صف مشتق من الصف BoxWeight ويدل على ثمن شحن الصندوق ، ويحتوي على بان افتراضي و بان عادي و بان وسيطه غرض و بان يستخدم عند إنشاء مكعب ، ويحتوي الصف على متحول واحد هو cost وهو ثمن الشحن .

٤. يحتوي البرنامج على صف اختبار اسمه DemoShipment يحتوي على التابع main لإنشاء الأغراض واستدعاء التوابع اللازمة والحصول على النتائج .

البرنامج :

```
//Extend BoxWeight to include shipping costs. Start with Box.
public class Box {
private double width;
private double height;
private double depth;
// construct clone of an object
public Box(Box ob) { // pass object to constructor
width = ob.width;
height = ob.height;
depth = ob.depth;
}
// constructor used when all dimensions specified
public Box(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
```

```

// constructor used when no dimensions specified
public Box() {
width = 1; // use -1 to indicate an uninitialized box
height = 1;
depth = 1;
}
public Box(double len) {
width = height = depth = len;
}
// compute and return volume
public double volume( ) {
return width * height * depth;
}
} // End Class Box
// Add weight..
public class BoxWeight extends Box {
double weight; // weight of box
// construct clone of an object
public BoxWeight(BoxWeight ob) { // pass object to constructor
super(ob);
weight = ob.weight;
}
// constructor when all parameters are specified
public BoxWeight( double w, double h, double d, double m) {
super(w, h, d); // call superclass constructor
weight = m;
}
// default constructor
public BoxWeight( ) {
super( );
weight = 1;
}

```

```

}
// constructor used when cube is created
public BoxWeight(double len, double m) {
super(len);
weight = m;
}
} // End BoxWeight
// Add shipping costs
public class Shipment extends BoxWeight {
double cost;
// construct clone of an object
Shipment(Shipment ob) { // pass object to constructor
super(ob);
cost = ob.cost;
}
// constructor when all parameters are specified
public Shipment(double w, double h, double d, double m, double c) {
super(w, h, d, m); // call superclass constructor
cost = c;
}
// default constructor
public Shipment( ) {
super( );
cost = 1;
}
// constructor used when cube is created
public Shipment(double len, double m, double c) {
super(len, m);
cost = c;
}
} // End Class Shipment

```

```

public class DemoShipment {
public static void main(String args[ ] ) {
Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);
Shipment shipment2 = new Shipment ( 2, 3, 4, 0.76, 1.28);
Shipment shipment3 = new Shipment ( ); // default
Shipment mycube = new Shipment (3, 2, 2.5); //cup
Shipment myclone =new Shipment (shipment1);
double vol;
vol = shipment1.volume( );
System.out.println("Volume of shipment1 is " + vol);
System.out.println("Weight of shipment1 is "+ shipment1.weight);
System.out.println("Shipping cost: $" + shipment1.cost);
System.out.println( );
vol = shipment2.volume( );
System.out.println("Volume of shipment2 is " + vol);
System.out.println("Weight of shipment2 is "+ shipment2.weight);
System.out.println("Shipping cost: $" + shipment2.cost);
System.out.println( );
vol = shipment3.volume( );
System.out.println("Volume of shipment3 is " + vol);
System.out.println("Weight of shipment1 is " + shipment3.weight);
System.out.println("Shipping cost: $" + shipment3.cost);
System.out.println( );
vol = mycube.volume( );
System.out.println("Volume of mycube is " + vol);
System.out.println("Weight of mycube is " + mycube.weight);
System.out.println("Shipping cost: $" + mycube.cost);
System.out.println( );
vol = myclone.volume( );
System.out.println("Volume of myclone is " + vol);
System.out.println("Weight of myclone is " + myclone.weight);

```



```

System.out.println("Shipping cost: $" + myclone.cost);
} //End main
} //End class DemoShipment

```

وعند تنفيذ البرنامج يظهر الخرج التالي :

Result
Volume of shipment 3 is 1.0
Weight of shipment3 is 1.0
Shipping cost: \$1
Volume of myclone is 3000.0
Weight of myclone is 10.0
Shipping cost: \$3.41
Volume of mycube is 27.0
Weight of mycube is 2.0
Shipping cost: \$2.5
volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: \$3.41
Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: \$1.28
Process completed

ملاحظة 1 :

- نلاحظ من البرنامج السابق وخاصة في صف الاختبار ما يلي :
- يتم انشاء غرض في الصف المشتق الثاني ويتم استدعاء التوابع اللازمة لطباعة حجم ووزن الصندوق وثمان الشحن .
- ويتم انشاء غرض آخر لحساب حجم ووزن وثمان شحن المكعب .

ملاحظة 2 :

- ما هي التعديلات التي يمكن إجراؤها على الصف المشتق BoxWeight وعلى الصف المشتق Shipment عندما نصرح عن المتحول weight والمتحول cost كمتحولين من نوع خاص private .

ملاحظة 3 : البواني والوراثة

لنذكر بعض الملاحظات المهمة تتعلق بالبواني والوراثة ، على الرغم من أن بعضها مذكورة سابقاً ، وهي :

١. البواني لا تورث و لذلك علينا كتابة جميع البواني المطلوبة في جميع الصفوف .
٢. الصف الذي لا يملك بانٍ ، يتم توليد بان واحد فقط له يدعى الباني الافتراضي وهو بان بدون وسطاء .
٣. يمكن استدعاء بان من ضمن بان آخر ، ويتم ذلك بمساعدة الكلمة المحجوزة this ، وفي هذه الحالة يجب أن تظهر كأول تعليمة في الباني
٤. يمكن استدعاء بان ما من الصف الأساسي ، وذلك باستخدام الكلمة المحجوزة super ، وفي هذه الحالة يجب أن تظهر هذه الكلمة كأول تعليمة في الباني .
٥. إذا لم نستخدم أيّاً من super أو this ، فإن المترجم سيقوم باستدعاء الـ super() .
٦. إذا قمنا بوراثة صف جميع بوانيها لها وسطاء ، فإن المترجم لن يقوم باستدعاء الـ super الافتراضية بشكل تلقائي ، لذلك علينا أن نقوم باستدعاء بان من الصف الأساسي صراحة .

=====++++++++=====

مسائل عامة محلولة

- ١- أكتب برنامجاً بلغة الـ java يتضمن الصفوف التالية:
- صفئاً أساسياً وليكن kia يحتوي على المعطيات الخاصة wheels (عدد العجلات) و range (الوزن الفارغ) وهي من النوع int . ويحتوي الصف على بان افتراضي يُسند القيمة الصفر للمعطيات الخاصة ،وبان عادي بوسيطين من النوع int يسند قيم للمعطيات الخاصة ، وتابع printKia لطباعة قيم المعطيات الخاصة.
 - صفئاً مشتقاً وليكن kiaTaxi يرث الصف الأساسي kia ويضيف عضو خاص من النوع int هو: عدد الركاب P و يحتوي على بان عادي يسند قيم للمعطيات الخاصة، وتابع عضو print للطباعة، وتابع عضو وليكن costWheels لحساب ثمن عجلات الـ kiaTaxi ، علماً بأن ثمن العجلة الواحدة يساوي 5000 ل.س.
 - صفئاً مشتقاً وليكن kiaPickup يرث الصف الأساسي kia ويحتوي على عضو خاص من النوع int هو: الحمولة loadL و يحتوي على أعضاء عامة هي: بان عادي يسند قيم للمعطيات الخاصة ، وتابع عضو print للطباعة، وتابع عضو وليكن FulRange لحساب الوزن القائم (الكلي) لـ kiaPickup .
 - صفئاً للاختبار وليكن اسمه KiaTest يتضمن التابع الرئيسي main والذي من خلاله يتم إنشاء غرضين من الصنفين: kiaTaxi و kiaPickup على الترتيب لتمرير القيم الموضحة في الجدول التالي :

الصف المشتق	wheels	range	P	loadL
-------------	--------	-------	---	-------

kiaTaxi	4	1200	4	
kiaPickup	6	2500		4000

ومن ثم يُطلب طباعة كافة المعلومات التي تم إدخالها وأيضاً طباعة ثمن العجلات

kiaTaxi والوزن الكلي لـ **kiaPickup**.

ملاحظة: الإدخال والإخراج يجب أن يكون باستخدام الواجهات GUI.

البرنامج :

```
import javax.swing.JOptionPane;
public class Kia {
    private int w,r;
    public Kia() { w=0; r=0;}
    public Kia(int w1 ,int r1){w=w1; r=r1;}
    public String printKia() {
        String output= "Wheels:"+ w+"\n";
        output=output+"range:"+ r+"\n";
        return output;
    }
    int getw() { return w;}
    int getr() {return r;}
} // end class car
public class kiaTaxi extends Kia {
    private int P;
    public kiaTaxi(int p1,int w1,int r1) {
        super(w1,r1);
        P=p1;
    }
    public String print() {
        String output= printKia();
        output=output+"passengers:"+P+"\n";
        return output;
    }
}
```

```

    }
    int costWheels( ){return 3500*getw( );}
    }// End class kiaTaxi
    public class kiaPickup extends Kia {
    private int load;
    public kiaPickup (int L,int w1,int r1) {
        super(w1,r1);
        load=L;
    }
    public String print( ) {
        String output=printKia();
        output=output+"loadlimit:"+load+"\n";
        return output;
    }
    int FulRange( ){ return getr( )+load ; }
    }// End class Pickup
    public class KiaTestGUI {
    public static void main(String args[ ]) {
        int w1,r1,p1,load1;
        String javaString = JOptionPane.showInputDialog(null,
            "Enter a value of w ", "My Kia Example",
            JOptionPane.QUESTION_MESSAGE);
        w1 = Integer.parseInt(javaString) ;
        javaString = JOptionPane.showInputDialog(null,
            "Enter a value of r ", "My Kia Example",
            JOptionPane.QUESTION_MESSAGE);
        r1 = Integer.parseInt(javaString) ;
        javaString = JOptionPane.showInputDialog(null,
            "Enter a value of p ", "My Kia Example",
            JOptionPane.QUESTION_MESSAGE);
        p1 = Integer.parseInt(javaString) ;
    }
}

```



```
kiaTaxi taxi = new kiaTaxi ( w1,r1,p1);
```

```
Kia ob = new Kia ();
```

```
javaString = JOptionPane.showInputDialog(null,  
    "Enter a value of w ", "My Kia Example",  
    JOptionPane.QUESTION_MESSAGE);
```

```
w1 = Integer.parseInt(javaString) ;
```

```
javaString = JOptionPane.showInputDialog(null,  
    "Enter a value of w ", "My Kia Example",  
    JOptionPane.QUESTION_MESSAGE);
```

```
r1 = Integer.parseInt(javaString) ;
```

```
javaString = JOptionPane.showInputDialog(null,  
    "Enter a value of load ", "My Kia Example",  
    JOptionPane.QUESTION_MESSAGE);
```

```
load1 = Integer.parseInt(javaString) ;
```

```
String output = "\n KiaTaxi:";
```

```
    output=output+taxi.print();
```

```
    output=output+" \n The cost of taxi Wheels is "+  
        taxi.costWheels()+"\n";
```

```
kiaPickup pic = new kiaPickup (w1,r1,load1);
```

```
    output=output+" \n KiaPickup: "+" \n";
```

```
    output=output+pic.print();
```

```
    output=output+" \n The ful range of Pickup is "+  
        pic.FulRange()+"\n";
```

```
JOptionPane.showMessageDialog(null, output, "My Kiacar Class ",  
    JOptionPane.ERROR_MESSAGE );
```

```
    } // end main
```

```
} // end CarTest
```


٢- أكتب برنامجاً بلغة java يتضمن مايلي :

- صفاً أساسياً يمثل متوازي مستطيلات وليكن اسمه Box معطياته الخاصة L (طول القاعدة) ، W (عرض القاعدة)، H (الارتفاع) من النوع float ، ويحتوي على التوابع العامة التالية:
بانٍ افتراضي ، وبانٍ عادي ، وتابع عضو وليكن اسمه sq مهمته حساب مساحة قاعدة متوازي المستطيلات ومن النوع float ، وتابع عضو من النوع float وليكن اسمه volume ، مهمته حساب حجم متوازي المستطيلات ، وتابع عضو وليكن اسمه print لطباعة قيم المعطيات الخاصة.
- صفاً مشتقاً من الصف Box وليكن اسمه Box1 ويمثل هرمأ متوضعاً داخل متوازي المستطيلات قاعدته هي قاعدة متوازي المستطيلات ورأسه يقع في نقطة تقاطع قطري القاعدة العلوية لمتوازي المستطيلات وليس له معطيات خاصة ولكن يتضمن التوابع العامة التالية:
بانٍ افتراضي ، وبانٍ عادي ، وتابع عضو وليكن اسمه volume1 لحساب حجم الهرم ، وتابع عضو من النوع float وليكن اسمه volume2 ، مهمته حساب الحجم المتبقي من متوازي المستطيلات بعد اقتطاع الهرم ، وتابع عضو وليكن اسمه printVolumes لطباعة قيم الأحجام التي تم حسابها.
- صفاً للاختبار وليكن اسمه TextBox يتضمن إنشاء الأغراض اللازمة من أجل إدخال قيمٍ للمعطيات الخاصة عن طريق واجهة GUI ومن ثم استدعاء ما يلزم من التوابع من أجل حساب وطباعة الأحجام المطلوبة وقيم المعطيات الخاصة للصف الأساسي على شكل واجهة GUI.
- ملاحظة مهمة: الإدخال والإخراج يجب أن يكون باستخدام الواجهات GUI .

```
import javax.swing.JOptionPane;

public class Box {
    private double L,w,h;
    public Box() {
        L=w=h=0;
    }
    public Box (double L1,double w1,double h1)
    {
        L=L1 ;
        w=w1 ;
        h=h1 ;
    }
    public double getL() {return L;}
    public double getw() {return w;}
    public double geth() {return h;}
    public double sq() {
        return L*w;
    }
    public double valume() {
        return L*w*h;
    }
    public String print() {
        String output= "Data of Box are =" + L + " " + w + " " + h + "\n";
        return output;
    }
    } //End class

    public class Box1 extends Box {
    public Box1() {
        super(0,0,0);
    }
}
```

```

public Box1(double L1,double w1,double h1) {
    super(L1,w1,h1);
}
double valume1( ) {
    double s;
    s=(1/3)*sq( )*geth( );
    return s;
}
public double valume2( ) {
    return valume( )- valume1( );
}
public String printValumes( ) {
    String output= "The valumes are =" + valume( )+ " "+
        valume1( )+" "+valume2()+"\n";
    return output;
}
} //End class

```

```

public class TestBox {
public static void main(String args[ ] ) {
    double L1,W1, h1;
    String javaString = JOptionPane.showInputDialog(null,
        "Enter a value of L1 ", "My Area Example",
        JOptionPane.QUESTION_MESSAGE);
    L1 = Double.parseDouble(javaString);
    javaString = JOptionPane.showInputDialog(null,
        "Enter a value of W1 ", "My Area Example",
        JOptionPane.QUESTION_MESSAGE);
    W1 = Double.parseDouble(javaString);
    javaString = JOptionPane.showInputDialog(null,
        "Enter a value of h1 ", "My Area Example",

```

```

JOptionPane.QUESTION_MESSAGE);
h1 =Double.parseDouble(javaString);
//-----
Box1 ob1 = new Box1();
Box1 ob2 = new Box1(L1,W1,h1);

String output ="\n The result is:\n";
output=output+"\n Data of Box is \n "+ob2.print( );
output=output+"\n The valumes is \n "+ ob2.printValumes( )+"\n";
JOptionPane.showMessageDialog(null, output, "My Areas ",
JOptionPane.ERROR_MESSAGE );
} //End main
} //End class

```

٣- أكتب برنامجاً بلغة Java يتضمن الصفوف التالية:

- صفّاً أساسياً اسمه Circle يحتوي على العضو الخاص r (نصف قطر الدائرة) من النوع double . ويحوي هذا الصف على التتابع الأعضاء العامة التالية :

بانٍ افتراضي يُسند قيمة الصفر ، وبانٍ عادي ، وتابع اسمه sq لحساب مساحة الدائرة ، وتابع اسمه printCircle لطباعة قيمة المعطى الخاص ومساحة الدائرة ، وتابع اسمه getRaduis للوصول إلى المعطى الخاص.

- صفّاً مشتقاً اسمه Cone (مخروط دوراني) يرث الصف الأساسي Circle ويحتوي على متحول خاص من النوع double هو h (ارتفاع المخروط) و يحتوي على الأعضاء العامة التالية:

بانٍ افتراضي ،و بانٍ عادي ، وتابع اسمه valumeCone لحساب حجم المخروط الدوراني ، وتابع اسمه PrintCone لطباعة قيمة المعطى الخاص

وحجم المخروط وكل ما يتعلق بالصف الأساسي ، و تابع اسمه getHeight للوصول إلى المعطى الخاص .

- صفاً مشتقاً اسمه Cylinder يرث الصف الأساسي Cone و يحتوي على الأعضاء العامة التالية:

بان افتراضي يُسند القيمة صف للمعطيات الخاصة ، و بان عادي ، و تابع اسمه volumeCylinder لحساب حجم الأسطوانة (حيث أن الارتفاع هو ضعف ارتفاع المخروط الدوراني) ، و تابع اسمه PrintCylinder لطباعة قيمة حجم الأسطوانة وكل ما يتعلق بالصفين Cone و Circle ، و تابع اسمه volumeExtra لحساب حجم الجزء المتبقي بعد حذف مخروطين متقابلين بالرأس من الأسطوانة آنفة الذكر .

- صفاً للاختبار اسمه ValumeTest يحوي على التابع الرئيسي main ، والذي بدوره يتضمن إنشاء أغراض من الصف Cylinder من أجل تمرير قيم لنصف القطر والارتفاع ومن ثم إجراء الاستدعاءات اللازمة من أجل حساب مساحة الدائرة والأحجام المطلوبة وطباعتها .

- ملاحظة: الإدخال والإخراج يجب أن يكون باستخدام الواجهات GUI.

البرنامج :

```
import javax.swing.JOptionPane;
public class Circle {
    private double r;
    public Circle() { r=0;}
    public Circle ( double r1 ){ r=r1;}
    public String PrintCircle( ) {
        String output= " Raduis is " + r+"\n";
        output=output+" The square of Circele is " + sq()+ "\n";
        return output;
    }
}
```



```

public double sq( ){return Math.PI*r*r;}
public double getRaduis( ){return r;}
} //End Class Circle
public class Cone extends Circle {
private double h;
public Cone ( ) {
    super( );
    h = 0 ;
}
public Cone( double r1 ,double h1 ) {
    super(r1);
    h = h1;
}
public String PrintCone( ) {
    String output= PrintCircle( );
    output=output+" height is " + h+"\n";
    output=output+" The valume of Cone is " + valumeCone()+"\n";
return output;
}
public double getHeight( ){return h;}
public double valumeCone( ){return (1/3.0)*sq()*h;}
} //End class Cone
public class Cylinder extends Cone {
public Cylinder( ) {super( ) ; }
public Cylinder(double r1 ,double h1 ){super(r1, h1 ) ;}
public String PrintCylinder( ) {
    String output=PrintCone( );
    output=output+" The valume of Cylinder is " +
                                valumeCylinder( )+"\n";
return output;
}

```

```

public double valumeCylinder( ){return sq()*2*getHeight();}
public double valumeExtra( ){
return valumeCylinder()-2*valumeCone( ); }
} //End Class Cylinder

public class ValumeTestGUI {
public static void main(String args[ ] ) {
    String ouput =" \n The output is\n";
    Cylinder cylind1 = new Cylinder( );
    double r1,h1;
    String javaString = JOptionPane.showInputDialog(null,
        "Enter a value of r ", "My Circle Example",
        JOptionPane.QUESTION_MESSAGE);
    r1 = Double.parseDouble(javaString) ;
    javaString = JOptionPane.showInputDialog(null,
        "Enter a value of h ", "My Circle Example",
        JOptionPane.QUESTION_MESSAGE);
    h1 = Double.parseDouble(javaString) ;
    //-----
    Cylinder cylind2 = new Cylinder(r1,h1 );
    ouput = ouput +cylind2.PrintCylinder( );
    ouput = ouput +(" The Extra valume is "+ cylind2.valumeExtra());

    JOptionPane.showMessageDialog(null, ouput,
        "My Circle Class ", JOptionPane.ERROR_MESSAGE );
    } //End main
} // End class ValumeTest

```

٤- أكتب برنامجاً بلغة Java يتضمن:

- صفّ أساسياً اسمه Equation يحتوي على ثلاثة متحولات خاصة a و b و c من نوع float ، ويحتوي الصف على التوابع العامة التالية : بان

افتراضي يُسند الصفر للمعطيات الخاصة، وبأن عادي لتمرير المعطيات ،
وتتابع وصول، وتابع عضو اسمه delta لحساب قيمة المميز
b2-4ac، وتابع عضو اسمه print لطباعة قيم المعطيات الخاصة.

- صفاً مشتقاً من الصف Equation وليكن اسمه Roots ويحتوي على
متحولات خاصة x1 و x2 من النوع float ويحتوي الصف على
التتابع العامة التالية: بأن افتراضي يُسند الصفر للمعطيات الخاصة، وبأن
عادي لتمرير المعطيات، تابع عضو اسمه computeRoots لحساب
جذور المعادلة $ax^2 + bx + c = 0$ وإسناد النتائج لمعطيات الصف Roots ،
وتابع عضو اسمه printRoots لطباعة قيم المعطيات الخاصة.
- أنشئ صفاً للاختبار اسمه Test يحتوي على التابع الرئيسي main ،
والذي يتضمن بدوره إنشاء أغراض من الصف Roots من أجل تمرير
قيم لمعطيات الصفوف ومن ثم استدعاء التتابع اللازمة من أجل حساب
وطباعة جذور المعادلة وقيم معطيات الصف الأساسي .

البرنامج :

// Create a superclass.

```
public class Equation {
    private int a , b, c;
    public Equation( ) {
        a = 0;    b = 0;    c = 0;
    }
    public Equation(int x, int y,int z) {
        a = x;    b = y;    c = z;
    }
    public int geta(){return a;}
    public int getb(){return b;}
    public int getc(){return c;}
```

```

public int delta( ) {
    int d;
    d=b*b-4*a*c;
    return d;
}

public void print( ) {
    System.out.println("The constants are " + a + " " + b + " " + c + "\n");
}
} //End Class Equation
// Create a subclass by extending class A.
public class roots extends Equation {
    private double x1,x2;
    public roots ( ) {
        super( );
        x1 = 0;    x2 = 0;
    }
    public roots( int x , int y ,int z ,int xx ,int yy) {
        super(x,y,z);
        x1=xx;    x2=yy;
    }
    public void computeRoots( ) {
        int d1;
        d1= delta( );
        if (d1>0) {
            x1=( -getb()-Math.sqrt(d1) )/(2*geta());
            x2=( -getb()+Math.sqrt(d1) )/(2*geta());
        }
        else if(d1==0){
            x1=( -getb() )/(2*geta());
            x2=x1;
        }
    }
}

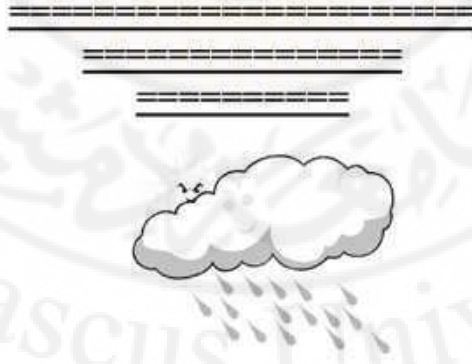
```

```

    else
        System.out.println("No real solutions"+" \n");
    }
    public void printRoots( ) {
        print( );
        System.out.println("The roots are " + x1 + " "+x2+"\n");
    }
} //End Class roots

public class SolvingEquation {
    public static void main(String args[ ] ) {
        roots r = new roots( );
        roots r1 = new roots(1,-4,4,0,0);
        r1.computeRoots( );
        r1.printRoots( );
    } //End main
} //end Class SolvingEquation

```



مسائل عامة

١ - اكتب برنامجاً بلغة Java لإنشاء صف اسمه Book ، يحتوي على ثلاثة وسطاء خاصة من النوع String ، هي : اسم الكتاب NameBook واسم دار النشر Publishing وسنة النشر Year ، يحتوي هذا الصف على بان افتراضي وتابع تعديل معطيات وتابع طباعة اسمه PrintBook .

- استخدم الوراثة لإنشاء صف مشتق من الصف الأساسي اسمه AddressBook ويرث من الصف الأساسي اسم الكتاب ودارا لنشر وسنة النشر ويضيف معلومات على هذا الكتاب هي : اسم المؤلف Writer والدولة Country والمدينة City وهي متحولات خاصة من نمط String ، ويقوم الصف المشتق بإدخال المعلومات السابقة عن الكتاب وطباعتها ، ويحتوي الصف المشتق على بان عادي وتابع طباعة اسمه ShowBook .
- يحتوي البرنامج على صف اختبار اسمه TestBook يحتوي على التابع main لإنشاء الأغراض واستدعاء التوابع اللازمة والحصول على النتائج .

=====

- ٢ - اكتب برنامجاً بلغة Java يحتوي على :
- صف أساسي اسمه date ، يحتوي على ثلاثة وسطاء (day , month , year) من النوع int كأعضاء private ويقوم بإدخال التاريخ وطباعته بالشكل (year : month : day) ويحتوي الصف على بان افتراضي لتحديد القيم الابتدائية للتاريخ وهي (2000 : 01 : 01) وقيمة الوسيط day ضمن المجال [1 - 31] والوسيط month ضمن المجال

- [1 - 12] والسنة year ضمن المجال [2000 - 2014] ويحتوي الصف على باني عادي وباني وسيطه غرض وعلى تابع طباعة () printf.
 - صف مشتق أول اسمه time يرث الصف date وراثته عامة public ، ويحتوي على ثلاثة وسطاء (Second , Minute , Hour) من النوع int كأعضاء private ويقوم بإدخال الوقت وطباعته بالشكلين القياسي والعادي (العسكري) ، ويحتوي على بان افتراضي لطباعة القيمة الابتدائية ، وعلى تابعي طباعة هما () printf لطباعة الوقت بالشكل العسكري و () prints لطباعة الوقت بالشكل القياسي ، ويحتوي الصف على بان وسيطه غرض وبان عادي حيث الوسيط Hour ضمن المجال [0 - 24] أما الوسيط Minute والوسيط Second ضمن المجال [0 - 59] .
 - صف مشتق ثاني اسمه Airline (مكتب الخطوط الجوية) ، يرث الصف time وراثته عامة public ، ويحتوي على وسيطين كأعضاء private هما: اسم الشخص المسافر name واسم الدولة التي سيسافر إليها CountryName وهي من النوع char ، ويحتوي الصف على بان افتراضي وبان عادي وعلى تابع طباعة () printAir حيث يطبع اسم المسافر واسم الدولة والتاريخ والوقت الذي سيغادر فيه المسافر .
 - صف للاختبار اسمه AirlineTest يحتوي على التابع الرئيسي main متضمناً أنشاء غرض في الصف المشتق الأخير من أجل طباعة كافة المعطيات لجميع الصفوف .
 - ما وظيفة الكلمة المحجوزة this ؟
 - ما وظيفة الكلمة المحجوزة super ؟
 - ما الواجهة Interfaces برمجياً ؟

٣- اكتب برنامجاً بلغة Java لإنشاء صف أساسي اسمه Faculty يحتوي على ثلاثة متحولات خاصة private من النوع int هي : عدد الأساتذة Teacher ، وعدد المهندسين Engineer ، وعدد الطلاب Student. ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه () PrintFaculty.

- يرث الصف المشتق Department الصف الأساسي Faculty ويحتوي على ثلاثة متحولات خاصة private نوع String هي : اسم القسم CharminName ، واسم رئيس القسم CharminName ، وأسنة تأسيس القسم year. ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه () PrintDepartment .

- يرث الصف المشتق Building الصف المشتق Department ويحتوي على ثلاثة متحولات خاصة private نوع int وهم : عدد المكاتب Office ، وعدد المخابر Laboratory ، وعدد القاعات الدراسية Room. ويحتوي الصف على بان افتراضي و بان عادي وتابع طباعة اسمه () PrintBuilding .

- يحتوي البرنامج على صف اختبار اسمه Test Faculty يحتوي على التابع main لإنشاء الأغراض واستدعاء التوابع اللازمة والحصول على النتائج .

=====

٤- أكتب برنامجاً بلغة Java يتضمن ما يلي:

- صفاً أساسياً اسمه Product يحتوي على وسيطين year , model ، النوع int كأعضاء خاصة ، ويحتوي على الأعضاء العامة (public) التالية:

- بان افتراضي يسند القيمة صفر للمعطيات الخاصة.

- بان عادي بوسيطين من النوع int يسند قيم للمعطيات الخاصة.
- تابع عضو print من النوع void لطباعة قيم المعطيات الخاصة.
- صفاً مشتقاً من الصف الأساسي Product اسمه computer ، يحتوي على الأعضاء الخاصة التالية : الاسم name ، السرعة speed ، حجم القرص الصلب hard (يقاس GB)، ويحتوي على الأعضاء العامة (public) التالية :
 - بان عادي بثلاثة وسطاء .
 - تابع عضو print من النوع void لطباعة قيم المعطيات الخاصة.
 - تابع عضو اسمه fir ، مهمته حساب عدد الأقراص الليزرية CD (بفرض أن سعة القرص هي 800 MB (ميغا بايت)) التي تكافئ حجم الـ hard.
 - تابع عضو اسمه extra من النوع float ، مهمته معرفة الفرق بعدد الأقراص بين نوعين من الهاردات (Two hard) .
- أنشئ صفاً للاختبار اسمه Test يحتوي على التابع الرئيسي main ثم صرح في التابع الرئيسي main عن غرضين من الصف computer ومرر لهما القيم الموضحة بالجدول التالي، ثم استدع التتابع اللازمة للحصول على النتائج .

Name	Model	Year
pentum3	1	1998
Pentum4	2	2000

- ٥- أكتب برنامجاً بلغة Java يتضمن الصفوف التالية:
- صفّاً أساسياً vehicle (مركبة) يحتوي على الأعضاء الخاصة (عدد العجلات) و r (الوزن) وهي من النوع int . ويحتوي هذا الصف على التوابع العامة التالية:
 - بانٍ افتراضي يُسند الصفر للمعطيات الخاصة.
 - بانٍ عادي بوسيطين من النوع int يسند قيم للمعطيات الخاصة .
 - تابع وصول () get_wr من النوع void للوصول للمعطيات الخاصة.
 - تابع () printv للطباعة.
 - صفّاً مشتقاً اسمه bus (باص) يرث الصف الأساسي vehicle ويرث منه عدد العجلات و الوزن ويضيف معطيات خاصة من النوع الصحيح int هي: عدد الركاب pass و يحتوي على أعضاء عامة هي: بانٍ عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات الخاصة للصف الأساسي vehicle وتابع عضو () print للطباعة.
 - صفّاً مشتقاً اسمه truck (جرار) يرث الصف الأساسي vehicle ويرث منه عدد العجلات و الوزن ويضيف معطيات (عضو خاص) من النوع الصحيح int هي: الحمولة load و يحتوي على أعضاء عامة هي: بانٍ عادي بثلاثة وسطاء من النوع int يسند قيم للمعطيات الخاصة للصف الأساسي vehicle وتابع عضو () print للطباعة.
 - أنشئ صفّاً للاختبار اسمه TestVehichle يحتوي على التابع الرئيسي main ثم صرح في التابع الرئيسي main عن غرضين من الصنفين: bus و truck على الترتيب ثم مرر لهما القيم الموضحة بالجدول التالي، ثم استدع التوابع اللازمة للحصول على النتائج .

Vehicle	W	R	Pass	Load
Bus	4	600	10	---
Truck	6	1400	---	4000

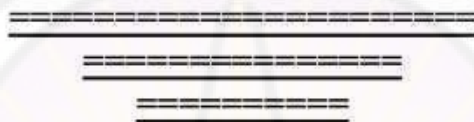
=====

٦- اكتب برنامجاً بلغة Java يتضمن الصفوف التالية:

- صفّاً أساسياً اسمه AreaCirc يحتوي على العضو المحمي rad (نصف قطر الدائرة) من النوع float ويحتوي على التوابع الأعضاء العامة التالية :
بانٍ افتراضي ، بانٍ عادي ، وتابع اسمه square لحساب مساحة الدائرة، وتابع اسمه ShowCirc لطباعة قيمة المعطى الخاص ومساحة الدائرة .
- صفّاً مشتقاً من الصف الأساسي AreaCirc اسمه NewCone (مخروط دوراني) ويحتوي على معطى محمي من النوع float هو he (الارتفاع) و يحتوي على الأعضاء العامة التالية:
بانٍ افتراضي ، وبانٍ عادي ، وتابع اسمه valCone لحساب حجم المخروط الدوراني، وتابع اسمه ShowCone لطباعة قيمة المعطى الخاص وحجم المخروط وكل ما يتعلق بالصف الأساسي .
- صفّاً مشتقاً من الصف الأساسي NewCone وليكن NewCylind و يحتوي على الأعضاء العامة التالية :
بانٍ افتراضي ، بانٍ عادي ، وتابع اسمه valCylind لحساب حجم الأسطوانة (حيث أن الارتفاع هو ضعف ارتفاع المخروط الدوراني) وتابع اسمه ShowCylind لطباعة قيمة حجم الأسطوانة وكل ما يتعلق بالصفين NewCone و AreaCirc ، وتابع اسمه ExtraVal لحساب حجم الجزء المتبقي بعد حذف مخروطين متقابلين بالرأس من الأسطوانة أنفة الذكر .

- صفًا للاختبار وليكن اسمه ValumeTest يحوي على التابع الرئيسي main ، والذي بدوره يتضمن إنشاء أغراض من الصف NewCylind من أجل تمرير قيم لنصف القطر والارتفاع ومن ثم إجراء الاستدعاءات اللازمة من أجل حساب وطباعة مساحة الدائرة والأحجام المطلوبة.
- ملاحظة: الإدخال والإخراج يجب أن يكون باستخدام الواجهات GUI.

=====



جامعة دمشق
Damascus University

الفصل الثامن

تعدد الأشكال والواجهات

Polymorphism & Interfaces





تعدد الأشكال والواجهات

Polymorphism & Interfaces

٨-١- تعدد الأشكال

إن تعدد الأشكال في لغة Java هي الميزة الأساسية الثالثة للبرمجة غرضية التوجه OOP ، في حين أن الميزتين الأوليتين هما الصفوف والوراثة. تقوم التغليف (الصفوف) بإنشاء أنماط معطيات جديدة عبر تجمع الخصائص والسلوكيات ، وتسمح ميزة التحكم بالوصول بفصل الواجهة عن التنفيذ عبر جعل المتحولات خاصة private . وبالتالي يعني هذا النوع من التنظيم الميكانيكي منطقاً جاهزاً لمن له خبرة في البرمجة البنيوية ، أما التوابع المجردة فتتعامل مع الفصل باستخدام مصطلحات الأنماط types . رأينا في الفصل السابع كيف تسمح الوراثة بمعاملة الغرض كأنه ينتمي لنمطه الأصلي أو نمط الصف الأساسي. وتعد هذه المقدرة أساسية لأنها تسمح بأن يتعامل معاملة العديد من الأنماط كما لو كانت نمطاً وحيداً . ويسمح التابع المجرد لنمط ما أن يعبر عن تميزه عن نمط آخر مشابه ما داماً مشتقين من الصف الأساسي نفسه . ويتم التعبير عن هذا التمييز من خلال الاختلافات في سلوك التوابع التي يمكن استدعاؤها بواسطة الصف الأساسي .

وتساعد تقنية تعدد الأشكال في تطوير برامج عامة بدلاً من كتابة برامج خاصة لهدف معين ومحدد ، إن فهم تعدد الأشكال يبدأ من فهم التوابع المجردة (abstract) ومتى نحتاج إلى استخدامها . وإن تعدد الأشكال هو معرفة كيفية انجاز خدمة مبنية ضمن الغرض نفسه وليس في الشيفرة التي تستخدم الغرض ، و استدعاء تابع محدد أو عملية محددة يمكن أن يأخذ عدداً غير محدد من الأشكال المختلفة ، و تعدد الأشكال ممكن باستخدام التوابع المجردة .

Abstract Classes

تُعد الصفوف المجردة حلاً لأحد أهم مشاكل الوراثة وهي إيجاد صف أساسي عام يحوي كل ما هو مشترك بين عدة مفاهيم . وقد توجد بعض التوابع المشتركة من حيث السلوك والمختلفة من حيث التحقيق ، فيأتي التجريد ليحل هذه المشكلة . وبدونه إما أن نتجاهل هذه التوابع فلا نضعها في الصف الأساسي (وهذا خطأ) أو أن نكتب لهذه الطريقة تحقيقات وهمية أو خيالية وهذا أيضاً خطأ ويتم التصريح عن الصفوف المجردة والتوابع المجردة باستخدام الكلمة المحجوزة **abstract** كما يلي :

```
abstract class Figure {
.....
abstract double area() ; // لا يوجد له جسم
.....
} //End abstract class Figure
```

يتم استخدام الصف المجرد بشكل نموذجي كصف أساسي ولا يمكن أخذ مثل منه ، ويمكن أن يحتوي على توابع مجردة (ليس لها جسم) ، وتوابع غير مجردة (لها جسم) ، كما يمكن أن يكون صفاً مشتقاً لصف مجرد أو غير مجرد. ولا يمكن لصف غير مجرد أن يحوي على توابع مجردة . ويجب تعريف جميع التوابع المجردة من قبل الصفوف التي ترث الصف المجرد ، ما لم يكن الصف المشتق صفاً مجرداً أيضاً ، وتعد الطريقة المجردة بمثابة تعهد بأن الصفوف المشتقة يجب أن تتجاوز **Override** التوابع المجردة وتعيد تعريفها بما يتناسب الصف المشتق ، وإلا سيكون هناك خطأ . وتحتوي التوابع المجردة فقط على تصريح التابع ، ويجب إعادة تعريفها من قبل أي صف غير مجرد يرثها :

```
public class Rectangle extends Figure {
```

```
.....
```

```
// override area for rectangle
```

```
double area( ) { // تم اعادة تعريف التابع المجرد من قبل الصف المشتق
```

```
System.out.println("Inside Area for Rectangle.");
```

```
return dim1 * dim2;
```

```
}//End area
```

```
}//End classRectangle
```

مثال (1-8)

ينشئ البرنامج صفاً أساسياً مجرداً اسمه Figure يحتفظ ببيدي شكل ما .
ويصرح أيضاً عن تابع مجرد اسمه area() والذي يعيد عند تجاوزه من قبل
الصف المشتق مساحة نوع الشكل المعروف في الصف المشتق .

وفي هذه الحالة يحدد تصريح التابع ، داخل الصف الأساسي المجرد ، شكل
الواجهة في حين أوكل التمثيل الفعلي لهذه الواجهة إلى الصفوف التي ورثت
الصف الأساسي ، حيث :

- يرث الصف المشتق الأول Rectangle الصف الأساسي Figure ، ويحسب مساحة المستطيل .
- يرث الصف المشتق الثاني Triangle الصف الأساسي Figure ، ويحسب مساحة مثلث .
- يحتوي الصف الاختيائي AbstractAreas على التابع الرئيسي main ، ثم نصرح في التابع الرئيسي main عن غرضين من الصنفين : Rectangle و Triangle على الترتيب ، ثم نستدعي التوابيع اللازمة للحصول على النتائج .

// Using abstract methods and classes.

```
abstract public class Figure {  
    double dim1;  
    double dim2;  
    public Figure(double a, double b) {  
        dim1 = a;  
        dim2 = b;  
    }  
}
```

// area is now an abstract method

```
abstract public double area( );  
}//End abstract class Figure  
public class Rectangle extends Figure {  
    public Rectangle(double a, double b) {  
        super(a, b);  
    }  
}
```

// override area for rectangle

```
public double area( ) {  
    System.out.println("Inside Area for Rectangle.");  
    return dim1 * dim2;  
}
```

//End class Rectangle

```
public class Triangle extends Figure {  
    public Triangle(double a, double b) {  
        super(a, b);  
    }  
}
```

// override area for right triangle

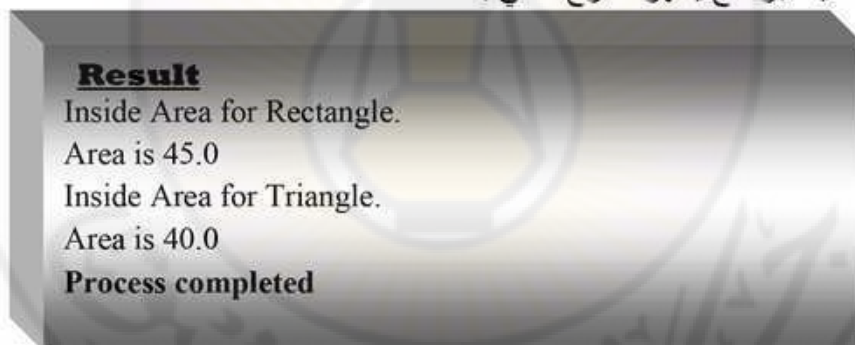
```
public double area( ) {  
    System.out.println("Inside Area for Triangle.");  
    return dim1 * dim2 / 2;  
}
```

```

} //End class Triangle
public class AbstractAreas {
public static void main(String args[ ] ) {
    //Figure f = new Figure(10, 10); // illegal now
    Rectangle r = new Rectangle(9, 5);
    Triangle t = new Triangle(10, 8);
    Figure figref; // this is OK, no object is created
    figref = r;
    System.out.println("Area is " + figref.area( ));
    figref = t;
    System.out.println("Area is " + figref.area( ));
} //End main
} //End class AbstractAreas

```

وعند تنفيذ البرنامج يظهر الخرج التالي :



إذا كانت المتحولات في الصف المجرد هي خاصة private عندها يجب كتابة المتحولات كتوابع وصول من أجل استخدامها بدلاً من المتحولات في الصفوف المشتقة لأنها غير مرئية لهذه الصفوف. ويصبح البرنامج كما يلي :

```

// Using abstract methods and classes.
abstract public class Figure {
private double dim1;
private double dim2;
public Figure(double a, double b) {

```

```

dim1 = a;
dim2 = b;
}
// area is now an abstract method
abstract public double area( );
// dim2, dim1  تتابع الوصول للمتحويلات
public double getdim1( ) {
    return dim1 ;
}
public double getdim2( ) {
    return dim2 ;
}
} //End abstract class Figure
public class Rectangle extends Figure {
public Rectangle(double a, double b) {
    super(a, b);
}
// override area for rectangle
public double area {
    System.out.println("Inside Area for Rectangle.");
    return getdim1( ) * getdim2( ) ;
}
} //End class Rectangle
public class Triangle extends Figure {
public Triangle(double a, double b) {
    super(a, b);
}
// override area for right triangle
public double area( ) {
    System.out.println("Inside Area for Triangle.");
    return getdim1( ) * getdim2( ) / 2 ;
}
}

```



```

}
} //End class Triangle
public class AbstractAreas {
public static void main(String args[ ] ) {
    //Figure f = new Figure(10, 10); // illegal now
    Rectangle r = new Rectangle(9, 5);
    Triangle t = new Triangle(10, 8);
    System.out.println("Area is " + r.area( ));
    System.out.println("Area is " + t.area( ));
} //End main
} //End class AbstractAreas

```

نعيد كتابة البرنامج من أجل إدخال المعطيات وطباعة النتيجة على واجهة.

// Using abstract methods and classes.

```

import javax.swing.JOptionPane;
abstract public class Figure {
    private double dim1;
    private double dim2;
    public Figure(double a, double b) {
        dim1 = a;    dim2 = b;
    }
    // area is now an abstract method
    abstract public double area( );
    // dim2, dim1  تتابع الوصول للمتغيرات
    public double getdim1( ) {
        return dim1 ;
    }
    public double getdim2( ) {
        return dim2 ;
    }
} //End abstract class Figure
public class Rectangle extends Figure {

```

```

public Rectangle(double a, double b) {
    super(a, b);
}
// override area for rectangle
public double area {
    System.out.println("Inside Area for Rectangle.");
    return getdim1( ) * getdim2( ) ;
}
} //End class Rectangle
public class Triangle extends Figure {
public Triangle(double a, double b) {
    super(a, b);
}
// override area for right triangle
public double area( ) {
    System.out.println("Inside Area for Triangle.");
    return getdim1( ) * getdim2( ) / 2 ;
}
} //End class Triangle
public class AbstractAreas {
public static void main(String args[ ] ) {
    double a1,b1,a2,b2;// 10,20,15
    String javaString = JOptionPane.showInputDialog(null,
        "Enter a a1 Number ", "My Rectangle class Example",
        JOptionPane.QUESTION_MESSAGE);
    a1 = Double.parseDouble(javaString) ;
    javaString = JOptionPane.showInputDialog(null,
        "Enter a b1 Number ", "My Rectangle class Example",
        JOptionPane.QUESTION_MESSAGE);
    b1 = Double.parseDouble(javaString) ;
    //-----

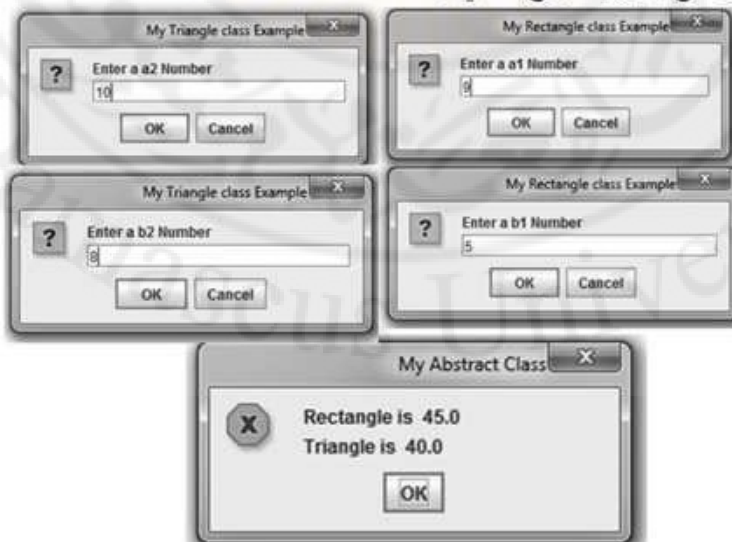
```

```

javaString = JOptionPane.showInputDialog(null,
    "Enter a a2 Number ", "My Triangle class Example",
    JOptionPane.QUESTION_MESSAGE);
a2 = Double.parseDouble(javaString) ;
javaString = JOptionPane.showInputDialog(null,
    "Enter a b2 Number ", "My Triangle class Example",
    JOptionPane.QUESTION_MESSAGE);
b2 = Double.parseDouble(javaString) ;
//-----
Rectangle r = new Rectangle(a1,b1);//9,5
Triangle t = new Triangle(a2,b2); //10,8
// display Rectangle
String output =" Rectangle is " + r.area( )+ "\n";
// display Triangle
output = output +" Triangle is "+ t.area( )+"\n";
JOptionPane.showMessageDialog(null, output, "My Abstract
Class ", JOptionPane.ERROR_MESSAGE );
} //End main
} //End class AbstractAreas

```

وعند تنفيذ البرنامج يظهر الخرج التالي :



٨-١-٢ - المعطيات الأعضاء والتوابع والثوابت الساكنة

١ - المعطيات الأعضاء الساكنة

للمعطيات الأعضاء الساكنة الميزات نفسها للتوابع الساكنة بالإضافة إلى أنه يتم تخزينها في موقع وحيد في الذاكرة ، ويتم استخدامها عند الحاجة إلى نسخة واحدة فقط من العضو لتشاركه جميع أمثال الصف .

مثال :

```
static int a ;  
System.out.println("a = " + a ) ؛
```

٢ - التوابع الساكنة

يتم التصريح عن التوابع الساكنة باستخدام الكلمة المجوزة **static** . لا تستطيع التوابع الساكنة الوصول إلى التوابع أو المتحولات غير الساكنة ، لأن التوابع الساكنة مرتبطة مع صف وليس مع غرض .

مثال :

```
public static void sum( )  
{  
.....  
}
```

٣ - الثوابت الساكنة

الثوابت الساكنة هي أعضاء ساكنة مصرح عنها على أنها ثوابت ، ولا يستطيع البرنامج تغييرها ، ونستخدم الكلمتين المحجوزتين **final , static** .

مثال :

```
static final int Age = 18 ;
```

٨-٢- الواجهات Interfaces

إن الواجهة هي أداة تستخدمها الأغراض غير المترابطة للتفاعل فيما بينها ،
فجهاز التحكم واجهة بيننا وبين التلفاز ، واللغة الإنكليزية واجهة بيننا وبين
الأشخاص الآخرين ، والقوانين والبروتوكولات المفروضة في الجيش واجهة بين
الجنود والضباط من الرتب المختلفة . . الخ
في لغة Java الواجهة هي أداة تستخدمها الأغراض غير المترابطة للتفاعل
فيما بينها .

الواجهة برمجياً هي صف :

✓ جميع توابعه عامة ومجردة public static حتى لو لم يذكر ذلك
صرحة .

✓ جميع أعضائه ثابتة وعامة وساكنة public static final حتى لو لم
يذكر ذلك صراحة .

٨-٢-١- كتابة الواجهات

يتضمن تعريف الواجهة قسمين :

❖ تصريح الواجهة .

❖ جسم الواجهة .

يوضح قسم التصريح العديد من خصائص الواجهة مثل اسمها ، وهل هي توسع
واجهة أخرى أم لا ، ويحتوي جسم الواجهة على تصريحات الثوابت والتوابع داخل
الواجهة .

مثال عن واجهة باسم Figure

```
public interface Figure {  
    public long x=1 ;
```



```

public long y = 2 ;
public void func( ) ;
}

```

حيث يمثل السطر الأول التصريح ، وأما الباقي فيشير إلى أن هذه الواجهة تحوي ثابتين هما x و y ، وتابع واحد مجرد هو func() .

٨-٢-٢- تحقيق الواجهة

للتصريح عن صف يحقق واجهة يجب استخدام العبارة implements . ويمكن للصف تحقيق أي عدد من الواجهات بحيث يتم فصلها عن بعضها البعض باستخدام الفاصلة (,) .

مثلاً: الصف Myclass يحقق الواجهة Figure .

```

public class Myclass implements Figure {
public void func( ) {
// .....
}
public static void main( String [ ] args ) {
//. ....
}
} //End class

```

نلاحظ كيف قام الصف Myclass بتحقيق التابع func() ، وتضمن أيضاً التابع main . وكل صف يحقق واجهة ما ، يلزمه بتحقيق جميع توابع الواجهة ، وفي حال عدم تحقيق جميع التوابع لابد من التصريح عن الصف على أنه صف مجرد abstract class .

٨-٢-٣- التصريح عن واجهة

يحتوي سطر التصريح على محدد الوصول الخاص بالواجهة إضافة إلى الكلمة المحجوزة interface ، ثم اسم الواجهة ، وأخيراً قائمة بالواجهات الأساسية Super interfaces لهذه الواجهة إن وجدت. المحدد public يُشير إلى أنه من الممكن استخدام هذه الواجهة من قبل أي صف في أي حزمة . إذا لم يتم التصريح عن الحزمة باستخدام public عندها ستُرى من قبل الصفوف والواجهات الموجودة في نفس الحزمة. ويمكن للواجهة أن توسع واجهة أخرى كما يوسع صف آخر ، مع الملاحظة أنه يمكن للواجهة توسيع أي عدد من الواجهات .ويتم فصل الواجهات باستخدام الفاصلة .

ترث الواجهة جميع الثوابت والتوابع من الواجهات الأساسية لها، ويمكن للواجهة إخفاء ثابت أو تابع عن طريق تعريف هذا الثابت أو التابع داخل الواجهة وبنفس الاسم . وهناك العديد من محددات الوصول التي لا يمكن استخدامها في الواجهات وهي :

- ✓ transient
- ✓ synchronized
- ✓ private
- ✓ protected

٨-٢-٤- جسم الواجهة

يحتوي جسم الواجهة تصريحات عن التوابع والثوابت المتضمنة داخل الواجهة . ويجب أن يكون التابع عاماً ومجرداً (ليس له جسم) وينتهي بفاصلة منقوطة ، لأن جميع توابع الواجهة هي توابع مجردة . يمكن لأي صف أن يستخدم الثوابت المعرفة داخل الواجهة على الشكل التالي :

Figure.x

ويمكن للصفوف التي تحقق الواجهة أن تتعامل مع ثوابت الواجهة مباشرة
كما يلي :

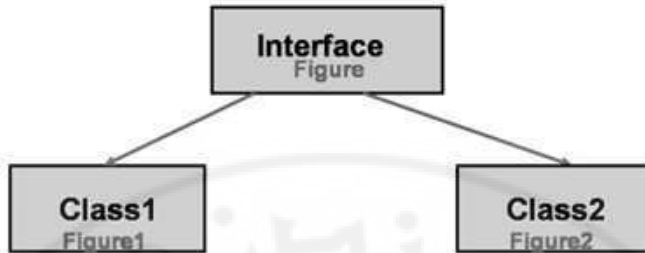
```
public class Myclass implements Figure {  
public void func( ) {  
// .....  
}  
public static void main( String [ ] args ) {  
System.out.println(x) ;  
// .....  
}  
} //End class
```

فوائد الواجهات

١. تُعدّ الواجهات من المكونات الأساسية المساعدة على البرمجة العامة ، فهي تستخدم لكتابة تطبيقات وخوارزميات يمكن تطبيقها على أي نوع من المعطيات ، وبالتالي يمكننا تمرير أي غرض من صف يحقق الواجهة ليحل محل نوع الواجهة في التطبيق .
٢. تُستخدم الواجهات لربط الصفوف غير المترابطة دون التأثير على البنية الهرمية للصفوف .
٣. تُستخدم الواجهات كبديل عن الوراثة المتعددة في Java .

مثال (2-8)

يوضح المثال التالي كيفية استخدام الواجهات التي تحقق الشكل التالي :



- تحتوي الواجهة Figure على تابعين هما :

public void Area() ;

public void Around() ;

وهي توابع عامة ومجردة .

إن الصف Figure1 يقوم بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث يقوم التابع Area() بحساب مساحة الشكل الهندسي " المستطيل " ، أما التابع الثاني Around() يقوم بحساب محيط المستطيل .

أما الصف Figure2 يقوم بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث يقوم التابع Area() بحساب مساحة الشكل الهندسي " الدائرة " ، أما التابع الثاني Around() يقوم بحساب محيط الدائرة .

البرنامج :

// Figure's interface

```

public interface Figure {
    public void Area(double param,double param1);
    public void Around(double param3,double param4);
}
  
```

```

public class Figure1 implements Figure {
  
```

// Implement Figure's interface

```

    public void Area(double p ,double p1) {
        double area =p*p1;
  
```

```

System.out.println("area of Rectangle = " + area);
}
public void Around(double p2, double p3) {
    double around = 2*(p2+p3);
    System.out.println("around of Rectangle = " + around);
}
} //End class

```

```

public class Figure2 implements Figure {
    // Implement Figure's interface
    public void Area(double p, double p1) {
        double area = 3.14*p*p;
        System.out.println("area of Circle = " + area);
    }
    public void Around(double p2, double p3) {
        double around = 2*3.14*p2;
        System.out.println("around of Circle = " + around);
    }
} //End class

```

```

public class TestInterface {
    public static void main(String args[ ]) {
        Figure1 ob = new Figure1( );
        ob.Area(4,5);
        ob.Around(4,5);
        Figure2 ob1 = new Figure2( );
        ob1.Area(4,5);
        ob1.Around(4,5);
    } //End main
} //End class

```

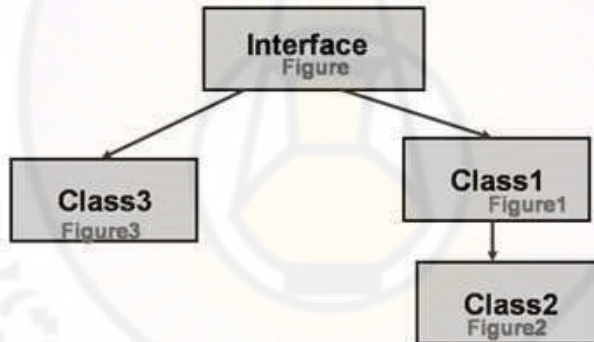
وعند تنفيذ البرنامج يظهر الخرج التالي :

Result

area of Rectangle = 20.0
around of Rectangle = 18.0
area of Circle = 50.24
around of Circle = 25.12
Process completed

مثال (3-8)

يوضح المثال التالي كيفية استخدام الواجهات التي تحقق الشكل التالي :



تحتوي الواجهة Figure على تابعين هما :

public void Area() ;

public void Around() ;

وهي توابع عامة ومجردة .

إن الصف Figure1 يحقق الواجهة ويقوم بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث التابع Area() يقوم بحساب مساحة الشكل الهندسي " المستطيل " ، أما التابع الثاني Around() يقوم بحساب محيط المستطيل .

أما الصف Figure2 يرث الصف Figure1 ويحقق الواجهة ، ويقوم بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث التابع Area() يقوم بحساب مساحة الشكل الهندسي " الدائرة " ، أما التابع الثاني Around() يقوم بحساب محيط الدائرة . أما الصف Figure3 يحقق الواجهة Figure ويقوم بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث التابع Area() يقوم بحساب مساحة الشكل الهندسي " المثلث " ، أما التابع الثاني Around() يقوم بحساب محيط المثلث .

يحتوي البرنامج على صف اختبار اسمه TestIface يحتوي على التابع main لإنشاء الأغراض واستدعاء التوابع اللازمة والحصول على النتائج .

البرنامج :

```
public interface Figure {
    public void Area( );
    public void Around( );
}
public class Figure1 implements Figure {
    double p, p1;
    public Figure1(double a, double b) {
        p = a ; p1 = b ;
    }
    // Implement Figure's interface
    public void Area( ) {
        double area = p*p1;
        System.out.println("area of Rectangle= " + area);
    }
    public void Around( ) {
        double around = 2*(p+p1);
        System.out.println("around of Rectangle = " + around);
    }
}
```

```

} //End class
public class Figure2 extends Figure1 implements Figure {
public Figure2(double a, double b) {
    super(a, b);}
// Implement Figure's interface
public void Area( ) {
    double area = 3.14*p*p;
    System.out.println("area of Circle= " +area );
}
public void Around( ) {
    double around =2*3.14*p;
    System.out.println("around of Circle= " + around);
}
} //End class
public class Figure3 implements Figure {
private double p,p1,p2;
public Figure3(double a, double b, double c) {
    p = a; p1 = b; p2 = c;
}
// Implement Figure's interface
public void Area( ) {
    double area =p*p1/2;
    System.out.println("area of Triangle = " + area);
}
public void Around( ) {
    double around =(p+p1+p2);
    System.out.println("around of Triangle = " + around);
}
} //End class

public class TestIface {

```

```

public static void main(String args[ ] ) {
Figure1 ob = new Figure1(4,5);
ob.Area( );
ob.Around( );
Figure2 ob1 = new Figure2(4,5);
ob1.Area( );
ob1.Around( );
Figure3 ob2 = new Figure3(4,5,6);
ob2.Area( );
ob2.Around( );
}//End main
}//End class

```

عند تنفيذ البرنامج يظهر الخرج التالي :

Result

area of Rectangle = 20.0
around of Rectangle = 18.0
area of Circle = 50.24
around of Circle = 25.12
area of Triangle = 10.0
around of Triangle = 15.0
Process completed

إذا كانت المتحولات p و p1 من نوع private ستكون التعديلات على البرنامج السابق كما يلي :

```

// Figure's interface
public interface Figure {
public void Area( );
public void Around( );
}

```

```

public class Figure1 implements Figure {
    private double p, p1;
    public Figure1(double a, double b) {
        p = a ; p1 = b ;
    }
    public double getp( ) { // تابع وصول
        return p ;
    }
    public double getp1( ) { // تابع وصول
        return p1 ;
    }
    // Implement Figure's interface
    public void Area( ) {
        double area = p * p1;
        System.out.println("area of Rectangle= " + area);
    }
    public void Around( ) {
        double around = 2*(p + p1 );
        System.out.println("around of Rectangle = " + around);
    }
} //End class

public class Figure2 extends Figure1 implements Figure {
    public Figure2(double a, double b) {
        super(a, b);
    }
    // Implement Figure's interface
    public void Area( ) { // استخدام توابع الوصول
        double area = 3.14* getp( ) * getp( );
        System.out.println("area of Circle= " + area);
    }
    public void Around( ) { // استخدام توابع الوصول
        double around = 2*3.14* getp( );
    }
}

```



```

System.out.println("around of Circle= " + around);
}
} //End class

public class Figure3 implements Figure {
    private double p,p1,p2;
    public Figure3(double a, double b, double c) {
        p = a; p1 = b; p2 = c;
    }
    // Implement Figure's interface
    public void Area( ) {
        double area = p*p1/2;
        System.out.println("area of Triangle = " + area);
    }
    public void Around( ) {
        double around = (p+p1+p2);
        System.out.println("around of Triangle = " + around);
    }
} //End class

public class TestIface {
    public static void main(String args[ ] ) {
        Figure1 ob = new Figure1(4,5);
        ob.Area( );
        ob.Around( );
        Figure2 ob1 = new Figure2(4,5);
        ob1.Area( );
        ob1.Around( );
        Figure3 ob2 = new Figure3(4,5,6);
        ob2.Area( );
        ob2.Around( );
    } //End main
} //End class

```

مسائل عامة محلولة

- ١- اكتب برنامجاً بلغة java يتضمن ما يلي :
 - واجهة interface اسمها Area تحتوي على تابع اسمه SideArea() ويعيد التابع قيمة من نوع String .
 - الصف Box (يمثل متوازي مستطيلات) ، معطاته الخاصة : طول القاعدة L وعرض القاعدة w والارتفاع h وهي من النوع double وهذا الصف يحقق الواجهة Area ويقوم بتجاوز التابع في الواجهة وإعادة تعريفه ، بحيث يقوم التابع SideArea() بحساب المساحة الجانبية لمتوازي المستطيلات وإسنادها إلى متحول من نوع String ، ويتضمن الصف أيضاً باني عادي.
 - الصف Cube (يمثل مكعب) يرث الصف Box ويحقق الواجهة Area ، ويقوم بتجاوز التابع في الواجهة وإعادة تعريفه، بحيث يقوم التابع SideArea() بحساب المساحة الجانبية للمكعب ، وإسنادها إلى متحول من نوع String، ويتضمن الصف أيضاً باني عادي.
 - الصف Circle (يمثل دائرة) ، معطاته الخاصة نصف قطر الدائرة وليكن r وهو من النوع double وهذا الصف يحقق الواجهة Area ويقوم بتجاوز التابع في الواجهة وإعادة تعريفه ، بحيث يقوم التابع SideArea() بحساب مساحة الدائرة ، وإسنادها إلى متحول من نوع String ويتضمن الصف أيضاً باني عادي .

- الصف Ball (يمثل كرة) يرث الصف Circle ويحقق الواجهة Area ، ويقوم بتجاوز التابع في الواجهة وإعادة تعريفه ، بحيث يقوم التابع SideArea() بحساب مساحة سطح الكرة وإسنادها إلى متحول من نوع String ويتضمن الصف باني عادي.
- صف اختبار اسمه TestArea يحتوي على التابع main لإنشاء الأغراض واستدعاء التوابع اللازمة والحصول على النتائج ، بحيث يتم إدخال النتائج عن طريق واجهة GUI وطباعة النتائج عن طريق واجهة GUI .

البرنامج :

```
import javax.swing.JOptionPane;
public interface Area {
    public String SideArea( );
}
public class Box implements Area {
    private double L,w,h;
    public Box (double L1,double w1,double h1) {
        L=L1;    w=w1;    h=h1;
    }
    public double getL() {return L;}
    public double getw() {return w;}
    public double geth() {return h;}
    // Implement Figure's interface
    public String SideArea( ) {
        double area = 2*L*w+2*w*h+2*L*h;
        String output= "area =" + area + "\n";
        return output;
    }
    //End class
```

```

public class Cube extends Box implements Area {
public Cube(double L1,double w1,double h1) {
    super(L1,w1,h1);
}
// Implement Figure's interface
public String SideArea() {
    double area = (getL()*getL())*6;
    String output= "area="+ area+"\n";
return output;
}
} //End class

public class Circle implements Area {
private double r;
public Circle(double r1) {
    r = r1;
}
// Implement Figure's interface
public String SideArea() {
    double area =Math.PI*r*r;
    String output= "area =" + area+"\n";
return output;
}
public double getr() {
    return r;
}
} //End class

public class Ball extends Circle implements Area {
public Ball (double a) {
    super(a);
}
}

```

// Implement Figure's interface

```
public String SideArea( ) {  
    double area =4*Math.PI*getr( )*getr( );  
    String output= "area =" + area+"\n";  
    return output;  
}  
//End class
```

```
public class TestArea {  
    public static void main(String args[ ] ) {  
        double L1,W1, h1;  
        String javaString = JOptionPane.showInputDialog(null,  
            "Enter a value of L1 ", "My Area Example",  
            JOptionPane.QUESTION_MESSAGE);  
        L1 = Double.parseDouble(javaString) ;  
        javaString = JOptionPane.showInputDialog(null,  
            "Enter a value of W1 ", "My Area Example",  
            JOptionPane.QUESTION_MESSAGE);  
        W1 = Double.parseDouble(javaString);  
        javaString = JOptionPane.showInputDialog(null,  
            "Enter a value of h1 ", "My Area Example",  
            JOptionPane.QUESTION_MESSAGE);  
        h1 =Double.parseDouble(javaString);  
        //-----  
        Box ob1 = new Box(L1,W1,h1);  
        Cube ob2 = new Cube(W1,W1,W1);  
        Circle ob3 = new Circle(L1);  
        Ball ob4 = new Ball (L1);  
        String output =" \n The result is:\n";  
        output=output+" \n Area of Box is \n "+ob1.SideArea( );  
        output=output+" \n Area of Cube is \n "+ ob2.SideArea( )+"\n";
```



```

output=output+"\n Area of Circele is \n "+ ob3.SideArea( )+"\n";
output=output+"\n Area of Ball is \n "+ ob4.SideArea( )+"\n";
JOptionPane.showMessageDialog(null, output, "My Areas ",
    JOptionPane.ERROR_MESSAGE );
    } //End main
} //End class

```

٢- أكتب برنامجاً بلغة Java يتضمن ما يلي:

- واجهة interface وليكن اسمها Figure تحتوي على تابعين هما : Area() و double Volume () .
- صفئاً أساسياً اسمه Circle (دائرة) يحقق الواجهة Figure ويحتوي على متحولين خاصين private هما : r (نصف قطر الدائرة) و h (ارتفاع الأسطوانة والمخروط) وهما من النوع double ، ويحتوي الصف على التوابع الأعضاء العامة التالية : باني افتراضي يُسند القيمة (1) للمعطيات الخاصة، باني عادي ، وتابع لحساب مساحة الدائرة وليكن اسمه CircleArea ، ويقوم الصف Circle بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث يقوم التابع Area() بحساب مساحة الشكل الهندسي " الكرة"، أما التابع الثاني Volume() يقوم بحساب حجم الكرة .
- صفئاً مشتقاً وليكناسمه Cone (مخروط دوراني) يرث الصف الأساسي Circle و يحتوي على التوابع الأعضاء العامة التالية: بان افتراضي ، باني عادي ، ويقوم الصف Cone بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث يقوم التابع Area() بحساب المساحة الكلية للمخروط الدوراني و يقوم التابع Volume() بحساب حجم المخروط الدوراني .
- صفئاً مشتقاً وليكناسمه Cylinder (أسطوانة) يرث الصف Circle ويحقق الواجهة ويحتوي على التوابع الأعضاء العامة التالية: بان افتراضي ، بان

عادي ، ويقوم الصف Cylinder بتجاوز التابعين في الواجهة وإعادة تعريفهما ، بحيث يقوم التابع Area() بحساب المساحة الكلية للأسطوانة أما التابع الثاني Volume() يقوم بحساب حجم الأسطوانة .

- صفّاً للاختبار وليكن اسمه VolumeTest يحتوي على التابع الرئيسي main ، والذي بدوره يتضمن إنشاء أغراض من الصفوف السابقة من أجل تمرير قيماً لنصف القطر والارتفاع و استدعاء التوابع اللازمة من أجل حساب وطباعة المساحة والحجم للأشكال الهندسية آنفة الذكر . بحيث يتم إدخال النتائج عن طريق واجهة GUI وطباعة النتائج عن طريق واجهة GUI .

البرنامج :

```
import javax.swing.JOptionPane;
public interface Figure1 {
    double Area();
    double Valume();
}
public class Circle implements Figure1 {
    private double r,h;
    public Circle(double r1, double h1) {
        r = r1;
        h = h1;
    }
    public Circle( ) { r = 1;h = 1;}
    public double getr( ) { return r ;}
    public double geth( ) {return h ;}
    // Implement Figure's interface
    public double CircleArea( ) {
        double area =Math.PI*r*r;
        return area;
```

```

}
public double Area( ) {
    double area = 4*Math.PI*r*r;
    return area;
}
public double Valume( ) {
    double valume = 4/3.0*(Math.PI*r*r*r);
    return valume;
}
} //end class

public class Cylinder extends Circle implements Figure1 {
public Cylinder(double r1, double h1) {
    super(r1, h1);
}
public Cylinder( ) {
    super( );
}
// Implement Figure's interface
public double Area( ) {
    double area1 = 2*Math.PI*getr()*geth( );
    double area2 = 2*Math.PI*getr()*getr( );
    double area = area1 + area2;
    return area;
}
public double Valume( ) {
    double valume = Math.PI*getr()*getr()*geth( );
    return valume;
}
} //end class

```

```

public class Cone extends Circle implements Figure1 {
public Cone(double r1, double h1) {
    super(r1, h1);
}
public Cone( ) {
    super( );
}
// Implement Figure's interface
public double Area( ) {
    double s=getr()*getr()+geth()*geth();
    double sid=Math.sqrt(s);
    double area1=Math.PI*getr()*sid;
    double area2=Math.PI*getr()*getr();
    double area =area1+ area2;
    return area;
}
public double Valume( ) {
    double valume = (1.0/3.0)*Math.PI*getr()*getr()*geth();
    return valume;
}
} //end class

```

```

public class TestIfaceGUI {
    public static void main(String args[]) {
        double r1,h1;
        String javaString = JOptionPane.showInputDialog(null,
            "Enter a value of r ", "My Circele Example",
            JOptionPane.QUESTION_MESSAGE);
        r1 = Double.parseDouble(javaString);
        javaString = JOptionPane.showInputDialog(null,
            "Enter a value of h ", "My Circele Example",

```



```

JOptionPane.QUESTION_MESSAGE);
h1 = Double.parseDouble(javaString) ;
//-----
Circle ob1 = new Circle(r1,h1);
String output =" \n The output is\n";
output=output+"Area of Circle = " + ob1.CircleArea( )+"\n";
output=output+"Area of shape = " + ob1.Area( )+"\n";
output=output+"Valume of shape = "+ob1.Valume( )+"\n";
/*-----*/
Cylinder ob2 = new Cylinder(r1,h1);
output=output+"Area of Cylinder= " +ob2.Area( )+"\n";
output=output+"Valume of Cylinder= " +ob2.Valume( )+"\n";
//-----
Cone ob3 = new Cone(r1,h1);
output=output+ "Area of Cone = "+ob3.Area()+"\n";
output=output+"Valume of Cone = "+ob3.Valume();
JOptionPane.showMessageDialog(null, output,
"My Figure1 interface ", JOptionPane.ERROR_MESSAGE );
}

```

=====

=====

=====

=====





مسائل عامة



١- اكتب برنامجاً بلغة Java يعرف صفّاً أساسياً مجرداً اسمه **Credit** يحتوي على ستة وسطاء $x1, x2, x3, x4, x5, x6$ من النوع `int` كأعضاء محمية `protected` وهي عبارة عن درجات المواد الدراسية للفصل الدراسي الأول ، حيث $x1, x2, x3, x4$ هي مقررات اختصاصية، ويحتوي الصف على بان افتراضي وعلى بان عادي و تابع مجرد `abstract` اسمه `( abstCredit )` .

• اشتق من الصف الأساسي صفّاً مشتقاً اسمه `sub1` وعند تجاوز الصف المشتق التابع المجرّد `( abstCredit )` من الصف الأساسي سيقوم بحساب معدل درجات الطالب في المقررات الاختصاصية. ويطبع كلمة ناجح `Passed` إذا كان معدلة أكبر أو يساوي 60، ويطبع كلمة راسب `Failed` إذا كان أصغر من 60.

• اشتق من الصف `sub1` صفّاً مشتقاً اسمه `sub2` و عند تجاوز الصف المشتق التابع المجرّد `( abstCredit )` يقوم بحساب معدل الطالب في المقررات الستة ويطبع كلمة ناجح `Passed` إذا كان معدلة أكبر أو يساوي 60، ويطبع كلمة راسب `Failed` إذا كان أصغر من 60.

• يحتوي البرنامج على الصف الاختيائي `Abstractclass` الذي يحتوي على التابع الرئيسي `main` ثم نصرّح في التابع الرئيسي `main` عن اغراض في الصفوف المشتقة واستدعاء التوابع اللازمة للحصول على النتائج .

٢- اكتب برنامجاً بلغة Java لإنشاء صف أساسي مجرد اسمه Formula يحتوي على ثلاثة وسطاء خاصة private ، حيث x من نمط double و n و m من نمط int ، ويحتوي الصف على باني افتراضي وباني عادي وتابع مجرد اسمه () Print .

• اشتق من الصف الأساسي صفاً مشتقاً اسمه FormulaSum وعند تجاوز الصف المشتق التابع المجرد () Print من الصف الأساسي سيقوم بحساب المقدار $\sum_{i=1}^n x^i$ ويطبع الناتج .

• اشتق من الصف FormulaSum صفاً مشتقاً اسمه FormulaFact وعند تجاوز الصف المشتق التابع الافتراضي () Print يقوم بحساب المقدار: $\frac{m!}{n!}$ ويطبع النتيجة .

• يحتوي البرنامج على الصف الاختيائي Abstractclass الذي يحتوي على التابع الرئيسي main ثم نصح في التابع الرئيسي main عن اغراض في الصفوف المشتقة واستدعاء التوابع اللازمة للحصول على النتائج .

=====

٣- أنشئ صفاً أساسياً مجرداً بلغة Java اسمه Student يحتوي على ثلاث درجات D1,D2,D3 لثلاثة مقررات وهي وسطاء من النوع Double كأعضاء خاصة private ، ويحتوي على بان افتراضي وبان عادي وتابع مجرد abstract واحد اسمه () Print .

• اشتق من الصف الأساسي Student صفاً مشتقاً اسمه Maximum ، وعند تجاوز الصف المشتق التابع المجرد () Print يقوم بحساب وطباعة أكبر درجة .

- اشتق من الصف المشتق الأول Max صفاً مشتقاً اسمه Minimum ،
و عند تجاوز الصف المشتق الثاني التابع المجرد () Print() ، يقوم بحساب
وطباعة أصغر درجة .
- يحتوي البرنامج على الصف الاختبائي Abstractclass الذي يحتوي على
التابع الرئيسي main ثم نصرح في التابع الرئيسي main عن اغراض في
الصفوف المشتقة واستدعاء التوابع اللازمة للحصول على النتائج .



الفصل التاسع

بناء واجهة المستخدم المرئية

Building Graphical User Interface





بناء واجهة المستخدم المرئية

Building Graphical User Interface

٩-١ - مقدمة

تقدم لغة Java مجموعة غنية من الصفوف التي تساعد في بناء واجهات المستخدم المرئية GUI ، ونستطيع استخدام العديد من صفوف GUI المبنية مسبقاً مثل (الاطارات frames ، واللوحات panels ، والأزرار Buttons ، وحقول النص text fields ، ومساحات النص text Areas ، وصناديق الخيارات المتاحة Combo Boxes ، وصناديق الاختيار check boxes ، وأزرار الخيار radio buttons ، والقوائم menus ، وأشرطة التمرير scroll bars ، واللوحات المجدولة tabbed panels) من أجل بناء واجهات المستخدم .

وتقدم لنا لغة Java حزمتين من أجل البرمجة المرئية هما java.awt (واجهات التخاطب مع المستخدم) و javax.swing (واجهات المستخدم) ، وتتضمن كل حزمة مجموعة من الصفوف والواجهات التي ستساعدنا في بناء واجهات المستخدم المرئية .

٩-٢ - بعض المكونات المرئية البسيطة

سنتعرف في هذا الفصل على أساسيات برمجة Java GUI ، وسنركز بشكل خاص على عناصر GUI ، وعلى العلاقات بينها وعلى الحاويات containers ، وعلى برامج إدارة التخطيط (التنسيق) ، وعلى الألوان والخطوط ، وسنتعلم كيفية رسم الأشكال الهندسية مثل (الخطوط ، والمستطيلات ، والأشكال البيضاوية ، والأقواس ، والمضلعات polygons ، ومتعددات الخطوط) .

قبل القيام بالتعرف على كيفية كتابة برنامج يتضمن واجهة مرئية ، فإننا سنقوم بالتعرف على أهم العناصر المرئية مع ذكر الصفوف التي تنتمي إليها، والتي نلزمنا عند بناء الواجهات . ويبين الجدول (٩-١) عناصر البيئة المرئية .

اسم الصف	استخدامها	اسم الأداة
JTextField	إدخال نص من قبل المستخدم	حقل نص
JLabel	عرض نص ثابت	لاصقة (لافتة)
JRadioButton	اختيار عنصر من مجموعة عناصر	زر اختيار
JCheckBox	اختيار عنصر أو أكثر من مجموعة عناصر	صندوق اختيار
JPanel	تجميع عدد من العناصر	إطار
JComboBox	تجميع عدد من العناصر على شكل قائمة	صندوق اختيار متاح
JPasswordField	إخفاء النص المدخل من قبل المستخدم	حقل كلمة المرور
JTextArea	إدخال نص مكون من عدة أسطر	مساحة نص
JProgressBar	إظهار تقدم عملية معينة	شريط التقدم
JButton	تنفيذ امر معين عند الضغط عليه	زر
JList	تجميع عدد من العناصر على شكل قائمة	قائمة

الجدول (٩-١) - يبين عناصر البيئة المرئية

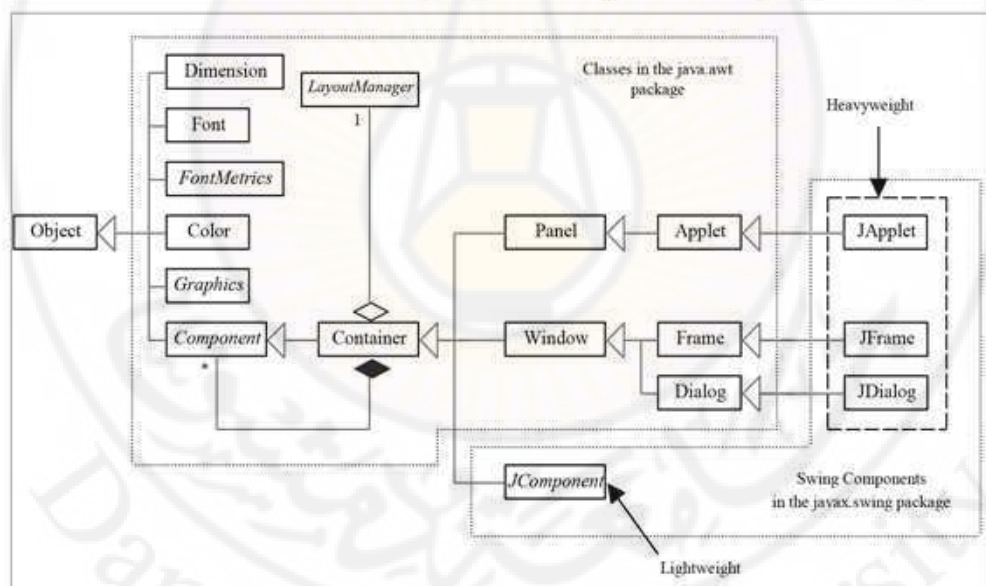
٩-٣- واجهة برمجة تطبيقات Java GUI

يمكن تصنيف الصفوف التابعة لواجهة المستخدم المرئية إلى ثلاث

مجموعات ، وهي :

- الصفوف الحاوية Container Classes
- الصفوف المساعدة Helper Classes
- صفوف العناصر Elementary Classes

وتستخدم الصفوف الحاوية مثل (JFrame و JPanel و JApplet) من أجل ضم عناصر أخرى ، وتستخدم الصفوف المساعدة مثل (Graphics و Font و Color و FontMetrics و Dimension) من قبل العناصر والحاويات من أجل رسم الأغراض وتحديد مكان توزيعها . أما صفوف عناصر واجهة المستخدم مثل (JButton و JList و JTextField و JLabel و JRadioButton و JCheckBox و JPanel و JComboBox و JPasswordField و JTextArea و JMenu) وهي صفوف مشتقة من الصف الأساسي JComponent . ويبين الشكل (٩-١) المخطط الهرمي للصفوف الحاوية والمساعدة التي تعتمد عليها برمجة Java GUI .



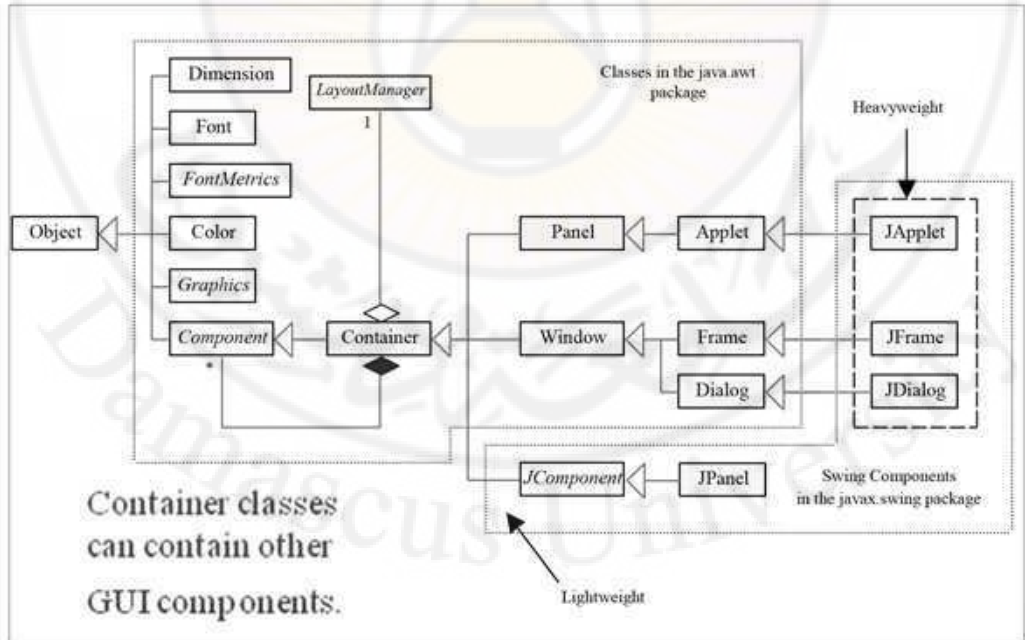
(٩-١) - يبين المخطط الهرمي للصفوف الحاوية والمساعدة

٩-٣-١ - الصفوف الحاوية

إن الصفوف Window و Panel و Applet و Dialog و Frame هي صفوف حاوية لعناصر الحزمة java.awt. لتتمكن هذه الصفوف مع عناصر الحزمة javax.swing من استخدام الصفوف JFrame و JPanel و JApplet و Container و JDialog ، ويبين (٩-٢) الصفوف الحاوية .

يُستخدم الصف **Container** من أجل تجميع العناصر ، ويتم استخدام مدير شكل التخطيط ضمن الحاوية Container من أجل وضع العناصر وترتيبها وفق المكان والنمط المرغوب بهما .

والصف **JFrame** عبارة عن نافذة لا يمكن وضعها ضمن نافذة أخرى . ويمثل هذا الصف الحاوية التي تحمل عناصر واجهة المستخدم المرئية التابعة للحزمة . javax.swing



(٩-٢) - يبين الصفوف الحاوية

والصف **JDialog** عبارة عن صندوق رسالة (أو نافذة) يُستخدم بشكل عام كنافذة مؤقتة من أجل استقبال معلومات إضافية من المستخدم أو من أجل تقديم تنبيه عن وقوع حدث ما .

والصف **JApplet** صف فرعي من الصف **Applet** . ويجب علينا وراثة الصف **Applet** من أجل صنع بريميج **Java** يعتمد على الحزمة **java.swing** . والصف **JPanel** عبارة عن حاوية غير مرئية يُستخدم من أجل حمل عناصر واجهة المستخدم ، كما أنه يستخدم من أجل رسم الخطوط والصور وعناصر الرسوم الأخرى .

٩-٣-٢ - الصفوف المساعدة

إن الصفوف المساعدة مثل **Graphics** و **Font** و **Color** و **FontMetrics** و **Dimension** (ليست صفوفاً مشتقة من الصف **Component** ، وتُستخدم هذه الصفوف من أجل وصف خصائص عناصر **GUI** مثل محتوى الرسوم ولونها ونوع الخط المستخدم وأبعادها .

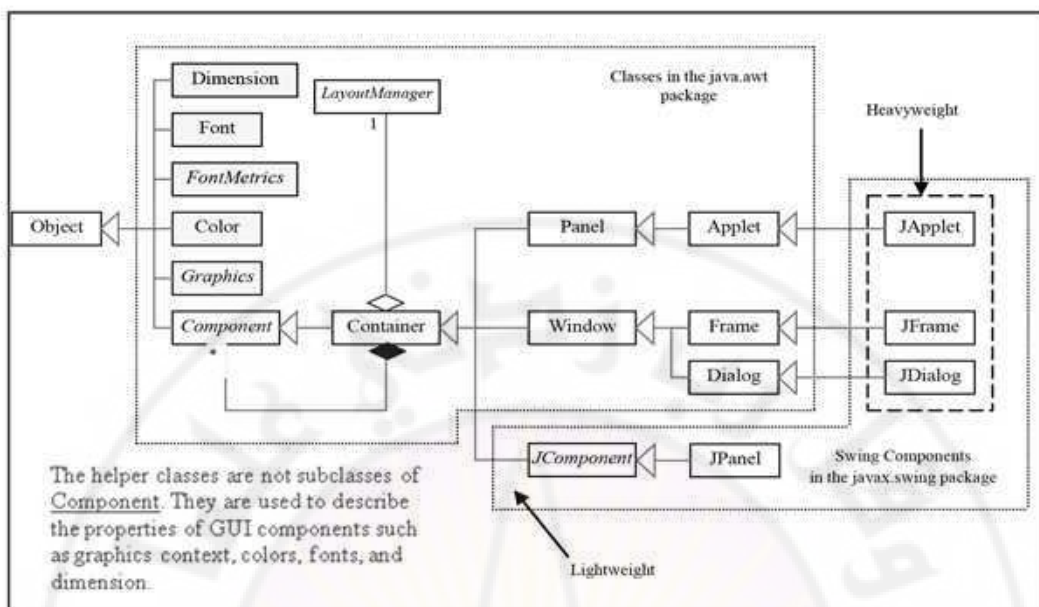
والصف **Graphics** صف مجرد يقدم محتوى رسم من أجل رسم النصوص والخطوط والأشكال البسيطة . ويبين الشكل (٩-٣) الصفوف المساعدة .

و يتعامل الصف **Color** مع ألوان عناصر **GUI** ، وتستطيع مثلاً تحديد الألوان الخلفية والألوان الأمامية للعناصر مثل **JFrame** و **JPanel** أو يمكننا تحديد ألوان الخطوط والأشكال والنصوص ضمن الرسوم المختلفة .

و يُستخدم الصف **Font** من أجل تحديد الخطوط المستقيمة من أجل النص والرسوم المختلفة ضمن عناصر **GUI** .

والصف **FontMetrics** صف مجرد يُستخدم من أجل الحصول على خصائص الخط .

و يغلف الصف **Dimension** عرض وارتفاع العنصر ضمن غرض وحيد.

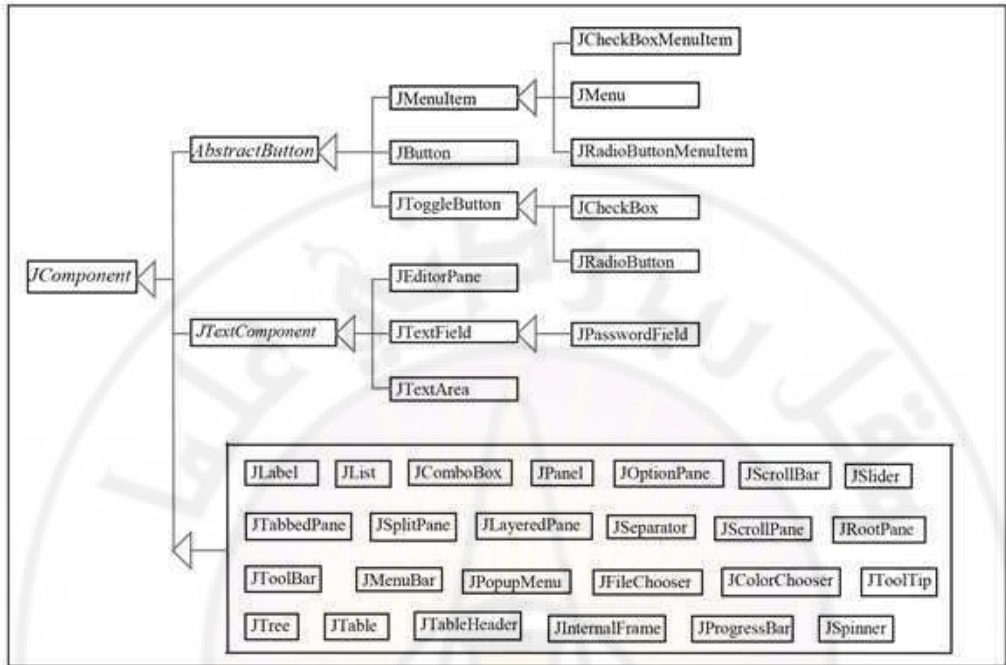


الشكل (٩-٣) - يبين الصفوف المساعدة

٩-٤ - عناصر واجهة المستخدم Swing GUI

يمثل الصف `Component` الصف الأساسي لجميع صفوف الواجهة المرئية ، ويمثل الصف `JComponent` الصف الأساسي لجميع عناصر الحزمة `javax.swing` . بما أن الصف `JComponent` هو صف مجرد لذلك لا نستطيع استخدام العبارة `new JComponent` من أجل إنشاء غرض منه .

أما الآن سنقوم ببناء أول واجهة مرئية باستخدام لغة `Java` و ذلك من خلال إدخال الاسم الأول والكنية والعمر ، وبعدها يقوم التطبيق بعرض المعطيات التي أدخلها المستخدم . ويبين الشكل (٩-٤) عناصر واجهة المستخدم `Swing GUI` .



الشكل (٩-٤) - يبين عناصر واجهة المستخدم Swing GUI

ويبين البرنامج في المثال (٩-١) صفاً اسمه Identity ويحتوي على التابع main حيث يتم إدخال الاسم والكنية والعمر بواسطة واجهة ، وإخراجها بواسطة واجهة أيضاً .

مثال (٩-١)

```

import javax.swing.*;

public class Identity {
    private String first_name , second_name ;
    private int age;
    private String identity;
    public static void main(String args[ ] ){
        Identity wind = new Identity ( );
        wind.first_name = JOptionPane.showInputDialog(null,
            "Enter your first name: ");
    }
}
  
```

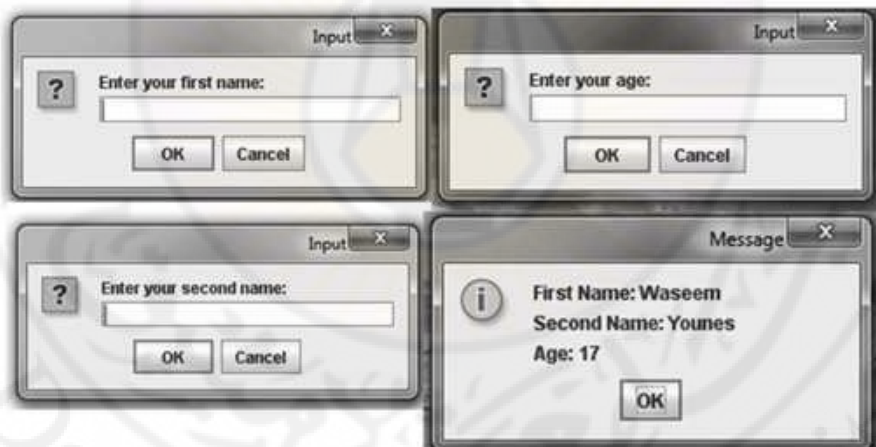
```

wind.second_name = JOptionPane.showInputDialog(null,
                                                "Enter your second name: ");
wind.age = Integer.parseInt(
    JOptionPane.showInputDialog(null, "Enter your age: "));
wind.identity = "First Name: " + wind.first_name + "\n"
               + "Second Name: " + wind.second_name + "\n"
               + "Age: " + wind.age;

JOptionPane.showMessageDialog(null, wind.identity);
System.exit(0);
} // End main
} // End class

```

وعند تنفيذ البرنامج سنحصل على الواجهات التالية:



نلاحظ أننا قمنا باستيراد الحزمة `javax.swing` وذلك من أجل استخدام الصف `JOptionPane` وهو أحد صفوف البرمجة المرئية ، حيث يتضمن هذا الصف :
 التابع `showInputDialog` المسؤول عن إظهار نافذة بسيطة تسمح للمستخدم بإدخال قيمة معينة . أما التابع `showMessageDialog` فوظيفته إظهار رسالة ما للمستخدم .

نلاحظ أننا استخدمنا التابع `exit(0)` ، هذا التابع يؤدي إلى إنهاء تنفيذ البرنامج والخروج منه ، وبدونه سيبقى البرنامج في الذاكرة ، وهو موجود في الصف `System` .

وسوف يتم شرح كيفية إنشاء إطار في واجهة المستخدم .

٩-٤-١ - الإطارات Frames

نحتاج من أجل إنشاء واجهة المستخدم إلى إطار يحمل عناصر واجهة المستخدم ، ويتم عرض الإطار افتراضياً في الزاوية اليسارية العليا من الشاشة ، وإن إحداثيات الزاوية اليسارية العليا هي $(0, 0)$ حيث تتزايد القيمة على المحور x من اليسار إلى اليمين ، وتتزايد القيمة على المحور y من الأعلى إلى الأسفل ، ويمكن استخدام التابع `setLocation(x,y)` من الصف `JFrame` من أجل عرض الإطار في المكان الذي نرغب به ، حيث يقوم هذا التابع بوضع الزاوية اليسارية العليا للإطار في المكان الموافق للإحداثيات (x, y) من الشاشة .

وبما أن الصف `JFrame` موجود في الحزمة `javax.swing` لذلك نستخدم عبارة الاستيراد `( javax.swing.* )` التي تجعل جميع الصفوف التابعة لهذه الحزمة متاحة ضمن هذا البرنامج بما فيها الصف `JFrame` . ويمكن استخدام البانين التاليين من أجل إنشاء غرض من الصف `JFrame` .

١. يقوم الباني `( JFrame() )` ببناء غرض `JFrame` يمثل الواجهة الرئيسية بدون عنوان .

٢. يقوم الباني `( JFrame( " Welcome" ) )` ببناء غرض يمثل الواجهة الرئيسية بعنوان محدد ، ويظهر هذا العنوان على شريط العنوان التابع للإطار .

□ لا يظهر الإطار حتى يتم استدعاء التابع `( setVisible(true) )` .

- ❑ يحدد التابع `fream.setSize(500,400)` عرض الإطار (500) بكسل وارتفاعه (400) بكسل ، وإذا لم نستخدم التابع `setSize` عندها تتحدد مساحة الإطار بحيث يتم عرض شريط العنوان التابع له فقط .
 - ❑ بما ان التابعين `setSize` و `setVisible` معرفان في الصف `Component` ، لذلك فإن الصف `JFrame` يرثهما .
 - ❑ يخبر التابع التالي البرنامج بأن ينتهي عند إغلاق الإطار (النقر على زر الإغلاق) .
- `window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`
- ويوضح المثال (9 - 2) كيفية انشاء إطار .

مثال (2-9)

```

1. import javax.swing.*;
2. public class MyWindows1{
3.     public static void main(String args[ ] ) {
4.         String statement =
           "Hello Mamoun Younes! My first interface.";
5.         JFrame window = new JFrame("Welcome");
6.         JLabel text = new JLabel(statement);
7.         window.add(text);
8.         window.pack( );
9.         window.setVisible(true);
10.        window.setDefaultCloseOperation(
           JFrame.EXIT_ON_CLOSE );
11.    } //End main
12.} //End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



لنشرح بعض اسطر البرنامج :

السطر 1 : تم استيراد الحزمة javax.swing والتي تُستخدم من أجل بناء عناصر الواجهة المرئية كالأزرار والقوائم وغيرها .

السطر 5 : عرفنا غرضاً window من الصف JFrame ، هذا الغرض يمثل الواجهة (النافذة) الرئيسية ، التي سيتم إضافة العناصر المرئية الأخرى عليها . لقد مررنا إلى باني الصف سلسلة محرفية ، وهذه السلسلة المحرفية ستظهر في شريط العنوان للنافذة .

السطر 6 : عرفنا غرضاً text على أنه لاصقة (لافتة) من الصف JLabel ، ومررنا إلى باني هذا الصف سلسلة محرفية تمثل النص الذي سيظهر للمستخدم

السطر 7 : استخدمنا التابع add من أجل إضافة العنصر text إلى النافذة window التي قمنا بإنشائها .

السطر 8 : إذا حاولنا تنفيذ البرنامج بدون استدعاء التابع pack لن يظهر من النافذة إلا شريط عنوانها فقط ، لذلك نحن نستخدم هذه الطريقة التي تقوم بحساب الحجم المناسب للنافذة من خلال حساب الحجم المفضل للمكونات الموجودة فيها .

السطر 9 : تم استدعاء التابع setVisible مع الوسيط true من أجل إظهار الغرض window (أي إظهار نافذة) .

السطر 10: استخدمنا التابع setDefaultCloseOperation من أجل تحديد سلوك النافذة window عندما يضغط المستخدم على زر إغلاق النافذة حيث نمرر لها أحد الثوابت التالية :

DO_NOTHING_ON_CLOSE □ : لن يتم إغلاق النافذة إذا ضغط المستخدم على زر الإغلاق ويتم إغلاق النافذة برمجياً ضمن التابع windowClosing.

□ **HIDE_ON_CLOSE** : ستختفي النافذة ولكنها ستبقى في الذاكرة وستبقى موارد الذاكرة محجوزة .

□ **DESPOSE_ON_CLOSE** : ستختفي النافذة وستحرر موارد الذاكرة المخصصة لها .

□ **EXIT_ON_CLOSE** : سيتم الخروج النهائي من التطبيق باستخدام التابع `System.exit` .

عندما نريد تحديد عرض الإطار نستخدم التابع `setSize(400,300)` الذي يحدد عرض الإطار بـ (400) بكسل وارتفاعه (300) بكسل ، على سبيل المثال . لذلك نضيف التابع إلى البرنامج في المثال (9 - 2) ، ونحذف التابع `pack` ، ويبين المثال (9 - 3) هذا التعديل والإطار الجديد لواجهة الخرج .

مثال (9-3)

```
1. import javax.swing.*;
2. public class MyWindows2 {
3.     public static void main(String args[ ]) {
4.         String statement =
           "Hello Mamoun Younes! My first interface.";
5.         JFrame window = new JFrame("Welcome");
6.         JLabel text = new JLabel(statement);
7.         window.add(text);
8.         window.setSize(400,300); // هنا التعديل
9.         window.setVisible(true);
10.        window.setDefaultCloseOperation(
           JFrame.EXIT_ON_CLOSE );
11.    } //End main
12.} //End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٩-٤-٢ - إضافة العناصر إلى الإطار

رأينا في المثال (9-3) كيفية استخدام التابع `add` لإضافة لافتة `Lable` إلى النافذة الرئيسية ، ويمكننا من خلال التابع `add` إضافة أي عنصر آخر إلى النافذة بأسلوب مشابه . إن إضافة العناصر لا تكون فقط إلى النافذة الرئيسية ، وإنما قد تكون إلى أي عنصر حاو `Container Component` ، حيث يوجد عدة أنواع للحاويات ولكن ما يهمنا منها هو حاويات المستوى الأعلى `Top Level Container` مثل : `JPanel` و `JApplet` و `JDialog` ، وهذا النوع من الحاويات يمكن أن يحوي أي عنصر آخر . ويبين المثال (9-4) كيفية إضافة زر ، وحقل نص ، ولافتة `Lable` إلى واجهة التطبيق (الإطار) .

مثال (9-4)

//Add components into a frame

1. **import** javax.swing.*;
2. **import** java.awt.*;
3. **public class** MyWindows3 {
4. **public static void** main(**String** args[]) {

```

5. JFrame wind = new JFrame("Add Components Into");
6. JButton btn = new JButton("Click her");
7. JTextField textfld = new JTextField( );
8. JLabel lbl = new JLabel("Enter your age");
9. wind.setLayout(new GridLayout( ) );
10. wind.add(btn);
11. wind.add(textfld);
12. wind.add(lbl);
13. wind.pack( );
14. wind.setVisible(true);
15. wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16. } //End main
17. } //End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



لنشرح الأوامر الجديدة في هذا المثال :

السطر 1 : تم استيراد الحزمة java.awt ، لأننا نستخدم مدير التخطيط (التنسيق) في السطر 9 ، وسوف نتحدث عنه لاحقاً .

الأسطر 6,7,8 : تم تعريف ثلاثة مكونات (أغراض) هي :
 btn من الصفوف JButton , textfld من JLabel , lbl على الترتيب .

السطر 9 : وسوف نتحدث عنه لاحقاً في فقرة مدراء التنسيق .

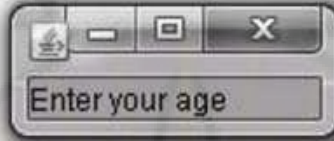
الأسطر 10,11,12 : نقوم بإضافة المكونات (الأغراض) الثلاثة السابقة التي قمنا بتعريفها إلى غرض النافذة wind .

٩-٤-٣ - مدراء التنسيق Layout Managers

قبل التحدث عن مدراء التنسيق لننفذ المثال (9 -4) بدون كتابة السطر 9 وهو :

```
9 wind.setLayout(new GridLayout( ) ) ;
```

وعند تنفيذ البرنامج ستكون نافذة الخرج على الشكل التالي :



السؤال أين يظهر الزر JButton("Click her") وحقل النص JTextField ؟
في الواقع إن كل منهما قد تم إضافته إلى النافذة wind ولكنهما لم يظهرًا وإنما ظهر فقط آخر عنصر تمت إضافته إلى النافذة ، والسبب في ذلك هو مدير التنسيق Layout Manager . وعندما يتم وضع عناصر Java GUI ضمن الحاوية عندها تقوم إدارة التنسيق التابعة لهذه الحاوية بتنظيمها وترتيبها . إذاً إن مدير التنسيق لنافذة (حاوية) هو المسؤول عن تحديد موضع وأبعاد العناصر التي يتم إضافتها إلى هذه النافذة (الحاوية) . والسؤال :

- لماذا نترك مهمة ترتيب العناصر ، وضبط أبعادها إلى مدير التنسيق ؟
- هل يمكن أن نقوم بضبط موضع وأبعاد العناصر دون اللجوء إلى مدير التنسيق ؟

في الواقع إن مدير التنسيق يُسهل علينا عملية تنسيق العناصر (أي تحديد موضعها وأبعادها) ، لتوضيح هذا المفهوم سنتعرف على مدراء التنسيق المختلفة التي توفرها لغة Java ، وهي :

- BorderLayout - CardLayout - GridBagLayout
- FlowLayout - GridLayout - BoxLayout

يعرّف المثال (9 -5) التالي مصفوفة من الأزرار Buttons ، ونضيفها إلى نافذة البرنامج ، ولكن دون أن نستخدم مدير التنسيق بشكل صحيح ، ثم سنقوم في الفقرات التالية بإضافة أحد مدراء التنسيق المختلفة . عند تنفيذ هذا المثال لا نرى أكثر من النافذة التالية .



مثال (5-9)

```
import javax.swing.*;
import java.awt.*;

public class MyLayoutManager {
    private static JButton btns [ ] = new JButton[8];
    public static void main(String args[ ]) {
        JFrame wind = new JFrame("Layout");
        // Here we will add Layout Manager
        for(int i =0; i<8; i++) {
            btns[i] = new JButton(Integer.toString(i));
            wind.add(btns[i]);
        }
        wind.pack( );
        wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } // End main
} // End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٩-٤-٣-١ - مدير التنسيق FlowLayout

هو مدير التنسيق الافتراضي للنافذة ، ويقوم مدير التنسيق FlowLayout بتنسيق العناصر وترتيبها من اليسار إلى اليمين وبنفس الترتيب الذي أضيفت به ، وعندما ينتهي السطر تنتقل العناصر إلى سطر جديد . ويمكننا تحديد الطريقة التي سيتم بها محاذاة العناصر باستخدام أحد الثوابت التالية:

- FlowLayout.LEFT
- FlowLayout.CENTER
- FlowLayout.RIGHT

لنضيف إلى المثال (9-5) السطر التالي الذي يحدد مدير التنسيق الواجب استخدامه في تنسيق العناصر وترتيبها ، كما هو موضح في المثال (9-6) .
wind.setLayout(new FlowLayout());

مثال (9-6)

```
import javax.swing.*;
import java.awt.*;

public class MyLayoutManager1 {
    private static JButton  btns [ ] = new JButton[8];
    public static void main(String args[ ]) {
        JFrame  wind = new JFrame("Layout");
        // Here we will add Layout Manager
        wind.setLayout(new FlowLayout( ) );
        for(int i =0; i<8; i++) {
            btns[i] = new JButton(Integer.toString(i));
            wind.add(btns[i]);
        }
        wind.pack( );
        wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```

} // End main
} // End class

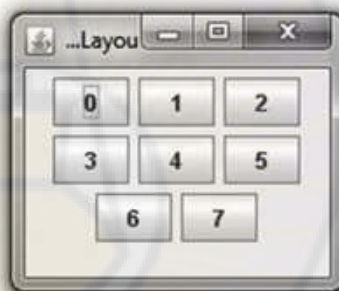
```



وإذا قمنا بتغيير حجم النافذة ، هذا يعني اضافة السطر التالي إلى المثال (9-6) `wind.setSize(180,150);` بدلاً من `wind.pack()` . سنلاحظ أن مدير التنسيق سيقوم بإعادة ترتيب مواضع الأزرار تلقائياً .



`wind.setSize(100,170);`



`wind.setSize(180,150);`



`wind.setSize(200,105);`

إذا قمنا بتكبير النافذة إلى أكبر حد ممكن ، سنلاحظ أن الأزرار لا تزال تحتفظ بالحجم الطبيعي لها كما هو موضح في الشكل التالي . أي إن مدير التنسيق `FlowLayout` يحافظ على الحجم الطبيعي للعناصر .



```
window.setSize(600,300);
```

ويمكننا تحديد محاذاة العناصر من خلال الوسيط الأول للبانى (FlwoLayout) والذي يأخذ أحد القيم التالية كما ذكرنا سابقاً .

- FlwoLayout.LEFT
- FlwoLayout.CENTER
- FlwoLayout.RIGHT

كما يمكننا من خلال الوسيط الثاني والثالث تحديد المسافة الأفقية والشاقولية على الترتيب بين العناصر ، علماً أن هذه المسافة تقدر بالبكسل Pixels كوحدة قياس فمثلاً إذا استبدلنا السطر
`wind.setLayout(new FlowLayout( ) );`
 بالسطر التالي :

```
wind.setLayout(new FlowLayout(FlowLayout.RIGHT,20,20));
```

هذا يعني أن العناصر سيتم محاذاتها إلى اليمين بالنسبة للحاوية مع مسافة أفقية و شاقولية بين العناصر مقدارها 20 pixels . ويوضح المثال (7-9) كتابة برنامج يضيف ثمانية أزرار تحمل الأسماء من 1 إلى 8 إلى لوحة محتوى الإطار وفق مدير التنسيق FlowLayout.

```
import javax.swing.*;
import java.awt.*;

public class MyLayoutManager2 {
    private static JButton btns [ ] = new JButton[8];
    public static void main(String args[ ]) {
        JFrame wind = new JFrame("FlowLayout ");
        // Here we will add Layout Manager
        wind.setLayout(new FlowLayout(FlowLayout.RIGHT,20,20));
        for(int i =0; i<8; i++) {
            btns[i] = new JButton(Integer.toString(i));
            wind.add(btns[i]);
        }
        wind.pack( );
        wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } // End main
} // End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٩-٤-٣-٢ - مدير التنسيق GridLayout

ينظم مدير التنسيق GridLayout العناصر وفق تنسيق شبكي (مصفوفة)، مع إمكانية تحديد عدد الأسطر والأعمدة في الباني . ويجعل مدير التنسيق GridLayout جميع العناصر بالحجم نفسه ، ويقوم بعرضها وفق عدد محدد

من الأسطر والأعمدة . ويتم وضع العناصر في الشبكة من اليسار إلى اليمين بحيث تبدأ من السطر الأول ثم تنتقل إلى الثاني ثم الثالث . . . إلخ ، ويتم وضع العناصر وفق ترتيب إضافتها نفسه . ويمتلك الصف GridLayout ثلاثة بواب :

١ . public GridLayout(int rows, int columns, int hGap, int vGap)
يقوم بإنشاء غرض جديد من الصف GridLayout مع إمكانية تحديد عدد الأسطر والأعمدة ، وإمكانية تحديد عدد الفجوات الأفقية والعمودية بين العناصر المضافة إلى الحاوية (النافذة) .

٢ . public GridLayout(int rows, int columns)
من الصف GridLayout مع إمكانية تحديد عدد الأسطر والأعمدة ، وتأخذ الفجوات الأفقية العمودية في هذا الباني القيمة صفر (0) .

٣ . public GridLayout(int rows)
GridLayout وينشئ عموداً واحداً ضمن صف وحيد .

من الممكن أن يكون عدد الأسطر أو عدد الأعمدة يساوي الصفر ، ولكن لا يمكن أن يكون عدد الأسطر وعدد الأعمدة يساوي الصفر في الوقت نفسه . إذا تم ضبط عدد الأسطر أو عدد الأعمدة على القيمة 0 ، عندها يتم تثبيت البعد الذي لا يساوي الصفر ، بينما يتحدد البعد الذي يساوي الصفر بشكل تلقائي من قبل مدير التنسيق GridLayout .

مثال : إذا كان لدينا مصفوفة تحتوي على 8 عناصر ، وحددنا عدد الأسطر بالقيمة 0 وحددنا عدد الأعمدة بالقيمة 3 { GridLayout(0,3) } عندها يقوم مدير التنسيق GridLayout بإنشاء ثلاثة أعمدة وثلاثة أسطر وذلك حسب الحاجة بحيث يحتوي السطر الأخير على عنصرين . إذا كان عدد الأسطر وعدد الأعمدة لا يساوي الصفر عندها يكون عدد الأسطر هو المحدد (البارامتر) المسيطر ، أي أنه يتم تثبيت عدد الأسطر بينما يقوم مدير التنسيق وبشكل تلقائي

بحساب عدد الأعمدة .

مثال : لنفرض لدينا مصفوفة تحتوي على 8 عناصر ، وحددنا أربعة أسطر وثلاثة أعمدة ، عندها يقوم مدير التنسيق GridLayout بإنشاء أربعة أسطر وعمودين فقط . أي بمعنى آخر ، إن مدير التنسيق GridLayout لن يحافظ على الحجم الطبيعي للعناصر ، إذا لم يتم تحديد عدد الأسطر والأعمدة في الباني ، فإنه سيتم عرض جميع العناصر على نفس السطر . أما إذا تم تغيير أبعاد النافذة فإن الأزرار ستتغير أبعادها لتلائم التغيير الحاصل، ويوضح المثال (8-9) التالي كتابة برنامج يضيف ثمان أزرار تحمل الأسماء من 1 إلى 8 إلى لوحة محتوى الإطار وفق مدير التنسيق GridLayout .

مثال (8-9)

```
import javax.swing.*;
import java.awt.*;
public class MyLayoutManager3 {
    private static JButton btns [ ] = new JButton[8];
    public static void main(String args[ ]) {
        JFrame wind = new JFrame("GridLayout ");
        // Here we will add Layout Manager
        wind.setLayout(new GridLayout(3,3,5,5));
        for(int i =0; i<8; i++) {
            btns[i] = new JButton(Integer.toString(i));
            wind.add(btns[i]);
        }
        wind.pack( );
        wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } // End main
} // End class
```

فإنه سيتم عرض الأزرار ضمن مصفوفة مكونة من ثلاثة أسطر وثلاثة أعمدة بينهما فجوات أفقية وعمودية على النحو التالي .



إذا استخدمنا السطر التالي في المثال السابق

```
wind.setLayout(new GridLayout(3,3));
```

فإنه سيتم عرض الأزرار ضمن مصفوفة مكونة من ثلاثة أسطر وثلاثة أعمدة بدون فجوات أفقية وعمودية على النحو التالي .



إذا استخدمنا السطر التالي في المثال السابق

```
wind.setLayout(new GridLayout(4,3));
```

فإنه سيتم عرض الأزرار ضمن مصفوفة مكونة من أربعة أسطر وعمودين بدون فجوات أفقية وعمودية على النحو التالي :



إذا استخدمنا السطر التالي في المثال السابق

```
wind.setLayout(new GridLayout(3,3));
```

وتم تغيير حجم النافذة كما يلي `wind.setSize(300,300);` وذلك بدلاً من السطر `wind.pack( )`.

سيتم عرض الأزرار ضمن مصفوفة مكونة من ثلاثة أسطر وثلاثة أعمدة بدون فجوات أفقية وعمودية وتتغير أبعاد الأزرار لتلائم التغيير الجديد على النحو التالي:



٩-٤-٣-٣- مدير التنسيق BorderLayout

هو المدير الافتراضي لكل من الحاويات JFrame و JApplet و JDialog

ويقسم مدير التنسيق BorderLayout النافذة إلى خمس مناطق هي :

- الشرق East
- الجنوب South
- الغرب West
- الشمال North
- والمركز (الوسط) Center

وتتم إضافة العناصر إلى مدير التنسيق BorderLayout باستخدام التابع
add(component , index) ، حيث يمثل المحدد index أحد القيم التالية :

- BorderLayout.NORTH
- BorderLayout.SOUTH
- = BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.CENTER

يمكن استخدام أحد البائنين التاليين من أجل إنشاء مدير التنسيق
: BorderLayout

1- public BorderLayout(int hGap , int vGap)

ويقوم بإنشاء غرض من الصف BorderLayout ، مع إمكانية تحديد الفجوة
الأفقية والعمودية بين العناصر .

2 - public BorderLayout()

ويقوم بإنشاء غرض من الصف BorderLayout ، بدون تحديد الفجوة الأفقية
والعمودية بين العناصر .

عند تكبير النافذة فإن المنطقة المركزية ستحتل أكبر قسم ممكن ، أما المناطق الأخرى فستمتد حسب الحاجة . وتستطيع العناصر الموجودة في الشمال North والجنوب South التمدد أفقياً بينما تستطيع العناصر الموجودة في الشرق East والغرب West من التمدد عمودياً . وأخيراً يستطيع العنصر الموجود في المركز Center التمدد أفقياً وعمودياً من أجل ملء جميع العناصر الفارغة .

ذكرنا بأن مدير التنسيق الافتراضي لـ JFrame هو BorderLayout ، لذلك من غير الضروري أن نحدده بشكل صريح ، وعند إضافة أكثر من عنصر على المنطقة نفسها ، فإن آخر عنصر تم إضافته إلى تلك المنطقة هو الذي سيظهر . ويوضح المثال (9-9) التالي كتابة برنامج يضيف خمسة أزرار تحمل الأسماء South و North و East و West و Center إلى نافذة محتوية الاطار وفق مدير التنسيق BorderLayout .

مثال (9-9)

```
import javax.swing.*;
import java.awt.*;

public class MyLayoutManager4 {
    public static void main(String args[ ] ) {
        JFrame wind = new JFrame("BorderLayout");
        // Here we will add Layout Manager
        wind.setLayout(new BorderLayout( ) );

        JButton north = new JButton("North");
        JButton south = new JButton("South");
        JButton west = new JButton("West");
        JButton east = new JButton("East");
        JButton center = new JButton("Center");
        wind.add(north, BorderLayout.NORTH)    //1
        wind.add(south, BorderLayout.SOUTH);    //2
```

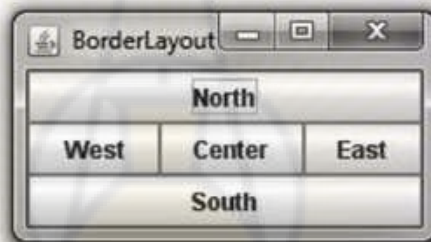
```

wind.add(west,BorderLayout.WEST);           //3
wind.add(east,BorderLayout.EAST);           //4
wind.add(center,BorderLayout.CENTER);       //5

wind.pack( );
wind.setVisible(true);
wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
} //End main
} //End class

```

تمت إضافة الأزرار إلى الاطار في الأسطر 1,2,3,4,5 .
وعند تنفيذ البرنامج سنحصل على الخرج التالي:



إذا استخدمنا السطر التالي في المثال السابق

```
wind.setLayout(new BorderLayout( 10,10) );
```

سنحصل على الخرج التالي :



نلاحظ إن التابع add التابع للصف BorderLayout يختلف عن التابع add التابع للصفين FlowLayout و GridLayout حيث يمكننا تحديد مكان إضافة العناصر في الغرض BorderLayout .

ليس من الضروري وضع العناصر بحيث تشغل جميع المساحات الفارغة ، إذا تم حذف الزر East من البرنامج ثم أعدت تشغيله عندها سنجد أن الزر Center قد تمدد على الناحية اليمينية لكي يشغل المساحة East ، كما هو موضح في الشكل التالي :



إذا استخدمنا السطر التالي في المثال السابق

```
wind.setSize(300,200);
```

(تغيير حجم النافذة) وذلك بدلاً من السطر

```
wind.pack();
```

سنحصل على الخرج التالي :



٩-٤-٣-٤ - مدير التنسيق CardLayout

يسمح مدير التنسيق CardLayout بوضع عدد من المكونات فوق بعضها بعضاً على شكل طبقات ، مع إمكانية التنقل بين هذه الطبقات . وتظهر طبقة واحدة فقط في لحظة معينة ، وللانتقال إلى الطبقات الأخرى فإننا نستخدم الطرائق التالية :

- first من أجل الانتقال إلى الطبقة الأولى .
 - next من أجل الانتقال من الطبقة الحالية إلى الطبقة التالية .
 - previous من أجل الانتقال من الطبقة الحالية إلى الطبقة السابقة.
 - last من أجل الانتقال إلى الطبقة الأخيرة .
- المثال (9-10) التالي يوضح مدير التنسيق CardLayout ، حيث قمنا بتعريف غرضين من الصف JButton ووضعنا منهما في حاوية من الصف JPanel ، ثم أضفنا كلتا الحاويتين إلى غرض من الصف JFrame .

مثال (9-10)

1. **import** javax.swing.*;
2. **import** java.awt.*;
3. **public class** MyLayoutManager5 {
4. **public static void** main(**String** args[]) {
5. JFrame wind = **new** JFrame("CardLayout");
6. CardLayout cardManager = **new** CardLayout();
7. Container cont = **wind**.getContentPane();
8. cont.setLayout(cardManager);
9. JButton first = **new** JButton("First");
10. JButton second = **new** JButton("Second");
11. JPanel panel1 = **new** JPanel();
12. JPanel panel2 = **new** JPanel();
13. panel1.add(first);
14. panel2.add(second);


```

15. cont.add(panel1,"First Panel");
16. cont.add(panel2,"Second Panel");
17. cardManager.first(cont);
18. wind.pack( );
19. wind.setVisible(true);
20. wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21. }//End main
22. }//End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي :



نلاحظ :

- ✓ في السطر 8 تم ضبط مدير التنسيق لغرض النافذة cont باستخدام الغرض cardManager والمعرف في السطر 6 على أنه مدير تنسيق من الصف CardLayout .
- ✓ في السطرين 11 و 12 تم تعريف حاويتين panel1 و panel2 من الصف JPanel .
- ✓ في السطرين 13 و 14 أضفنا الزرين first و second إلى الحاويتين panel1 و panel2 على الترتيب .
- ✓ في السطرين 15 و 16 قمنا بإضافة الحاويتين panel1 و panel2 إلى الحاوية الرئيسة cont ، وبهذا الشكل أصبحت الحاوية panel1 تشكل الطبقة الأولى والحاوية panel2 تشكل الطبقة الثانية .
- ✓ أخيراً استخدمنا التابع first من أجل تحديد الطبقة التي تم عرضها .

٩-٤-٣-٥ - مدير التنسيق GridBagLayout

يُعدّ مدير التنسيق GridBagLayout الأكثر تعقيداً ومرونة في الوقت نفسه فهو يقوم بترتيب المكونات ضمن شبكة مؤلفة من مجموعة من الأسطر والأعمدة ، أي أنه يشبه مدير التنسيق GridLayout إلا أنه يتميز بقدرته على جعل عنصر واحد يمتد على أكثر من سطر أو عمود ، كما أنه ليس بالضرورة أن يكون لجميع الأسطر الارتفاع نفسه ، وكذلك عرض الأعمدة يمكن أن يختلف من عمود إلى آخر . ويوضح المثال (9 - 11) التالي مدير التنسيق GridBagLayout .

مثال (9-11)

```
1. import javax.swing.*;
2. import java.awt.*;
3. public class MyLayoutManager6 {
4.     public static void main(String args[ ] ) {
5.         JFrame wind = new JFrame("GridBagLayout");
6.         GridBagLayout grid = new GridBagLayout( );
7.         GridBagConstraints gridConst = new GridBagConstraints( );
8.         wind.setLayout(grid);
9.         JButton btns[ ] = new JButton[5];
10.        btns[0] = new JButton("Welcome");
11.        btns[1] = new JButton("Hi");
12.        btns[2] = new JButton("Bonjour");
13.        btns[3] = new JButton("السلام عليكم ورحمة الله");
14.        btns[4] = new JButton("Guten Tag");
15.        gridConst.gridx = 0; gridConst.gridy = 0;
16.        grid.setConstraints(btns[0],gridConst);
17.        wind.add(btns[0]);
18.        gridConst.gridx = 1; gridConst.gridy = 0;
19.        grid.setConstraints(btns[1],gridConst);
```

```

20. wind.add(btns[1]);
21. gridConst.gridx = 2; gridConst.gridy = 0;
22. grid.setConstraints(btns[2],gridConst);
23. wind.add(btns[2]);
24. gridConst.gridx = 0; gridConst.gridy = 1;
25. gridConst.gridwidth = 3;
26. grid.setConstraints(btns[3],gridConst);
27. wind.add(btns[3]);
28. gridConst.gridx = 0; gridConst.gridy = 2;
29. grid.setConstraints(btns[4],gridConst);
30. wind.add(btns[4]);
31. wind.pack( );
32. wind.setVisible(true);
33. wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34. //End main
35. //End class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



`gridConst.gridwidth = 3;`



`gridConst.gridwidth = 2;`



`gridConst.gridwidth = 1;`

نلاحظ :

✓ في السطر 6 قمنا بإنشاء غرض من الصف `GridBagLayout` ، ولضبط خصائص هذا الغرض نحتاج إلى تعريف غرض من الصف `GridBagConstraints` والذي قمنا بتعريفه في السطر 7 ، هذا الغرض يحدد لنا كيفية توضع العناصر ضمن الغرض `grid` ، أي تحديد الخلية (السطر والعمود) التي سيتوضع فيها عنصر ما ، وذلك من خلال الخاصيتين : `gridx` التي تحدد العمود و `gridy` التي تحدد السطر ، مع العلم بأن الترقيم يبدأ من الصفر ، كما يسمح لنا بتحديد ما إذا كان العنصر سيتوضع على أكثر من خلية من خلال الخاصيتين `gridwidth` و `gridheight` .

✓ لقد قمنا في هذا المثال بتعريف عدد من الأزرار ، وفي السطر 15 ضبطنا الغرض `gridConst` على الخلية ذات العمود 0 والسطر 0 ، وبعدها أسندنا هذا الغرض إلى الزر `btns[0]` .

✓ وكررنا هذه العملية من أجل بقية الأزرار ، مع الملاحظة أننا في السطر 25 ضبطنا خاصية إضافية للغرض `gridConst` وهي `gridwidth` وذلك للتنبيه إلى أن العنصر سيعتمد على ثلاثة أعمدة .

✓ إذا أردنا أن يأخذ العنصر كامل مساحة الخلية الذي تم إضافته إليها ، فإنه يمكننا استخدام الخاصية fill التي تأخذ أحد الثوابت التالية :

- gridBagConstraints.VERTICAL يؤدي إلى تغيير البعد الشاقولي للعنصر .

- gridBagConstraints.HORIZONTAL يؤدي إلى تغيير البعد الأفقي للعنصر .

وهناك ثوابت أخرى تتحكم أيضا بتوضع العنصر في مركز الخلية ، أو أعلاها أو أسفلها ، . . . الخ .

يمكننا إضافة السطر التالي إلى المثال (9-11) السابق بعد السطر 24 .

`gridConst.fill = GridBagConstraints.BOTH`

هذا السطر يؤدي إلى إظهار النافذة كما هو مبين في الشكل التالي .



نلاحظ أن تأثير السطر الذي قمنا بإضافته يمتد على جميع العناصر التي سيتم إضافتها لاحقاً ، لقد أثر على الزر `btns[0]` ، أما إذا أردنا ألا يمتد التأثير إلى بقية العناصر علينا إعادة ضبط الخاصية fill من جديد .

٩-٤-٣-٦ - مدير التنسيق BorderLayout

يضع مدير التنسيق BorderLayout العناصر إما في سطر أو عمود واحد ، وهو يأخذ بعين الاعتبار العنصر ذا الحجم الأكبر ، أي أن ارتفاع السطر هو نفسه ارتفاع أكبر عنصر ، وعرض العمود هو نفسه عرض أكبر عنصر ، ويوضح المثال (9-12) التالي مدير التنسيق BorderLayout . لا يوجد شيء جديد في هذا المثال إلا استخدام مدير التنسيق BorderLayout في السطر 1 .

نلاحظ أننا مررنا للبانى وسيطين ، الأول يدل على الحاوية المراد تنسيقها ، والثاني يدل على الاتجاه المراد ترتيب العناصر وفقه ، فمثلاً لترتيب العناصر بشكل شاقولي نستخدم الثابت BorderLayout.Y_AXIS ، أما لترتيبها بشكل أفقي فنستخدم الثابت BorderLayout.X_AXIS .

مثال (9-12)

```
import javax.swing.*;
import java.awt.*;

public class MyLayoutManager6{
    public static void main(String args[ ]){
        JFrame wind = new JFrame(" BorderLayout ");
        Container cont = wind.getContentPane();
        cont.setLayout(new BorderLayout(cont, BorderLayout.Y_AXIS)); //1

        JButton btns[ ] = new JButton[5];
        btns[0] = new JButton("Welcome");
        btns[1] = new JButton("Hi");
        btns[2] = new JButton("Bonjour");
        btns[3] = new JButton("السلام عليكم ورحمة الله");
        btns[4] = new JButton("Guten Tag");
        for(int i=0; i<5; i++)
```



```

cont.add(btns[i]);

wind.pack( );
wind.show( );
wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
} //End main
} //End class

```

عند تنفيذ هذا البرنامج ستظهر نافذة الخروج على الشكل التالي :



عند استخدام الثابت BoxLayout.X_AXIS ستظهر النافذة على الشكل التالي:



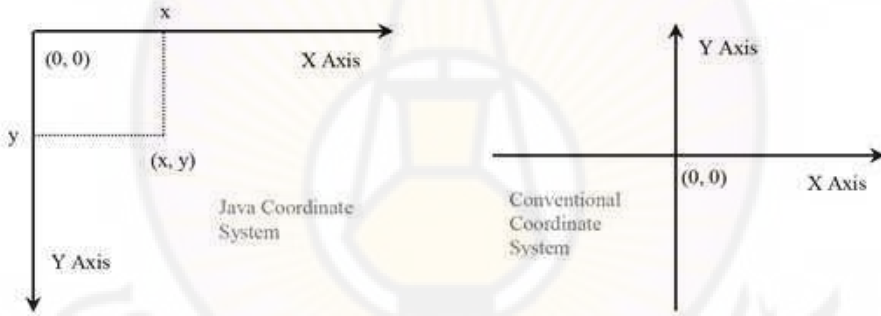
٩-٥- الرسم البياني Graphics

تزدونا جافا من خلال الصف Graphics بعدد من التوابع التي تسمح لنا برسم صورة معينة أو شكل هندسي ما (خط ، مستطيل ، قطع ناقص ، متعدد الأضلاع ، . . الخ) على مكون ما ، ويوجد نوعان من التوابع :

١. الأول يسمح لنا برسم حدود الشكل الهندسي فقط (شكل غير مصمت) .

٢. الثاني يسمح لنا برسم الشكل الهندسي مصمتاً .

ومن الجدير بالذكر أن مبدأ الإحداثيات هو الزاوية اليسارية العليا لمنطقة الرسم ، مع العلم بأن اتجاه محوري الرسم موضحان في الشكل (٩-٥) التالي :



الشكل (٩-٥) - يوضح محوري الرسم

عند استخدام توابع الرسم يوجد عدد من الوسطاء التي يجب استخدامها ، ومن هذه الوسطاء نقطة بدء الرسم مثلاً والتي تمثل الزاوية اليسارية العليا من العنصر وتزايد قيمة محور السينات x من اليسار إلى اليمين وقيمة محور العيّنات y من الأعلى إلى الأسفل . كما علينا أن نحدد لون خط الرسم قبل استدعاء التابع الذي سيقوم برسم الشكل ، حيث يتم تحديد لون خط الرسم من خلال التابع `setColor`. أيضاً يمكننا أن نحدد شكل خط الرسم هل هو مستمر أم متقطع ، وتملك جميع توابع الرسم محددات (بارامترات) تحدد أماكن رسم الأغراض .

٩-٥-١ - إعادة رسم المكونات

توجد آلية تلقائية لرسم مكونات البيئة المرئية ، حيث يتم إعادة رسم أي مكون swing عند الضرورة ، فمثلاً يتم إعادة رسم المكون عند تغيير حجم النافذة ، أو تصغيرها ، أو حتى ضبط خاصية ما لهذا المكون . فإن إعادة رسم المكونات تتم بصورة هرمية ، أي أنه يتم إعادة رسم العنصر الأساسي، ومن ثم العناصر المشتقة تدريجياً ، لذلك فإن أي رسم ننجزه على مكون ما سوف يتم مسحه عند تكبير أو تصغير النافذة ، أو حتى ظهور نافذة أخرى فوق نافذة الشكل الذي تتم عليه عملية الرسم . وتتم عملية الرسم من خلال استدعاء التابع repaint والذي يمكننا من إعادة تحقيقه من أجل رسم الشكل الذي تم مسحه .

إن الصف Graphics هو صف مجرد ويُستخدم من أجل عرض الأشكال والصور على الشاشة ضمن منصات مختلفة . وعندما نستخدم التابع paintComponent من أجل رسم الأشياء على المحتوى الرسومي g ، فإن g هو عبارة عن غرض من الصف المشتق للصف المجرد Graphics . كلما يتم عرض العنصر يتم إنشاء غرض من الصف Graphics من أجل هذا العنصر . تستخدم عناصر الحزمة Swing التابع paintComponent من أجل رسم الأشياء، ويتم استدعاء هذا التابع بشكل تلقائي من أجل رسم المحتوى الرسومي عند عرض العنصر الأول مرة أو كلما يحتاج العنصر إلى إعادة العرض ومن الضروري استدعاء التابع (g) super.paintComponent وذلك للتأكد من تنظيف مساحة الرؤية (من الرسوم القديمة) قبل عرض الرسوم الجديدة .

يستطيع المستخدم طلب إعادة رسم العنصر بواسطة التابع (repaint) المعروف في الصف Component ، حيث يؤدي استدعاء التابع (repaint) إلى استدعاء التابع paintComponent من قبل آلة جافة الافتراضية .

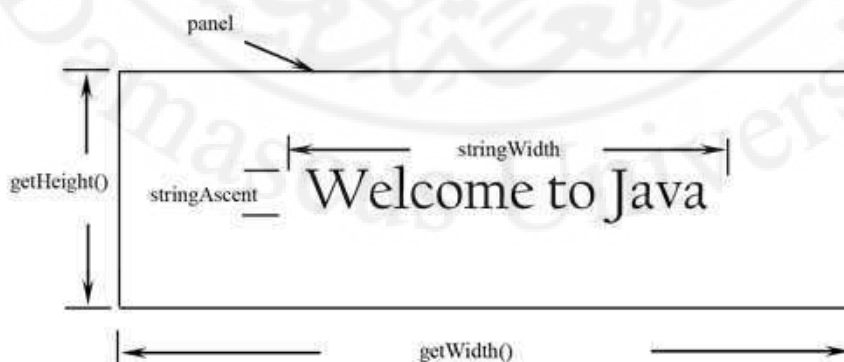
٩-٥-٢- رسم الخطوط والرسومات الأخرى ضمن اللوحات

إن اللوحات (الإطار) هي عناصر غير مرئية وتستخدم كحاويات صغيرة من أجل تجميع العناصر بهدف الحصول على شكل التنسيق المطلوب وكما تستخدم اللوحات (JPanel) أيضاً من أجل الرسم ، ولكي نستطيع الرسم على غرض من الصف JPanel يجب إنشاء صف جديد يرث الصف JPanel ويتجاوز التابع paintComponent من أجل إخبار اللوحة بكيفية رسم الأشياء . إن التابع paintComponent معرف في الصف JComponent وله التعريف التالي :

```
protected void paintComponent( Graphics g )
```

يتم إنشاء غرض g الذي ينتمي إلى الصف Graphics بواسطة آلة جافا الافتراضية وبشكل تلقائي من أجل كل عنصر GUI مرئي ، يتحكم هذا الغرض بكيفية رسم المعلومات ، ويمكن استخدام توابع الرسم المختلفة والمعرفة في الصف Graphics من أجل رسم النصوص والأشكال الهندسية . ونستطيع رسم النصوص (على سبيل المثال العبارة Welcome to Java) باستخدام التابع التالي :

```
drawString ( String ,x , y ) ;
```



يقوم البرنامج التالي برسم الرسالة التالية :

" Welcome To Java Programming ! "

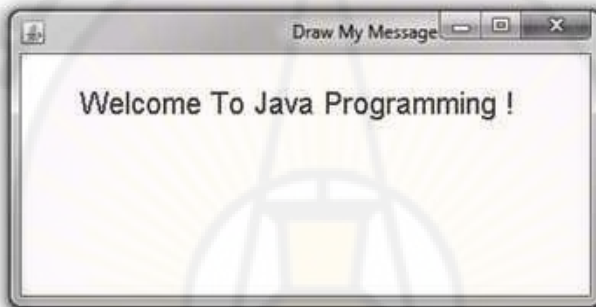
مثال (9-13)

```
1. // Display a message on a JPanel
2. import javax.swing.*;
3. import java.awt.*;
4. public class DrawMessage extends JPanel {
5.     /** Main method */
6.     public static void main(String[ ] args) {
7.         JFrame frame = new JFrame("Draw My Message");
8.         DrawMessage m = new DrawMessage();
9.         m.setBackground(Color.WHITE);
10. // m.setForeground( Color.BLUE );
11. // m.setBackground(Color.YELLOW);
12. // m.setFont(new Font( "Serif",Font.BOLD+ Font.ITALIC, 20));
13. m.setFont(new Font("TimeRoman", Font.PLAIN, 20));
14. frame.getContentPane().add(m);
15. frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16. frame.setSize(400, 200);
17. frame.setVisible(true);
18. } //End main
19. /** Paint the message */
20. protected void paintComponent(Graphics g) {
21.     super.paintComponent(g);
22.     g.drawString("Welcome To Java Programming !", 40, 40);
23. }
24. } // End class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



إذا أضفنا السطر 10 `m.setForeground( Color.RED ) ;` نحصل على
الخرج التالي :



إذا أضفنا السطر 11 `m.setBackground(Color.YELLOW);` أيضاً نحصل
على الخرج التالي :



إذا أضفنا السطر 12 بدلاً من السطر 13
`m.setFont(new Font( "Serif",Font.BOLD+ Font.ITALIC, 20));`

نحصل على الخرج التالي :



٩-٥-٢-١ - الصف Color

يمكن تحديد الألوان من أجل عناصر GUI باستخدام الحزمة `java.awt.Color`. يتم إنشاء الألوان من الألوان الثلاثة الرئيسية التالية : `RED` , `GREEN` , `BLUE` ، حيث يتم تمثيل كل منهم باستخدام `Byte` يصف شدته (والتي تتراوح من القيمة 0 " خيال غامق " إلى القيمة 255 " خيال كاشف ") ، ويُعرف هذا النموذج اللوني بنموذج `RGB` ، والصيغة المستخدمة لإنشاء غرض من الصف `Color` هي :

```
Color color = new Color ( r , g , b ) ;
```

حيث تحدد البارامترات `r` , `g` , `b` اللون بواسطة عناصره الثلاث (الأحمر ، الأخضر ، الأزرق) ، على سبيل المثال :

```
Color color = new Color ( 128 , 100 , 100 ) ;
```

نستطيع استخدام التابعين : `setBackground( Color c )` و `setForeground( Color c )` المعرفين في الصف `Component` من أجل تحديد لون الخلفية واللون الأمامي للعنصر.

مثلاً نقوم الشيفرة التالية بتحديد لون الخلفية للوحة `mypanel` باستخدام اللون `color` :

```
JPanel mypanel = new JPanel( ) ;
```

mypanel . setBackground(color c) ;

يمكن تحديد اللون بطريقة ثانية وذلك باستخدام واحد من الألوان الثلاثة عشر القياسية التالية :

(white , red , pink , orange , megenta , lightGray , green , gray , darkgray , cyan , blue , black, yellow) .

ويبين الجدول (٩-١) ثوابت الألوان وقيم RGB .

Color constant	Color	RGB value
public final static Color RED	red	255, 0, 0
public final static Color GREEN	green	0, 255, 0
public final static Color BLUE	blue	0, 0, 255
public final static Color ORANGE	orange	255, 200, 0
public final static Color PINK	pink	255, 175, 175
public final static Color CYAN	cyan	0, 255, 255
public final static Color MAGENTA	magenta	255, 0, 255
public final static Color YELLOW	yellow	255, 255, 0
public final static Color BLACK	black	0, 0, 0
public final static Color WHITE	white	255, 255, 255
public final static Color GRAY	gray	128, 128, 128
public final static Color LIGHT_GRAY	light gray	192, 192, 192
public final static Color DARK_GRAY	dark gray	64, 64, 64

الجدول (٩-١) - يبين ثوابت الألوان وقيم RGB

ويبين الجدول (٩-٢) التتابع وبواني الألوان وتتابع الرسم البياني ووترج الألوان .

Method	Description
<i>Color constructors and methods</i>	
<code>public Color(int r, int g, int b)</code>	Creates a color based on red, green and blue components expressed as integers from 0 to 255.
<code>public Color(float r, float g, float b)</code>	Creates a color based on red, green and blue components expressed as floating-point values from 0.0 to 1.0.
<code>public int getred()</code>	Returns a value between 0 and 255 representing the red content.
<code>public int getGreen()</code>	Returns a value between 0 and 255 representing the green content.
<code>public int getBlue()</code>	Returns a value between 0 and 255 representing the blue content.
<i>Graphics methods for manipulating Colors</i>	
<code>public Color getColor()</code>	Returns Color object representing current color for the graphics context.
<code>public void setColor(Color c)</code>	Sets the current color for drawing with the graphics context.

الجدول (٩-٢) - يبين التتابع وبواني الألوان وتتابع الرسم البياني

ويبين المثال (9-14) تغيير ألوان الرسم البياني .

مثال (9-13)

```
//Demonstrating Colors.
import java.awt.*;
import javax.swing.*;
public class ColorJPanel extends JPanel {
    // draw rectangles and Strings in different colors
    public void paintComponent( Graphics g ){
        super.paintComponent( g ); // call superclass's paintComponent
        this.setBackground( Color.WHITE );
        // set new drawing color using integers
        g.setColor( new Color( 255, 0, 0 ) );
        g.fillRect( 15, 25, 100, 20 );
        g.drawString( "Current RGB: " + g.getColor(), 130, 40 );
        // set new drawing color using floats
        g.setColor( new Color( 0.50f, 0.75f, 0.0f ) );
        g.fillRect( 15, 50, 100, 20 );
        g.drawString( "Current RGB: " + g.getColor(), 130, 65 );
        // set new drawing color using static Color objects
        g.setColor( Color.BLUE );
        g.fillRect( 15, 75, 100, 20 );
        g.drawString( "Current RGB: " + g.getColor(), 130, 90 );
        // display individual RGB values
        Color color = Color.MAGENTA;
        g.setColor( color );
        g.fillRect( 15, 100, 100, 20 );
        g.drawString( "RGB values: " + color.getRed() + ", " +
            color.getGreen() + ", " + color.getBlue(), 130, 115 );
    } // end method paintComponent
} // end class ColorJPanel
//-----
```

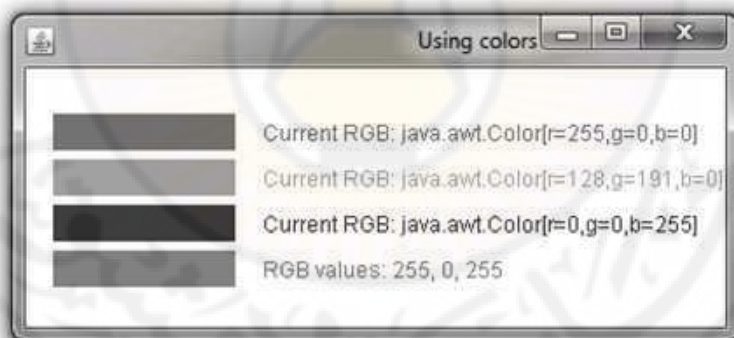


```

public class ShowColors {
    // execute application
    public static void main( String args[] ) {
        // create frame for ColorJPanel
        JFrame frame = new JFrame( "Using colors" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        // create ColorJPanel
        ColorJPanel ob= new ColorJPanel( );
        frame.add( ob ); // add colorJPanel to frame
        frame.setSize( 400, 180 ); // set frame size
        frame.setVisible( true ); // displayt frame
    } // end main
} // end class ShowColors

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



توفر حزمة javax.swing المكون JColorChooser لواجهة المستخدم الرسومية التي تمكن المستخدمين لتحديد الألوان. كما هو موضح في المثال (9-14) الذي يعرض صندوق الحوار JColorChooser ، وعند النقر على زر تغيير اللون، يظهر صندوق حوار JColorChooser ، وعند تحديد اللون والضغط على زر موافق في صندوق الحوار، تتغير لون خلفية نافذة التطبيق

// Choosing colors with JColorChooser.

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
```

```
public class ShowColors2JFrame extends JFrame {
    private JButton changeColorJButton;
    private Color color = Color.LIGHT_GRAY;
    private JPanel colorJPanel;
```

// set up GUI

```
    public ShowColors2JFrame( ) {
        super( "Using JColorChooser" );
```

// create JPanel for display color

```
        colorJPanel = new JPanel();
        colorJPanel.setBackground( color );
```

// set up changeColorJButton and register its event handler

```
        changeColorJButton = new JButton( "Change Color" );
        changeColorJButton.addActionListener(
            new ActionListener() // anonymous inner class
            {
```

// display JColorChooser when user clicks button

```
                public void actionPerformed( ActionEvent event ) {
                    color = JColorChooser.showDialog(
                        ShowColors2JFrame.this, "Choose a color", color );
```

// set default color, if no color is returned

```
                    if ( color == null )
                        color = Color.LIGHT_GRAY;
```

// change content pane's background color

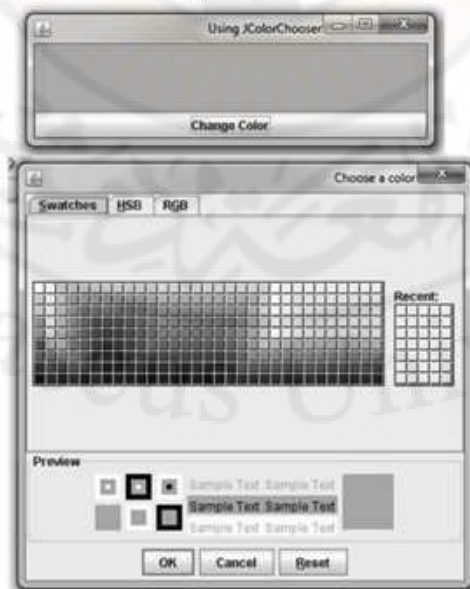
```
                    colorJPanel.setBackground( color );
```

```

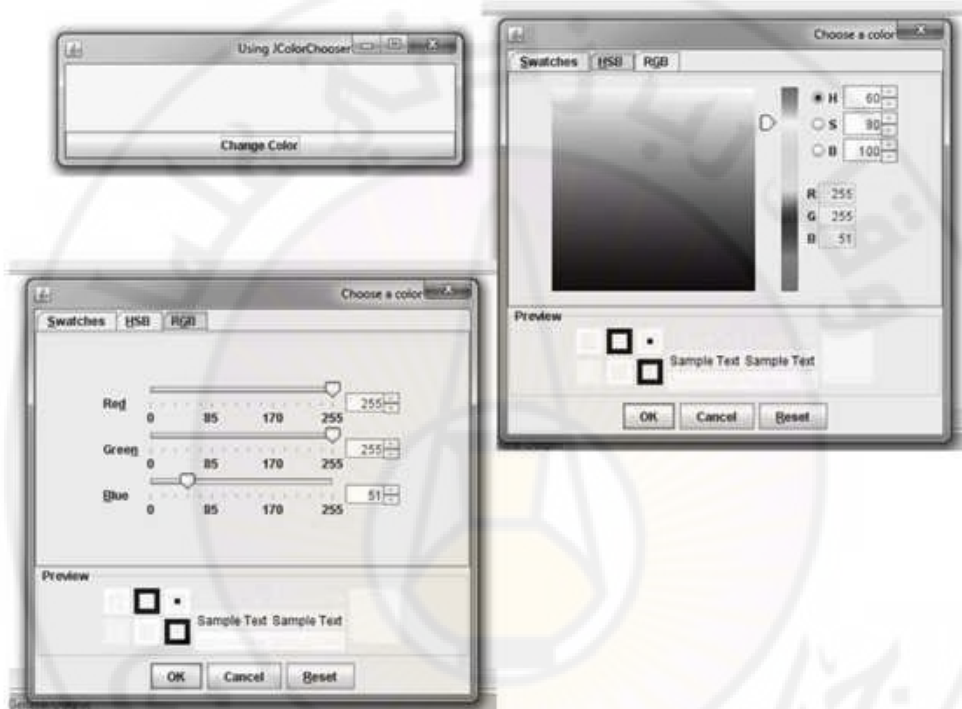
        } // end method actionPerformed
    } // end anonymous inner class
); // end call to addActionListener
add( colorJPanel, BorderLayout.CENTER ); // add colorJPanel
add( changeColorJButton, BorderLayout.SOUTH ); // add button
setSize( 400, 130 ); // set frame size
setVisible( true ); // display frame
} // end ShowColor2JFrame constructor
} // end class ShowColors2JFrame
// Choosing colors with JColorChooser.
public class ShowColors2 {
// execute application
public static void main( String args[ ] ){
    ShowColors2JFrame ob = new ShowColors2JFrame( );
    ob.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
} // end main
} // end class ShowColors2

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



الزر RGB يسمح لنا بتحديد اللون باستخدام الممزج لتحديد مكونات اللون الأحمر والأخضر والأزرق. وتظهر علامات التبويب HSB و RGB في الشكل.



٩-٥-٢-٢- FontMatrics والصف Font

يمكن تحديد الخطوط من أجل العناصر أو الأغراض التي نرسمها . كما يمكن استخدام قياسات الخط من أجل حساب حجم الخط . حيث تم تغليف الخط وقياساته ضمن الصفيين Font و FontMatrics . ونحتاج من أجل تحديد الخط إلى إنشاء غرض من الصف Font كما يلي :

Font myFont = new Font (name , style , size) ;

يمكن اختيار الخط من الأسماء التالية :

DialogInput ,Dialog , Monospaced , Serif , SansSerif

واختيار نمط الخط من الأنماط التالية :

Font.ITALIC , Font.BOLD , Font.PLAIN

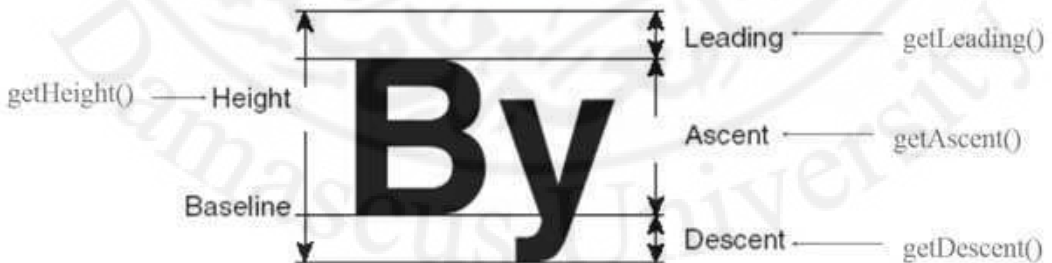
كما يمكن مزج أكثر من خط في الوقت نفسه كما يلي :

```
Font myFont = new Font ( "SansSerif",Font.BOLD, 16) ;
```

```
Font myFont = new Font ( "Serif",Font.BOLD+ Font.ITALIC, 14 ;
```

يمكن استخدام الصف FontMetrics من أجل حساب الطول والعرض الفعليين للنص ، وتعتبر هذه المعلومات مفيدة جداً في حساب حجم النص من أجل عرضه في الوضع الصحيح . ويمكن وضع النص في منتصف مساحة الرؤية بمساعدة الصف FontMetrics . ويمكن استخدام التوابع التالية :

```
public int getAscent( ) ;  
public int getDescent( ) ;  
public int getLeading( ) ;  
public int getHeight( ) ;  
public int stringWidth(String str) ;
```



ويوضح الجدول (٩-٢) بعض توابع التحكم بالخط ورسمه .

Method or constant	Description
<i>Font constants, constructors and methods</i>	
<code>public final static int PLAIN</code>	A constant representing a plain font style.
<code>public final static int BOLD</code>	A constant representing a bold font style.
<code>public final static int ITALIC</code>	A constant representing an italic font style.
<code>public Font(String name, int style, int size)</code>	Creates a Font object with the specified font name, style and size.
<code>public int getStyle()</code>	Returns an integer value indicating the current font style.
<code>public int getSize()</code>	Returns an integer value indicating the current font size.
<code>public String getName()</code>	Returns the current font name as a string.
<code>public String getFamily()</code>	Returns the font's family name as a string.
<code>public boolean isPlain()</code>	Returns <code>TRUE</code> if the font is plain, else <code>false</code> .
<i>Graphics methods for manipulating Fonts</i>	
<code>public Font getFont()</code>	Returns a Font object reference representing the current font.
<code>public void setFont(Font f)</code>	Sets the current font to the font, style and size specified by the Font object reference <code>f</code> .

الجدول (٩-٢) - يبين بعض توابع التحكم بالخط ورسمه .

أما المثال (9-15) يظهر النص بأربعة أنماط خطوط مختلفة وكل نمط بحجم مختلف ، أما باني الخط Font constructor يستخدم لإنشاء أغراض الخط Font objects وتمرير المعطيات إلى تابع الرسم setFont method لتغيير نمط الخط . كل استدعاء لباني الخط يمرر اسم الخط Serif, Monospaced or SansSerif كسلسلة ونمط الخط Font.PLAIN, Font.ITALIC or Font.BOLD وحجم الخط ويتم استدعاء التابع getFont من الصف Graphics يعيد غرض الذي يمثل الخط الحالي ، أما التابع getName يعيد اسم الخط الحالي كسلسلة ويعيد التابع getSize حجم الخط .

مثال (9-15)

```
// Display strings in different fonts and colors.
import java.awt.*;
import javax.swing.*;

public class FontJPanel extends JPanel {
// display Strings in different fonts and colors
public void paintComponent( Graphics g ) {
super.paintComponent( g ); // call superclass's paintComponent

// set font to Serif (Times), bold, 12pt and draw a string
g.setFont( new Font( "Serif", Font.BOLD, 12 ) );
g.drawString( "Serif 12 point bold.", 20, 50 );

// set font to Monospaced (Courier), italic, 24pt and draw a string
g.setFont( new Font( "Monospaced", Font.ITALIC, 24 ) );
g.drawString( "Monospaced 24 point italic.", 20, 70 );

// set font to SansSerif (Helvetica), plain, 14pt and draw a string
g.setFont( new Font( "SansSerif", Font.PLAIN, 14 ) );
g.drawString( "SansSerif 14 point plain.", 20, 90 );
```

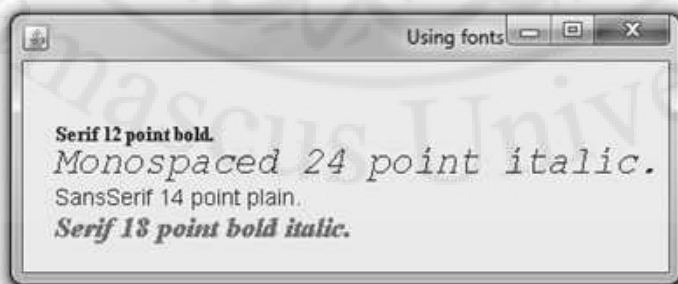
```

// set font to Serif (Times), bold/italic, 18pt and draw a string
g.setColor( Color.RED );
g.setFont( new Font( "Serif", Font.BOLD + Font.ITALIC, 18 ) );
g.drawString( g.getFont().getName() + " " + g.getFont().getSize()
              + " point bold italic.", 20, 110 );
    } // end method paintComponent
} // end class FontJPanel
// Using fonts.
public class Fonts {
// execute application

public static void main( String args[ ] ) {
// create frame for FontJPanel
    JFrame frame = new JFrame( "Using fonts" );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    FontJPanel ob = new FontJPanel( ); // create FontJPanel
    frame.add( ob ); // add fontJPanel to frame
    frame.setSize( 420, 170 ); // set frame size
    frame.setVisible( true ); // display frame
    } // end main
} // end class Fonts

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



الجدول (٣-٩) يوضح عدداً من التوابع الموجودة في الصف
FontMetrics للحصول على أحجام خط مختلفة، ويوضح المثال (9-16)
بعض هذه التوابع للحصول على أحجام لنوعين من الخطوط .

Method	Description
<i>FontMetrics methods</i>	
<code>public int getAscent()</code>	Returns the ascent of a font in points.
<code>public int getIDescent()</code>	Returns the descent of a font in points.
<code>public int getLeading()</code>	Returns the leading of a font in points.
<code>public int getHeight()</code>	Returns the height of a font in points.
<i>Graphics methods for getting a Font's FontMetrics</i>	
<code>public FontMetrics getFontMetrics()</code>	Returns the FontMetrics object for the current drawing Font.
<code>public FontMetrics getFontMetrics(Font f)</code>	Returns the FontMetrics object for the specified Font argument.

الجدول (٣-٩) - يوضح أحجام الخطوط


```

/* FontMetrics and Graphics methods useful for
   obtaining font metrics. */
import java.awt.Font;
import java.awt.Color;
import java.awt.FontMetrics;
import java.awt.Graphics;
import javax.swing.JPanel;
import javax.swing.JFrame;
public class MetricsJPanel extends JPanel {
// display font metrics
public void paintComponent( Graphics g ) {
    super.paintComponent( g ); // call superclass's paintComponent
    g.setFont( new Font( "SansSerif", Font.PLAIN, 12 ) );
    FontMetrics metrics = g.getFontMetrics( );
    g.drawString( "Current font: " + g.getFont( ), 10, 40 );
    g.drawString( "Ascent: " + metrics.getAscent( ), 10, 55 );
    g.drawString( "Descent: " + metrics.getDescent( ), 10, 70 );
    g.drawString( "Height: " + metrics.getHeight( ), 10, 85 );
    g.drawString( "Leading: " + metrics.getLeading( ), 10, 100 );

    g.setColor( Color.BLUE );
    Font font = new Font( "Serif", Font.BOLD+ Font.ITALIC, 14 );
    metrics = g.getFontMetrics( font );
    g.setFont( font );

    g.drawString( "Current font: " + font, 10, 130 );
    g.setColor( Color.RED );
    g.drawString( "Ascent: " + metrics.getAscent( ), 10, 145 );
    g.drawString( "Descent: " + metrics.getDescent( ), 10, 160 );

```



```

g.setColor( Color.DARK_GRAY );
g.drawString( "Height: " + metrics.getHeight( ), 10, 175 );
g.drawString( "Leading: " + metrics.getLeading( ), 10, 190 );
} // end method paintComponent
} // end class MetricsJPanel

// Displaying font metrics.
public class Metrics {
// execute application
    public static void main( String args[ ] ) {
// create frame for MetricsJPanel
        JFrame frame = new JFrame( "Demonstrating FontMetrics" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        MetricsJPanel ob = new MetricsJPanel( );
        frame.add( ob ); // add metricsJPanel to frame
        frame.setSize( 510, 250 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class Metrics

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



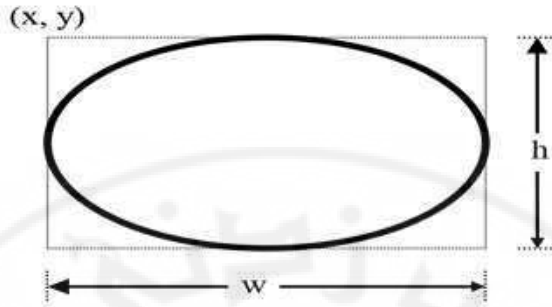
٩-٥-٣ - رسم الأشكال الهندسية

Drawing Geometric Shapes

يعرض الجدول (٩-٤) التالي عدداً من التوابع التي تُستخدم في رسم بعض الأشكال الهندسية . فهي محملة بشكل زائد وبالتالي فإن لكل منها عدة أشكال ، لذلك أخذنا أحد هذه الأشكال عند وصف الوسطاء الممررة إليها ، بالنسبة للتابع drawOval() فإن الوسطاء الممررة تمثل زوايا مستطيل وهمي يتم ضمنه رسم القطع الناقص ، بحيث تكون أضلاع هذا المستطيل عبارة عن مماسات للقطع الناقص ، كما هو مبين الشكل (٩-١) .

اسم التابع	وظيفته	الوسطاء الممررة
drawLine()	رسم خط	إحداثيات تغطي البداية والنهاية
drawaRect()	رسم مستطيل	إحداثيات الزاوية اليسارية العليا واليمنى السفلية .
drawRoundRect()	رسم مستطيل بزوايا مستديرة	إحداثيات الزاوية اليسارية العليا واليمنى السفلية .
drawArc()	رسم قوس	إحداثيات نقطة بدء الرسم (عرض وطول القوس ، وزاوية القوس)
drawOval()	رسم قطع ناقص	الزاوية اليسارية العليا للقطع ، وعرض وطول القطع .
drawString()	رسم لسلسلة محرفية	السلسلة المراد رسمها ، مع إحداثيات نقطة بدء الرسم .
drawPolygon()	رسم مضلع	مصفوفة من النقاط تمثل زوايا المضلع بالإضافة إلى عدد الزوايا .

الجدول (٩-٤) يبين بعض التوابع التي تُستخدم في رسم الأشكال الهندسية



الشكل (٩-١) - يبين رسم القطع الناقص

٩-٥-٣-١ رسم الخطوط Drawing Lines

يمكن استخدام التابع التالي من أجل رسم خط مستقيم :

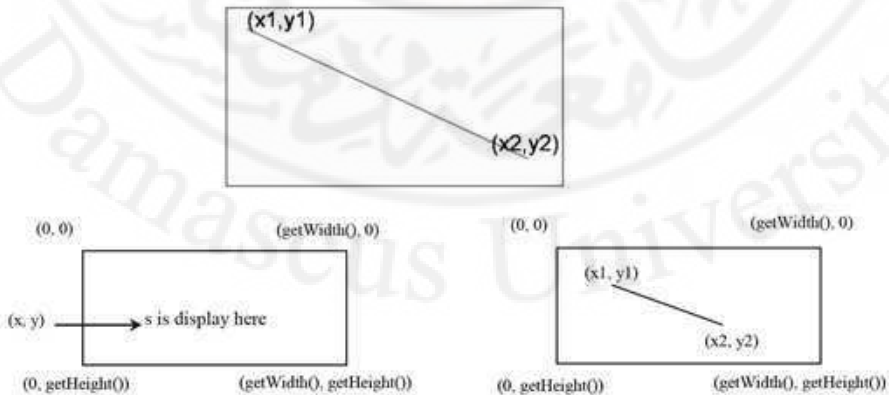
public void drawLine(int x1, int y1, int x2, int y2)

تمثل المتحولات $x1, y1, x2, y2$ نقطة البداية $(x1, y1)$ ونقطة النهاية $(x2, y2)$ والمثال التالي يرسم خط أحمر :

`g.setColor( Color.RED );`

`g.drawLine( 5, 5, 700, 300 );`

كما هو مبين في الشكل (٩-٢) .



الشكل (٩-٢) - يبين رسم مستقيم أحمر

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;
import javax.swing.JFrame;
public class Lines extends JPanel {
    // display various lines
    public void paintComponent( Graphics g ) {
        super.paintComponent( g ); // call superclass's paint method
        this.setBackground( Color.WHITE );

        g.setColor( Color.GRAY );
        g.drawLine( 0, 0, 700, 300 );

        g.setColor( Color.RED );
        g.drawLine( 0, 0, 380, 380 );
        g.setColor( Color.BLUE );
        g.drawLine( 80, 80, 380, 380 );
        g.setColor( Color.RED );
        g.drawLine( 100, 10, 20, 20 );
        g.setColor( Color.BLUE );
        g.drawLine( 10, 100, 80, 80 );

        g.setColor( Color.BLACK );
        g.drawLine( 0, 35, 500, 35 );
        g.setColor( Color.BLACK );
        g.drawLine( 0, 100, 500, 100 );

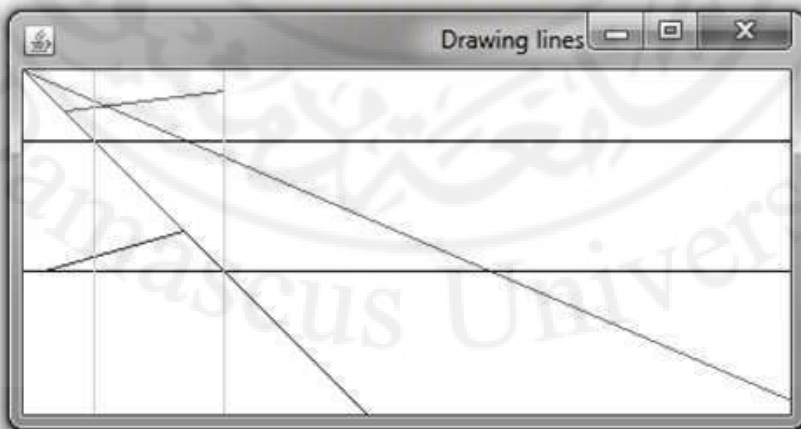
        g.setColor( Color.GREEN );
        g.drawLine( 35, 0, 35, 500 );
        g.setColor( Color.GREEN );
```

```

g.drawLine( 100, 0, 100, 500 );
    } // end method paintComponent
} // end class Lines
// program Test
public class LinesT {
    // execute application
    public static void main( String args[ ] ) {
        // create frame for Lines
        JFrame frame = new JFrame( "Drawing lines " );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        Lines ob = new Lines ( );
        ob.setBackground( Color.WHITE );
        frame.add( ob ); // add panel to frame
        frame.setSize( 400, 210 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class Lines

```



٩-٥-٢- رسم المستطيلات Rectangles

تقدم لغة java ستة توابيع من أجل رسم المستطيلات الفارغة والممتلئة ،

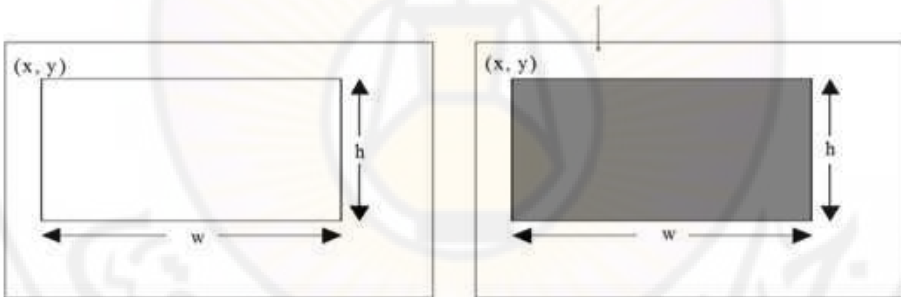
ويمكن رسم :

- مستطيلات منبسطة
- مستطيلات مدورة
- مستطيلات ثلاثية الأبعاد

نستخدم التابع التالي من أجل رسم مستطيل منبسط :

public void drawRect(int x, int y, int width, int height)

حيث يمثل المتحولان (x,y) الزاوية اليسارية العليا للمستطيل بينما يمثل المتحولان (width , height) عرض وطول المستطيل كما هو مبين في الشكل (٩-٢).



drawRect(int x, int y, int w, int h); fillRect(int x, int y, int w, int h);

الشكل (٩-٢) - يبين رسم مستطيل

التابع التالي يُستخدم من أجل ملء مستطيل منبسط :

public void fillRect(int x, int y, int width, int height)

المثال التالي يبين رسم مستطيل منبسط وممتلئ :

g.setColor(Color.BLUE);

g.drawRect(10, 40, 90, 55);

g.fillRect(10, 40, 90, 55);

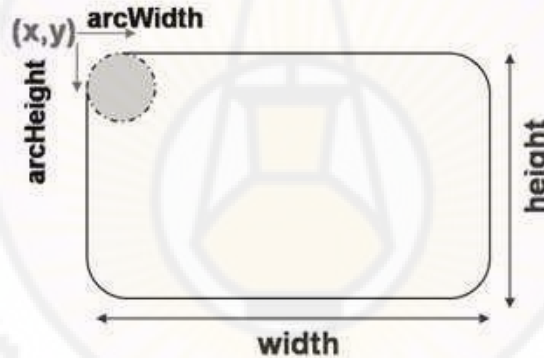
- التابع التالي يستخدم من أجل رسم مستطيل مدور :

public void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

- حيث المتحول arcWidth يمثل القطر الأفقي للأقواس التي سترسم في زوايا المستطيل .

- بينما يمثل المتحول arcHeight القطر العمودي لهذه الأقواس .

يمكننا القول إن المتحولين arcWidth, arcHeight هما عرض وارتفاع الشكل البيضائوي الذي يولد ربع دائرة في كل زاوية . ويوضح الشكل (٩-٢) رسم مستطيل مدور .



الشكل (٩-٢) - يبين رسم مستطيل مدور

- التابع التالي يستخدم من أجل ملئ مستطيل مدور :

public void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

المثال التالي يبين رسم مستطيل مدور وممتلئ :

`g.setColor( Color.CYAN );`

`g.fillRoundRect( 195, 40, 90, 55, 50, 50 );`

```
g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
```

التابع التالي يستخدم من أجل رسم مستطيل ثلاثي الأبعاد :

```
public void draw3DRect( int x, int y, int width, int height,  
                        boolean b )
```

التابع التالي يستخدم من أجل ملئ مستطيل ثلاثي الأبعاد :

```
public void fill3DRect( int x, int y, int width, int height,  
                       boolean b )
```

المتحول **boolean** b هو عبارة عن قيمة منطقية تحدد فيما إذا كان سيتم رسم المستطيل بشكل بارز فوق السطح أو سيتم حفره داخل السطح .

المثال التالي يبين رسم مستطيل ثلاثي الأبعاد وممتلئ :

```
g.setColor( Color.YELLOW );
```

```
g.draw3DRect( 5, 100, 90, 55, true );
```

```
g.fill3DRect( 100, 100, 90, 55, false );
```

مثال (9-17)

```
import java.awt.Color;
```

```
import java.awt.Graphics;
```

```
import javax.swing.JPanel;
```

```
import javax.swing.JFrame;
```

```
public class Rectangle extends JPanel {
```

```
    // display various rectangles
```

```
public void paintComponent( Graphics g ) {
```

```
    super.paintComponent( g ); // call superclass's paint method
```

```
    this.setBackground( Color.WHITE );
```

```
        g.setColor( Color.BLUE );
```

```
        g.drawRect( 5, 40, 200, 55 );
```

```
        g.setColor( Color.RED );
```

```

g.fillRect( 100, 100, 59 , 55 );
g.setColor( Color.CYAN );
g.fillRoundRect( 290, 40, 90, 55, 30, 30 );
g.setColor( Color.BLACK );
g.drawRoundRect( 20, 40, 90, 55, 50, 50 );
g.setColor( Color.RED );
g.draw3DRect( 20 , 10 , 150, 150, true );
g.setColor( Color.ORANGE );
g.fill3DRect( 200, 100, 90, 55, false );

} // end method paintComponent
} // end class

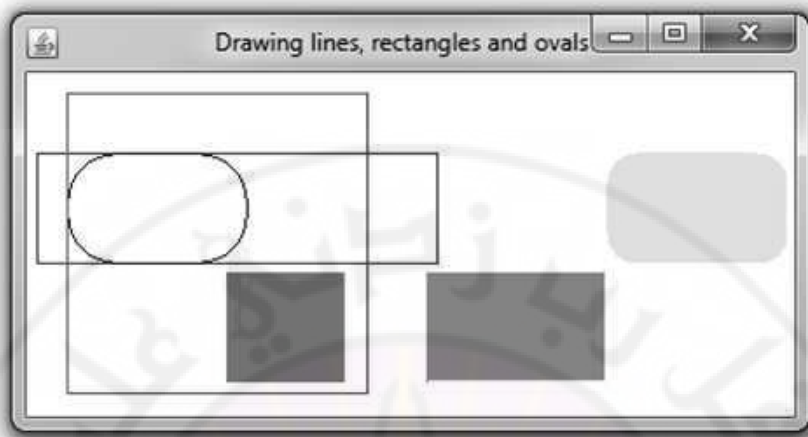
// execute application
public class Rects {

    public static void main( String args[ ] ) {
        // create frame for Rectangles
        JFrame frame = new JFrame( "Drawing rectangles " );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        Rectangle ob = new Rectangle ( );
        ob.setBackground( Color.WHITE );
        frame.add( ob ); // add panel to frame
        frame.setSize( 400, 210 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class Rects

```

وعند تنفيذ البرنامج يظهر الخرج التالي :

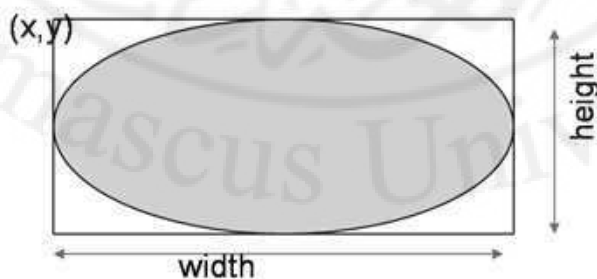


٩-٥-٣-٣ - رسم الأشكال البيضاوية Drawing Ovals

يمكن رسم شكل بيضاوي (قطع ناقص) في لغة java بالاعتماد على المستطيل الذي يحيط به ، لذلك نكتب المتحولات في توابع رسم القطع الناقص وكأننا نرسم مستطيلاً . ونستخدم التابع التالي لرسم أي شكل بيضاوي (قطع ناقص) فارغ :

public void drawOval(int x, int y, int width, int height)

يحدد المتحولان x,y الزاوية اليسارية العليا للمستطيل الذي يحيط بالقطع الناقص بينما يحدد المتحولان width, height عرض وطول المستطيل . ويبين الشكل (٩-٣) رسم الشكل البيضاوي .



الشكل (٩-٣) - يبين رسم الشكل البيضاوي

يستخدم التابع التالي من أجل ملئ قطع ناقص :

```
public void fillOval( int x, int y, int width, int height )
```

يبين المثال رسم قطع ناقص وممتليء :

```
g.setColor( Color.MAGENTA );
```

```
g.drawOval( 195, 100, 90, 55 );
```

```
g.fillOval( 290, 100, 90, 55 );
```

ويبين المثال (9-18) رسم مستقيم ومستطيل منبسط ومدور وقطع ناقص.

مثال (9-18)

```
import java.awt.Color;  
import java.awt.Graphics;  
import javax.swing.JPanel;  
import javax.swing.JFrame
```

```
public class LROJPanel extends JPanel {  
    // display various lines, rectangles and ovals  
    public void paintComponent( Graphics g ) {  
        super.paintComponent( g ); // call superclass's paint method  
        this.setBackground( Color.WHITE );
```

```
        g.setColor( Color.RED );  
        g.drawLine( 5, 30, 380, 30 );  
        g.setColor( Color.BLUE );  
        g.drawRect( 5, 40, 90, 55 );  
        g.fillRect( 100, 40, 90, 55 );
```

```
        g.setColor( Color.BLACK );  
        g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
```

```

g.drawRoundRect( 290, 40, 90, 55, 20, 20 );

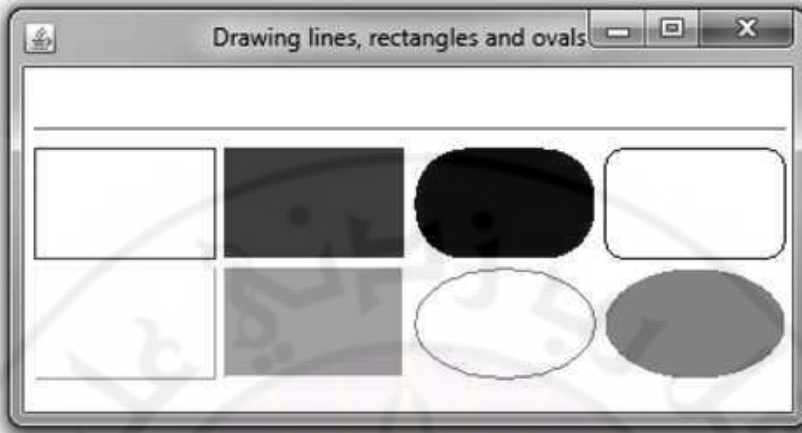
g.setColor( Color.YELLOW );
g.draw3DRect( 5, 100, 90, 55, true );
g.fill3DRect( 100, 100, 90, 55, false );

g.setColor( Color.MAGENTA );
g.drawOval( 195, 100, 90, 55 );
g.fillOval( 290, 100, 90, 55 );
} // end method paintComponent
} // end class LROJPanel

// Drawing lines, rectangles and ovals.
public class LinesRectsOvals {
    // execute application
    public static void main( String args[ ] ) {
        // create frame for LinesRectsOvals.JPanel
        JFrame frame = new JFrame( "Drawing lines,
                                   rectangles and ovals");
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
        LROJPanel ob = new LROJPanel ( );
        ob.setBackground( Color.WHITE );
        frame.add(ob ); // add panel to frame
        frame.setSize( 400, 210 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class LinesRectsOvals

```

وعند تنفيذ البرنامج يظهر الخرج التالي :



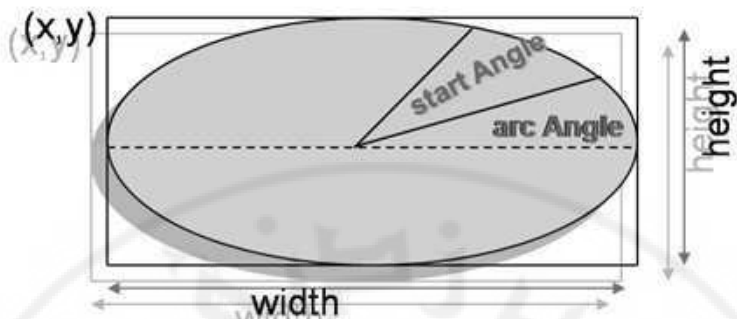
٩-٥-٣-٤ - رسم الأقواس Drawing Arcs

يعتبر القوس من القطع الناقص (الشكل البيضاوي) لذلك يتم رسم القوس بالاعتماد على المستطيل المحيط بالشكل البيضاوي . و يُستخدم التابعين التاليين من أجل رسم قوس فارغ وقوس ممتلئ :

```
public void drawArc( int x, int y, int width, int height,  
int startAngle, int arcAngle )  
public void fillArc( int x, int y, int width, int height, int  
startAngle, int arcAngle )
```

المتحولات x,y و width, height هي نفسها لرسم القطع الناقص في التابع drawOva() بينما يمثل المتحول startAngle زاوية البداية والمتحول arcAngle زاوية الامتداد (أي الزاوية التي يغطيها القوس) .

تقاس الزوايا بالدرجات وتتبع التقاليد الرياضية المعروفة ، أي تقع الدرجة 0 في الاتجاه الشرقي ، وتدل الزاوية الموجبة على الدوران من الاتجاه الشرقي وبعكس عقارب الساعة . يمثل الشكل (٩-١) التالي قطع ناقص والقوس المطلوب رسمه بالاعتماد على القطع الناقص .



الشكل (٩-١) - كيفية رسم قوس

المثال التالي يبين رسم مستطيل وقوس ممثل:

```
// start at 0 and sweep 110 degrees
g.setColor( Color.RED );
g.drawRect( 100, 35, 80, 80 );
g.setColor( Color.BLACK );
g.drawArc( 100, 35, 80, 80, 0, 110 );
```

مثال (٩-١٩)

```
// Drawing arcs.
```

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;
```

```
public class ArcsJPanel extends JPanel {
// draw rectangles and arcs
public void paintComponent( Graphics g ) {
// call superclass's paintComponent
super.paintComponent( g );
// start at 0 and sweep 360 degrees
g.setColor( Color.RED );
```

```

g.drawRect( 15, 35, 80, 80 );
g.setColor( Color.BLACK );
g.drawArc( 15, 35, 80, 80, 0, 360 );

// start at 0 and sweep 110 degrees
g.setColor( Color.RED );
g.drawRect( 100, 35, 80, 80 );
g.setColor( Color.BLUE );
g.drawArc( 100, 35, 80, 80, 0, 110 );

// start at 0 and sweep -270 degrees
g.setColor( Color.RED );
g.drawRect( 185, 35, 80, 80 );
g.setColor( Color.BLACK );
g.drawArc( 185, 35, 80, 80, 0, -270 );

// start at 0 and sweep 360 degrees
g.fillArc( 15, 120, 80, 40, 0, 360 );

// start at 270 and sweep -90 degrees
g.fillArc( 100, 120, 80, 40, 270, -90 );

// start at 0 and sweep -270 degrees
g.fillArc( 185, 120, 80, 40, 0, -270 );
} // end method paintComponent
} // end class ArcsJPanel

//-----
// Drawing arcs.
import javax.swing.JFrame;
public class DrawArcs {
    // execute application
    public static void main( String args[ ] ) {

```

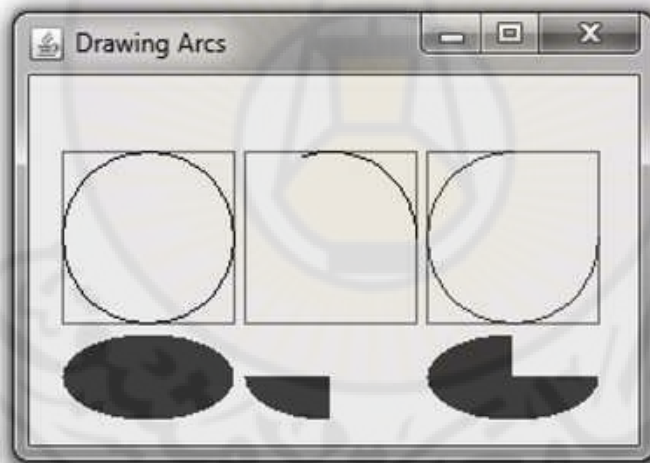


```

// create frame for ArcsJPanel
Jframe frame = new JFrame( "Drawing Arcs" );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
// create ArcsJPanel
ArcsJPanel arcsJPanel = new ArcsJPanel();
// add arcsJPanel to frame
frame.add( arcsJPanel );
frame.setSize( 300, 210 ); // set frame size
frame.setVisible( true ); // display frame
} // end main
} // end class DrawArcs

```

وعند تنفيذ البرنامج يظهر الخرج التالي :



٩-٥-٣-٥ - رسم المضلعات والخطوط المتعددة

Drawing Polygons and Polylines

يغلف الصف Polygon وصفاً كاملاً لمنطقة مغلقة وثنائية البعد ضمن الاحداثيات الفضائية ، وتحدد المنطقة بعدد معين من القطع المستقيمة حيث تمثل كل قطعة مستقيمة أحد الجوانب أو الحواف edge التابعة للمضلع .

يتألف المضلع داخلياً من قائمة من الإحداثيات الثنائية (x,y) والتي تعرف كل منها ذروة تابعة للمضلع ، كما يمثل كل إحداثيتان متعاقبتان نقطتي البداية والنهاية لخط يشكل أحد أضلاع المضلع ، وتتصل النقطة الأولى مع النقطة الأخيرة بواسطة القطعة المستقيمة التي تغلق المضلع .

يُستخدم البانيين التاليين من أجل صنع غرض من الصف Polygon :

- الباني `public Polygon()` يبني مضلعاً فارغاً .

- الباني

`public Polygon( int[ ] xpoints, int[ ] ypoints, int npoints )`

يبني هذا الباني المضلع ويهيئه بالنقاط الممررة ، وإن الوسيطين `xpoints` و `ypoints` هما عبارة عن مصفوفتين تمثلان الإحداثيات على المحور `x` و المحور `y` ، أما الوسيط `npoints` يمثل عدد النقاط .

ويجب استخدام التابع `( int x , int y ) addPoint()` من أجل إضافة نقطة إلى المضلع . ويملك الصف `Polygon` حقول المعطيات العامة `xpoints` و `ypoints` و `npoints` والتي تمثل مصفوفة الإحداثيات على المحور `x` و `y` والعدد الكلي للنقاط . ويبين البرنامج الجزئي التالي كيفية رسم مضلع وإضافة النقاط إليه :

```
Polygon polygon = new Polygon( ) ;
```

```
polygon.addpoint(40,20) ;
```

```
polygon.addpoint(70,40) ;
```

```
polygon.addpoint(60,80) ;
```

```
polygon.addpoint(45,45) ;
```

```
polygon.addpoint(20,60) ;
```

نستخدم أحد التوابع التالية من أجل رسم المضلعات الفارغة والممتلئة :

```

drwPolygon(Polygon polygon1);
fillPolygon(Polygon polygon1);
drwPolygon( int[ ] xpoints,int[ ] ypoints,int npoints );
fillPolygon( int[ ] xpoints,int[ ] ypoints,int npoints );

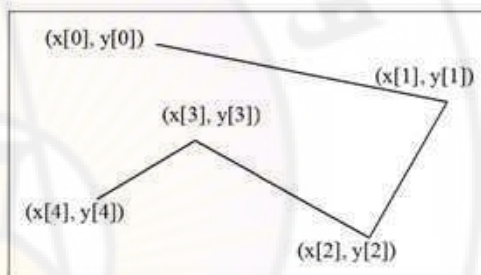
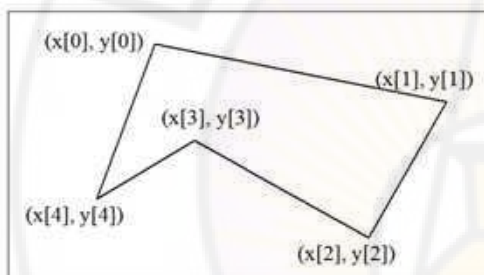
```

مثال :

```

int x = { 40, 70 , 60 , 45 , 20 } ;
int y = { 20, 40 , 80 , 45 , 60 } ;

```



```

g. drwPolygon( x, y ,length); g.drawPolyline(x, y, x.length);

```

//Drawing polygons.

```

import java.awt.Graphics;
import java.awt.Polygon;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.Color;

public class PolygonsJPanel extends JPanel {
    // draw polygons and polylines paintComponent
    public void paintComponent( Graphics g ) {
        super.paintComponent( g ); // call superclass's
        g.setColor( Color.BLACK );
    }
}

```

مثال (20-9)

```

// draw polygon with Polygon object
    int xValues[ ] = { 20, 50, 100, 100, 20, 15 };
    int yValues[ ] = { 50, 50, 50, 80, 80, 60 };
    Polygon polygon1 = new Polygon( xValues, yValues, 6 );
    g.drawPolygon( polygon1 );

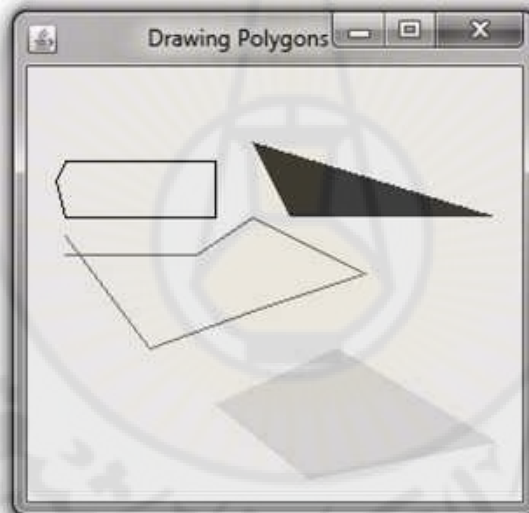
    g.setColor( Color.RED );
// draw polylines with two arrays
    int xValues2[ ] = { 20, 90, 120, 180, 120, 65, 20 };
    int yValues2[ ] = { 100, 100, 80, 110, 130, 150, 90 };
    g.drawPolyline( xValues2, yValues2, 7 );
    g.setColor( Color.BLUE );
// fill polygon with two arrays
    int xValues3[ ] = { 120, 140, 180, 250 };
    int yValues3[ ] = { 40, 80, 80, 80 };
    g.fillPolygon( xValues3, yValues3, 4 );
    g.setColor( Color.GREEN );
// draw filled polygon with Polygon object
    Polygon polygon2 = new Polygon();
    polygon2.addPoint( 165, 150 );
    polygon2.addPoint( 165, 150 );
    polygon2.addPoint( 250, 200 );
    polygon2.addPoint( 150, 220 );
    polygon2.addPoint( 100, 180 );
    g.fillPolygon( polygon2 );
} // end method paintComponent
} // end class Polygons.JPanel
// Drawing polygons
public class DrawPolygons {
// execute application
    public static void main( String args[ ] ) {

```



```
// create frame for Polygons.JPanel
JFrame frame = new JFrame( "Drawing Polygons" );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
Polygons.JPanel polygonsJPanel = new Polygons.JPanel();
frame.add( polygonsJPanel ); // add polygonsJPanel to frame
frame.setSize( 280, 270 ); // set frame size
frame.setVisible( true ); // display frame
} // end main
} // end class DrawPolygons
```

وعند تنفيذ البرنامج يظهر الخرج التالي :



٩-٥-٤ - التحكم بمظهر الشكل المراد رسمه

توفر Java 2D API التحكم بمظهر الشكل المراد رسمه وخاصة الشكل ثنائي الأبعاد ونستخدم الحزم التالية لمعالجة الرسومات المعقدة مثل الخطوط الفنية والنصوص والصور .

```
java.awt.color, java.awt.font, java.awt.geom,
java.awt.image.renderable, java.awt.print
```


ونبين في هذه الفقرة بعض الإمكانيات المختلفة في التحكم بمظهر الرسم :

- كشكل خط الرسم

- نوع التعبئة أو التظليل الخ

وهذه التقنيات موجودة في الصف Graphics2D الذي يرث الصف Graphics ويزودنا الصف Graphics2D بعدد من التوابع بعضها يتحكم بشكل خط الرسم مثل التابع setStroke . ومنها ما يتحكم بأسلوب تعبئة الشكل المرسوم setPaint ، كما أن هذا الصف يختلف عن الصف Graphics ، بحيث نستخدم تابع واحد في الرسم يدعى draw أو fill للشكل المصمت ونمرر له الشكل المراد رسمه . ويأخذ التابع setStroke وسيطاً من أحد الصفوف التي تحقق الواجهة Stroke ومن الصفوف التي تحقق الواجهة يوجد الصف BasicStroke . أما التابع setPaint فيأخذ وسيطاً من أحد الصفوف التي تحقق الواجهة Paint ومنها الصفوف Color , Graphics , TexturePaint . الصف Graphics2D هو صف مشتق من الصف الأساسي Graphics ونستخدم الحزمة java.awt . للوصول إلى قدرات الصف Graphics2D ويجب علينا تحويل غرض من (g) الصف Graphics إلى غرض من الصف Graphics2D الذي سيتم تمريره إلى التابع paintComponent كما يلي :

```
Graphics2D g2d = ( Graphics2D ) g;
```

لنأخذ المثال (9-21) الذي يوضح التحكم بخط الرسم .

مثال (9-21)

```
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;
public class DrawingDecoredShape extends JPanel {
```

//1

```

public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2d = (Graphics2D)g; //2
    g2d.setStroke(new BasicStroke(9)); //3
    g2d.draw(new Line2D.Double(0,0,100,100)); //4
    float dash[ ] = {5};
    g2d.setStroke(new BasicStroke(1.0f,
        BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER,
        10.0f, dash, 0.0f)); //5
    g2d.draw(new Arc2D.Double(10,50,100,100,20,140,Arc2D.PIE));
    g2d.setPaint(new GradientPaint( 0,100,
        Color.green,200,100,Color.white));
    g2d.fill(new RoundRectangle2D.Double(0,100,200,200,10,10));
}

    float dash[ ] = {5};
    g2d.setStroke(new BasicStroke(1.0f,
        BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER,
        10.0f, dash, 0.0f)); //5
    g2d.draw(new Arc2D.Double(10,50,100,100,20,140,Arc2D.PIE));
    g2d.setPaint(new GradientPaint( 0,100,
        Color.green,200,100,Color.white));
    g2d.fill(new RoundRectangle2D.Double(0,100,200,200,10,10));
}

public static void main(String[ ] args)
{
    JFrame main = new JFrame("Graphics2D");
    main.add(new DrawingDecoredShape());
    main.setVisible(true);
    main.setSize(new Dimension(300,300));
    main.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

```

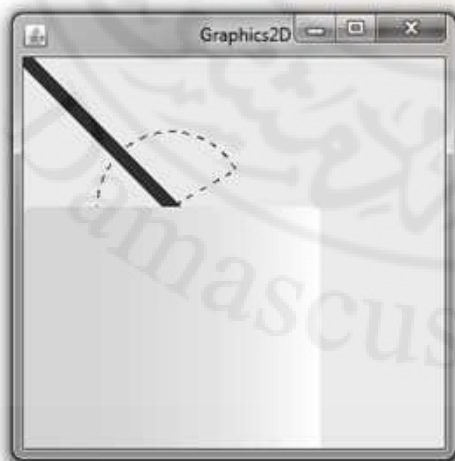
```
} // end main
} // end class
```

وعند تنفيذ البرنامج يظهر الخرج التالي :

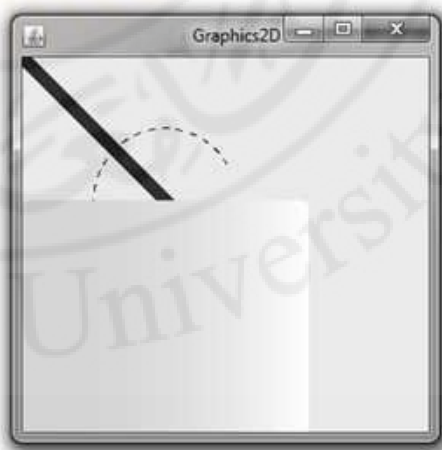


- في السطر /1/ قمنا باستيراد الحزمة `import java.awt.geom.*;` وذلك من أجل التمكن من استخدام بعض توابع الرسم .
- في السطر /2/ التابع `paint` يقوم بتحويل الغرض `g` من الصف `Graphics` إلى غرض من الصف `Graphics2D` وذلك من أجل الاستفادة من إمكانيات هذا الصف .
- في السطر /3/ استخدمنا التابع `setStroke` ومررنا له غرض من الصف `BasicStroke` ، من أجل التحكم بشكل خط الرسم ، وهنا مررنا القيمة 9 من أجل جعل خط الرسم عريض ، أي أن القيمة الممررة إلى باني هذا الصف تمثل عرض خط الرسم الذي سيتم استخدامه في رسم الشكل .

- في السطر /4/ استدعينا التابع draw وهو تابع للغرض g2d ومررنا له غرضاً من الصف Line2D.Double المشتق من الصف Line2D وذلك من أجل رسم خط وفق الإحداثيات الممررة .
 - ثم نمرر للتابع draw غرضاً من الصف Arc2D.Double لرسم قوس وفق الاحداثيات الممررة ، أما الثابت الأخير فهو أحد القيم التالية :
 - **Arc2D.CHORD** : ويُستخدم لرسم قوس مغلق ، أي رسم خط مستقيم بين نقطتي بداية القوس ونهايته .
 - **Arc2D.OPEN** : رسم قوس مفتوح ، أي لا يوجد أي اتصال بين نقطتي بداية القوس ونهايته .
 - **Arc2D.CHORD** : رسم قوس مغلق من خلال رسم خط مستقيم بين نقطة بداية القوس ونقطة المركز للقطع الناقص (الذي تم اقتطاع القوس منه) ومن ثم رسم خط مستقيم من تلك النقطة إلى نقطة نهاية القوس .
- ويبين الشكل التالي الثوابت المختلفة المستخدمة في رسم الأقواس.



CHORD



OPEN

إن الصفوف المستخدمة مع التابع setPaint هي : الصف Color والصف GradientPaint و الصف TexturePaint ، حيث يتم استخدام أغراض من هذه الصفوف كوسطاء للتابع setPaint . حيث يمكن استخدام أغراض من الصف Color لتعيين لون ملء الشكل ، أما الصف GradientPaint فيستخدم من أجل ملء الشكل بتدرج لوني ابتداءً بلون ما وانتهاءً بلون آخر ، حيث نقوم بتحديد نقطة من الشكل والتي سيبدأ من عندها التدرج اللوني ، ومن ثم اللون المراد الابتداء به ، وبعدها النقطة التي سيتم الانتهاء إليها ، وأخيراً اللون الذي سيتوقف عنده التدرج اللوني ، وهذا ما قمنا به في المثال السابق . أما الصف TexturePaint فيمكن استخدامه عندما نريد استخدام صورة لملء الشكل المراد رسمه .

٩-٥-٤-١ الصف BasicStroke

يعرف هذا الصف مجموعة من الخصائص التي تُستخدم لضبط شكل حدود الأشكال المرسومة باستخدام غرض من الصف Graphics2D . أي أنه يُعرف نوع خط الرسم الذي سيستخدم لرسم الحدود الخارجية للشكل outline . و تتضمن هذه الخصائص عدة أمور :

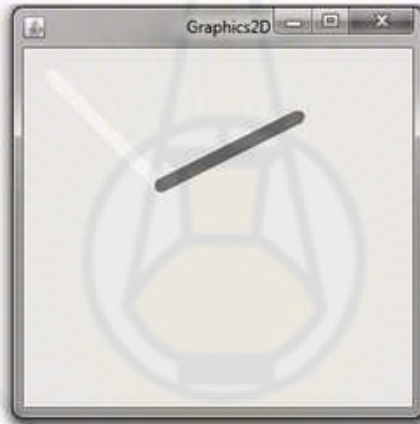
- عرض خط الرسم .
 - شكل نهاية الرسم .
 - شكل نقاط الربط بين المسارات الجزئية .
 - تعريف نموذج ما pattern لشكل الخط المنقط dashed Line .
- إن باني الصف BasicStroke محمل بشكل زائد وله 5 صيغ ولتوضيح كل منها سنقوم بعرض مثال بسيط ، أخذين بعين الاعتبار عدة حالات :

الحالة الأولى : بانى بوسيط واحد

للبناني وسيط واحد من نمط float يمثل عرض خط الرسم كما في المثال (9-21) السابق .

الحالة الثانية : بانى بثلاثة وسطاء

لنستبدل في المثال (9-21) النص البرمجي للتابع () paint بالنص في المثال (9-22) التالي. في الواقع نريد تغيير قيم الوسطاء للتابع () setStroke عند تنفيذ البرنامج سنلاحظ أن الخط المرسوم سيكون كما في الشكل (9-1) التالي :



الشكل (9-1) - تمرير الوسيط BasicStroke.CAP_ROUND

مثال (9-22)

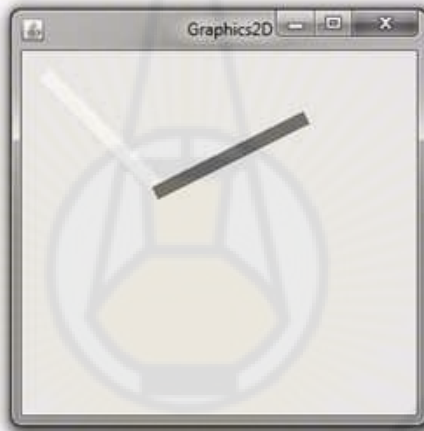
```
public void paint(Graphics g) {  
    super.paint(g);  
    Graphics2D g2d = (Graphics2D)g;  
    // BasicStroke.CAP_SQUARE  
    g2d.setStroke(new BasicStroke(10, BasicStroke.CAP_ROUND,  
        BasicStroke.JOIN_ROUND));  
    g2d.setColor( Color.yellow);  
}
```

```

g2d.draw(new Line2D.Double(20,20,100,100));
g2d.setColor( Color.red);
g2d.draw(new Line2D.Double(100,100,200,50));
} //end paint

```

نلاحظ أن نهايتي الخطين المرسومين هي دائرية ، وذلك لأننا مررنا الوسيط BasicStroke.CAP_ROUND ، وإذا غيرنا هذا الوسيط إلى القيمة BasicStroke.CAP_SQUARE فسنحصل على الشكل (٩-٢) التالي :



(٩-٢) - يبين تمرير الوسيط BasicStroke.CAP_SQUARE

لتوضيح معنى الوسيط الثالث فنستخدم الصف generalPath وهو أحد صفوف الحزمة java.awt.geom ، ويُستخدم الصف لتمثيل مجموعة من الأشكال الهندسية أي مجموعة من المسارات .

سنقوم باستبدال النص البرمجي الموجود في التابع paint() في المثال (9-21) بالنص البرمجي في المثال (9-23) . وعند تنفيذ البرنامج سنحصل على الشكل (٩-٣) التالي :



الشكل (٩-٣) - تمرير الوسيط الثالث BasicStroke.CAP_SQUARE

مثال (٩-٢٣)

```
public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2d = (Graphics2D)g;
    g2d.setStroke(new BasicStroke(10,
        BasicStroke.CAP_SQUARE, BasicStroke.JOIN_ROUND));
    GeneralPath generalPath = new GeneralPath(new
        Line2D.Double(20,20,50,50) );
    generalPath.append(new
        Line2D.Double(100,100,180,20),true);
    g2d.setColor( Color.red);
    g2d.draw(generalPath);
} //end pain
```

في الحقيقة يوجد عدة ملاحظات حول النص البرمجي :

١. لقد مررنا أولاً إلى الغرض generalPath المعرف في الصف GeneralPath الشكل الأول (المسار الأول) وهو عبارة خط ، ثم استخدمنا التابع (append) من أجل إضافة خط جديد إلى الأول ، أي

أن الصف `GeneralPath` يمثل مجموعة من الأشكال الهندسية التي نقوم بتشكيلها الواحد تلو الآخر .

٢. نلاحظ أن الخط الأول ينتهي عند النقطة (50,50) ، وأما الثاني فيبدأ عند النقطة (100,100) ، أي أن هناك فاصل بين الخطين ، إلا أن الشكل السابق لا يظهر هذا الفاصل ، والحقيقة أن ذلك يعود للوسيط `true` الممرر إلى التابع `append()` والذي يقوم بدوره بوصل المسارين ولتوضيح ذلك نستبدل القيمة `true` بالقيمة `false` .

٣. بعد ذلك نقوم بتمرير الغرض `generalPath` على التابع `draw()` الذي يقوم بدوره برسم مجموعة الأشكال الذي يتضمنها هذا الغرض .

الحالة الثالثة : وهي الحالة العامة - باتي بستة وسطاء

في هذه الحالة نمرر ستة وسطاء لكل وسيط مهمة معينة ، الوسطاء الثلاثة الأولى تم مناقشتها من خلال الحالتين السابقتين ، أما الوسطاء المتبقية فهي تعرف كيفية تطبيق نموذج معين `pattern` عند رسم الشكل ، كما يوضح المثال (9-24) التالي ، ويمكن استبدال النص البرمجي في التابع `paint()` في المثال (9-21) بالنص البرمجي في المثال (9-24) التالي ، وعند تنفيذ البرنامج سنحصل على الشكل التالي .



الشكل (٩-٤) - يبين تمرير الوسطاء الستة للتابع `setStroke`

```
public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2d = (Graphics2D)g;
    float dash[ ]= { 10, 20, 5, 5 } ;
    g2d.setStroke(new BasicStroke(10.0f,
        BasicStroke.CAP_BUTT,
        BasicStroke.JOIN_MITER,
        10.0f, dash, 10.0f));
    g2d.setColor( Color.blue);
    g2d.draw(new Line2D.Double( 0,10,300,10));
    float dash2[ ]= { 10, 20, 5, 5 } ;
    g2d.setStroke(new BasicStroke(2.0f, BasicStroke.CAP_BUTT,
        BasicStroke.JOIN_MITER, 10.0f, dash, 0.0f));
    g2d.setColor( Color.red);
    g2d.draw(new Line2D.Double( 0,30,300,30));
} // end pain
```

في الحقيقة إن الوسيط الخامس يمثل النموذج pattern الذي سيتم تطبيقه عند الرسم ، وتمثل عناصر المصفوفة سلسلة من الأرقام ، حيث يدل الرقم الأول على طول أول قطعة سيتم رسمها من الخط ، أما الرقم الثاني من هذه المصفوفة فيمثل طول الفراغ بعد القطعة التي تم رسمها وهكذا بالتناوب . أما الوسيط السادس فيمثل الإزاحة التي سيبدأ تطبيق النموذج من عندها ، فمثلاً في النص البرمجي السابق ، ومن أجل الخط الأول تم ضبط قيمة الإزاحة على القيمة 10 لذلك نلاحظ أنه لم يبدأ من النقطة (0,10) على عكس الخط الثاني والذي تم ضبط إزاحته على القيمة 0 .

٩-٥-٤-٢- التحكم بالرسم من خلال التدوير

يمكننا التحكم من خلال التابع `rotate( )` ، بحيث نقوم بإجراء دوران للرسومات حول مبدأ الإحداثيات ، ويمكن استبدال النص البرمجي في التابع `paint( )` في المثال (9-21) بالنص البرمجي في المثال (9-25) التالي ، وعند تنفيذ البرنامج سنحصل على الشكل (٩-٤) التالي .

```
public void paint(Graphics g) {  
    super.paint(g);  
    Graphics2D g2d = (Graphics2D)g;  
    g2d.rotate(45);  
    g2d.setColor( Color.blue);  
    g2d.setFont(new Font("TimeRoman", Font.PLAIN, 20));  
    g2d.drawString("Mmoun !!!", 75, 5);  
    g2d.rotate(-135);  
    g2d.setColor( Color.red);  
    g2d.setFont(new Font("TimeRoman", Font.PLAIN, 20));  
    g2d.drawString("Younes !!!",-170 , 100);  
} //end pain
```



الشكل (٩-٤) - يبين مخرجات المثال (9-25)

نستخدم في الرسم بواسطة API 2D Java الصف المشتق المجرد Graphics2D من الصف الأساسي Graphics ونستخدم الحزمة java.awt للوصول إلى قدرات Graphics2D يجب علينا تحويل Graphics reference (g) الذي تم تمريره إلى التابع paintComponent إلى Graphics2D reference مع التعليمة كما يلي :

```
Graphics2D g2d = ( Graphics2D ) g;
```

البرنامج التالي يستخدم هذه التقنية .

يوضح المثال التالي عدة أشكال Java 2D باستخدام الحزمة java.awt.geom، بما في ذلك «Rectangle2D.Double، «Line2D.Double، «Ellipse2D.Double، «Arc2D.Double، «RoundRectangle2D.Double كل صف من هذه الصفوف يمثل شكل ذات أبعاد محددة ودقيقة . في هذا المثال، نرسم الأشكال ونعدل خصائص الرسم، مثل تغيير سمك الخط، وملء الأشكال مع أنماط ورسم الخطوط المتقطعة.



مسائل عامة محلولة

١- اكتب برنامجاً بلغة Java يقوم برسم وجه مبتسم باستخدام الألوان والأشكال الممثلة.

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;
import javax.swing.JFrame;

public class DrawSmiley extends JPanel {
    public void paintComponent( Graphics g ) {
        super.paintComponent( g );
        // draw the face
        g.setColor( Color.YELLOW );
        g.fillOval( 10, 10, 200, 200 );

        // draw the eyes
        g.setColor( Color.BLACK );
        g.fillOval( 55, 65, 30, 30 );
        g.fillOval( 135, 65, 30, 30 );

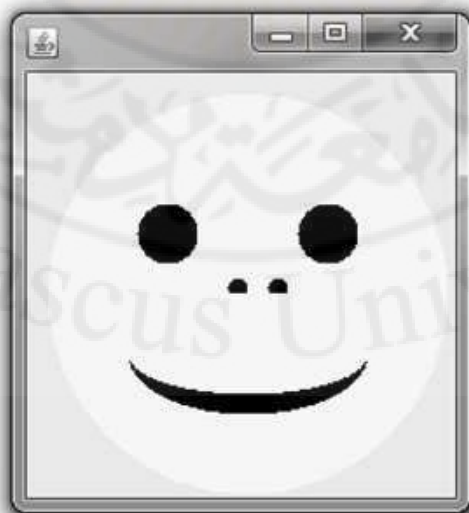
        // draw the nose
        g.setColor( Color.BLACK );
        g.fillOval( 100, 102, 10, 10 );
        g.fillOval( 120, 102, 10, 10 );

        // draw the mouth
        g.fillOval( 50, 110, 120, 60 );
```

```
// "touch up" the mouth into a smile
    g.setColor( Color.YELLOW );
    g.fillRect( 50, 110, 120, 30 );
    g.fillOval( 50, 120, 120, 40 );
} // end method paintComponent
} // end class DrawSmiley

// Test application that displays a smiley face.
public class DrawSmileyTest {
public static void main( String args[ ] ) {
    DrawSmiley panel = new DrawSmiley( );
    JFrame application = new JFrame();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    application.add( panel );
    application.setSize( 230, 250 );
    application.setVisible( true );
} // end main
} // end class DrawSmileyTest
```

وعند تنفيذ البرنامج سنحصل على الشكل التالي .



٢- يوضح المثال التالي رسم عدة أشكال باستخدام تقنية Java 2D وذلك باستخدام الحزمة java.awt.geom، بما في ذلك الصفوف Line2D.Double، Rectangle2D.Double، RoundRectangle2D.Double، Ellipse2D.Double، Arc2D.Double كل صف من هذه الصفوف يمثل شكل ذات أبعاد محددة ودقيقة . في هذا المثال، نرسم الأشكال ونعدل خصائص الرسم، مثل تغيير سمك الخط، وملء الأشكال مع أنماط ورسم الخطوط المنقطعة. وبالتالي رسم الأشكال الهندسية التالي:

Lines, Rectangles, Round Rectangles, Arcs and Ellipses

البرنامج :

// Demonstrating some Java 2D shapes.

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.BasicStroke;
import java.awt.GradientPaint;
import java.awt.TexturePaint;
import java.awt.Rectangle;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import java.awt.geom.RoundRectangle2D;
import java.awt.geom.Arc2D;
import java.awt.geom.Line2D;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

public class ShapesJPanel extends JPanel {
    // draw shapes with Java 2D API
    public void paintComponent( Graphics g ) {
        super.paintComponent( g ); // call superclass's paintComponent
```



```

Graphics2D g2d = ( Graphics2D ) g; // cast g to Graphics2D

// draw 2D ellipse filled with a blue-yellow gradient
g2d.setPaint( new GradientPaint( 5, 30, Color.BLUE, 35, 100,
                                Color.YELLOW, true ) );
g2d.fill( new Ellipse2D.Double( 5, 30, 65, 100 ) );

// draw 2D rectangle in red
g2d.setPaint( Color.RED );
g2d.setStroke( new BasicStroke( 10.0f ) );
g2d.draw( new Rectangle2D.Double( 80, 30, 65, 100 ) );

// draw 2D rounded rectangle with a buffered background
BufferedImage buffImage = new BufferedImage( 10, 10,
                                                BufferedImage.TYPE_INT_RGB );

// obtain Graphics2D from buffImage and draw on it
Graphics2D gg = buffImage.createGraphics( );
gg.setColor( Color.YELLOW ); // draw in yellow
gg.fillRect( 0, 0, 10, 10 ); // draw a filled rectangle
gg.setColor( Color.BLACK ); // draw in black
gg.drawRect( 1, 1, 6, 6 ); // draw a rectangle
gg.setColor( Color.BLUE ); // draw in blue
gg.fillRect( 1, 1, 3, 3 ); // draw a filled rectangle
gg.setColor( Color.RED ); // draw in red
gg.fillRect( 4, 4, 3, 3 ); // draw a filled rectangle

// paint buffImage onto the JFrame
g2d.setPaint( new TexturePaint( buffImage,
                                new Rectangle( 10, 10 ) ) );
g2d.fill(
    new RoundRectangle2D.Double( 155, 30, 75, 100, 50, 50 ) );

// draw 2D pie-shaped arc in BLACK
g2d.setPaint( Color.BLACK );

```

```

    g2d.setStroke( new BasicStroke( 6.0f ) );
    g2d.draw(
    new Arc2D.Double( 240, 30, 75, 100, 0, 270, Arc2D.PIE ) );

// draw 2D lines in green and blue
    g2d.setPaint( Color.GREEN );
    g2d.draw( new Line2D.Double( 395, 30, 320, 150 ) );

// draw 2D line using stroke
    float dashes[ ] = { 10 }; // specify dash pattern
    g2d.setPaint( Color.BLUE );
    g2d.setStroke( new BasicStroke( 4, BasicStroke.CAP_ROUND,
    BasicStroke.JOIN_ROUND, 10, dashes, 0 ) );
    g2d.draw( new Line2D.Double( 320, 30, 395, 150 ) );
} // end method paintComponent
} // end class Shapes.JPanel
//-----
// Demonstrating some Java 2D shapes.
import javax.swing.JFrame;

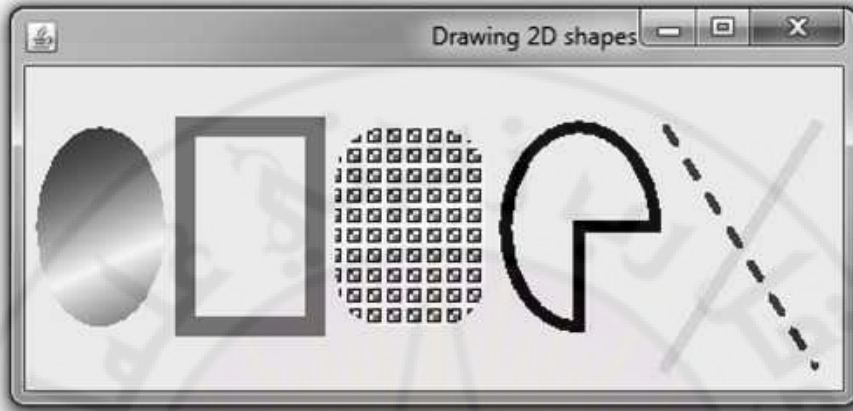
public class Shapes {
// execute application
    public static void main( String args[ ] ) {
// create frame for Shapes.JPanel
        JFrame frame = new JFrame( "Drawing 2D shapes" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

// create Shapes.JPanel
        Shapes.JPanel shapesJPanel = new Shapes.JPanel();

        frame.add( shapesJPanel ); // add shapesJPanel to frame
        frame.setSize( 425, 200 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class Shapes

```

وعند تنفيذ البرنامج سنحصل على الشكل التالي .



٣- يوضح المثال التالي استخدام الصف `generalPath` وهو أحد صفوف الحزمة `java.awt.geom`، ويُستخدم الصف لتمثيل مجموعة من الأشكال الهندسية أي مجموعة من المسارات . المثال التالي يعرض المسار العام `generalPath` في شكل نجمة خماسية .

البرنامج :

// Demonstrating a general path.

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.GeneralPath;
import java.util.Random;
import javax.swing.JPanel;

public class Shapes2JPanel extends JPanel {
    // draw general paths
    public void paintComponent( Graphics g )
    {
```

```

super.paintComponent( g ); // call superclass's paintComponent
// get random number generator
Random random = new Random( );
    int xPoints[] = { 55, 67, 109, 73, 83, 55, 27, 37, 1, 43 };
    int yPoints[] = { 0, 36, 36, 54, 96, 72, 96, 54, 36, 36 };

Graphics2D g2d = ( Graphics2D ) g;
    // create GeneralPath object
GeneralPath star = new GeneralPath( );
    // set the initial coordinate of the General Path
star.moveTo( xPoints[ 0 ], yPoints[ 0 ] );
    // create the star--this does not draw the star
for ( int count = 1 ; count < xPoints.length; count++ )
    star.lineTo( xPoints[ count ], yPoints[ count ] );
star.closePath(); // close the shape
g2d.translate( 200, 200 ); // translate the origin to (200, 200)
    // rotate around origin and draw stars in random colors
for ( int count = 1 ; count <= 20; count++ )
{
    g2d.rotate( Math.PI / 10.0 ); // rotate coordinate system

    // set random drawing color
g2d.setColor( new Color( random.nextInt( 256 ),
    random.nextInt( 256 ), random.nextInt( 256 ) ) );
    g2d.fill( star ); // draw filled star
} // end for
} // end method paintComponent
} // end class Shapes2JPanel

// Demonstrating a general path.
import java.awt.Color;

```



```

import javax.swing.JFrame;
public class Shapes2 {
    // execute application
    public static void main( String args[] ) {
        // create frame for Shapes2JPanel
        JFrame frame = new JFrame( "Drawing 2D Shapes" );
        frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        Shapes2JPanel shapes2JPanel = new Shapes2JPanel( );
        frame.add( shapes2JPanel ); // add shapes2JPanel to frame
        // set frame background color
        frame.setBackground( Color.WHITE );
        frame.setSize( 400, 400 ); // set frame size
        frame.setVisible( true ); // display frame
    } // end main
} // end class Shapes2

```

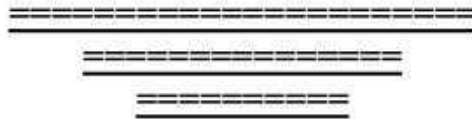
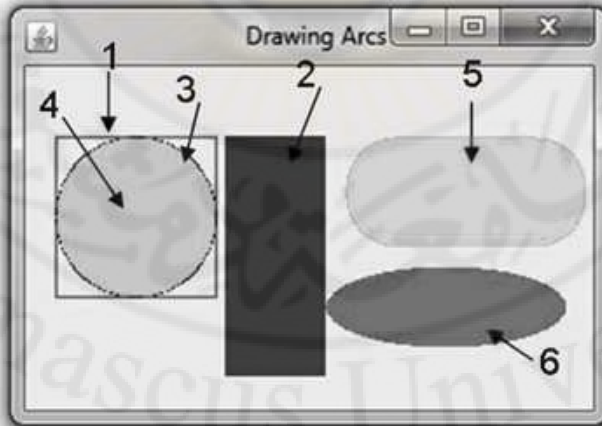
وعند تنفيذ البرنامج سنحصل على الشكل التالي ، وكل عملية تنفيذ سنحصل على الوان للنجوم مختلفة عن الحالة السابقة بشكل عشوائي.



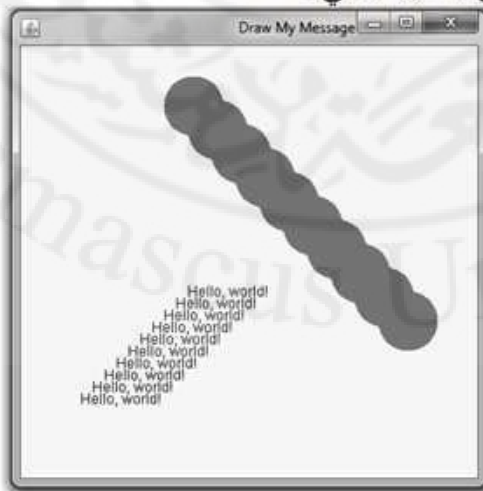
مسائل عامة

- ١- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عضو عام `public void paintComponent( Graphics g)` المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار :
 - رسم مستطيل منبسط لونه `RED` بحيث إحداثيات الزاوية اليسارية العليا هي `(15,35)` وطول وعرض المستطيل `width=80 , height = 80` (رقم 1 في الشكل).
 - رسم مستطيل ممتلئ لونه `BLUE` حيث إحداثيات الزاوية اليسارية العليا هي `(100,35)` وطول وعرض المستطيل `width=50 , height = 120` (رقم 2 في الشكل).
 - رسم قوس مغلق (دائرة) لونه `BLACK` بحيث إحداثيات الزاوية اليسارية العليا هي `(15,35)` وطول وعرض المستطيل `width=80 , height = 80` وزاوية البداية والامتداد `startAngle=0, startAngle=360` (رقم 3 في الشكل).
 - رسم قوس مغلق ممتلئ (دائرة) لونه `ORANGE` بحيث إحداثيات الزاوية اليسارية العليا هي `(15,35)` وطول وعرض المستطيل `width=80 , height = 80` وزاوية البداية والامتداد `startAngle=0, startAngle=360` (رقم 4 في الشكل).

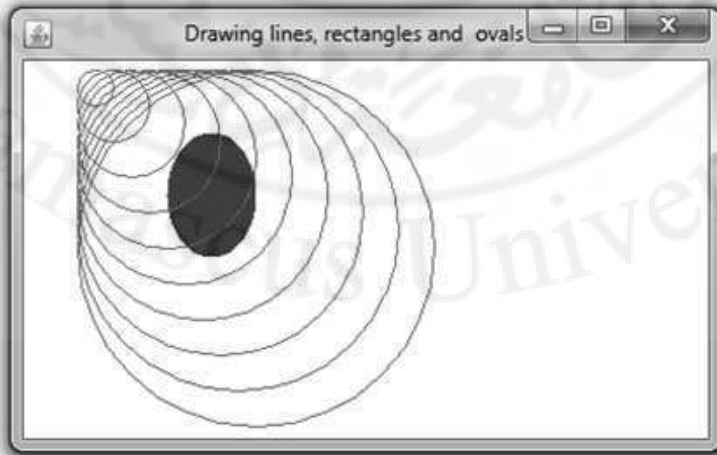
- رسم مستطيل مدور ممثل لونه GREEN بحيث إحداثيات الزاوية اليسارية العليا هي (160,35) وطول وعرض المستطيل $width=120$, $height = 55$ والقطر الأفقي والعمودي لهذا القوس $arcwidth=50$, $arheight=50$ (رقم 5 في الشكل).
- رسم قطع ناقص ممثل لونه RED حيث إحداثيات الزاوية اليسارية العليا هي (150,100) وطول وعرض المستطيل الذي يحيط بالقطع الناقص $width=50$, $height = 120$ (رقم 6 في الشكل).
- أنشئ صفا اسمه TestPanel يحتوي على التابع main بحيث يتم إنشاء إطار (نافذة) من الصف JFrame حجمه (300,210) ، ويتم إنشاء غرض من الصف DrawingPanel بحيث يتم إضافة الأشكال الهندسية أفقة الذكر إلى الإطار .
- يجب أن يكون الخرج على الشكل التالي :



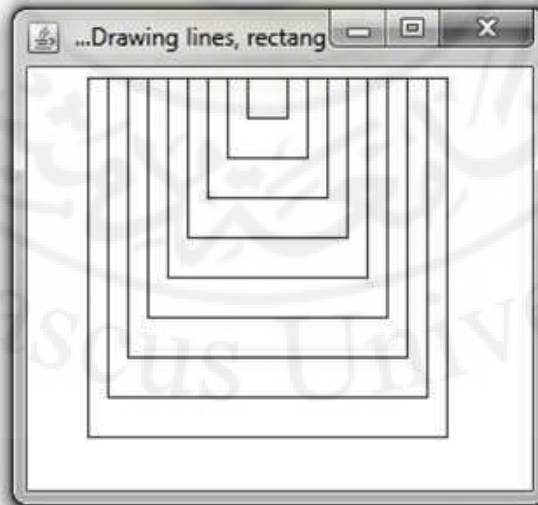
- ٢- - اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عضو عام `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار :
- رسم السلسلة `Hello, world!` عشرة مرات باللون الأزرق وباستخدام حلقة `for` وذلك من خلال تغيير الاحداثيات (x,y) اعتباراً من النقطة $(150,200)$. كما هو مبين في الشكل .
 - رسم قطع ناقص ممثلي لونه `RED` (عشرة مرات) حيث إحداثيات الزاوية اليسارية العليا هي $(5, 100)$ وطول وعرض المستطيل `width=50` , `height = 50` . وذلك من خلال تغيير الاحداثيات (x,y) اعتباراً من النقطة $(5,100)$. كما هو مبين في الشكل .
 - أنشئ صفا اسمه `TestPanel` يحتوي على التابع `main` بحيث يتم إنشاء إطار (نافذة) من الصف `JFrame` حجمه $(210,300)$ ، ويتم إنشاء غرض من الصف `DrawingPanel` بحيث يتم إضافة الأشكال الهندسية أنفة الذكر إلى الإطار .
 - يجب أن يكون الخرج على الشكل التالي :



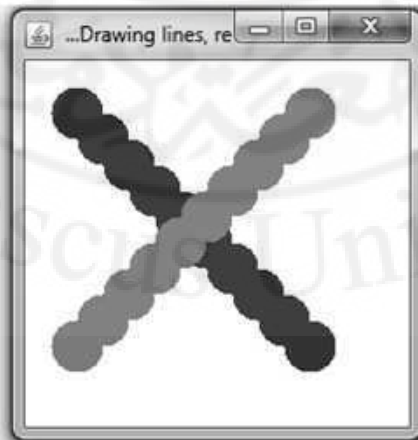
- ٣- اكتب برنامجاً بلغة Java يحتوي على صف اسمه DrawingPanel يرث الصف JPanel يحتوي على تابع عضو عام
- `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي g بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار :
- رسم قطع ناقص منبسط لونه RED (عشرة مرات) حيث إحداثيات الزاوية اليسارية العليا هي (5, 100) وطول وعرض المستطيل `width=50` , `height = 50` . وذلك من خلال تغير الاحداثيات (x,y) اعتباراً من النقطة (100,5). كما هو مبين في الشكل .
 - رسم قطع ناقص ممتلئ لونه BLUE حيث إحداثيات الزاوية اليسارية العليا هي (40, 80) وطول وعرض المستطيل `width=50` , `height = 70` . كما هو مبين في الشكل .
 - أنشئ صفا اسمه TestPanal يحتوي على التابع main بحيث يتم إنشاء إطار (نافذة) من الصف JFrame حجمه (300,210) ، ويتم إنشاء غرض من الصف DrawingPanel بحيث يتم إضافة الأشكال الهندسية أنفة الذكر إلى الإطار .
 - يجب أن يكون الخرج على الشكل التالي :



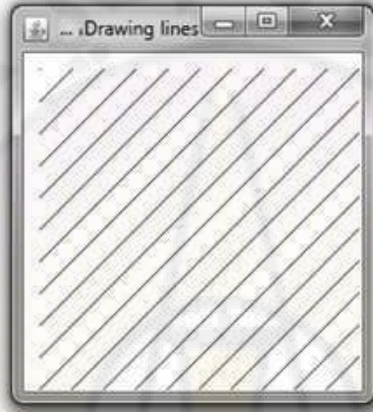
- ٤ - اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عام `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار :
- رسم قطع مستطيل منبسط لونه `BLUE` (عشرة مرات) حيث إحداثيات الزاوية اليسارية العليا هي `(5, 20)` وطول وعرض المستطيل `width=200` , `height = 200` . وذلك من خلال تغيير الاحداثيات `(x,y)` اعتباراً من النقطة `(5,20)` . كما هو مبين في الشكل .
 - أنشئ صفا اسمه `TestPanel` يحتوي على التابع `main` بحيث يتم إنشاء إطار (نافذة) من الصف `JFrame` حجمه `(300,210)` ، ويتم إنشاء غرض من الصف `DrawingPanel` بحيث يتم إضافة الأشكال الهندسية أفقة الذكر إلى الإطار .
 - يجب أن يكون الخرج على الشكل التالي :



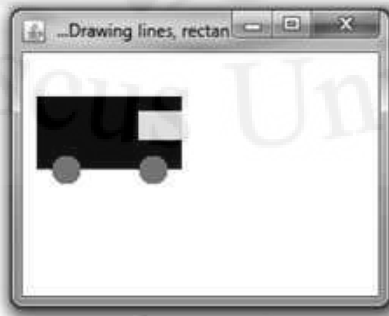
- ٥- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عضو عام `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار :
- رسم قطع ناقص ممثل لونه `BLUE` (عشرة مرات) حيث إحداثيات الزاوية اليسارية العليا هي `(15, 15)` وطول وعرض المستطيل `width=30` , `height = 30` . وذلك من خلال تغيير الاحداثيات `(x,y)` اعتباراً من النقطة `(15,15)` . كما هو مبين في الشكل .
 - رسم قطع ناقص ممثل لونه `MAGENTA` (عشرة مرات) حيث إحداثيات الزاوية اليسارية العليا هي `(15, 15)` وطول وعرض المستطيل `width=30` , `height = 30` . وذلك من خلال تغيير الاحداثيات `(x,y)` اعتباراً من النقطة `(15,15)` . كما هو مبين في الشكل .
 - أنشئ صفاً اسمه `TestPanel` يحتوي على التابع `main` بحيث يتم إنشاء إطار (نافذة) من الصف `JFrame` حجمه `(300,210)` ، ويتم إنشاء غرض من الصف `DrawingPanel` بحيث يتم إضافة الأشكال الهندسية أنفة الذكر إلى الإطار . يجب أن يكون الخرج على الشكل التالي :



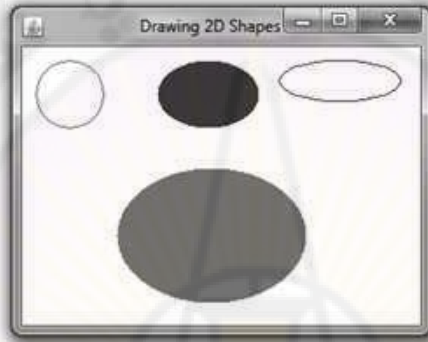
- ٦- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عضو عام `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار .
- رسم خط مستقيم لتهشير الشكل الهندسي كما هو مبين في الشكل :



- ٧- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عضو عام `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار .
- رسم سيارة كما هو مبين في الشكل :



٨- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingPanel` يرث الصف `JPanel` يحتوي على تابع عضو عام `public void paintComponent( Graphics g)` من أجل رسم الأشياء على المحتوى الرسومي `g` بحيث يتم رسم الأشكال الهندسية التالية ضمن إطار كما هو مبين في الشكل التالي :



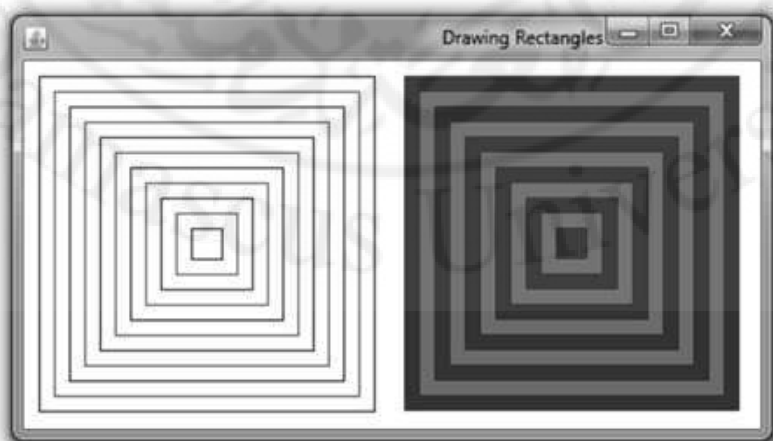
٩- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingString1` يرث الصف `JPanel` بحيث يتم رسم السلسلة التالية إحدى عشرة مرة ضمن إطار باستخدام حلقة `for` . بحيث تتناوب ألوان السلسلة الأحمر ثم الأزرق بحيث يكون لدينا في الخرج ست سلاسل حمراء وخمس سلاسل زرقاء ، ولون الإطار أو اللوحة أصفر ، بحيث يكون الخرج على الشكل التالي .



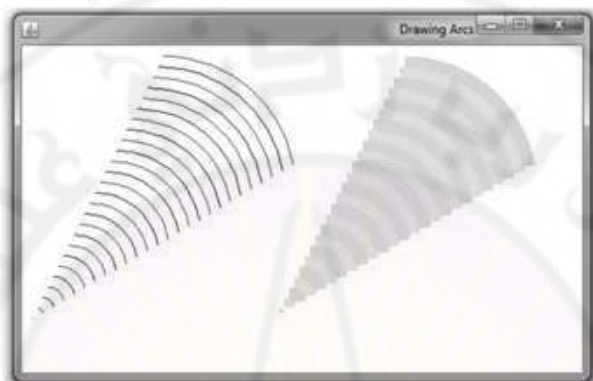
١٠- اكتب برنامجاً بلغة Java يحتوي على صف اسمه `DrawingString2` يرث الصف `JPanel` بحيث يتم رسم السلاسل التالية ضمن إطار باستخدام حلقة `for`: السلسلة `Welcome` عشرة مرات باللون الأزرق، والسلسلة `Hello` ثمان مرات باللون الزهري أما السلسلة `Hi` عشرة مرات باللون الأحمر ، بحيث نحصل على الخرج التالي :



١١- اكتب برنامجاً بلغة Java ليحتوي صف اسمه `DrawingRec2` يرث الصف `JPanel` بحيث يتم رسم الشكل الهندسي التالي ضمن إطار (رسم 10 مستطيلات أو مربعات زرقاء وحمراء بشكل متناوب باستخدام حلقة `for`).



١٢- اكتب برنامجاً بلغة Java يحتوي على صف اسمه DrawingArcs1 يرث الصف JPanel بحيث يتم رسم الشكل الهندسي التالي ضمن إطار (رسم 24 قوس زرقاء وحمراء بشكل متناوب باستخدام حلقة for).



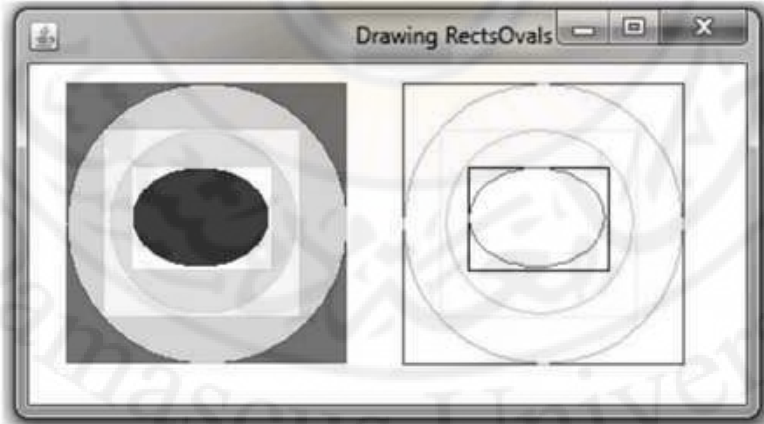
١٣- اكتب برنامجاً بلغة Java يحتوي على صف اسمه DrawingLine1 يرث الصف JPanel بحيث يتم رسم الشكل الهندسي التالي ضمن إطار (رسم 20 خط مستقيم لونه أحمر) باستخدام حلقة for .



١٤- اكتب برنامجاً بلغة Java يحتوي على صف اسمه DrawingLine2 يرث الصف JPanel بحيث يتم رسم الشكل الهندسي التالي ضمن إطار (رسم 20 خط مستقيم أحمر + رسم 20 خط مستقيم أزرق) باستخدام حلقة for .



١٥- اكتب برنامجاً بلغة Java يحتوي صف اسمه DrawingRec3 يرث الصف JPanel بحيث يتم رسم الشكل الهندسي التالي ضمن إطار (رسم مربع خارجي أحمر اللون + دائرة خضراء اللون + مربع داخلي أصفر اللون + دائرة سماوية اللون + مربع صغير داخلي ذات إطار أزرق ومملوء أبيض + دائرة داخلية زرقاء اللون) .



الفصل العاشر

معالجة الأحداث

Events Handler





معالجة الأحداث

Events Handler

١-١٠ - البرمجة المقتادة بالحدث

Event-Driven Programming

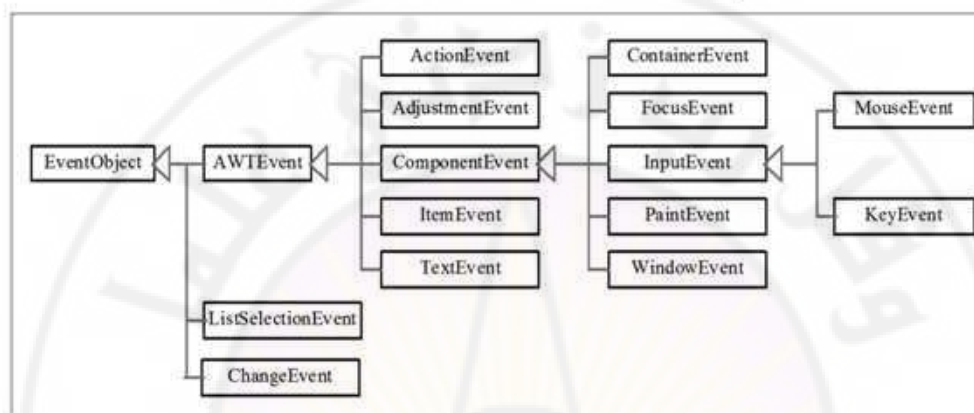
يتم تنفيذ جميع البرامج غير المتعلقة بواجهة المستخدم المرئية وفق ترتيب إجرائي معين ، وإن برامج Java GUI برامج مقتادة بالحدث . الحدث هو أي تفاعل يقوم به المستخدم مع الواجهة المرئية ، كالضغط على زر ، أو اختيار عنصر من قائمة ، أو النقر بزر الفأرة فوق أحد المكونات ، أو النقر على زر من لوحة المفاتيح ... الخ ، أو من خلال نظام التشغيل كالمؤقت ، وأما معالجة الحدث فيقصد بها كتابة نص برمجي يتم تنفيذه تلقائياً عند وقوع الحدث . ويستطيع البرنامج الاستجابة إلى الأحداث أو تجاهلها . ويبين الجدول (١-١٠) بعضاً من الأحداث التي يمكن معالجتها في لغة Java ، وتفاعل المستخدم المولد لكل حدث.

الحدث الناتج	أفعال المستخدم
ActionEvent	الضغط على زر
KeyEvent	ضغط زر من لوحة المفاتيح
MouseEvent	الضغط بالفأرة على JPanel
DocumentEvent	إضافة حرف إلى JTextField
ListSelectionEvent	اختيار عنصر من JList
FocusEvent	انتقال التركيز إلى عنصر
WindowEvent	التفاعل مع النافذة (تكبير ، تصغير ، استعادة)

الجدول (١-١٠) - يبين بعض الأحداث الرئيسية

يدعى العنصر الذي تم توليد الحدث من خلاله بالغرض المصدري Source Object ، وإن الحدث هو عبارة عن غرض ينتمي إلى صف يمثل

الحدث ، والصف الأساسي للصفوف الذي يمثل الأحداث هو :
 java.util.EventObject ، ويبين الشكل (١٠-١) السلسلة الوراثية لصفوف
 الأحداث المستخدمة في هذا الكتاب .



الشكل (١٠-١) - يبين السلسلة الوراثية لصفوف الأحداث

يحتوي غرض الحدث على جميع الخصائص المتعلقة بالحدث ، وتتعامل
 الصفوف المشتقة التابعة للصف EventObject مع أنماط محددة من الأحداث ،
 مثل الأحداث المتعلقة بالزر والأحداث المتعلقة بالنافذة والأحداث المتعلقة
 بالعنصر والأحداث المتعلقة بالحركة والأحداث المتعلقة بضغط المفتاح .

إذا كان العنصر يستطيع توليد حدث ما فإن جميع الصفوف المشتقة التابعة
 لهذا العنصر تستطيع توليد هذا الحدث ، فعلى سبيل المثال تستطيع عناصر
 GUI توليد الأحداث MouseEvent و KeyEvent و ComponentEvent .
 وذلك لأن الصف Component هو الصف الأساسي لجميع عناصر الـ GUI .

١٠-٢- كتابة معالجة الحدث

Writing Event handler

هناك ثلاثة أمور رئيسية تتعلق بأي حدث يقوم به المستخدم على الواجهة

المرئية وهي :

- مصدر الحدث Event Source
 - غرض الحدث Event Object والذي يتضمن معلومات حول الحدث الحاصل .
 - متنصت الحدث Event Listener وهو الغرض الذي سيقوم بالاستجابة على هذا الحدث الحاصل .
- ويحتاج الغرض إلى أمرين لكي يصبح غرضاً متنصتاً على الحدث التابع للغرض المصدري :

١. يجب على الصف المعرف للغرض المتنصت تنفيذ واجهة المتنصت على الحدث المتوافقة مع الحدث ، تقدم Java واجهة تنصت خاصة من أجل كل نمط من أحداث GUI ، وتحمل واجهة التنصت الاسم ActionListener من أجل الحدث Xevent باستثناء الواجهة MousMotionListener مثلاً واجهة التنصت الموافقة للحدث ActionEvent هي ActionListener ، ويبين الجدول (١٠-٢) أنماط الأحداث وواجهات التنصت الموافقة لها والتتابع المعرفة في واجهات التنصت . وتحتوي واجهات التنصت على عدد من التتابع التي تعالج الأحداث وتسمى هذه التتابع بالمعالجات Handlers .

٢. يجب تسجيل الغرض المتنصت بواسطة الغرض المصدري ، وتعتمد تتابع التسجيل على نمط الحدث فمثلاً التابع الخاص بالحدث ActionEvent هو addActionListener ، بشكل عام ، يحمل التابع المتنصت addActionListener من أجل الحدث Xevent ، ومن الممكن للغرض المصدري أن يطلق عدة أنماط من الأحداث ، حيث يحتفظ الغرض المصدري بقائمة من المتنصات وينبه جميع المتنصات المسجلة بواسطة

استدعاء المعالج Handler على الغرض المتتصت لكي يستجيب على الحدث ، كما هو مبين في الشكل (١٠-٣) .

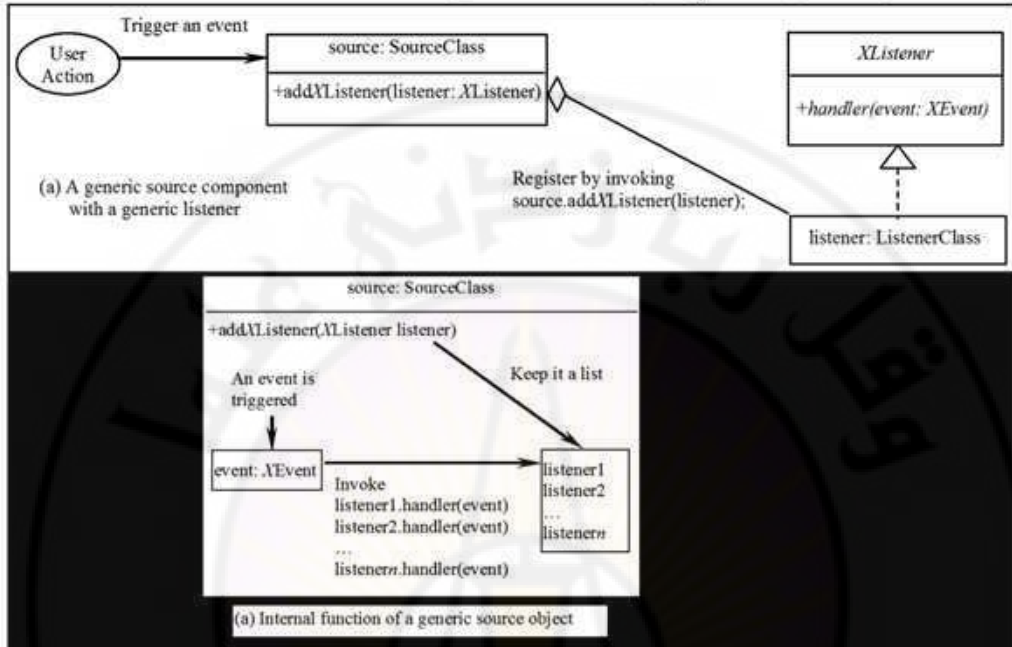
Event Class	Listener Interface	Listener Methods (Handlers)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent e) componentRemoved(ContainerEvent e)
MouseEvent	MouseListener	mousePressed(MouseEvent e) mouseReleased(MouseEvent e) mouseClicked(MouseEvent e) mouseExited(MouseEvent e) mouseEntered(MouseEvent e)

الجدول (١٠-٢) - يبين أنماط الأحداث وواجهات التنصت الموافقة

مثال يوضح كيفية كتابة معالج الحدث ، فعلى سبيل المثال انقر على زر هو غرض من النوع JButton ، علماً أن كتابة أي معالج لأي حدث آخر ستتم بالطريقة نفسها .

- حتى نكتب معالج حدث للزر يجب أولاً أن نعرف غرض متتصت Listener على هذا الزر .
- ثم نخبر الزر أن هذا المتتصت يتتصت عليه .
- وبعدها عند وقوع الحدث على الزر سيتم إرسال رسالة إلى المتتصت ،

وهذه الرسالة هي غرض حدث Event Object .



الشكل (١٠-٣) - يبين تسجيل الغرض المنتصت بواسطة الغرض المصدري

لتوضيح هذه العملية ، سنقوم ببناء واجهة بسيطة مكونة من زر واحد وحقل نص ولافتة (لصاقة)، بحيث عندما نضغط على الزر يتم إظهار النص المُدخل في خانة النص ضمن صندوق رسالة صغير كما هو موضح في الشكل (١٠-٤) .
ويبين المثال (10-1) معالج الحدث .



الشكل (١٠-٤) - صندوق رسالة يظهر النص المدخل

مثال (1-10)

```

1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class ClickAction {
5. public static void main(String args[ ] ) {
6. JFrame wind = new JFrame("Log in");
7. wind.setLayout(new FlowLayout(FlowLayout.LEFT));
8. JButton clickMe = new JButton("Click Here");
9. final JTextField name = new JTextField();
10. JLabel nameLbl = new JLabel("Enter your name: ");
11. wind.add(nameLbl); wind.add(name); wind.add(clickMe);
12. name.setPreferredSize(new Dimension(141,21));
13. wind.pack( ); wind.setVisible(true);
14. wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15. clickMe.addActionListener(new ActionListener( ) {
16. public void actionPerformed(ActionEvent e) {
17. JOptionPane.showMessageDialog(null,
                                "Welcome"+name.getText( ) );
18. }
19. });
20. } //End main
21. } //End class

```

- يقوم السطر 3 باستيراد الحزمة `java.awt.event.*` التي تتضمن الواجهة `ActionListener` التي سننشئ منها غرض متتصت فيما بعد .
- استخدمنا في السطر 12 التابع `setPreferredSize( )` من أجل ضبط حقل النص , فعلى الرغم من استخدام مدير التنسيق `FlowLayout` إلا أن حقل النص سيظهر صغير جداً لأن حجمه يتعلق بحجم النص الذي

- يحتويه ، ويمكن حل المشكلة كما يلي :
- تغيير مدير التنسيق إلى مدير آخر مناسب
 - وضع قيمة افتراضية لحقل النص
 - استخدام التابع `setPreferredSize` ، وهذا التابع هو الذي تم استخدامه في البرنامج .
- في السطر 15 استخدم مفهوم الصف المجهول لإنشاء غرض متتصت من الواجهة `ActionListener` ، هذه الواجهة تتضمن تابعاً واحداً فقط هو `actionPerformed` الذي يأخذ وسيطاً من الصف `ActionEvent` وهو يمثل معلومات حول الحدث الحاصل ، وهذا التابع تم تحقيقه في السطر 16 .
 - بعد إنشاء غرض متتصت يتم تمريره إلى التابع `addActionListener` التابع للزر ، وبذلك نكون قد ربطنا ما بين المتتصت والزر الذي يتتصت عليه .
 - لقد استخدمنا التابع `showMessageDialog` في السطر 17 التابع للصف `JOptionPane` من أجل إظهار صندوق رسالة المستخدم . حيث يأخذ هذا التابع عدة وسطاء وهي محملة بشكل زائد ، فالوسيط الأول يمثل النافذة التي سيظهر فيها صندوق الرسالة ، فإذا كان `null` يعني أن صندوق الرسالة سيظهر في وسط الشاشة ، أما الوسيط الثاني فهو الرسالة التي ستظهر للمستخدم ، والوسيط الثالث (إن وجد) يمثل النص الذي سيظهر على شريط العنوان لصندوق الرسالة ، بينما الوسيط الرابع (إن وجد) فهو يمثل أحد الثوابت التالية، مبين في الجدول (١٠-٣) .

Message dialog type	Icon	Description
ERROR_MESSAGE		A dialog that indicates an error to the user.
INFORMATION_MESSAGE		A dialog with an informational message to the user.
WARNING_MESSAGE		A dialog warning the user of a potential problem.
QUESTION_MESSAGE		A dialog that poses a question to the user. This dialog normally requires a response, such as clicking a Yes or a No button.
PLAIN_MESSAGE	no icon	A dialog that contains a message, but no icon.

الجدول (١٠-٣) - يبين ثوابت الوسيط الربع في التابع showMessageBox

١٠-٣- الصف الداخلي و الصف المجهول

Inner class and Anonymous class

لقد استخدمنا في المثال (10-1) مفهوم الصف المجهول Anonymous class وذلك في السطر 15 من البرنامج ، تم إنشاء غرض من صف يحقق الواجهة ActionListener وحققنا التابع actionPerformed مباشرة دون أن نعرّف صف يحقق هذه الواجهة بالطريقة التقليدية . لقد كان من الممكن بدلاً من ذلك تعريف صف داخلي جديد يحقق الواجهة ActionListener ، كما هو موضح في المثال (10-2) .

إن الصف الداخلي يمكن أن يكون له أي محدد وصول ، public ، private ، protected ، كما يمكن أن نصرّح عن الصف الداخلي على أنه مجرد أو final أو static . في الحقيقة لا يمكن الوصول إلى الصف الداخلي إلا من خلال الصف الحاوي له ، أي لا يمكن الوصول إلى أغراض الصف

الداخلي إلا من أغراض الصف الحاوي له . إن الصف الداخلي يمكنه الوصول إلى جميع أعضاء الصف الحاوي ، بما فيها الأعضاء ذات المحدد `private` ، وهذا أمر منطقي لأن العناصر ذات المحدد `private` غير مرئية خارج الصف . بإمكاننا أن نعرف صفّاً داخلياً داخل تابع من توابع الصف ، ولكن في هذه الحالة الصف الداخلي لا يستطيع الوصول إلى الأعضاء ذات المحدد `final` داخل ذلك التابع ، وكذلك أعضاء الصف الحاوي له . إذاً الصف المجهول هو صف داخلي ليس له اسم .

مثال (2-10)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Test {
    JFrame wind = new JFrame("Log in");
    JButton clickMe = new JButton("Click Here");
    JTextField name = new JTextField( );
    JLabel nameLbl = new JLabel("Enter your name: ");
    public Test( ) {
        wind.setLayout(new FlowLayout(FlowLayout.LEFT));
        wind.add(nameLbl); wind.add(name); wind.add(clickMe);
        name.setPreferredSize(new Dimension(141,21));
        wind.pack(); wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ActionListener listener = new ActionListener();
        clickMe.addActionListener(listener);
    }
    // Start of inner class
    class ActionListen implements ActionListener {
        public void actionPerformed(ActionEvent e ) {
            JOptionPane.showMessageDialog(null, "Welcome "
```

```

+ name.getText( ) );
}
} // end ActionListener class
// End of inner class

public static void main(String args[ ])
{
    Test win = new Test( );
} //end main
} //end test class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

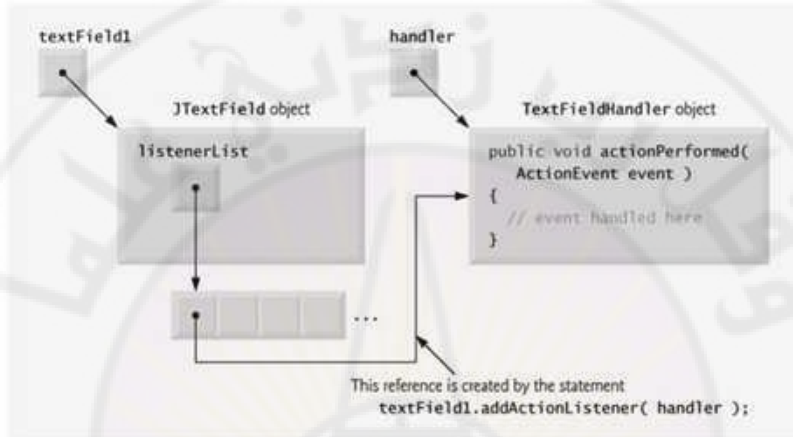


١٠-٤- كتابة معالج الحدث للعنصر JTextField

نوضح في المثال (10-3) كيف يمكننا كتابة معالج حدث لخانة النص JTextField ، حيث يقوم المستخدم بإدخال اسمه في حقل النص ، ثم يضغط Enter ، وبعدها سيقوم معالج الحدث بالحصول على الاسم المدخل وعرضه في مربع الرسالة . يوضح الشكل (١٠-٥) معالجة الحدث JTextField . ونلاحظ أننا استخدمنا في السطر 1 التابع (getActionCommand) التابع التابع لغرض الحدث الذي تم إرساله إلى المتنصت ، حيث يقوم هذا التابع بإعادة النص الذي تم إدخاله من قبل المستخدم في حقل النص وذلك عند ضغط المفتاح Enter .

ملاحظة :

يمكن الاستفادة من التابع (`getSource()`) التابع للصف `ActionEvent` للحصول على اسم الغرض الذي ولد الحدث .



الشكل (١٠-٥) - يوضح معالجة الحدث `JTextField`

مثال (10-3)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Test2 {
    JFrame wind = new JFrame("Log in");
    JTextField name = new JTextField( );
    JLabel nameLbl = new JLabel("Enter your name: ");

    public Test2( ) {
        wind.setLayout(new FlowLayout(FlowLayout.LEFT));
        wind.add(nameLbl); wind.add(name);
        name.setPreferredSize(new Dimension(141,21));
        wind.pack( ); wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        name.addActionListener(new ActionListener( ) {
```



```

public void actionPerformed( ActionEvent e) {
    JOptionPane.showMessageDialog(null,
                                e.getActionCommand() ); //1
}
});
} //public Test2
public static void main(String args[ ])
{
    Test2 win = new Test2();
} //end main
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



١٠-٥- كتابة معالج الحدث للعنصر JButton

الزر عبارة عن عنصر يقترح حدث الفعل (trigger a specific action) عندما يتم النقر عليه ، يتم تمثيل الزر في الإصدار swing بالصف JButton .

استخدمنا فيما سبق أزراراً لا تؤدي أي فعل ، وأن النقر على زر هو حدث ، وهذا الحدث هو غرض يتم إرساله إلى تابع التتبع ، لذا علينا أن ننشئ هذه

التوابع . في لغة جافا يتم استخدام عدة أنواع من الأزرار ، بما في ذلك أزرار الأوامر وهي : check boxes, toggle buttons and radio buttons ، ويبين الشكل (١٠ - ٥) الوراثة الهرمية للأزرار في الحزمة swing .

يملك الصف JButton بالإضافة للبانى الافتراضي البوانى التالية :

- public JButton (**String** text) ;

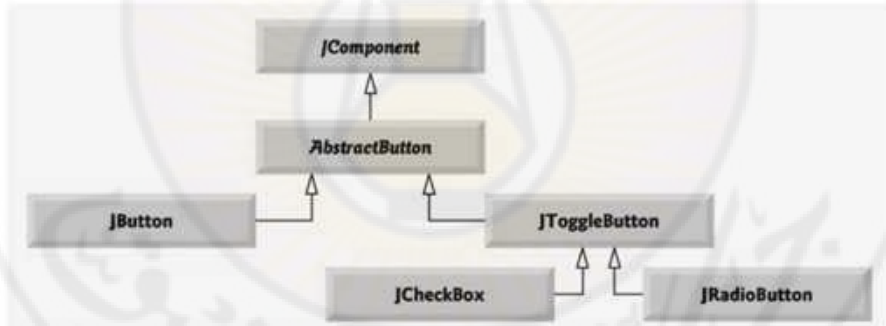
يصنع زرأ يحمل اسم المحدد للوسيط text

- public JButton (**Icon** icon) ;

يصنع زرأ مع الرمز icon المحدد في الوسيط

- public JButton (**String** text , **Icon** icon) ;

يصنع زرأ مع النص والرمز المحددان في الوطاء .



الشكل (١٠ - ٥) - يبين الوراثة الهرمية للأزرار في الحزمة swing

تستطيع الأزرار توليد عدة أنماط من الأحداث ، لكن غالباً ما نحتاج إلى الاستجابة للحدث(ActionEvent) ، ويجب علينا تنفيذ التابع actionPerformed التابع للواجهة ActionListener لكي تجعل الزر يستجيب للحدث(ActionEvent) . ويوضح في المثال (10-4) كيف يمكن كتابة معالج حدث لإنشاء زر، حيث يقوم المستخدم بالنقر على الزر ، فإن أزرار

الأوامر تُنشئ بواسطة الصف JButton ، حيث تم تسمية الزر الأول بـ plainJButton وعند النقر عليه يُعطي رسالة *You pressed: plainJButton* ، أما الزر الثاني هو fancyJButton وعند النقر عليه يُعطي رسالة *You pressed : fancyJButton*

مثال (4-10)

// Creating JButtons.

```
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;

public class ButtonFrame extends JFrame {
    private JButton plainJButton; // button with just text
    private JButton fancyJButton; // button with icons
    // ButtonFrame adds JButtons to JFrame
    public ButtonFrame( ) {
        super( "Testing Buttons" );
        setLayout( new FlowLayout() ); // set frame layout
        plainJButton = new JButton( "Plain Button" ); // button with text
        add( plainJButton ); // add plainJButton to JFrame
        fancyJButton = new JButton( "Fancy Button" ); // set image
        add( fancyJButton ); // add fancyJButton to JFrame
        // create new ButtonHandler for button event handling
        ButtonHandler handler = new ButtonHandler();
        fancyJButton.addActionListener( handler );
        plainJButton.addActionListener( handler );
```

```

    } // end ButtonFrame constructor
    // inner class for button event handling
private class ButtonHandler implements ActionListener {
    // handle button event
    public void actionPerformed( ActionEvent event ) {
        JOptionPane.showMessageDialog( ButtonFrame.this,
        String.format(
            "You pressed: %s", event.getActionCommand( ) ) );
    } // end method actionPerformed
    } // end private inner class ButtonHandler
} // end class ButtonFrame
//-----
// Testing ButtonFrame.
import javax.swing.JFrame;
public class ButtonTest {
public static void main( String args[ ] )
{
    // create ButtonFrame
    ButtonFrame buttonFrame = new ButtonFrame( );
    buttonFrame.setDefaultCloseOperation(
        JFrame.EXIT_ON_CLOSE );
    buttonFrame.setSize( 275, 110 ); // set frame size
    buttonFrame.setVisible( true ); // display frame
} // end main
} // end class ButtonTest

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



عند النقر على الزر plainJButton تظهر الرسالة التالية :



وعند النقر على الزر fancyJButton تظهر الرسالة التالية :



١٠-٦- معالج حدث لمكونات الاختبار

١- صناديق الاختبار JCheckBox

يتيح صندوق الاختبار للمستخدم إمكانية الانتقال بين حالتي التفعيل on وإلغاء التفعيل off . يملك الصف JCheckBox عدداً من البواني نذكر منها:

- public JCheckBox () ;
يصنع صندوق اختبار فارغاً وغير فعال (off)
- public JCheckBox (String text) ;
يصنع صندوق اختبار غير فعال (off) مع نص .
- public JCheckBox (Icon icon) ;
يصنع صندوق اختبار غير فعال (off) مع icon
- public JCheckBox (String text , boolean selected) ;

يصنع صندوق اختبار مع نص ويكون حسب **boolean** فعالاً أو غير فعال .
 يستطيع العنصر `JCheckBox` توليد الحدثين `ActionEvent` و `ItemEvent` بالإضافة إلى العديد من الأحداث الأخرى . يبين المثال (5-10) كيفية استخدام غرضين من الصف `JCheckBox` لتحديد نمط الخط المطلوب من النص المعروض في الـ `JTextField` إما مائل `Italic` أو عريض `Bold` أو الاثنين معاً.

مثال (5-10)

```
// Creating JCheckBox buttons.
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JCheckBox;

public class CheckBoxFrame extends JFrame {
    private JTextField textField; // displays text in changing fonts
    private JCheckBox boldJCheckBox; // to select/deselect bold
    private JCheckBox italicJCheckBox; // to select/deselect italic
    // CheckBoxFrame constructor adds JCheckBoxes to JFrame
    public CheckBoxFrame() {
        super( "JCheckBox Test" );
        setLayout( new FlowLayout( ) ); // set frame layout
        // set up JTextField and set its font
        textField = new JTextField( "Watch the font style change", 20 );
        textField.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
        add( textField ); // add textField to JFrame
        boldJCheckBox = new JCheckBox( "Bold" ); // create bold checkbox
```



```

italicJCheckBox = new JCheckBox( "Italic" ); // create italic
    add( boldJCheckBox ); // add bold checkbox to JFrame
    add( italicJCheckBox ); // add italic checkbox to JFrame
    // register listeners for JCheckBoxes
CheckBoxHandler handler = new CheckBoxHandler( );
boldJCheckBox.addItemListener( handler );
italicJCheckBox.addItemListener( handler );
} // end CheckBoxFrame constructor
    // private inner class for ItemListener event handling
private class CheckBoxHandler implements ItemListener {
private int valBold = Font.PLAIN; // controls bold font style
private int valItalic = Font.PLAIN; // controls italic font style

    // respond to checkbox events
public void itemStateChanged( ItemEvent event ) {
    // process bold checkbox events
    if ( event.getSource( ) == boldJCheckBox )
        valBold = boldJCheckBox.isSelected( ) ? Font.BOLD :
Font.PLAIN;

    // process italic checkbox events
    if ( event.getSource( ) == italicJCheckBox )
        valItalic = italicJCheckBox.isSelected( ) ? Font.ITALIC :
Font.PLAIN;

    // set text field font
    textField.setFont(
        new Font( "Serif", valBold + valItalic, 14 ) );
} // end method itemStateChanged
} // end private inner class CheckBoxHandler
} // end class CheckBoxFrame
//-----

```

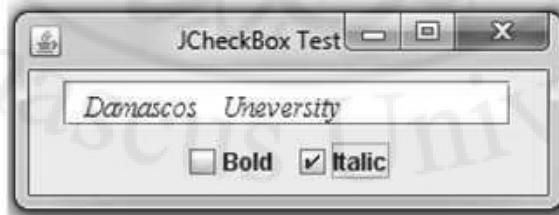
```
// Testing CheckBoxFrame.
import javax.swing.JFrame;

public class CheckBoxTest {
    public static void main( String args[ ] ) {
        CheckBoxFrame checkBoxFrame = new CheckBoxFrame( );
        checkBoxFrame.setDefaultCloseOperation(
JFrame.EXIT_ON_CLOSE );
        checkBoxFrame.setSize( 275, 100 ); // set frame size
        checkBoxFrame.setVisible( true ); // display frame
    } // end main
} // end class CheckBoxTest
```

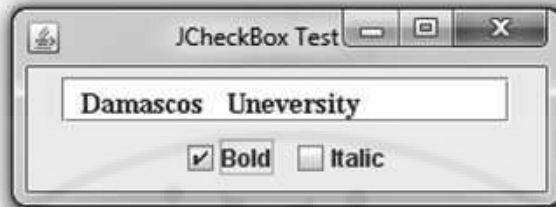
وعند تنفيذ البرنامج سنحصل على الخرج التالي:



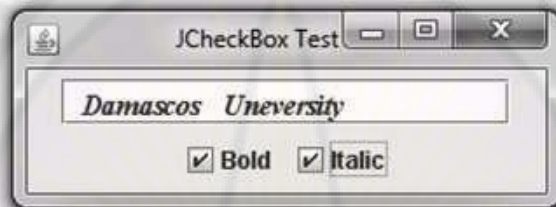
عند اختيار Italic سيظهر النص كما يلي :



عند اختيار Bold سيظهر النص كما يلي :



عند اختيار Bold و Italic سيظهر النص كما يلي :



٢- أزرار الخيار JRadioButton

تتيح أزرار الخيار JRadioButton إمكانية اختيار عنصر وحيد من مجموعة خيارات ، تتشابه أزرار الخيار مع صناديق الخيار من الناحية الشكلية ، إلا أنه يتم عرض صناديق الاختيار على شكل مربعات وتكون المربعات إما فعالة checked أو غير فعالة unchecked ، بينما تعرض أزرار الخيار على هيئة دائرة إما ممثلة (فعالة) أو فارغة (غير فعالة) . تتشابه بواني العنصر JRadioButton مع بواني العنصر JCheckBox نذكر منها :

```
public JRadioButton ( ) ;
```

يصنع زر خيار فارغاً وغير فعال (off)

```
public JRadioButton ( String text ) ;
```

يصنع زر خيار غير فعال (off) مع نص .

```
public JRadioButton ( Icon icon ) ;
```

يصنع زر خيار غير فعال (off) مع icon

`public JRadioButton ( String text , boolean selected ) ;`
يصنع زر خيار مع نص ويكون حسب **boolean** فعالاً أو غير فعال .
يستطيع الصف `JRadioButton` توليد الحدثين `ActionEvent` و `ItemEvent` بالإضافة إلى العديد من الأحداث الأخرى. وبين المثال (6-10)
إنشاء أزرار خيار لتحديد نمط الخط المطلوب من النص المعروض في الـ `JTextField` إما مائل *Italic* أو عريض **Bold** أو الأثنين معاً.

مثال (6-10)

// Creating radio buttons using ButtonGroup and JRadioButton.

```
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JRadioButton;
import javax.swing.ButtonGroup;
```

```
public class RadioButtonFrame extends JFrame {
    private JTextField textField; // used to display font changes
    private Font plainFont; // font for plain text
    private Font boldFont; // font for bold text
    private Font italicFont; // font for italic text
    private Font boldItalicFont; // font for bold and italic text
    private JRadioButton plainJRadioButton; // selects plain text
    private JRadioButton boldJRadioButton; // selects bold text
    private JRadioButton italicJRadioButton; // selects italic text
    private JRadioButton boldItalicJRadioButton; // bold and italic
    private ButtonGroup radioGroup;
```


// RadioButtonFrame constructor adds JRadioButtons to JFrame

```
public RadioButtonFrame() {  
    super( "RadioButton Test" );  
    setLayout( new FlowLayout( ) ); // set frame layout  
  
    textField = new JTextField( "Watch the font style change", 25 );  
    add( textField ); // add textField to JFrame  
        // create radio buttons  
    plainJRadioButton = new JRadioButton( "Plain", true );  
    boldJRadioButton = new JRadioButton( "Bold", false );  
    italicJRadioButton = new JRadioButton( "Italic", false );  
    boldItalicJRadioButton = new JRadioButton( "Bold/Italic" false );  
    add( plainJRadioButton ); // add plain button to JFrame  
    add( boldJRadioButton ); // add bold button to JFrame  
    add( italicJRadioButton ); // add italic button to JFrame  
    add( boldItalicJRadioButton ); // add bold and italic button  
        // create logical relationship between JRadioButtons  
    radioGroup = new ButtonGroup( ); // create ButtonGroup  
    radioGroup.add( plainJRadioButton ); // add plain to group  
    radioGroup.add( boldJRadioButton ); // add bold to group  
    radioGroup.add( italicJRadioButton ); // add italic to group  
    radioGroup.add( boldItalicJRadioButton ); // add bold and italic  
        // create font objects  
    plainFont = new Font( "Serif", Font.PLAIN, 14 );  
    boldFont = new Font( "Serif", Font.BOLD, 14 );  
    italicFont = new Font( "Serif", Font.ITALIC, 14 );  
    boldItalicFont = new Font( "Serif", Font.BOLD +  
                                                Font.ITALIC, 14 );  
    textField.setFont( plainFont ); // set initial font to plain  
        // register events for JRadioButtons  
    plainJRadioButton.addItemListener(
```



```

        new RadioButtonHandler( plainFont ) );
boldJRadioButton.addItemListener(
        new RadioButtonHandler( boldFont ) );
italicJRadioButton.addItemListener(
        new RadioButtonHandler( italicFont ) );
boldItalicJRadioButton.addItemListener(
        new RadioButtonHandler( boldItalicFont ) );
} // end RadioButtonFrame constructor
// private inner class to handle radio button events
private class RadioButtonHandler implements ItemListener{
    private Font font; // font associated with this listener
    public RadioButtonHandler( Font f ) {
        font = f; // set the font of this listener
    } // end constructor RadioButtonHandler
    // handle radio button events
    public void itemStateChanged( ItemEvent event ) {
        textField.setFont( font ); // set font of textField
    } // end method itemStateChanged
} // end private inner class RadioButtonHandler
} // end class RadioButtonFrame
//-----
// Testing RadioButtonFrame.
import javax.swing.JFrame;
public class RadioButtonTest {
    public static void main( String args[ ] ) {
        RadioButtonFrame radioButtonFrame = new
RadioButtonFrame( );
        radioButtonFrame.setDefaultCloseOperation(
JFrame.EXIT_ON_CLOSE );
        radioButtonFrame.setSize( 300, 100 ); // set frame size
        radioButtonFrame.setVisible( true ); // display frame

```

```

} // end main
} // end class RadioButtonTest

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



عند اختيار الزر Bold سيظهر النص كما يلي :



عند اختيار الزر Italic سيظهر النص كما يلي :



عند اختيار الزر Bold/Italic سيظهر النص كما يلي :



٣- صناديق الخيارات المتاحة JComboBox

يحتوي صندوق الخيارات المتاحة على قائمة من العناصر التي يستطيع المستخدم الاختيار منها ما يؤدي إلى تحديد مجال الخيارات المتاحة للمستخدم وبالتالي نتخلص من عملية فحص معطيات الدخل المتعبة . لصنع غرض من

الصف JComboBox نستخدم الباني الافتراضي أو نستخدم الباني التالي الذي يحتوي على مجموعة من السلاسل المحرفية :

```
public JComboBox ( object [ ] stringtext ) ;
```

الوسيط stringtext هو مصفوفة محرفية من النوع String .

يمكن استخدام التوابع التالية من أجل التعامل مع صندوق الخيارات المتاحة :

```
public void addItem ( object item ) ;
```

إضافة الغرض الممرر إلى القائمة .

```
public object getItemAt ( int index ) ;
```

إعادة العنصر عند الدليل الممرر .

```
public void removeItem ( object anobject ) ;
```

حذف غرض محدد من صندوق الخيارات المتاحة.

```
public void removeAllItem ( ) ;
```

حذف جميع العناصر من صندوق الخيارات المتاحة.

للحصول على المعطيات من العنصر JComboBox يمكن استخدام التابع

getSelectedItem() من أجل استعادة العنصر المحدد حالياً أو استخدام التابع

e.getItem() من أجل الحصول على العنصر من المعالج

. itemStateChanged(ItemEvent e)

يستطيع الصف JComboBox توليد الحدثين(ActionEvent و

ItemEvent بالإضافة إلى العديد من الأحداث الأخرى . كلما تم اختيار عنصر

جديد يولد الكائن JComboBox الحدث ItemEvent مرتين : الأولى بسبب

إزالة الاختيار من العنصر السابق الذي كان محدداً والثانية بسبب اختيار العنصر

الحالي الذي سنحدده . يولد الكائن JComboBox الحدث(ActionEvent بعد

توليد الحدث ItemEvent ، نحتاج من أجل الاستجابة للحدث ItemEvent إلى

تنفيذ المعالج itemStateChanged(ItemEvent e) وذلك بهدف معالجة

الخيار . يظهر المثال (10-7) شكل خاص للصف الداخلي الذي يُستخدم في

كثير من الأحيان لمعالجة الحدث ، ويُظهر قائمة من أربعة أسماء ملفات للصور
يستطيع المستخدم اختيار صورة واحدة لعرضها .

مثال (7-10)

```
// Using a JComboBox to select an image to display.
import java.awt.FlowLayout;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.Icon;
import javax.swing.ImageIcon;

public class ComboBoxFrame extends JFrame
{
    // combobox to hold names of icons
    private JComboBox imagesJComboBox
    private JLabel label; // label to display selected icon
    private String names[ ] =
        { "bug1.gif", "bug2.gif", "travelbug.gif", "buganim.gif" };
    private Icon icons[ ] = {
        new ImageIcon( getClass( ).getResource( names[ 0 ] ) ),
        new ImageIcon( getClass( ).getResource( names[ 1 ] ) ),
        new ImageIcon( getClass( ).getResource( names[ 2 ] ) ),
        new ImageIcon( getClass( ).getResource( names[ 3 ] ) ) };
    // ComboBoxFrame constructor adds JComboBox to JFrame
    public ComboBoxFrame( ) {
        super( "Testing JComboBox" );
        setLayout( new FlowLayout( ) ); // set frame layout
        // set up JComboBox
    }
}
```



```

imagesJComboBox = new JComboBox( names );
    // display three rows
imagesJComboBox.setMaximumRowCount( 3 );

imagesJComboBox.addItemListener(
    new ItemListener( )    // anonymous inner class
    {
        // handle JComboBox event
        public void itemStateChanged( ItemEvent event ) {
            // determine whether check box selected
            if ( event.getStateChange() == ItemEvent.SELECTED )
                label.setIcon( icons[
                    imagesJComboBox.getSelectedIndex() ] );
        } // end method itemStateChanged
    } // end anonymous inner class
); // end call to addItemListener
add( imagesJComboBox ); // add combobox to JFrame
label = new JLabel( icons[ 0 ] ); // display first icon
add( label ); // add label to JFrame
} // end ComboBoxFrame constructor
} // end class ComboBoxFrame
//=====
// Testing ComboBoxFrame.
import javax.swing.JFrame;
public class ComboBoxTest {
public static void main( String args[] ) {
    ComboBoxFrame comboBoxFrame = new ComboBoxFrame( );
    comboBoxFrame.setDefaultCloseOperation(
        JFrame.EXIT_ON_CLOSE );
    comboBoxFrame.setSize( 350, 150 ); // set frame size
    comboBoxFrame.setVisible( true ); // display frame

```

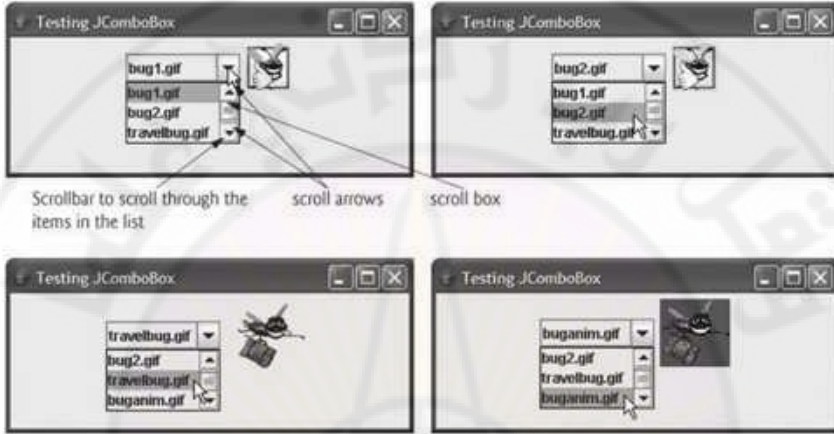


```

} // end main
} // end class ComboBoxTest

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٤ - اللوائح JList

إن وظيفة اللوائح الأساسية إنجاز المهام نفسها التي يقوم بها صندوق الخيارات المتاحة JComboBox إلا أنها تتميز ببعض الصفات الأخرى مثل تمكين المستخدم من اختيار قيمة واحدة أو عدة قيم ، إن العنصر JList التابع للحزمة Swing متعدد الاستخدامات ، ويملك العديد من الصفات المتقدمة. نستخدم الباني التالي من أجل صنع لائحة من السلاسل المحرفية :

```
public JList ( Object [ ] stringItems )
```

حيث الوسيط stringItems هو مصفوفة محرفية من نوع String .
يحتوي العنصر JList على الخصائص التالية :

- selectedIndex : يحتوي على قيمة صحيحة تدل على دليل العنصر الذي تم اختياره من اللائحة .
- selectedIndex : يحتوي على مصفوفة من القيم الصحيحة int تمثل دلائل العناصر التي تم اختيارها من اللائحة .

- selectedValue : يمثل القيمة الأولى التي تم اختيارها من اللائحة .
- selectedMode : تأخذ إحدى القيم التالية :
 - SINGLE_SELECTION : تدل على إمكانية اختيار عنصر وحيد .
 - SINGLE_INTERVAL_SELECTION : تدل على إمكانية اختيار عدة عناصر بشرط أن تكون هذه العناصر متجاورة ومتلاصقة ز
 - MULTIPLE_INTERVAL_SELECTION : تدل على إمكانية اختيار عدة عناصر وبدون أي شروط .
- يولد العنصر JList الحدث javax.swing.event.ListSelectionEvent من أجل تنبيه الأغراض المتتصته على الخيارات ، ويجب على المتتصات تنفيذ المعالج valueChanged من أجل معالجة الحدث . يوضح المثال (8-10) كيفية إنشاء لائحة من الألوان ، وعند اختيار لون ما ، يتم تلوين الأطار بهذا اللون .

مثال (8-10)

// Selecting colors from a JList.

```
import java.awt.FlowLayout;
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.ListSelectionModel;

public class ListFrame extends JFrame {
    private JList colorJList; // list to display colors
    private final String colorNames[] = { "Black", "Blue",
        "Cyan", "Dark Gray", "Gray", "Green", "Light Gray",
        "Magenta", "Orange", "Pink", "Red", "White", "Yellow" };
    private final Color colors[] = { Color.BLACK, Color.BLUE,
```

```

Color.CYAN, Color.DARK_GRAY, Color.GRAY,
Color.GREEN , Color.LIGHT_GRAY, Color.MAGENTA,
Color.ORANGE, Color.PINK, Color.RED, Color.WHITE,
Color.YELLOW  };

```

```

// ListFrame constructor add JScrollPane containing , JList to JFrame
public ListFrame( ) {
    super( "List Test" );
    setLayout( new FlowLayout( ) ); // set frame layout
    colorJList = new JList( colorNames ); // create with colorNames
    colorJList.setVisibleRowCount( 5 ); // display five rows at once
    // do not allow multiple selections
    colorJList.setSelectionMode(
        ListSelectionModel.SINGLE_SELECTION );
    // add a JScrollPane containing JList to frame
    add( new JScrollPane( colorJList ) );
    colorJList.addListSelectionListener(
        new ListSelectionListener( ) // anonymous inner class
        {
            // handle list selection events
            public void valueChanged( ListSelectionEvent event ) {
                getContentPane( ).setBackground(
                    colors [ colorJList.getSelectedIndex( ) ] );
            } // end method valueChanged
        } // end anonymous inner class
    ); // end call to addListSelectionListener
} // end ListFrame constructor
} // end class ListFrame

//=====

```

```

// Selecting colors from a JList.

```

```

import javax.swing.JFrame;

```

```

public class ListTest {
    public static void main( String args[ ] ) {
        ListFrame listFrame = new ListFrame( ); // create ListFrame
        listFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        listFrame.setSize( 350, 150 ); // set frame size
        listFrame.setVisible( true ); // display frame
    } // end main
} // end class ListTest

```

عند اختيار Cyan سيظهر الخرج كما يلي :



عند اختيار Blue سيظهر الخرج كما يلي :



يوضح المثال (9-10) التالي كيفية كتابة معالج الحدث لمكونات الاختبار، وهي :

JList , JComboBox , JRadioButton, JCheckBox

في هذا المثال ، يقوم المستخدم بتحديد معلومات معينة مثل لقبه : Mr, Mrs, Miss ، وجنسه (M, F) ، واللغات التي يعرفها ، والبلد الذي يعيش فيه ، وبعدها يقوم المستخدم بتحديد أية معلومة تتعلق به سيتم إضافتها مباشرة إلى عنصر JTextArea . نظراً لطول النص البرمجي في هذا المثال ، سنقوم

بسرد بعض الملاحظات التي تساعدنا في فهم المثال :

١. من أجل كتابة معالج حدث للأغراض من الصف JRadioButton أو JCheckBox فإنه يمكننا أن نحقق الواجهة ItemListener ، والتي تتضمن تابعاً واحداً فقط هو ItemChangeListener والذي يأخذ وسيطاً واحداً من النمط ItemEvent .
٢. لربط غرض من الصف JRadioButton أو JCheckBox مع المتتصت ItemListener نستخدم التابع addItemListener
٣. من أجل كتابة حدث للأغراض من الصف JList فإنه يجب تحقيق الواجهة ListSelectionListener ، والتي تتضمن تابعاً واحداً فقط هو ValueChenged الذي يأخذ وسيطاً من الصف ListSelectionEvent
٤. لربط غرض من الصف JList مع المتتصت ListSelectionListener فإنه علينا استخدام التابع addListSelectionListener .
٥. يُستخدم التابع getSource للحصول على الغرض الذي وقع عليه الحدث. لذلك سنجد أننا استخدمناه ضمن النص البرمجي لنعرف أي عنصر قد ضغطه المستخدم ، ويُستخدم التابع getStatechange لمعرفة ما إذا كانت حالة العنصر JCheckBox أو JRadioButton قد تغيرت ، ونستخدم الثابت SELECTED للدلالة على أن العنصر قد تم اختياره ، أما الثابت DESELECTED يعني أن العنصر كان مختاراً ، وأزال المستخدم اختياره .
٧. التابع append التابع للصف JTextArea يقوم بإضافة السلسلة المحرفية الممررة إليه أي إلى النص الذي يحتويه غرض من ذلك الصف.
٨. علينا إضافة مجموعة الأغراض Jradiobutton التي نعرفها من أجل هدف محدد ضمن غرض من الصف ButtonGroup ، فمثلاً نريد من

المستخدم أن يحدد هل هو ذكر أم انثى ، وكذلك نريده أن يحدد هل هو متزوج أم عزب أم أرمل ، عندها علينا وضع الأغراض التي تتعلق بالهدف الأول (ذكر/أنثى) ضمن غرض من الصف ButtonGroup ، والأغراض التي تتعلق بالهدف الثاني (متزوج/أعزب/أرمل) ضمن غرض آخر من الصف ButtonGroup .

مثال (9-10)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing. event.*;
public class Test3 {
    // Creation & initialization the GUI objects
    JFrame wind = new JFrame("Hi everybody!");
    String items[ ] = {"Mr", "Mrs", "Miss"};
    JComboBox title = new JComboBox(items);
    JPanel p1, p2;

    JCheckBox arabic = new JCheckBox("Arabic");
    JCheckBox english = new JCheckBox("English");
    JCheckBox french = new JCheckBox("Russian");
    JRadioButton male = new JRadioButton("M");
    JRadioButton female = new JRadioButton("F");
    ButtonGroup radioGroup = new ButtonGroup();

    String country[ ] = {"Syria", "Russia", "Germany"};
    JList countries = new JList(country);
    JTextArea info = new JTextArea( );
    // Implementation of ItemListener interface
    class CheckListen implements ItemListener {
```

```

public void itemStateChanged( ItemEvent e) {
    if (e.getSource( ) == arabic)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n Arabic");
    if (e.getSource( ) == english)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n English");
    if (e.getSource( ) == french)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n French");
    if (e.getSource( ) == male)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n Male");

    if (e.getSource( ) == female)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n female");
    if (e.getSource( ) == english)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n English");

    if (e.getSource( ) == french)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n French");

    if (e.getSource( ) == male)
    if (e.getStateChange( ) == ItemEvent.SELECTED)
    info.append("\n Male");

    if (e.getSource( ) == female)
    if (e.getStateChange( ) == ItemEvent.SELECTED)

```

```

        info.append("\n female");
        title.addItemListener(listener);
        countries.addListSelectionListener(new ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                info.append("\n" + countries.getSelectedValue().toString());
            }
        });
    }

    private void buildInterface() {
        p1 = new JPanel(new GridLayout(1,7));
        p2 = new JPanel(new BorderLayout());
        p1.add(title);
        p1.add(arabic); p1.add(english); p1.add(french);
        p1.add(male); p1.add(female);
        p1.add(countries);
        p2.add(info);
        wind.add(p1,BorderLayout.NORTH);
        wind.add(p2,BorderLayout.CENTER);
        wind.pack();
        wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        radioGroup.add(male); radioGroup.add(female);
    }

    public static void main(String args[] ) {
        Test3 win = new Test3();
    } //end main
} //end class Test3

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



٥- مساحات النصوص JTextArea

عند الحاجة لإدخال عدة أسطر نصية يجب علينا إنشاء عدة أغراض من الصف `JTextField` لكن هذه الطريقة غير مناسبة لذلك من الأفضل استخدام الصف `JTextArea` الذي يتيح للمستخدم إمكانية إدخال عدة أسطر نصية . ويصنع الباني الافتراضي التابع للصف `JTextArea` مساحة نصية فارغة ، ويملك الصف `JTextArea` البواني التالية بالإضافة إلى الباني الافتراضي :

`public JTextArea ( int rows , int columns ) ;`

- يصنع مساحة نصية مؤلفة من عدد محدد من الأسطر والأعمدة .

`public JTextArea (String text , int rows , int columns ) ;`

- يصنع مساحة نصية مؤلفة من النص الممر ومن عدد محدد من الأسطر والأعمدة .

يمكن استخدام التوابع التالية :

`public void insert ( String s , int pos ) ;`

- يقوم هذا التابع بحشر النص `S` ضمن المساحة النصية في المكان المحدد `pos`

`public void append ( String s ) ;`

- يضيف النص `s` إلى نهاية النص .

- يستبدل النص الجزئي الممتد ضمن المجال المحدد من start إلى end بالنص المحدد s . ويبين المثال (10-10) كيفية نسخ عدد من الأسطر من مساحة نص إلى أخرى .

مثال (10-10)

// Copying selected text from one textarea to another.

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.JScrollPane;

public class TextAreaFrame extends JFrame {
    private JTextArea textArea1; // displays demo string
    private JTextArea textArea2; // highlighted text is copied here
    private JButton copyJButton; // initiates copying of text

    // no-argument constructor
    public TextAreaFrame() {
        super( "TextArea Demo" );
        Box box = Box.createHorizontalBox( ); // create box
        String demo = "This is a demo string to\n" +
            "illustrate copying text\nfrom one textarea to \n" +
            "another textarea using an\nexternal event\n" +
            "\nالجمهورية العربية السورية";

        textArea1 = new JTextArea( demo, 10, 15 ); // create textArea1
        box.add( new JScrollPane( textArea1 ) ); // add scrollpane
```



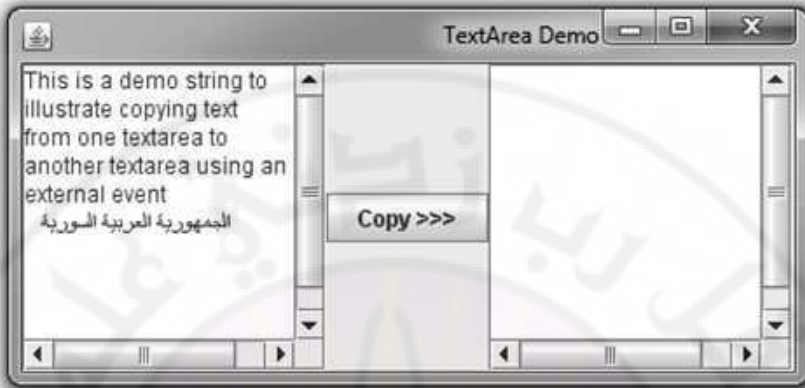
```

copyJButton = new JButton( "Copy >>>" ); // create copy button
box.add( copyJButton ); // add copy button to box
                                // anonymous inner class
copyJButton.addActionListener( new ActionListener( )
{
    // set text in textArea2 to selected text from textArea1
    public void actionPerformed((ActionEvent event) )
    {
        textArea2.setText( textArea1.getSelectedText( ) );
    } // end method actionPerformed
} // end anonymous inner class
); // end call to addActionListener
textArea2 = new JTextArea( 10, 15 ); // create second textarea
textArea2.setEditable( false ); // disable editing
box.add( new JScrollPane( textArea2 ) ); // add scrollpane

add( box ); // add box to frame
} // end TextAreaFrame constructor
} // end class TextAreaFrame
//-----
// Copying selected text from one textarea to another.
import javax.swing.JFrame;
public class TextAreaDemo {
    public static void main( String args[ ] ) {
        TextAreaFrame textAreaFrame = new TextAreaFrame( );
        textAreaFrame.setDefaultCloseOperation(
                                JFrame.EXIT_ON_CLOSE );
        textAreaFrame.setSize( 425, 200 ); // set frame size
        textAreaFrame.setVisible( true ); // display frame
    } // end main
} // end class TextAreaDemo

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



١٠-٧- صفوف الـ Adapters للأحداث

في بعض الأحيان تتضمن الواجهة التي نريد تحقيقها عدة توابع , ويجب تحقيق جميع التوابع التي تتضمنها تلك الواجهة في الوقت الذي نكون بحاجة إلى تابع واحد فقط . ما العمل ؟؟؟

في الواقع حل المشكلة يكمن في استخدام صفوف adapter لمتنصتي الأحداث (event listener adapter) ، إن كل صف من هذه الصفوف يحقق واجهة معينة لحدث ، حيث إنه يحوي تحقيقات مسبقة التعريف لجميع توابع الواجهة (بحيث يكون جسم التابع فارغاً) ، وبهذا الشكل فإننا لسنا مضطرين إلى تحقيق جميع التوابع وإنما فقط إلى التابع الذي نحتاجه . المثال (10-11) يوضح ذلك ، حيث يقوم المستخدم بتحريك الفأرة على نافذة البرنامج ، فتظهر إحداثيات الفأرة في شريط العنوان للنافذة ، وعلينا تحقيق الواجهة التي تتضمن المتنصت على حدث تحريك الفأرة ، هذه الواجهة هي MouseMotionListener والتي تتضمن التابعين mouseMoved و

mouseDragged ، حيث يتم تنفيذ الأول عند تحريك الفأرة بينما يتعلق الثاني بعملية السحب والإفلات .

مثال (11-10)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Mouse extends JFrame
    implements MouseMotionListener
{
    public Mouse( ) {
        super("Mouse Move");
    }
    public void mouseMoved(MouseEvent e) {
        /* getX and getY methods are
        used for getting the coordinate of mouse cursor. */
        setTitle("(X,Y)=" + "(" + e.getX( ) + "," + e.getY( ) + ")");
    }
    public void mouseDragged(MouseEvent e)
    {
        // do nothing
    }
    public static void main(String args[ ]) {
        Mouse wind = new Mouse( );
        wind.setLayout(null);
        wind.setSize(300,200);
        wind.addMouseMotionListener(wind);
        wind.setVisible(true);
        wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    } //end main
} //end class
```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



نلاحظ أننا وضعنا التابع `mouseDragged` ضمن الصف على الرغم من أننا لسنا بحاجة إليه (وذلك لأن تحقيق الواجهة إجباري) . الحل الأفضل هو استخدام صف `adapter` مناسب للحدث , سوف نعيد المثال السابق باستخدام الصف `MouseMotionAdapter` والذي يقوم بتحقيق الواجهة `. MouseMotionListener`

مثال (10-12)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

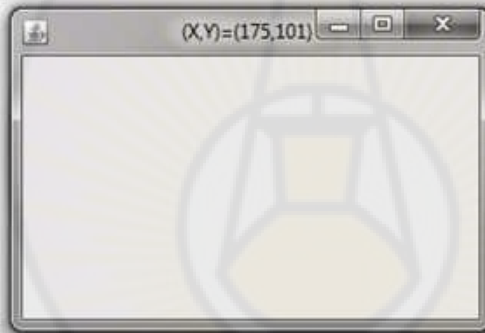
public class Mouse5 extends JFrame {
    public Mouse5()
    {
        super(" Mouse Move");
        addMouseMotionListener(new MouseMotionAdapter(){
            public void mouseMoved(MouseEvent e)
            {
                setTitle("(X,Y)=" + "(" + e.getX() + "," + e.getY() + ")");
            }
        });
    }
} //end Mouse5()
```

```

public static void main(String args[ ]) {
    Mouse5 wind = new Mouse5( );
    wind.setLayout(null);
    wind.setSize(300,200);
    wind.setVisible(true);
    wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
} //end main
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



يبين الجدول (5 - 10) عدداً من صفوف **adapter** مع الواجهات التي يحققها

Event-adapter class in java.awt.event	Implements interface
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

الجدول (5 - 10) - يبين عدداً من صفوف **adapter** مع الواجهات التي يحققها

١٠ - ٨ - التعامل مع أحداث لوحة المفاتيح والفأرة

المثال (10-13) يوضح كيفية التعامل مع أحد أحداث لوحة المفاتيح ، حيث سنقوم بكتابة معالجة حدث يمنع المستخدم من إدخال أي شيء في حقل النص باستثناء الأرقام . في هذا المثال سنستخدم الحدث KeyEvent مع التابع keyTyped ، ونستخدم التابع getKeyChar الذي يعيد متحولاً من النوع char لمعرفة المحرف الذي تم ضغطه ، ومن ثم نختبر انتماء هذا المحرف إلى المجال [0 - 9] فإذا كان خارج هذا المجال فإننا نتجاهله من خلال تغيير قيمته باستخدام التابع setKeyChar("\0") .

أخيراً يمكننا الحصول على قيمة الكود المقابل للمحرف المضغوط من خلال التابع getKeyCode ، ويمكن معرفة المفتاح المضغوط Alt, Shift, Ctrl من خلال التوابع التالية ، وهي على الترتيب : isAltDown, isShiftDown, isControlDown والتي تعيد قيمة بولانية bool تشير فيما إذا كان المفتاح المقابل مضغوط أم لا .

مثال (10-13)

```
import javax.swing.*;
import java.awt.event.*;

public class keyKeyNumbers extends JFrame {
    private JTextField textField =
        new JTextField("You can only write NUMBERS");
    public keyKeyNumbers() {
        super("write NUMBERS");
        textField.addKeyListener(new KeyAdapter() {
            public void keyTyped(KeyEvent e)
            {
                if(e.getKeyChar() < '0' || e.getKeyChar() > '9')
                {
```

```

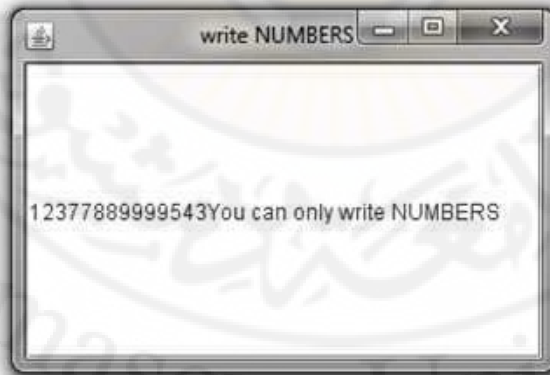
        e.setKeyChar('\0');
    }
}
});

textField.setSize(114,30);
add(textField);
} //end keyKeyNumbers( )

public static void main(String args[ ]) {
    keyKeyNumbers win = new keyKeyNumbers( );
    win.setSize(300,200);
    win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    win.setVisible(true);
} //end main
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



يبين المثال (10-14) تعقب أحداث الفأرة في اللوحة JPanel ، و
 نستخدم في هذا المثال اللوحة JPanel كمنطقة للرسم التي يمكن للمستخدم رسم

طريق سحب الفأرة ، ويوضح هذا المثال صفوف adapter للأحداث ، يستخدم التابع paintComponent من أجل الرسم .

مثال (10-14)

// Using class MouseMotionAdapter.

```
import java.awt.Point;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionAdapter;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Color;

public class PaintPanel extends JPanel {
    private int pointCount = 0 ; // count number of points
    // array of 10000 java.awt.Point references
    private Point points[ ] = new Point[ 10000 ];
    // set up GUI and register mouse event handler
    public PaintPanel( ) {
        // handle frame mouse motion event
        addMouseListener(
            new MouseMotionAdapter( ) // anonymous inner class
            {
                // store drag coordinates and repaint
                public void mouseDragged( MouseEvent event )
                {
                    if ( pointCount < points.length )
                    {
                        points[ pointCount ] = event.getPoint(); // find point
                        pointCount++; // increment number of points in array
                    }
                }
            }
        );
    }
}
```

```

        repaint( ); // repaint JFrame
    } // end if
} // end method mouseDragged
} // end anonymous inner class
); // end call to addMouseListener
} // end PaintPanel constructor

// draw oval in a 4-by-4 bounding box at specified location on window
public void paintComponent( Graphics g )
{
    super.paintComponent( g ); // clears drawing area
    g.setColor( Color.RED );
    // draw all points in array
    for ( int i = 0 ; i < pointCount; i++ )
        g.fillOval( points[ i ].x, points[ i ].y, 4, 4 );
} // end method paintComponent
} // end class PaintPanel
//-----
// Testing PaintPanel.
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class Painter
{
    public static void main( String args[] )
    {
        // create JFrame
        JFrame application = new JFrame( "A simple paint program" );
        PaintPanel paintPanel = new PaintPanel( ); // create paint panel
                                                // in center
    }
}

```

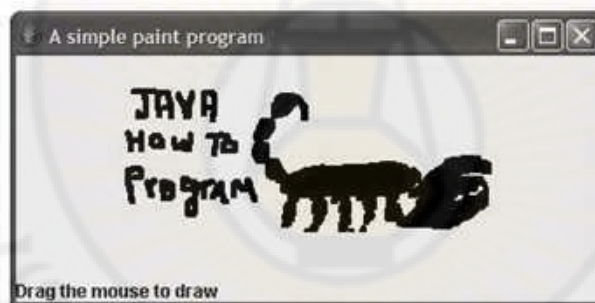


```

application.add( paintPanel, BorderLayout.CENTER );
// create a label and place it in SOUTH of BorderLayout
application.add( new JLabel( "Drag the mouse to draw" ),
    BorderLayout.SOUTH );
application.setDefaultCloseOperation(
    JFrame.EXIT_ON_CLOSE );
application.setSize( 400, 200 ); // set frame size
application.setVisible( true ); // display frame
} // end main
} // end class Painter

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



١٠-٩- استخدام العنصر Timer والعنصر Border

يقوم العنصر Timer بتكرار عمل ما وذلك عند انقضاء فترة زمنية محددة . ويقوم المثال (10-15) بالتعرف على كيفية استخدام العنصر Timer ، ونلاحظ أن لون العبارة "One for All & All for One" سيتغير كل ثانية ، وأن لون العبارة الثانية " بسم الله الرحمن الرحيم " يتغير كل ثانية واحدة أيضاً ولكن بشكل غير متزامن ويعود هذا إلى استخدام غرض timer من الصف Timer ، وأن أغراض هذا الصف مسؤولة عن تكرار عمل معين بعد

انقضاء فترة زمنية معينة . يجب أن نضع النص البرمجي الذي نريد تكراره داخل الحدث الخاص بالغرض timer وهو من النوع `ActionEvent` .

ستبدأ عملية تكرار العمل بمجرد استدعاء التابع `start` من قبل الغرض `timer` ، أما إذا أردنا إيقاف تكرار العمل ، فيجب استدعاء التابع `stop` .

لإضافة حدود للعبارة `One for All & All for One` في هذا البرنامج (خطأ يحد العبارة) يجب استيراد الحزمة `*. java.swing.border` ثم إنشاء غرض من الصف `TitledBorder` ، وتمريره إلى التابع `setBorder` التابع للغرض `panel` من الصف `Jpanel` وذلك من أجل إضافة حدود له . يتم ضبط المدة الزمنية للغرض `timer` من خلال الوسيط الأول لباني الصف `Timer` الذي يمثل قيمة صحيحة مقدرة بالميلي ثانية . ويمكننا من خلال التابع `setTitleColor` التحكم بلون عنوان الحدود لأغراض الصف `TitledBorder` .

يوجد أنواع أخرى من الحدود يمكن استخدامها ، فمثلاً يمكن استخدام الصف `LineBorder` والذي يستخدم على النحو التالي :

```
LineBorder border = new LineBorder ( Color.RED,10 );
panel.setBorder(border);
```

فيمثل الوسيط الأول اللون الذي سيتم استخدامه لرسم الحدود ، أما الوسيط الثاني فيمثل عرض الحدود وهو قيمة صحيحة .

مثال (10-15)

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.util.Random;
```

```
public class UsingTimer extends JFrame{
private JPanel panel = new JPanel( );
```

```

private JLabel text = new JLabel ("One for All & All for One");
private JLabel text1 = new JLabel ( " بسم الله الرحمن الرحيم " );
private Timer timer;
private Timer timer1;
public UsingTimer( ) {
    super.setTitle("Using Timer and Border");
    text.setFont(new Font("Arial", Font.BOLD, 24));
    text.setHorizontalAlignment(JLabel.CENTER);
    text1.setFont(new Font("Arial", Font.BOLD, 40));
    text1.setHorizontalAlignment(JLabel.CENTER);
    TitledBorder titledBorder =
        new TitledBorder("That sounds nice!!!");
    //LineBorder titledBorder = new LineBorder(Color.RED,10);
    titledBorder.setTitleColor(Color.RED) ;
    panel.setBorder(titledBorder);
    panel.add(text);
    panel.add(text1);
    // creating timer object
    timer = new Timer(1000, new ActionListener( ) {
        public void actionPerformed(ActionEvent e) {
            Random random = new Random( );
            int r = Math.abs(random.nextInt( ) % 225);
            int g = Math.abs(random.nextInt( ) % 225);
            int b = Math.abs(random.nextInt( ) % 225);
            Color c = new Color(r, g, b);
            text.setForeground(c);
            Random random1 = new Random( );
            int r1 = Math.abs(random1.nextInt( ) % 220);
            int g1 = Math.abs(random1.nextInt( ) % 220);
            int b1 = Math.abs(random1.nextInt( ) % 220);
            Color c1 = new Color(r1, g1, b1);

```

```

        text1.setForeground(c1);
    }
});
timer.start( );
add(panel);
} //end func. UsingTimer()
public static void main(String ag[ ]) {
    UsingTimer wind = new UsingTimer( );
    wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    wind.setSize(350, 160);
    wind.setVisible(true);
} //end main
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي في الحالتين :

١- استخدام الصف `TitledBorder`



٢- استخدام الصف `LineBorder`



١٠-١٠ - استخدام العنصر JcolorChooser

يسمح لنا العنصر JcolorChooser باختيار لون ما من مجموعة كبيرة من الألوان ، ويتمتع هذا العنصر بوجود ثلاث بوابات نتحكم من خلالها باللون المطلوب . سنقوم في المثال (10 - 16) بإضافة عنصرين إلى الواجهة المرئية ، عبارة عن عنصر من الصف Jlabel ، والثاني من الصف JColorChooser. هذا المثال بسيط حيث ينتقي المستخدم لوناً ما من مربع الألوان وعندها يصبح لون النص وفق اللون المختار ، وعندما يقوم المستخدم باختيار لون ما يتم اصدار الحدث ChangeEvent ، ويتم معالجته من خلال التابع stateChanged . ويمكننا بدلاً من إضافة مربع الألوان JColorChooser إلى النافذة بحد ذاتها يمكننا إظهاره بشكل مستقل ، ولتحقيق ذلك نستدعي التابع showDialog والذي يعيد غرضاً من الصف Color يمثل اللون الذي اختاره المستخدم ، أما وسطاء هذا التابع فهي كما يلي :

حيث الوسيط الأول هو النافذة الأساسية والثاني عنوان مربع حوار الألوان والثالث اللون الافتراضي وفيما يلي توضيح ذلك .

```
color = colorChooser.showDialog(  
    null, "Color Chooser", Color.RED );
```

حيث color غرض من الصف Color ، وفي هذه الحالة نحن لسنا بحاجة إلى كتابة معالج حدث للعنصر JColorChooser .

مثال (10-16)

```
import javax.swing.*;  
import javax.swing.event.*;  
import java.awt.*;  
public class UsingColors extends JFrame{  
    private JLabel label =  
        new JLabel("Let's start!!!",JLabel.CENTER);
```

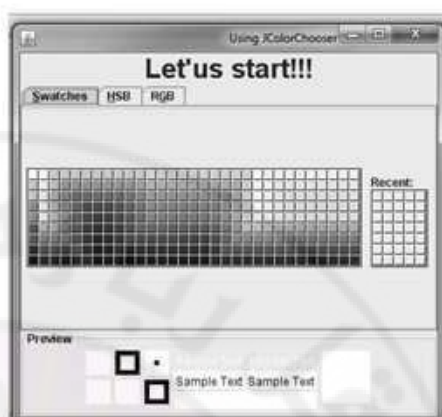
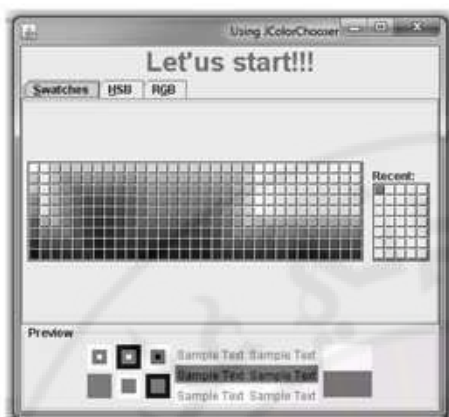


```

private JColorChooser colorChooser = new JColorChooser( );
public UsingColors( ) {
    super("Using JColorChooser");
    label.setFont(new Font("Arial", Font.BOLD, 28));
    this.add(label);
    this.add(label, BorderLayout.NORTH);
    this.add(colorChooser, BorderLayout.SOUTH);
    colorChooser.getSelectionModel().addChangeListener(
        new ChangeListener( )
    {
        public void stateChanged(ChangeEvent e)
        {
            label.setForeground(colorChooser.getColor( ));
        }
    });
} //end func. UsingColors()
public static void main(String ag[ ]) throws Exception
{
    UsingColors wind = new UsingColors( );
    wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    wind.pack( );
    wind.setVisible(true);
} //end main
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



١٠-١١ - استخدام العنصر JTabbedPane

يُعد العنصر JTabbedPane أحد أنواع الحاويات ، تتألف الحاوية من عدة بوابات نستطيع أن نصنع في كل عنصر مجموعة من العناصر ، ويوضح المثال التالي كيفية استخدام هذا العنصر ، حيث سنقوم بإنشاء غرض من الصف JTabbedPane ، وإنشاء ثلاثة أغراض من الصف JLabel ، الأول نضيفه إلى البوابة الأولى من العنصر JTabbedPane ، أما الغرض الثاني فنضيفه للبوابة الثانية ، و الغرض الثالث نضيفه للبوابة الثالثة .

في الواقع في هذا المثال لا يوجد عناصر جديدة أو تباع كثيرة لشرحها ، فالجديد هو استخدام التابع addTab الذي يستخدم من أجل إنشاء Tab ، ويملك هذا التابع أربعة وسطاء الأول هو عنوان البوابة ، الثاني عبارة عن غرض من الصف Icon وذلك إذا أردنا إظهار أيقونة على البوابة ، والثالث يمثل العنصر الذي نريد إضافته إلى البوابة وقد يكون العنصر عبارة عن JPanel يتضمن عناصر أخرى ، وأما الرابع فهو عبارة عن تلميح يظهر عند وضع المؤشر على البوابة .

مثال (10-17)

```
import javax.swing.*;
import java.awt.*;

public class UsingTabs extends JFrame{
    private JTabbedPane tabPane = new JTabbedPane( );
    private JLabel firstLabel =new JLabel("HELLO everyone");
    private JLabel secondLabel =new JLabel("BONJOUR a tous");
    private JLabel thirdLabel =new JLabel("السلام عليكم");

    public UsingTabs( ) {
        super.setTitle("JtabbedPane");
        firstLabel.setForeground(Color.RED);
        secondLabel.setForeground(Color.BLUE);
        thirdLabel.setForeground(Color.BLACK);

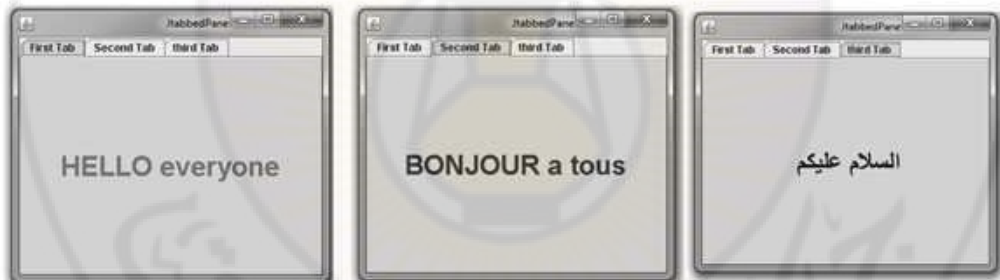
        firstLabel.setFont(new Font("Arial", Font.BOLD, 30));
        secondLabel.setFont(new Font("Arial", Font.BOLD, 30));
        thirdLabel.setFont(new Font("Arial", Font.BOLD, 30));
        firstLabel.setHorizontalAlignment(JLabel.CENTER);
        secondLabel.setHorizontalAlignment(JLabel.CENTER);
        thirdLabel.setHorizontalAlignment(JLabel.CENTER);
    }
}
```

```

tabPane.addTab("First Tab", null, firstLabel, "English");
tabPane.addTab("Second Tab", null, secondLabel, "French");
tabPane.addTab("third Tab", null, thirdLabel, "Arabic");
this.add(tabPane);
} //end func. UsingTabs()
public static void main(String ag[ ]) {
    UsingTabs wind = new UsingTabs( );
    wind.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    wind.setSize(350, 300);
    wind.setVisible(true);
} //end main
} //end class

```

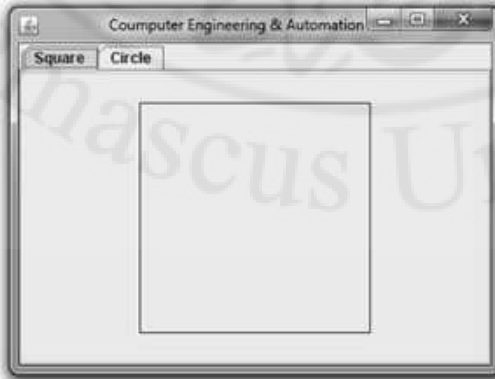
وعند تنفيذ البرنامج سنحصل على الخرج التالي:



مسائل عامة

١- اكتب برنامجاً بلغة Java يستخدم الصف JTabbedPane يحتوي على بوابتين (صفحتين) من أجل عرض شكلين هما : المربع والدائرة ، بحيث يجب النقر على البوابة المطلوبة حتى يتم عرض الشكل المطلوب ، ويحتوي البرنامج على الصفوف التالية :

- صف اسمه myTabbedPane يرث الصف JFrame يحتوي على بان افتراضي ويستدعي باني الصف الأساسي لكتابة عنوان الاطار: Coumputer Engineering & Automation ، يتم إنشاء بوابتين من الصف JTabbedPane بحيث يكون اسم البوابة الأولى Square أما الثانية Circle .
- صف اسمه myFigurePanel يرث الصف JPanel من أجل رسم الدائرة والمستطيل .
- صف اختباري tabbedPaneTest يحتوي على التابع main من أجل انشاء عرض من الصف myTabbedPane واستدعاء التوابع اللازمة من أجل عرض الإطار مع عنوانه والشكل الهندسي المناسب ، بحيث نحصل على الخرج التالي :

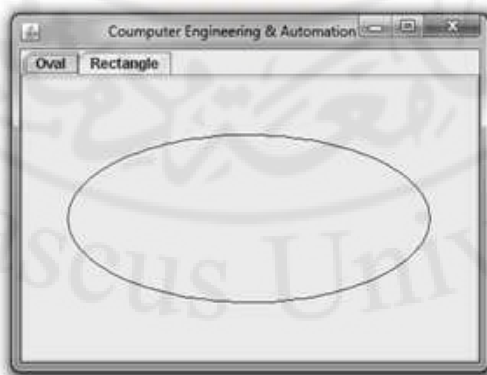


٢- اكتب برنامجاً بلغة Java يستخدم الصف JTabbedPane يحتوي على بوابتين (صفحتين) من أجل عرض شكلين هما : المربع والدائرة ، بحيث يجب النقر على البوابة المطلوبة حتى يتم عرض الشكل المطلوب ، ويحتوي البرنامج على الصفوف التالية :

- صف اسمه myTabbedPane2 يرث الصف JFrame يحتوي على بان افتراضي ويستدعي بانني الصف الأساسي لكتابة عنوان الإطار: Coumputer Engineering & Automation ، يتم إنشاء بوابتين من الصف JTabbedPane بحيث يكون اسم البوابة الأولى Rectangle أما الثانية Oval .

- صف اسمه myFigurePanel2 يرث الصف JPanel من أجل رسم الدائرة والمستطيل .

- صف اختباري tabbedPane2 يحتوي على التابع main من أجل إنشاء عرض من الصف myTabbedPane2 واستدعاء التوابع اللازمة من أجل عرض الإطار مع عنوانه والشكل الهندسي المناسب ، بحيث نحصل على الخرج التالي :

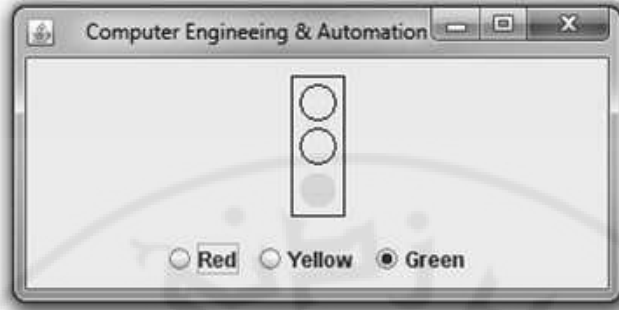


٣- اكتب برنامجاً بلغة Java يحاكي عمل إشارات المرور يستخدم الصف JButton من أجل إنشاء ثلاثة أزرار خيار لثلاثة ألوان (RED, GREEN, YELLOW) و الصف ButtonGroup يستخدم من أجل تجميع أزرار الخيار الثلاثة التي تم إنشاؤها من الصف JButton .
يحتوي البرنامج على الصفوف التالية :

- صف اسمه myRadioButton يرث الصف JFrame ويحقق الواجهة ItemListener يتم إنشاء ثلاثة أغراض (أزرار خيار) من الصف JButton وإنشاء غرض من الصف ButtonGroup من أجل تجميع الأزرار ، ويحتوي على بانٍ افتراضي ويستدعي باني الصف الأساسي لكتابة عنوان الإطار : Computer Engineering & Automation ، ثم إنشاء غرض (لوحة) من الصف JPanel من أجل إضافة إشارات المرور إلى اللوحة و ثم إضافة اللوحة إلى الإطار ، ثم إنشاء غرض ثانٍ (لوحة ثانية) من الصف JPanel من أجل إضافة أزرار الخيار إلى اللوحة ومن ثم إضافة اللوحة إلى الإطار .

- صف اسمه Light يرث الصف JPanel يحتوي على ثلاثة متحولات خاصة red, green, yellow من نوع boolean ويحتوي على التابع paintComponent من أجل رسم دائرة حمراء ودائرة صفراء ودائرة خضراء ومستطيل يحتوي على الدوائر الثلاث .

- صف اختباري testRadio يحتوي على التابع main من أجل إنشاء غرض من الصف myRadioButton واستدعاء التابع اللازمة من أجل عرض الإطار مع عنوانه وشكل إشارات المرور كما هو مبين في الشكل .



٤- أكتب برنامجاً بلغة Java يعرض رسالة على اللوحة ويستخدم الزين (`<=>`) في اللوحة من أجل تحريك الرسالة الموجودة على اللوحة إلى اليمين وإلى اليسار ، بحيث يحتوي البرنامج على الصفوف التالية :

- صف اسمه `MessagePanel` يرث الصف `JPanel` من أجل عرض الرسالة على اللوحة ويتيح هذا الصف للمستخدم من تحديد مكان الرسالة ومركزها ، كما يتيح لنا تحريك الرسالة وفق الفاصل المحدد .
- صف اسمه `ButtonDemo` يرث الصف `JFrame` ويحقق الواجهة `ActionListener` ، يتم إنشاء زررين `<=>` و `>=<` في اللوحة ثم وضع هذه اللوحة في أسفل (`south`) الإطار . نستخدم متتصت الحدث لكل زر `addActionListener` ونستخدم معالج الحدث `actionPerformed` من أجل تحريك الرسالة إلى اليسار أو اليمين وذلك بحسب الزر الذي تم النقر عليه .
- صف اختباري `testString` يحتوي على التابع `main` من أجل إنشاء غرض من الصف `ButtonDemo` واستدعاء التوابيع اللازمة من أجل عرض الإطار مع عنوانه وشكل إشارات المرور كما هو مبين في الشكل .



٤- اكتب برنامجاً بلغة Java يحول درجات الحرارة من مقياس فهرنهايت إلى مقياس سلسيوم وبالعكس ، بحيث إذا قمنا بإدخال القيمة إلى حقل النص الخاص بمقياس سلسيوم ثم ضغطنا على الزر Enter عندها سيتم عرض درجة الحرارة المطابقة وفق مقياس فهرنهايت ضمن حقل النص الخاص بمقياس فهرنهايت ، وبالعكس إذا قمنا بإدخال القيمة إلى حقل النص بمقياس فهرنهايت ثم ضغطنا على الزر Enter عندها سيتم عرض درجة الحرارة المطابقة وفق مقياس سلسيوم في حقل النص الخاص بمقياس سلسيوم . ويحتوي البرنامج على الصفوف التالية :

- صف اسمه TextFieldTemp يرث الصف JFrame ويحقق الواجهة ActionListener ، إنشاء لوحة P1 وتحديد مدير التنسيق لها GridLayout(2, 1) من أجل إنشاء لافتتين ، إنشاء لوحة ثانية P2 وتحديد مدير التنسيق GridLayout(2, 1) . من أجل إنشاء حقل نص ، ونضيف اللوحة الأولى إلى اليسار (west) والثانية إلى مركز لوحة المحتوى التابعة للإطار . نستخدم متتصت الحدث addActionListener على الحقلين ونستخدم معالج الحدث actionPerformed من أجل تحديد حقل النص الذي تمت عملية الضغط ضمنه .

- صف اختباري testTextField يحتوي على التابع main من أجل إنشاء عرض من الصف TextFieldDemo واستدعاء التوابع اللازمة من أجل عرض الإطار مع عنوانه وشكل إشارات المرور كما هو مبين في الشكل .

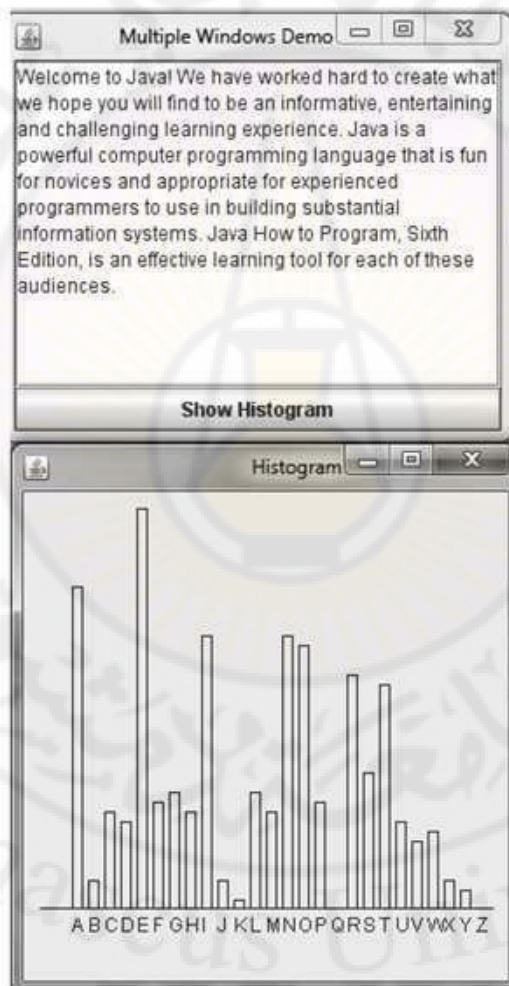


٥- اكتب برنامجاً بلغة Java يقوم بإنشاء نافذة رئيسية تحتوي على مساحة نصية موجودة ضمن لوحة تمرير وعلى زر يحمل الاسم show Histogram ، وعندما ينقر المستخدم على هذا الزر تظهر نافذة جديدة من أجل عرض المخطط البياني الذي يعبر عن عدد مرات ظهور كل حرف موجود ضمن السلسلة الممررة إلى المساحة النصية ، يحتوي البرنامج على الصفوف التالية :

- صف رئيسي اسمه MultipleWindowsDemo يرث الصف JFrame يحقق الواجهة ActionListener ، حيث يتم إنشاء زر يحمل الاسم Show Histogram ووضعه في أسفل الإطار ، إضافة مساحة نصية داخل لوحة التمرير ثم وضع لوحة التمرير في مركز الإطار .
 - صف مشتق اسمه Histogram يرث الصف JPanel يحتوي الصف على حقل بيانات يدعى count [] int مهمته حساب عدد مرات ظهور الأحرف وعرض القيم الموجودة في هذا الحقل على هيئة مخطط بياني . نستخدم معالج الحدث actionPerformed في الإطار
- MultipleWindowsDemo كما يلي :

- إنشاء عرض من الصف Histogram وحساب عدداً لأحرف في المساحة النصية ثم تمرير العدد إلى الغرض histogram .

- إنشاء إطار جديد ووضع الغرض histogram في مركز الإطار .
- صف اختباري testHistogram يحتوي على التابع main من أجل إنشاء غرض من الصف MultipleWindowsDemo واستدعاء التتابع اللازمة من أجل عرض الإطار مع عنوانه وشكل إشارات المرور كما هو مبين في الشكل التالي :



الفصل الحادي عشر

معالجة الاستثناءات

Exception Handling





معالجة الاستثناءات

Exception Handling

١١-١ - مقدمة

تقدم Java تقنية تُدعى الاستثناء Exception التي تمكن المبرمج من تحديد الأخطاء ومعالجتها ، فعندما يحدث خطأ ما فإن البرنامج يرمي استثناء ، هذا يعني أن التدفق الطبيعي لتنفيذ البرنامج يتوقف ، وتحاول بيئة التنفيذ أن تجد معالج استثناء Exception Handler الذي يُشكل مجموعة من التعليمات تقوم بمعالجة نوع خاص من الأخطاء ، ويحاول معالج الاستثناء أن يعود من حالة الخطأ ، وفي حال عدم قدرته على العودة عن الخطأ فإنه يُقدم شكلاً أنيقاً للخروج من البرنامج . إذاً الاستثناء هو حدث ما ، يحصل خلال تنفيذ البرنامج ، يؤدي إلى توقف تنفيذ تعليمات البرنامج بشكل طبيعي .

هناك أنواع كثيرة من الأخطاء التي يمكن أن تُسبب الاستثناءات ، حيث تتدرج هذه المشاكل من أخطاء على مستوى العتاد Hardware إلى أخطاء برمجية بسيطة . عندما يحدث خطأ ما في تابع ما فإن التابع يقوم بإنشاء غرض استثناء ، وهذا الغرض يتضمن معلومات حول نوعه وعن حالة البرنامج عندما حدث الخطأ ، عندها تكون بيئة التنفيذ Runtime System مسؤولة عن إيجاد طريقة ما لمعالجة الخطأ ، ويوجد ثلاثة أنواع من الأخطاء : أخطاء قواعدية وأخطاء زمن التنفيذ وأخطاء المنطقية . تظهر الأخطاء القواعدية عند عدم إتباع قواعد لغة البرمجة ويتم كشفها بواسطة المترجم Compiler ، وتظهر أخطاء زمن التنفيذ أثناء عمل البرنامج في حال اكتشفت البيئة وجود عملية مستحيلة التنفيذ ، وأخيراً تحدث الأخطاء المنطقية عندما لا ينجز البرنامج العمليات المطلوبة منه وفق الطريقة المخطط لها .

وبشكل عام تُعد الأخطاء القواعدية سهلة الكشف ، وسهلة التصحيح ، وذلك لأن المترجم يُحدد مصدرها وسبب حدوثها . وفي هذا الفصل سنتعرف على كيفية استخدام الاستثناءات من أجل التأكد من صحة البرنامج .

١١-٢- الاستثناءات وأنماطها

تتسبب أخطاء زمن التنفيذ في رمي الاستثناءات ، التي هي عبارة عن أحداث تقع أثناء تنفيذ البرنامج وتقاطع التدفق الطبيعي للتحكم . **مثال :** لنفرض أنه لدينا برنامجاً يُحاول نقل النقود من حساب التوفير إلى الحساب الجاري ، ولكن بسبب حدوث خطأ أثناء زمن التنفيذ تم إنهاء هذا البرنامج بعد سحب النقود من حساب التوفير وقبل وضعها في الحساب الجاري ، عندها يفقد الزبون نقوده التي كان سيسحبها .

تُزود لغة Java المبرمجين بالعديد من الإمكانيات التي تمكنهم من معالجة أخطاء زمن التنفيذ والتي تساعدهم في تطوير برامج قوية من أجل تنفيذ المهام الحرجة . فعلى سبيل المثال يتم إنهاء البرنامج في المثال (1-11) بشكل غير طبيعي وذلك بسبب عملية التقسيم على الصفر .

مثال (1-11)

// An application that attempts to divide by zero.

```
import java.util.Scanner;
```

```
public class DivideByZeroNoExceptionHandling {
```

```
// demonstrates throwing an exception when a divide-by-zero occurs
```

```
    public static int quotient( int x, int y ) {
```

```
        return x / y ; // possible division by zero
```

```
    } // end method quotient
```

```
public static void main( String args[ ] ) {
```

```
    Scanner input = new Scanner( System.in ); // scanner for input
```

```
    System.out.print( "Please enter an integer numerator: " );
```

```

int x = input.nextInt( );
System.out.print( "Please enter an integer denominator: " );
int y = input.nextInt();
int result = quotient( x, y );
System.out.printf( "\nResult: %d / %d = %d\n", x, y, result );
} // end main
} // end class DivideByZeroNoExceptionHandling

```

عندما نقسم البسط على المقام الذي يساوي العدد 5 نحصل على الخرج التالي :

RESULT

```

Please enter an integer numerator: 50
Please enter an integer denominator: 5
Result: 50 / 5 = 10
Process completed.

```

عندما نقسم البسط على المقام الذي يساوي العدد صفر نحصل على الخرج التالي:

RESULT

```

Please enter an integer numerator: 50
Please enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at DivideByZeroNoExceptionHandling.quotient(e1.java:9)
    at DivideByZeroNoExceptionHandling.main(e1.java:21)
Result: 50 / 5 = 10
Process completed.

```

عندما نقسم البسط على المقام الذي يساوي السلسلة المحرفية hello

RESULT

```
Please enter an integer numerator: 50
Please enter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:840)
    at java.util.Scanner.next(Scanner.java:1461)
    at java.util.Scanner.nextInt(Scanner.java:2091)
    at java.util.Scanner.nextInt(Scanner.java:2050)
    at DivideByZeroNoExceptionHandling.main(e1.java:19)
```

Process completed.

تُمكن لغة Java المبرمج من التقاط الخطأ في حال حدوثه وإنجاز بعض الأفعال الملائمة ، وتتضمن هذه الأفعال إمكانية إيقاف البرنامج أولاً ، وتُعالج الخطأ الذي حدث في البرنامج السابق وذلك باستخدام بنية جديدة تُدعى الكتلة `try - catch block` والتي تُتيح للمبرمج إمكانية التقاط الخطأ والاستمرار في التنفيذ . تحدث أخطاء زمن التنفيذ لعدة أسباب منها : يمكن أن يُدخل المستخدم بيانات غير صالحة أو ربما يحاول البرنامج فتح ملف غير موجود أو ربما ينقطع الاتصال الشبكي أو ربما يحاول البرنامج الوصول إلى خارج حدود المصفوفة ، وعندما يحدث الخطأ أثناء التنفيذ ترمي لغة Java استثناءً . نُعيد كتابة المثال (1-11) باستخدام الكتلة `try- catch` كما هو موضح في المثال (2-11).

مثال (2-11)

```
// An application that attempts to divide by zero.
import java.util.Scanner;
public class DivideByZeroNoExceptionHandling {
    // demonstrates throwing an exception when a divide-by-zero Occurs
    public static int quotient( int x, int y ) {
        return x / y; // possible division by zero
    } // end method quotient
```

```

public static void main( String args[ ] ) {
Scanner input = new Scanner( System.in ); // scanner for input
System.out.print( "Please enter an integer numerator: " );
    int x = input.nextInt( );
    System.out.print( "Please enter an integer denominator: " );
    int y = input.nextInt();
try {
int result = quotient( x, y );
    System.out.printf("\\nResult: %d / %d = %d\\n", x, y, result );
}
catch( Exception ex ) {
    System.out.println ( " Error : " + ex.getMessage() );
}
System.out.println ( " Excution continues ");
} // end main
} // end class DivideByZeroNoExceptionHandling

```

عند تنفيذ البرنامج نحصل على الخرج التالي :

RESULT

```

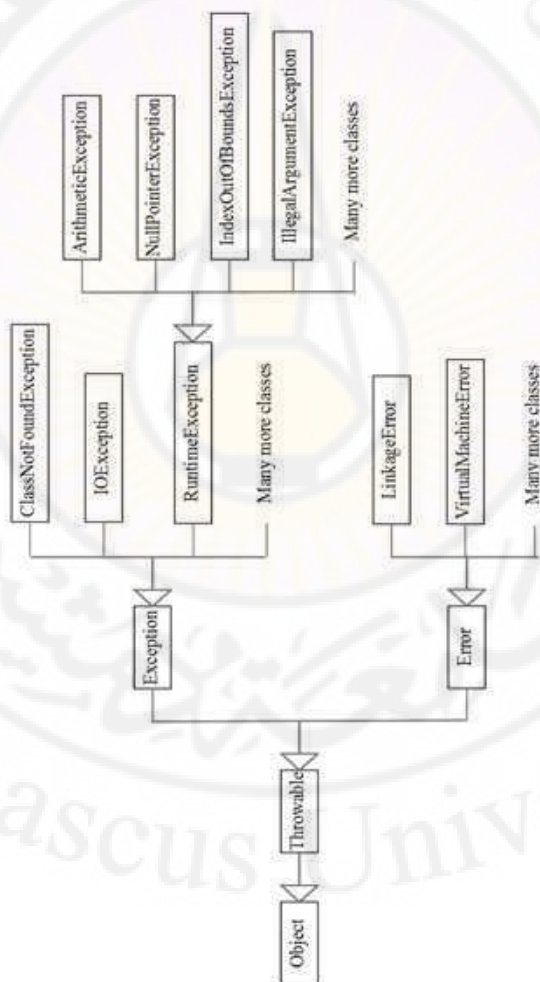
Please enter an integer numerator: 5
Please enter an integer denominator: 0
Error: / by zero
Execution continues
Process completed.

```

١١-٢-١ - صفوف الاستثناءات

الاستثناءات عبارة عن أغراض تنتمي إلى صفوف مشتقة من الصف **Throwable** ، ويوجد الصف **Throwable** في الحزمة **java.lang** بينما توجد الصفوف المشتقة من الصف **Throwable** في حزم متنوعة ، ويتم تضمين الأخطاء المتعلقة بعناصر **GUI** في الحزمة **java.awt** وتضمن الاستثناءات

العديدية في الحزمة java.lang . ونستطيع إنشاء صفوف للاستثناءات الخاصة بنا بواسطة الوراثة من الصف Throwable أو الوراثة من الصفوف المشتقة من الصف Throwable . ويبين الشكل (١-١١) بعض صفوف الاستثناءات المعرفة في لغة Java . إن جميع الاستثناءات التي يتم رميها في Java هي إما أغراض تنتمي إلى الصفوف الموجودة في الشكل (١-١١) أو أغراض تنتمي للصفوف المشتقة من هذه الصفوف .



الشكل (١-١١) - يبين بعض صفوف الاستثناءات

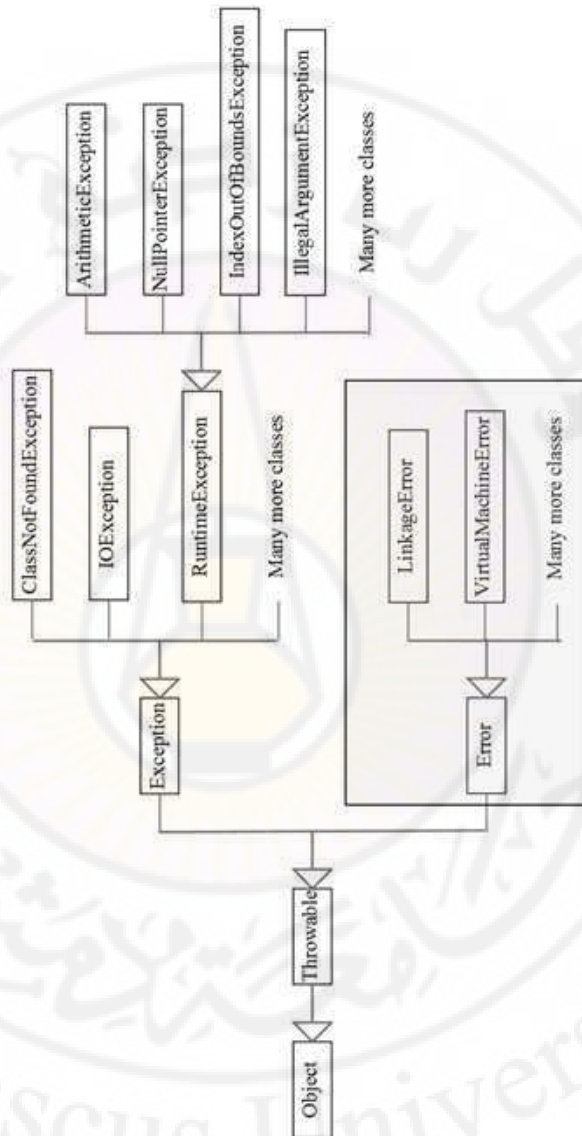
١- الصف Error :

يُعبّر الصف Error عن أخطاء النظام الداخلية ، وعند حدوث خطأ من هذا النوع يجب تنبيه المستخدم على أن البرنامج ينتهي بشكل طبيعي. وعندما يفشل الربط الميكانيكي أو يحدث فشل من نوع آخر في آلة جافا الافتراضية JVM ، فإن الآلة ترمي الخطأ Error . ويبين الشكل (١١-٢) الأخطاء الناتجة عن الصف Error . لنأخذ الصفوف المشتقة عن الصف Error كمثال عن هذه الأخطاء :

- الصف LinkageError : يحدث هذا الخطأ عندما يدل على أن الصف الأول يعتمد بعض الشيء على الصف الثاني بشكل غير متوافق وذلك بعد ترجمة الصف الأول الذي يعتمد على الصف الثاني .
- الصف VirtualMachineError : يحدث هذا الخطأ عندما يدل هذا الصف على أن آلة جافا الافتراضية JVM معطلة أو تعمل خارج نطاق المصادر التي تحتاجها من أجل الاستمرار في التنفيذ .
- الصف AWTError : يحدث هذا الخطأ عند وقوع خطأ قاتل في نظام زمن التنفيذ التابع لواجهة المستخدم الرسومية GUI .
- الصف AssertionError : يحدث هذا الخطأ عند فشل التأكيد .
assertion .

٢- الصف Exception :

يُعبّر الصف Exception عن الأخطاء التي يتسبب بها البرنامج والأخطاء الناتجة عن الظروف الخارجية ومن الممكن التقاط هذه الأخطاء ومعالجتها . معظم البرامج ترمي أو تلتقط استثناءات مشتقة من هذا الصف ، وإن هذا الصف يُشير إلى أن مشكلة ما قد حصلت ، ولكن المشكلة ليست مشكلة نظام خطيرة .



الشكل (١١-٢) - يبين الأخطاء الناتجة عن الصف Error

يملك هذا الصف عدة صفوف مشتقة معرفة في حزم Java كما هو مبين في الشكل (١١-٣) ، وهذه الصفوف تُشير إلى أنواع مختلفة من الاستثناءات التي يمكن أن تحدث .و كمثال عن هذه الأخطاء :

➤ الصف `ClassNotFoundException` : يحدث هذا الاستثناء عندما تُحاول استخدام صف غير موجود .

➤ الصف `ClassNotSupportedException` : يحدث هذا الاستثناء عندما تُحاول نسخ غرض ينتمي إلى صف آخر لا يُنفذ الواجهة `Cloneable` .

➤ الصف `RuntimeException` : يُعبر عن الأخطاء البرمجية خلال زمن التنفيذ .

➤ الصف `IOException` : يُعبر عن الاستثناءات المتعلقة بعمليات الدخل والخرج ، مثلا : قراءة دخل غير صالح أو الاستمرار بالقراءة على الرغم من انتهاء الملف أو فتح ملف غير موجود ، والصفوف المشتقة منه هي :

- الصف `InterruptedException` .

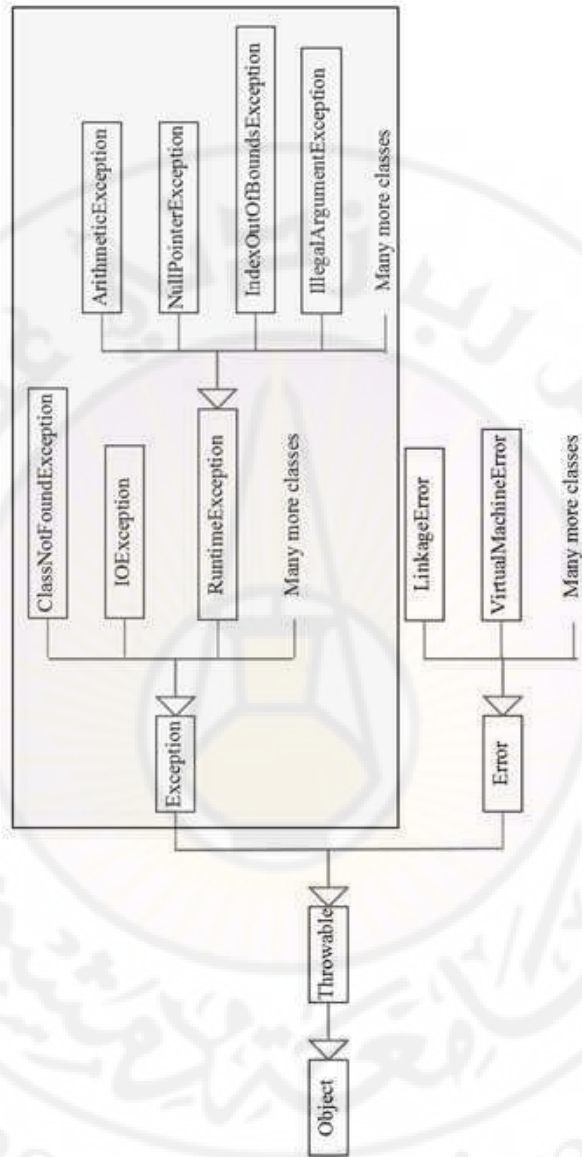
- الصف `EOFException` .

- الصف `FileNotFoundException` .

➤ الصف `AWTException` : يُعبر عن الاستثناءات المتعلقة بعناصر GUI

➤ الصف `IllegalAccessException` : يشير إلى طريقة ما لا يمكن إيجادها لسبب من الأسباب .

➤ الصف `NegativeArraySizeException` : يُشير إلى أن البرنامج يُحاول إنشاء مصفوفة بحجم سالب .



الشكل (١١-٣) - يبين الأخطاء الناتجة عن الصف Exception

٣- الصف RuntimeException :

يعبر الصف RuntimeException عن الأخطاء البرمجية ، مثل الأخطاء العددية وتنفيذ عملية تحويل غير صحيحة على النمط أو الوصول إلى خارج حدود المصفوفة ، وكمثال عن الصفوف المشتقة من الصف RuntimeException هي :

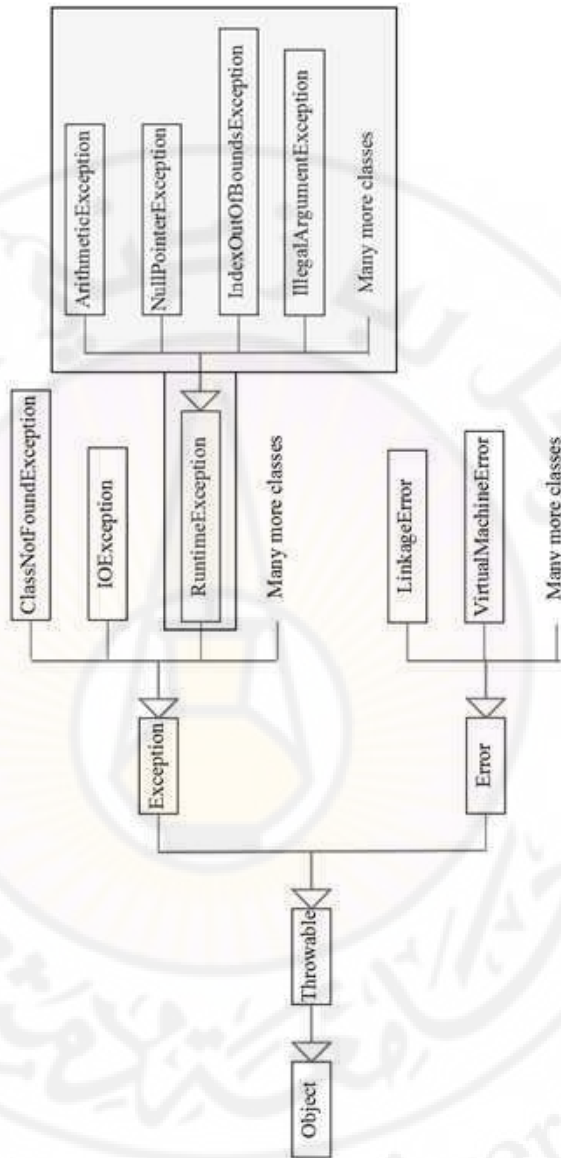
➤ الصف ArithmeticException : يحدث هذا الاستثناء عندما يتعلق هذا الصف بالعمليات الحسابية الصحيحة ، حيث تتعامل Java مع العمليات الحسابية الصحيحة بشكل مختلف عن تعاملها مع العمليات الحسابية ذات الفاصلة العائمة ، مثلاً ، تُعد عملية القسمة على الصفر غير مقبولة في العمليات الحسابية الصحيحة وتؤدي إلى رمي الاستثناء ArithmeticException بينما لا ترمي العمليات ذات الفاصلة العائمة استثناء عند القيام بمثل هذه العملية إذ يُعد ناتج القسمة على الصفر لانهائي .

➤ الصف NullPointerException : يتم رمي هذا الاستثناء عند محاولة الوصول إلى غرض ما من خلال متحول يملك القيمة null بدلاً من مرجع إلى هذا الغرض .

➤ الصف IndexOutOfBoundsException : يتم رمي هذا الاستثناء إذا أصبح فهرس المصفوفة يحمل قيمة تقع خارج مجال المصفوفة .

➤ الصف IllegalArgumentException : يتم رمي هذا الاستثناء في حال تمرير وسيط غير شرعي أو غير مناسب إلى التابع .

➤ يبين الشكل (١١-٤) استثناءات الصف RuntimeException .



الشكل (٤-١١) - يبين الأخطاء الناتجة عن الصف RuntimeException

١١-٢-٢- الاستثناءات Checked & Unchecked

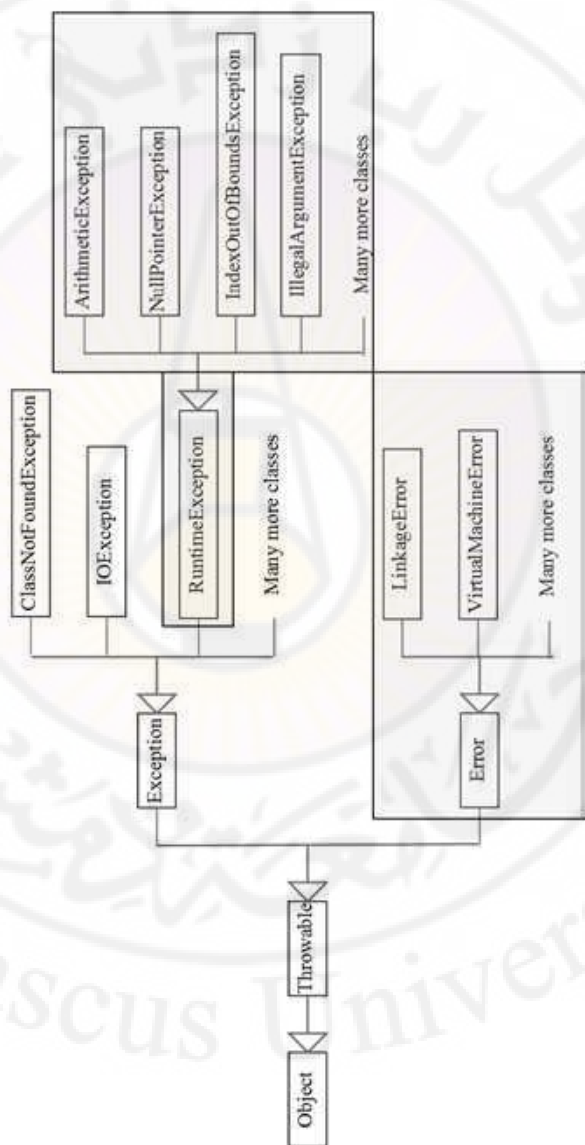
يُعرف الصفان RuntimeException و Error والصفوف المشتقة منهما بالاستثناءات غير المفحوصة unchecked exception وتُعرف بقية الصفوف بالاستثناءات المفحوصة ، أي الاستثناءات التي يُجبر المترجم المبرمج على فحصها والتعامل معها . وتُعتبر في معظم الحالات الاستثناءات غير المفحوصة عن أخطاء البرمجة المنطقية التي تكون غير قابلة للإصلاح ، فعلى سبيل المثال : يتم رمي الاستثناء NullPointerException عند محاولة الوصول إلى غرض ما عبر متحول مرجعي لا يملك مرجع إلى هذا الغرض ، كما يتم رمي الاستثناء IndexOutOfBoundsException عند محاولة الوصول إلى عنصر يقع خارج حدود المصفوفة ، ونلاحظ أن الأسباب التي أدت إلى رمي الاستثناءين السابقين هي أخطاء منطقية يجب تصحيحها ضمن البرنامج . الأخطاء غير المفحوصة يمكن أن تحدث في أي مكان ضمن البرنامج ، ولكي نتجنب الاستعمال الزائد والمتعب لكثلة try - catch فإن لغة Java لا تُطالب بكتابة برنامج من أجل النقاط الاستثناءات غير المفحوصة بل جعلت هذا الأمر اختياريًا يعود إلى المبرمج . ويبين الشكل (١١-٥) الاستثناءات غير المفحوصة .

١١-٣- التعرف على كيفية معالجة الاستثناءات

يعتمد نموذج معالجة الاستثناء على ثلاثة عمليات : التصريح عن الاستثناء ورمي الاستثناء والتقاط الاستثناء . ويبين الشكل (١١-٦) نموذج معالجة الاستثناءات في Java ، ولمعالجة أي استثناء هناك ثلاثة أقسام :

١. عبارة try وفيها نضع كتلة التعليمات التي يمكن أن تسبب الاستثناء .
٢. عبارة catch وفيها نضع التعليمات التي يمكن أن تُعالج نوعاً خاصاً من الاستثناءات ، وهذه التعليمات سيتم تنفيذها فقط إذا حدث استثناء

- ضمن التعليمات الموجودة في القسم try .
٣. _عبارة finally ويمكننا أن نضع فيها التعليمات التي نريد تنفيذها بغض النظر إن حدث استثناء أم لم يحدث .



الشكل (١١-٥) - يبين الاستثناءات غير المفحوصة unchecked exception

أما الصيغة العامة للاستثناء فهي على الشكل التالي :

```
try {  
    statement (s) ;  
}  
catch( Exception type name ) {  
    statement (s) ;  
}  
finally {  
    statement (s) ;  
}
```

١١-٣-١- التصريح عن الاستثناءات

Declaring Exceptions

كل عبارة في لغة Java يتم تنفيذه حالياً تعود إلى تابع ما method، يستدعي مفسر Java التابع () main من أجل تطبيقات Java ، بينما يستدعي مستعرض الويب باني البرميج الافتراضي ثم التابع () init من أجل برميجات Java ، ويجب على كل تابع method الإعلان عن الاستثناءات غير المفحوصة التي يمكن أن يرميها ، وتُعرف هذه العملية بالتصريح عن الاستثناء . بما أن أخطاء النظام وأخطاء زمن التنفيذ يمكن أن تحدث في أي شيفرة code لذلك لا تطلب لغة Java التصريح عن الاستثناءات التي تنتمي للصفين Error و RuntimeException (الاستثناءات غير المفحوصة) بشكل صريح في التابع. على أي حال يجب التصريح بشكل صريح عن جميع الاستثناءات الأخرى الذي يمكن للتابع أن يرميها وذلك ضمن التصريح عنه ، وذلك لكي يتم إعلام مُستدعي هذا التابع بحدوث الاستثناء .

□ نستخدم الكلمة المحجوزة throws من أجل التصريح عن الاستثناء ضمن

التصريح عن التابع كما يلي :

public void myMethod() throws IOException

□ وتدل الكلمة المحجوزة throws على أن التابع myMethod() ربما يرمي الاستثناء IOException ، وربما يرمي التابع عدة استثناءات لذلك يمكننا إضافة قائمة من الاستثناءات بعد الكلمة المحجوزة throws كما يلي :

public void myMethod() throws IOException , OtherException

إن مستدعي هذا التابع يجب أن يعلم أن هذا التابع يرمي استثناء لكي يحدد بدقة ما سيفعل بشأنه ، لذلك يجب تحديد الاستثناءات الذي يمكن أن يقوم التابع برميها عند تعريف ذلك التابع . ولتوضيح ذلك سنعيد كتابة التابع div مستخدمين طريقة التصريح عن الاستثناء وليس النقاط الاستثناء .

مثال (3-11)

```
public class divException {  
    public static double div(int x, int y) throws ArithmeticException  
    {  
        return x/y;  
    }  
    public static void main ( String arg[ ]) {  
        System.out.println(div(10,0));  
    } //end main  
} // end class
```

عند تنفيذ البرنامج نحصل على الخرج التالي :

RESULT

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at divException.div(divException.java:5)  
at divException.main( divException.java:9)  
Process completed.
```

نلاحظ أننا صرحنا عن الاستثناء المراد رميه من خلال الكلمة المحجوزة **throws** ، ولم يقوم التابع بأي عملية معالجة للخطأ في حال حدوثه ، وهنا يجب على مستدعي هذا التابع أن يقوم بمعالجة الخطأ الذي قد يرميه هذا التابع .

١١-٣-٢- رمي الاستثناءات

Throwing Exceptions

يستطيع البرنامج الذي يكتشف وجود خطأ ما بأن يُنشئ غرضاً من نمط الاستثناء المناسب ثم يرميه ، وتُعرف هذه العملية برمي الاستثناءات Throwing Exceptions ، أي :

```
throw new TheException( ) ;
```

أو

```
TheException ex = new TheException( );  
throw ex;
```

لنأخذ المثال التالي : لنفرض أن برنامجنا اكتشف أن الوسيط الممرر إلى تابع ما يخرق قواعد هذا التابع (مثلاً تنص قواعد التابع على تمرير وسيط موجب في حين تم تمرير وسيط سالب) عندها يستطيع برنامجنا إنشاء غرض من الاستثناء **IllegalArgumentException** ثم رميه كما يلي :

مثال (4-11)

```
/** Set a new radius */
```

```
public class TestRadius{  
    private double radius;  
    public void setRadius(double newRadius)  
        throws IllegalArgumentException {  
        if (newRadius >= 0)  
            radius = newRadius;  
        else  
            throw new IllegalArgumentException(  

```

```

        "Radius cannot be negative");
    }
    } // end class TestRadius
public class testmain{
    public static void main(String arg[ ]) {
        TestRadius ob = new TestRadius( );
        ob.setRadius(-10) ;
    } //end main
} //end class

```

عند تنفيذ البرنامج يظهر الخرج التالي :

RESULT

Exception in thread "main" java.lang.IllegalArgumentException:
 Radius cannot be negative
 at Test2.setRadius(TestRadius.java:9)
 at testmain.main(TestRadius.java:18)
 Process completed.

١١-٣-٣ - النقاط الاستثناءات

Catching Exceptions

عندما يتم رمي استثناء ما فإنه من الممكن التقاطه ومعالجته باستخدام الكتل try - catch كما يلي :

```

try {
    statements; // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
    handler for exception1;
}
catch (Exception2 exVar2) {
    handler for exception2;
}

```



```

}
...
...
catch (ExceptionN exVarN) {
    handler for exceptionN;
}

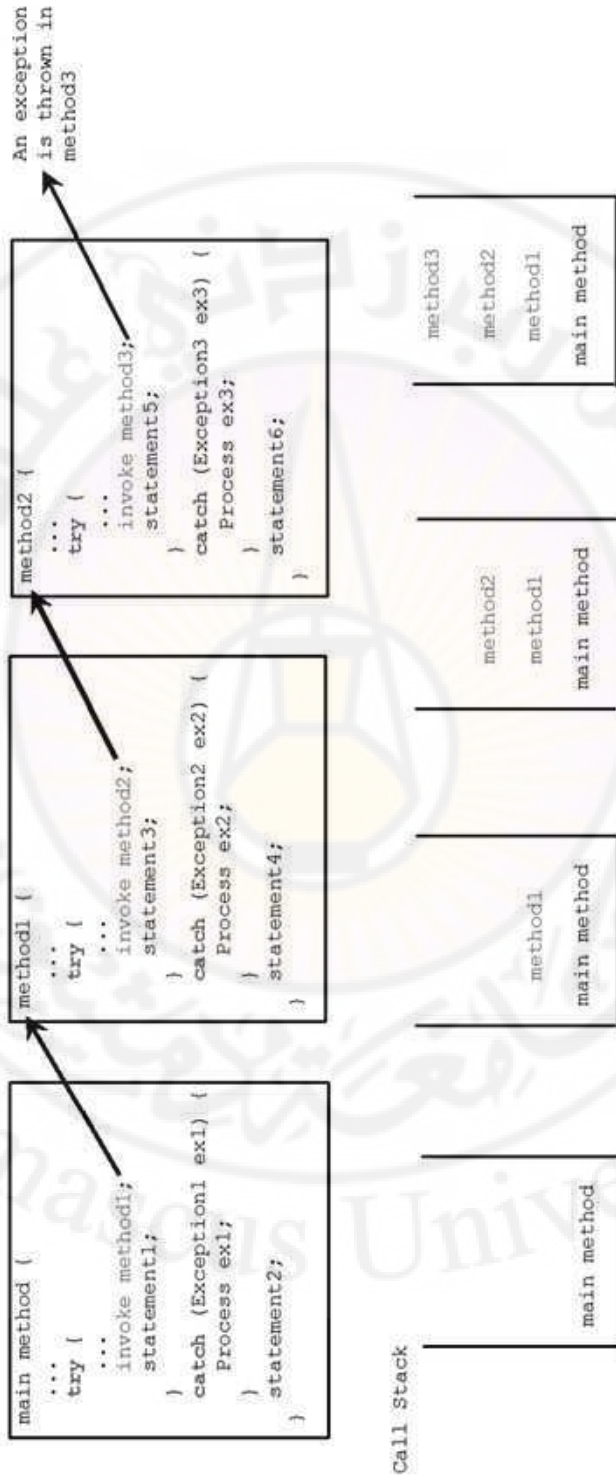
```

➤ إذا لم تظهر الاستثناءات أثناء تنفيذ العبارة try عندها يتم تجاوز عبارات catch الموجودة في البرنامج .

➤ إذا رمت أي عبارة موجودة ضمن الكتلة catch - try استثناء عندها يتم تجاوز جميع العبارات المتبقية في العبارة try وتنفيذ معالجة جديدة من أجل البحث عن الشيفرة الخاصة بمعالجة الاستثناء . ويطلق على الشيفرة البرمجية التي تُعالج الاستثناء بمعالج الاستثناء exception handler ، ويتم البحث عن معالج الاستثناء ضمن سلسلة استدعاءات التوابع methods وبشكل تراجمي حيث تبدأ عملية البحث من التابع الحالي .

➤ يتم فحص كل عبارة catch حسب ترتيب وجودها من أول عبارة catch إلى الأخيرة ، لمعرفة فيما إذا كان نمط غرض الاستثناء الذي تم رميه يمثل غرضاً من صف الاستثناء الموجود في العبارة catch ، وإذا تحقق ذلك يتم إسناد غرض الاستثناء إلى المتحول المصرح عنه في العبارة catch ثم تنفيذ هذه الشيفرة الموجودة ضمن العبارة catch ، أما إذا لم يتحقق ما سبق ، أي لم نستطع إيجاد معالج الاستثناء في هذا التابع ، عندها تخرج Java من هذا التابع وتمرر الاستثناء إلى التابع الذي يستدعي التابع الحالي وتكرر الخطوات نفسها السابقة من أجل البحث عن المعالج .

➤ إذا لم يوجد معالج للاستثناء ضمن سلسلة التوابع التي تم استدعاؤها عندها ينهي البرنامج ويطبّع رسالة خطأ على الشاشة console تُدعى عملية إيجاد المعالج بالتقاط الاستثناء catching an exception .



الشكل (٦-١١) - يبين كيفية التقاط الاستثناء أو تمريره للتابع main

➤ يوضح المثال (11 - 5) التقاط الاستثناء ، ويحتوي البرنامج على تابع بوسيطين ويقوم بحساب قسمة الوسيط x على y ويرمي استثناء `ArithmeticException` في حال كون $y = 0$ ويعيد القيمة `NaN` التي تمثل قيمة اللانهاية الرياضية .

مثال (5-11)

```
public class DivExceptionHandling {
    // demonstrates catching an exception when a divide-by-zero occurs
    public static double div( int x, int y ) {
        try {
            return x/y ; // possible division by zero
        }
        catch( ArithmeticException ex ) {
            return Double.NaN ;
        }
    } // end div
    public static void main( String args[ ] ) {
        System.out.println ( " Execution continues " + div(10, 0));
    } // end main
} // end class DivExceptionHandling
```

عند تنفيذ البرنامج نحصل على الخرج التالي :

RESULT

Execution continues NaN
Process completed.

- لنفرض ان التابع `main` يستدعي التابع `method1` والتابع `method1` يستدعي التابع `method2` والتابع `method2` يستدعي التابع `method3` كما هو مبين في الشكل (11-6) ونلاحظ ما يلي :

١. إذا لم يستطع التابع method3 معالجة الاستثناء عندها يتم الخروج من التابع method3 ويعود التحكم إلى التابع method2 . وإذا كان نمط الاستثناء هو Exception3 عندها يتم التقاطه بواسطة العبارة catch من أجل معالجة الاستثناء ex3 في التابع method2 ، عندها يتم تجاوز العبارة statment5 وتنفيذ العبارة statment6 .

٢. إذا كان نمط الاستثناء هو Exception2 عندها يتم الخروج من التابع method2 ويعود التحكم إلى التابع method1 ثم يتم التقاط الاستثناء بواسطة العبارة catch من أجل معالجة الاستثناء ex2 في التابع method1 ، عندها يتم تجاوز العبارة statment3 وتنفيذ العبارة statment4 .

٣. إذا كان نمط الاستثناء هو Exception1 عندها يتم الخروج من التابع method1 ويعود التحكم إلى التابع main ثم يتم التقاط الاستثناء بواسطة العبارة catch من أجل معالجة الاستثناء ex1 في التابع main ، عندها يتم تجاوز العبارة statment1 وتنفيذ العبارة statment2 .

٤. إذا كان نمط الاستثناء لا ينتمي إلى الاستثناءات التالية :
Exception1, Exception2, Exception3 عندها لا يتم التقاط هذا الاستثناء وينتهي البرنامج ، عندها لا يتم تنفيذ العبارتين statment1 statment2 .

يحتوي غرض الاستثناء على معلومات مهمة حول الاستثناء الذي حدث ، يمكننا استخدام التوابع التالية والمعرفة في الصف java.lang.Throwable من أجل الحصول على المعلومات المتعلقة بالاستثناء . ومنها :

□ التابع () public String getMessage() : يعيد هذا التابع رسالة مفصلة عن الغرض القابل للرمي .

- التابع () `public String toString()` : يعيد هذا التابع مجموع ثلاثة نصوص ، النص الأول هو الاسم الكامل لصف الاستثناء أما النص الثاني : " نقطتان ومسافة فارغة والنص الثالث التابع () `getMessage()`
- التابع () `public String getLocalizedMessage()` : يعيد وصف محلي عن الغرض القابل للرمي .
- التابع () `public void printStackTrace()` : يطبع الغرض القابل للرمي والمعلومات المتعلقة بملاحقته (ضمن المكس) على الشاشة `console` .
- يشرح المثال (6-11) عملية التصريح عن الاستثناءات ورميها ومعالجتها من خلال تعديل التابع `setRadius` والمعرف في الصف `Circle` ، يرمي التابع `setRadius` استثناءً إذا كانت قيمة نصف القطر سالبة .

مثال (6-11)

// Circle.java: setRadius throws an exception

```
public class Circle {
    /** The radius of the circle */
    private double radius;
    /** The number of the objects created */
    private static int numOfObjects = 0;
    /** Default constructor */
    public Circle() {
        this(1.0);
    }
    /** Construct a circle with a specified radius */
    public Circle(double newRadius) {
        setRadius(newRadius);
        numOfObjects++;
    }
    /** Return radius */
```



```

public double getRadius( ) {
    return radius;
}
/** Set a new radius */
public void setRadius(double newRadius)
    throws IllegalArgumentException {
    if (newRadius >= 0)
        radius = newRadius;
    else
        throw new IllegalArgumentException(
            "Radius cannot be negative");
}
/** Return numObjects */
public static int getNumOfObjects( ) {
    return numObjects;
}
/** Return the area of this circle */
public double findArea( ) {
    return radius * radius * 3.14159;
}
} // end class circle

public class TestCircleWithException {
    /** Main method */
    public static void main(String[ ] args) {
        try {
            Circle c1 = new Circle(5);
            Circle c2 = new Circle(-5);
            Circle c3 = new Circle(0);
        }
        catch (Exception ex) {

```

```

        System.out.println(ex);
    }
    System.out.println("Number of objects created: " +
        Circle.getNumOfObjects( ) );
} // end main
} // end class TestCircleWithException

```

عند تنفيذ البرنامج نحصل على الخرج التالي :

RESULT

java.lang.IllegalArgumentException: Radius cannot be negative

Number of objects created: 1

Process completed.

يُنشئ برنامج الاختبار ثلاثة أغراض C1,C2,C3 من أجل فحص كيفية معالجة الاستثناءات ، ويؤدي استدعاء الباني new Circle(-5) إلى استدعاء التابع setRadius والذي يرمي بدوره الاستثناء IllegalArgumentException وذلك لأن قيمة نصف القطر سالبة ، ويتوافق نمط الغرض ex الموجود في العبارة catch مع غرض الاستثناء الذي تم رميه بواسطة التابع setRadius لذل يُلقط هذا الاستثناء في العبارة catch . يقوم معالج الاستثناء ببساطة بطباعة رسالة قصيرة (ex.toString()) حول الاستثناء باستخدام العبارة (ex) System.out.println(). نلاحظ أن تنفيذ البرنامج يستمر على الرغم من حدوث الاستثناء ، وإذا لم يلتقط الاستثناء عندها ينتهي البرنامج بشكل مفاجئ .

١١-٤ - إعادة رمي الاستثناءات

Rethrowing Exceptions

عندما يحدث الاستثناء في التابع method فإن التابع يخرج مباشرة في حال عدم التقاط هذا الاستثناء ، لكن لنفرض أننا نحتاج إلى إنجاز بعض المهام قبل الخروج من التابع ، ما هو الحل ؟؟

إحدى الطرائق المستخدمة من أجل تحقيق هذه الغاية هي التقاط هذا الاستثناء في التابع ثم إعادة رميه إلى المعالج الحقيقي باستخدام البنية التالية :

```
try {  
    statements;  
}  
catch (TheException ex) {  
    perform operations before exits;  
    throw ex;  
}
```

تقوم العبارة **throw ex** بإعادة رمي الاستثناءات بحيث يعطي الفرصة لمعالجات أخرى بالتعامل مع هذا الاستثناء .

١١-٥ - العبارة finally

The finally Clause

نحتاج في بعض الأحيان إلى تنفيذ بعض كتل البرنامج بغض النظر عن حدوث الاستثناء أو التقاطه ، وتكون صيغة العبارة **finally** كما يلي :

```
try {  
    statements;  
}  
catch (TheException ex) {  
    handling ex;
```

```

}
finally {
    finalStatements;
}

```

يتم تنفيذ التعليمات الموجودة ضمن الكتلة **finally** بغض النظر عن حدوث الاستثناء في الكتلة **try** أو التقاطه ن كما هو موضح في المثال (7-11) .

مثال (7-11)

```

public class DivExceptionHandling {
    /* demonstrates throwing an exception when a divide-by-zero occurs */
    public static void div( int x, int y ) {
        try {
            int z ;
            z = x / y ; // possible division by zero
            System.out.println( " z = " + z ) ;
        }
        catch(Exception ex ) {
            System.out.println( ex.toString( ) ) ;
        }
        finally {
            System.out.println( " mission is finished " ) ;
        }
    } // end div
    public static void main( String args[ ] ) {
        div(10, 0) ;
    } // end main
} // end class DivExceptionHandling

```

عند تنفيذ البرنامج نحصل على الخرج التالي :

RESULT

```
java.lang.ArithmeticException: / by zero  
mission is finished  
Process completed.
```

لنأخذ الحالات التالية :

١. إذا لم يظهر الاستثناء في الكتلة try فإنه يتم تنفيذ الكتلة الموجودة في العبارة finally والعبارة التي تقع بعد try .
٢. إذا تسببت إحدى العبارات الموجودة ضمن الكتلة try بحدوث استثناء من الممكن التقاطه في العبارة catch عندها يتم تجاوز بقية العبارات في الكتلة try وتنفيذ العبارتين catch و finally .
 - إذا لم تقم العبارة catch بإعادة رمي الاستثناء عندها يتم تنفيذ العبارة التالية التي تقع بعد العبارة try .
 - إذا كانت العبارة catch تعيد رمي الاستثناء سيتم تمرير الاستثناء إلى مُستدعي هذا التابع
٣. إذا تسببت إحدى العبارات الموجودة ضمن الكتلة try بحدوث استثناء لا يمكن التقاطه في العبارة catch عندها يتم تجاوز بقية العبارات في الكتلة try وتنفيذ العبارة finally وتمرير الاستثناء إلى مُستدعي التابع الحالي .

١١-٦- متى نستخدم الاستثناءات

When to Use Exceptions

يجب الحذر عند استخدام الاستثناءات ، حيث تحتاج عادة معالجة الاستثناءات أكثر مما تتطلبه البرمجة العادية ، إذ تتطلب إنشاء غرض جديد من أجل الاستثناء واستعادة مكدس الاستدعاء ونشر الاستثناء ضمن سلسلة التوابع

التي تم استدعاؤها وذلك من أجل البحث عن المعالج . إذا كنا نرغب في أن تتم معالجة الاستثناء بواسطة مستدعيه عندها يجب إنشاء غرض من الاستثناء ورميه ، أما إذا كنا نستطيع التعامل مع الاستثناء في التابع الذي حدث فيه فأنا لا نحتاج إلى رميه . وبشكل عام يتم رمي الاستثناءات العامة التي من الممكن أن تحدث ضمن صفوف متعددة في المشروع إلى صفوف الاستثناءات ، ومن الأفضل معالجة الأخطاء البسيطة التي من الممكن أن تحدث ضمن توابع منفصلة بشكل محلي وبدون رمي الاستثناءات .

أين يجب استخدام الكتلة try - catch ؟

يجب استخدام الكتلة try - catch في الأوضاع التي تكون فيها الأخطاء غير متوقعة ، ولا تُستخدم الكتلة try - catch للتعامل مع الأوضاع البسيطة والمتوقعة . لنأخذ على سبيل المثال مجموعة التعليمات التالية في البرنامج :

```
try {  
    System.out.println(refVar.toString());  
}  
catch (NullPointerException ex) {  
    System.out.println("refVar is null");  
}
```

من الأفضل استخدام مجموعة التعليمات التالية بدلاً من مجموعة التعليمات السابقة في البرنامج :

```
if (refVar != null)  
    System.out.println(refVar.toString());  
else  
    System.out.println("refVar is null");
```

في بعض الأحيان يكون من الصعب التمييز بين الأوضاع الاستثنائية والأوضاع المتوقعة ، لذلك فإن النقطة الجوهرية المقصودة في المثال البرمجي السابق هي

عدم إساءة استعمال معالجة الأحداث في استخدامها مع الاختبارات المنطقية
البسيطة .



الفصل الثاني عشر

تعددية المسالك

MultiThreading





تعددية المسالك

MultiThreading

١٢-١- مقدمة

لقد تم تصميم وحدة المعالجة المركزية CPU في الحاسوب لتقوم بتنفيذ مهمة واحدة في لحظة واحدة ، وهذا ما ينعكس على تنفيذ البرامج حيث يتم تنفيذ تعليمة واحدة في لحظة محددة ، ولكن كيف يمكن تشغيل أكثر من مهمة و تنفيذ أكثر من برنامج في الحاسوب في الوقت نفسه ؟ .

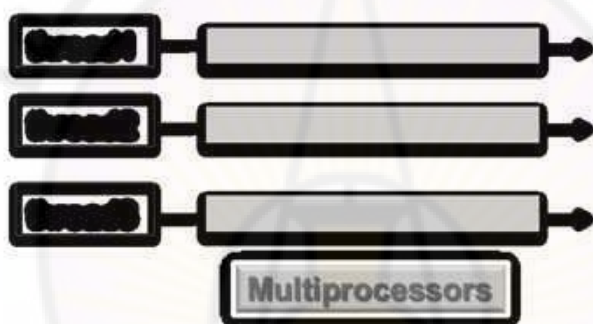
في الحقيقة يتم بفضل تقنية تُدعى تعددية المسالك Multithreading أو تعددية المهام Multitasking ، والتي تسمح للمعالج بنقل التنفيذ بين عدة مهام، حيث يتم وضع كل مهمة في مسلك ، ويتم التنقل بين المسالك بسرعة كبيرة ما يجعلنا نظن أنها تُنفذ معاً ، إلا أنه في الواقع يجري تنفيذ تعليمة واحدة في لحظة واحدة . إحدى أهم الصفات في لغة جافا هو دعمها المضمن لتعددية المسالك ، فتعددية المسالك هي القدرة على تنفيذ عدة مهام بشكل متساير concurrently ضمن البرنامج الواحد. تحتاج في معظم لغات البرمجة إلى استدعاء إجراءات وتوابع معتمدة على النظام من أجل تنفيذ تعددية المسالك ، وسنتعرف في هذا الفصل على مفهوم المسالك وكيفية تطوير برامج ذات مسالك متعددة في لغة Java .

١١-٢- مفهوم تعددية المسالك

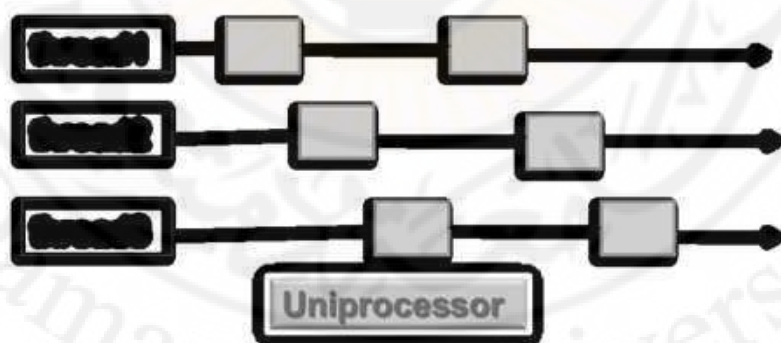
المسلك عبارة عن تدفق للتنفيذ من أجل مهمة ما في البرنامج ، ويمكن إطلاق عدة مسالك من البرنامج الواحد بشكل متساير بحيث من الممكن تنفيذ تلك المسالك بشكل متواقت في الأنظمة المتعددة المعالجات multiprocessor ، كما هو موضح في الشكل (١١-١) .

تتشارك المسالك المتعددة في الأنظمة أحادية المعالج بوقت المعالج كما هو مبين في الشكل (١١-٢) . حيث يكون نظام التشغيل مسؤولاً عن عملية الجدولة scheduling وعن تخصيص المصادر لتلك المسالك ، ويُعد هذا التنظيم عملياً وذلك لأن المعالج يكون في حالة خمول في معظم الوقت ، وتجعل تعددية المسالك البرنامج أكثر تفاعلية وسريع الاستجابة بالإضافة إلى تحسين الأداء .

مثال : يتيح لنا معالج النصوص word إمكانية طباعة ملف أو حفظه بينما نقوم بعملية الكتابة .



الشكل (١٢-١) - يبين تنفيذ المسالك المتعددة باستخدام معالجات متعددة



الشكل (١٢-١) - يبين تنفيذ المسالك المتعددة باستخدام معالجات متعددة

في بعض الحالات من الممكن أن تنفذ البرامج متعددة المسالك بشكل أسرع من البرامج أحادية المسلك حتى ضمن الأنظمة أحادية المعالج .

تقدم لغة Java دعماً جيداً من أجل عملية إنشاء وتنفيذ المسالك ومن أجل عملية إغلاق المصادر بهدف منع التضارب ، ويمكن إنشاء مسالك إضافية من أجل تنفيذ مهام متسايرة في البرنامج ، وكل مسلك جديد هو غرض من صف class يحقق الواجهة Runnable أو يرث صفاً يحقق الواجهة Runnable ويُدعى هذا الغرض الجديد بالغرض القابل للتنفيذ Runnable Object . يمكننا صنع المسلك إما من خلال وراثة الصف Thread أو من خلال تحقيق الواجهة Runnable وكلاهما معرف في الحزمة java.lang ، وفي الحقيقة يقوم الصف Thread بتحقيق الواجهة Runnable .

١٢-٣ - إنشاء المسالك من خلال وراثة الصف Thread

يحتوي الصف Thread على عدة بواني من أجل صنع المسالك بالإضافة إلى العديد من التوابع المفيدة من أجل التحكم في هذه المسالك ، ومن أجل صنع مسلك وتشغيله يجب تعريف صف يقوم بوراثة الصف Thread وعلى صف المسلك أن يتجاوز التابع (run) الذي يخبر النظام عن كيفية تنفيذ المسلك عند تشغيله ، وأخيراً يجب إنشاء غرض يعمل على المسلك . ويبين المثال (12-1) العناصر الأساسية لصف المسلك الذي يرث الصف Thread ، ويبين كيف يمكن استخدام هذا الصف من أجل إنشاء مسلك في الصف الرئيسي . إن الغرضين yin و yang قابلاً للتنفيذ، تم إنشاؤهما من الصف YinYang وإن التابع start يخبر النظام أن المسلك جاهز للعمل .

مثال (12-1)

```
public class YinYang extends Thread {
    private String word;           // what to say
    public YinYang(String whatToSay) {
        word = whatToSay;
    }
}
```

```

public void run( ) {
    for (int i = 0; i < 10; i++) {
        System.out.print(word + " ");
        yield( ); // to give another thread a chance
    } //end for
} //end run
public static void main(String[ ] args) {
    // to create Yin thread
    YinYang yin = new YinYang("Yin");
    // to create Yang thread
    YinYang yang = new YinYang("Yang");
    yin.start( ); // to start Yin thread
    yang.start( ); // to start Yang thread
} //end main
} // end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

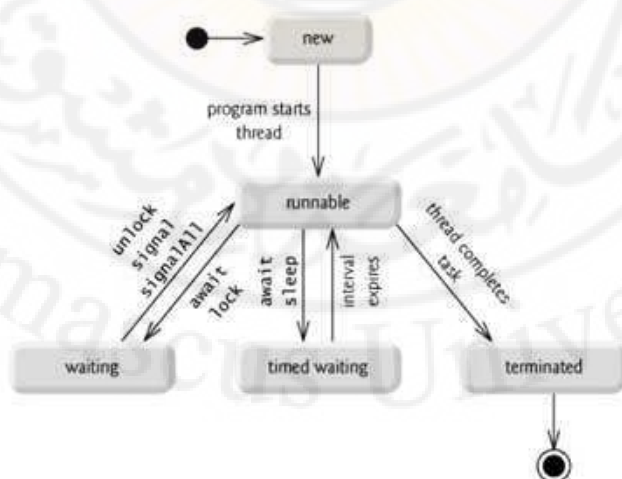
Result
Yin Yin Yin Yang Yang Yang Yin Yin Yin Yang Yang
Yang Yang Yin Yin Yin Yang Yang Yin
Process completed.

في هذا البرنامج البسيط :

١. الصف YinYang يرث الصف Thread .
٢. الصف Thread يتضمن التابع المجرد run() لذلك قمنا بتحقيقه .
٣. عرفنا في التابع main() مسلكين yin و yang ومررنا لكل منهما وسيطاً Yin و Yang .
٤. ثم استدعينا التابع start() والذي يستدعي التابع run ضمناً على كل مسلك من أجل بدء عمله .

١٢-٤- إنشاء المسالك من خلال تحقيق الواجهة Runnable

إن طريقة إنشاء المسالك من خلال وراثة الصف Thread تعمل بشكل جيد عندما يرث صف المسلك صفاً واحداً فقط هو الصف Thread ، ولا يمكن تطبيقها عندما يرث صف المسلك عدة صفوف وبالتالي من أجل تحقيق عملية الوراثة من عدة صفوف يجب تحقيق الواجهة Runnable ، وتقدم لغة Java الواجهة Runnable كبديل للصف Thread . أما الواجهة Runnable هي واجهة بسيطة تحتوي فقط على التابع (run()) ونحتاج إلى تنفيذ هذه الواجهة من أجل إخبار النظام كيف سيتم تنفيذ المسلك الخاص بنا . وبين المثال (11-2) العناصر الأساسية لصف المسلك الذي يحقق الواجهة Runnable ويوضح كيف يتم إنشاء مسلك في الصف الرئيسي . يجب أولاً من أجل تشغيل مسلك جديد مع الواجهة Runnable إنشاء غرض من الصف الذي يحقق الواجهة Runnable ثم استخدام بائي الصف Thread من أجل إنشاء مسلك . وبين الشكل (١١-٣) حالات المسلك مع الواجهة Runnable .



الشكل (١١-٣) حالات المسلك مع الواجهة Runnable .


```

public class YinYang implements Runnable {
    private String word;           // what to say
    public YinYang(String whatToSay) {
        word = whatToSay;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.print(word + " ");
            Thread.yield(); // to give another thread a chance
        }
    } //end class
    public static void main(String[ ] args) {
        // to instantiate YangYin
        YinYang rYang = new YinYang("Yang");
        // to instantiate again
        YinYang rYin = new YinYang("Yin");
        // to create Yang thread
        Thread yang = new Thread(rYang);
        Thread yin = new Thread(rYin); // to create Yin thread
        yang.start(); // to start Yang thread
        yin.start(); // to start Yin thread
    } //end main
} //end class

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Result

Yang Yang Yin Yang Yin Yang Yang Yin Yang Yin Yang
 Yin Yang Yin Yang Yin Yang Yin Yin Yin
 Process completed.

في هذا البرنامج :

١. نلاحظ أنه تم تحقيق الواجهة Runnable ومن ثم تحقيق التابع (run) .
٢. في التابع (main) تم إنشاء غرضين yin و yang من الصف Thread وتم تمرير غرض من الصف YinYang لكل منهما مع التنبيه بأن الصف YinYang يحقق الواجهة Runnable ، وأخيراً تم استدعاء التابع (start) لكلا الغرضين yin و yang .

١٢ - ٥ - التحكم بالمسالك والاتصالات بين المسالك

- يحتوي الصف Thread على التوابع التالية من أجل التحكم في المسالك :
١. التابع (public void run()) : يتم استدعائه من أجل تنفيذ المسالك ، ويجب تجاوز هذا التابع وكتابة التعليمات البرمجية التي نرغب أن يقوم المسلك بتنفيذها ضمن صف المسلك الذي أنشأناه ، ولا يمكن استخدام هذا التابع بشكل مباشر بواسطة الغرض القابل للتنفيذ في البرنامج.
 ٢. التابع (public void start()) : يُشغل المسلك الذي يؤدي إلى استدعاء التابع (run()) ، ويُستدعى التابع من خلال الغرض القابل للتنفيذ .
 ٣. التابع (public static void sleep(Long millis) throws InterruptedException) يضع هذا التابع الغرض القابل للتنفيذ في حالة ركود (سبات) من أجل زمن محدد مقدراً بالميلي ثانية .
 ٤. التابع (public static void yield()) يؤدي إلى توقف غرض المسلك الذي يتم تنفيذه حالياً بشكل مؤقت ويسمح بتنفيذ المسالك الأخرى .
 ٥. التابع (public void interrupt()) يقاطع عمل المسلك الذي يتم تنفيذه حالياً .

٦. التابع () `public static Boolean isInterrupted()` : يفحص فيما إذا تمت مقاطعة تنفيذ المسلك الحالي أم لا .
٧. التابع () `public boolean isAlive()` : يفحص فيما إذا كان المسلك يعمل حالياً أم لا .
٨. التابع () `public void setPriority(int p)` : يحدد الأولوية (تقع ضمن المجال من 1 إلى 10) من أجل هذا المسلك .
٩. التابع () `public final void notify()` : يوقظ أحد المسالك التي تنتظر في هذا الغرض ، ويتوقف اختيار المسلك على تنفيذ النظام .
١٠. التابع () `public final void notifyAll()` : يوقظ جميع المسالك التي تنتظر في هذا الغرض .
١١. التابع () `public void suspend()` : يعلق تنفيذ المسلك .
١٢. التابع () `public void resume()` : يستعيد تنفيذ المسلك .
١٣. التابع () `public final void wait( Long millis ) throws InterruptedException` : يجبر هذا التابع المسلك على الانتظار حتى يتم استدعاء التابع `notify` أو `notifyAll` من أجل الغرض الذي تم استدعاء التابع `wait` معه .

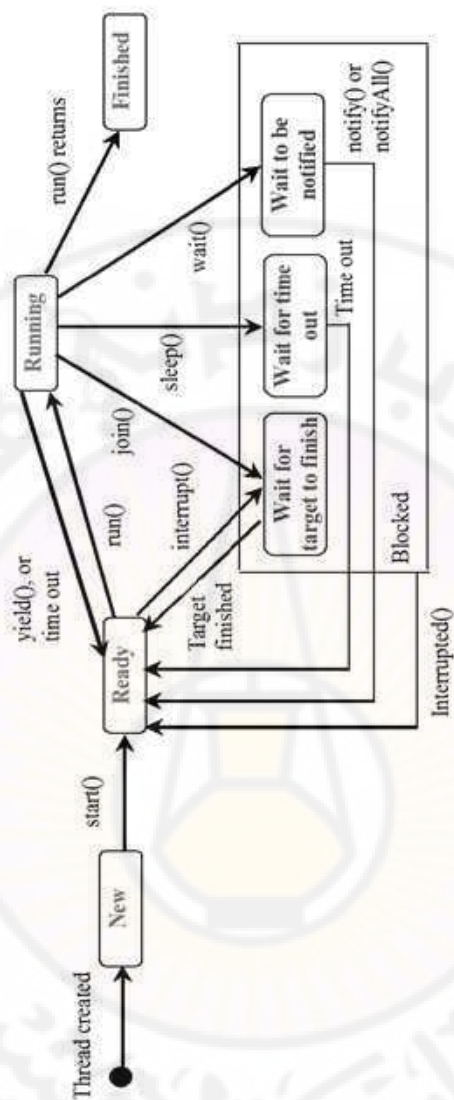
١٢-٦- حالات المسلك

إن كل مسلك له دورة حياة يمر فيها بعدة حالات أو مراحل خاصة ، ومن الممكن أن يوجد المسلك في واحدة من الحالات الخمس التالية :

`new , ready , running , blocked , finished` .

و يبين الشكل (١١-٤) حالات المسلك .

عندما يتم انشاء المسلك لأول مرة فإنه يدخل إلى الحالة `new` ، وبعد تشغيل المسلك من خلال استدعاء التابع `start` التابع له يدخل إلى الحالة `ready` ،



شكل (١٢-٤) - يبين حالات المسلك

حيث يكون المسلك في هذه الحالة قابلاً للتنفيذ لكن ربما لم يتم تنفيذه بعد، ويقوم نظام التشغيل في هذه المرحلة بتخصيص زمن المعالج له . عندما يتم تنفيذ المسلك الجاهز يدخل إلى الحالة running ، يمكن أن يدخل المسلك الموجود في الحالة running إلى الحالة ready إذا انتهت صلاحية زمن المعالج الخاصة به أو إذا تم استدعاء التابع yield() التابع له .

من الممكن أن يدخل المسلك إلى الحالة blocked (أي يصبح غير فعالاً) لعدة أسباب ، ربما يستدعي هذا المسلك التابع sleep() أو التابع wait() أو ربما تقوم توابع أخرى باستدعاء التابع sleep() التابع له ومن الممكن أن ينتظر المسلك انتهاء عملية دخل أو خرج . من الممكن إعادة تفعيل المسلك الموجود في حالة blocked عندما يتم عكس الحدث الذي تسبب في إيقاف تفعيله . وأخيراً يدخل المسلك إلى الحالة finished إذا انتهى من تنفيذ التابع run() التابع له أو إذا تم استدعاء التابع stop() التابع له .

يستخدم التابع isAlive() من أجل معرفة حالة المسلك ، حيث يعيد هذا التابع القيمة true إذا كان المسلك في وضع الجاهزية ready أو غير فعال أو في الحالة running ، ويعيد القيمة false إذا كان المسلك حديثاً ولم يتم تشغيله بعد أو إذا كان المسلك في الحالة finished .

١٢-٧- مجموعات المسالك

تحتوي بعض البرامج على عدد لا بأس به من المسالك التي تملك مهام وظيفية متشابهة ، لذلك من أجل السهولة يمكن تجميع هذه المسالك معاً وإنجاز العمليات على كامل المجموعة ، إذاً هي عبارة عن مجموعة من المسالك .

مثال :

يمكن تعليق تنفيذ أو استعادة تنفيذ جميع المسالك التي تنتمي إلى المجموعة نفسها في الوقت نفسه . سنعرض الخطوط الأساسية لاستخدام مجموعات المسالك .

- استخدام الباني ThreadGroup من أجل بناء مجموعة من المسالك :

ThreadGroup g = new ThreadGroup(" threadgroup ") ;

تُنشئ هذه العبارة مجموعة المسالك g التي تحمل الاسم " threadgroup " ، ويتألف الاسم من سلسلة نصية ويجب أن يكون فريداً .

- استخدام الباني Thread من أجل وضع المسلك في مجموعة المسالك :

Thread t = new Thread (g , new ThreadClass () , "this thread ") ;
تُنشئ هذه العبارة مسلكاً جديداً وتضعه في مجموعة المسالك g ، ونستطيع
إضافة مجموعة مسالك ضمن مجموعة مسالك أخرى من أجل تشكيل شجرة تتألف
من مجموعة مسالك تملك كل منها جذراً باستثناء مجموعة المسالك الابتدائية .

- لمعرفة عدد المسالك التي تعمل حالياً في المجموعة نستخدم
التابع activeCount () ، تعرض العبارة التالية عدد المسالك الفعالة في
المجموعة g :

System.out.println(" The Number of Runnable threads in the group ") ;
g.activeCount () ;

- لمعرفة المجموعة التي ينتمي إليها المسلك نستخدم التابع
getThreadGroup () .

١١-٨ - التزامن Thread Synchronization

نلاحظ بشكل عام نوعين من المسالك :

➤ الأول : يكون فيه المسلك متضمناً جميع المعطيات والتوابع اللازمة لتنفيذه ،
ولا يتطلب أي موارد أو توابع أخرى خارجية ، وفي هذا النوع لايهتم المسلك
بحالة أو نشاط أية مسلك آخر يجري تنفيذه بشكل متزامن معه .

➤ الثاني : يكون فيه عدة مسالك تجري بشكل متزامن ، وتشارك بمعطيات
محددة ، لذلك فإن كل منها يجب أن ينتبه إلى حالة ونشاط المسالك الأخرى
كما هو موضح في المثال (12-3) المنتج / المستهلك .

مثال (12-3)

يقوم الصف Producer بتوليد رقم صحيح بين 0 و 9 ضمناً ، ويخزنه
في غرض من الصف CubbyHole ويطبّع الرقم المولد . لجعل مشكلة التزامن

أكثر إمتاعاً ، سنجعل Producer ينام (sleeping) لفترة من الزمن يتم تحديدها بشكل عشوائي بين القيمتين 0 و 100 من الملي ثانية ، قبل تكرار عملية توليد رقم جديد . الصف Consumer يقوم بجلب (استهلاك) جميع الأرقام الصحيحة من الغرض المنشأ من الصف CubbyHole (وهو الغرض نفسه الذي وضعت فيه الأرقام من خلال الصف Producer) وذلك كلما كان هناك توفر في عدد من الأرقام

• النص البرنامجي :

```
public class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number + " put: " + i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        } //end for
    } // end run
} //end class Producer

public class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;
    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
```

```

        this.number = number;
    }
    public void run ( ) {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get( );
            System.out.println("Consumer #" + this.number + " got: " + value);
        } //end for
    } // end run
} //end class Consumer

public class CubbyHole {
    private int contents;
    private boolean available = false;
    public synchronized int get( ) {
        while (available == false) {
            try {
                wait( ); // wait for Producer to put value
            } catch (InterruptedException e) { }
        } //end while
        available = false;
        notifyAll( ); // notify Producer that value has been retrieved
        return contents;
    } //end get

    public synchronized void put(int value) {
        while (available == true) {
            try {
                wait( ); // wait for Consumer to get value
            } catch (InterruptedException e) { }
        } // end while
        contents = value;
        available = true;
    }
}

```

```

        notifyAll( ); // notify Consumer that value has been set
    } //end put
} //end class CubbyHole

public class ProducerConsumerTest {
    public static void main(String[ ] args) {
        CubbyHole c = new CubbyHole( );
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);
        p1.start( );
        c1.start( );
    } //end main
} //end class Test

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:

Producer #1 put: 0	Producer #1 put: 5
Consumer #1 got: 0	Consumer #1 got: 5
Producer #1 put: 1	Producer #1 put: 6
Consumer #1 got: 1	Consumer #1 got: 6
Producer #1 put: 2	Producer #1 put: 7
Consumer #1 got: 2	Consumer #1 got: 7
Producer #1 put: 3	Producer #1 put: 8
Consumer #1 got: 3	Consumer #1 got: 8
Producer #1 put: 4	Producer #1 put: 9
Consumer #1 got: 4	Consumer #1 got: 9

إن كل من الصنفين Producer و Consumer يتشاركان في المعطيات من خلال غرض عام مشترك من الصف CubbyHole . نلاحظ أن كل من الصنفين Producer و Consumer لا يقوم بأي جهد أو محاولة ، ليتحقق فيما إذا كان Consumer يأخذ كل قيمة مولدة مرة واحدة فقط ، وتحدث المزامنة بين

هذين المسلكين في مستوى منخفض ، وذلك ضمن التابعين get و put للغرض cubbyhole ، ومن المشاكل المحتملة يمكن أن يكون Producer أسرع من Consumer ويولد رقمين قبل أن يستهلك Consumer الرقم الأول . في هذه الحالة سيقوم Consumer بتخطي ذلك الرقم ويبدو الخرج كما يلي :

```
Producer #1 put: 5
Producer #1 put: 6
Consumer #1 got: 6
Producer #1 put: 7
```

المشكلة الأخرى ، يمكن أن تظهر عندما يكون Consumer أسرع من Producer حيث يقوم باستهلاك القيمة التي ينتجها Producer مرتين ، وفي هذه الحالة سيقوم Consumer بطباعة القيمة مرتين وعندها نحصل على الخرج التالي :

```
Producer #1 put: 5
Consumer #1 got: 5
Consumer #1 got: 5
Producer #1 put: 6
```

كل من النتيجتين السابقتين هي نتيجة خاطئة ، ولحل هذه المشكلة ، يجب أن تتم مزامنة نشاطات كل من Consumer و Producer بطريقتين :

١. يجب ألا يصل كل من المسلكين إلى CubbyHole معاً ، ويمكن منع الحدوث من خلال قفل الغرض Locking . عندما يتم قفل غرض من مسلك ما سيتم تجميده block حتى يقوم ذلك المسلك بإلغاء القفل unlock عن ذلك الغرض .

٢. يجب على المسلكين أن يقوموا بتنسيق بسيط بينهما ، يجب على الصف Producer ان يكون لديه أسلوب ما يشير إلى الصف Consumer أن

قيمة ما أصبحت موجودة ، وكذلك يجب على الصف Consumer أن يكون لديه أسلوب ما ليشير إلى أنه قد استلم تلك القيمة .
على الصف Thread أن يوفر مجموعة من التوابع مثل : wait و notify و notifyAll لمساعدة المسالك من أجل أن تنتظر شرط ما ، ومن ثم تنبه المسالك الأخرى عندما يتغير الشرط .

يشكل التابعان put و get من الصف CubbyHole القسم الحرج ، لا يجب أن يصل Consumer إلى CubbyHole عندما يقوم Producer بتغييره ، وكذلك لا يجب على Producer أن يعدله عندما يريد Consumer الحصول القيمة . لذلك يجب أن نحدد كل من التابعين put و get في الصف CubbyHole بالكلمة المحجوزة synchronized .

في أي لحظة يدخل التحكم إلى تابع محدد بالكلمة المحجوزة **synchronized** ، فإن المسلك الذي يستدعي ذلك التابع يقفل الغرض ، وبهذا لن تتمكن المسالك الأخرى من استدعاء هذا التابع على الغرض نفسه ، وذلك حتى يتم إلغاء القفل على ذلك الغرض .

إذاً عندما يستدعي Producer التابع put للصف CubbyHole فإنه يقفله ، وبذلك يمنع الصف Consumer من استدعاء التابع get للصف CubbyHole ، وعندما ينهي التابع put عمله فإن الصف Producer يلغي قفل CubbyHole وبالمثل عندما يستدعي الصف Consumer التابع get فإنه يقفله وذلك يمنع Producer من استدعاء التابع put ، وعندما ينهي التابع get عمله فإن الصف Consumer يلغي قفل CubbyHole .

١٢-٩- التحكم بالتحريك باستخدام المسالك

يستخدم المثال (12-4) التالي عدة توابع للرسم وعدة توابع مثلثية من أجل رسم ساعة تعرض الوقت الحالي ضمن إطار . نحتاج من أجل رسم الساعة إلى رسم دائرة وثلاثة عقارب (ساعات ، دقائق ، ثواني) كما هو مبين في الشكل (١٢-٥)، ونحتاج من أجل رسم العقرب إلى تحديد نهايتي الخط بحيث تقع إحدى النهايتين في مركز الدائرة (xCenter , yCenter) وتقع النهاية الأخرى في النقطة الإحداثية (xEnd , yEnd) والتي تحسب من العلاقة التالية :

$$xEnd = xCenter + handLength \times \sin(\alpha)$$

$$yEnd = yCenter - handLength \times \cos(\alpha)$$

تُقاس الزاوية α بالراديان .

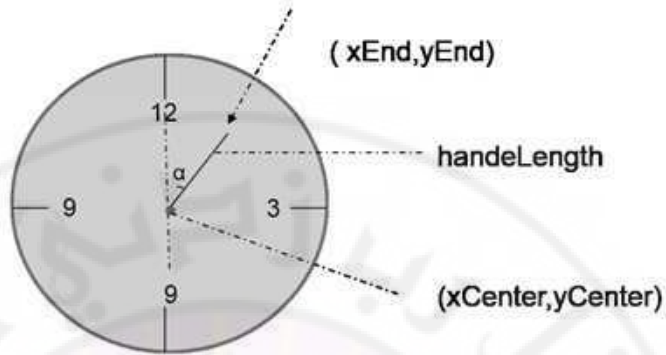
- بما أن الدقيقة تتألف من 60 ثانية وبالتالي فإن الزاوية التابعة لعقرب الثواني هي : $(2\pi/60) \times \text{second}$

- يتم تحديد مكان عقرب الدقائق بواسطة الدقائق والثواني ، حيث تتألف قيمة الدقائق الفعلية من جمع الدقائق مع الثواني أي : $\text{minute} + \text{second}/60$

بما أن الساعة تتألف من 60 دقيقة وبالتالي فإن الزاوية التابعة لعقرب الدقائق هي : $(2\pi/60) \times (\text{minute} + \text{second}/60)$

- يمكن تقسيم الدورة الواحدة إلى 12 ساعة لذلك فإن الزاوية التابعة لعقرب الساعات هي :

$$(Hour + \text{minute}/60 + \text{second}/(60 \times 60)) \times (2\pi/12)$$



الشكل (١٢-٥) - يبين تحديد إحداثيات العقرب

للسهولة يمكن حذف الثواني من عملية حساب الزوايا من أجل عقرب الدقائق والساعات وذلك لأن قيمتها صغيرة جداً ومن الممكن تجاهلها لذلك يتم حساب نقط النهاية من أجل عقرب الثواني والدقائق والساعات كما يلي :

$$xSecond = xCenter + secondHandLength \times \sin(second \times (2\pi/60))$$

$$ySecond = yCenter - secondHandLength \times \cos(second \times (2\pi/60))$$

$$xMinute = xCenter + MinuteHandLength \times \sin(minute \times (2\pi/60))$$

$$yMinute = yCenter - MinuteHandLength \times \cos(minute \times (2\pi/60))$$

$$xHour = xCenter + HourHandLength \times \sin((hour + minute / 60) \times (2\pi/60))$$

$$yHour = yCenter - HourHandLength \times \cos((hour + minute / 60) \times (2\pi/60))$$

مثال (4-12)

```
// Display a clock
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.util.*;
```

```
import javax.*;  
import javax.swing.Timer;
```

```
public class StillClock extends JPanel {  
    private int hour;  
    private int minute;  
    private int second;  
  
    /** Construct a default clock with the current time */  
    public StillClock( ) {  
        setCurrentTime( );    }  
    /** Construct a clock */  
    public StillClock(int hour, int minute, int second) {  
        this.hour = hour;  
        this.minute = minute;  
        this.second = second;  
    }  
    /** Return hour */  
    public int getHour() {  
        return hour;  
    }  
    /** Set a new hour */  
    public void setHour(int hour) {  
        this.hour = hour;  
        repaint( );  
    }  
    /** Return minute */  
    public int getMinute( ) {  
        return minute;  
    }  
    /** Set a new minute */  
    public void setMinute(int minute) {  
        this.minute = minute;  
        repaint( );  
    }  
}
```

```

}
/** Return second */
public int getSecond( ) {
    return second;
}
/** Set a new second */
public void setSecond(int second) {
    this.second = second;
    repaint( );    }
/** Draw the clock */
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    // Initialize clock parameters
    int clockRadius =
        (int)(Math.min(getWidth( ), getHeight( )) * 0.8 * 0.5);
    int xCenter = getWidth( ) / 2;
    int yCenter = getHeight( ) / 2;

    // Draw circle
    g.setColor(Color.black);
    g.drawOval(xCenter - clockRadius, yCenter - clockRadius,
               2 * clockRadius, 2 * clockRadius);
    g.drawString("12", xCenter - 5, yCenter - clockRadius + 12);
    g.drawString("9", xCenter - clockRadius + 3, yCenter + 5);
    g.drawString("3", xCenter + clockRadius - 10, yCenter + 3);
    g.drawString("6", xCenter - 3, yCenter + clockRadius - 3);

    // Draw second hand
    int sLength = (int)(clockRadius * 0.8);
    int xSecond = (int)(xCenter + sLength *
        Math.sin(second * (2 * Math.PI / 60)));
    int ySecond = (int)(yCenter - sLength *
        Math.cos(second * (2 * Math.PI / 60)));

```

```
g.setColor(Color.red);
g.drawLine(xCenter, yCenter, xSecond, ySecond);
```

```
// Draw minute hand
```

```
int mLength = (int)(clockRadius * 0.65);
int xMinute = (int)(xCenter + mLength *
    Math.sin(minute * (2 * Math.PI / 60)));
int yMinute = (int)(yCenter - mLength *
    Math.cos(minute * (2 * Math.PI / 60)));
g.setColor(Color.blue);
g.drawLine(xCenter, yCenter, xMinute, yMinute);
```

```
// Draw hour hand
```

```
int hLength = (int)(clockRadius * 0.5);
int xHour = (int)(xCenter + hLength *
    Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
int yHour = (int)(yCenter - hLength *
    Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
g.setColor(Color.green);
g.drawLine(xCenter, yCenter, xHour, yHour);
}
```

```
public void setCurrentTime( ) {
```

```
// Construct a calendar for the current date and time
```

```
Calendar calendar = new GregorianCalendar( );
```

```
// Set current hour, minute and second
```

```
this.hour = calendar.get(Calendar.HOUR_OF_DAY);
```

```
this.minute = calendar.get(Calendar.MINUTE);
```

```
this.second = calendar.get(Calendar.SECOND);
```

```
}
```

```
public Dimension getPreferredSize( ) {
```

```
return new Dimension(200, 200);
```



```

    }
} // end class StillClock
//-----

public class ClockWithTimer extends StillClock
    implements ActionListener {
    // Create a timer with delay 1000 ms
    protected Timer timer = new Timer(1000, this);
    public void suspend( ) {
        timer.stop( ); // Suspend clock
    }

    public void resume( ) {
        timer.start( ); // Resume clock
    }
    /** Handle the action event */
    public void actionPerformed(ActionEvent e) {
        // Set new time and repaint the clock to display current time
        setCurrentTime( );
        repaint( );
    }
} // end class ClockWithTimer
//-----

public class ClockControl extends JPanel implements
    ActionListener {
    private ClockWithTimer clock = new ClockWithTimer( );
    private JButton jbtSuspend = new JButton("Suspend");
    private JButton jbtResume = new JButton("Resume");
    public ClockControl( ) {
        // Group buttons in a panel
        JPanel panel = new JPanel( );
        panel.add(jbtSuspend);
        panel.add(jbtResume);
        // Add clock and buttons to the panel
        setLayout(new BorderLayout());
    }
}

```

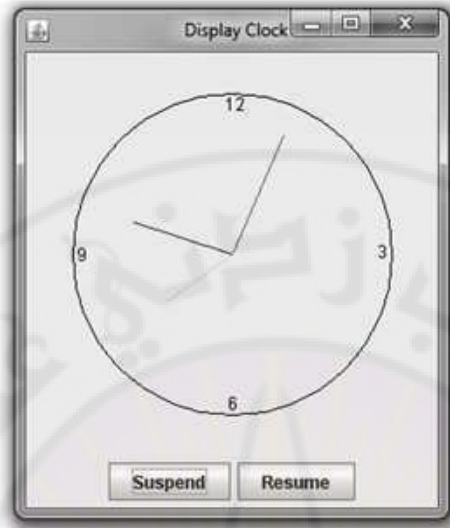
```

add(clock, BorderLayout.CENTER);
add(panel, BorderLayout.SOUTH);

// Register listeners
jbtSuspend.addActionListener(this);
jbtResume.addActionListener(this);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jbtSuspend)
        clock.suspend();
    else if (e.getSource() == jbtResume)
        clock.resume();
}
} //end class ClockControl
public class test {
public static void main(String[] args) {
    JFrame frame = new JFrame();
    ClockControl ob = new ClockControl();
    frame.add(ob);
    frame.setTitle("Display Clock");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 350);
    frame.setVisible(true);
} //end main
} //end class test

```

وعند تنفيذ البرنامج سنحصل على الخرج التالي:



□ يقوم الصف StillClock برسم ساعة لأظهار الوقت إلا أن هذه الساعة لا تتحرك بها عقارب الساعة .

الحل هو إعادة رسم الساعة في كل ثانية مع إضافة الوقت الحالي الجديد ، نستطيع استخدام المسلك للتحكم بكيفية إعادة رسم الساعة .

□ يرث الصف Clock الصف StillClock من أجل عرض الساعة ويحقق الواجهة Runnable من أجل تمكين الساعة من العمل .

بما أنه لا يفضل استعمال التابعين suspend() و resume() التابعين للصف Thread ، لذا قمنا بصنع توابع جديدة من أجل تعليق تنفيذ المسلك واستعادته . ويدل المتحول suspended الذي تم التصريح عنه على حالة المسلك . وتضمن الكلمة المحجوزة synchronized تزامن التابعين suspend و resume لتجنب الوقوع في شروط التسابق التي قد تنتج بسبب القيمة غير الثابتة للمتحول suspended . تم تنفيذ التابع run() من أجل تحديد قيمة الزمن الجديد وإعادة رسم الساعة في كل ثانية وبشكل مستمر ، ويتم داخل جسم الحلقة while إعاقه تنفيذ المسلك إذا كانت قيمة المتحول suspended هي true ، وبسبب

الاستدعاء للتابع (`waitForNotificationToResume()`) يُعلق عمل المسلك ويجعله ينتظر الإشعار (التنبية) من التابع (`notifyAll()`) الذي يتم استدعاؤه من خلال التابع (`resume()`) .

□ يرث الصف `ClockControl` الصف `Jpanel` من أجل عرض الساعة وزري التحكم ويحقق الواجهة `ActionListener` من أجل معالجة الحدث `ActionEvents` الناتج عن الزرين ، وعند النقر على الزر `Suspend` يتم استدعاء التابع `suspend` التابع للصف `Clock` من أجل تعليق عمل الساعة وعند النقر على الزر `Resume` يتم استدعاء التابع `resume` التابع للغرض `clock` من أجل استعادة عمل الساعة .

□ يتم استدعاء التوابع (`wait()`) و (`notify()`) و (`notifyAll()`) من أجل الاتصال بين المسالك .

١٢-١٠ - التحكم بالتحريك باستخدام الصف `Timer`

في المثال (11-4) تم إنشاء مسلك من أجل تنفيذ الحلقة `while` التي تقوم بإعادة رسم الساعة في اللوحة كل ثانية ، بما أن التحريك في `Java` يتطلب إعادة رسم اللوحات بشكل متكرر وفق تردد معروف مسبقاً لذلك قدمت `Java` الصف `Timer` الذي من الممكن استخدامه من أجل التحكم بإعادة رسم اللوحة وفق تردد معروف مسبقاً ، ويُعد الصف `Timer` مصدراً للحدث `ActionEvents` حيث يقوم بقدر هذا الحدث وفق تردد ثابت ، ويتم تسجيل المتتصات من أجل الحدث بواسطة الغرض `timer` .

يجب عند إنشاء غرض من الصف `Timer` تحديد مدة التأخير والمتتصة باستخدام الباني التالي :

```
public Timer ( int delay , ActionListener listener )
```


يحدد الوسيط delay المدة بين كل حدثي عمل بالميلي ثانية . ويمكننا إضافة متتصات أخرى باستخدام التابع addActionListener وتعديل مدة التأخير باستخدام التابع setDelay .

يستخدم التابع start من أجل تشغيل المؤقت والتابع stop من أجل إيقافه . ويمكن تنفيذ المعالج actionPerformed بواسطة استدعاء التابع repaint من أجل إعادة رسم اللوحة ، حيث يتم استدعاء التابع actionPerformed وفق التردد المحدد بواسطة الوسيط delay .

ويمكن استبدال الصف Clock في المثال (11-4) باستخدام الصف ClockWithTimer ، ويبين المثال (11-5) الصف ClockWithTimer

مثال (5-12)

```
import javax.*;
import java.awt.event.*;
import javax.swing.Timer;

public class ClockWithTimer extends StillClock
    implements ActionListener {
    // Create a timer with delay 1000 ms
    protected Timer timer = new Timer(1000, this);
    public void suspend( ) {
        timer.stop( ); // Suspend clock
    }
    public void resume( ) {
        timer.start( ); // Resume clock
    }
    /** Handle the action event */
    public void actionPerformed(ActionEvent e) {
        // Set new time and repaint the clock to display current time
        setCurrentTime( );
        repaint( );
    }
}
```



```
}
} //end class ClockWithTimer
```

مثال (12-6)

هذا المثال يعرض كرة ترتد من حواف اللوحة ، ونستخدم زرّين من أجل تعليق حركة الكرة واستعادتها واستخدام شريط الإزاحة من أجل التحكم بسرعة الارتداد والحركة . يرث الصف Ball الصف JPanel من أجل عرض الكرة المرتدة ، ثم ننشئ الغرض timer من أجل عرض الكرة بشكل مستمر وفق تردد ثابت ، حيث يتغير مركز الكرة الموجودة عند الموضع (x,y) إلى الموضع (x+dx,y+dy) عند العرض التالي . من الممكن استخدام التابعين suspend و resume من أجل إيقاف وتشغيل المؤقت . يرث الصف BallControl الصف JPanel من أجل عرض الكرة وشريط الإزاحة وزرّي التحكم ويحقق الواجهة ActionListener من أجل معالجة الأحداث الناتجة عن الأزرار . ويحقق الصف BallControl الواجهة AdjustmentListener من أجل معالجة الأحداث الناتجة عن تغيير القيمة من شريط الإزاحة . عند النقر على الزر Suspend يتم استدعاء التابع suspend الذي يقوم بإيقاف حركة الكرة . وعند النقر على الزر Resume يقوم باستدعاء التابع resume الذي يقوم باستعادة حركة الكرة ، ومن الممكن تغيير سرعة ارتداد الكرة باستخدام شريط الإزاحة .

```
import java.awt.event.*;
import javax.swing.Timer;
import java.awt.*;
import javax.swing.*;
```

```
public class Ball extends JPanel implements ActionListener {
    private int delay = 10;
```

```

// Create a timer with delay 1000 ms
protected Timer timer = new Timer(delay, this);

private int x = 0; private int y = 0; // Current ball position
private int radius = 15; // Ball radius
private int dx = 2; // Increment on ball's x-coordinate
private int dy = 2; // Increment on ball's y-coordinate

public Ball() {
    timer.start();
}

/** Handle the action event */
public void actionPerformed(ActionEvent e) {
    repaint();
}

public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.red);

    // Check boundaries
    if (x < radius) dx = Math.abs(dx);
    if (x > getWidth() - radius) dx = -Math.abs(dx);
    if (y < radius) dy = Math.abs(dy);
    if (y > getHeight() - radius) dy = -Math.abs(dy);

    // Adjust ball position
    x += dx;
    y += dy;
    g.fillOval(x - radius, y - radius, radius * 2, radius * 2);
}

public void suspend() {

```

```

    timer.stop( ); // Suspend clock
}

public void resume( ) {
    timer.start( ); // Resume clock
}

public void setDelay(int delay) {
    this.delay = delay;
    timer.setDelay(delay);
}
}
//-----
import javax.swing.*;
import java.awt.event.*;
import java.awt.BorderLayout;

public class BallControl extends JPanel
implements ActionListener, AdjustmentListener {
    private Ball ball = new Ball( );
    private JButton jbtSuspend = new JButton("Suspend");
    private JButton jbtResume = new JButton("Resume");
    private JScrollBar jsbDelay = new JScrollBar( );
    public BallControl( ) {
        // Group buttons in a panel
        JPanel panel = new JPanel( );
        panel.add(jbtSuspend);
        panel.add(jbtResume);
        // Add ball and buttons to the panel
        ball.setBorder(BorderFactory.createEtchedBorder());
        jsbDelay.setOrientation(JScrollBar.HORIZONTAL);
        ball.setDelay(jsbDelay.getMaximum());
        setLayout(new BorderLayout());
        add(jsbDelay, BorderLayout.NORTH);
    }
}

```

```

add(ball, BorderLayout.CENTER);
add(panel, BorderLayout.SOUTH);

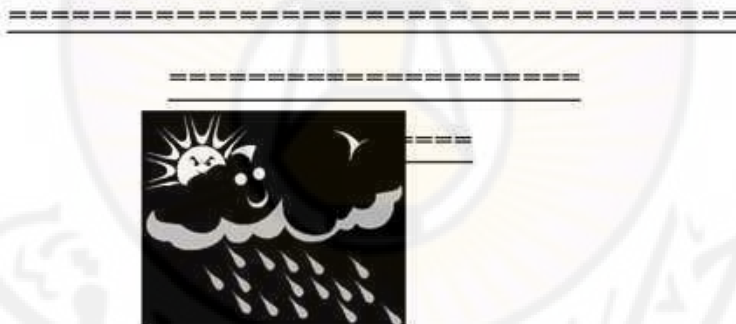
// Register listeners
jbtSuspend.addActionListener(this);
jbtResume.addActionListener(this);
jsbDelay.addAdjustmentListener(this);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == jbtSuspend)
        ball.suspend();
    else if (e.getSource() == jbtResume)
        ball.resume();
}
public void adjustmentValueChanged(AdjustmentEvent e) {
    ball.setDelay(jsbDelay.getMaximum() - e.getValue());
}
}
//-----
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
public class testBall {
public static void main(String[] args) {
    BallControl applet = new BallControl();
    JFrame frame = new JFrame();
    frame.add(applet);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setTitle("Bouncing Ball Control");

    frame.setSize(400, 320);
    frame.setVisible(true);
} //end main

```

```
}//end class
```

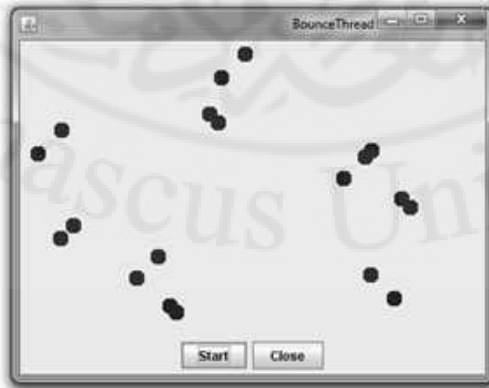
وعند تنفيذ البرنامج سنحصل على الخرج التالي:



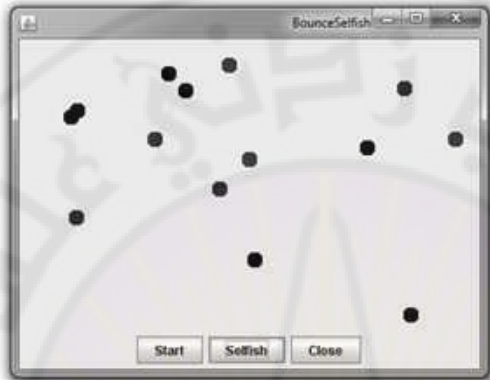
مسائل عامة

١- اكتب برنامجاً بلغة Java يعرض إطلاق كرة من زر start وترتد من حواف الكرة لعدد محدد من المرات ثم يتم إنهاؤها ، وكل مرة يتم الضغط على الزر start يتم إطلاق كرة بحيث نحصل على عدد من الكرات ترتد من حواف اللوحة وتخفي ويوجد زر Close من أجل إنهاء البرنامج ، ويحتوي البرنامج على الصفوف التالية :

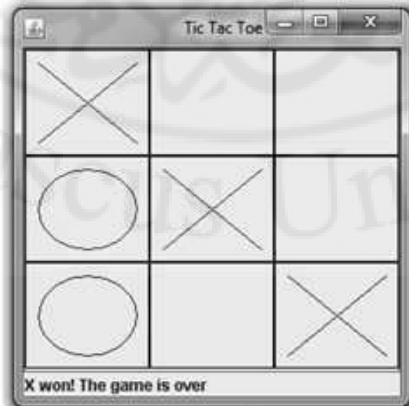
- صف اسمه BounceFrame يرث الصف JFrame من أجل إنشاء إطار والتحكم بالكرة المرتدة من إطلاق عدد من الكرات والخروج من البرنامج .
- صف اسمه BallThread يرث الصف Thread من أجل تنشيط المسالك للحصول على أكثر من كرة مرتدة على اللوحة ، بحيث لا يحصل تصادم بين الكرات المرتدة من اللوحة .
- صف اسمه BallCanvas يرث الصف JPanel من أجل رسم الكرات على اللوحة .
- يجب أن نحصل على الخرج التالي :



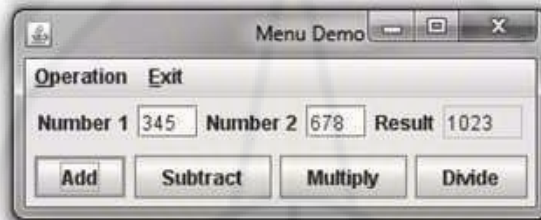
٢- أعد السؤال السابق بحيث يكون لدينا مصدرين لإطلاق الكرات ، الزر الأول start أما الزر الثاني Selfish من أجل إصدار كرات بلون آخر ، ويوجد زر ثالث هو زر الإغلاق Close ، بحيث نحصل على الخرج التالي :



٣- اكتب برنامجاً بلغة Java يمثل اللعبة TicTacToe ، تتألف هذه اللعبة من لاعبين يأخذ كل منهما دوراً من أجل تحديد خلية متاحة (فارغة) ضمن شبكة مؤلفة من ثلاثة أعمدة وثلاثة صفوف (3×3) بالرمز الخاص به (إما X أو O) عندما يستطيع أحد اللاعبين الحصول على ثلاثة رموز متوضعة وفق صف أفقي أو وفق صف عمودي أو وفق صف قطري تنتهي اللعبة ويفوز اللاعب . تحدث حالة التعادل (draw) في هذه اللعبة عندما يتم تحديد جميع خلايا الشبكة بالرموز وبدون إحراز الفوز من قبل احد اللاعبين . بحيث نحصل على الخرج التالي للعبة :



٤- اكتب برنامجاً بلغة Java يصنع واجهة مستخدم من أجل إنجاز العمليات الرياضية ، وتحتوي واجهة المستخدم على اللافتات وعلى حقول النص من أجل العدد الأول (Number1) والعدد الثاني (Number2) والنتيجة (Result) ، ويعرض حقل النص Result ناتج العملية الرياضية بين العدد الأول (Number1) والعدد الثاني (Number2) ، بحيث نحصل على الخرج التالي :



Operation	Exit
Number 1	345
Number 2	678
Result	1023

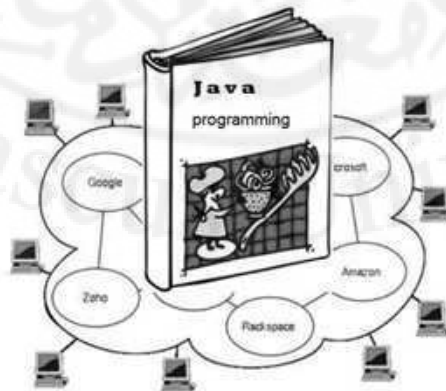
Add Subtract Multiply Divide



المراجع الأجنبية

References

1. Building Java Programs: A Back to Basics Approach (4th Edition) , by Stuart Reges and Marty Stepp , Feb 15, 2016 .
2. Beginning Java 2, Ivor Horton ,Wrox Press Ltd. ® © 2012
3. Java™ How to Program, By H. M. Deitel - Deitel & Associates, Inc., P. J. Deitel - Deitel & Associates, Inc.2016
4. Java™ 2: The Complete Reference, By Herbert Schildt, McGraw-Hill ,2016 .
5. Java: A Beginner's Guide, Sixth Edition , by Herbert Schildt , May 6, 2014
6. Java: A Beginner's Guide, Sixth Edition , by Herbert Schildt ,May 6, 2014
7. Java™ How to Program, By H. M. Deitel - Deitel & Associates, Inc., P. J. Deitel - Deitel & Associates, Inc.2016
8. Java Concurrency in Practice 1st Edition ,by Brian Goetz and Tim Peierls, Dec 16, 2015



المراجع العربية

١. البرمجة غرضية التوجه باستخدام Java - بلال مروان الإسماعيل & محمد ربيع حاتم شاهين - دار مهارات للعلوم - سورية - حمص - 2008 م .
٢. البرمجة (1) : البرمجة غرضية التوجه باستخدام C++ - الدكتور المهندس مأمون يونس & الدكتور أكرم مذكور - جامعة دمشق - سورية - 2015 م .
٣. بناء التطبيقات باستخدام Java - بلال مروان الإسماعيل ، محمد ربيع حاتم شاهين - دار مهارات للعلوم - سورية - حمص - 2008 م .
٤. تعلم Java باستخدام بيئة التطوير JBuilder - المهندس جلال خرسة - شعاع للنشر والعلوم - سورية - حلب - 2008 .
٥. كيف تبرمج بلغة C++ - الدكتور صلاح الدوه جي - شعاع للنشر والعلوم - سورية - حلب - 2004 .
٦. لغة البرمجة Java - المهندس فادي حجار - شعاع للنشر والعلوم - سورية - حلب - 2012 .
٧. مدخل إلى الحاسوب والبرمجة - الدكتور المهندس مأمون يونس & الدكتور المهندس جمال الياسين & الدكتور أكرم مذكور - جامعة دمشق - سورية - 2015 م .



اللجنة العلمية :

الدكتور جمال الياسين
الدكتور محمد مازن المحايري
الدكتور سمير كرمان

المدقق اللغوي :

الدكتور دياب راشد

حقوق الطبع والنشر محفوظة لمديرية الكتب والمطبوعات الجامعية

