بسم الله الرحمن الرحيم

جامعة السّودان المفتوحة

برنامج الحاسوب

البرمجة بلغة جافا

رمز المقرر ورقمه: حسب 4093

إعداد المادة العلمية: أ.نعمة عزالدين محمد عثمان

التحكيم العلمي : د. نهى مدثر بحيري
تصمـــيم تعلـــيمي : د.عبد الباسط محمد شريف
التدقيق اللغوي : أ. الهدي عبد الله محمد
التصميم الفني :أ. أماني الأمين مبارك

منشورات جامعة السودان المفتوحة، الطبعة الأولى 2008م
جميع الحقوق محفوظة لجامعة السودان المفتوحة، لا يجوز إعادة إنتاج أيّ جزء من هذا الكتاب، وبأيّ وجه من الوجوه، إلاّ بعد الموافقة المكتوبة من الجامعة.

مقدمة المقرر

الحمد لله الذي لا يخيب راجيه، ولا يردّ داعيه، وسلامٌ على عباده الذين اصطفى، وعلى خير نبيٍ في العالمين ارتقى، سيدنا وقدوتنا محمد صلى الله عليه وسلّم تسليماً كثيراً إلى يوم الدين. الحمد لله الذي هدانا لهذا وما كنا لنهتدي لولا أن هدانا الله. الحمد لله الذي علّم بالقلم، علّم الإنسان ما لم يعلم. فهذا الكتاب " البرمجة بلغة جافا " يحتوي على كل ما يحتاجه الطالب المبتدئ والمتقدم لتعلم مفاهيم ومهارات لغة جافا، حيث يقدم شرحاً وافياً مصحوباً بالأمثلة للدروس الأساسية والمتقدمة في جافا. ليجعل فهم لغة جافا والبرمجة باستخدامها أكثر سهولة ومتعة.

يشتمل هذا الكتاب على الوحدات التالية:

الوحدة الأولى وعنوانها "**مبادئ لغة جافا**" وفيها يتحدث الكتاب بإسهاب عن أساسيات لغة جافا كدعامة أساسية تستند عليها جميع المفاهيم الأخرى باللغة. ثم يعرج في الوحدة الثانية إلى "**البرمجة بالكائنات**" لكونها أقوى مميزات لغة جافا، شارحاً المفاهيم المتعلقة بالبرمجة بالكائنات كالوراثة وتعدد الأشكال. يقدم الكتاب في الوحدة الثالثة والتي عنوانها "**Graphics and Graphical User InterFace(GUI)** " وهي تبحث عن الرسم وتصميم الشاشات باستخدام جافا لأهمية تركيز المبرمج على تهيئة مظهر البرنامج الخارجي للمستخدم، ليضفي عليه جاذبية وسهولة استخدام. الوحدة الرابعة وعنوانها " **موضوعات متقدمة في لغة جافا** " وتحتوي هذه الوحدة على خمسة أقسام ، حيث يتناول القسم الأول البريمجات التي مكنت من كتابة برامج صفحات الإنترنت بلغة جافا، ثم قسم يتحدث عن الأخطاء المختلفة التي تحدث أثناء تنفيذ البرنامج وكيفية معالجتها، يليه قسم يشرح الطرق المختلفة للتعامل مع الملفات في جافا. تعتبر البرمجة المتعددة من الموضوعات المهمة في البرمجة عموماً، ولذلك أفردنا لها قسماً. وفي القسم الأخير، نبين أهمية الشبكات في برامج الحاسوب، وما توفره لغة جافا لبرمجة تطبيقات الشبكات.

عزيزي الدارس، لقد كُتِبَتْ هذه الوحدات بأسلوب سهل وواضح يتفق مع أسلوب التعليم الذاتي. لذا يتوقع منك أن تقرأ وتستعد لكل موضوع قبل مناقشته في اللقاء مع مشرفك الأكاديمي، لتكون مشاركتك وتفاعلك في اللقاء عاملاً مساعداً على نجاح الدرس وإثرائه. كذلك اشتملت هذه الوحدات على عدد من الأسئلة التي وضعت لتساعدك في إثراء معلوماتك وتعزيز ما تتعلمه وتقييم درجة فهمك واستيعابك للمفاهيم والمعلومات والأفكار التي وردت في هذه الوحدات. نأمل أن تقدم هذه الوحدات صورة واضحة عن البرمجة بلغة حافا.

عزيزي الدارس، لدى قراءتك هذا الكتاب أرجو ألاَّ يغيب عن ذهنك أن مادة هذا الكتاب هي الحد الأدنى من المعرفة في موضوعه، الذي يسمح بإثرائه على الدوام بالرجوع إلى القراءات المساعدة والاطلاع عليها.

وفيما يلي أهم الأهداف العامة لهذا المقرر.

الأهداف العامة للمقرر



عزيزي الدارس، بعد فراغك من دراسة هذا المقرر يؤمل أن تكون لـــك القدرة على:

- **تعدادِ** مميزات لغة جافا.
- **تعدادِ** مكونات لغة جافا.
- **تعدد** أنواع البيانات الأساسية في لغة جافا.
- **تَذكُّرِ** المبادئ التي تقوم عليها البرمجة بالكائنات.
- **تَذكُّرِ** الأدوات التي توفرها لغة جافا في الرسم والتلوين على شاشة البرنامج، وكذلك كيفية استخدام هذه الأدوات.
- **الشرح** بالمثال برنامجاً لكل من المخدم والزبون باستخدام برتوكوليTCP و UDP.
- **تعدادِ** حالات البرمجة المتعددة.

وختاماً نرجو أن نكون قد وفقنا في تقديم عمل مفيد لمكتبة جامعة السودان المفتوحة من خلال هذا الكتاب. وأسأل الله تعالى أن يجعله خالصا لوجهه الكريم، وأن يجعله في ميزان حسناتنا يوم لا ينفع مال ولا بنون إلا من أتى الله بقلب سليم. والحمد لله أولاً وآخراً، وهو من وراء القصد، وهو يهدي السبيل.

وفيما يلي قائمة بعناوين الوحدات الواردة في هذا المقرر مع أرقام صفحاتها.

محتويات المقرر

الوحدة	اسم الوحدة	الصفحة
1	مبادئ لغة جافا	**1**
2	البرمجة بالكائنات	**137**
3	Graphics and Graphical User InterFace(GUI)	**189**
4	موضوعات متقدمة في لغة جافا	**241**


الوحدة الأولى
مبادئ لغة جافا

محتويات الوحدة

الموضوع	الصفحة
المقدمة	4
تمهيد	4
أهداف الوحدة	5
1. مميزات ومكونات لغة جافا	7
2. أنواع البيانات الأساسية في لغة جافا	10
1.2 تعريف المتغيرات	12
2.2 تعريف الثوابت	15
3. العمليات الرياضية التي توفرها جافا	16
4. قراءة البيانات من المستخدم	25
5. التعبيرات المنطقية والشروط	30
6.اختصار الاختبارات على المتغير	38
7. العبارات التكرارية	44
1.7 العبارة التكرارية for	44
2.7 العبارة التكرارية while	60
3.7 العبارة التكرارية do – while	63
4.7 ملاحظات حول العبارات التكرارية الثلاث	66
8. تخزين مجموعات البيانات	67
9. الدوال في جافا	87
10. النـــداء الذاتي	108
الخلاصة	113
لمحة مسبقة عن الوحدة الدراسية التالية	118

إجابات التدريبات	119
مسرد المصطلحات	129
المراجع	130

المقدمة

تمهيد

عزيزي الدارس

مرحباً بك في الوحدة الأولى من مقرر **"البرمجة بلغة جافا"** والتي تحمل عنوان "مبادئ لغة جافا". تبدأ الوحدة بخلفية تاريخية عن ظهور لغة جافا والغرض منها ومفهوم البرنامج، وتحتوي هذه الوحدة على عشرة أقسام تشتمل على أساسيات لغة جافا كدعامة أساسية تستند عليها جميع المفاهيم الأخرى باللغة ،ويتم تناول كل هذه الأقسام مع شرح وافٍ مصحوب بالأمثلة.

يتناول القسم الأول أهم مميزات لغة Java، كما يتناول مكونات لغة Java . أما القسم الثاني فيتناول أنواع البيانات الأساسية في لغة Java، وفيه أيضاً نتناول تعريف المتغيرات، وتعريف الثوابت ونجد هنا تعريفاً لبعض الثوابت، ويأتي القسم الثالث متناولاً العمليات الرياضية التي توفرها Java وهي العمليات الرياضية الأربع المعروفة إضافة إلى عمليات باقي القسمة ... وكيفية كتابة عبارات هذه العمليات بصورة مختصرة، أما القسم الرابع فيتناول قراءة البيانات من المستخدم في لغة Java حيث نجد عدة دوال تستخدم لقراءة البيانات، وكل دالة تقرأ قيمة بيانات من نوع معين وتخزنها في متغير من نفس النوع، ويأتي القسم الخامس متناولاً التعبيرات المنطقية والشروط ، أما القسم السادس من هذه الوحدة فيتناول اختصار الاختبارات على المتغير، أما القسم السابع فيتناول العبارات التكرارية وفيها يتم التناول بالتفصيل لثلاث عبارات مختلفة للتكرار توفرها لغة Java، وفي القسم الثامن يتم تناول تخزين مجموعات البيانات، أما القسم التاسع فيأتي متناولاً الدوال في Java، أما القسم العاشر والأخير فيتناول النداء الذاتي.

أهداف الوحدة



عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

- تعدد مميزات لغة جافا.
- تعدد مكونات لغة جافا.
- تذكر شروط تركيب الكلمات في لغة جافا.
- تعدد أنواع البيانات الأساسية في لغة جافا.
- تعرف المتغيرات والثوابت وفق شروط لغة جافا.
- تكتب العبارات الرياضية بصورة مختصرة .
- تعدد الأدوات التي بها يمكن وضع الشروط أو التعبيرات المنطقية
- تستخدم طرق اختصار الاختبارات على المتغير التي توفرها جافا.
- تعبر وبصورة سليمة عن أي فكرة بها تكرار بأي من العبارات التكرارية الثلاثة التي توفرها جافا .
- تستخدم المصفوفات لتخزين مجموعات البيانات .
- تستخدم المتغير من النوع string لتخزين مجموعات الحروف.
- تستخدم حلقة for للتعامل مع جميع عنا صر المصفوفة .
- تذكر شروط استخدام الدوال في جافا .
- تستخدم الدوال بصورة صحيحة .
- تصمم الدوال الخاصة بك .

توطئة

البرنامج هو مجموعة من التعليمات أو الأوامر نعطيها للحاسوب ليؤدي وظيفة معينة. وبرامج الحاسوب هذه يمكن أن نكتبها بأي من لغات البرمجة الموجودة. لغة البرمجة هي التي تخضع البرنامج لقوانينها وقواعدها، وأي برنامج حاسوب مكتوب بلغة برمجة معينة يجب أن يلتزم بقواعد هذه اللغة ولا يخرج عنها، وإلا فلن ينفذ البرنامج.

تعتبر لغة جافا من اللغات الحديثة جداً في عالم البرمجة، حيث ظهرت بصورة رسمية عام 1995م وقد قامت بوضع مفاهيمها شركة Sun Microsystems. وكان الغرض من ابتكار هذه اللغة برمجة صفحات الإنترنت. انتشرت لغة جافا حول العالم بسرعة كبيرة مع انتشار برمجة صفحات الإنترنت وبرمجة التطبيقات الحديثة الأخرى التي توفرها اللغة مثل برمجة شرائح الهاتف المحمول والبيجر والحواسيب الدفترية وغيرها.

1. مميزات ومكونات لغة جافا

- إنها لغة قوية تحتوي على أدوات كثيرة تساعد في كتابة البرامج.
- لكون جافا لغة حديثة مكنها من تلافي عيوب كثير من اللغات قبلها، من أهم هذه العيوب إمكانية الوصول المباشر لمواقع الذاكرة الخاصة بالبرنامج والذي يؤدي إلى ضعف سرية المعلومات وسهولة تدميرها.
- إن البرنامج المكتوب بلغة جافا يمكن نقله وتشغيله على جهاز حاسوب آخر يحتوي على نظام تشغيل يختلف عن الحاسوب الأول (مثلاً يحتوي Windows, Linux وغيرهما) بدون مشاكل.
- تعتبر لغة جافا لغة برمجة بالكائنات (Object Oriented Programming Language) (راجع الباب الثاني)، ويعتبر هذا الصنف من لغات البرمجة من أوسعها انتشاراً وأكثرها استخدماً اليوم.

لغة جافا كغيرها من لغات البرمجة لاتخلو من العيوب، ويكمن اعتبار لغة جافا بطيئة نسبياً. إن السرعة ميزة مهمة، ولكن يجب التضحية ببعض المميزات لاكتساب مميزات أهم.

ذكرنا أن البرنامج عبارة عن تعليمات تخضع لقواعد لغة معينة. ولغة البرمجة كأي لغة طبيعية أخرى تتكون من جمل، والجمل تتكون من كلمات، والكلمة عبارة عن مجموعة حروف من أبجديات اللغة. لذلك دعونا نتعرف أولاً على الحروف التي تتركب منها هذه اللغة.

- الأرقام: من 0..9.
- الحروف مثل: A ... Z, a ... z
- الرموز والعلامات الخاصة مثل: الأقواس (,) , { , } ,...

العلامات الرياضية مثل: + ، − ، x ، / ، = ، وغيرها

علامات الترقيم مثل: ; ، . ، , ، ...

هذه الحروف هي التي تتكون منها الكلمات والعبارات التي تشكل تعليمات لغة جافا.

أما الكلمات فيمكن أن تتركب من أي مجموعة من الحروف أو الأرقام أو العلامة $ أو العلامة _ (underscore) بالشروط التالية:

1. ألا يبدأ الاسم برقم.
2. ألا يحتوي على حرف خاص أو مسافة space.

أمثلة لبعض الأسماء الصحيحة في لغة جافا:

count, student_name, sum1

أمثلة لبعض الأسماء غير الصحيحة في لغة جافا:

Count*, student name, 1sum.

هناك بعض الكلمات التي عرّفتها اللغة مسبقاً لتحمل معاني معينة عند ظهورها في البرنامج. ولا يجوز استخدامها من قبل المبرمج إلا لما خصصت له. تعرف هذه الكلمات بالكلمات المحجوزة (reserved words)، ولمعرفة الكلمات المحجوزة في لغة جافا يمكن الرجوع إلى الملحق الأول.

تعتبر لغة جافا (case sensitive) ، وهذا التعبير يعني أنها تفرّق بين الحروف الكبيرة Capital والصغيرة Small، أي أن الكلمة name تختلف تماماً عن الكلمة Name أو الكلمة NAME وبالتالي يجب الانتباه خصوصاً عند التعامل مع الكلمات المحجوزة، فيجب كتابتها بصورة صحيحة تماماً.

بإمكاننا الآن النظر إلى أول برنامج مكتوب بلغة جافا وشرح مفرداته ومحتوياته.

```
1.  class First
2.  {
3.      public static void main(String args[])
4.      {
5.          System.out.print("my first Java program");
6.      } //end of main
7.  }// end of Class
```

يبدأ برنامج **جافا** بالكلمة المحجوزة **class** وهي ضرورية في برنامج **جافا** كما سنرى في الباب الثاني. يلي هذه الكلمة الاسم الذي يختاره المبرمج لهذا الـــ **class**، وهنا أسميناه First وكان بالإمكان أن نسميه أي اسم آخر مثل MyProgram أو Welcome أو خلافه. محتويات هذا الـــ **class** تكتب بين قوسين { } وهنا محتويات الـــ **class هي** الأسطر 3 إلى 6. في السطر 3 نجد تعريف الدالة الرئيسية main، وهو الجزء الذي يبدأ منه تنفيذ البرنامج. الكلمات **public**, **static**, **void**, main جميعها كلمات محجوزة، وسنتعرف لاحقاً على معاني هذه المفردات.

محتويات الجزء main كما نلاحظ نضعه أيضاً بين قوسين { } وفي هذا المثال محتويات الجزء main موجودة على السطر 5، وحقيقة فإن السطر 5 يحتوي على الأمر المباشر الوحيد في هذا البرنامج، وهو عبارة عن أمر للطباعة على الشاشة، قمنا بتنفيذ هذا الأمر مستخدمين دالة موجودة في لغة **جافا** يكون نداؤها على الصورة:

System.out.print ("هنا نضع النص المطلوب طباعته");

نلاحظ أن النص المراد ظهوره على الشاشة نضعه بين قوسين " " ويسمى نص text أو سلسلة حروف **String**. كما ونلاحظ أن الأمر في لغة **جافا** ينتهي بفاصلة منقوطة (; semicolon). في السطر 6 و7 نلاحظ وجود العلامة // وهذه العلامة نضعها لكتابة التعليقات أو الملاحظات في البرنامج فقط للتوضيح، ويتم تجاهل كل ما يكون بعد العلامة // حتى نهاية السطر. أما لكتابة التعليقات الطويلة أو التي تتكون من عدة أسطر فيمكن كتابتها بين العلامتين /* و */ أو /* و **/ كالآتي:

```
/* This is my first program in java.
 It is easy.
Good bye */
```

الناتج من تنفيذ هذا البرنامج هو طباعة العبارة my first Java program على الشاشة.

أسئلة تقويم ذاتي



1. تعتبر لغة جافا من اللغات ..... في عـــالم البرمجـــة، حيـــث ظهرت بصورة رســـمية عـــام ..... وقـــد قامـــت بوضـــع مفاهيمهـــا شركة..... وكان الغرض من ابتكار هذه اللغة برمجة .....
2. تتميز لغة جافا بمزايا كثيرة اذكر بعضها.
3. تعتبر لغة جافا (case sensitive). ماذا يعني هذا التعبير ؟

2. أنواع البيانات الأساسية في لغة جافا

عزيزي الدارس،

لإجراء أي عملية معينة أو تنفيذ برنامج محدد نحتاج للبيانات، فمثلاً لجمع عددين نحتاج لتعريف متغيرين يحملان قيمة عددية معينة، وقد نحتاج لوضع حاصل الجمع في متغير ثالث تتغير محتوياته بتغير قيمتي العددين. وعموماً لكي يتم الاحتفاظ بأي بيانات في الذاكرة نحتاج للمتغيرات variables، والثوابت constants، ويتم تعريف هذه المتغيرات والثوابت بحيث يمكن أن تحمل بيانات من أنواع معينة.

- الحرف character type (**char**)

المتغير المعرف من النوع **char** يستطيع أن يحمل حرف واحد يكتب بين قوسين صغيرين مثلاً:

'.' , 'Q', 'a', …

مجموعة الحروف الصحيحة لهذا النوع هي التي تعرفها الشفرة الموحدة للحروف Unicode (راجع الملحق الثاني) وتشمل حروف وأرقام ورموز جميع لغات العالم المعروفة. طول المتغير المعرف من نوع **char** هو 2 byte في الذاكرة.

- العدد الصحيح integer type :

ويشمل الأعداد الصحيحة مثل 2 ، 100 وله أربعة أنواع:

النوع	طوله في الذاكرة
Byte	1 byte
short	2 byte
int	4 byte
long	8 byte

- العدد الحقيقي real type

ويشمل الحروف الحقيقية (الكسرية) مثل 1.5 ، 1.3 ، 7- ، ... . وله نوعان:

النوع	طوله في الذاكرة
Float	4 byte
Double	8 byte

- النوع المنطقي Boolean type (**boolean**)

والمتغير من هذا النوع يحتوي على إحدى قيمتين: خطأً **false** أو صواب **true** .

هناك نوعٌ من البيانات شائع الاستخدام رغم أنه ليس من أنواع البيانات الأساسية في جافا وهو النص **String**، والمتغير من هذا النوع يحتوي على نص أو مجموعة حروف تكتب بين تنصيصين، مثلاً:"Java Language" أو "Sudan"

وقد لاحظنا أن الدالة print في المثال السابق تتعامل مع **String**، حيث يتم كتابة النص بين قوسين فتقوم بطباعته على الشاشة.

أسئلة تقويم ذاتي



1. طول المتغير المعرف من نوع **char** هو.....
2. أكمل الجدول التالي والذي يمثل أنواع المتغيرات التي ضمن integer type

النوع	طوله في الذاكرة
byte	.....
.....	2 byte
int	.....
long	.....

3. اذكر انواع المتغيرات التي ضمن العدد الحقيقي real type وأطوالها في الذاكرة.

1.2 تعريف المتغيرات

لتعريف أي متغير في البرنامج، نكتب أولاً نوع البيانات التي سيحتويها هذا المتغير، ثم نكتب اسم المتغير شرط أن يكون اسماً صحيحاً في لغة **جافا** حسب ما ورد سلفاً. ويفضل أن يكون اسماً ذا معنى يعرف منه الغرض من المتغير. عادةً يكتب اسم المتغير بحروف صغيرة، وإذا كان اسم المتغير يتكون من أكثر من كلمة فتكتب بداية كل كلمة بحرف كبير Capital. ولا ننسى وضع فاصلة منقوطة في نهاية الأمر.

مثال لتعريف متغير:

```
int number;
```

كذلك يمكن تعريف أكثر من متغير مباشرة:

```
int number1, number2;
double price;
char letter;
String studentName;
boolean endOfGame;
```

لوضع قيمة معينة في متغير تستخدم العلامة = وتأخذ الصورة:

variable name = value;

مثلاً:

number = 6;

وهذا يعني وضع القيمة 6 في المتغير المسمى number، ويجب أن ننتبه ونراعي نوع المتغير من حيث قابليته لاحتواء هذه القيمة، فمثلاً لا يمكن أن نضع القيمة 6 في متغير من نوع **boolean** وهكذا.

لوضع قيمة ابتدائية في متغير، نحدد هذه القيمة مباشرةً عند تعريفه. مثلاً:

```
int number1 = 3, number2 = 4;
double price = 44.50;
char letter = 'w';
String studentName = "Ali";
boolean endOfGame = false;
```

مثال:

```
1.      class Today
2.      {
3.              public static void main(String args[])
4.              {
5.                      String today="Friday";
6.                      System.out.print("Today is ");
7.                      System.out.print(today);
8.                      today ="Sunday";
9.                      System.out.print(" Today is not ");
10.                     System.out.print(today);
11.             }
12.     }
```

في السطر 6 تم طباعة رسالة نصية today is، ونستدل على أنها نص بوجود قوسي التنصيص. وفي السطر 7 تمت طباعة قيمة المتغير today وهو متغير تم تعريفه مسبقاً ويحمل القيمة "Friday". إذن سيقوم البرنامج بطباعة الكلمة Friday. وفي السطر 8 تم تغيير قيمة المتغيرtoday من Friday إلى Sunday وعليه وفي السطر 10 ستتم طباعة كلمة Sunday وليس Friday، لتكون بذلك مخرجات البرنامج هي:

Today is Friday Today is not Sunday

يمكن تحسين البرنامج السابق وذلك باستخدام الدالة print لتقوم بطباعة الرسالة النصية "Today" إضافة إلى قيمة المتغير today في نفس عبارة واحدة. نحن نعلم أن الدالة print تستقبل بين قوسيها نصاً (إذا كان هذا النص رسالة نصية أو متغير)، ونحن نريد طباعة معلومتين في نفس الوقت. علينا أولاً أن نجري عملية بسيطة، وهي أن نقوم بدمج النصين ليصبحا نصاً واحداً حتى نستطيع استخدام الدالة print. لدمج نصين نستخدم العلامة + كالآتي:

إذا كانت srting1 = "Hello"

وكانت string2 = "student"

وإذا كانت string3 = string1 + string2

فإن قيمة string3 = "Hello student"

طريقة أخرى:

إذا كانت string1 = "Su"

وكانت string2 = string1 + "dan"

فإن قيمة string2 = "Sudan"

وبنفس الطريقة بإمكاننا تعديل المثال السابق ليصبح شكله كالآتي:

```
class today
{
        public static void main(String args[])
        {
                String today = "Friday";
                System.out.print ("Today is " + today);
                today="Sunday";
                System.out.print (" Today is not " + today);
        }
}
```

هناك دالة أخرى تشبه الدالة print وهي println وهي تقوم بنفس وظيفة الدالة print إلا أنها بعد الفراغ من عملية الطباعة تنتقل إلى سطر جديد في الشاشة. مثلاً: إذا استبدلنا دالة print في حل المثال أعلاه بدالة println تصبح مخرجاته كالآتي:

```
Today is Friday
Today is not Sunday
```

أسئلة تقويم ذاتي



1. ما الشروط تعريف أي متغير في البرنامج ؟
2. أكمل الآتي :
   - لوضع قيمة معينة في متغير تستخدم العلامة .....
   - لدمج نصين نستخدم العلامة .....
3. وضح الفرق بين وظيفتي الدالتين print و println .

2.2 تعريف الثوابت

إذا كان بإمكاننا تغيير قيمة المتغير المعين في أي لحظة أثناء البرنامج، فإن الثابت هو الذي لا يمكن تغيير قيمته في البرنامج، ولكننا فقط نقوم بتعريفه ووضع قيمة ابتدائية له لحظة التعريف، وتظل هذه القيمة ثابتة طوال البرنامج. تعريف الثوابت لا يختلف عن المتغيرات إلا في الكلمة المحجوزة **final** والتي نكتبها أمام التعريف لنستدل بها على أنه ثابت.

تعريف لبعض الثوابت:

```
final char PLUS = '+';
final double PI = 3.14;
final String CAPITAL = "Khartoum";
```

وعادةً نكتب اسم الثابت بحروف كبيرة capital لتكون واضحة وبينة داخل البرنامج، ولكن هذا ليس شرطاً. بعد أن نعطي الثابت قيمته الابتدائية تدل كلمة **final** على أن هذا

الثابت لا يمكن أن نعطيه أي قيمة جديدة خلال البرنامج، فلو حاولنا كتابة أيٍ من الأوامر التالية داخل البرنامج، تعطينا جميعها رسائل خطأ:

```
PLUS = '*';
P1 = 0.0;                    أو
CAPITAL = "Medeni";          أو
```

أسئلة تقويم ذاتي



1. وضح الفرق بين تعريف الثوابت وتعريف المتغيرات .

3. العمليات الرياضية التي توفرها جافا

الجدول التالي يبين العمليات الرياضية المختلفة التي توفرها لغة **جافا:**

العملية	العلامة	طريق الكتابة
الجمع Addition	+	a + b
الطرح Subtraction	−	a −b
الضرب Multiplication	*	a * b
القسمة Division	/	a/b
باقي القسمة Modulus	%	a % b

وهي العمليات الرياضية الأربع المعروفة إضافة إلى عملية باقي القسمة. وتعطي الباقي الصحيح لحاصل قسمة العددين، مثلاً:

9/5 = 1 والباقي 4

10/3 = 3 والباقي 1

8/2 = 4 والباقي 0

9%5 = 4 وعليه

10%3 = 1
8%2 = 0

إذا كان المعامل a والمعامل b عددين صحيحين يكون ناتج إجراء جميع هذه العمليات عدداً صحيحاً أيضا. ولذلك نلاحظ أن ناتج إجراء العملية 15/2 يساوي 7 وليس 7.5 لأن الناتج عبارة عن عدد صحيح أيضا. أي أنه خارج القسمة الصحيح. أما إذا كان المعامل a والمعامل b عددين حقيقيين، يكون ناتج جميع العمليات عدد حقيقي أيضا. وبالتالي فإن ناتج إجراء العملية 15.0/2.0 يساوي 7.5 لأن الناتج عدد حقيقي أيضا. هنالك حالات حين يكون أحد المعاملين من نوع صحيح والآخر من نوع حقيقي. في هذه الحالة يكون الناتج من النوع الحقيقي. ومن الواضح أن ذلك أفضل لأن النوع الحقيقي يحتوي على الجزء الكسري، ولو كانت النتيجة من النوع الصحيح لتم اقتطاع هذا لجزء.

مثلاً: 15.0/2 تساوي 7.5 وليس 7

وكذلك 15/2.0 أيضا تساوي 7.5

كما في الجبر الرياضي. عمليات الضرب والقسمة وباقي القسمة تسبق عمليتي الجمع والطرح، ويمكن استخدام الأقواس لتحديد أولويات إجراء العمليات مثلاً: 2 * 7 + 8

هنا يتم إجراء عملية الضرب أولاً. وإذا أردنا الجمع أولا نكتب:

2 * (7 + 8)

مثال:

```
1.    class Mathematics
2.    {
3.          public static void main(String args[])
4.          {
5.                int y,x =5;
6.                y= x * 6 + ( 2 - 4 ) * 3;
7.                System.out.print("y = "  + y);
8.          }
9.    }
10.
```

في السطر 6 قمنا بتعريف متغيرين، لأحدهما x قيمة ابتدائية. في السطر 7 تم حساب قيمة المتغير الآخر y من خلال تعبير رياضي. من الطبيعي أن يتم إجراء العمليات بالتتالي من اليسار إلى اليمين، وتأجيل عمليتي الجمع والطرح. هنا تم حساب حاصل الضرب x*6 وهو 5*6=30. وهناك عملية ضرب أخرى، لكننا نقوم بحساب ما بين القوسين أولاً وهو 2-=4-2. ثم إجراء عملية الضرب التي أصبحت 6-=3*2-. والآن بقيت لدينا عملية جمع واحدة وهي 24=(6-)+(30) وهي القيمة النهائية للمتغير y والتي تمت طباعتها في السطر 8 حيث نلاحظ أن العلامة + استخدمت للدمج بين الرسالة النصية " y =" وقيمة المتغير y رغم أنها ليست **String**. لكن هنا يحدث تحويل ضمني لقيمة المتغير y وهي 24 إلى النص "24" تلقائياً،ثم يتم دمجهما مع النص " y =" لتكون مخرجات البرنامج هي y = 24.

لاحظنا في المثال السابق أن المتغير y حصل على قيمته بإجراء تعديل على قيمة المتغير x. من الممكن أن نحصل على قيمة متغير بإجراء تعديلات على قيمته الحالية للحصول على القيمة الجديدة. فمثلاً في العبارة:

n = n + 4;

إذا كانت قيمة n هي 2 فيمكننا الحصول على قيمة المتغير n الجديدة بإضافة 4 إلى القيمة القديمة لتكون n=6. يمكن كتابة العبارة السابقة بصورة مختصرة وهي:

n+ = 4;

وبنفس الطريقة:

x- =3; تكتب x = x-3;

i* = 6i; تكتب i = i*6i;

a/ =2; تكتب a = a/2;

k%=5; تكتب k= k%5;

هناك عبارة رياضية كثيرة الاستخدام وهي تقوم بزيادة قيمة المتغير بواحد (x = x + 1;) أو إنقاص قيمته بواحد (x = x - 1;). ولذلك فبالإضافة للاختصار السابق، هناك اختصار أفضل لهاتين العبارتين:

(x = x + 1;) تكتب x ++ أو ++ x

أما (x = x - 1;) فتكتب x -- أو -- x

x++ تعني أن يتم إضافة 1 إلى المتغير x بعد إكمال أي عمليات أخرى في العبارة. أما x ++ فتعني إضافة 1 إلى قيمة المتغير x قبل تنفيذ باقي التعبير. نوضح ذلك بمثال.

إذا كان:

```
1.      int x = 6;
2.      System.out.print (x ++);
3.      System.out.print (x);
```

هنا x++ تعني إضافة 1 لقيمة المتغير x (6) بعد إكمال عملية الطباعة. وعليه فإن الناتج من تنفيذ السطر 2 هو طباعة الرقم 6. ثم بعد ذلك يتم إضافة 1 للمتغير x لتكون قيمته الجديدة هي 7=1+6. وبالتالي عند إجراء عملية الطباعة في السطر 3 تكون المخرجات 7.

إذن المخرجات هي: 6 7

والأمر سيان بالنسبة لـــ - -x . وإذا فرضنا أن:

```
1.      x = 3;
2.      System.out.print (--x);
3.      System.out.print (x);
```

في هذه الحالة x-- تعني إنقاص قيمة المتغير x بواحد قبل إجراء عملية الطباعة. عليه يتم حساب القيمة الجديدة للمتغير x أولا وهي 2=1-3. ثم بعد ذلك يتم إجراء عملية الطباعة في السطر 2 وتمون المخرجات 2. بعد ذلك يتم الانتقال للسطر 3 حيث تتم طباعة قيمة المتغير x وهي 2 نفسها.

إذن المخرجات هي:

2 2

نحتاج أحياناً إلى وضع قيمة متغير من نوع محدد في متغير آخر من نوع مختلف. يمكن وضع محتويات المتغير الصغير في المتغير الكبير بلا مشاكل، مثلاً:

```
int i = 1500;
long number;
number = i;
```

لأنه مهما كانت قيمة المتغير i كبيرة فإن المتغير number قادر على احتوائها، حيث إن سعة المتغير من النوع **long** أكبر من سعة المتغير من النوع **int**.

مثال آخر:

```
double d;
int i= 1000;
d = i;
```

أيضاً المتغير من النوع **double** يستطيع أن يضع قيمة متغير من نوع **int** في الجزء الصحيح من المتغير. هذا النوع من التحويل (من النوع الأصغر إلى النوع الأكبر) يحدث ضمنياً. لكننا قد نحتاج إلى إجراء العكس، أي وضع قيمة المتغير الكبير في المتغير الصغير. في هذه الحالة يجب التعبير صراحة عن هذا التحويل في نص البرنامج، فإذا كان:

```
int i;
long number = 5000;
```

فإن العبارة:

```
i = number;
```

هي عبارة غير مقبولة، حيث إننا لا نضمن أن تكون قيمة المتغيرnumber دائماً صغيرة بما يكفي لكي يستطيع المتغير i احتواءها. ولكن يكتب هذا الأمر كالآتي:

```
i = (int) number;
```

وهي عبارة تعني تحويل القيمة **long** الموجودة في المتغير number إلى النوع **int**، ثم وضعها في المتغير i.

مثال آخر:

```
1.      double d = 1000;
2.      int i;
3.      i  = d ;
```

العبارة في السطر 3 خاطئة، خصوصاً أن التحويل هنا قد يؤدي إلى فقد الخانات الكسرية بالعدد الحقيقي، لذلك تعتبر غير مقبولة. رغم ذلك لو تعمد المبرمج هذا التحويل، وكان على علم تام بهذا الفقد فله ذلك، ولكن عليه كتابة أمر التحويل صراحة. ويتم كتابة الأمر بصورة صحيحة كالآتي:

```
3. i=(int) d;
```

حيث سيتم تحويل القيمة الحقيقية الموجودة في المتغير d إلى قيمة **int** (وبالتالي اقتطاع الجزء الكسري) . ثم وضع هذه القيمة في المتغير الصحيح i.

هناك نوع خاص من التحويل، وهو التحويل من حرف **char** إلى عدد صحيح **int** أو العكس. ولشرحه نستعين بالمثال التالي:

```
1.      char letter = 'A';
2.      int i = (int) letter;
3.      System.out.print (i);
```

في السطر 1 تعريف لمتغير حرفي بقيمة ابتدائية 'A'. وفي السطر 2 تعريف لعدد صحيح له قيمة ابتدائية، تم الحصول على هذه القيمة الابتدائية من تحويل المتغير letterإلى عدد. في هذه الحالة أي عند تحويل الحرف إلى عدد صحيح ينتج العدد الذي يقابل ذلك الحرف في شفرة Unicode (راجع الملحق الثالث) وفي هذا المثال ينتج مقابل الحرف 'A' في شفرة Unicode وهو الرقم65. وهو ما تطبعه الدالة في السطر3.

وعند التحويل من عدد إلى حرف يحدث العكس، أي أنه ينتج الحرف المقابل لذلك العدد في شفرة Unicode مثلاً:

97(**char**) هو الحرف 'a'

لأن ترتيب الحرف 'a' في شفرة Unicode هو97.

مثال:

المطلوب كتابة برنامج يحسب الراتب السنوي للموظف ويطبعه في رسالة مناسبة، علماً بأن راتبه الشهري هو 375.

الحل:

```
1.      class Salary
2.      {
3.          public static void main (String args[])
4.          {
5.                  int sal = 375*12;
6.                  System.out.print ("Employee annual salary is:" + sal);
7.          }
8.      }
```

في السطر 5 تم تعريف متغير، وفيه تم حساب الراتب السنوي مباشرةً. يمكن أيضاً أن يتم هذا الحساب في خطوتين:

```
int sal;
sal = 375*12;
```

كلاهما صحيحتان، ولكن الطريقة الأولى أفضل وأسرع.

مخرجات البرنامج في هذه الحالة:

```
Employee annual salary is: 4500
```

مثال:

لدينا 463 يوم، ونريد أن نعرف عدد السنين والشهور والأيام الموجودة فيها. بافتراض أن عدد أيام الشهر = 30يوم.

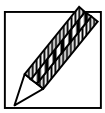
الحل:

```
1.      class Days
2.      {
3.              public static void main (String args[])
4.              {
5.                      int days = 463;
6.                      int months, years;
7.                      years = days/360;
8.                      days %= 360;
9.                      months = days/30;
10.                     days %= 30;
11.                     System.out.println("There are " + years + " years and
                        " + months + " months and " + days + " days");
12.             }
13.     }
```

في السطر 7، حصلنا على عدد السنين من قسمة عدد الأيام على360. والباقي من هذه الأيام هو الذي ستحسب منه الشهور والأيام. هذا الباقي حصلنا عليه بإجراء عملية باقي قسمة الأيام على 360 (السطر 8) ووضعناها في نفس المتغير days. إذن المتغير days الآن يحتوي على ما تبقى من أيام بعد أن اقتطعنا منه السنوات، ونريد أن نستخلص منها الشهور. ولذلك في السطر 9 حسبنا الشهور. وذلك بقسمة الأيام على30. أما باقي القسمة والذي حسبناه في السطر 10، فهو يمثل الأيام. لتكون المخرجات هي:

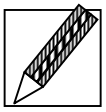
There are: 1 years and 3 months and 13 days

تدريب (1)



المطلوب كتابة برنامج يحسب مساحة الدائرة التي نصف قطرها 6، ومن ثم يطبعه.

تدريب (2)



المطلوب كتابة برنامج يحسب ويطبع المقاومة المكافئة لمقاومتين R1=5 و R2=2 إذا كانتا موصلتين على التوازي R= (R1+R2) / (R1*R2)

مثال:

أراد إخوة (ثلاثة نساء ورجلان) اقتسام ميراثهم من والدهم المقدر بـــ 14000دينار، والذي توفى وعليه دين قدره 1190دينار المطلوب طباعة نصيب المرأة الواحدة ونصيب الرجل الواحد افتراضا بأنه لا يوجد ورثة غيرهم وللذكر مثل حظ الانثيين.

الحل:

```
1.          class Inheritance
2.          {
3.                  public static void main (String args[])
4.                  {
5.                          double money = 14000, dept = 1190;
6.                          double maleShare, femaleShare;
7.                          int noOfMale = 2;
8.                          int noOfFemale = 3;
9.                          money -= dept;
10.                         femaleShare = money/( noOfFemale +
                            noOfMale *2);
11.                         maleShare = femaleShare*2;
12.                         System.out.println("Every male takes: "
                            + maleShare + " Diners");
13.                         System.out.println("Every female takes: " +
                            femaleShare + " Diners");
14.                 }
15.     }
```

قمنا بتعريف المتغيرات اللازمة لحل البرنامج. money يحتوي على المبلغ، dept يحتوي على الدين، عدد الرجال والنساء كلٌ في متغير، ومتغيران ليحملا حصة كل منهما.

بداية نطرح الديون من المبلغ ليتبقى صافي الميراث الذي سنقسمه على الإخوة. في السطر 10 قمنا بحساب نصيب كل امرأة وذلك عن طريق قسمة الميراث على عدد الأبناء الكلي (ضربنا عدد الرجال في 2 لأن نصيبهم هو نصيب امرأتين) ثم حصلنا على نصيب الرجل بضرب نصيب المرأة في 2، ثم بطبعنا هذه القيم.

مخرجات البرنامج:

```
Every male takes: 3660.0 Diners
Every female takes: 1830.0 Diners
```

أسئلة تقويم ذاتي

1- أكمل الجدول التالي والذي يبين العمليات الرياضية المختلفة التي توفرها لغة **جافا:**

العملية	العلامة	طريق الكتابة
الجمع Addition	.....	.....
الطرح Subtraction	.....	.....
....	*	a * b
القسمة Division	.....	.....
باقي القسمة Modulus	.....	.....

4. قراءة البيانات من المستخدم

في جميع الدروس السابقة، كانت هناك تعليمات وأوامر معينة يقوم البرنامج بتنفيذها للوصول إلى النتائج المطلوبة. هذه النتائج هي مخرجات البرنامج، والتي يراها من ينفّذ البرنامج على شكل رسالة أو أكثر تظهر على الشاشة، أي أن المستخدم لم يساهم في البرنامج بأية قيم للبيانات. بصورة أخرى أن تلك البرامج لم يكن فيها مدخلات، بل يأخذ البرنامج قيم البيانات من متغيرات مخزنة القيم داخل البرنامج.

مدخلات البرنامج هي قيم للبيانات يتم إدخالها للبرنامج بعد بداية تشغيله. بعد أن يقوم المستخدم بتشغيل البرنامج، يطلب منه البرنامج إدخال بيانات معينة بواسطة لوحة المفاتيح. يقوم البرنامج باستلام هذه البيانات وتخزينها في متغيرات يستفيد منها البرنامج خلال التنفيذ.

إن لمقدرة لغات البرمجة على قراءة البيانات من المستخدم أهمية كبيرة، فهو يمثل التفاعل أو التخاطب المباشر بين البرنامج ومستخدمه. وتختلف مخرجات البرنامج باختلاف المدخلات، لأن البرنامج يمكن أن يقوم بعمليات مختلفة حسب المدخلات، ويتوصل إلى قيم مختلفة كذلك. فكيف يتم قراءة البيانات من المستخدم في لغة **جافا؟**

نظراً لتعدد وسائل إدخال البيانات إلى برنامج **جافا** مثل: (لوحة المفاتيح، الملفات المختلفة، الأقراص المرنة، الـ CDs، وغيرها ...)، فقد وفّرت لغة **جافا** الكثير من الأدوات التي تمكّن المبرمج من بناء دواله الخاصة لقراءة وتحديد مصدرها بنفسه (في هذه المرحلة لوحة المفاتيح). وسوف نقوم خلال الباب الرابع بشرح تفاصيل قراءة البيانات من لوحة المفاتيح والملفات. ولأغراض هذا الدرس نستخدم برنامج 4-3-3 والمعد خصيصاً لقراءة البيانات المختلفة من المستخدم، وذلك دون الخوض في تفاصيله التي نتركها لحينها. ولاستخدام البرنامج في الوقت الحالي، يجب نسخ البرنامج وحفظه في ملف يسمى **Input.java** في نفس المجلد (folder) الذي يوجد فيه برنامج **جافا** الذي سيستخدم دوال إدخال البيانات. استخدمنا فيما سبق الدَّالتينِ:

```
System.out.print( )
System.out.println( )
```

تستخدم هاتان الدالتان لطباعة المخرجات على الشاشة. وهما كما نعلم تستقبلان قيمة من النوع **String** وإن لم تكن كذلك تقوم بتحويلها ضمنياً إلى **String** ثم تقوم بطباعتها على الشاشة. لقراءة البيانات نستخدم عدة دوال، كل دالة تقرأ قيمة بيانات من نوع معين وتخزنها في متغير من نفس النوع.

- الدالة: **Input**.readString( )

تقرأ قيمة **String** من المستخدم وتضعها في متغير من النوع **String**.

نداء الدالة: name=**Input**.readString( );

حيث name متغير معرف من النوع **String**.

- الدالة: **Input**.readChar( )

تستخدم لقراءة حرف من المستخدم وتضعه في متغير من النوع **char**.

نداء الدالة: key = **Input**.readChar( );

حيث key متغير معرف من النوع **char**.

- الدالة: **Input**.readInt( )

تقرأ عدداً صحيحاً وتضعه في متغير من النوع **int**.

نداء الدالة: x = **Input**.readInt( )

حيث x متغير معرف من النوع **int**.

- الدالة: **Input**.readDouble( )

لقراءة عدد حقيقي تضعه في متغير من النوع **double**.

نداء الدالة: degree = **Input**.readDouble( )

حيث degree متغير النوع **double**.

- الدالة: **Input**.readBoolean( )

وهي دالة تنتظر من المستخدم إدخال قيمة **true** أو **false** وتخزنها في متغير من النوع **boolean**.

نداء الدالة: b= **Input**.readBoolean( )

حيث b هو متغير نوعه **boolean**.

بالطبع كان من الممكن كتابة المزيد من الدوال لقراءة قيم من الأنواع **long**, **float**, **short**, **byte**. ولكننا في هذا الكتاب نكتفي بهذه الأنواع الأساسية.

مثال:

المطلوب كتابة برنامج يستقبل من المستخدم اسمه، ثم يطبعه على الشاشة مع رسالة ترحيبية.

الحل:

```
1.  class HelloUser
2.  {
3.          public static void main (String args[])
4.          {
5.                  String user;
6.                  user = Input.readString( );
7.                  System.out.println("Hello " + user);
8.          }
9.  }
```

عند تنفيذ البرنامج يتوقف المؤشر على الشاشة منتظراً للمستخدم. حتى يقوم بإدخال قيمة نصية. بعد أن يطبع المستخدم اسمه ويضغط على مفتاح الإدخال Enter. يطبع البرنامج الرسالة Hello يليها الاسم الذي طبعه المستخدم. مثلاً إذا أدخل المستخدم الاسم Ali، تكون مخرجات البرنامج هي:

Hello Ali

لاحظ أن مخرجات البرنامج ستختلف باختلاف الاسم الذي أدخله المستخدم.

نلاحظ أيضا أن المستخدم عندما يقوم بتشغيل البرنامج، يجد المؤشر في انتظاره لإدخال قيمةٍ ما. وقد لا يدري المستخدم المطلوب منه تحديداً، لذلك (من ذوق المبرمج) ولكي يكون أكثر وضوحاً، الأفضل أن يضع رسالة نصية توضيحية قبل كل عملية قراءة، يوضح فيها للمستخدم القيمة التي يتوقعها منه المبرمج لكي يقوم بعمله بصورة صحيحة. في هذا المثال يمكننا أن نقوم بإضافة سطر قبل السطر 6 أي قبل قراءة البيانات. ويمكن أن يكون في أي مكان من بداية البرنامج حتى السطر 6، المهم أن يكون قبل عبارة إدخال البيانات. مثلاً:

```
System.out.println("Enter your name, Please");
```

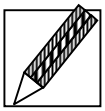
ليبدأ تنفيذ البرنامج برسالة يفهم منها المستخدم أن المطلوب منه في الخطوة التالية كتابة اسمه باستخدام لوحة المفاتيح. وبعد الضغط على مفتاح الإدخال Enter تظهر له المخرجات.

مثال: اكتب برنامج يستقبل حرف، ويطبع مقابل الحرف في شفرة Unicode.

الحل:

```
 class Unicode
{
        public static void main (String args[]);
        {
                System.out.println("Enter any letter to get its Unicode
                number ");
                char letter = Input.readChar( );
                System.out.println("Unicode number for letter: " + letter + "
                is " + (int) letter);
        }
}
```

تدريب (3)



اكتب برنامج يطلب من المستخدم إدخال عددين صحيحين، ويطبع حاصل جمعهما وطرحهما وضربهما وقسمتهما، ثم أعط أي نموذج مختار لتنفيذ البرنامج .

مثال:

اكتب برنامج يطلب من المستخدم إدخال عددين صحيحين، ويطبع حاصل جمعهما وطرحهما وضربهما وقسمتهما.

الحل:

```
class Calculate
{
        public static void main (String args[]);
        {
                System.out.println("Enter a number, please");
                int number1 = Input.readInt( );
                System.out.println("Enter another number, please");
                int number2 = Input.readInt( );
                System.out.println( number1 + "+" + number2 + "=" +
                (number1 + number2) );
                System.out.println( number1 + "-" + number2 + "=" +
```

```
            (number1 - number2) );
            System.out.println( number1 + "*" + number2 + "=" +
            (number1 * number2) );
            System.out.println( number1 + "/" + number2 + "=" +
            (number1 / number2) );
        }
}
```

نموذج لتنفيذ البرنامج:

```
Enter a number, please
10
Enter another number, please
2
10+2=12
10-2=8
10*2=20
10/2=5
```

5. التعبيرات المنطقية والشروط

عزيزي الدارس،

في كثير من الأحيان يتطلب تنفيذ جزء معين من البرنامج تحقق شرط محدد. وقد يتعين على البرنامج فحص المدخلات والاستجابة بتنفيذ الجزء المناسب من البرنامج حسب هذه المدخلات. فلذلك كان لابد من وجود طريقة للاستفسار عن تحقق شرط معين داخل البرنامج والتصرف حسب نتيجة الاستفسار.

بداية نتعرف إلى الأدوات التي بها يمكن وضع الشروط أو التعبيرات المنطقية:

>	أكبر من
>=	أكبر من أو يساوي
<	أصغر من
<=	أصغر من أو يساوي
==	يساوي
!=	لا يساوي

جميع هذه الأدوات تختبر المعاملات، وتحدد ما إذا كان التعبير خطأ أو صواب، وتعرِّف بالأدوات المنطقية.

إذا كان:

x=4, y=5

فإن:

x>y	خطأ
x>=y	خطأ
x<y	صواب
x<=3	خطأ
y == 5	صواب
x!=y	صواب

جميع هذه التعبيرات المنطقية تحمل قيمة منطقية **true** أو **false**. يمكن وضع هذه القيمة في متغير من النوع **boolean**:

boolean b = x>y;

قيمة b هنا **false**.

يمكن أن يكون التعبير المنطقي بأي درجة من التعقيد ، مثلاً:

x+y>10	خطأ
x%2= = 0	صواب
y<=x*2-6	خطأ
x*(y+2)= = y*y+3	صواب
'a'<'b'	صواب

المقارنة بين الحروف تتم حسب ترتيبها في شفرة Unicode (راجع الملحق الثالث).

عبارة الشرط If IF Statement

وتأخذ الصيغة:

```
1.    if (condition)
2.        statement;
```

لوضع شرط على تنفيذ العبارة في السطر 2، نكتب الكلمة المحجوزة if، ثم نضع بين قوسين شرط (condition) ، وهو عبارة عن تعبير منطقي قيمته إما صواب أو خطأ. فإذا كانت قيمته صواب، يتم تنفيذ العبارة في السطر 2، وإذا كانت قيمته خطأ يتم تجاهل السطر 2 تماماً.

مثلاً:

```
System.out.println("Enter a number");
int n= Input.readInt( );
if (n>0)
        System.out.println("This is a positive number");
System.out.println("Thank you");
```

في هذا المثال، إذا أدخل المستخدم عدداً أكبر من صفر، تكون قيمة التعبير المنطقي n>0 صواب وعليه تكون مخرجات البرنامج هي:

```
This is a positive number
Thank you
```

أما إذا قام المستخدم بإدخال عدد أقل من (أو يساوي) صفراً، قيمة التعبير المنطقي n>0 خطأ، وعليه لا يتم تنفيذ عبارة الطباعة. لأن شرط تنفيذها أن تكون قيمة n أكبر من صفر، وعليه تكون مخرجات البرنامج في هذه الحالة هي:

Than you

إذا أردنا طباعة العبارة Thank you فقط إذا كانت قيمة الشرط هي صواب، أي إذا كان ناتج تحقق الشرط n>0 هو تنفيذ أكثر من عبارة واحدة، نضع العبارات المراد تنفيذها بين قوسي بداية ونهاية، كالآتي:

```
if (n>0)
{
        System.out.println("This is a positive number");
        System.out.println("Thank you");
}
```

في هذه الحالة لا تتم طباعة الرسالة Thank you إلا إذا تحقق الشرط، وهي ملازمة للعبارة التي تسبقها، لأن شرط طباعتهما واحد. في هذا المثال إذا كانت قيمة n المدخلة أقل من أو تساوي صفر لن يتم طباعة أية عبارة.

مثال:

عدل البرنامج السابق بحيث إذا تم إدخال عدد سالب يطبع This is a negative number ويطبع رسالة الشكر في نهاية البرنامج.

الحل:

```
1. System.out.println("Enter a number");
2. int n= Input.readInt( );
3. if (n>0)
4.         System.out.println("This is a positive number");
5. if (n<0)
6.         System.out.println("This is a negative number ");
7. System.out.println("Thank you");
```

إذا كان العدد المدخل أكبر من صفر، يكون قيمة الشرط في السطر 3 صواب، فيطبع الرسالة في السطر 4. لكن قيمة الشرط في السطر 5 تكون خطأ، فيتجاهل الرسالة في السطر 6، مثلاً:

```
Enter a number
6
This is a positive number
Thank you
```

أما إذا كان العدد المدخل أقل من صفر، تكون الشرط في السطر 3 خطأ، فيتجاهل أمر الطباعة في السطر 4. أما في السطر 5، فتكون قيمته صواب، وعليه يتم طباعة الرسالة في السطر 6. وفي كل الأحوال يتم طباعة الرسالة الأخيرة، مثلاً:

```
Enter a number
-3
This is a negative number
Thank you
```

نحن نعلم أن العدد إما أن يكون موجباً أو أن يكون سالباً. وليس هناك خيار ثالث، أي إذا كان هناك عدد n قيمة الشرط n>0 له هي خطأ، يكون العدد سالباً تلقائياً. بصيغة أخرى، الشرط n>0 نستفيد منه مرتين، إذا كانت قيمته صواب يكون العدد موجباً، وقد قمنا بتنفيذ عملية نتيجة لذلك. كذلك إذا كانت قيمته خطأ، فهذه معلومة جديدة، وهي أن العدد سالب. نريد أن ننفذ عملية مختلفة إذا كان الشرط خطأ.

الكلمة المحجوزة **else** تلازم عبارة if معينة، وتأخذ الصيغة التالية:

```
if (condition)
        Statement1;
else
        Statement2;
```

عندما تكون قيمة الشرط condition صواباً، يتم تنفيذ statement1 وتجاهل else والعبارة التي تليها. وعندما يكون الشرط condition خطأ يتم تجاهل العبارة statement1وتنفيذ العبارة statement2. وكما في عبارة if، إذا كان المطلوب تنفيذ أكثر من أمر واحد في حالة قيمة الشرط خطأ، توضع الأوامر بين قوسي بداية ونهاية.

مما سبق، يمكن تعديل المثال السابق ليكون أكثر كفاءة لاحتوائه على شرط واحد فقط.

```
System.out.println("Enter a number");
int n= Input.readInt( );
```

```
if (n>0)      // أي أن العدد موجب
        System.out.println("This is a positive  number");
else          // أي أن العدد سالب
        System.out.println("This is a negative number ");
System.out.println("Thank you");
```

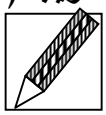
مثال لتنفيذ البرنامج:

```
Enter a number
-2
This is a negative number
Thank you
```

مثال آخر لتنفيذ البرنامج:

```
Enter a number
8
This is a positive number
Thank you
```

تدريب (4)



اكتب جزءاً من برنامج يقرأ عددين صحيحين a و b، إذا كان a أكبر من b، يطبع حاصل طرح a-b، وإذا كان b أكبر من a يطبع حاصل طـــرح b-a. أما إذا كانا متساوين، يطبع رسالة مناسبة، ثم ضع نموذجـــاً لتنفيـــذ برنامج تختاره أنت .

مثال:

اكتب برنامجاً يقرأ عدداً، إذا كان العدد من بين الأعداد 1 إلى 10، يطبع الرسالة In the range، وإلا يطبع Out of rang.

الحل:

في هذا المثال نلاحظ أن الشرط على العدد المدخل أن يكون أكبر من أو مساوياً 1، ويكون كذلك أصغر من أو مساوياً 10. أي أن هناك شرطان يجب تحققهما معاً لطباعة الرسالة.

توفر **جافا** مجموعة من العمليات على العبارات المنطقية لحل هذه المشاكل، وهي مختصرة في الجدول أدناه:

العملية	معناها	قيمتها
&&	x and y	صواب فقط إذا كان كل من x وy صواب
||	x or y	خطأ فقط إذا كان كل من x وy خطأ
!	not z	خطأ إذا كان z صواب، وصواب إذا كان z خطأ

في مثالنا المطلوب ألا نطبع الرسالة إلا إذا كان العدد في المدى المحدد، أي إذا تحقق الشرطان وكانت قيمتهما صواب. وبالتالي نستخدم العملية && (and):

```
1.      System.out.println ("Enter a number");
2.      int num= Input.readInt( );
3.      if ((num>=1) && (num<=10))
4.          System.out.println("In the range");
5.      else
6.          System.out.println("Out of range");
```

هنا لن تتم طباعة العبارة في السطر 4 ما لم يتحقق كل من الشرطين وتكون قيمة كل منهما صواباً. ولا ننسى وضع كل شرط بين قوسين.

مثال:

اكتب برنامجاً يحدد ما إذا كان العدد يقبل القسمة على 6 أو لا.

الحل:

يقبل العدد القسمة على 6 إذا كان يقبل القسمة على 2 و 3 معاً.

```
System.out.println ("Enter a number");
int x= Input.readInt( );
if ((x % 2 == 0) && (x %3 == 0))
        System.out.println ("Number devisable by six");
else
        System.out.println ("Number not devisable by six");
```

مثال:

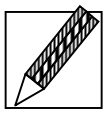
اكتب عبارة تقوم بقراءة درجتي طالب في اختبارين لمادة، وإذا كان قد رسب في أي من الاختبارين يعتبر محروماً من الجلوس للامتحان النهائي للمادة. علماً بأن درجة المرور لكل اختبار 15 من أصل 30 درجة.

الحل:

```
System.out.println ("Enter first test mark");
int first = Input.readInt( );
System.out.println ("Enter second test mark");
int second = Input.readInt( );
if ((first <15) || (second <15))
   System.out.println("This student is not allowed to sit for the final exam");
else
   System.out.println ("This student has passed his class work");
```

في المثال استخدمنا عملية أو (||) لأنه إذا كانت قيمة أيٍ من شرطيها صواب تعتبر صواباً. وبالمثال إذا رسب الطالب في أيٍ من الاختبارين لا يجلس للامتحان النهائي.

تدريب (5)



اكتب برنامجاً يطلب من التاجر إدخال قيمة بضاعته التي حال عليها الحول. يقوم البرنامج بتحديد زكاة أموال التاجر، (الزكاة تساوي ربع العشر). لكن يجب أولاً التأكد من أن الأموال قد بلغت نصاب الزكاة وهو

أسئلة تقويم ذاتي



1- أكمل الجدول التالي والذي بوضح الأدوات التي بها يمكن وضع الشروط أو التعبيرات المنطقية:

.....	أكبر من
>=	.....
.....	أصغر من
<=	أصغر من أو يساوي
==	.....
!=	.....

6.اختصار الاختبارات على المتغير

في بعض الأحيان تكون لدينا عمليات اختبار كثيرة على قيمة متغير. يمكن التعبير عنها بعبارات if-else كثيرة. هناك طريقة أخرى توفرها **جافا** تسهّل وتختصر عمليات الاختبار على المتغير. وهي عبارة **switch**، وتأخذ الصيغة:

```
switch (variable)
{
        case value1: statement1;
                        break;
        case value2: ststment2;
                        break;
          :              :
        default:      statement;

}
```

حيث variable هو اسم المتغير المطلوب إجراء الاختبارات على قيمته، ويشترط فيه أن يكون من النوع **int** أو **char**. value2, value1 عبارة عن قيم يمكن أن يأخذها المتغير. عند إجراء الاختبار على المتغير variable، إذا ساوت قيمته أيّاً من القيم الموجودة بعد كلمة **case**، يتم تنفيذ العبارة أو العبارات التالية حتى الوصول إلى نهاية **switch** أو العثور على الكلمة **break**، والتي تقوم بإيقاف تنفيذ عبارات **case** التالية لعبارة **case**

التي تم تنفيذها. أما إذا احتوى المتغير على قيمة غير موجودة ضمن عبارات **case**، عندئذ يتم تنفيذ العبارات التالية للكلمة المحجوزة **default**.

مثال:

```
int digit =2;
switch (digit)
{
        case 1: System.out.println("one");
                break;
        case 2: System.out.println("two");
                break;
        case 3: System.out.println("three");
                break;
        default:
                System.out.println("no value match");
}
```

المتغير الذي سنختبر قيمته هنا هو digit وهو عدد صحيح يحتوي على قيمة معينة (هنا 2). العبارة 1 **case** تعني في حالة أن المتغير digit يحتوي على القيمة 1، سيتم إجراء عملية طباعة يليها الخروج من عبارة switch كلها، وذلك بسبب وجود **break**. أما إذا لم يحتوِ المتغير digit على القيمة 1، فيتم الانتقال إلى الاختبار التالي. ولأن digit يحتوي على القيمة 2، سيتم طباعة الكلمة two ثم الخروج من **switch**.

إذا افترضنا في المثال السابق أن المتغير يحتوي على القيمة 4، فلن تكون هناك عبارة **case** تحمل القيمة 4 لتنفيذ محتوياتها، وبالتالي يتم تنفيذ العبارات الموجودة داخل الجزء **default**.

مثال:

البرنامج التالي يقرأ حرف S,E,N أو W يرمز لإحدى الاتجاهات الرئيسية الأربعة. يقوم البرنامج بطباعة الاتجاه ككلمة كاملة حسب الحرف المدخل.

الحل:

```
System.out.println("Enter the direction");
char dir=Input.readChar( );
switch(dir)
```

```
{
        case 'N': System.out.println("North");
                  break;
        case 'E': System.out.println("East");
                  break;
        case 'S': System.out.println("South");
                  break;
        case 'W': System.out.println("West");
                  break;
        default: System.out.println("Wrong input");
}
```

لاحظ طريقة كتابة الخيارات، مثلاً 'N' بين قوسين صغيرين لأنه حرف. إذا كان المتغير dir يحتوي على إحدى القيم الأربع، سيطبع الكلمة المقابلة. وإلا سيطبع رسالة خطأ.

هناك عيب في هذا البرنامج، وهو أنه لا يأخذ في الاعتبار إدخال الحروف الصغيرة، وكما نعلم فإنها مختلفة تماماً عن الحروف الكبيرة. فإذا أدخلنا الحرف n، لن يطبع كلمة North، حيث أن n تختلف عن N. ونفس الطريقة لباقي الاختبارات. العبارة (**case** 'n': **case** 'N':) تعني إذا كانت قيمة المتغير dir هي 'N' أو 'n' يطبع الكلمة North. يتم تعديل البرنامج السابق كالآتي:

```
switch(dir)
{
        case 'N': case 'n': System.out.println("North");
                            break;
        case 'E': case 'e':  System.out.println("East");
                            break;
        case 'S': case 's':  System.out.println("South");
                            break;
        case 'W': case 'w': System.out.println("West");
                             break;
        default:                 System.out.println("Wrong input");
}
```

وهكذا إذا تم إدخال الحرف المناسب على صورة حرف كبير capital أو حرف صغير small، ستتم عملية الطباعة المقابلة.

مثال:

اكتب برنامجاً يطلب من المستخدم إدخال عددين صحيحين، ثم إدخال عملية حسابية (+، -، *، /). يقوم البرنامج بحساب وطباعة ناتج إجراء العملية الحسابية على العددين.

الحل:

```
System.out.println("Enter two numbers");
int num1= Input.readInt( );
int num2= Input.readInt( );
System.out.println("Enter operation [+,-,*,/]");
char op= Input.readChar( );
switch (op)
{
        case '+': System.out.println(num1+ num2);
                  break;
        case '-': System.out.println(num1- num2);
                  break;
        case '*': System.out.println(num1* num2);
                  break;
        case '/': System.out.println(num1/ num2);
                  break;
        default: System.out.println("Wrong operator");
}
```

في هذا المثال، يتم اختبار قيمة المتغير op، ويتم تنفيذ العملية المناسبة حسب هذه القيمة. ثم الخروج من عبارة **switch** لوجود الكلمة المحجوزة **break**. أما إذا لم يدخل المستخدم إحدى العمليات الأربع، يطبع البرنامج رسالة خطأ.

لكي نفهم أكثر أهمية وجود الكلمة **break**، لنقم بحذفها من المثال السابق ومتابعة التنفيذ.

```
switch(op)
{
        case '+': System.out.println(num1+ num2);
        case '-': System.out.println(num1- num2);
        case '*': System.out.println(num1* num2);
        case '/':  System.out.println(num1/ num2);
        default:  System.out.println("Wrong operator");
}
```

هنا وبعد اختبار الجزئية المناسبة للتنفيذ وطباعة العبارة، لا توجد وسيلة للخروج من عبارة **switch**، وعليه تتواصل عمليات الطباعة إلى نهاية عبارة **switch**، مثلاً: إذا أدخل المستخدم عملية الطرح، لن يتم طباعة حاصل الجمع بالتأكيد، سيتم طباعة حاصل الطرح، ولكنه هنا يستمر يطبع حاصل الضرب والقسمة وكذلك رسالة الخطأ.

مثال للتنفيذ:

```
Enter two numbers
12
3
Enter on operation[+,-,*,/]
-
9
36
4
Wrong operator
```

وهي نتيجة غير مرغوبة تماماً في هذا المثال. وبالمثل إذا تم إدخال عملية الضرب، يتم طباعة حاصل الضرب والقسمة ورسالة الخطأ. وفي حال إدخال عملية القسمة، سيطبع حاصل القسمة ورسالة الخطأ.

مثال:

اكتب برنامجاً يستقبل عدداً صحيحاً من 1 إلى 5، يقوم البرنامج بطباعة الأعداد تنازلياً من العدد المدخل وحتى 1.

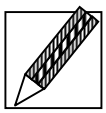
الحل:

```
System.out.println("Enter a number 1..5");
int number= Input.readInt( );

switch (number)
{
        case 5: System.out.println (number--);
```

```
        case 4: System.out.println (number--);
        case 3: System.out.println(number--);
        case 2: System.out.println(number--);
        case 1: System.out.println(number--);
                break;
      default: System.out.println("The number must be in the range 1..5");
}
```

عند إدخال القيمة 4 مثلاً، يقوم بطباعة العدد 4، ثم إنقاص قيمة المتغير إلى العدد 3. وبسبب عدم وجود عبارة **break**، يواصل عمليات الطباعة، فيطبع 3 ثم تنقص قيمة المتغير إلى 2. وهكذا.

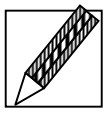
تدريب (6)



1- اكتب برنامجاً يقرأ الدرجة التي أحرزها الطالب (A,B,C,D أو F)، ويطبع التقدير المقابل حيث:

A	Excellent
B	Very Good
C	Good
D	Pass
F	Fail

تدريب (7)



اكتب برنامجاً يقرأ حرفاً، إذا كان هذا الحرف ضمن الحروف الإنجليزية المتحركة، يطبع Vowel Letter. وإذا كان ضمن الحروف Z, Y, X يطبع Math Variable.

أسئلة تقويم ذاتي



1- هناك طريقة أخرى توفرها **جافا** تسهّل وتختصر عمليات الاختبار على المتغير ما هي؟ وما هي صيغتها ؟ .

2- ما هي المهمة التي تقوم بها الكلمة **break**; ؟

2- وضّح كيف تم تلافي مشكلة عدم أخذ البرنامج في الاعتبار إدخال الحروف الصغيرة.

7. العبارات التكرارية

عزيزي الدارس

في كثير من البرامج، نحتاج لتكرار تنفيذ جزئية معينة من البرنامج لعدد من المرات، مثلاً إذا كان البرنامج يقوم بقراءة أسماء 50 موظفاً، ليس من المنطقي أن نكتب 50 عبارة قراءة مختلفة. أو إذا كان البرنامج يطبع الأعداد من 1 إلى 1000، فلا يمكن تصور برنامج يحتوي على 1000 عبارة طباعة، لأنه سيكون طويلاً جداً، وفي نفس الوقت يحتوي على مجموعة من العمليات المتشابهة، وهي عملية الطباعة.

من المتوقع أن نحتاج إلى تكرار تنفيذ العبارات في أغلب البرامج، وخاصة البرامج الكبيرة والأنظمة لأنها تتعامل مع مجموعات من البيانات. ففي نظام للمرتبات، يتم حساب المرتب لكل موظف على حده. أي تكرار عملية حساب المرتب بعدد الموظفين. وفي نظام بنكي، للبحث عن اسم عميل بواسطة رقم حسابه، يتم المرور على جميع عملاء البنك واختبار أرقام الحساب إلى أن نجده أو ينتهي العملاء. ولذلك نجد أن للتكرار أهمية كبرى يكاد لا يستغني عنها أي نظام.

توفر لغة **Java** ثلاث عبارات مختلفة للتكرار. سنتناولها بالتفصيل.

1.7 العبارة التكرارية for

وغالباً تأخذ الصيغة:

```
for(counter=initialValue; counter<=finalValue; incrementExpression)
        Statement;
```

عبارة أو حلقة **for** تقوم بتكرار تنفيذ التعليمة statement لعدد معلوم من المرات. هذا العدد المعلوم عبارة عن عدد القيم التي يأخذها عداد الحلقة counter. يأخذ العداد القيمة الابتدائية initialValue ويتم تنفيذ العبارة statement، وبعد كل تنفيذ تزداد قيمة المتغير counter حسب ما هو معرّف في incrementExpression حتى يصل إلى القيمة النهائية finalValue، وعندها يتوقف التكرار.

مثلاً:

```
 for(int i=1; i<=5; i++)
        System.out.println("My first for loop");
```

في هذه الحلقة العداد هو i، ويأخذ القيمة الابتدائية 1. ومن ++i نعلم أنه يزداد بمعدل 1 كل تكرار. أما القيمة النهائية التي يأخذها المتغير i فهي 5. أي أن الحلقة تكرر طباعة الجملة 5 مرات لأن المتغير i يأخذ القيم 1، 2، 3، 4، 5.

مثال آخر:

```
for(int counter=0; counter<10; counter ++)
        System.out.print(counter);
```

تبدأ قيمة العداد counter من الصفر، حيث يتم تنفيذ عبارة الطباعة للمرة الأولى، فيطبع البرنامج 0 على الشاشة. ثم تزداد قيمة counter بواحد فتتنفذ عبارة الطباعة وتطبع 1. وتستمر عمليات الطباعة وزيادة قيمة counter بواحد، ومع كل زيادة تتم عملية المقارنة للتأكد من أن قيمة counter لا تزال counter<10. لأن الحلقة ستتوقف إذا لم يكن الأمر كذلك. عندما تصل قيمة counter إلى 9 تتم طباعة الرقم 9 على الشاشة، ثم زيادة قيمة counter بواحد لتكون قيمته 10، وعند إجراء المقارنة 10>9 تكون غير صحيحة ويتوقف التكرار. لتكون مخرجات هذه الحلقة هي:

0 1 2 3 4 5 6 7 8 9

في المثال الأول قمنا بتكرار عملية طباعة لعدد معلوم من المرات، وفي المثـــال الثاني قمنا بكتابة برنامج يطبع الأعداد من 0 إلى 9، لذلك صممنا حلقة **for** تعمـــل 10 مرات (من0 إلى9) واستفدنا من عداد الحلقة، حيث إنه سيأخذ نفس القيم المراد طباعتها.
يمكن حساب عدد مرات تكرار عمل الحلقة **for** من القانون الآتي:

$$\text{عدد التكرارات} = \left\lfloor 1 + \left(\frac{\text{القيمة النهائية} - \text{القيمة الابتدائية}}{\text{معدل زيادة العداد لكل تكرار}} \right) \right\rfloor$$

علامة $\lfloor \ \rfloor$ تعني أنه إذا كانت القيمة النهائية لعدد التكرارات عبارة عن كســـر، يـــتم حذف الكسر وتقريبه إلى العدد الصحيح الأصغر.

مثلاً:

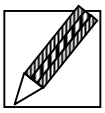
لمعرفة تكرار الحلقة الآتية:

```
for(int i=1; i<=10; i++)
Iterations= 10-1 +1 =10
              1
for(int k=6; k<20; K++)
Iterations= 19-6 +1 =14
              1
```

لأحظ أن القيمة النهائية لهذه الحلقة هي 19 وليس 20.

```
Iterations= 30-0 +1 =11
              3
```

تدريب (8)



مستعيناً بالقانون أوجد تكرار الحلقة الآتية:

```
for(int counter=0; counter<=30; counter+=3)
```

مثال:

المطلوب كتابة حلقة تقوم بطباعة الأعداد 2، 4، 6، 8، ... إلى 20.

الحل:

عدد الأعداد المطلوب طباعتها 10، فيمكن عمل حلقة تكرر 10 مرات يأخذ عدادها القيم 1، 2، 3، 4، ...، 10. تكون القيم المطلوب طباعتها هي 2، 4، 6، ...، 20 أي 2*1، 2*2، 2*3، ...، 2*10 (أو 2*قيمة العداد) لتكون الحلقة هي:

```
for(int k=1; k<=10; k++)
        System.out.print(2*k);
```

حل آخر:

يمكن تصميم الحلقة بحيث يأخذ عداد الحلقة 2 كقيمة ابتدائية ويكون معدل زيادته كل تكرار بمعدل 2 (k+=2) ليأخذ العداد القيم 2، 4، 6، 8، ... ونضع له قيمة نهائية 20، لتكون الحلقة هي:

```
for(int k=2; k<=20; k+=2)
        System.out.print(k);
```

مثال:

اكتب حلقة for لطباعة جدول 5

الحل:

يحتوي جدول 5 على 12عملية

```
5*1=5
5*2=10
5*3=15
..      ..
..      ..
5*12=60
```

العدد 5 ثابت في بداية كل عملية، وكذلك علامة الضرب.. العدد التالي متغير يأخذ القيم 1، 2، 3، ...، 12 بالتتالي، علامة التساوي ثابتة. قيمة عملية الضرب تنتج عن ضرب 5 في العدد المتغير.

سنقوم بتصميم الحلقة بحيث تعمل 12 مرة، وسيأخذ عدادها القيم 1، 2، 3، ...، 12. وبالتالي يمكن الاستفادة من قيمته لأنها نفس قيم العدد المتغير في جدول الضـــرب. ناتج عملية الضرب هو 5*1، 5*2، 5*3، ...، 5*12. وهي 5 في قيمة عداد الحلقة.

```
for(int i=1; i<=12; i++)
        System.out.println("5*" + i + "= " + (5*i));
```

في التكرار الأول تكون قيمة i تساوي 1، لذلك تكون المخرجات:

5*1=5

في التكرار الثاني تكون قيمة i تساوي 2، لذلك تكون المخرجات:

5*2=10

وهكذا حتى نصل آخر تكرار، حيث تكون قيمة i هي 12 لتكون المخرجات هي:

5*12=60

مثال:

اكتب برنامجاً لحساب مجموع الأعداد من 1 إلى 50.

الحل:

لدينا 50 عملية جمع، يمكن إجراؤها داخل حلقة **for**. الأعداد المطلوب جمعهـــا هي: 1، 2، 3، ...، 50. ولذلك يمكن جعل عداد حلقة **for** يأخذ هذه القيم. سنحتاج إلـــى متغير (وليكن sum) للاحتفاظ بالمجموع بداخله.

```
int sum=0; //                    وضع القمة الابتدائية هنا مهم
for(int counter=1; counter <=50; counter ++)
        sum=sum+ counter;    // sum+ = counter; أو
```

بعد أن تكمل الحلقة تكرارها، يكون المتغير sum يحتوي على مجموع الأعداد من 1 إلى 50، بعد ذلك يمكن طباعة قيمته:

```
System.out.println(sum);
```

مثال:

اكتب برنامجاً يقرأ 20 عدداً صحيحاً، ويحدد عدد الأعداد السالبة ضمن هذه الأعداد.

الحل:

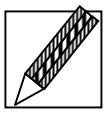
العمليات المطلوب تكرارها 20 مرة هي:

1. قراءة عدد صحيح.
2. اختبار قيمته: إذا كانت سالبة تزيد قيمة عداد معين بواحد، وعليه نحتاج لتعريف متغير نقرأ فيه قيمة العدد الصحيح لاختبارها، ومتغير آخر يعمل كعداد يحسب عدد الأعداد السالبة.

```
int number, negatives=0; //    العداد
for(int i=1; i<=20; i++)
{
        number=Input.readInt( );
        if (number<0) //                إذا كان العدد سالب
                negatives++; //          يزيد العداد بواحد
 }
System.out.println("There are" + negatives + " negative numbers");
```

عندما نريد تكرار أكثر من تعليمة واحدة بواسطة حلقة **for**، نضع هذه التعليمات بين قوسي بداية نهاية. في هذا المثال إذا لم نضع قوسي البداية والنهاية، كانت ستكون عملية التكرار خاصة بالتعليمة الأولى فقط، وهي القراءة. أي أن البرنامج كان سيقوم بعشرين عملية قراءة. ثم بعد ذلك يختبر قيمة العدد الأخير (رقم 20) إذا كان موجباً يكون عدد الأعداد السالبة يساوي 0، وإذا كان سالباً يكون عددها 1، وهذا خطأ شائع.

تدريب (9)



اكتب برنامجاً يقرأ 10 أعداد، ويطبع عدد الأعداد الفردية والزوجية فيها.

تدريب (10)



المطلوب كتابة برنامج لحساب مضروب 4 (factorial 4).

تدريب (11)



المطلوب كتابة حلقة تقوم بحساب 6^4.

مثال:

اكتب حلقة لطباعة الأعداد التي تقبل القسمة على 7 بين 1 و 100.

الحل:

نصمم حلقة تعمل من 1 إلى 100. وفي كل تكرار نختبر قيمة العداد، إذا كان يقبل القسمة على 7 نقوم بطباعته.

```
System.out.println("The numbers are:  ");
for (int i=1; i<=100; i++)
{
        if (i%7 ==0)// 7 يقبل القسمة على
        System.out.print( i + " ");
}
```

مثال:

المطلوب طباعة نفس الأعداد السابقة تنازلياً.

الحل:

نجعل عداد الحلقة يبدأ من 100 نزولاً إلى 1. أي أنه في كل تكرار تقل قيمة العداد بواحد (i--).

```
for (int i=100; i>=1; i--)
```

باقي الحل كالمثال السابق.

نلاحظ أن عداد الحلقة يأخذ القيمة الابتدائية 100 والقيمة النهائية 1، لذلك سيتم طباعة الأعداد بدايةً بالأكبر (98) لتكون آخر قيمة يتم طباعتها هي 7.

مثال:

المطلوب كتابة حلقة تقوم بقراءة 10 أعداد ثم طباعة متوسطهم.

الحل:

نصمم حلقة تكرر عمليتي قراءة العدد وحساب المجموع10 مرات. بعد إكمال حساب مجموع الأعداد نحسب المتوسط.

```
int sum=0, number;
for(int i=1; i<=10; i++)
{
        System.out.println("Enter number " + i + ":");
        number=Input.readInt( );
        sum += number; // حساب المجموع
}
double avg = sum/10.0; // حساب المتوسط
System.out.println("The average is: " + avg);
```

نلاحظ أننا قمنا بتعريف المتغير number خارج الحلقة، واستخدمنا نفس المتغير لقراءة الأعداد العشرة. من الخطأ تعريف المتغير داخل الحلقة، لأن هذا يعني تعريف متغير بنفس الاسم لعدد من المرات، وهو سيئ برمجياً.

العبارة "Enter number " + i + ":"

تظهر في الشاشة على الشكل:

Enter number 1:

Enter number 2: وهكذا

1.1.7 حلقات for المتداخلة

لنفرض أننا نريد كتابة حلقة تقوم بطباعة الأعداد من 1 إلى 10 في سطر واحد على الشاشة.

```
for (int i=1; i<=10; i++)
```

```
        System.out.print(i);
```

لتكون المخرجات هي:

```
12345678910
```

لنفرض أننا نريد تكرار هذه العملية ثلاث مرات. سنقوم بكتابة حلقة تكرر ثلاث مرات وتقوم بالعملية السابقة في كل دورة من دوراتها. إذن محتويات هذه الحلقة هو أيضاً حلقة. وهذا هو مفهوم الحلقات المتداخلة.

```
for (int j=1; j<=3; j++)
        for (int i=1; i<=10; i++)
                System.out.print(i);
```

لتكون المخرجات هي:

```
12345678910123456789101234567891O
```

أي أن ما تقوم الحلقة بتكراره هو عبارة عن حلقة أيضاً. يمكن أن نقوم بتعديل الحلقة السابقة لتكون المخرجات:

```
12345678910
12345678910
12345678910
```

أي أننا سنصمم حلقة تعمل ثلاث مراتٍ، في كل مرة يتم طباعة الأعداد والانتقال إلى السطر الجديد. عملية طباعة الأعداد هي نفسها حلقة، ولذلك نجد حلقتين متداخلتين.

```
1.      for (int j=1; j<=3; j++)
2.      {
3.              for (int i=1; i<=10; i++)
4.                      System.out.print(i);
5.              System.out.print("\n");
6.      }
```

حيث \n هو حرف يؤدي إلى نزول المؤشر إلى السطر الجديد على الشاشة. نلاحظ أننا نستخدم عدادات للحلقات المتداخلة بأسماء مختلفة، حيث j هو عداد الحلقة الخارجية، و i هو عداد الحلقة الداخلية.

في التكرار الأول للحلقة الخارجية، أي عندما يكون j=1، تقوم الحلقة الداخلية بتنفيذ تكراراتها كاملة (10 مرات)، ثم تنفيذ السطر 5 مرة واحدة. عندئذ يكون التكرار

الأول للحلقة الخارجية قد اكتمل. وعندها تزداد قيمة j إلى 2، فتؤدي الحلقة الداخلية تكراراتها جميعاً ثم الانتقال إلى السطر التالي بواسطة n\. بعدها يبدأ التكرار الثالث والأخير للحلقة الخارجية، ومن جديد تقوم الحلقة الداخلية بطباعة عشرة أعداد، يتبعها تنفيذ السطر 5 وينتهي التنفيذ. تقوم الحلقة الداخلية بعشرة تكرارات، وتؤديها ثلاث مرات، ليكون عدد التكرارات الكلي للحلقة الداخلية هو 30 تكرار.

وعموماً:

عدد تكرارات الحلقة الداخلية = عدد تكرارت الحلقة الداخلية * عدد تكرارت الحلقة الخارجية

مثال:

اكتب عبارات لقراءة وحساب مضروب ثلاثة أعداد وطباعته.

الحل:

نعلم سلفاً العبارات اللازمة لقراءة وحساب مضروب عدد ثم طباعته، وهي:

```
System.out.println("Enter a number");
int number=Input.readInt( );
int fact = 1;
for(int i=2; i<= number; i++)
        fact *=i;
System.out.println("Factorial is: " + fact);
```

نقوم بوضع جميع الخطوات السابقة داخل حلقة **for** تقوم بثلاثة تكرارات. وهناك تعديلات يجب أن نلاحظها، فلا يجوز أن نقوم بتعريف متغيرات داخل حلقة، ولذلك سيتم تعريف المتغيرين number و fact أولاً خارج الحلقة الخارجية.

```
int number, fact;
for(int k=1; k<=3; k++)
{
        System.out.println("Enter a number");
        number=Input.readInt( );
        fact=1;
        for(int i=2; i<= number; i++)
                fact *=i;
        System.out.println("factorial is: " + fact);
```

}

من الأخطاء الشائعة وضع fact=1 خارج الحلقة الخارجية، وعندها عندما يـتم حساب مضروب العدد الثاني يكون المتغير fact محتفظاً بقيمة مضـروب العـدد الأول، وبالتالي يحسب مضروب العددين الثاني والثالث خطأ.

مثال:

اكتب برنامجاً يقرأ أسماء ثلاثة طلاب ودرجات كل منهم في أربع مواد، يقـوم البرنامج بحساب متوسط كل طالب وطباعة اسمه ومتوسط درجاته.

الحل:

العبارات اللازمة لقراءة درجات طالب واحد في أربع مواد وطباعة متوسط درجاته هي:

```
System.out.println("Enter student Name");
String name=Input.readString( );
int sum=0, degree;
for(int i=1; i<=4; i++)
{
        System.out.println("Enter degree for subject number" + i);
        degree= Input.readInt( );
        sum+= degree;
}
double avg = sum/4.0;
System.out.println(name + " average is " + avg);
```

مثال للتنفيذ:

```
Enter student name
Fatima Hassan
Enter degree for subject number 1
64
Enter degree for subject number 2
80
Enter degree for subject number 3
69
```

```
Enter degree for subject number 4
85
Fatima Hassan average is 74.5
```

نستطيع الآن تعديل البرنامج ليتعامل مع بيانات ثلاثة طلاب مع مراعاة أن يـتم تعريف كل من المتغيرات sum, name, degree, avg خارج الحلقة الخارجيـة، ليكـون البرنامج كالآتي:

```
String name;
int sum, degree;
double avg;
for (int i=1; i<=3; i++)
{
        System.out.println("Enter name of student number " + i);
        name= Input.readString( );
        sum =0;
        for (int k=1; k<=4; k++)
        {
                System.out.println("Enter degree of subject number " + k);
                degree = Input.readInt( );
                sum+= degree;
        }
        avg=sum/4.0;
        System.out.println(name + " average is " + avg);
}
```

لاحظ وجود العبارة sum=0 مباشرةً قبل بداية الحلقة الداخلية للتخلص من درجات الطالب السابق.

مثال:

اكتب برنامجاً لطباعة الشكل:

```
2       3       4
3       4       5
4       5       6
5       6       7
```

الحل:

لدينا جدول يحتوي على أربعة أسطر، مما يوحي لنا بحلقة تعمل أربع مـــرات. في السطر الواحد ثلاث عمليات طباعة، أو حلقة تعمل ثلاث مرات.

لطباعة الأشكال والجداول، نحاول دائماً إيجاد العلاقة بين كل عدد والعدد الـــذي يليه في الصف، والعلاقة بين الأعداد في بداية كل صف، والعلاقة بينهـــا وبـــين رقـــم الصف الموجودة فيه.

في هذا الشكل إذا قمنا بترقيم كل صف وعمود في الجدول، ســـنلاحظ أن كـــل عنصر من عناصر الجدول عبارة عن حاصل جمع رقم الصف والعمود الذي يقع عليـــه ذلك العنصر.

مثلاً:

العدد 5 يظهر في الصف الثالث والعمود الثاني 5=2+3. ويظهر مرة أخرى في الصف الثالث والعمود الثاني، وأيضا 5=3+2. وهكذا.. فإذا كان عداد الحلقة الخارجيـــة (حلقة الصفوف) هو i، وعداد الحلقة الداخلية (حلقة الأعمدة) هو j، فإن كل عنصر فـــي الجدول هو عبارة عن i+j. عليه يمكن كتابة الحلقات على الصورة:

```
for (int i=1; i<=4; i++)                    //   عدد الأسطر
{
        for (int j=1; j<=3; j++)
                System.out.print(i+j);      //  المسافة هنا يمكن إضافة مسافة
        System.out.print("\n");
}
```

مثال:

اتب عبارات لطباعة الشكل:

```
    *
  *   *
*   *   *
```

الحل:

هنا لدينا حلقة تعمل ثلاث مرات لإجراء عمليات معينة على ثلاثة أسطر. ما يتم طباعته هنا ثابت (*) إلا أنه لا يتم تكراره لعدد ثابت من المرات في كل سطر. لكن نلاحظ أن عدد النجوم المطبوعة على السطر = رقم السطر نفسه، وهو الرقم الذي يحمله عداد الحلقة. عليه يمكن عمل حلقة داخلية تعمل لعدد من المرات حسب رقم الدورة للحلقة الخارجية. أي في الدورة الأولى للحلقة الخارجية تعمل الحلقة الداخلية مرة واحدة وفي الدورة الثانية لها تعمل مرتين، وفي الثالثة تعمل ثلاث مرات. يكون ذلك بأن نجعل الحلقة الداخلية تعمل ابتداءً من 1 وحتى قيمة عداد الحلقة الخارجية، ليكون البرنامج:

```
for (int i=1; i<=3; i++)
{
        for (int j=1;j<= i; j++)
                System.out.print("* ");
        System.out.print("\n");
}
```

مخرجات هذه العبارات هو:

```
*
*       *
*       *       *
```

وهو يختلف قليلاً عن الشكل السابق، حيث تنقصه المسافات في بداية كل سطر.

رقم السطر	عدد المسافات
1	2 (3 – 1)
2	1 (3 – 2)
3	0 (3 – 3)

أي أن عدد المسافات في كل سطر يساوي 3 ناقصاً رقم ذلك السطر. يمكن أن نصمم حلقة داخلية جديدة لطباعة المسافات تبدأ التكرار من 1 وحتى (3 – i) حيث i هو عداد الحلقة الخارجية لأنه يعبر عن رقم السطر. ليكون البرنامج النهائي هو:

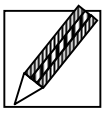
```
for(int i=1; i<=3; i++)
{
        for(int k=1; k<= (3 – i); k++)      // طباعة المسافات
                System.out.print(" ");
        for(int j=1; j<=  i; j++)           // طباعة النجوم
                System.out.print("* ");
        System.out.print("\n");
}
```

لطباعة المسافات، يمكن أيضا استخدام حلقة تنازلية كالآتي:

```
for(int k=3; k>i; k--)
```

نلاحظ أن حلقة طباعة المسافات لا تؤدي أي تكرار في السطر الثالث، وهو المطلوب في هذا المثال لأن السطر الثالث لا يحتوي على مسافات.

تدريب (12)



اكتب برنامجاً لطباعة الشكل:

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

مثال:

المطلوب كتابة برنامج لإيجاد مجموع المتوالية:

$X + X^2/2! + X^3/3! + X^4/4! + \ldots + X^9/9!$

الحل:

المتوالية عبارة عن تسعة حدود: كل حد يتكون من بسط وهو عبارة عن أس. ومقام وهو عبارة عن مضروب. في مثالين سابقين عرفنا كيف نحصل على قيمة عدد مرفوع لقوة معينة، وكيف نحصل على مضروب عدد، واستخدمنا لذلك حلقة **for**. للحصول على حد من حدود المتوالية، نقسم قيمة الأس على قيمة المضروب. عدد حدود

المتوالية هو 9. الأس والمقام عبارة عن ترتيب الحد في المتوالية، مثلاً الحد السادس هو $X^6/6!$ نستخدم حلقة تعمل تسع مرات. في كل مرة تحسب قيمة الحد وتضيفه إلى مجموع المتوالية. نستفيد من عداد هذه الحلقة لتحديد عدد تكرارات حلقتي الأس والمضروب.

```
long power, fact;
double term, sum;
System.out.println("Enter x");
int x=Input.readInt( );
sum=x;
for(int i=2; i<=9; i++)
{
        power=1;
        for(int j=1; j<=i; j++);
                power *=x;      //    x^i
        fact=1;
        for(int k=2; k<= i; k++)
                fact *=k;                       //     مضروب i
        term= (double) power/fact;    //        حساب الحد
        sum += term;                  //        مجموع المتوالية
}
System.out.println("The summation is " + sum);
```

نلاحظ أن الحلقة الخارجية تبدأ من 2، ذلك لأن التكرار الأول لا معنى له، حيث إن قيمة الحد الأول معروفة وهي x، لذلك قمنا بوضعها مباشرة كقيمة ابتدائية في المتغير الذي يحمل مجموع المتوالية sum.

تم تعريف المتغيرين power و fact من النوع **long**، وذلك لأن المتغير من النوع **int** لا يتسع لقيم كبيرة مثل!9 أو 6^9. وتم تحويل القيمة power/fact إلى **double** أولاً قبل تخزينها في المتغير term، وذلك لكي لا نفقد الجزء الكسري من ناتج القسمة، والذي نفقده لأن ناتج قسمة عددين صحيحين تعطي عدداً صحيحاً.

أسئلة تقويم ذاتي



أكمل التالي والذي يمثل قانون حساب عدد مرات تكرار عمل الحلقة **for** :

عدد التكرارات = ⌊ (القيمة النهائية - .....؟ / .....؟) + 1 ⌋

ثم وضح ما القصود بعلامة ⌊ ⌋؟ .

2.7 العبارة التكرارية while

تأخذ الصيغة التالية:

```
while(condition)
{
        statement(s);
}
```

تقوم الحلقة **while** بتكرار العبارات بداخلها مادامت قيمة الشرط condition هي true، مثلاً:

```
int x=1;
while (x<10)
{
        System.out.print(x);
        x++;
}
```

عند محاولة الدخول لحلقة **while**، يتم اختبار قيمة الشرط أولاً. وبما أن قيمة x أقل من 10، قيمة شرط حلقة **while** يكون **true**، وبالتالي يتم الدخول إلى جسم الحلقة **while**، حيث يتم تنفيذ العبارات المراد تكرارها، ونلاحظ وجود عبارة x++ من بينها، وهي العبارة التي تقوم بتغيير قيمة المتغير الذي له علاقة بشرط تنفيذ الحلقة. ومن

الضروري وجود عبارة تغير قيمة المتغير الموجود بالشرط لكي تتغير قيمته بعد تكـــرارٍ ما إلى **false**، لتنتهي تكرارات الحلقة **while**. ففي هـــذا المثـــال نجـــد أن حلقـــة **while** تواصل تكراراتها مادامت x<10، وبداخل الحلقة تقوم بزيادة قيمة المتغير x، حتى يصل بعد عدد من التكرارات إلى القيمة 10، عندها وعند اختبار شرط عمل الحلقة تكون قيمة x<10 هي **false** لينتهي عمل الحلقة، وتكون المخرجات:

123456789

في حلقة **while** نلاحظ الآتي:

1. وجود شرط يتم اختبار قيمته قبل بداية تنفيذ الحلقة.
2. لمتغير الشرط قيمة ابتدائية قبل الوصول لبداية الحلقة.
3. وجود عبارة لتغيير قيمة متغير الشرط داخل جسم الحلقة.
4. قد لا يتم تنفيذ الحلقة نهائياً، وذلك إذا كانت قيمة متغير الشرط الابتدائية تجعـــل قيمة الشرط **false** ولا يتم الدخول إلى جسم الحلقة. مثلاً إذا كانت قيمة متغيـــر الشـــرط الابتدائية في المثال السابق x=20، لن يتم الدخول إلى الحلقة أصلاً.

مثال:

اكتب برنامجاً يقرأ مجموعة من الأعداد ويطبـــع مربعاتهـــا. ينتهـــي البرنـــامج بإدخال صفر.

الحل:

```
int num;
System.out.println("Enter a number");
num=Input.readInt( );
while (num != 0)
{

        System.out.println(num * num);
        System.out.println("Enter a number");
        num=Input.readInt( );
}
```

يمكن لهذه الحلقة ألا تعمل أبداً، وذلك إذا تم إدخال قيمة المتغير num صفر منذ البداية. نلاحظ أن أول قيمة للمتغير num والتي قرأناها خارج الحلقة يجب أن نطبع مربعها أولاً قبل أن نطلب إدخال قيمة جديدة، وأن القيمة الجديدة يجب اختبارها أولاً في شرط الحلقة قبل طباعة مربعها، لذلك هذا هو الترتيب الأمثل للعبارات داخل الحلقة.

مثال:

اكتب برنامجاً يسأل المستخدم ويطلب إليه الإجابة y (yes) أو n (no). يكرر البرنامج الطلب حتى يجيب المستخدم بإحدى الإجابتين ثم يطبع إجابة المستخدم ويعلق عليها حسب الإجابة.

الحل:

```
System.out.println("Do you like Java?(y/n)");
char answer=Input.readChar( );
while((answer!='y')&&(answer!='n'))
{
        System.out.println("Do you like Java?(y/n)");
        answer=Input.readChar( );
}
```

لا تتوقف هذه الحلقة حتى يدخل المستخدم y أو n. يمكن تعديل شرط الحلقة بحيث تشمل الحروف الكبيرة Y و N. عند الخروج من الحلقة، تكون قيمة المتغير answer إما y أو n، لذلك لابد من فحص محتواه للتعليق، لأن لكل قيمة منهما تعليقها الخاص.

```
if (answer =='y')
        System.out.println("I know, Java is the best");
else        //    answer =='n'
        System.out.println("I hope you have good reasons");
```

مثال:

اكتب برنامجاً يقرأ عدداً من المستخدم، ويقوم بطباعة الأعداد تنازلياً من العدد المدخل إلى 1.

الحل:

```
System.out.println("Enter a number");
int number=Input.readInt( );
while(number>=1)
        System.out.println(number--);
```

ماذا يحدث عند إدخال عدد سالب؟ وكيف يمكن أن نحل هذه المشكلة؟

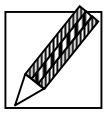
أسئلة تقويم ذاتي



1- ما الصيغة للعبارة التكرارية while ؟.

2- ما الملاحظات في حلقة while ؟

تدريب (13)



اكتب برنامجاً يقرأ عدداً صحيحاً ويحدد عدد خانات العدد المدخل .

3.7 العبارة التكرارية do - while

وتأخذ الصيغة:

```
do
{
        statement(s);
}while (condition);
```

وهي شبيهة بحلقة **while** إلا أنه يتم اختبار شرطها في نهاية الحلقة. أي أنها تقوم بتنفيذ العبارات الموجودة بداخلها ثم اختبار قيمة الشرط لتحديد استمرارية تكرار عباراتها أو توقفها، ونلاحظ في حلقة **do-while** الآتي:

1. وجود شرط يتم اختبار قيمته بعد نهاية تنفيذ الحلقة.
2. لا يشترط وجود قيمة ابتدائية لمتغير الشرط.

3. وجود عبارة لتغيير قيمة متغير الشرط داخل جسم الحلقة.

4. يتم تنفيذ الحلقة لمرة واحدة على الأقل، وذلك لأن اختبار الشرط يتم في نهاية الحلقة، وبالتالي حتى إذا كان شرط تكرار الحلقة غير متحقق، لن يتم اكتشاف ذلك إلا بعد انتهاء التكرار الأول.

مثال:

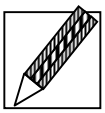
اكتب برنامجاً يطبع مضاعفات العدد 12 بين 1 و 100.

الحل:

```
int number=13;
do
{
        System.out.println(number);
        number+=13;
}while(number<100);
```

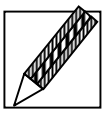
تقوم هذه الحلقة بطباعة قيمة المتغير number، وزيادة قيمته بمقدار 13، وتستمر الحلقة في تكرار هذه العبارات حتى تخرج قيمة المتغير number عن المدى المحدد للطباعة، وهو 100، لذلك كان شرط عمل هذه الحلقة أن تكون قيمة المتغير number أقل من 100.

تدريب (14)



اكتب برنامجاً يقرأ مجموعةً من الأعداد الصحيحة، ويطبع العدد الأكبر بين الأعداد. تنتهي الأعداد بإدخال الصفر.

تدريب (15)



اكتب برنامجاً يطلب من المستخدم تخمين عدد من 1 إلى 10، وعلى المستخدم أن يواصل التخمين حتى يتوصل إلى القيمة المخزنة بالبرنامج. يطبع البرنامج عدد المحاولات التي قام بها المستخدم إلى أن خمّن العدد الصحيح.

مثال:

اكتب برنامجاً يقرأ ثلاث قيم تمثل أطوال أضلاع، يحسب البرنامج محيط المثلث ويطبعه، ويكرر هذه العمليات مادامت جميع القيم المدخلة موجبة.

الحل:

```
int edge1, edge2, edge3, perimeter;
do
{
        System.out.println("Enter length of  3 edges");
        edge1= Input.readInt(  );
        edge2= Input.readInt (  );
        edge3= Input.readInt (  );
        perimeter= edge1+ edge2+ edge3;
        System.out.println("Perimeter of triangle of triangle is: " + perimeter);
}while((edge1>0)&&(edge2>0)&&(edge3>0));
```

نلاحظ وجود عيب في هذا البرنامج، فمن المفروض أن ينتهي البرنامج عند إدخال عدد سالب أو صفر كقيمة لأي ضلع من أضلاع المثلث. لكن في هذا البرنامج أعلاه عند إدخال قيمة سالبة رغبة في إنهاء البرنامج فإنه يحسب محيط المثلث ويطبعه أولاً ثم ينتهي البرنامج. وهذا غير طبيعي.

نقوم الآن بتعديل البرنامج بإعادة ترتيب عباراته لتكون قراءة أضلاع المثلث في نهاية الحلقة، لكي ينتهي التكرار إذا شملت القيم المدخلة عدداً سالباً. وبذلك تكون أول عبارة في الحلقة هي عملية حساب وطباعة المحيط. وعليه نحتاج لقراءة قيم أضلاع المثلث قبل دخول الحلقة، ليصبح البرنامج كالآتي:

```
int edge1, edge2, edge3, perimeter;
System.out.println("Enter length of  3 edges");
edge1= Input.readInt(  );
edge2= Input.readInt(  );
edge3= Input.readInt(  );
```

```
do
{
        perimeter= edge1+ edge2+ edge3;
        System.out.println("Perimeter of triangle is: " + perimeter);
        edge1= Input.readInt( );
        edge2= Input.readInt( );
        edge3= Input.readInt( );
}while((edge1>0)&&(edge2>0)&&(edge3>0));
```

إذا تم إدخال قيم موجبة لأطوال المثلث في البداية، يستمر عمل البرنامج بصورة صحيحة. وعند إدخال قيمة سالبة أو صفر كأحد أضلاع المثلث، ينتهي عمل الحلقة مباشرة دون حساب أو طباعة محيط المثلث، وهذا أفضل من البرنامج السابق.

لكن هناك عيب آخر لا يمكن تفاديه باستخدام حلقة **do – while** إذا تم إدخال قيمة سالبة أو صفر كقيمة لأحد أضلاع المثلث منذ البداية، من المنطقي ألا يقوم البرنامج بأية عمليات حسابية للمحيط، ولكن هذا البرنامج (في الصيغة الأولى) يقوم في هذه الحالة بحساب المحيط وطباعته ثم بعد ذلك ينتهي البرنامج. هذا العيب سببه أن حلقة **do – while** تنفذ تكراراً واحداً على الأقل، وهو ما يميزها عن حلقة **while**. وإذا أردنا رغم ذلك تصحيح البرنامج باستخدام حلقة **do – while**، سنحتاج إلى عدة عمليات اختبار على قيم المدخلات، وهو تعقيد غير مبرر يمكن تفاديه باستخدام حلقة **while**، عليه يجب الانتباه عند اختيار الحلقة المناسبة للبرنامج المحدد، فإذا كان من الممكن ألا يتم تنفيذ تكرارات الحلقة مطلقاً كما في هذا المثال، وجب استخدام حلقة **while**. حل المثال السابق باستخدام حلقة **while** شبيه بالصيغة الثانية. وهو الحل الأمثل للبرنامج. إذا تم إدخال قيمة سالبة منذ بداية التنفيذ، يتم اختبار شرط حلقة **while** وتكون قيمته **false** وينتهي البرنامج بدون حساب المحيط.

تسمى حلقة **while** و **do – while** الحلقات ذات الشرط لأن عملها وتوقفها يعتمد على تحقق شرط.

4.7 ملاحظات حول العبارات التكرارية الثلاث

- عندما نعلم سلفاً عدد التكرارات التي ستنفذها الحلقة، الأفضل استخدام حلقة **for**.
- إذا كنا لا نعلم عدد التكرارات تحديداً، وخصوصاً إذا كان التكرار يعتمد على قيمة يقوم بإدخالها المستخدم، في هذه الحلقة يفضل استخدام **while** أو **do – while**.
- إذا كنا نحتاج لعداد لمعرفة رقم التكرار أو استخدام قيمته في البرنامج يمكن استخدام حلقة for للاستفادة من عدادها، حيث أن قيمته تبين رقم التكرار.
- إذا كان من الممكن ألا يتم تنفيذ الحلقة أصلاً، فالأصح استخدام حلقة **while**، أما إن كان تنفيذ الحلقة يكتمل للمرة الأولى في كل الأحوال، يتساوى حينها استخدام **while** و **do–while**.
- عموماً عند استخدام لغة **Java** يمكن أن نعبر عن أي فكرة بها تكرار بأيٍ من العبارات التكرارية الثلاث التي توفرها اللغة، وبصورة سليمة وصحيحة، ولكننا دائماً نختار الحلقة الأمثل والأفضل والتي تجعل كتابة البرنامج أسهل وأقل تعقيداً، وتؤدي المطلوب بصورة أكفأ وذلك حسب خواص الحلقة وطبيعة البرنامج المطلوب.

أسئلة تقويم ذاتي



1- اكتب صيغة العبارة التكرارية do - while ؟
2- ما الملاحظات في حلقة do – while ؟
3- ما الملاحظات حول العبارات التكرارية الثلاث ؟

8. تخزين مجموعات البيانات

خلال الدروس السابقة، تعاملنا مع مجموعات بيانات، ونقصد بها عدداً من البيانات من نفس النوع ذات صلة ببعضها، مثل: أسماء عشرة موظفين، درجات طالب في خمس مواد، عشرين عدداً صحيحاً. كنا نقرأ أو نحسب هذه البيانات ونتعامل معها بدون تخزينها، مثلاً نجمع درجات الطالب في متغير واحد لحساب نسبته، أو نطبع

الأعداد على الشاشة دون تخزين. ولكن في كثير من الأحيان نحتاج إلى تخـــزين هـــذه البيانات. فمثلاً لطباعة جدول من أسماء الطلاب ودرجاتهم نحتاج للاحتفاظ بهذه البيانات لعرضها في وقت واحد. أو لطباعة الأعداد مرتبة تصاعدياً نحتاج لتخزينها لكـــي نعيـــد ترتيبها قبل الطباعة.

من غير المنطقي أن نقوم بتعريف متغير مقابل كل مادة من مواد الطالب، لأنها ستكون صعبة الإدارة خصوصاً إذا كان عدد المواد كبير. ولا يمكـــن كـــذلك أن يكـــون الطريقة الوحيدة لترتيب عشرين عدداً تصاعدياً بأن نعرّف عشرين متغيـــراً للاحتفـــاظ بالقيم ومن ثم كتابة مئات العبارات لاختبار محتوياتها واحداً تلو الآخر. ولو كان تخيّـــل ذلك ممكناً، ماذا لو كان لدينا مائة عدد أو ألف موظف؟

يمكن تخزين مجموعة من البيانات بصورة تسهل التعامل معها والوصـــول إليهـــا فيمـــا يعرف بالمصفوفة Array. المصفوفة عبارة عن صف من البيانات ذات علاقة ببعضـــها من نفس نوع البيانات، يكون للمصفوفة اسم واحد وعدد مـــن الحجـــرات توضـــع بهـــا البيانات.

لتعريف مصفوفة من النوع **int** تتكون من 10 خانات:

```
int myArray[ ] = new int[10];
```

حيث نبدأ بالتعريف العادي لمتغير من النوع **int** وأمامه قوسين [ ] للدلالة علـــى أنه اسم مصفوفة. الكلمة المحجوزة **new** نستعملها عند تعريف المصفوفة لتحجـــز لنـــا خانات معينة في الذاكرة، نوضح عدد هذه الخانات بين قوسين يتبعان نوع البيانات الذي نكتبه مرة أخرى بعد كلمة **new**.

التعريف أعلاه لمصفوفة تسمى myArray بها 10 خانات. تحتوي كل خانة على متغير من نوع **int**. كل خانة يشار إليها بعدد فريد يســـمى فهـــرس المصـــفوفة index للتمييز بين الحجرة والأخرى. يبدأ الفهرس من الصفر حيث يشار للعنصـــر الأول فـــي المصفوفة بالاسم myArray[0] ويكون التعامل مع العنصـــر الأول بهـــذا الاســـم. هـــذه المصفوفة تحتوي على عشرة عناصر، لذلك يشار إلى عناصرها بالأسماء الآتية:

العنصر الأول myArray[0]

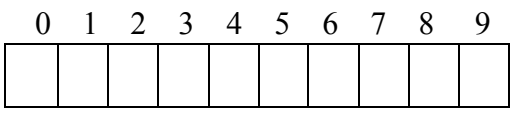
العنصر الثاني myArray[1]

العنصر الثالث myArray[2]

: :

العنصر العاشر myArray[9]

myArray

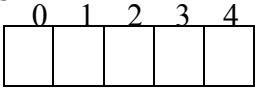




ويجب الانتباه هنا إلى أنه لاوجود للعنصر myArray[10]، حيث إن المصفوفة تحتوي على عشرة عناصر، لذلك فالفهرس تتراوح قيمته من0 إلى 9.

يتم التعامل مع عناصر المصفوفة بصورة عادية مثل المتغيرات. نلاحظ أن المصفوفات تسهل التعامل مع البيانات الكثيرة، لأننا نخاطبها جميعاً باسم واحد مشترك هو اسم المصفوفة، وسنرى لاحقاً سهولة هذا التعامل من خلال الأمثلة.

vowels



char vowels = **new char**[5];

هذه المصفوفة تتكون من خمسة عناصر من النوع char، يمكن تخزين محتوياتها تماماً كالمتغيرات:

عبـــارات تعيـــين

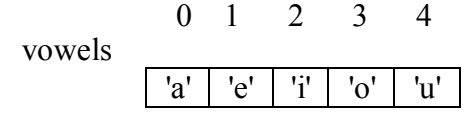
vowels[0]='a'; أو عبارات قراءة

vowels[1]= **Input**.readChar( );

كذلك يمكن وضع قيم ابتدائية لعناصر المصفوفة كالآتي:

char vowels= {'a', 'e', 'i', 'o', 'u'};

ولا نحتاج عندها لكتابة **new char**[5]، حيث يتم الحجز لخمسة عناصـــر فـــي الـــذاكرة تلقائياً.



تعامل حجرات المصفوفة في البرنامج تعامل طبيعي، ويمكن أن تظهر فـــي أي مكـــان يمكن أن تظهر فيه المتغيرات مثلاً:

عبارات الطباعة

System.out.println(array[6]);

التعبيرات الرياضية sum = sum + number[0];

مثال:

المطلوب كتابة برنامج يقرأ قيمة خمسة أعداد مـــن المســـتخدم ويخزنهـــا فـــي مصفوفة تسمى a، ثم يقوم البرنامج بحساب مربعات هذه الأعداد ويخزنها في المصفوفة ثم يطبعها.

الحل:

بما أننا سنقوم بخمس عمليات قراءة، ومثلها لحساب المربعات والطباعة، يمكننا أن نستخدم حلقة **for**، ونستفيد من القيم التي يأخذها عداد الحلقـــة فـــي تحديـــد فهـــرس المصفوفة.

```
int a[ ]=new int[5];
for(int i=0; i<=4; i++)
{
        System.out.println("Enter value number " + i);
        a[i]= Input.readInt( );
}
```

```
for(int i=0; i<=4; i++)
{
        a[i]*=a[i];      //        حساب مربع العدد وتخزينه
        System.out.println("New value number " + i + " is " + a[i]);
}
```

مثال للتنفيذ:

```
Enter value number 0
3
Enter value number 1
2
Enter value number 2
8
Enter value number 3
1
Enter value number 4
5
New value number 0 is 9
New value number 1 is 4
New value number 2 is 46
New value number 3 is 1
New value number 4 is 25
```



عندما نريد التعامل مع جميع عناصر المصفوفة، فالأفضل استخدام حلقة **for**، حيث تتميز بوجود العداد الذي يأخذ قيماً متتاليةً مثل فهرس المصفوفة، وبالتالي يمكن الاستفادة منه للمرور على جميع عناصر المصفوفة لتأدية الوظيفة المطلوبة.

مثال:

المطلوب كتابة برنامج يملأ مصفوفة من عشر خانات بقيم، بحيث إن كل عنصر عبارة عن حاصل جمع العنصرين السابقين. ووضع القيمة 1 في العنصر الأول والثاني للمصفوفة ليبدأ منهما الحساب.

الحل:

سيبدأ الحساب لملء الخانة الثالثة:

```
myArray[2] = myArray[1] + myArray[0]
```

ونتبع نفس الخطوات للخانة الرابعة:

```
myArray[3] = myArray[2] + myArray[1]
```

وعليه لملء العنصر i في المصفوفة:

```
myArray[i] = my Array[i-1] + myArray[i-2]
```

لذلك يمكن كتابة البرنامج على النحو التالي:

```
int myArray[ ]=new int[10];
myArray[0]=1;
myArray[1]=1;
for(int i=2; i<=9; i++)
        myArray[i]=myArray[i-1] + myArray[i-2];
```

المصفوفة الناتجة تمثل سلسلة معروفة تسمى سلسلة فيبوناشي (Fibonacci series)، وعناصرها هي:

1,1,2,3,5,8,13,21,…

مثال:

المطلوب تعريف ثلاث مصفوفات:

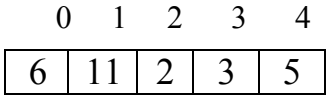
- المصفوفة a وتحتوي على القيم الابتدائية 6، 11، 2، 3، 5.
- المصفوفة b وتحتوي على القيم الابتدائية 1، 4، 9، 18، 7.
- المصفوفة c، وللحصول على عنصر في c، يتم جمع العنصرين المقابلين في a و b. ثم يقوم البرنامج بطباعة المصفوفة c على الشكل:

```
c[0]=
c[1]=
```

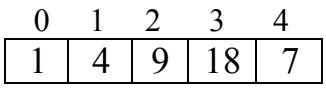
وهكذا :

الحل:

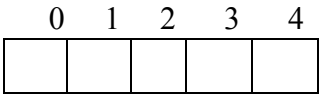
int a[ ]={6, 11, 2, 3, 5}; a



int b[ ]={1, 4, 9, 18, 7}; b



int c[ ]= **new int**[5]; c




```
for(int k=0; k<=4; k++)
        c[k] = a[k] + b[k];
for(int k=0; k<=4; k++)
        System.out.println("c[" + k + "]=" + c[k]);
```

يمكن اختصار الحلقتين في حلقة واحدة تحتوي على عبارتي حساب عناصر c وطباعتها معاً.

مثال:

لدينا مصفوفة تتكون من 30 خانة تحتوي على درجات الحرارة، تم رصدها خلال شهر في مدينة، المطلوب: إيجاد أعلى درجة حرارة، وأدنى درجة حرارة ومتوسط درجات الحرارة خلال الشهر (افترض وجود المصفوفة temp).

الحل:

نقوم بتعريف متغير لتخزين قيمة أعلى درجة حرارة، ونعطي المتغير قيمة ابتدائية صغيرة بحيث نضمن وجود قيمة أعلى منها في البيانات. نعرّف كذلك متغيراً لتخزين قيمة أدنى درجة حرارة، ونعطي المتغير قيمة ابتدائية كبيرة بحيث نضمن وجود قيمة أدنى منها في البيانات.

```
int maximum= -100;      // لا توجد درجة حرارة جوية بهذا الصغر
int minimum=100;        // لا توجد درجة حرارة جوية بهذا الكبر
double sum=0.0, avg=0.0;
```

كلما مررنا على خانة معينة. نجمع قيمتها إلى المتغير sum لحساب المتوسط فيما بعد، ونقارن قيمتها مع أكبر قيمة موجودة حتى الآن (ونجدها في المتغير maximum) وإذا كانت أكبر منها جعلناها هي الأكبر. كذلك نقارن قيمتها مع أصغر قيمة لدرجات الحرارة تم إيجادها إلى الآن، ونجدها في المتغير minimum، وإذا كانت أصغر منها جعلناها هي الأصغر.

```
for(int i=0; i<=29; i++)
{
        sum = sum + temp[i];
        if (temp[i]>maximum)
                maximum=temp[i];
        if (temp[i]<minimum)
                minimum=temp[i];
}
avg= sum/30;
System.out.println("Maximum temperature was " + maximum);
System.out.println("Minimum temperature was " + minimum);
System.out.println("Average temperature was " + avg);
```

يمكن تعديل البرنامج بحيث يكون المطلوب تحديد اليوم الذي وصلت فيه درجة الحرارة إلى أعلى وأدنى درجة خلال الشهر. في هذه الحالة لن يكون المطلوب تخزين درجة الحرارة وحدها، بل الاحتفاظ برقم الخانة أو فهرس المصفوفة الذي وجدنا فيه هذه القيمة الأعلى أو الأدنى لدرجة الحرارة. أي نقوم بتعريف متغير للاحتفاظ بقيمة i التي وجدنا عندها temp[i] أكبر من maximum أو أصغر من minimum:

```
int maxIndex=0, minIndex=0;          //          نضيف متغيرين
```

ويكون التعديل في عبارات الشرط:

```
if (temp[i]>maximum)
{
        maximum=temp[i];
        maxIndex=i;        //          الاحتفاظ باليوم
}
if (temp[i]<minimum);
{
```

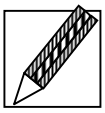
```
        minimum=temp[i];
        minIndex=i;
}
```

وتعديل عبارات الطباعة لتوضيح اليوم الأعلى والأدنى حرارةً:

```
System.out.println("Maximum temperature was " + maximum + " It was on day number " + maxIndex);
System.out.println("Minimum temperature was " + minimum + " It was on day number " + minIndex);
```

يعمل هذا البرنامج بصورة صحيحة عندما تكون درجات الحرارة جميعها مختلفة عن بعضها البعض. لكن إذا كانت درجة الحرارة الأعلى أو الأدنى مكررة، يتم طباعة أول يوم ظهرت فيه هذه الدرجة.

تدريب (16)



المطلوب تعريف مصفوفة من الحروف تسمى lang وتحتوي على حروف كلمة arabic كقيم ابتدائية. ثم كتابة عبارات لإزاحة حروف المصفوفة إلى اليسار خانة واحدة وطباعة المصفوفة الناتجة. ثم تكرار هذه العملية حتى تعود المصفوفة إلى حالتها الأصلية. الحرف الذي تتم إزاحته من أقصى يسار المصفوفة يتم إدخاله في أقصى يمين المصفوفة.

مثال:

اكتب برنامجاً يقرأ من المستخدم عشرة أعداد، ثم يطبعها مرتبة تنازلياً.

الحل:

لترتيب مجموعة من الأعداد – تصاعدياً أو تنازلياً – يجب أن نخزنها في مصفوفة أولاً. توجد الكثير من الطرق المعروفة لترتيب المصفوفات. ولكل طريقة قوانينها ومميزاتها، وهي خارجة عن نطاق هذا الكتاب. ولكن سنقوم باستخدام طريقة من هذه الطرق لشرح كيفية ترتيب مصفوفة من الأعداد تنازلياً.

تنص هذه الطريقة على المرور على جميع عناصر المصفوفة وإيجاد أكبر عنصر بها وتخزينه في الخانة رقم 0، أي استبدال أكبر عنصر مع العنصر الموجود في الخانة رقم 0. ثم تكرار البحث في المصفوفة (ماعدا العنصر رقم0) لإيجاد أكبر عنصر، وهو ثاني أكبر عنصر بعد العنصر المخزن الآن في الخانة رقم 0، يوضع هذا العنصر في الخانة رقم 1 ثم إيجاد أكبر عنصر وتخزينه في الخانة رقم 2، وهكذا إلى نهاية المصفوفة.

نعرف متغير، ونضع فيه أول عنصر بالمصفوفة كقيمة ابتدائية، يتم البحث عن عنصر أكبر منه في المصفوفة، فنحتفظ بفهرسه، لاستبداله بالعنصر الأول في المصفوفة. يمكن التعبير عن هذه الخطوة بالعبارات الآتية:

```
int large = number[0], largeIndex;
for(int i=1; i<=9; i++)
        if (number[i]>large)
        {
                large = number[i];      //        الاحتفاظ بالعدد الأكبر
                largeIndex = i;         //                الاحتفاظ بالفهرس
        }
number [largeIndex] = number[0];
```

العبارة الأخيرة تقوم بوضع أول عنصر في المصفوفة في الخانة التي تحتوي على أكبر عنصر.

```
number[0] = large;
```

بعدها نضع أكبر قيمة وجدناها في أول خانة بالمصفوفة

هذه هي الخطوة الأولى. بتكرار هذه الخطوة يتم ترتيب المصفوفة عنصراً تلو الآخر. عليه نحتاج إلى حلقة خارجية لتكرار هذه الخطوة. ومادام المتغير large يتعامل مع العنصر رقم 0، ثم العنصر رقم 1 ثم 2 .. وهكذا، يمكن أن نجعل عداد الحلقة الخارجية يأخذ هذه القيم لنجعل المتغير large يتعامل مع العنصر التالي في المصفوفة كل تكرار، وهو number[k] ليكون البرنامج الكامل هو:

```
int number[ ] = new int[10];
int large, largeIndex=0;
System.out.println("Enter ten numbers: ");
for (int i=0; i<=9; i++)                    //        إدخـــــال عناصـــــر المصفوفة
{
        System.out.println("Number " + i);
        number[i]=Input.readInt( );
}
for (int k=0; k<=8; k++)
{
        large = number[k];
        for (int i= k+1; i<=9; i++)
        {

                if(number[i]>large)
                {
                        large = number[i];
                        largeIndex=i;
                }
        }
        number[largeIndex] = number[k];
        number[k] = large;
}
//        طباعة المصفوفة تنازلياً
System.out.println("The sorted array is: ");
for(int i=0; i<=9; i++)
        System.out.println(number[i]);
```

نلاحظ أن الحلقة الخارجية تنتهي عند العنصر قبل الأخير (رقم 8)، وذلـــك أن العنصر تتم مقارنته مع العناصر التالية له. فبعد مقارنة العنصر قبل الأخير مع العنصر الأخير، تكون المصفوفة قد ترتبت تماماً. ولا يوجد عنصر تالٍ للعنصر الأخير (رقم 9) لإجراء المقارنات.

مثال للتنفيذ:

```
Enter ten numbers:
5
```

1
6
12
8
10
3
7
4
2

number

0	1	2	3	4	5	6	7	8	9
5	1	6	12	8	10	3	7	4	2

الخطوة الأولى:

ابتداء تكون قيمة المتغير large=5، وبعد اكتمال البحث عن العدد الأكبر بواسطة الحلقة الداخلية يكون:

number

0	1	2	3	4	5	6	7	8	9
5	1	6	12	8	10	3	7	4	2

large = 12, largeIndex =3

ثم نضع في الخانة رقم 3 محتويات الخانة رقم 0، وفي الخانة رقم 0 قيمة المتغير large:

number

0	1	2	3	4	5	6	7	8	9
12	1	6	5	8	10	3	7	4	2

وتنتهي بذلك الخطوة الأولى. وفي الخطوة الثانية:

ابتداء تكون قيمة المتغير large = 1، وبعد اكتمال البحث:

number

0	1	2	3	4	5	6	7	8	9
12	1	6	5	8	10	3	7	4	2

large =10, largeIndex=5

وبعد الاستبدال:

number

0	1	2	3	4	5	6	7	8	9
12	10	6	5	8	1	3	7	4	2

في الخطوة الثالثة يكون large = 6، وبعد اكتمال البحث:

number

0	1	2	3	4	5	6	7	8	9
12	10	6	5	8	1	3	7	4	2

large = 8, largeIndex = 4;

وبعد الاستبدال:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	5	6	1	3	7	4	2

في الخطوة الرابعة يكون large = 5، وبعد اكتمال البحث:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	5	6	1	3	7	4	2

large = 7, largeIndex = 7

وبعد الاستبدال:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	7	6	1	3	5	4	2

في الخطوة الخامسة يكون large = 6، وبعد اكتمال البحث:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	7	6	1	3	5	4	2

large = 6, largeIndex = 4

وبعد الاستبدال:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	7	6	1	3	5	4	2

نلاحظ في هذا التكرار أن العنصر 6 كان في مكانه المناسب إثر استبدال سابق، لذلك لا يطرأ تغيير فعلى على المصفوفة، حيث إنه لا يوجد عنصر أكبر من 6 إلى يمينه.

في الخطوة السادسة يكون large = 1، وبعد اكتمال البحث:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	7	6	1	3	5	4	2

large = 5, largeIndex = 7

وبعد الاستبدال:

number	0	1	2	3	4	5	6	7	8	9
	12	10	8	7	6	5	3	1	4	2

في الخطوة السابعة، يكون large= 3، وبعد اكتمال البحث:

number	0	1	2	3	4	5	6	7	8	9

12	10	8	7	6	5	3	1	4	2

large = 4, largeIndex = 8

وبعد الاستبدال:

number 0 1 2 3 4 5 6 7 8 9

12	10	8	7	6	5	4	1	3	2

في الخطوة الثامنة ، يكون large =1، وبعد اكتمال البحث:

number 0 1 2 3 4 5 6 7 8 9

12	10	8	7	6	5	4	1	3	2

large = 3, largeIndex = 8

وبعد الاستبدال:

number 0 1 2 3 4 5 6 7 8 9

12	10	8	7	6	5	4	3	1	2

وفي الخطوة التاسعة والأخيرة، يكون large =1، وبعد اكتمال البحث:

number 0 1 2 3 4 5 6 7 8 9

12	10	8	7	6	5	4	3	1	2

large = 2, largeIndex = 9

وبعد الاستبدال:

number 0 1 2 3 4 5 6 7 8 9

12	10	8	7	6	5	4	3	2	1

وينتهي عمل الحلقات، ولا يبقى سوى طباعة المصفوفة بعد ترتيبها تنازلياً:

The sorted array is:
12
10
8
7
6
5
4
3
2
1

يمكن تعديل هذا البرنامج ليقوم بترتيب المصفوفة تصاعدياً.

مثال:

اكتب برنامجاً يقرأ ثمانية أسماء لدولٍ يخزنها في مصفوفة تسمى country، ويطبع البرنامج عدد مرات ظهور اسم sudan ضمنها.

الحل:

نقوم بتخزين أسماء الدول في مصفوفة، العنصر فيها عبارة عن **String**. نقوم باختبار قيمة النص، فإذا كانت sudan تزداد قيمة عداد معين، ثم نطبع قيمة هذا العداد النهائية.

```
String country[] = new String[8];
int counter = 0;
System.out.println("Enter eight countries: ");
for(int i=0; i<=7; i++)
{
        country[i]= Input.readString( );
        if (country[i].equals("Sudan"))
        counter++;
}
System.out.println("You entered Sudan " + counter + " times");
```



نلاحظ أن اختبار قيمة النص counter[i] لم تتم باستخدام علامة التساوي ==، لأن هذه العلامة خاصة بمقارنة البيانات الأساسية فقط في **Java** مثل: **char**, **boolean**, **float**, **int** وغيرها. أما النوع **String** فكما أسلفنا، ليس من أنواع بيانات **Java**، وإنما هو **class** (راجع الملحق الأول)، وأي متغير من نوع **class** محدد يجب أن يتعامل مع الدوال المخصصة والمعرفة له.

ولدى **String** مجموعة من الدوال سنتعرف على عدد منها. من هذه الدوال الدالة equals( ) وتستخدم لمقارنة نصين، فمثلاً إذا كان لدينا:

String s="no";

if(s.equals("no")) فإن

عبارة عن اختبار لقيمة s، مادامت s="no" فإن قيمة s.equals("no") تكون **true**. ولكن s.equals("Hello") تكون قيمتها **false** لأن s="no" وليس "Hello".

المتغير country[i] أيضاً عبارة عن **String**. ولذلك لم تستخدم بالاسم قيمته العلامة ==، بل الدالة equals( ). وهنا يزيد العداد counter بواحد إذا كانت قيمة country[i] تساوي "sudan". ولكن ماذا يحدث إذا قام المستخدم بإدخال كلمة "sudan" بالحروف الصغيرة أو "SUDAN" بالحروف الكبيرة، حيث نعلم أن الحرف الصغير يختلف عن الحرف الكبير. ولذلك فقيمة الاختبار country[i].equals("Sudan") تكون **false** لأن Sudan تختلف عن sudan و SUDAN وحتى SudaN.

إذا أردنا أن نأخذ بالاعتبار كلمة Sudan بصرف النظر عن حالة الحروف (كبيرة أو صغيرة) نستخدم الدالة equalsIgnoreCase( ). واسمها يدل على وظيفتها، فهي تقوم باختبار قيمة **String** معينة دون الأخذ بالاعتبار كون الحروف صغيرة أو كبيرة، مثلاً إذا كان:

String s="no";

لنلاحظ نتائج الاختبارات التالية:

s.equals("no"); **true**
s.equals("yes"); **false**
s.equals("No"); **false**

لأن "N" تختلف عن "n"، ولكن:

s.equalsIgnoreCase("No"); **true**

من الأفضل استبدال الدالة equals( ) بالمثال السابق بالدالة equalsIgnoreCase( )، لكي تقوم بحساب كل كلمة Sudan تم إدخالها بصرف النظر عن حالة حروفها.

مثال للتنفيذ:

Enter eight Countries:
Iraq
Chad
Sudan
Italy
Japan
Sudan
Canada
Sudan
You entered Sudan 3 times

مثال:

اكتب برنامجاً يسأل المستخدم عن عدد أبنائه. يقوم البرنامج بعد ذلك بقراءة أسماء أبناء المستخدم وأعمارهم، ثم يطبع هذه المعلومات على النحو التالي:

Name Age

الحل:

في هذا المثال، نلاحظ أننا لا نعلم طول المصفوفة سلفاً. فالمستخدم هو الذي يحدد طول المصفوفة بناءً على عدد أبنائه. من الممكن تعريف مصفوفة بطول كبير – مثلاً 30– بحيث نضمن كفايتها لتخزين جميع البيانات. ولكن من غير المعقول أن نحجز كميات من الذاكرة دون الحاجة إليها. وقد يكون عدد الأبناء صفراً أو واحداً، ولا نريد أن نحجز خانات من الذاكرة دون استغلالها.

يمكن أن نقرأ من المستخدم عدد الأبناء، ثم نعرف المصفوفة بعد ذلك بالطول الذي اختاره، وبالتالي نحجز فقط بالقدر المطلوب الذي نحتاجه في البرنامج. سنقوم بتعريف مصفوفتين، واحدة من النوع **String** لا يتم بأسماء الأبناء، والأخرى من النوع **short** لا يتم بأعمارهم.

```
System.out.println("How many children have you got?");
int noOfchildren = Input.readInt( );
```

```
String name[ ]= new String[noOfchildren]; //        مصفوفة الأسماء
short age[ ] =new short[noOfchildren];        //        مصفوفة الأعمار
for(int i=0; i<noOfchildren; i++)
{
        System.out.println("Name of child number " + (i+1));
        name[i] = Input.readString( );
        System.out.println("His/her age");
        age[i]= (short) Input.readInt();
}
System.out.println("Name                Age");
System.out.println("---------------------");
for(int i=0; i< noOfchildren; i++)
        System.out.println(name[i] + "\t\t" + age[i]);
```

نلاحظ أن عداد حلقة **for** يبدأ من صفر وينتهي عند العدد الأقل من قيمة noOfChildren، ذلك أن المصفوفة إذا تكونت من 4 خانات، يكون فهرس هذه الخانات هو 0, 1, 2, 3، ولكن العدد 4 نفسه لا يمثل أحد فهارس المصفوفة.

تم طباعة عمر الابن إلى جانب اسمه، حيث إن فهرس اسم الابن في المصفوفة name يقابل فهرس اسم نفس الابن في المصفوفة age. قمنا بعمل مسافة بين الاسم والعمر بواسطة "\t" وهي نص يعني طباعة tab أو ثمانيَ مسافات على الشاشة.

مثال للتنفيذ:

```
How many children have you got?
4
Name of child number 1
Dalia
His/her age
11
Name of child number 2
Abdalla
His/her age
9
Name of child number 3
Abdallatif
```

```
His/her age
8
Name of child number 4
Dalal
His/her age
3
        Name            Age
   ---------------------------------
        Dalia                   11
        Abdalla                  9
        Abdallatif               8
        Dalal                    3
```

مثال:

اكتب برنامجاً يقرأ عدداً من الحروف المتتالية، يحدد البرنامج عدد مرات ظهور الحرف e بينهما.

الحل:

لتخزين مجموعة من الحروف، لا نحتاج دائماً إلى مصفوفة من الحروف، وبدلاً عن ذلك يمكن أن نستخدم متغير من النوع **String**، لأنه يمثل مجموعة حروف أيضـــا، ويتوفر للمتغير من هذا النوع الكثير من الدوال التي تســـاعد علـــى إجـــراء العمليـــات والاختبارات على المتغير. تعرضنا خلال الأمثلة السابقة إلى عدد مـــن هـــذه الـــدوال، ونتعرف في هذا المثال على دالتين إضافيتين.

```
int counter =0;
System.out.println("Enter characters");
String str = Input.readString( );
for(int i=0; i<str.length( ); i++)
{
        if(str.charAt(i)== 'e')
        counter++;
}
System.out.println("The letter e was found " + counter + " times");
```



نلاحظ أننا لم نسأل عن عدد الحروف المدخلة، وبدلاً عن ذلك استخدمنا الدالة length( )، والتي تعطينا طول النص أو عدد الحروف الموجودة به.

ولإجراء الاختبار يجب أن نختبر الحروف كلاً على حدة. فاستخدامنا الدالة charAt() والتي تأخذ بين قوسيها رقماً، وتوضح الحرف الموجود عند ذلك الموقع من النص مثل المصفوفات تماماً.

من المثال نلاحظ أن النصوص يمكن أن تغنينا عن استخدام مصفوفات من الحروف، والذي قد يسهل البرنامج خصوصاً في عمليات القراءة والطباعة، وبوجود الدوال الخاصة بالنصوص.

مثال للتنفيذ:

Enter characters
I have learned a lot about Java Language
The letter e was found 4 times

عند التعامل مع المصفوفة، يجب أن نوضح العنصر الذي نتعامل معه، وذلك بتحديد فهرسه، مثلاً:

Array[1], a[n], name[i]

ومن الخطأ ظهور اسم المصفوفة دون الفهرس، لأنه اسم لمصفوفة، أي مجموعة من العناصر، و يجب توضيح العنصر المطلوب.

أما المتغير المعرف من النوع **String**، فرغم أنه مجموعة من الحروف التي يمكن التعامل معها كل على حدة كما حدث في المثال السابق، إلا أنه يمكن أن يعامل معاملة المتغير الواحد كما رأينا في كثير من الأمثلة.

من أهم العمليات على النصوص هي الدمج concatenation، ويعني دمج نصين في نص واحد، وله عدة طرق:

1- باستخدام علامة +، وقد استخدمناها لدمج نصوص بين قوسي الدالة print، مثلاً:

```
String s= "Java" + " is easy";
```

هنا يكون

s= "Java is easy"

ويمكن أن نقوم بدمج متغيرات أيضاً:

```
String S1= "Java", S2= " is easy", S3;
S3= S1+S5;
```

هنا يكون

S3= "Java is easy"

2- باستخدام الدالة ( )concat ونضع بين قوسيها النص الذي نريد دمجه، مثلاً:

```
String s= "Java";
s.concat(" is easy");
```

هنا يكون s="Java is easy"

أو دمج متغير:

```
String s= "Java", t= " is easy";
s.concat(t);
s= "java is easy"
```

وهناك الكثير من الدوال المفيدة الخاصة بالمتغيرات من النوع **String**.

9. الدوال في جافا

عزيزي الدارس،

الدالة هي مجموعة من التعليمات التي تؤدي وظيفة معينة، يتم نداؤها خلال البرنامج عند الحاجة بواسطة اسمها. في لغة **Java** تسمى الدوال methods. وهناك

الكثير من الدوال الموجودة والمعرّفة أصلاً في لغة **Java** والتي يمكن أن نستخدمها عندما نريد. من اكثر دوال لغة **Java** التي استخدمناها خلال الأمثلة الدالة print والدالة println، وهما دالتان الغرض منهما الطباعة على الشاشة. وخلال الدرس السابق تعرفنا على بعض الدوال الخاصة بالنصوص مثل: equals و concat، وجميع هذه الدوال موجودة باللغة، وإذا أردنا استخدامها لأداء وظيفتها خلال برنامج، وجب علينا معرفة اسمها ليتم نداؤها بصورة صحيحة، ويجب علينا كذلك معرفة الوسائط التي تحتاجها، والوسيط هو ما تحتاج الدالة لمعرفته واستقباله من المستخدم لتبدأ إنجاز عملها، فمثلاً الدالة print تستقبل رسالة نصية كوسيط بين قوسيها، وهو عبارة عن الرسالة المطلوب طباعتها، ووسيط الدالة equals عبارة عن النص المطلوب اختبار تساويه مع المتغير من النوع **String**.

أخيراً يجب علينا معرفة القيمة التي تعيدها الدالة بعد الفراغ من أداء وظيفتها. فمثلاً الدالة equals تقوم باختبار تساوي نصين، وبعد أن تنتهي هذه المهمة تقوم بإعادة قيمة من نوع **boolean**، ولذلك عندما نناديها إما أن يكون النداء بين قوسي شرط if (لأنه يتوقع وجود قيمة **boolean**) أو أننا سنقوم بتعريف متغير نوعه **boolean** ليحمل هذه القيمة. أيضاً الدالة readInt، لا تأخذ وسائط، ولذلك نجد قوسيها خاليين، لكنها تعيد قيمة **int** بعد أداء وظيفتها وهي قراءة قيمة من المستخدم، وبالتالي دائماً نجد نداء هذه الدالة على الصورة:

```
int n=Input.readInt( );
```

ليقوم المتغير n بحمل القيمة التي تعيدها الدالة. وكذلك جميع دوال القراءة المختلفة.

تحتوي لغة **Java** على عدد هائل جداً من الدوال ذات الوظائف المختلفة في شتى المجالات، ولا يتسع المجال لذكرها، بل ولا يمكن حصر جميع الدوال في متناول اليد، ولكن يبحث المبرمج عن الدوال التي يحتاجها بناءً على مجالها. ويمكن أن نتطرق لبعض الدوال التي تتعامل مع أنواع البيانات البسيطة في مثالين.

مثال:

اكتب برنامجاً يقرأ رقم جواز المستخدم، يقوم البرنامج بطباعة رقم الجواز التالي للرقم المدخل. بافتراض أن رقم الجواز يتكون من حرف متبوع بأربعة أعداد، مثلاً B1652.

الحل:

الفكرة في هذا المثال هو أن نقرأ من المستخدم رقم الجواز، وبما أنه يتكون من حروف وأرقام معاً، لا نستطيع أن نستخدم متغير من نوع عددي، لكننا نستطيع أن نستخدم متغير من النوع **String** حيث يمكنه حمل أي مجموعة حروف، فالنص "123" هو نص صحيح تماماً مثل النص "abc". بعد أن نقرأ رقم الجواز نقوم باقتطاع الجزء العددي وهو الخانات الأربع الأخيرة في النص، وتوجد دالة خاصة لهذه الوظيفة وهي الدالة substring. نحول هذا الجزء المقطوع إلى عدد ثم نزيد قيمته بمقدار الواحد ثم نعيد تركيب رقم الجواز الجديد بعد أن نقوم بتحويل العدد إلى نص مرة أخرى. توجد دوال خاصة لعمليات التحويل.

```
1.System.out.println("Insert your passport number");
2.String passport = Input.readString( );
3.String numberPart = passport.substring(1);          //   الاقتطاع
4.int num = Integer.parseInt(numberPart);             //   التحويل إلى عدد
5.num++;                                              //   إيجاد العدد التالي
6.numberPart = String.valueOf(num);                   //   إعادة التحويل إلى نص

7.String newPassport = passport.substring(0,1);
8.newPassport = newPassport.concat(numberPart);       //   الدمج
9.System.out.println("Next passport number is: " + newPassport);
```

في السطر 3 استخدمنا الدالة substring لاقتطاع جزء من النص في متغير نصّي منفصل. هذه الدالة تستقبل عدداً كوسيط. هذا العدد يمثل الفهرس الذي يبدأ من

الاقتطاع حيث إن المتغير من النوع **String** يشبه مصفوفة من الحروف، أول حرف فيها يقع في الخانة رقم 0 ونحن نريد اقتطاع الخانة الثانية إلى نهاية المصفوفة.

في السطر 4 قمنا بتحويل المتغير النصي numberPart والذي يحتوي على نص من الأرقام إلى **int**، وذلك بواسطة الدالة **Integer**.parseInt وهي دالة تستقبل **String** كوسيط وتقوم بتحويله إلى عدد، وتم وضع هذا العدد في المتغير num، والذي قمنا بزيادة قيمته للحصول على الرقم الجديد.

في السطر 6 تم تحويل الرقم الجديد إلى نص وتخزينه في المتغير numberPart مرة أخرى، وذلك باستخدام الدالة **String**.valueOf والتي تأخذ رقماً كوسيط وتقوم بتحويله إلى نص.

في السطر 7 قمنا بتعريف متغير لحمل رقم الجواز الجديد. الرقم الجديد عبارة عن حرف متبوع بأربعة أعداد أيضاً، وهي الحرف القديم إضافةً إلى الرقم الجديد الذي حصلنا عليه في المتغير numberPart، حصلنا على الحرف باقتطاعه من رقم الجواز القديم، وهذا الحرف موجود في أول خانة من المتغير passport، لذلك استخدمنا الدالة substring مرة أخرى. لكن لو استخدمنا الأمر:

String newPassport = passport.substring(0);

كان ذلك يعني اقتطاع النص passport ابتداءً من الخانة رقم 0 وحتى نهاية النص، أي اقتطاع كل النص. ولكن نحن نريد اقتطاع الحرف الأول فقط من النص (حرف الجواز)، لذلك استخدمنا نسخة أخرى من الدالة substring تستقبل عددين كوسيطين، الأول يوضح فهرس الحرف الذي يبدأ منه الاقتطاع، والآخر يوضح فهرس الحرف الذي ينتهي عنده الاقتطاع، وفي هذا المثال يكون الوسيطان 0 و 1 لأننا نريد اقتطاع حرف واحد فقط من النص.

أصبح المتغير newPassport يحتوي على حرف الجواز. وبقى أن نضيف إليه الرقم الجديد، وهو موجود في المتغير numberPart، لذلك استخدمنا الدالة concat في

السطر 8 لإجراء عملية الدمج، ليصبح المتغير newPassport يحتوي على رقم الجـــواز الجديد.

مثلاً: إذا قام المستخدم بإدخال رقم الجواز B1652

نص	numberPart = "1652"
عدد	num = 1652
عدد	num = 1653
نص	numberPart = "1653"
نص	newPassport = "B"
الرقم المطلوب	new Passport = "B1653"

مثال:

اكتب برنامجاً يقرأ عدداً صحيحاً من المستخدم، يطبع البرنامج:

- القيمة المطلقة للعدد المدخل.
- الجذر التربيعي للقيمة المطلقة.
- قيمة العدد مرفوع للقوة4.

الحل:

توفر **Java** الكثير من الدوال لحساب مثل هذه العمليـــات الحســـابية الشـــائعة، إضافة إلى حساب الدوال المثلثية وعمليات أخرى. ولحل هذا المثال سنستخدم بعضاً من هذه الدوال وهي:

- الدالة **Math**.abs(x) وتحسب القيمة المطلقة للعدد
- الدالة **Math**.sqrt(x) وتحسب الجذر التربيعي للعدد x
- الدالة **Math**.pow(x, y) وتحسب قيمة x مرفوع للقوة y

```
System.out.println(“Enter a number: “);
int number = Input.readInt( );
int absolute = Math.abs(number);
double squareRoot = Math.sqrt(absolute);
double power = Math.pow(number,4);
System.out.println(“The absolute value is: “ + absolute);
System.out.println(“The square root is: “ + squareRoot);
System.out.println(number + “ to the 4 is “ + power);
```

مثال للتنفيذ:

```
Enter a number:
-9
The absolute value is: 9
The square root is: 3.0
-9 to the 4 is: 6561.0
```

خلال المثاليين السابقين قمنا باستخدام عدد من دوال لغة **Java**، وذلك لنتعلم كيفية استخدام الدوال بصورة صحيحة، وذلك بوضع الوسائط الصحيحة بين قوسي الدالة إذا كانت تتطلب تمرير وسائط، وتعريف المتغير المناسب لاستقبال القيمة التي تعيدها الدالة بعد إنجاز مهمتها. لكن الغرض من هذا الدرس ليس أن نستخدم دوال لغة **Java** المكتوبة أصلاً، وإنما الغرض هو أن نتعلم كيف نكتب الدوال الخاصة بنا. أي أن نصمم دوالياً بأنفسنا ثم نستعين بها خلال البرنامج كما استعنا بدوال لغة **Java**. الغرض هو أن نصمم دوالياً يتم نداؤها في البرنامج لتؤدي وظيفتها في المكان المناسب.

ولنتعلم كيف نكتب دالة بلغة **Java**، يجب أن نتعرف إلى الشكل العام للدالة. أول دالة تعرفنا إليها هي الدالة الرئيسية main method، وهي دالة ذات طبيعة خاصة ويجب أن تكون موجودة في جميع البرامج التي كتبناها خلال هذا الكتاب. الدالة main يبدأ منها تنفيذ البرنامج، ويتم نداؤها تلقائياً بمجرد تنفيذ البرنامج مرة واحدة فقط ولا يناديها المبرمج مرة أخرى. لكن الشكل العام لجميع الدوال يشبه الدالة main.

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

```
    // method body
}
```

الكلمة المحجوزة **public** في بداية تعريف أي دالة توضح أن هذه الدالة يمكن أن يتم نداؤها من برنامج **Java** آخر. ولها معانٍ أكبر وخيارات أخرى بديلة، ولكن خلال هذه المرحلة سنقصر استخدامنا على الكلمة المحجوزة **public**. ظهور الكلمة المحجوزة **static** **في** أي دالة أخرى تفيد بأن هذه الدالة يمكن نداؤها من داخل الدالة main، وكذلك لها معانٍ أوسع خارجة عن نطاق هذا الكتاب، وسنلتزم بكتابتها أيضاً.

الكلمة التالية في تعريف الدالة توضح نوع البيانات التي تعيدها الدالة بعد الفراغ من أداء وظيفتها، مثلاً الدالة **Input**.readInt( ) تعيد قيمة من النوع **int**، لذلك يمكن أن نتوقع أن يكون تصميمها على النحو التالي:

public static int readInt( )

أما الدالة التي لا تعيد قيمة، نضع في هذه المساحة الكلمة المحجوزة **void**، مثل الدالة main.

في نهاية تعريف الدالة نجد اسمها، ويمكن أن يكون أي اسم يخضع لشروط التسمية في لغة **Java**، ويفضل أن يبدأ بحرف صغير small ويكون دالاً على وظيفة الدالة مثل print و equals. نجد بعد اسم الدالة قوسين، إذا كانت الدالة تستقبل قيمة معينة، نجد بين القوسين تحديداً لنوع وعدد الوسائط التي تستقبلها الدالة، مثلاً الدالة print تستقبل متغيراً من نوع **String** لتقوم بطباعة محتوياته على الشاشة، لذلك نتوقع أن يكون تعريفها على النحو التالي:

public static void print(**String** s)

أي أن الدالة print تستقبل وسيطاً واحداً من النوع **String** تضعه في المتغير s لتتعامل معه في داخلها. كذلك الدالة **Math**.pow( ) تأخذ وسيطين عدديين مثل النوع **int**، وتقوم بحساب قيمة العدد الأول مرفوعاً للقوة المحددة في العدد الثاني، وتقوم بإعادة هذه القيمة كعدد من النوع **double**، لذلك نتوقع أن يأخذ تعريفها الشكل :

public static double pow(**int** x, **int** y)

أما الدالة التي لا تستقبل وسائط، فيكون قوساها خاليين.

مثلاً: الدالة readChar، لا تستقبل وسائط، ولكنها تعيد متغيراً من النوع **char**، بالتالي يمكن أن يكون تعريفها كالآتي:

public static char readChar( )

توضع جميع عبارات الدالة التي تقوم بها بين قوسي بداية ونهاية، وبذلك تكون الدالة جاهزة ليتم نداؤها متى احتاج المبرمج ذلك. وكما ذكرنا سابقاً، لنداء دالة معينة والتعامل معها يجب معرفة اسمها ووظيفتها ووسائطها ونوع البيانات التي تعيدها كنتيجة لتنفيذها. بإمكاننا الآن أن نبدأ بتعريف دالة خاصة بنا واستخدامها في البرنامج.

مثال:

اكتب دالةً تسمى hello، تقوم هذه الدالة بطباعة رسالة على الشاشة، ثم نادي هذه الدالة من الدالة main.

الحل:

```
class MathodTest1
{
        public static void hello( )
        {
                System.out.println("This is my first method");
        }
        public static void main(String args[ ])
        {
                System.out.println("The main");
        }
}
```

هذا تعريف لدالة تسمى hello بداخل برنامج. والبرنامج بهذه الصورة صحيح لغوياً، ولكن الدالة hello لن يتم تنفيذها، ذلك لأنه لم يتم نداؤها. الدالة main هي الدالة الوحيدة التي يتم نداؤها تلقائياً في بداية تنفيذ كل برنامج. ولكن كل الدوال سواها يجب أن يتم نداؤها صراحةً في المكان المطلوب. لذلك فالبرنامج السابق ينفذ عبارة الطباعة الموجودة داخل الدالة main فقط، وإذا أردنا تنفيذ واختبار الدالة hello يجب أن نناديها بواسطة اسمها، ولذلك يجب أن نعيد كتابة الدالة main كالآتي:

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

```
{
        System.out.println("The main");
        hello( );
}
```

عبارة ( )hello هو نداء للدالة. فعندما ننادي دالة معينة، نناديها بواسطة اسمها، ونلاحظ في هذه الدالة أنه لا توجد وسائط لتمريرها مع النداء، ولا تعيد الدالة قيمة معينة.

مثال:

اكتب دالةً تسمى nextNumber، تستقبل الدالة عدداً صحيحاً وتقوم بطباعة العدد التالي على الشاشة.

الحل:

```
class MethodTest2
{
        public static void nextNumber(int x)
        {
                x++;
                System.out.println(x);
        }
        public static void main(String args[ ])
        {
                int myNumber = 6;
                nextNumber(myNumber);
                // or nextNumber(6);
        }
}
```

تستقبل الدالة nextNumber متغيراً من النوع **int**، وتضع قيمة هذا المتغير في متغير اسمه x تتعامل معه بداخلها.

عندما ننادي هذه الدالة من داخل الدالة main يجب أن نضع وسيطاً من النوع **int** بين قوسي نداء الدالة، هذا المتغير هو myNumber، ويحتوي على القيمة 6، تقوم الدالة بزيادة قيمته بواحد ثم تطبعه. نلاحظ أن العمليات التي تجريها الدالة على المتغير

x لا تؤثر على المتغير myNumber، لأننا فقط نقوم بوضع قيمة المتغير myNumber في المتغير x. نلاحظ كذلك أنه يجب أن نمرر إلى الدالة متغيراً أو قيمةً من نفس نوع البيانات التي تستقبلها الدالة كوسيط.

النداءات التالية للدالة nextNumber جميعها نداءات صحيحة:

```
nextNumber(3);
nextNumber(20000);
nextNumber((int) 'a');          //        تحويل البيانات
int i=10;
nextNumber(i);
```

والنداءات التالية للدالة nextNumber جميعها خطأ:

```
nextNumber(3.3);
nextNumber(1000000000000);      //        خارج مدى النوع int
nextNumber("error");
  oolean b= false;
nextNumber(b);
```

مثال:

اكتب دالةً تسمى factorial، تستقبل هذه الدالة قيمة من النوع **int**، تقوم الدالة بحساب مضروب العدد الوسيط وتعيد قيمة المضروب كقيمة من النوع **long**.

الحل:

في هذا المثال، تقوم الدالة بإعادة قيمة من نوع **long**، وعليه يجب مراعاة نقطتين:

أولا: عند نداء الدالة يجب أن نضع القيمة التي تعيدها الدالة بمتغير من النوع **long**، أي من نفس نوع البيانات التي تعيدها الدالة.

ثانياً: آخر عبارة في الدالة يجب أن توضح القيمة التي ستعيدها الدالة.

هذه الدالة factorial تستقبل عدداً من النوع **int** وتعيد متغيراً من النوع long، لذلك يكون تعريفها كالآتي:

```
public static long factorial(int n)
```

وبين قوسي الدالة نكتب عبارات عادية لحساب مضروب العدد n.

```
Long fact = 1;
for(int i = 2; i<= n; i++)
        fact *= i;
```

وآخر عبارة في الدالة يجب أن توضح أن القيمة التي ستعيدها الدالة هي الموجودة بالمتغير fact، وذلك يكون بكتابة الكلمة المحجوزة **return** يليها القيمة التي نريد إعادتها.

```
Return fact;
```

نلاحظ أن الدالة لا تطبع قيمة المضروب، بل تعيده، ولطباعته نستخدم المتغير الذي حمل قيمة المضروب في الدالة main.

```
Class MethodTest3
{
        public static long factorial(int n)
        {
                long fact = 1;
                for(int i=2; i<=n; i++)
                        fact *= i;
                return fact;
        }
        public static void main(String args[ ])
        {
                long num = factorial(5);
                System.out.println("factorial 5 = " + num);
        }
}
```



نلاحظ الآتي:

1. يجب أن يتطابق نوع البيانات الذي تعيده الدالة مع نوع البيانات الموجودة في عبارة **return**.
2. يجب أن يتطابق نوع الوسيط الذي نمرره مع نداء الدالة مع نوع البيانات الذي تستقبله الدالة والموضح بين قوسيها في تعريفها.
3. يجب أن يتطابق نوع بيانات المتغير الذي سيحمل القيمة التي تعيدها الدالة بعد ندائها مع نوع البيانات الذي تعيده الدالة والموضح في تعريفها.

يمكن تعديل الدالة factorial بحيث تضع اعتباراً خاصاً للعددين 0 و 1، حيث إن مضروبهما يساوي 1، لكن الدالة لا تعطي قيماً صحيحة لأنه تم إعدادها لحساب مضروب الأعداد من 2 فأكثر.

```
Public static long factorial(int n)
{
        long fact = 1;
        if((n != 1) && (n != 0))
        {
                for(int i=2; i<= n; i++)
                        fact *= i;
        }
        return fact;
}
```

for وذلك لكي تعيد القيمة 1 إذا كان العدد المدخل هو 0 أو 1 ولا تعمل حلقـــة في هذه الحالة.

يمكن كذلك أن نضع شرطاً في بداية الدالة:

```
if ((n == 0) || (n == 1))
        return 1;
else
{
        long fact = 1;
        for(int i=2; i<=n; i++)
                fact *= i;
```

```
        return fact;
}
```

ولكن الأفضل دائماً أن تكون هنالك عبارة **return** واحدة فقط في نهاية الدالـــة، لأنه في غير هذه الحالة نضطر إلى التأكد من أن كل تفرع أو شرط في الدالة يوجد في نهايته عبارة **return**، وأنه لا يوجد أي تفرع أو شرط يحتوي خلال تنفيذه علـــى أكثـــر من عبارة **return** واحدة. وهذه تعقيدات لاداعي لها طالما كان هنـــاك فرصـــة لجعـــل الأمر أسهل، وهو كتابة عبارة **return** واحدة في نهاية الدالة تعيد قيمة معينة يمكـــن أن تختلف باختلاف الشرط الذي تحقق والتفرع الذي سلكه تنفيذ البرنامج.

مثال:

اكتب دالةً تسمى prime، تختبر الدالة العدد من النوع **int** الذي تستقبله كوسيط، وتحدد ما إذا كان العدد هو عدد أولي، تعيد الدالة هذه النتيجة على شكل متغير من النوع **oolean** .

الحل:

يكون العدد أولياً إذا لم يقبل القسمة إلا على نفسه والعدد 1.

```
Class MethodTest4
{
        public static    oolean prime(int num)
        {
                  oolean isPrime=true;
                for(int i=2; i<num; i++)
                        if(num % i == 0)
                                isPrime = false;
                return isPrime;
        }
        public static void main(String args[ ])
        {
                  oolean b=prime(18);
```

```
        if(b)           // if(b = = true) (prime)
                System.out.println("18 is a prime number");
        else
                System.out.println("18 is not a prime number");
    }
}
```

نفترض ابتداءً أن العدد أولي. لكن إذا كان يقبل القسمة علـــى أي عـــدد مـــن 2 وحتى num-1، يكون العدد غير أولي، وبالتالي ستعيد الدالة القيمة **false**. هـــذه القيمـــة التي تعيدها الدالة يستقبلها المتغير b، وبناءً على قيمته التي توضح إن كان العدد أولـــي أو لا، يتم طباعة الرسالة المناسبة.

مثال:

اكتب دالةً تسمى absolute لإيجاد القيمة المطلقة للعدد، أي دالة تحـــاكي عمـــل الدالة **Math**.abs( ).

الحل:

```
class MethodTest5
{
    public static int absolute(int x)
    {
        if(x<0)
            x = -x;
        return x;
    }
    public static void main(String args[ ])
    {
        System.out.println(absolute(-5));
        System.out.println(absolute(3));
    }
}
```

مخرجات البرنامج:

```
5
3
```

نلاحظ من خلال الأمثلة أننا نقوم بكتابة الدالة main، وذلك لتوضيح طريقة من طرق نداء الدالة التي قمنا بتصميمها والتعامل معها حسب وسائطها والقيمة التي تعيدها. هذا لا يعني عدم صحة طرق أخرى. فمثلاً في المثال السابق، كان بالإمكان قراءة الأعداد من المستخدم، وكتابة الدالة main على الشكل:

```
public static void main(String args[ ])
{
        System.out.println("Enter a number");
        int num= Input.readInt( );
        int abs= absolute(num);
        System.out.println("Absolute value is " + abs);
}
```

أية عبارات صحيحة يتم كتابتها داخل الدالة main لاختبار عمل الدالة الأخرى يعتبر صحيحاً. ولا مانع أن يكون نداء الدالة بداخل عبارة طباعة كما في الحل الأول.

مثال:

أكتب برنامج يستقبل رقم خيار المستخدم، حيث:

1. لحساب مساحة مربع.
2. لحساب مساحة دائرة.

يتم نداء الدالة المناسبة حسب الخيار الذي اختاره المستخدم، ويتم قراءة أبعاد الشكل الهندسي المطلوب، وطباعة مساحته.

الحل:

```
class ShapeArea
{
        public static void main(String args[ ])
        {
                System.out.println("Choose 1 or 2:");
                System.out.println("1. Square");
                System.out.println("2. Circle");
                int choice = Input.readInt( );
                if(choice ==1)
                        squareArea( );
                else
```

```
            if(choice ==2)
                circleArea( );
            else
                System.out.println("Wrong Choice");
    }
    public static void squareArea( )
    {
            System.out.println("Enter the square's edge");
            int edge = Input.readInt( );
            System.out.println("The area is " + edge*edge);
    }
    public static void circleArea( )
    {
            System.out.println("Enter the circle's radius");
            double radius = Input.readDouble( );
            System.out.println("The area is " + 3.14*radius*radius);
    }
}
```

مثال للتنفيذ:

```
Choose 1 or 2:
1. Square
2. Circle
1
Enter the square's edge
8
The area is 64
```

مثال آخر للتنفيذ:

```
Enter 1 or 2:
1. Square
2. Circle
2
Enter the circle's radius
4
The area is 50.24
```

يوضح هذا المثال إمكانية وجود دوالٍ معرفة بالبرنامج ورغم ذلك لا يتوقف استخدامها. إذا اختار المستخدم الخيار 1، يتم نداء الدالة ( )squareArea، ولا يتم نداء الدالة ( )circleArea, ولكن إذا اختار المستخدم الخيار 2، في هذه الحالة يتم نداء الدالة ( )circleArea ولا يتم نداء الأخرى رغم وجودها بالبرنامج. وهذا يعزز ما ذكرناه عن الدالة في بداية الدرس حيث أشرنا إلى أنها جزء من برنامج يتم إعداده للاستخدام، وينادي المبرمج الدالة عند الحاجة.

جميع المتغيرات التي تم تعريفها داخل الدالة تسمى متغيرات محلية، وتكون معرفة بداخل الدالة فقط. فمثلاً المتغير edge تم تعريفه بداخل الدالة ( )squareArea ولذلك يمكن التعامل معه من أي مكان داخل الدالة ( )squareArea، ولكنه لا يكون معروفاً داخل الدالة ( )circleArea ولا الدالة main. وكذلك المتغيرات المحلية داخل الدالة main لا يمكن مخاطبتها في أي مكان آخر خارج الدالة main، لأنها محلية، ومكانها هو الدالة main حيث تم تعريفها. وأيّ محاولة لمخاطبة متغير محلي في دالة غير الدالة التي تم تعريفه فيها، ينتج عنه خطأ.

مثال:

اكتب برنامجاً يقرأ ثلاثة أعداد من المستخدم، وينادي الدالة ( )getAverage التي تحسب وتعيد متوسط الأعداد. تستعين هذه الدالة بدالة أخرى ( )getSummation لحساب مجموع الأعداد.

الحل:

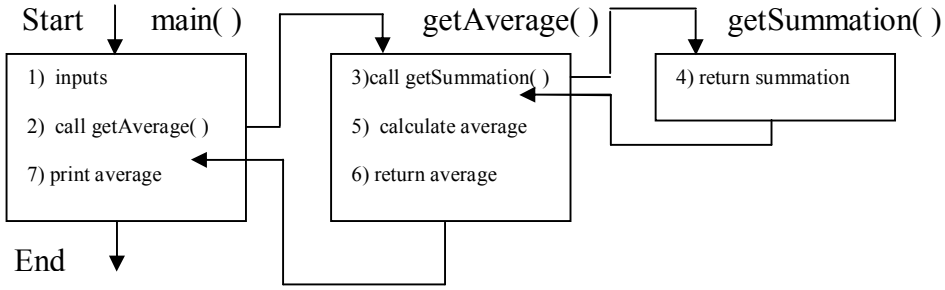
```
class Average
{
        public static void main(String args[ ])
        {
                System.out.println("Enter three numbers");
                int num1 = Input.readInt( );
```

```
            int num2 = Input.readInt( );
            int num3 = Input.readInt( );
            double avg = getAverage(num1, num2, num3);
            System.out.println("The average is: " + avg);
        }
        public static double getAverage(int x, int y, int z)
        {
            int sum = getSummation(x, y, z);
            double average = sum/3;
            return average;
        }
        public static int getSummation(int a, int b, int c)
        {
            return a+b+c;
        }
}
```

يبدأ تنفيذ البرنامج من الدالة main حيث يتم إدخال ثلاث قيم للمتغيرات num1, num2, num3، والتي يتم تمريرها كوسائط للدالة getAverage( )، والتي تضع قيمهم في المتغيرات المحلية x, y, z على التوالي. يتم تمرير هذه القيم مباشرة كوسائط عند نداء الدالة getSummation( ) والتي تضع قيمهم في المتغيرات a, b, c على التوالي، وتعيد الدالة getSummation( ) مجموع المتغيرات الثلاثة، ليتم استقبال المجموع في المتغير sumفي الدالة getAverage( )، لتواصل الدالة getAverage( ) عملها وتحسب المتوسط وتعيده، هذا المتوسط يتم استقباله في الدالة main في المتغير avg. لتكمل الدالة main تنفيذ عباراتها حيث تقوم بطباعة المتوسط.

يوضح هذا المثال أنه بإمكان أي دالة أن تنادي دالة أخرى مادامت تحتاجها لأداء وظيفة تساعدها على إنجاز وظيفتها نفسها.

الشكل التالي يوضح انتقال التنفيذ بين الدوال بالمثال السابق:



يمكن تعديل المثال بحيث يتم تخزين الأعداد الثلاثة في مصفوفة، وبدلاً عن تمرير ثلاثة وسائط بين الدوال، نمرر مصفوفة واحدة وعدد عناصرها. ولكن تمرير المصفوفات يختلف قليلاً عما سبق. والأمثلة التالية تشرح ذلك.

مثال: اكتب مخرجات البرنامج التالي:

```
class MethodTest6
{
        public static void addOne(int n)
        {
                n++;
                System.out.println("Inside method: " + n);
        }
        public static void main(String args[])
        {
                int x = 5;
                System.out.println("Before method: " + x);
                addOne(x);
                System.out.println("After method: " + x);
        }
}
```

عند نداء الدالة addOne يأخذ المتغير n قيمة الوسيط x وهي 5، ثم تزيد قيمة n بواحد. لكن المتغير x يظل على حالته، لأن قيمته فقط هي التي يتم تمريرها إلى الدالة. لذلك قيمة x لا تتغير، وبالتالي تكون المخرجات هي:

```
Before method: 5
Inside method: 6
```

After method: 5

وينطبق هذا على جميع أنواع البيانات البسيطة في **Java**، فعند استخدامها كوسائط، تنتقل قيمتها إلى المتغير المناظر في الدالة، ولا تتأثر هي بعد ذلك بأي عملية تحدث على ذلك المتغير. لكن هذا ينطبق فقط على أنواع البيانات البسيطة مثل **int**, **char**, **float** وغيرها. أما المصفوفات فتتغير قيمتها تبعاً لأي تعديل يحدث على خاناتها داخل الدالة. فيما يلي مثالانِ للشرح.

مثال:

اكتب برنامجاً يستقبل خمسة أعداد من المستخدم ويضعها في مصفوفة، ثم ينادي دالة تقوم بعكس المصفوفة، أي جعل العنصر الأول مكان العنصر الأخير والأخير مكان الأول، وهكذا.

الحل:

```
class ArrayTest
{
        public static void main(String args[])
        {
                int numbers[] = new int[5];
                System.out.println("Enter five numbers");
                for (int i = 0; i<= 4; i++)
                        numbers[i] = Input.readInt( );
                inverse(numbers, 5);
                System.out.println("The numbers after inversing:");
                for (int i = 0; i<= 4; i++)
                        System.out.println(numbers[i]);
        }
        public static void inverse(int array[ ], int length)
        {
                int index=length – 1, temp;
                for (int i = 0; i<= length/2; i++)
                {
                        temp = array[i];
                        array[i] = array[index];
                        array[index] = temp;
```

```
                index--;
            }
        }
}
```

تنادي الدالة main الدالة inverse، وترسل لهـا المصـفوفة numbers وطولهـا كوسيطين. تستقبل الدالة inverse المصفوفة في مصفوفة من نفس النوع تسـمى array، وتستقبل طولها في المتغير length. تقوم الدالة inverse بعكـس محتويـات المصـفوفة، وذلك كالآتي:

- المتغير index يحمل رقم الخانة الأخيرة في المصفوفة (في هذا المثال 4)
- عداد الحلقة i يبدأ من رقم الخانة الأولى في المصفوفة 0.
- يتم استبدال العددين في الخانتين رقم index ورقم i، وذلك بالاستعانة بالمتغير temp لا يـتم بالقيمـة array[i] كـي لا تضـيع بعـد تنفيـذ العبـارة:

```
array[i] = array[index];
```

- عدد التكرارات هو نصف عدد الخانات، لأننا عندما نسـتبدل النصـف الأول من الخانات مع النصف الثاني، تكون عملية عكس المصفوفة قد انتهت. وعند نهاية كـل تكرار تقل قيمة المتغير index بواحد، لأنه يحمل قيمة العنصر الأخير ثم العنصر قبـل الأخير ثم الذي قبله وهكذا.

بعد انتهاء تنفيذ الدالة inverse، يتم طباعة عناصر المصفوفة فـي الدالـة main لنلاحظ أن محتويات المصفوفة قد تأثرت بعمليات الاستبدال التي تمت في الدالة main.

مثال للتنفيذ:

```
Enter five numbers
6
7
10
5
8
The numbers after inversing:
8
5
```

10
7
6

من المثال السابق، نستنتج أن المصفوفات تتأثر عند تمريرها إلى دوال كوسائط متى ما تم إجراء تعديل على قيمها داخل تلك الدوال.

10. النـــداء الذاتي

خلال الأمثلة السابقة، رأينا كيف أن الدالة يمكن أن تنادي دالة أخرى بداخلها للاستفادة منها في تنفيذ عمليات معينة. يمكن للدالة كذلك أن تنادي نفسها لإعادة تنفيذ عباراتها والاستفادة منها في إكمال عملها. يعرف ذلك بالنداء الذاتي، ولتوضيحه أكثر سنستخدم المثال التالي.

مثال:

اكتب دالةً تسمى sum تستقبل عدداً صحيحاً وتقوم بإيجاد مجموع الأعداد من 1 إلى العدد المدخل باستخدام النداء الذاتي.

الحل:

مجموع العدد n = مجموع الأعداد من 1 إلى n

مجموع الأعداد من 1 إلى (n-1) + n = sum(n)

sum(n-1) + n = sum(n)

أي لحساب مجموع العدد n يمكن أن ننادي نفس الدالة sum لحساب مجموع n-1 ثم نضيف إليه n. ولحساب مجموع n-1 يمكن أن ننادي نفس الدالة sum لحساب

مجموع n-2 لنضيف إليه n-1 للحصول على مجموع n-1 وهكذا، حتى نصل إلى العـــدد 1، وعندها نعلم أن مجموع العدد 1 يساوي 1. يتم كتابة الدالة كالآتي:

```
public static int sum(int n)
{
        int summation =0;
        if(n ==1)
                summation = 1;
        else
                summation = n + sum(n – 1);
        return summation;
}
```

خطوات تنفيذ المثال:

نفرض أنه تم نداء الدالة لحساب مجموع4. يبدأ التنفيذ الأول للدالة sum حيث n لا يساوي 1 وعليه:

summation = 4 + sum(3);

حيث sum(3) عبارة عن نداء ذاتي، أي نداء لنفس الدالة sum. هذه المرة تم نداء الدالـــة sum بالعدد 3، فيبدأ التنفيذ الثاني للدالة، حيث n لا يساوي 1 وعليه:

summation = 3 + sum(2);

حيث sum(2) هو نداء الدالة sum أيضا، وتم نداؤها بالعدد 2، فيبدأ التنفيذ الثالث للدالـــة، حيث n لا يساوي 1 وعليه:

summation = 2 + sum(1);

sum(1) عبارة عن نداء ذاتي، ليبدأ التنفيذ الرابع للدالة sum، وحيـــث إن n يســـاوي 1 عليه:

summation = 1;

وهذه هي النقطة حيث تتوقف النداءات، وتعيد الدالة القيمة 1 بواســـطة العبـــارة **return**. يتم إعادة هذه القيمة إلى التنفيذ الذي نادى sum(1) وهو التنفيذ الثالث sum(2)، ليصبح:

summation = 2 + sum(1);

= 2 + 1 = 3;

ويكتمل حينها التنفيذ الثالث ويعيد القيمة 3 إلى التنفيذ الثاني للدالة sum(2)، ليصبح:

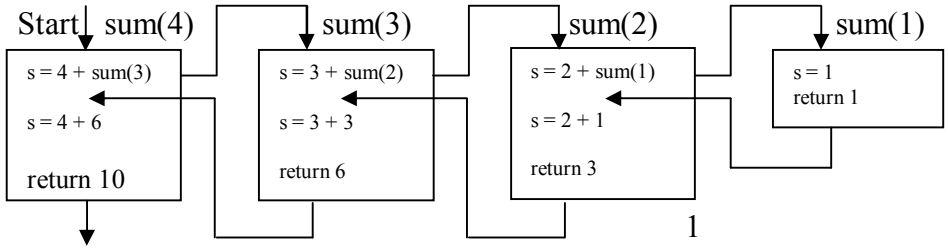
summation = 3 + sum(2);
= 3 + 3 = 6;

ليكتمل حساب sum(3)، ويعيد قيمته إلى التنفيذ الأول sum(4)، ليصبح:

summation = 4 + sum(3);
= 4 + 6 = 10;

هنا نكون قد حصلنا على مجموع العدد 4، وتكون جميع النداءات الذاتية قد أكملت عملها.

الشكل التالي يوضح انتقال التنفيذ بين النداءات الذاتية:



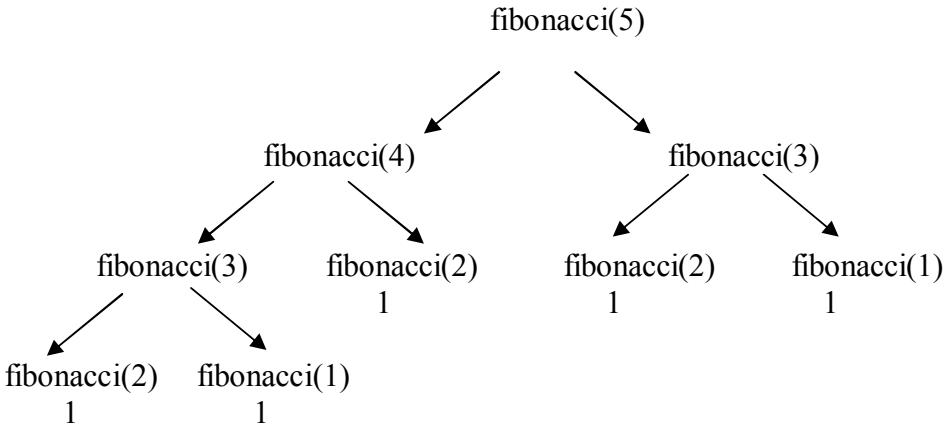
مثال: اكتب دالةً تستقبل رقم الحد، وتقوم بحساب قيمة الحد في سلسلة فيبوناشي باستخدام النداء الذاتي. قيمة الحد في السلسلة يساوي حاصل جمع الحدين السابقين، وكل من الحد الأول والحد الثاني يساوي 1.

الحل:

```
public static int fibonacci(int f)
{
        int fibo = 0;
        if ((f ==1) || (f == 2))
                fibo =1;
        else
                fibo = fibonacci(f – 1) + fibonacci(f – 2);
        return fibo;
```

}

خطوات نداءات تنفيذ المثال:



وعند تجميع القيم المعروفة نجد أن fibonacci(5) = 5، حيث إن سلسلة فيبوناشي هي:

1, 1, 2, 3, 5, 8, 13, … والحد الخامس قيمته 5 فعلاً.

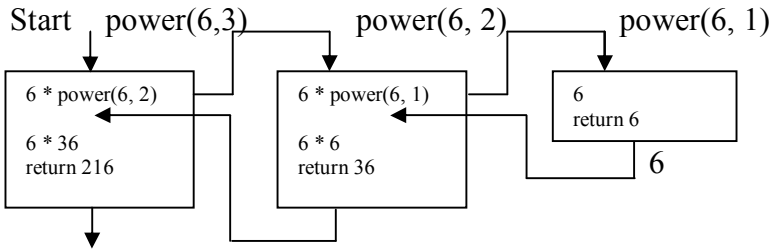
مثال:

باستخدام النداء الذاتي اكتب دالة لحساب x^n

الحل:

```
public static long power(int x, int n)
{
        long p = 1;
        if (n == 1)
                p = x;
        else
                p = x * power(x, n – 1);
        return p;
}
```

الشكل التالي يوضح انتقال التنفيذ بين النداءات الذاتية:



End 216 36

عند استخدام النداء الذاتي يجب الانتباه إلى ضرورة وجود شرط معين لإيقاف النداء الذاتي، وإلا ستتواصل النداءات لعدد لانهائي من المرات، وعندها لا يتوقف البرنامج عن التنفيذ، تماماً مثل ما قد يحدث عند تشغيل حلقة **while** شرطها دائماً متحقق. فمثلاً في دالة النداء الذاتي التالية:

```
public static int sum(int n)
{
        int s=0;
        s= s + sum(n – 1);
        return s;
}
```

لا توجد قيمة تجعل النداء الذاتي يتوقف، ولا يبقى شرط تنتهي عنده النداءات بمجرد تحققه. فمثل هذه الدالة لا ينتهي تنفيذها، ولا يتوقف البرنامج إلا إذا قطعناه قسراً.

جميع أمثلة النداء الذاتي التي تعرّضنا لها، كانت قد قابلتنا في دروس سابقة، وقد قمنا بحلها بدون استخدام النداء الذاتي. أي أن النداء الذاتي هو طريقة بديلة من طرق حل البرامج التي تحتوي على التكرار، ويمكن الاستغناء عنه باستخدام الحلقات التكرارية كما رأينا سابقاً.



عند استخدام النداء الذاتي يجب الاحتراس والتأكد من وجود شرط توقف النداءات. لكن الأفضل استبداله بالحلقات لأن تنفيذ البرنامج بالنداء الذاتي يستغرق زمناً أطول في التنفيذ ويستهلك ذاكرة أكبر من تنفيذ نفس البرنامج باستخدام الحلقات.

الخلاصة

تناولت الوحدة مميزات لغة جافا ونجد أن لغة جافا لغة قوية تحتوي على أدوات كثيرة تساعد في كتابة البرامج. كما تلافت عيوب كثير من اللغات قبلها، من أهم هذه العيوب إمكانية الوصول المباشر لمواقع الذاكرة الخاصة بالبرنامج والذي يؤدي إلى ضعف سرية المعلومات وسهولة تدميرها، كما تعتبر لغة جافا لغة برمجة بالكائنات ويعتبر هذا الصنف من لغات البرمجة من أوسعها انتشاراً وأكثرها استخدماً اليوم.

وتناولت الوحدة مكونات لغة جافا وهي الحروف التي تتركب منها هذه اللغة وهي: الأرقام، و الحروف، و الرموز والعلامات الخاصة، و العلامات الرياضية، و علامات الترقيم

وذكرنا ان الكلمات تتركب من هذه المكونات وفق الشروط التالية:

- ألا يبدأ الاسم برقم.
- ألا يحتوي على حرف خاص أو مسافة space.

كذلك تطرقنا في هذه الوحدة الى الكلمات المحجوزة وهي الكلمات التي عرفتها اللغة مسبقاً لتحمل معاني معينة عند ظهورها في البرنامج .

وذكرنا أن لغة جافا تعتبر (case sensitive)، وهذا التعبير يعني أنها تفرّق بين الحروف الكبيرة والصغيرة، يبدأ برنامج **جافا** بالكلمة المحجوزة **class** يلي هذه الكلمة الاسم الذي يختاره المبرمج لهذا الـــ **class**، و تكتب محتويات هذا الـــ **class** بين قوسين { }، الدالة الرئيسية main، هو الجزء الذي يبدأ منه تنفيذ البرنامج. أما الكلمات **public**, **static**, **void**, main جميعها كلمات محجوزة

محتويات الجزء main توضع بين قوسين { }، النص المراد ظهوره على الشاشة نضعه بين قوسين " " ويسمى نص text أو سلسلة حروف **String**. الأمر في لغة **جافا** ينتهي بفاصلة منقوطة (; semicolon)، العلامة // نضعها لكتابة التعليقات أو الملاحظات في البرنامج فقط للتوضيح، ويتم تجاهل كل ما يكون بعد العلامة // حتى

نهاية السطر. أما لكتابة التعليقات الطويلة أو التي تتكون من عدة أسطر فيمكن كتابتها بين العلامتين /* و */ أو /* و **/ .

كما تناولت الوحدة أنواع البيانات الاساسية في جافا وأوضحت أنه وعموماً لكي يتم الاحتفاظ بأي بيانات في الذاكرة نحتاج للمتغيرات variables، والثوابت constants، ويتم تعريف هذه المتغيرات والثوابت بحيث يمكن أن تحمل بيانات من أنواع معينة ، وفصلت الوحدة انواع البيانات في اللآتي :

الحرف (char) – العدد الصحيح integer type – العدد الحقيقي real type

النوع المنطقي Boolean type (boolean)

و لتعريف أي متغير في البرنامج، نكتب أولاً نوع البيانات التي سيحتويها هذا المتغير، ثم نكتب اسم المتغير– يمكن تعريف أكثرمن متغير مباشرة – لوضع قيمة ابتدائية في متغير، نحدد هذه القيمة مباشرةً عند تعريفه– لوضع قيمة معينة في متغير تستخدم العلامة = .

وبينت ان الدالة print تستقبل بين قوسيها نصاً (إذا كان هذا النص رسالة نصية أو متغير)، و لدمج نصين نستخدم العلامة +.

كما تناولت **الثوابت** وهو الذي لا يمكن تغيير قيمته في البرنامج، وتعربف الثوابت لايختلف عن تعريف المتغيرات إلا في الكلمة المحجوزة final .

تناولت الوحدة العمليات الرياضية وهي العمليات الرياضية الاربع إضافة الى عملية باقي القسمة، كما تناولت أيضاً كيفية كتابة عبارات هذه العمليات الرياضية بصورة مختصرة وقدمت اختصار للعبارتين :

(x = x + 1;) تكتب x ++ أو ++ x

(x = x - 1;) فتكتب x -- أو -- x

تناولت الوحدة قراءة البيانات من المستخدم في لغة جافا وقدمنا عدة دوال تستخدم لقراءة البيانات وكل دالة تقرأ قيمة بيانات من نوع معين وتخزنها في متغير من نفس النوع ومن هذه الدوال:

- الدالة: **System**.out.print( )
- الدالة: **System**.out.println( )

تستخدم هاتان الدالتان لطباعة المخرجات على الشاشة.

- الدالة: **Input**.readString( )

تقرأ قيمة **String** من المستخدم وتضعها في متغير من النوع **String**.

نداء الدالة: name=**Input**.readString( );

حيث name متغير معرف من النوع **String**.

- الدالة: **Input**.readChar( )

تستخدم لقراءة جرف من المستخدم وتضعه في متغير من النوع **char**.

نداء الدالة: key = **Input**.readChar( );

حيث key متغير معرف من النوع **char**.

- الدالة: **Input**.readInt( )

تقرأ عدد صحيح وتضعه في متغير من النوع **int**.

نداء الدالة: x = **Input**.readInt( )

حيث x متغير معرف من النوع **int**.

- الدالة: **Input**.readDouble( )

لقراءة عدد حقيقي تضعه في متغير من النوع **double**.

نداء الدالة: degree = **Input**.readDouble( )

حيث degree متغير النوع **double**.

- الدالة: **Input**.readBoolean( )

وهي دالة تنتظر من المستخدم إدخال قيمة **true** أو **false** وتخزنها في متغير من النوع **boolean**.

نداء الدالة: b= **Input**.readBoolean( )

حيث b هو متغير نوعه **boolean**.

تناولت الوحدة التعبيرات المنطقية والشروط ، والأدوات التي بها يمكن وضع الشروط

أو التعبيرات المنطقية: > ، =< ، < ، => ، == ، =!

كما تناولت عبارة الشرط If وأوضحنا الصيغة التي تأخذها هي :

1. **if** (condition)
2. statement;

كما تناولت الكلمة المحجوزة **else** وهي تلازم عبارة if معينة، وتأخذ الصيغة التالية:

```
if (condition)
        Statement1;
else
        Statement2;
```

كما تناولت الوحدة اختصار الاختبارات على المتغير وقدمت عبارة **switch**

وتأخذ عبارة **switch** الصيغة :

```
switch (variable)

{
        case value1: statement1;
                        break;
        case value2: ststment2;
                        break;
          :              :
        default:      statement;

}
```

كما تناولت الوحدة ثلاثة عبارات مختلفة للتكرار توفرها لغة جافا وهي :

- العبارة التكرارية **for**

وغالباً تأخذ الصيغة:

```
for(counter=initialValue;                    counter<=finalValue;
incrementExpression)
        Statement;
```

ويمكن حساب عدد مرات تكرار عمل الحلقة **for** من القانون الآتي:

$$\text{عدد التكرارات} = \left\lfloor 1 + \left(\frac{\text{القيمة النهائية} - \text{القيمة الابتدائية}}{\text{معدل زيادة العداد لكل تكرار}} \right) \right\rfloor$$

- **العبارة التكرارية while**

وتأخذ الصيغة التالية:

```
while(condition)
{
        statement(s);
}
```

- **العبارة التكرارية do - while**

وتأخذ الصيغة:

```
do
{
        statement(s);
}while (condition);
```

وبينت الوحدة تخزين مجموعات البيانات حيث يمكن تخزين مجموعة من البيانات بصورة تسهل التعامل معها والوصول إليها فيما يعرف بالمصفوفة Array، والمصفوفة هي عبارة عن صف من البيانات ذات علاقة ببعضها من نفس نوع البيانات .

كما تناولت الوحدة الدوال في جافا methods ، والدالة هي مجموعة من التعليمات التي تؤدي وظيفة معينة . وتناولت الوحدة النداء الذاتي وهو أن تنادي الدالة نفسها لإعادة تنفيذ عباراتها والاستفادة منها في إكمال عملها .

لمحة مسبقة عن الوحدة التالية

تبحث الوحدة التالية في البرمجة بالكائينات، حيث تعتبر لغة جافا أشهر لغات البرمجة بالكائينات، وسوف نتعرف على خصائص ومميزات البرمجة بهذا النوع من اللغات، كذلك تجد في هذه الوحدة واحدة من أهم المميزات التي توفرها لغات البرمجة بالكائنات وهي الوراثة.

إجابات التدريبات

تدريب (1)

```
1.      class MyCircle
2.      {
3.          public static void main (String args[])
4.          {
5.                  final double PI = 3.14;
6.                  double area = PI*6*6;
7.                  System.out.println (" The area is: " + area);
8.          }
9.      }
```

في السطر 5 قمنا بتعريف ثابت ليحمل قيمة الثابت PI باعتبارها قيمة لا تتغير. ثم عرفنا في السطر 6 المتغير area الذي سيحمل قيمة مساحة الدائرة بعد حسابها. وبعد ذلك تمت طباعة قيمة مساحة الدائرة مع رسالة مناسبة في السطر 7.

تدريب (2)

```
1.    class Resistance
2.    {
3.          public static void main (String args[])
4.          {
5.                  double R1 = 5;
6.                  double R2 = 2;
7.                  double R = (R1 + R2) / (R1 * R2);
8.                  System.out.println("Equivalent Resistance is: " + R);
9.          }
10.   }
```

تدريب (3)

```
class Calculate
{
        public static void main (String args[]);
        {
                System.out.println("Enter a number, please");
                int number1 = Input.readInt( );
                System.out.println("Enter another number, please");
                int number2 = Input.readInt( );
                System.out.println( number1 + "+" + number2 + "=" +
                (number1 + number2) );
                System.out.println( number1 + "-" + number2 + "=" +
                (number1 - number2) );
                System.out.println( number1 + "*" + number2 + "=" +
                (number1 * number2) );
                System.out.println( number1 + "/" + number2 + "=" +
                (number1 / number2) );
        }
}
```

نموذج (مختار) لتنفيذ البرنامج:

```
Enter a number, please
10
Enter another number, please
2
10+2=12
10-2=8
10*2=20
10/2=5
```

تدريب (4)

```
1.  System.out.println("Enter two numbers, please");
2.  int a = Input.readInt();
3.  int b = Input.readInt( );
4.  if (a>b)        //          الأكبر a
5.          System.out.println("a-b=" + (a-b));
6.  else if (b>a)   //          الأكبر b
```

```
7.          System.out.println("b-a=" + (b-a));
8.    else          //      العددان متساويان
9.          System.out.println("Numbers are equal");
```

إذا تحقق الشرط الأول في السطر 4، يقوم البرنامج بتجاهل جميع العبارات (6–9)، لأنها عبارات **else**. أما إذا لم يتحقق الشرط الأول، يتجاهل عبارة الطباعة في السطر 5 وينتقل إلى الاختبار التالي، إذا تحقق ينفذ الأمر في السطر 7 ويتجاهل بقية العبارات. أما إذا لم يتحقق، في هذه الحالة لم يكن a أكبر من b ولا b أكبر من a، إذن ليس هناك أي احتمال آخر سوى أن يكون a وb متساويين، وبالتالي لا نحتاج إلى شرط ثالث (مثلاً **if** a == b) لأن العبارة في السطر 9 لن تنفذ إلا في هذه الحالة فقط.

مثال للتنفيذ:

```
Enter two numbers, please
5
6
b – a = 1
```

تدريب (5)

```
System.out.println ("How mach is your money?");
double money = Input.readDouble( );
if (money <30000)
        System.out.println ("Your money is less than the charity amount");
else
{
        double charity = money/40;
        System.out.println ("Your charity is " + charity);
}
```

تدريب (6)

```
System.out.println("Insert A,B,C,D or F");
char rank= Input.readChar( );
switch (rank)
{
        case 'A': System.out.println("Excellent");
```

```
                break;
        case 'B': System.out.println("Very Good");
                  break;
        case 'C': System.out.println("Good");
                  break;
        case 'D': System.out.println("Pass");
                  break;
        case 'F': System.out.println("Fail");
                  break;
        default:  System.out.println("Wrong rank");
}
```

تدريب (7)

```
System.out.println("Enter a letter");
char letter= Input.readChar( );
switch (letter)
{
        case 'a': case 'e': case 'i': case 'o': case 'u':
                System.out.println("Vowel Letter");
                break;
        case 'x': case 'y': case 'z':
                System.out.println("Math Variable");
                break;
}
```

لاحظ في عبارة **switch** عدم وجود الجزء **default**، لأنه يمكن استخدامه أو تركه حسب الحاجة.

تدريب (8)

$$\frac{30-0+1}{3}=11 \text{ Iterations} =$$

تدريب (9)

نصمم حلقة تعمل 10 مرات، كل تكرار يحتوي على:

1. قراءة عدد
2. اختبار قيمة العدد:
   - إذا كان العدد فردياً تزيد قيمة عداد الأعداد الفردية بواحد.
   - إذا كان العدد زوجياً تزيد قيمة عداد الأعداد الزوجية بواحد.

   وعليه في هذا المثال نحتاج لعدادين.

```
int oddCounter=0, evenCounter=0;
int number;
for(int j=1; j<=10; j++)
{
        number=Input.readInt( );
        if(number%2 ==0)            //        العدد زوجي
                evenCounter ++;
        else                                  //        العدد فردي
                oddCounter ++;
}
System.out.println("Odd numbers: "+ oddCounter);
System.out.println("Even numbers: "+ evenCounter);
```

حل آخر:

العدد إما أن يكون زوجياً وإما أن يكون فردياً. يمكن أن نحسب عــدد الأعــداد الزوجية فقط. ونحن نعلم أن عدد الأعداد هو 10 فإذا كان عدد الأعداد الزوجية هو 6، فلا بد أن يكون عدد الأعداد الفردية هو 4 أي 10 ناقص عدد الأعداد الزوجية. لتتحول عبارة الطباعة إلى:

```
System.out.println("Odd numbers: "+ (10- oddCounter));
```

تدريب (10)

مضروب العدد 4 هو: 1*2*3*4

أي تكرار لعملية ضرب. العدد الذي يدخل في عملية الضرب متغير. يبدأ من 1 ويزداد بمعدل واحد حتى يصل إلى 4. ولذلك يمكن أن نجعل عداد الحلقة التي تقوم بتكرار الضرب يأخذ هذه القيم.

```
int factorial=1;
for(int i=1; i<=4; i++)
        factorial= factorial*i;
        // factorial *=i;          أو
```

يمكن تعديل الحلقة أعلاه لحساب مضروب n حيث n أي عدد صحيح موجب أكبر من 1:

```
int factorial=1;
for(int i=1; i<=n; i ++)
        factorial= factorial*i;
        // factorial *=i;          أو
```

هذه الحلقة تقوم بحساب حاصل ضرب الأعداد من 1 إلى العدد n، وهو يمثل مضروب العدد n. ونلاحظ أن أول تكرار للحلقة (عندما i=1) لا فائدة منه، لأن الضرب *1 لا يغير من قيمة المتغير factorial، ولذلك يمكن الاستغناء عن هذا التكرار لكي نوفر زمن تنفيذ هذا التكرار غير المؤثر لتصبح الصورة النهائية للحلقة:

```
int factorial=1;
int n=Input.readInt( );
for(int i=2; i<=n; i ++)
factorial *=i;
```

تدريب (11)

$6^{4=}$ 6*6*6*6

أي تكرار عملية الضرب * الأساس 4 مرات.

```
int power=1;
for(int i=1; i<=4; i++)
        power=power*6;
        // power*=6;   أو
```

يمكن تعديل الحلقة أعلاه لحساب x^n. وهي عبارة عن حلقة تكرر n مـــرة عمليـــة الضرب في الأساس x. سنقوم أولاً بقراءة قيمتي الأس والأساس.

```
System.out.println("Enter the base");
int x= Input.readInt( );
System.out.println("Enter the power");
int n= Input.readInt( );
int power=1;
for(int i=1; i<=n; i++) // مرة n
        power*=x; //  الأساس* الضرب
System.out.println("result is: " + power);
```

تدريب (12)

الشكل توجد به أعداد من الواضح أنه لا علاقة بينها وبين رقم الصف. لكنها أعداد متتالية تبدأ من 1. عليه يمكن تعريف متغير يبدأ من 1 ويزداد كلما تمت طباعته. عمليـــة الطباعة هذه تتم في حلقة تبدأ من 1 وتنتهي في قيمة عداد الحلقة الخارجية. هـــذه الحلقـــة إضافة إلى عملية الطباعة تحتوي على أمر لزيادة قيمة المتغير ليتم طباعة قيمة جديدة فـــي التكرار التالي ليكون البرنامج على الصورة:

```
int number =1;
for(int i=1; i<=3; i++)
{
        for(int k=1; k<= (3 – i); k++)
                System.out.print(" ");
        for(int j=1; j<= i; j++)
                System.out.print(number ++);
        System.out.print("\n");
}
```

تدريب (13)

نصمم حلقة بها عبارة لقسمة العدد على 10 كل تكـــرار. حتـــى لا تتبقـــى خانات صحيحة في العدد أو (n/10 == 0). وعدد خانات الرقم هي عدد تكرارات الحلقة.

```
System.out.println("Enter an integer");
int n=Input.readInt( );
int counter=1;
while (n/10 !=0)
{
        n/=10;          // n= n/10;              أو
        counter++;
}
System.out.println("This number has " + counter + " digits");
```

إذا لم تعمل هذه الحلقة لأي عدد من المرات، هذا يعني أن العدد مكون من خانة واحدة، ولذلك وضعنا القيمة الابتدائية 1 للعداد.

تدريب (14)

نقوم بتعريف متغير للاحتفاظ بالعدد الأكبر. وبعد كل عملية قراءة نختبر قيمتـــه مع قيمة المتغير الذي تمت قراءته، وإذا كانت أكبر من قيمته نضع القيمة الأكبر في هذا المتغير.

```
int number, large=0;
do
{
        System.out.println("Enter a number");
        number =Input.readInt( );
        if (number>large)
                large = number;
}while(number!=0);
System.out.println("The largest number is: " + large);
```

تدريب (15)

نصمم حلقة تكرر قراءة العدد مادام أنه لا يساوي قيمة العدد المخزن بالبرنامج (على شكل ثابت). ونقوم بتعريف عداد ليحسب عدد المرات التي تم فيها تكرار عبارات الحلقة، حيث تمثل عدد مرات التخمين.

```
int count=0, number;
final int correct=6;
do
{
        System.out.println("Try to guess my number");
        number= Input.readInt( );
        count++;
}while(number != correct);
System.out.println("My number was " + correct + " you guessed " + count +
" times");
```

تدريب (16)

```
char lang[ ]= {'a', 'r', 'a', 'b', 'i', 'c'};
```

لنقوم بعملية الإزاحة لمرة واحدة نعرف متغير:

```
char temp = lang[0];
```

ونحتفظ فيه بالعنصر الأول أو أقصى يسار المصفوفة. بعد ذلك نقوم بإزاحة عناصر المصفوفة خطوة إلى اليسار.

```
for (int i=0; i<=4; i++)
lang[i] = lang[i+1];
```

ونلاحظ أن قيمة i هنا تصل إلى 4 وليس 5 لأن العنصر الأخير نحصل عليه من المتغير temp:

```
lang[5] = temp;
```

عند طباعة المصفوفة الناتجة نجد أن الحرف 'a' انتقل من بداية المصفوفة إلى نهايتها، وجميع الحروف المتبقية انتقلت خانة إلى اليمين. يمكننا الآن تكرار عبارات الإزاحة أعلاه ست مرات لتكرار عملية الإزاحة وطباعة المصفوفة الناتجة إلى أن تعود المصفوفة إلى حالتها الأولى.

```
char lang[ ]={'a', 'r', 'a', 'b', 'i', 'c'};
char temp;
for(int k=0; k<=5; k++)
{
        temp = lang[0];
        for(int i=0; i<=4; i++)
                lang[i] = lang[i+1];
        lang[5] = temp;
        for(int i=0; i<=5; i++)          //      طباعة المصفوفة
                System.out.print(lang[i]);
        System.out.print("\n");
}
```

مخرجات البرنامج:

```
rabica
abicar
bicara
icarab
carabi
arabic
```

مسرد المصطلحات

Reserved Words :

هي كلمات عرفتها اللغة مسبقاً لتحمل معاني معينة عند ظهورها في البرنامج، ولا يجوز استخدامها من قبل المبرمج إلا لما خصصت له.

Case Sensitive :

هو تعبير يعني أن لعة البرمجة تفرق بين الحروف الكبيرة Capital والصغيرة Small .

Unicode الشفرة الموحدة:

هي عبارة عن جدول ضخم يخصص رقماً فريداً لكل حرف في جميع اللغات العالمية، وذلك بغض النظر عن نوع الحاسوب أو البرامج المستخدمة. وقد تم تبني مواصفات الشفرة العالمية من قبل قادة صناع أنظمة الحاسوب في العالم، لتتلاءم معها جميعاً. وتشمل الشفرة جميع حروف اللغات العالمية واللهجات المحلية والرموز العلمية في جميع هذه اللغات.

Methods :

هي الدوال في لغة جافا، و الدالة هي مجموعة من التعليمات التي تؤدي وظيفة معينة، يتم نداؤها خلال البرنامج عند الحاجة بواسطة اسمها.

Array المصفوفة :

المصفوفة عبارة عن صف من البيانات ذات علاقة ببعضها من نفس نوع البيانات، يكون للمصفوفة اسم واحد وعدد من الحجرات توضع بها البيانات.

المراجع

1. Y. D. Liang
   Introduction to Java Programming, Third Edition
   ISBN: 0-13-031997-X
   Prentice Hall 2001
2. Deitel and Deitel
   Java How to Program, Third Edition
   ISBN: 0-13-012507-5
   Deitel 2000
3. Deitel and Deitel
   Java How to Program, Fifth Edition
   ISBN: 0-13-101621-0
   Deitel 2003
4. Stephan J. Chapman
   Introduction to Java
   ISBN: 0-13-919416-9
   Prentice Hall 1999
5. نعمة عزالدين محمد عثمان

   البرمجة بلغة جافا للمبتدئين

   الإصدارة الثانية 2007م

الوحدة الثانية
البرمجة بالكائنات



محتويات الوحدة

الموضوع	الصفحة
المقدمة	134
تمهيد	134
أهداف الوحدة	136
1 . البرمجة بالكائنات Object Oriented Programming	137
1 .1 التعامل مع البيانات وفق قوانين وشروط الصنف	139
2 . استخدام المشيد	141
3 . استخدام المؤشر "this" reference	144
4 . العلاقة بين الكائنات composition	145
5 . تعريف متغير مشترك بين جميع الكائنات المشتقة من class محـــدد باستخدام الكلمة المحجوزة static	148
6 . الوراثة Inheritance	150
6 . 1 أمثلة على الوراثة	151
6 .2 كيف يستخدم الوارث متغيرات ودوال ومشيدات الفئة الأب	152
6 .3 تعريف المتغيرات والدوال	155
6 .4 استخدامات الكلمة المحجوزة final	155
7 . منع إنشاء الكائنات من نوع فئة محددة	156
8 . استخدام الواجهات Interface	159
9 . استخدام الواجهات كواجهة بين الكائن والمستخدم	162
9 .1 خاصية تعدد الأشكال Polymorphism	165
10 . مجموعات الفئات Packages	169
الخلاصة	171

لمحة مسبقة عن الوحدة الدراسية التالية	174
مسرد المصطلحات	175
المراجع	176

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك في الوحدة الثانية من مقرر " **البرمجة بلغة جافا**" والتي تبحث في البرمجة بالكائنات، حيث تعتبر لغة جافا أشهر لغات البرمجة بالكائنات، وفي هذه الوحدة سوف نتعرف على خصائص ومميزات البرمجة بهذا النوع من اللغات.

في القسم **الأول** من الوحدة نتناول البرنامج 2 -1-1 الذي يحتوي على بيانات من نوع String . كما يتناول القسم البرنامج 2- 1 -2 الذي يمتاز عن البرنامج السابق بأن بيانات الصنف في هذا البرنامج لها خصوصية وسرية لعدم إمكانية الوصول إليها إلا عن طريق الدالة set والدالة get ، وهذا يضمن أن كل من يتعامل مع البيانات سيتعامل معها وفق قوانين وشروط الصنف. القسم **الثاني** يتناول البرنامج 2 -1 -3 وهو مثال لاستخدام المشيد. القسم **الثالث** وفيه نجد البرنامج 2 – 1 – 4 حيث نجد أن كل class يستطيع أن يشير إلى محتوياته من متغيرات ودوال باستخدام المؤشر "this" reference وأيضاً نجد كيفية كتابة البرنامج السابق باستخدام مشيدات متعددة وباستخدام المؤشر this. وفي القسم **الرابع** نتناول العلاقة بين الكائنات أو ما يعرف بالـ composition وذلك من خلال البرنامج 2 – 1 – 5 وهو مثال يوضح علاقة الـ composition بين كائنين . في القسم **الخامس** نتعرف كيف يمكن تعريف متغير مشترك بين جميع الكائنات المشتقة من class محدداً باستخدام الكلمة المحجوزة static تسبق تعريف المتغير. يكون المتغير من هذا النوع موجوداً في الذاكرة حتى إذا لم يتم إنشاء أي كائن، ولا يتم حجز مساحة خاصة بهذا المتغير لكل كائن يتم إنشاؤه كما يحدث في حالة المتغيرات العادية، بل تتخاطب جميع هذه الكائنات مع متغير واحد مشترك بينها. القسم **السادس** يبحث في واحدة من أهم المميزات التي توفرها لغات البرمجة بالكائنات

وهي الوراثة، في هذا القسم نقدم أمثلة للوراثة ، كما نتناول برنامج 2 – 2 – 1 والـذي يشرح كيف يستخدم الوارث متغيرات ودوال ومشيدات الفئة الأب ، وأيضاً نجد ملخصـاً لطرق ثلاث لتعريف المتغيرات والدوال وشرحاً لكل معرف منهم في مفهوم الوراثة ... وأيضاً نجد استخدامات الكلمة المحجوزة final والتي يمكن من خلالها التحكم في توريث الفئات والدوال. أما القسم **السابع** فنتناول فيه كيفية منع إنشاء الكائنات مـن نـوع فئـة محددة وذلك بتعريفها من نوع abstract، ومن خلال برنامج 2 – 2 – 2 نوضح كيفيـة تعريف الفئات والدوال من النوع abstract. أما القسم **الثـامن** فنتنـاول فيـه اسـتخدام الواجهات interface باعتباره حلاً وضعته لغة جافا لتلافي عيب الوراثة الوحيدة المتمثل في عدم التمكن من وراثة خصائص معرفة في عدد من الفئات، ومن خـلال البرنـامج 2 –2 –3 نتناول تعريفاً لواجهة وكيفية استخدام الفئة لهذه الواجهة، وأيضاً كيفية تعريف عدد من الواجهات بالفئة الواحدة، وتعريف جميع دوال الواجهات داخل الفئة. أما القسـم **التاسع** فنوضح فيه كيف أن مفهوم الواجهات لا يقتصر على تعريـف أسـماء الـدوال والثوابت لاستخدامها داخل الفئة فحسب بل تستخدم كواجهة بين الكائن ومسـتخدم هـذا الكائن فتحدد له الجزء المصرح له باستخدامه وتمنعه من الوصول إلى بقيـة محتويـات الكائن، ويتم توضيح هذا المفهوم من خلال البرنامـج 2 – 2 – 4. كما يتنـاول هـذا القسم من الوحدة أيضا خاصية تعدد الأشكال polymorphism. أما القسم **العاشر** فيتنـاول مجموعات الفئات.

أهداف الوحدة



عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

- **تذكر** المبادئ التي تقوم عليها البرمجة بالكائنات.
- **تعرف** الصنف الـــ class .
- **تصمم** class.
- **تستخدم** الـــ classلاشتقاق الكائنات والتعامل معها.
- **تعدد** طرق تعريف المتغيرات والدوال في جافا
- **تشرح** طرق تعريف المتغيرات والدوال في مفهوم الوراثة.
- **تستخدم** جميع الخصائص والدوال والمشيدات المعرفة بالفئة الأب.
- **تعيد** تعريف الدوال لاستخدامها بصورة أفضل (overriding).
- **تضيف** مكونات جديدة على المكونات الموروثة.

1. البرمجة بالكائنات Object Oriented Programming

عزيزي الدارس،

تعتبر لغة جافا من أشهر لغات البرمجة بالكائنات Object Oriented Programming Languages وفي هذا الفصل نتعرف على خصائص ومميزات البرمجة بهذا النوع من اللغات، واختلافها عن البرامج التي تناولناها خلال الفصول السابقة. تقوم البرمجة بالكائنات على مبدأ أن كل فكرة أو موضوع في النظام المعين هو عبارة عن كائن object له صفات properties وسلوك behavior يظهر بها في النظام ويتفاعل عن طريقها مع الكائنات الأخرى في النظام. وتقوم على مبدا أن النظام هو مجموعة من الكائنات التي تحتوي على صفاتها الخاصة والتي لا تسمح الكائنات الأخرى بالوصول إليها. وهذه الكائنات تتفاعل مع بعضها البعض بواسطة طرق محددة سلفا وهي الدوال الخاصة بالكائن.

الصنف class هو عبارة عن هيكل برمجي يحتوي على بيانات attributes ودوال methods تصف معا الشكل الذي ستكون عليه الكائنات عند تشغيل البرنامج. هذا يعني أن الـ class يمثل قالباً تصنع منه الكائنات في البرنامج، أي أنه يتم تصميم الصنف مرة واحدة ثم اشتقاق اي عدد من الكائنات من هذا الصنف. يمكن أن يتم إنشاء الكائنات داخل الدالة الرئيسية main أو بداخل كائن آخر إذا كان يتعامل معه. ولأن لغة جافا لا تسمح بتنفيذ أي برنامج إلا إذا احتوى على class، كان من اللازم تعريف class لكل برنامج قمنا بتنفيذه من قبل رغم أننا لم نستخدمه بالطريقة القياسية، والأمر كذلك بالنسبة إلى الدالة الرئيسية main والتي يبدأ منها التنفيذ. وفيما يلي مثال لصنف وكيفية استخدامه لاشتقاق عدد من الكائنات والتعامل معها.

```
public  class Student {
        public string name;
        public void printName ( ) {
                System.out.printIn("Student name: " + name);
        }
```

```
        public static void main(string args[ ] ) {
                Student s1 = new Student( );
                s1.name = "safa";
                s1.printName( );
                Student s2 = new Student( );
                s2.name = "Ali";
                s2.printName( );
        }
}
```

برنامج 2-1-1

يعرف هذا البرنامج الصنف Student والذي يحتوي على بيانات من نوع String هي اسم الطالب، وله دالة واحدة لطباعة هذا الاسم على الشاشة. الدالة الرئيسية main لا علاقة لها بالصنف Student، ولكن يجب أن يتم تعريفها بداخل صنف محدد حسب شروط لغة جافا. يتم إنشاء كائنين داخل الدالة main وذلك عن طريق كتابة اسم الصنف ثم اسم الكائن الذي نرغب في انشائه – وهو هنا s1 و s2 – ثم علامة = والكلمة المحجوزة new يتبعها اسم الصنف كما هو موضح. تقوم الكلمة المحجوزة new بحجز موقع جديد بالذاكرة بالحجم الذي يحتاجه الكائن لتخزين بياناته ودواله، ويشار لهذا الموقع في الذاكرة باسم الكائن لنستطيع التعامل مع هذا الموقع فيما بعد. باستخدام اسم الكائن متبوعاً بنقطة نستطيع الوصول إلى محتويات الكائن من بيانات ودوال، حيث وضعنا قيمة في المتغير name وقمنا بتفيذ الدالة printName لكل كائن، وتكون مخرجات البرنامج هي:

```
Student name: Safa
Student name: Ali
```

أسئلة تقويم ذاتي



1- وضح المبدأ الذي تقوم عليه البرمجة بالكائنات؟
2- عرف الصنف class؟

1 .1 التعامل مع البيانات وفق قوانين وشروط الصنف

عزيزي الدارس،

الكلمة المحجوزة public قبل اسم الصنف أو المتغير أو الدالة تعني أنَّه عام الاستخدام، ويمكن لجميع البرامج الوصول واستخدام هذا المعرف. يسمح وجود كلمة public قبل اسم الصنف للبرامج الأخرى والدالة الرئيسية باستخدام الصنف ومن ثمّ اشتقاق كائنات منه، ووجودها قبل اسم الدالة يسمح للكائنات بمناداة الدالة وتنفيذها. أما كتابة الكلمة المحجوزة public قبل تعريف البيانات فإنّه يخرق المبدأ الرئيسي للكائنات وهو خصوصية بيانات الكائن وعدم السماح للكائنات والبرامج الأخرى بالاطلاع عليها أو تعديلها مباشرة، ولذلك نستخدم الكلمة المحجوزة private عادة مع بيانات الصنف لتمنع الوصول إليها إلا من داخل الصنف. لذلك الأفضل تعديل تعريف المتغير name إلى:

```
private String name;
```

وفي هذه الحالة فإنَّ عبارة s1.name أو s2.name ينتج عنها خطأ. لكن رغم أنَّ اسم الطالب يعتبر من خصوصيات الطالب، إلا أننا نرغب في أن نتمكن من قراءة أو كتابة أو تعديل اسم الطالب في البرنامج، ولذلك نقوم بتصميم دالة مختصة بوضع قيمة في المتغير الخاص، وتعرف عادة بدالة get، وتصميم دالة أخرى مختصة بقراءة وارجاع قيمة المتغير الخاص، وتعرف عادة بدالة set، وهذا يعني تعريف دالتين مقابل كل متغير خاص داخل الصنف. على هذا الأساس يمكن تعديل البرنامج السابق ليصبح كالتالي:

```
public class Student {
    private String name;
    public void setName(String n) {
        name = n;
    }
```

```
    public String getName( ) {
        return name;
    }
    public void printName( ) {
        System.out.println ("Student name: " + name);
}
    public static void main(String args[ ]) {
        Student s1 = new Student( );
        s1.setName("Safa");
        s1.printName( );
        Student s2 = new student( );
        s2.setName("Ali");
        s2.printName( );
    }
}
```

برنامج 2-1-2

يقوم هذا البرنامج بنفس وظيفة البرنامج السابق، ولهما نفس المخرجات، ولكن الفرق بينهما أنّ لبيانات الصنف في البرنامج الثاني خصوصية وسرية لعدم إمكانية الوصول إليها إلا عن طريق الدالة set والدالة get، وهذا يضمن أن كل من يتعامل مع البيانات سيتعامل معها وفق قوانين وشروط الصنف. فمثلاً في البرنامج الأول كان يمكن أن نكتب فيه:

```
s1.name = "123";
```

وهي قيمة غير منطقية لاسم الطالب، ولكن الصنف ليس لديه أي تحكم على القيم القادمة من البرامج الأخرى. لكن في البرنامج الثاني يمكن أن نضع جميع الشروط والاختبارات على صحة وملاءمة البيانات التي يحاول برنامج آخر إسنادها إلى بيانات الصنف داخل الدالة set، وفي حالة عدم توفر الشروط تقوم الدالة بإجراء اللازم كوضع قيمة معينة في المتغير أو إظهار رسالة خطأ أو غيره.

أسئلة تقويم ذاتي



1- ماذا تعني الكلمة المحجوزة public قبل اسم الصنف أو المتغير أو الدالة ؟

2- ما وظيفة كل من الدالتين get و set في البرنامج 2 -1 -2 ؟

3- وضح الفرق بين البرنامج 2 -1 -1 والبرنامج 2 – 1 – 2 ؟

2 . استخدام المشيد

يمتلك كل class دالة خاصة تعرف باسم المشيد constructor، وهي دالة يتم تنفيذها تلقائياً عند إنشاء كائن من نوع الصنف. تستخدم هذه الدالة لإجراء العمليات التي نرغب في تنفيذها ابتدائياً لحظة إنشاء الكائن وقبل تعامل أي جهة مع هذا الكائن، ومثال لهذه العمليات وضع القيم الابتدائية للمتغيرات وإنشاء الكائنات الأخرى التى يتعامل معها هذا الكائن. يتم تعريف هذه الدالة بصيغة معينة، حيث يأخذ تعريف أي constructor الشكل:

```
public ClassName(parameter list) {
        // constructor code
}
```

ويمكن تمرير وسائط parameters إلى المشيد لاستخدامها في تنفيذ العمليات المعينة، كما يمكن أن يخلو تعريفه من الوسائط. كما علمنا سابقاً، لإنشاء كائن محدد يتم استخدام الصيغة:

```
ClassName objectName = new ClassName( );
```

حيث تقوم الكلمة المحجوزة new بحجز الخانات اللازمة لتخزين الكائن في الذاكرة، أما الجزء ClassName( ) فيمثل نداء المشيد. وعن انتهاء تنفيذ هذه العبارة يكون قد تم حجز مكان للكائن في الذاكرة وتحديد اسم معين له يستخدم للتعامل معه، وكذلك تم تنفيذ جميع العمليات الأولية داخل الكائن. والبرنامج التالي هو مثال لاستخدام المشيد.

```
public class MyCircle {
    private double radius;
    public MyCircle( ) {   // constructor
          radius = 1 ;
    }
    public void setRadius(double   r) {
           radius = r ;
    }
    public double getRadius( ) {
          return  radius;
    }
    public double calculateArea( ) {
          return  3.14 * radius * radius;
    }
    public static void main(String args[ ]) {
         MyCircle circle = new MyCircle( );
         System.out.println("Area = " + circle.calculateArea());
circle.setRadius(4);
         System.out.println("Area = " + circle.calculateArea());
   }
}
```

برنامج 2-1-3

عند إنشاء الكائن بالاسم circle، فإنَّ أول ما يتم تنفيذه هو المشيد، حيث يتم تحديد قيمة إبتدائية لنصف القطر هي 1. وتظل هذه القيمة ثابتة إذا لم يتم استخدام الدالة setRadius لتغييرها، لذلك تستخدم القيمة 1 للمتغير radius عند حساب مساحة الدائرة في المرة الأولى. ويتم تعديلها بعد ذلك إلى القيمة 4 لينعكس ذلك على مساحة الدائرة عند حسابها فى المرة الثانية، لتكون مخرجات البرنامج هي:

```
3.14
50.24
```

يمكن تعديل المشيد بحيث يتم تمرير القيم الابتدائية لمتغيرات الـ class من الدالة الرئيسية، وتوضع هذه القيم بين قوسي المشيد عند ندائه لحظة إنشاء الكائن كالآتي:

```
MyCircle  circle = new MyCircle(10);
```

ويستقبل المشيد هذه القيمة بعد تعديل تعريفه كالآتي:

```
public MyCircle(double  r) {
```

```
        radius = r ;
}
```

كما يمكن أن يحتوي نفس الـ class على أكثر من مشيد واحد تختلف في عدد أو نوع بيانات الوسائط فيما يعرف بالـ constructor overloading بحيث يتم تنفيذ المشيد المناسب حسب عدد ونوع بيانات الوسائط الموجودة في نداء المشيد كما سنرى في تعديل المثال السابق.

يحتوي المشيد الذي تمَّ تعديله أعلاه على وسيط واحد من نوع double اسمه r، ونعلم أنّ من حق أي دالة أن تسمي متغيراتها بأى اسم. في مفهوم الفئات، إنّ أى متغير معرف في الفئة (خارج الدوال) يعبر عن خاصية من خواص الفئة ويكون معروف في أي منطقة داخل الفئة ويمكن الوصول إليه من داخل جميع دوال الفئة، لذلك تكون عبارة radius = r مفهومة لدى المشيد. أما المتغيرات المحلية المعرفة داخل الدوال فهي تخص الدالة ولا تكون معروفة خارجها مثل المتغير r. كان من الممكن أن نسمي المتغير r بالاسم radius ليكون هناك متغيرين بنفس الاسم داخل الفئة، أحدهما هو المتغير المعروف لدى جميع الدوال ويمكن التخاطب معه من داخل الدوال الأخرى مثل calculateArea، والآخر هو الخاص بالمشيد ولا يمكن التعامل معه من داخل المشيد. لكن أصبح لدينا متغيران بنفس الاسم يمكن التعامل معهما من داخل المشيد، وعند ظهور عبارة مثل: radius = 1 داخل المشيد، يكون المقصود هو المتغير الخاص بالمشيد لكونه الأقرب. لذلك نرغب في طريقة نصل بها إلى المتغير radius الخاص بالفئة من داخل المشيد.

أسئلة تقويم ذاتي



1- يمتلك كل class دالة خاصة تعرف باسم المشيد constructor. ما استخدامات هذه الدالة ؟

2- ما صيغة التعريف التي يأخذها أي constructor ؟

3 . استخدام المؤشر "this" reference

يستطيع كل class أن يشير إلى محتوياته من متغيرات ودوال باستخدام المؤشر "this" reference وخير مثال لذلك المشيد في المثال أعلاه:

```
public MyCircle(double  radius) {
        this.radius = radius;
}
```

فهذه العبارة تضع قيمة المتغير radius المعرف بالمشيد بالمتغير radius الخاص بالـ class، والذي يتم الوصول إليه عن طريق المؤشر this، ويمكن استخدامه حتى إذا لم يكن هناك متغير آخر يحمل نفس الاسم، فمثلاً يمكن أن تكون محتويات الدالة calculateArea هي:

```
return  3.14 * this.radius * this.radius;
```

رغم أنَّه لا يوجد متغير بداخل الدالة calculateArea بالاسم radius. يمكن إعادة كتابة البرنامج السابق باستخدام مشيدات متعددة وباستخدام المؤشر this.

```
public class MyCircle {
      private double radius;
     public MyCircle( ) {
              this.radius = 1 ;
    }
   public MyCircle(double  radius) {
              this.radius = radius;
  }
  public void setRadius(double  radius) {
              this.radius = radius;
  }
  public double getRadius( ) {
              return  this.radius;
  }
public double calculateArea( ) {
              return  3.14*  this.radius* this.radius;
  }
public static void main(String args[ ]) {
      MyCircle  circle = new MyCircle( );
      System.out.println("Area = " + circle.calculateArea( ));
```

```
    MyCircle  circle2 = new MyCircle( 3 );
    System.out.println("Area = " + circle2.calculateArea(  ));
  }
}
```

برنامج 2-1-4

عند تعريف الكائن circle يتم نداء المشيد الأول لأنَّه لم يتم تمرير وسائط فيتم تنفيذ المشيد الذي لا يستقبل وسائط. أما عند تعريف الكائن circle2 فيحتوي نداء المشيد على قيمة عددية، ولذلك يتم تنفيذ المشيد الذي يستقبل قيمة من نوع double، لتكون مخرجات البرنامج هي:

```
3.14
28.26
```

لاحظ أنَّ الفرق بين هذا المثال والمثال السابق هو أن هذا المثال ينشئ كائنين من نوع MyCircle بينما يغير المثال السابق قيمة نصف القطر لنفس الكائن.

4 . العلاقة بين الكائنات composition

يمكن للكائن أن يمتلك إلى جانب المتغيرات والدوال مؤشرات لكائنات أخرى إذا كان يتعامل معه، مثلاً يمكن أن تكون الشركة عبارة عن كائن يحتوي على كائنات أخرى مثل الموظفين والسيارات وغيرها. والموظف الذي هو عبارة عن كائن له أيضاً كائنات أخرى مثل المنزل والهاتف وغيرها. تعرف هذه العلاقة بين الكائنات بالـ composition، وتكون العلاقة بين كائنين من نوع composition إذا استطعنا وضع كلمة (له/لها) بين الكائنين وكانت الجملة منطقية، مثلاً: الشركة لها موظف، والموظف له هاتف، لأن الأساس في وجود مؤشر لكائن بداخل كائن آخر هو أن يملكه. وفيما يلي مثال يوضح علاقة الـ composition بين كائنين:

```
public class Employee {
      private string name ;
      private int roomNumber;
      public Employee(String  name) {
            this.name = name;
  }
```

```
   public Employee (String  name, int roomNumber) {
                this.name = name;
               this.roomNumber = roomNumber;
  }
       public void displayEmployeeDetail (  ) {
               System.out.println ("Employee name: " + this.name                + " ,
room number: " + this.roomNumber);
      }
}
public class Department{
       private String name;
       private Employee  employee;
       public Department(String  name) {
                this.name = name;
      }
      public void createEmp(String empName, int  empRoomNumber ) {
          employee = new Employee(empName, empRoomNumber);
}
public void showEmployee(  ) {
      employee.display EmployeeDetails(  );
}
public static void main(String args [  ]) {
    Department department = new  Department("Computing");
                String   name = "Ahmad";
                int number = 1;
               department.createEmp(name,  number);
               department.showEmployee(  );
       }
}
```

برنامج 2-1-5

يحتوي هذا المثال لأول مرة على أكثر من class واحد، يمكن حفظ كل class منهما في ملف مستقل Department.java وEmployee.java، أو يمكن أن يحفظا في ملف واحد بشرط ألا يحتوي الملف الواحد على أكثر من class مسبوق بـ public، بل يجب أن يكون به class واحد public هو الـ class الذي يبدأ منه تنفيذ البرنامج، أي الذي يحتوي على الدالة الرئيسية main، وهو في هذا المثال Department. إذن لحفظ

Departmentو Employee في نفس الملف، يجب إعادة تعريف class Employee ليصبح:

class Employee {…

كما كان من الممكن أن تكتب الدالة الرئيسية main داخل class Employee، وعندها يحتفظ Employee بتعريفه public، ولكن يتم حذفها من أمام class Department.

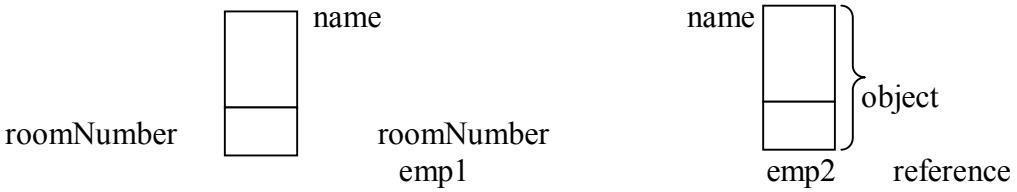
يبدأ تنفيذ البرنامج من الدالة main حيث يتم تعريف قسم عن طريق إنشاء كائن من النوع Department مع تحديد اسمه. يتم بعدها إنشاء موظف، لكنه ليس موظفاً مستقلاً بل موظف معرف بداخل القسم ولا يمكن الوصول إليه والتعامل معه إلا عبر القسم. لذلك لا يتم إنشاؤه باستخدام عبارة مثل:

Employee employee = new Employee ("Ahmad", 1);

لأنّه لا يمثل الموظف الذي ينتمي إلى القسم، وهو الذي نرغب في تعريفه. يتم الوصول إلى هذا الموظف عن طريق الدالة createEmployee والتي تستقبل البيانات الضرورية لإنشاء الموظف، والتي تستخدم لتعريف الموظف الذي يتم إنشاؤه من داخل الدالة createEmployee. رغم أن هذا الموظف الخاص بالقسم له دوال public مثل Details displayEmployeeإلا أننا لا يمكن أن نناديها مباشرة باستخدام اسم الكائن employee لأنه عبارة عن بيانات private تخص class Department، ولذلك نستخدم دوال القسم للوصول إلى الموظف بطريقة غير مباشرة. ومخرجات البرنامج هي:

Employee name: Ahmed, room number: 1

عند إنشاء كائن معين، يتم حجز الخانات اللازمة لتخزين جميع المتغيرات الخاصة بالكائن. وإذا كان هناك عدد من الكائنات يكون لكل كائن منها متغيراته الخاصة. فمثلاً إذا كان لدينا كائنان emp2 و emp1 من النوع Employee كما هو معرف في المثال السابق، يتم حجز خانات في الذاكرة كالآتي:



ولذلك فإن قيمة المتغير emp1.name مخزنة في موقع لا علاقة له بالمتغير emp2.name. تسمى مواقع الذاكرة المحجوزة بالكائن object، بينما يعرف اسم الكائن بالمؤشر أو المرجع reference.

5 . تعريف متغير مشترك بين جميع الكائنات المشتقة من class محدد باستخدام الكلمة المحجوزة static

في بعض الأحيان يتطلب عمل البرنامج أن يكون هناك متغير مشترك لجميع الكائنات المشتقة من class محدد، مثلاً: إذا لم يكن البرنامج يسمح بوجود أكثر من خمسة موظفين فى القسم الواحد، سنحتاج لمعرفة عدد الكائنات التي تمّ إنشاؤها من class Employee. يمكن تعريف متغير مشترك بين جميع الكائنات المشتقة من class محدد باستخدام الكلمة المحجوزة static تسبق تعريف المتغير. يكون المتغير من هذا النوع موجوداً في الذاكرة حتى إذا لم يتم إنشاء أي كائن، ولا يتم حجز مساحة خاصة بهذا المتغير لكل كائن يتم إنشاؤه كما يحدث في حالة المتغيرات العادية، بل تتخاطب جميع هذه الكائنات مع متغير واحد مشترك بينها. إذا أعدنا تعريف class Employee بإضافة متغير static لحصر عدد الموظفين الذين تم إنشاؤهم في أية لحظة ويكون ذلك بوضع القيمة الابتدائية لهذا المتغير العداد بالصفر، ثم زيادة قيمة العداد كلما تمّ إنشاء كائن.

```
public  class Employee {
        private String name;
      private int roomNumber;
      private static int numberOfEmployees = 0;
     public Employee(  ) {
          System.out.prntln ("New employee added");
          numberOfEmployees ++;
}
    public Employee(String  name, int roomNumber) {
        this(  );
        this.name = name;
```

```
        this. roomNumber = roomNumber;
  }
   public static void getEmployeeCount ( ) {
        System.out.println("There are: " + numberOfEmployees + "
employees.");
}
   public static void main(String args[ ]) {
         Employee.getEmployeeCount( );
         Employee  emp1 = new Employee("Ali", 1);
         emp1.getEmployeeCount( );
         Employee emp2 = new Employee("Maha", 5);
         emp1.getEmployeeCount( );
     }
}
```

برنامج 2-1-6

المتغير numberOfEmployees هو متغير static مشترك بين الموظفين، ويتم إعطاؤه القيمة الابتدائية صفراً عند تعريف الـ class. يوجد مشيدان بالـ class، أحدهما لا يستقبل قيماً حيث يقوم بزيادة قيمة العداد دلالة على إنشاء كائن اضافي للموظفين، ومشيد آخر يستقبل الوسائط، هذا المشيد ينادي المشيد الأول باستخدام المؤشر this( ) وهو نداء لمشيد الـ class، حيث يمكن للمشيدات أن تنادي بعضها البعض بدلاً عن تكرار العبارات بداخلها بشرط أن يكون النداء هو أول عبارة داخل المشيد. نلاحظ أيضاً أن الدالة getEmployeeCount هي أيضاً static، وهذا يعني استقلالها عن الكائنات ووجودها مسبقاً وتكون مشتركة بين الكائنات مثل المتغيرات من نوع static، لذلك يمكن نداؤها باستخدام اسم الـ class كما هو موضح بالدالة الرئيسية. وإذا كان هناك متغير public static يمكن نداؤه أيضاً باستخدام اسم الـ class ومثال لذلك الثابت Math.PI الذي يوفره class Math، حيث إن تعريف PI هو:

```
public static final double PI = 2.14;
```

في المثال السابق، كلما تم انشاء كائن من نوع Employee، يتم زيادة العداد numberOfEmployees بواحد من داخل المشيد الذي ينادى تلقائياً عند إنشاء الكائن، لتكون مخرجات البرنامج هي:

```
There are 0 employees.
New employee added
There are 1 employees.
New employee added
There are 2 employees.
```

مخرجات برنامج 2-1-6

يمكن للفئة أن تحتوي على أى عدد من المؤشرات لكائنات من فئات أخرى مادام يستخدمها الكائن. كما يمكن أن يحتوي تعريف الفئة بداخله على فئة أخرى أيضاً في بعض الحالات، خاصة إذا لم يكن لهذه الفئة الداخلية أي أهمية أو وجود بدون الفئة الخارجية. كذلك يمكن تعريف دالة على أنها private، وهذا يعني أنه لا يمكن نداؤها خارج الفئة باستخدام اسم الكائن والنقطة، بل يتم نداؤها من داخل الفئة فقط لتستخدم لإجراء بعض العمليات داخل الفئة.

6 . الوراثة Inheritance

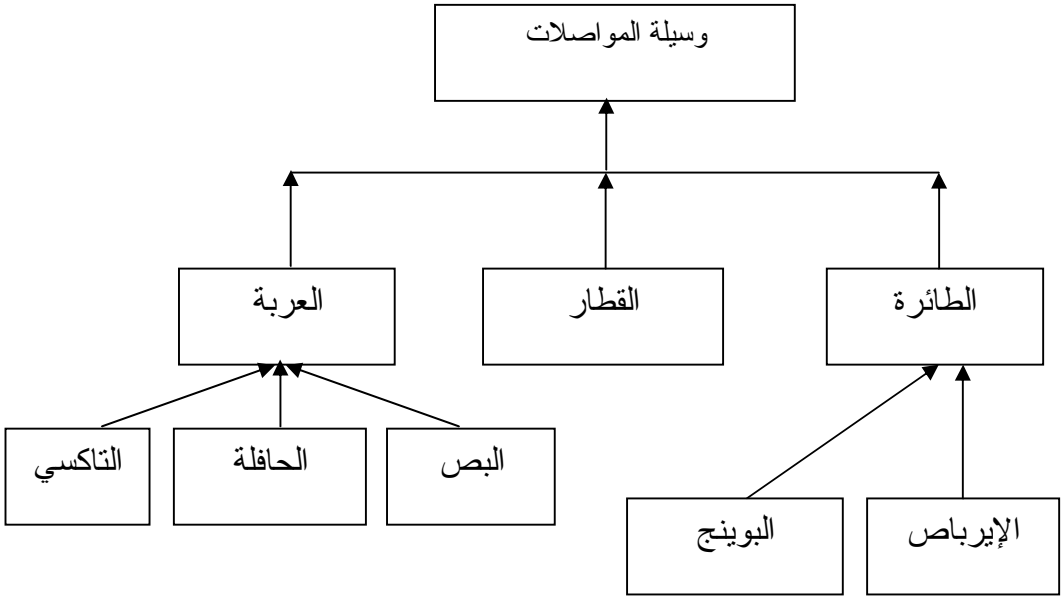
تعتبر الوراثة واحدة من أهم المميزات التي توفرها لغات البرمجة بالكائنات، وتسمح بالاستفادة من خصائص ودوال الفئات مسبقة التعريف لتعريف فئات جديدة، بحيث لا يضطر المبرمج إلى إعادة كتابة تلك الخصائص لمرة ثانية، ولعمل علاقات جديدة تربط بين الكائنات. عندما ترث فئة معينة خصائص فئة أخرى تسمى الفئة الوارثة بالابن subclass وتسمى الفئة الموروثة بالأب supercalss. يمكن أن تكون الفئة الأب أي فئة معرفة سابقا وتأخذ شكل الفئات التي تعرضنا لها سابقا. ولكي ترث فئة معينة فئة معرفة سلفا يتم تحديد هذه العلاقة بالكلمة المحجوزة extends التي تظهر بعد اسم الفئة مباشرة يليها اسم الفئة التي سترثها هذه الفئة، مثلا:

Class SubclassName extends SuperclassName{

يجب أن تكون الفئة الوارثة على علم باسم ومحتويات ووظائف الفئة الأب، على عكس الفئة الأب والتي قد لا تعلم بأن هناك فئة قد ورثتها.

6 . 1 أمثلة على الوراثة

مثلاً لوسيلة المواصلات اسم وخط سير ووجهة ومكان صنع ... إلخ، ترث كل من الطائرة والقطار والعربة هذه الخصائص لتستفيد من كونها معرفة سلفا بدلا من إعادة تعريفها. ونلاحظ أن كل منها يمثل نوعا من أنواع وسائل المواصلات. أي أن العلاقة بين كائنين تكون علاقة وراثة إذا استطعنا وضع عبارة (نوع من) بين الكائنين وكانت الجملة منطقية، مثلا القطار نوع من وسائل المواصلات. للطائرة والقطار والعربة صفات مشتركة هي تلك الموجودة في الفئة وسيلة المواصلات، كما ولكل منها صفاتها المستقلة وهي الصفات الإضافية الموجودة بكل فئة. على هذا تستخدم الكائنات المشتقة من فئة الطائرة جميع الخصائص الموجودة بتلك الفئة، كما يمكن أن تستفيد من الخصائص الموجودة في الفئة وسيلة المواصلات. إذا أراد مبرمج كتابة فئة اسمها الإيرباص، يمكن أن يقوم بتعريفها كاملة ومستقلة، رغم ما فيه من إضاعة للجهد والوقت نسبة لتوفر الخصائص العامة للإيرباص في فئة الطائرة، لأن الإيرباص نوع من الطائرات. وبدلا عن ذلك يمكن أن يدرس الصفات المتوفرة حاليا في الفئات الطائرة ووسيلة المواصلات، ثم إضافة الخصائص الجديدة إلى الفئة الإيرباص، وهي عادة تكون الخصائص التي تميز الإيرباص عما سواها من الطائرات، بنفس الطريقة التي تكون بها محتويات فئة الطائرة هي الخصائص التي تميز الطائرة عن غيرها من وسائل المواصلات.



الشكل 2-2-1

2.6 كيف يستخدم الوارث متغيرات ودوال ومشـــيدات الفئة الأب

فيما يلي برنامج مبسط يعرف الفئة وسيلة المواصلات Transportation والتـــي تحتـــوي على المتغير passenger والذي يمثل العدد الأقصى للركاب، ويعرف الفئة الطائرة plane والتي تحتوي على المتغير flightNo وهو رقم الرحلة، ويشرح المثـــال كيـــف يســـتخدم الوارث متغيرات ودوال ومشيدات الفئة الأب.

```
public  class Transportation {
        private int passengers;
        public Transportation(int passengers) {
                this.passengers = passengers;
        }
        public void printLuggageAllowed (){
                System.out.println("maximum luggage allowed:"+ (passengers*
                10) + "kilos");
```

```
        }
        public int getPassengers (){
                return passengers;
        }
}
public class Plane extends Transportation{
        private int flightNo;
        public Plane (int flightNo, int passengers){
                super(passengers);
                this.flightNo = flightNo;
        }
        public void printLuggageAllowed( ) {
                int passengers = super.getPassengers( );
                System.out.println("Maximum luggage allowed: " + (passengers
                *20) + "kilos");
        }
        public static void main(String args[ ] ){
                Plane sudanair = new plane( 77, 300 );
                sudanair.printLuggageAllowed( );
                System.out.println("Number of passengers: " +
                sudanair.getPassengers( ));
        }
}
```

برنامج 2-2-1

يبدأ تنفيذ البرنامج من الدالة الرئيسية main حيث يتم تعرف كائن من النوع plane وتحديد رقم الرحلة والعدد الأقصى للركاب، ويستقبل مشيد الفئة هذه الوسائط، نلاحظ أن الفئة plane لها متغير عبارة عن رقم الرحلة إضافة إلى المتغير الآخر الذي ترثه من الفئة الأب، ولوضع قيمة ابتدائية في هذا المتغير الثاني يجب أن نستخدم مشيد الفئة الأب. ينادى مشيد الفئة الأب باستخدام المؤشر reference "super"، والذي نستطيع بواسطته الوصول إلى متغيرات ودوال ومشيدات الفئة الأب ما عدا تلك المعرفة private فهي تخص الفئة الأب وحدها ولا يمكن الوصول إليها إلا باستخدام دوال نفس الفئة. يقوم مشيد الفئة Plane بتمرير القيمة الابتدائية 300 إلى مشيد الفئة

Transportation والذي يضعها في المتغير passengers، ويضع القيمة الابتدائية لرقم الرحلة بالمتغير flightNo. نلاحظ وجود دالة بالاسم printLuggageAllowed في كل من الفئتين فيما يعرف بالـ overriding وهو إعادة تعريف دالة في الفئة الابن موجودة مسبقا بالفئة الأب. ويكون ذلك إذا كانت الدالة المعرفة بالفئة الأب غير كافية أو غير مكتملة أو حتى غير صحيحة بالنسبة للفئة الابن مثلا: الدالة printLuggageAllowed تستخدم لحساب الوزن الأقصى للأمتعة، وفي الفئة الأب تم افتراض أن الحد الأقصى لكل راكب هو عشرة كيلو جرام، وعند تعريف الفئة Plane لم تكن هذه هي المعادلة المطلوبة، حيث تسمح الطائرة لكل راكب بعشرين كيلو جرام من الأمتعة. ولذلك عند تنفيذ العبارة ()sudanair.printLuggageAllowed يتم نداء الدالةprintLuggageAllowed الموجودة في الفئة Plane، ونلاحظ أن هذه الدالة تستعين بالدالة getPassengers الموجودة في الفئة الأب باستخدام المؤشر super. ولتنفيذ العبارة ( ) sudanair.getPassengers يتم البحث عن الدالة getPassengers بداخل الفئة plane، وإذا وجدت يتم تنفيذها، وإن لم توجد – كما في هذا المثال – يتواصل البحث في الفئة الأب، وعندما يتم إيجادها يتم تنفيذها. ولأن الدالة getPassengers مكتوبة بصورة مرضية للفئة Plane لم تعمل على تعديلها كما حدث في حالة الدالة printLuggageAllowed، بل سمحت للكائنات من نوع الفئة Plane باستخدامها مباشرة. مخرجات البرنامج أعلاه هي:

```
Maximum luggage allowed: 6000 kilos
Number of passengers: 300
```

مخرجات برنامج 2-2-1

من هذا المثال نخلص إلى أن الفئة الابن يمكن أن:

1- تستخدم جميع الخصائص والدوال والمشيدات المعرفة بالفئة الأب.

2- تعيد تعريف الدوال لاستخدامها بصورة أفضل (overriding).

3- تضيف مكونات جديدة على المكونات الموروثة.

6 .3 تعريف المتغيرات والدوال

توجد طريقة ثالثة لتعريف المتغيرات والدوال إلى جانب private وpublic وهي protected، وفيما يلي شرح لكل معرف منهم في مفهوم الوراثة:

1- public: يمكن الوصول للمتغيرات والدوال المعرفة public من داخل الفئة المعرفة بها ومن داخل الفئات التي ترث تلك الفئة وباستخدام أسماء الكائنات المعرفة من نوع تلك الفئة.

2- private: يمكن الوصول للمتغيرات والدوال المعرفة private من داخل الفئة المعرفة بها، ولكن لا يمكن الوصول إليها من داخل الفئات التي ترث تلك الفئة ولا باستخدام أسماء الكائنات المعرفة من نوع الفئة، أي أنها خاصة بالفئة فقط.

3- protected: يمكن الوصول للمتغيرات والدوال المعرفة protected من داخل الفئة المعرفة بها ومن داخل الفئات التي ترث تلك الفئة، ولكن لا يمكن الوصول إليها باستخدام أسماء الكائنات المعرفة من نوع الفئة، أي أنها خاصة بالفئة والفئات التي ترث منها.

تعتبر المعرفات الثلاثة public – private – protected عبارة عن طرق للتحكم في إمكانيات الوصول والتعامل مع المتغيرات والدوال المعرفة بداخل الفئات.

6 .4 استخدامات الكملة المحجوزة final

يمكن كذلك التحكم في توريث الفئات والدوال عن طريق استخدام الكملة المحجوزة final، حيث إن استخدامات final المختلفة هي:

1- المتغير المعرف final لا يمكن تغيير قيمته، أي أنه ثابت.

2- الفئة المعرفة final لا يمكن الوراثة منها، أي أنها لا يمكن أن تكون أب لفئة أخرى.

3- الدالة المعرفة final في فئة أب لا يمكن إعادة تعريفها (override) في الفئة الابن، بل يجب استخدامها كما هي. إن الدوال المعرفة private أو static في أي فئة أب تكون تلقائيا final لأنها أيضا لا يمكن إعادة تعريفها.

7 . منع إنشاء الكائنات من نوع فئة محددة

في المثال السابق قمنا بتعريف الفئة Transportation، وورثتها فئات أخرى مثل العربة والقطار والطائرة. وهي الفئات التي يتم إنشاء الكائنات منها للتعامل معها في البرامج ولكن نلاحظ أن تعريف كائنات من نوع Transportation ليس له معنى في البرنامج. لأن الكائن من نوع القطار مثلا يسمح للمسافرين بالحجز والركوب، ويتحرك القطار من نقطة إلى أخرى في ساعة معينة. وتتفاعل معه الكائنات الأخرى مثل المسافرين والمحطات وغيرها. أما وسيلة المواصلات فلا تمثل كائن في الحياة له تفاعل إلا بعد تحديد نوع وسيلة المواصلات، وهو الفئات الأخرى. عليه، يتم تعريف الفئة Transportation من أجل الوراثة لتكون الفئة الأب لعدة فئات لخلق علاقة بينها وتعريف الخصائص المشتركة بينها لتنظيمها. ولمنع إنشاء الكائنات من نوع فئة محددة، يتم تعريفها من نوع abstract والتي تدل على أن هذه الفئة مصممة لأغراض الوراثة ولا يمكن اشتقاق كائنات منها. يمكن تعديل تعريف الفئة Transportation كالآتي:

```
public abstract class Transportation{
```

كما ذكرنا سابقا، إذا احتوت الفئات الأبناء على خصائص مشتركة يتم تعريفها في الفئة الأب بدلا من تكرارها بداخل كل من الفئات الأبناء. لكن أحيانا تكون هناك دوال مشتركة بين الأبناء ومختلفة من بعضها البعض في التفاصيل. فمثلا، لكل من القطار والطائرة والعربة دالة لشراء التذاكر يتم عند ندائها تحديد عدد التذاكر المطلوبة. ولكن قد تختلف طريقة حساب المبلغ الإجمالي من وسيلة نقل إلى أخرى حتى إذا افترضنا أنها جميعا نشترك في سعر التذكرة الواحدة. على سبيل المثال في الطائرة يتم دفع نصف

السعر لتذاكر الأطفال، وفي البص يمنح تخفيض 10% عن السعر الإجمالي إذا زادت التذاكر عن الخمس، وفي القطار يكون سعر التذكرة أقل إذا تم شراؤها قبل تاريخ السفر بعشرة أيام فأكثر. نلاحظ هنا رغم أن لجميع هذه الفئات نفس الدالة لشراء التذاكر، إلا أن طريقة عمل كل منها مختلف. المطلوب هنا هو أن تتمتع كل فئة بحرية تعريف دالتها بما يناسبها، وفي نفس الوقت نرغب في الإشارة إلى أن جميع هذه الفئات الوراثة تشترك في هذه الخاصية، ولزيادة النظام بين هذه الفئات نريد أن نوحد اسم هذه الدالة وتعريفها لجميع هذه الفئات. يمكن أن نعرف الدالة في الفئة الأب من النوع abstract وهي تعني أن الدالة خالية من التعليمات أي أن بها تعريف الدالة فقط، وعلى جميع الفئات الوارثة من هذه الفئة كتابة تفاصيل هذه الدالة. يؤدي هذا إلى توحيد اسم الدالة بين جميع الفئات الوارثة مع حرية تعريف تفاصيل هذه الدالة في كل فئة. فيما يلي تعريف لعدد من الفئات المذكورة في الشكل 2-2-1 لتوضيح كيفية تعريف الفئات والدوال من النوع abstract:

```
public abstract class Transportation  {
        private int passengers;
        protected final int ticketPrice = 50;
        public Transportation(int passengers) {
                this.passengers = passengers;
        }
        public abstract void calculateTotalPay(int numberOfTickets);
}
public class Plane extends Transportation {
        private int flightNo;
        public Plane(int flightNo, int passengers){
                super(passengers);
                this.flightNo = flightNo;
        }
        public void calculateTotalPay(int numberOfTickets){
                System.out.println("Enter number of children:");
                int children = Input.readInt( );
                int adult = numberOfTickets – children;
                int totalPay = (adult * ticketPrice) + (children * ticketPrice / 2);
                System.out.println("please pay: " + totalPay);
```

```
        }
}
public class Bus extends Transportation{
        public Bus(int passengers) {
                super (passengers);
        }
        public void calculateTotalPay(int numberOfTickets) {
                int totalPay = numberOfTickets * ticketPrice;
                if (numberOfTickets >5)
                totalPay -= numberOfTickets * ticketPrice * 10/100; // 10%
discount
        }
}
```

برنامج 2-2-2

الفئة الأولى هي الفئة الأب Transportation وتحتوي على المتغيرات والدوال المشتركة بين وسائل المواصلات المختلفة. متمثلة في المتغير passengers والذي يعبر عن العدد الأقصى للركاب، والثابت ticketPrice يحدد سعر التذكرة الواحدة. تقوم جميع وسائل المواصلات بحساب ثمن جملة التذاكر، ولكنها تختلف عن بعضها في طريقة حساب هذه الجملة، لذلك فقد تم تعريف الشكل العام للدالة calculateTotalPay في الفئة الأب مع جعلها abstract بدون تفاصيل. وبما أن جميع الفئات الأخرى ترث من الفئة Transportation، فيجب عليها تعريف الدالة calculateTotalPay بما يناسبها. تقوم الفئة Plane بتعريف الدالة calculateTotalPay بحيث تحسب نصف قيمة التذكرة للأطفال، ولذلك تستفسر أولا عن عدد الأطفال من بين العدد الكلي للتذاكر المطلوبة numberOfTicket. أما عند شراء تذاكر على البص فيتم خصم 10% عن القيمة الكلية للتذاكر إذا تجاوز عددها الخمس تذاكر.



إذا احتوت فئة على دالة أو أكثر abstract، يجب تعريف الفئة على أنها abstract وهو شرط منطقي حيث إنه عندما تكون الدالة abstract تكون غير مكتملة وغير محددة الوظيفة، فإذا كان من الممكن تعريف كائنات من فئة تحتوي على دالة abstract، يؤدي هذا إلى لبس عند نداء الدالة غير المكتملة. ومن المنطقي أيضا عدم رغبتنا في إنشاء كائنات من فئة كهذه لأن احتواءها على دالة abstract يدل على عدم اكتمال تعريف الفئة مما يعني استخدامها كفئة أب لفئات أخرى.

أسئلة تقويم ذاتي



1- توجد ثلاثة طرق لتعريف المتغيرات والدوال اذكرها. ثم تناول بالشرح كل معرَّف منها في مفهوم الوراثة ؟

2- ما استخدامات الكلمة المحجوزة final في التحكم في توريث الفئات والدوال ؟

8 . استخدام الواجهات Interface

عزيزي الدارس،

تعتبر لغة جافا من اللغات وحيدة الوراثة single inheritance languages، وتعني انه يمكن للفئة أن ترث خصائص فئة واحدة على الأكثر. من مميزات هذا النوع سهولة إدارة الكائنات وتحديد علاقتها وصلاحيتها، وله عيب هو عدم التمكن من وراثة خصائص معرفة في عدد من الفئات. تدعم بعض اللغات الوراثة المتعددة multiple inheritance، وفيها يمكن للفئة أن ترث خصائص أكثر من فئة واحدة. وميزة هذا النوع

من الوراثة هو الاستفادة من خصائص عدد أكبر من الفئـــات، وعيبهـــا صـــعوبة إدارة الكائنات وتحديد صلاحياتها. وضعت لغة جافا حلا لتلافـــي عيـــب الوراثـــة الوحيـــدة باستخدام الواجهات Interface، وهو هيكل يشبه في تركيبه الفئة، إلا أن جميـــع دوالـــه خالية من التعليمات وجميع بياناته ثوابت. يمكن للفئة أن تستفيد من الخصائص المعرفـــة بعده interfaces ولكن لا يطلق عليها وراثة بل تسمى تعريفاً implementation. وفيمـــا يلي تعريف لواجهة وكيفية استخدام الفئة لهذه الواجهة.

```
interface Operations{
        public int add(int x, int y);
        public int sub(int x, int y);
        public int mult(int x, int y);
        public double div(int x, int y);
}
public class Calculator implements Operations{
        public int add(int x, int y){
                return x + y;
        }
        public int sub(int x, int y){
                return x - y;
        }
        public int mult(int x, int y){
                return x * y;
        }
        public double div(int x, int y){
                return (double ) x / (double) y;
        }
        public static void main(String args[ ]) {
                Calculator c = new Calculator( ) ;
                int a = 10, b=2;
                System.out.println("a + b = "+ c.add (a,b) );
                System.out.println("a – b = " + c.sub(a,b) );
                System.out.println("a * b = " + c.mult(a,b) );
                System.out.println("a / b = " + c.div(a,b) );
        }
}
```

برنامج 2-2-3

في هذا البرنامج تعريف للواجهة Operations والتي تقوم بتعريفها الفئة Calculator باستخدام الكلمة المحجوزةimplements. ويكون هذا الاستخدام عن طريق تعريف جميع الدوال الموجودة بالواجهة، حيث يؤدي عدم تعريف أي دالة منها بنفس الطريقة الموجودة بها في الواجهة إلى حدوث خطأ في ترجمة البرنامج.

يمكن كتابة ترويسة الدالة داخل الفئة ثم تركها خالية إذا لم يكن هناك حاجة لاستخدامها، ولكن لابد من وجودها بالفئة. تم تعريف الدوال الأربع داخل الفئة ثم كتابة وظائفها ومن ثم استخدامها بواسطة كائن تم إنشاؤه داخل الدالة الرئيسية. يمكن تعريف أي عدد من الواجهات بالفئة الواحدة، ويفصل بين أسمائها في ترويسة الفئة بواسطة الفاصلة، مثلا:

```
public class MyClass implements Interface1, Interface2, Interface3{
```

ويتم تعرف جميع دوال الواجهات الثلاث داخل الفئة. يمكن كذلك الجمع بين وراثة فئة وتعريف واجهة أو أكثر، مثلا:

```
public class MyClass extends OtherClass implements MyInterface{
```

وهنا نلاحظ أن استخدام الواجهات جعل بالإمكان استخدام الخصائص الموجودة في أكثر من هيكل كما سنلاحظ في عدد من البرامج خلال الفصول المختلفة. عند تنفيذ البرنامج السابق، تكون المخرجات هي:

```
a + b = 12
a – b = 8
a * b = 20
a / b = 5.0
```

أسئلة تقويم ذاتي



1- وضح مميزات وعيوب كل من الوراثة الوحيدة single inheritance languages ، والوراثة المتعددة multiple inheritance؟

2- ما الحل الذي وضعته لغة جافا لتلافي عيب الوراثة الوحيدة ؟

9 . استخدام الواجهات كواجهة بين الكائن والمستخدم

لا يقتصر مفهوم الواجهات على تعريف أسماء الدوال والثوابت لاستخدامها داخل الفئة فحسب، بل تستخدم كواجهة بين الكائن ومستخدم هذا الكائن، فتحدد له الجزء المصرح له باستخدامه من الكائن، و تمنعه من الوصول إلى بقية محتويات الكائن. ولهذا السبب سميت الواجهة بهذا الاسم، لأنها تعمل كواجهة أو وسيط بين الكائن ومستخدمه. ولتوضيح هذا المفهوم نستعين بالبرنامج التالي والذي يمثل نظاماً في مدرسة، حيث يتم حفظ كمية كبيرة من البيانات للطالب الواحد متمثلة في اسمه والرسوم الدراسية المقررة ومقدار ما دفعه الطالب منها، إضافة إلى درجاته في سبع مواد. كل طالب عبارة عن كائن، وهناك ثلاث جهات تتعامل مع هذا الكائن وهي:

1- الطالب: حيث يستطيع الاطلاع على جميع البيانات السابقة، مع عدم إمكانية التعديل فيها.

2- الأستاذ: يحق له الاطلاع على اسم الطالب ودرجاته فقط، مع إمكانيه تعديل الدرجات.

3- المحاسب: ويسمح له الاطلاع على اسم الطالب ورسومه وما تم دفعه منها، وإمكانية تعديل المبلغ الذي تم دفعه.

سيتم تعريف كل من هذه العناصر في النظام على شكل واجهات تقوم بتعريفها الفئة الاساسية StudentData.

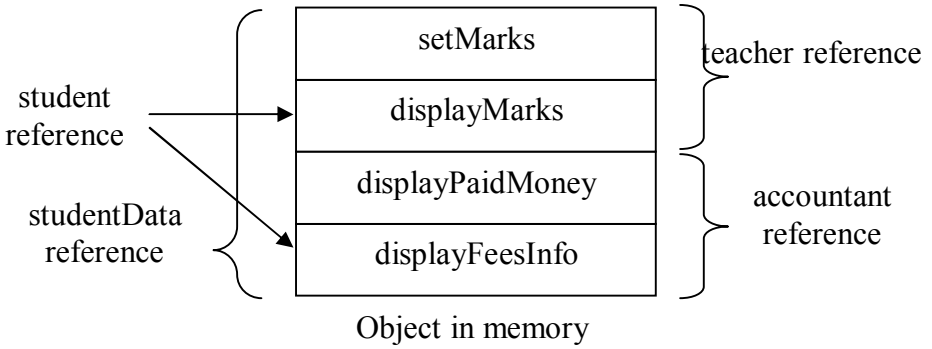
```
 interface Student{
        public void displayMarks( );
        public void displayFeesInfo( );
}
interface Teacher {
        public void displayMarks( );
        public void setMarks( );
}
interface Accountant {
        public void displayFeesInfo( ) ;
        public void setPaidMoney(int paidMoney);
}
```

```
public class StudentData implements Student, Teacher, Accountant {
        private String name;
        private int paidMoney, marks[ ];
        private final int schoolFees = 1000;
        public StudentData(String name) {
                this.name = name;
                marks[ ] = new int[ 7 ];
        }
        public void setPaidMoney(int paidMoney){
                this.paidMoney = paidMoney;
        }
        public void displayFeesInfo(  ){
                System.out.println("School fees = "+ schoolFees
                                + "\n Money paid = "+ paidMoney
                                + "\n Remaining = " + (schoolFees – paidMoney)
);
        }
        public void setMarks( ) {
                for (int i=0; i < 7; i++) {
                        System.out.print("Enter mark of subject " + (i + 1) + ": ");

                        marks[i] = Input.readInt( );
                }
        }
        public void displayMarks(  ) {
                for (int i = 0; i < 7; i++)
                        System.out.println("Mark of subject: " + (i +1) + " is: " +
        marks[ i ] );
        }
        public static void main(String args[ ] ) {
                StudentData studentData = new StudentData("Fatima");
                Accountant accountant = studentData;
                accountant.setPaidMoney(700);
                Teacher teacher = studentData; teacher.setMarks( );
                teacher.displayMarks ( );
                Student student = studentData;
                student.displayFeesInfo( );
        }
}
```

برنامج 2-2-4

في هذا البرنامج تعريف لثلاث واجهات تمثل كل منها مستخدماً مختلفاً يتعامل مع البيانات بطريقة معينة. تحتوي الفئة StudentData على بيانات الطالب إضافة إلى جميع الدوال التي يتعامل معها المستخدمون. الجديد في هذا البرنامج هو عدم نداء الدوال مباشرة عن طريق استخدام اسم الكائن studentData، وبدلاً عن ذلك قمنا بتعريف ثلاثة مؤشرات references من أنواع الواجهات الثلاث، ثم جعلناها تشير إلى الكائن الذي يحتوي على البيانات. يتعامل المؤشر من نوع Student حسب تعريفه مع دالتي الطباعة فقط، ولذلك لا يستطيع نداء دالة تعديل الدرجات أو الرسوم المدفوعة، لأنه يتعامل مع البيانات عبر واجهة مخصصة للطالب. بالنسبة للأستاذ فيستطيع التعامل مع الدرجات قراءةً وكتابةً، لأن واجهة الأستاذ تسمح له بذلك، ولكنه لا يستطيع الوصول إلى أي معلومات عن رسوم الطالب. تسمح واجهة المحاسب للمحاسب أن يتعامل مع بيانات الطالب الخاصة بالرسوم دون درجات المواد.

تؤدي محاولة تنفيذ أي شخص لدالة لا توفرها له واجهته الخاصة به إلى خطأ بالبرنامج، مما يضمن خصوصية وسرية البيانات، والتحكم في الوصول لهذه البيانات. يرجع السبب إلى أن الأستاذ مثلاً لا يستطيع الوصول للمعلومات المالية إلى أن المتغير المستخدم للوصول معرف من الواجهة Teacher، وهو مؤشر له إمكانية التعرف إلى دالتي الدرجات فقط، على العكس من المتغير المعرف من نوع الفئة StudentData.



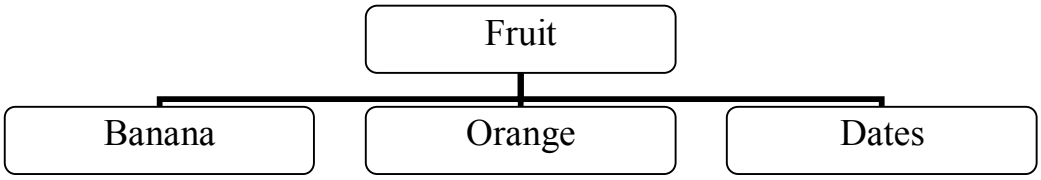
شكل 2-2-2

فيما يلي مثال لتنفيذ البرنامج السابق:

```
Enter mark of subject 1: 55
Enter mark of subject 2: 80
Enter mark of subject 3: 71
Enter mark of subject 4: 66
Enter mark of subject 5: 89
Enter mark of subject 6: 61
Enter mark of subject 7: 75
Mark of subject: 1 is: 55
Mark of subject: 2 is: 80
Mark of subject: 3 is: 71
Mark of subject: 4 is: 66
Mark of subject: 5 is: 89
Mark of subject: 6 is: 61
Mark of subject: 7 is: 75
School fees = 1000
Money paid = 700
Remaining = 300
```

1.9 خاصية تعدد الأشكال Polymorphism

هناك خاصية هامة جداً توفرها بعض لغات برمجة الكائنات، تضفي هذه الخاصية المرونة على الوراثة، تعرف هذه الخاصية بتعدد الأشكال Polymorphism. نستطيع عن طريق هذه الخاصية التعامل مع كائنات من أنواع مختلفة باستخدام reference واحد معرف من نوع الفئة الأب لهذه الفئات أو الواجهة المشتركة بينهم بدون معرفة نوع الكائن بالتحديد. والمثال التالي يشرح تعدد الأشكال.



شكل 2-2-2

يوضح الشكل أعلاه العلاقة بين مجموعة من الفئات تمثل أنواعاً من الفاكهة لكـــل منهـــا وزن معين وسعر للوحدة الواحدة، جميع هذه الفواكه تعرف محتويـــات الواجهـــة Fruit وهي الدالة printPrice. وفيما يلي تعريف لهذه الفئات:

```
public interface Fruit {
      public void printPrice( );
}
public class Banana implements Fruit {
       private final int KILO_PRICE = 2;
       private int kilos;
       public Banana(int kilos) {
              this.kilos = kilos;
     }
      public void printPrice( ) {
            System.out.println("The price of " + kilos + " kilos of
bananas is: " + kilos * KILO_PRICE + " pounds.");
      }
}
public class Orange implements Fruit {
       private final int DOZEN_PRICE = 6;
       private int dozens;
       public Orange(int dozens) {
              this.dozens = dozens;
      }
      public void printPrice( ) {
          System.out.println("The price of " + dozens + " dozens              of
pounds.");    oranges is: " + dozens * DOZEN_PRICE + "
     }
}
public class Dates implements Fruit {
       private final int MALWA_PRICE = 15;
       private int malwas;
       public Dates(int malwas) {
              this.malwas = malwas;
       }
       public void printPrice( ) {
              System.out.println("The price of " + malwas + "
malwas of dates is: " + malwas * MALWA _PRICE + " pounds. ");
     }
```

```
}
public class MainClass {
        public static void main (String args [  ]) {
                Fruit fruit[  ] = new Fruit [6];
                fruit[0] = new Banana(3);
                fruit[1] = new Orange(4);
                fruit[2] = new Dates(2);
                fruit[3] = new Banana(6);
                fruit[4] = new Orange(5);
                fruit[5] = new Dates(3);

               for (int i = 0; i < 6;   i ++)
                    fruit[ i ].printPrice(  );
      }
}
```

برنامج 2-2-5

يحتوي هذا المثال على ثلاثة أنواع من الفواكه:

1) الموز: وله سعر محدد للكيلو، عند تعريف كائن من هذا النوع يتم تحديد عدد الكيلوجرامات المطلوبة.

2) البرتقال: وللدستة سعر معين، يتم تحديد عدد الدستات عند تعريف كائن من نوع البرتقال.

3) التمر: وله سعر معين للملوة، يتم تحديد عدد الملوات المطلوبة منه لحظة تعريف الكائن.

تعرف كل من هذه الفئات الواجهة Fruit ومن ثمَّ تعرف كل فئة محتويات الدالة printPrice والمستخدمة لحساب سعر الكمية المطلوبة من الفاكهة المعينة، ونلاحظ أنَّ لكل فئة طريقتها الخاصة في حساب هذا السعر.

تظهر خاصية تعدد الأشكال في الدالة الرئيسية، حيث تم تعريف مصفوفة من نوع الواجهة Fruit، وهو ممكن بالنسبة للواجهات والفئات المعرفة abstract، لأنَّه يمكن تعريف مؤشرات من هذه الأنواع ولكن لا يمكن تعريف كائنات من نوعها. فالعبارات:

```
Fruit   f1, f2;
```

f1 = f2;

عبارات صحيحة لأى واجهة أو abstract class لأن f1 و f2 ليست كائنات. أما العبارة:

Fruit f1= new Fruit( );

تعتبر خطأ، لأنه غير مسموح بإنشاء الكائنات من الواجهة، أما كلمة new الموجودة في تعريف المصفوفة فهى لإنشاء المصفوفة نفسها، وينتج عن العبارة:

Fruit fruit[ ] = new Fruit[6];

إنشاء مصفوفة من ستة مؤشرات تستطيع الوصول إلى كائنات تعرف الواجهة Fruit. يتم بعد ذلك تعريف الكائنات التي تشير إليها هذه المؤشرات، حيث نلاحظ أننا استطعنا إنشاء كائنات من فئات مختلفة ووضعها في نفس المصفوفة. تعتبر حلقة for هي قلب خاصية تعدد الأشكال، والمتمثل في النداء:

fruit[i].printPrice( );

حيث يتم استخدام نفس هذه العبارة لتنفيذ دوال مختلفة معرفة في فئات مختلفة حسب نوع الكائن الذي يشير إليه المؤشر fruit[i]، فمثلاً يشير fruit [0] إلى كائن من نوع Banana، ولذلك يتم تنفيذ الدالة printPrice الموجودة فى الفئة Banana وهكذا، لتكون مخرجات البرنامج هي:

The price of 3 kilos of bananas is: 6 pounds.
The price of 4 dozens of oranges is: 24 pounds.
The price of 2 malwas of dates is: 30 pounds.
The price of 6 kilos of bananas is: 12 pounds.
The price of 5 dozens of oranges is: 30 pounds.
The price of 3 malwas of dates is: 45 pounds.

مخرجات برنامج 2-2-5

نود الإشارة إلى أن المؤشر من نوع الواجهة Fruit يستطيع نداء الدالة printPrice فقط في الكائن الذي يشير إليه، لأنها الدالة الوحيدة المعرفة داخل الواجهة، وإذا احتوت الفئات الأخرى على دوال أكثر فإنه أيضاً لا يستطيع التعامل معها رغم وجودها داخل الكائن.

من هذا المثال نلاحظ الفرق الواضح بين المؤشر reference والكائن object، حيث يمثل الكائن موقعاً في الذاكرة يحتوي على بيانات ودوال الكائن، بينما المؤشر هو اسم المتغير

الذي من خلاله نستطيع الوصول إلى موقع الكائن في الذاكرة. يمكن أن يتطابق المؤشر مع الكائن كما يحدث غالباً، مثلاً:

```
Banana b = new Banana(4);
```

حيث نستطيع من خلال المؤشر b مناداة جميع الدوال المعرفة داخل الفئة Banana. وقد لا يتطابق المؤشر مع الكائن، كما حدث في المثال السابق، فمثلاً في العبارة:

```
Fruit  f = new Banana(4);
```

يستطيع المؤشر f مناداة الدوال المعرفة داخل الواجهة Fruit فقط.

10 . مجموعات الفئات Packages

توضع مجموعة الفئات والواجهات التي تنتمي لنفس التطبيق في وحدة تسمى package، وتحفظ في ملف يحمل اسم الـ package التي تنتمي إليها فئة معينة. يتم كتابة الكلمة المحجوزة package في بداية الملف الذي يحتوي على الفئة ثمّ كتابة اسم الـ package، مثلاً:

```
package  fruits;
```

ويلي ذلك تعريف الفئة أو الواجهة بالطريقة المعروفة. يوجد عدد كبير من الـ packages المعرفة في لغة جافا ليستعين المبرمج بفئاتها وواجهاتها عند الحاجة. من هذه الواجهات java.lang ويحتوي على الفئات الأساسية في جافا مثل System و Object و Math، لذلك يتم تضمينها داخل أي برنامج جافا دون الحاجة لكتابة عبارة صريحة. أما الـ packages الأخرى فيجب تضمينها في البرنامج بعبارة صريحة للتمكن من استخدامها، ويكون ذلك قبل تعريف الفئة كالآتي:

```
package  PackageName;
import java.util.*;
public class MyClass {
```

وعلامة * تعنى تضمين جميع محتويات الـ package في البرنامج، أما لتضمين فئة أو واجهة معينة فيتم كتابة اسمها، مثلاً:

```
import java.util.Date;
```

حيث Date هو class معرف في اللغة يستخدم للتعامل مع التـــواريخ. نتعـــرف خـــلال الفصول القادمة على عدد كبير من الـــ packages والفئات والواجهات الموجـــودة بهـــا واستخداماتها المختلفة.

الخلاصة

عزيزي الدارس،

وضحت الوحدة الفـرق بين البرنامج 2 – 1 – 1 والبرنـامج 2 – 1 – 2 الـذي لـه خصوصية في بيانات الصنف وسرية لعدم إمكانية الوصول إليها إلا عن طريـق الدالـة set والدالة get ، وهذا يضمن ان كل من يتعامل مع البيانات سيتعامل معها وفق قوانين وشروط الصنف .

كما أوضحت الوحدة أن كل class يمتلك دالة خاصة تعرف باسم المشيد ، تستخدم هذه الدالة لإجراء العمليات التي نرغب في تنفيذها ابتدائياً لحظة إنشاء الكائن وقبل تعامل أي جهة مع هذا الكائن ، ويأخذ تعريف أي constructor الشكل:

```
public ClassName(parameter list) {
    // constructor code
}
```

ولإنشاء كائن محدد يتم استخدام الصيغة:

```
ClassName objectName = new ClassName( );
```

- يمكن تعديل المشيد بحيث يتم تمرير القيم الابتدائية لمتغيرات الـ class مـن الدالـة الرئيسية، وتوضع هذه القيم بين قوسي المشيد عند ندائه لحظة إنشاء الكائن كالآتي:

```
MyCircle circle = new MyCircle(10);
```

ويستقبل المشيد هذه القيمة بعد تعديل تعريفه كالآتي:

```
public MyCircle(double r) {
    radius = r ;
}
```

- يمكن أن يحتوى نفس الـ class على أكثر من مشيد واحد يختلفون في عدد أو نـوع بيانات الوسائط فيما يعرف بالـ constructor overloading

- من حق أي دالة أن تسمى متغيراتها بأى اسم.

– في مفهوم الفئات إنّ أى متغير معرف في الفئة (خارج الدوال) يعبر عن خاصية من خواص الفئة ويكون معروف في أي منطقة داخل الفئة ويمكن الوصول إليه من داخل جميع دوال الفئة.

– المتغيرات المحلية المعرفة داخل الدوال تخص الدالة ولا تكون معروفة خارجها.

كما بينت الوحدة أن كل class يستطيع أن يشير إلى محتوياته من متغيرات ودوال باستخدام المؤشر "this" reference .

قدمت الوحدة العلاقة بين الكائنات أو ما يعرف بالـ composition وذكرت أن الأساس فى وجود مؤشر لكائن بداخل كائن آخر هو أن يملكه.

كما بينت الوحدة كيف يمكن تعريف متغير مشترك بين جميع الكائنات المشتقة من class محدد باستخدام الكلمة المحجوزة static تسبق تعريف المتغير. كما أوضحت الوحدة أيضاً انه يمكن للفئة أن تحتوى على أى عدد من المؤشرات لكائنات من فئات أخرى مادام يستخدمها الكائن. كما يمكن أن يحتوى تعريف الفئة بداخله على فئة أخرى أيضاً في بعض الحالات، خاصة إذا لم يكن لهذه الفئة الداخلية أي أهمية أو وجود بدون الفئة الخارجية. كذلك يمكن تعريف دالة على أنها private، وهذا يعنى أنه لا يمكن نداؤها خارج الفئة باستخدام اسم الكائن والنقطة، بل يتم نداؤها من داخل الفئة فقط لتستخدم لإجراء بعض العمليات داخل الفئة.

وتناولت الوحدة الوراثة Inheritance والتي تتمثل أهميتها في أنها تسمح بالاستفادة من خصائص ودوال الفئات مسبقة التعريف لتعريف فئات جديدة ، ولعمل علاقات جديدة تربط بين الكائنات ، وأوضحت : عندما ترث فئة معينة خصائص فئة أخرى تسمى الفئة الوارثة بالابن subclass وتسمى الفئة الموروثة بالأب supercalss – لكي ترث فئة معينة فئة معرفة سلفا يتم تحديد هذه العلاقة بالكلمة المحجوزة extends التي تظهر بعد اسم الفئة مباشرة يليها اسم الفئة التي سترثها هذه الفئة – يجب أن تكون الفئة الوارثة على علم باسم ومحتويات ووظائف الفئة الأب، على عكس الفئة الأب والتي قد لا تعلم بان هناك فئة قد ورثتها

وقدمت الوحدة برنامج 2 –2 –1 وخلصنا من هذا المثال أو البرنامج إلى أن الفئة الابن يمكن أن :

– تستخدم جميع الخصائص والدوال والمشيدات المعرفة بالفئة الأب.

– تعيد تعريف الدوال لاستخدامها بصورة أفضل (overriding).

3– تضيف مكونات جديدة على المكونات الموروثة.

وتطرقت الوحدة لثلاثة طرق لتعريف المتغيرات والدوال كما قامت بشرح لكل معرف منهم في مفهوم الوراثة وهي : Protected - Private - Public

وأوضحت الوحدة أن هذه المعرفات الثلاثة عبارة عن طرق للتحكم في امكانيات الوصول والتعامل مع المتغيرات والدوال المعرفة بداخل الفئات. وذكرت أيضاً انه يمكن كذلك التحكم في توريث الفئات والدوال عن طريق استخدام الكلمة المحجوزة final، و استخدامات final هي:

- المتغير المعرف final لا يمكن تغيير قيمته، أي انه ثابت.

- الفئة المعرفة final لا يمكن الوراثة منها، أي أنها لا يمكن أن تكون أب لفئة أخرى.

- الدالة المعرفة final في فئة لا يمكن إعادة تعريفها (override) في الفئة الابن، بل يجب استخدامها كما هي. إن الدوال المعرفة private أو static في اي فئة أب تكون تلقائيا final لأنها أيضا لا يمكن إعادة تعريفها.

كما بينت الوحدة كيفية منع إنشاء الكائنات من نوع فئة محددة وذلك بتعريفها من نوع abstract والتي تدل على ان هذه الفئة مصممة لاغراض الوراثة ولا يمكن اشتقاق كائنات منها، وبينت أنه إذا احتوت فئة على دالة أو أكثر abstract يجب تعريف الفئة على أنها abstract. تناولت الوحدة استخدام الواجهات Interfaces وهو حل وضعته لغة جافا لتلافي عيب الوراثة الوحيدة، وأوضحت أن الـ Interface هو هيكل يشبه في تركيبه الفئة إلا أن جميع دواله خالية من التعليمات وجميع بياناته ثوابت، ويمكن للفئة أن تستفيد من الخصائص المعرفة بعدة interfaces ولكن لا يطلق عليها وراثة بل تسمى

تعريف implementation، و يمكن تعريف أي عدد من الواجهات بالفئة الواحدة، ويتم تعرف جميع دوال الواجهات داخل الفئة. يمكن كذلك الجمع بين وراثة فئة وتعريف واجهة أو أكثر، وبينت الوحدة أن استخدام الواجهات جعل بالإمكان استخدام الخصائص الموجودة في أكثر من هيكل. كما أوضحت الوحدة أن مفهوم الواجهات لايقتصر على تعريف اسماء الدوال والثوابت لاستخدامها داخل فحسب بل تستخدم كواجهة بين الكائن ومستخدم هذا الكائن، فتحدد له الجزء المصرح له باستخدامه من الكائن، و تمنعه من الوصول إلى بقية محتويات الكائن

تناولت الوحدة مجموعات الفئات Packages وأوضحت أن مجموعة الفئات والواجهات التي تنتمى لنفس التطبيق توضع في وحدة تسمى package، وتحفظ في ملف يحمل اسم الـ package التي تنتمى إليها فئة معينة. ويتم كتابة الكلمة المحجوزة package في بداية الملف الذي يحتوى على الفئة ثمّ كتابة اسم الـ package

ويلي ذلك تعريف الفئة أو الواجهة بالطريقة المعروفة. يوجد عدد كبير من الـ packages المعرفة في لغة جافا ليستعين المبرمج بفئاتها وواجهاتها عند الحاجة وهذه يتم تضمينها داخل أي برنامج جافا دون الحاجة لكتابة عبارة صريحة. أما الـ packages الأخرى فيجب تضمينها في البرنامج بعبارة صريحة للتمكن من استخدامها، ويكون ذلك قبل تعريف الفئة .

لمحة مسبقة عن الوحدة التالية

عزيزي الدارس،

الوحدة التالية تعرفنا على الأدوات التي توفرها لغة جافا للرسم والتلوين على شاشة البرنامج، كما نستعرض فيها مجموعة من أدوات لغة جافا الخاصة بتصميم واجهات المستخدم أو شاشات البرنامج.

مسرد المصطلحات

البرمجة بالكائنات Object Oriented Programming Languages:

تقوم البرمجة بالكائنات على مبدأ أن كل فكرة أو موضوع في النظام المعين هو عبارة عن كائن object له صفات properties وسلوك behavior يظهر بها في النظام ويتفاعل عن طريقها مع الكائنات الأخرى في النظام. وتقوم على مبدأ أن النظام هو مجموعة من الكائنات التي تحتوي على صفاتها الخاصة والتي لا تسمح الكائنات الأخرى بالوصول إليها وهذه الكائنات تتفاعل مع بعضها البعض بواسطة طرق محددة سلفا وهي الدوال الخاصة بالكائن.

الصنف class

هو عبارة عن هيكل برمجي يحتوي على بيانات attributes ودوال methods تصف معا الشكل الذي ستكون عليه الكائنات عند تشغيل البرنامج.

لغات الوراثة المتعددة multiple inheritance

وفيها يمكن للفئة أن ترث خصائص أكثر من فئة واحدة. وميزة هذا النوع من الوراثة هو الاستفادة من خصائص عدد أكبر من الفئات، وعيبها صعوبة إدارة الكائنات وتحديد صلاحياتها.

الواجهات Interface

هو هيكل يشبه في تركيبه الفئة، إلا أن جميع دواله خالية من التعليمات وجميع بياناته ثوابت .

المراجع

1. Y. D. Liang
   Introduction to Java Programming, Third Edition
   ISBN: 0-13-031997-X
   Prentice Hall 2001
2. Deitel and Deitel
   Java How to Program, Third Edition
   ISBN: 0-13-012507-5
   Deitel 2000
3. Deitel and Deitel
   Java How to Program, Fifth Edition
   ISBN: 0-13-101621-0
   Deitel 2003
4. Stephan J. Chapman
   Introduction to Java
   ISBN: 0-13-919416-9
   Prentice Hall 1999
5. نعمة عزالدين محمد عثمان

   البرمجة بلغة جافا للمبتدئين

   الإصدارة الثانية 2007م

الوحدة الثالثة

Graphics and Graphical User InterFace(GUI)



محتويات الوحدة

الموضوع	الصفحة
المقدمة	**179**
تمهيد	**179**
أهداف الوحدة	**180**
1.أدوات الرسم والتلوين في جافاJava Graphics	**181**
1.1 Class Graphics	182
2.1 Class Color	183
3.1 Class Font	184
2. رسم الأشكال الهندسية باستخدام الدوال في Class Graphics	**186**
3. واجهة المستخدم الرسومية Graphical User Interface (GUI)	**196**
1.3 الأزرار Buttons	204
2.3 القوائم JComboBox وJList	211
3.3 Panel	217
الخلاصة	**224**
لمحة مسبقة عن الوحدة الدراسية التالية	**224**
مسرد المصطلحات	**225**
المراجع	**227**

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك في الوحدة الثالثة من مقرر "البرمجة بلغة جافا" والتي تحمل عنوان" Graphics and Graphical User InterFace(GUI)". القسم الأول من الوحدة يتناول " Java Graphics "حيث توفر لغة جافا مجموعة كبيرة وغنية من أدوات الرسم والتلوين، ويتناول القسم برنامجاً لكيفية رسم نافذة على الشاشة ، وفي القسم تجد أيضاً Class Color الذي يوفر إمكانية تحديد وتغيير ألوان الأشكال والخلفيات وكذلك تجد في هذا القسم الـــ Class Font الذي يساعدنا في اختيار نوع وحجم وشكل خط الرسم على الشاشة.

القسم الثاني في الوحدة يبحث في رسم الأشكال الهندسية باستخدام الدوال في Class Graphics حيث يعدد القسم عدداً من الدوال التي تستخدم في رسم الأشكال الهندسية على نافذة البرنامج.

القسم الثالث من الوحدة يتناول واجهة المستخدم الرسومية حيث توفر لغة جافا مجموعة كبيرة من الفئات والدوال لتصميم الواجهات الرسومية.

أهداف الوحدة



عزيزي الدارس ، بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

- **تذكر** الأدوات التي توفرها لغة جافا في الرسم والتلوين على شاشة البرنامج، وكذلك كيفية استخدام هذه الأدوات.
- **تشرح** كيفية استخدام دوال Class Graphics لرسم وتلوين الأشكال الهندسية المختلفة.
- **تستنتج** الفرق بين الدالة Draw Polygon والدالة Draw Polyline.
- **تعطي** أمثلة عن كيفية إنشاء Labels بطرق مختلفة عن طريق استخدام الـــ Constructors المختلفة.
- **تعدد** بعض الدوال المستخدمة في رسم الأشكال الهندسية المختلفة على نافذة البرنامج.

توطئة

عزيزي الدارس،

اقتصرت جميع برامج جافا التي قمنا بكتابتها وتنفيذها حتى الآن على ظهور مخرجات البرامج والتفاعل معها عن طريق محث التعليماتcommand prompt، الشاشة التي نعطي عبرها الأوامر بتشغيل برنامج معين أو كتابة مدخلات محددة أو قراءة مخرجات البرنامج. ولكن عملياً، تطبيقات الحاسوب التي نشاهدها ونستخدمها في حياتنا تظهر بصورة أكثر وضوحاً وأجمل شكلاً وأسهل استخداما وأقرب لفهم غير المختص بالحاسوب. فنجد الأشكال والرسوم والألوان المختلفة التي تحسن شكل البرنامج للمستخدم، كما نجد الأدوات المختلفة مثل القوائم والأزرار التي تسهل تفاعل المستخدم مع البرنامج. يتكون هذا الباب من فصلين، الفصل الأول Java Graphics يعرفنا على الأدوات التي توفرها لغة جافا للرسم والتلوين على شاشة البرنامج. الفصل الثاني Graphical User Interface(GUI) يستعرض مجموعة من أدوات لغة جافا الخاصة بتصميم واجهات المستخدم أو شاشات البرنامج، والتي تساعد المستخدم على التفاعل مع البرنامج بصورة أسهل وأبسط ولا تستدعي أي معرفة بالبرمجة ولغاتها، وهو ما نحتاجه في برامج الحاسوب اليوم.

1.أدوات الرسم والتلوين في جافا Java Graphics

توفر لغة جافا مجموعة كبيرة وغنية من أدوات الرسم والتلوين موجودة في package java.awt وقبل أن نتطرق إليها يجب أولا أن نتعرف على JFrame class والموجود في package javax.swing. يظهر أي class يرث من class JFrame تلقائيا على الشاشة على شكل نافذة مستطيلة عند التنفيذ، ولهذا الـ class خصائص ودوال كثيرة للتحكم في حجم وموقع وظهور وتنسيق هذه الشاشة. نتعرض في هذا القسم لبعض خصائص class JFrame، ونتعرض للبعض الآخر خلال القسم الثالث. تكمن أهمية هذا الـ class بالنسبة للرسم في أنه يمثل اللوحة التي سيتم عليها الرسم أو التلوين، وفي إمكانية نداء الدالة

paint تلقائياً عند توليد كائن من فئة ترث خصائص class JFrame. يوضح البرنامج التالي كيفية رسم نافذة على الشاشة تحتوي على العبارة Java Graphics.

```
import java.awt.*;
import javax.swing.*;
public class FirstGraphics extends JFrame{
        public FirstGraphics() {
                super("My first window");
                setSize(200, 100);
                setVisible(true);
        }
        public void paint(Graphics g) {
                g.drawString(" Java Graphics ", 50, 50);
        }
        public static void main(String args[ ] ) {
                FirstGraphics fg = new FirstGraphics(  );
        }
}
```

برنامج 3-1-1

Class Graphics 1.1

عند تعريف Object من class FirstGraphics، يتم رسم نافذة صغيرة على الشاشة ويتم الحصول على مواصفات هذه النافذة من الـ constructor، فيمكن كتابة عنوان لهذه النافذة باستخدام constructor الـ class supper عن طريق نداء super() وتمرير عنوان النافذة كوسيط، يمكن كذلك تحديد أبعاد النافذة باستخدام الدالة setSize والتي تستقبل طول وعرض النافذة، والوحدة المستخدمة هي Pixel، تستخدم الدالة setVisible لتحديد ما إذا كنا نرغب في إظهار النافذة التي تم إعدادها أم لا، فالوسيط true يعني إظهارها فورا والوسيط false يعني عدم إظهارها حالياً وربما تم إعدادها للظهور لاحقا خلال التنفيذ، الدالة paint هي دالة يتم نداؤها تلقائيا بعد تنفيذ الـ constructor مباشرة،

وتحتوي على جميع تعليمات البرنامج الخاصة بالرسم والتلوين، وذلك باستخدام object من class Graphics والذي يحتوي على جميع الدوال الخاصة بالرسم والتلوين، نجد في هذا المثال الدالة drawString والتي تستخدم لرسم نص على النافذة، وهو في هذا المثالJava Graphics، ويحتل الرقمان التاليان اللذان تستقبلهما الدالة الأبعاد التي يبدأ عندها رسم النص.



مخرجات برنامج 3-1-1

أسئلة تقويم ذاتي



وضح أهمية الـ JFrame class بالنسبة للرسم؟

Class Color 2.1

تحتوي package java.awt إضافة إلى class Graphics على Class color والذي يوفر إمكانية تحديد وتغيير ألوان الأشكال والخلفيات، ويمكن إعداد لون معين لاستخدامه في الرسم بإحدي طريقتين:

1. باستخدام الألوان التي يحتوي عليها Class color أصلاً على شكل ثوابت.

```
Color  c = color. RED;
```

ومن الألوان الموجودة أيضاً WHITE, BLACK, YELLOW, GRAY, PINK, GREEN, BLUE وغيرها.

2. باستخدام نظام RGB (Red- Green- Blue) للألوان، والذي ينص على أن كل لون هو ناتج مزج نسبة معينة من اللون الأحمر والأخضر والأزرق، تتراوح هذه النسبة بين الصفر و 225 لكل لون، فمثلاً يتكون اللون الأحمر من 255 من اللون الأحمر و0 من اللونين الأخضر والأزرق، ولذلك فإن مكونات اللون الأحمر هي (255.0.0) وبنفس الطريقة ينتج اللون الأبيض من مزج الألوان الثلاثة بنسبة 225 ليكون رمز اللون الأبيض هو (225.255.255). ونجد أن رمز اللون البرتقالي هو (255,200,0) لاستخدامنا الألوان الرئيسة بهذه النسب للحصول على اللون البرتقالي وهكذا. وبالتالي يمكننا الحصول على عدد كبير من الألوان بمختلف الدرجات باستخدام نظام RGB ويستخدم الكثير من برامج الرسم بالحاسوب هذا النظام.

```
color   c = new  color (128, 64, 20);
```

Class Font 3.1

يساعدنا Class Font على اختيار نوع وحجم وشكل خط الرسم على الشاشة.

```
Font  font = new  font (“ times new roman “, Font.PLAIN, 20);
```

لإنشاء Font، نحتاج لثلاثة معطيات، أولها اسم الخط المستخدم، ويمكن أن يكون أي خط من الخطوط المتعارف عليها من قبل نظام التشغيل الذي يستخدمه الحاسوب، ثانياً شكل الخط، فيجب تحديد ما إذا كان الخط عادي plain أو مائل italic أو عريض bold، وذلك باستخدام الثوابت الموجودة بـ Class font، وأخيرا يجب تحديد حجم خط الكتابة.

يوضح المثال التالي استخدام الألوان والخطوط المختلفة لرسم عبارات على الشاشة.

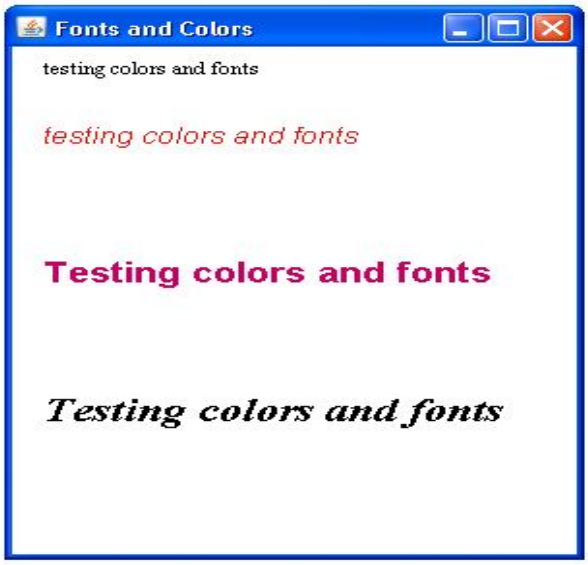
```
import java.awt.*;
import javax.swing.*;
public class FontsAndColors extends JFrame  {
        public FontsAndColors( ) {
```

```
        super( " Fonts and Colors ") ;
        setSize(300, 400);
        setVisible(true);

    }
    public void paint(Graphics  g) {
        Color   color = Color.BLACK;
        Font  font  = new Font("serif", Font.PLAIN, 12);
        g.setColor(color);
        g.setFont(font);
        g.drawString("testing colors and fonts", 20, 50);
        g.setColor(Color.RED);
        g.setFont(new Font("monospaced ", Font.ITALIC, 16));
        g.drawString("testing colors and fonts", 20, 100);
        g.setColor(new Color(200, 0, 100));
        g.setFont(new Font("sans serif", Font.BOLD, 20));
        g.drawString("Testing colors and fonts", 20, 200);
        g.setColor(color);
        g.setFont(new Font("times new roman", Font.BOLD +
    Font.ITALIC, 24));
        g.drawString("Testing colors and fonts", 20, 300);
    }
    public static void main(String args[   ]) {
        new FontsAndColors(   );
    }
}
```

برنامج 3-2-2

2. رسم الأشكال الهندسية باستخدام الدوال في Class Graphics

تستخدم الدالة setColor الموجودة في Class Graphics لتحديد اللون الذي سيتم استخدامه للرسم على النافذة وتستقبل اللون المطلوب كوسيط، كذلك تستخدم الدالة setFont لتحديد خصائص الخط الذي ستظهر به الكتابة. نلاحظ أنه يمكن للكتابة أن تكون Bold و Italic معاً، وذلك باستخدام علامة الجمع.



مخرجات برنامج 3-1-2

أسئلة تقويم ذاتي

1- يمكن إعداد لون معين لاستخدامه في الرسم بإحدي طريقتين ما هما ؟

2- نظام RGB للألوان، ينص على أن كل لون هو ناتج مزج نسبة معينة من اللون .....؟ و.....؟ و.....؟، تتراوح هذه النسبة بين .....؟ و ..... .

3- لإنشاء Font، نحتاج لثلاثة معطيات. ما هي ؟

يوفر class graphics عدداً من الدوال لرسم الأشكال الهندسية المختلفة ويوضح الجدول التالي بعض الدوال المختلفة ووظيفتها والوسائط التي تستقبلها.

اسم الدالة	الوظيفة	الوسائط
Drawline(x1, y1, x2, y2)	رسم خط مستقي	x1: الإحداثي السيني لنقطة بداية الخط. y1: الإحداثي الصادي لنقطة بداية الخط. x2: الإحداثي السيني لنقطة نهاية الخط y2: الإحداثي الصادي لنقطة نهاية الخط.
drawRect(x, y, width, height) FillRect(x, y, width, height) Draw3Drect(x, y, width, height, b)	رسم مستطيل رسم مستطيل ممتلئ	x: الإحداثي السيني لنقطة أعلى شمال

Fill3DRect(x, y, width, height, b)	رسم مستطيل ثلاثي الأبعاد. رسم مستطيل ثلاثي الأبعاد ممتلئ،	المستطيل y: الإحداثي الصادي للنقطة أعلى شمال المستطيل. width: عرض المستطيل. height: ارتفاع المستطيل. b: متغير Boolean يأخذ القيمة true ليبدو المستطيل مرتفع القيمة false ليبدو منخفضاً.
drawOval(x, y, width, height) FillOval(x, y, width, height)	رسم شكل بيضاوي رسم شكل بيضاوي ممتلئ	x: الإحداثي السيني للنقطة أعلى شمال المستطيل الذي يمس الشكل البيضاوي أضلاعه من الداخل y: الإحداثي الصادي للنقطة أعلى شمال المستطيل الذي يمس

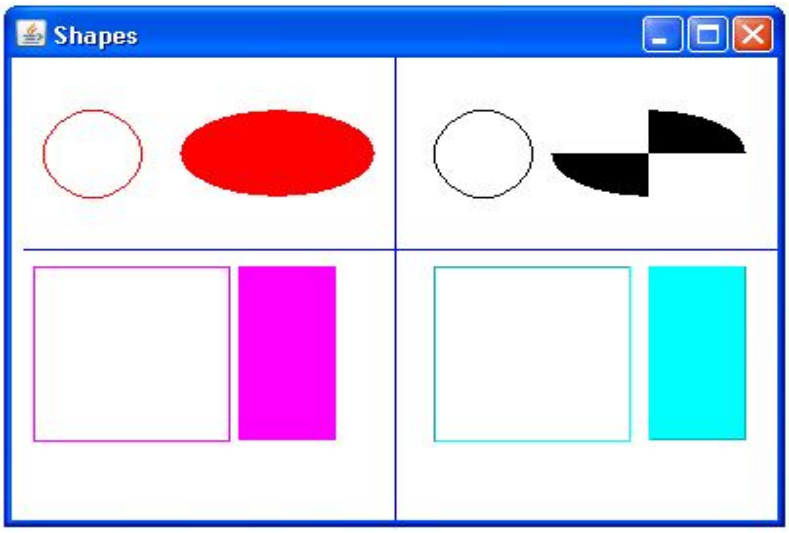
		الشكل البيضاوي أضلاعه من الداخل. width: عرض الشكل البيضاوي. height: ارتفاع الشكل البيضاوي
drawArce(x, y, width, height, start, angle) FillArc(x, y, width, height, start, angle)	رسم قوس من دائرة رسم قطاع دائرة ممتلئ	x :الإحداثي السيني لمركز الدائرة. y: الإحداثي الصادي لمركز الدائرة. width: عرض الدائرة. height: ارتفاع الدائرة. start: الزاوية التي يبدأ منها رسم القوس أو القطاع من (5 إلى 360). angle: قيمة الزاوية المركزية واتجاه رسم القوس أو القطاع (من 360- إلى 360).

المثال التالي يستخدم بعض هذه الدوال لرسم الأشكال الهندسية على نافذة البرنامج:

```
import java.awt.*;
import javax.swing.*;
public  class Shapes extends JFrame{
        public Shapes( ) {
                super("Shapes");
                setSize(400,300);
                setVisible(true);
        }
        public  void  paint(Graphics  g) {
                g.setColor(Color.BLUE);
                g.drawLine(10, 140, 400, 140);
                g.drawLine(200, 30, 200, 300);

                g.setColor(Color.RED);
                g.drawOval(20, 60, 50, 50);  //  circle
                g.fillOval(90, 60, 100, 50);

                g.setColor(Color.MAGENTA);
                g.drawRect(15, 150, 100, 100);  // square
                g.fillRect(120, 150, 50, 100);

                g.setColor(Color.CYAN);
                g.draw3DRect(220, 150, 100, 100, false);
                g.Fill3DRect(330, 150, 50, 100, true);

                g.setColor(Color.BLACK);
                g.drawArc(220, 60, 50, 50, 0, 360);  // circle
                g.fillArc(280, 60, 100, 50, 0, 90);
                g.fillArc(280, 60, 100, 50, 270, -90);
        }
        public static void main(String args[ ]){
                new  Shapes( );
        }
}
```

برنامج 3-1-3

نلاحظ في هذا المثال أننا رسمنا دائرة بطريقتين: عن طريق الدالة DrawOval ورسم شكل بيضاوي عرضه يساوي ارتفاعه، وعن طريق الدالة DrawArc ورسم قطاع زاويته المركزية 360. مستخدمين الدالة DrawArc، قمنا برسم قطاع دائرة يبدأ من الزاوية صفر، ويفتح بزاوية 90 في الاتجاه الموجب، وقطاع دائرة آخر يبدأ من الزاوية 270 ويفتح بزاوية 90 في الاتجاه السالب، أي يبدأ القطاع عند الزاوية 270 وينتهي عند الزاوية 180.



مخرجات برنامج3-1-3

أسئلة تقويم ذاتي



أكمل الجدول التالي الذي يوضح بعض الدوال المختلفة ووظيفتها والوسائط التي تستقبلها.

اسم الدالة	الوظيفة	الوسائط
Drawline(x1, y1, x2, y2)	.....؟	x1: الإحداثي السيني لنقطة بداية الخط. y1: الإحداثي الصادي لنقطة بداية الخط. x2: الإحداثي السيني لنقطة نهاية الخط. y2: الإحداثي الصادي لنقطة نهاية الخط.
.....؟ FillRect(x, y, width, height) Draw3Drect(x, y, width, height, b) .....؟	رسم مستطيل .....؟ رسم مستطيل ثلاثي الأبعاد رسم مستطيل ثلاثي الأبعاد ممتلئ	x: الإحداثي السيني للنقطة أعلى شمال المستطيل y: الإحداثي الصادي للنقطة أعلى شمال المستطيل width: عرض المستطيل. height: ارتفاع المستطيل. b: متغير Boolean يأخذ القيمة true ليبدو المستطيل مرتفع القيمة false ليبدو منخفضاً.

تابع أسئلة تقويم ذاتي



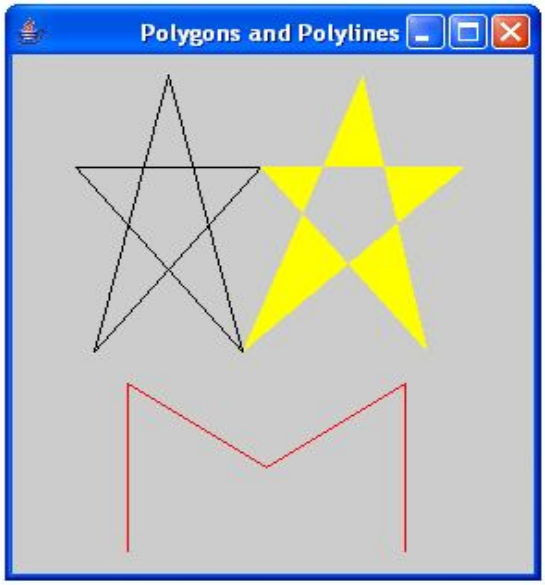
drawOval(x,y,width, height)……..?	…… ……..؟ رسم شكل بيضاوي ممتلئ	x: الإحداثي السيني للنقطة أعلى شمال المستطيل الذي يمس الشكل البيضاوي أضلاعه من الداخل. y: الإحداثي الصادي للنقطة أعلى شمال المستطيل الذي يمس الشكل البيضاوي أضلاعه من الداخل. width: عرض الشكل البيضاوي. height: ارتفاع الشكل البيضاوي
drawArce(x,y,width,height, start, angle) FillArc(x,y,width,height, start, angle)	……..؟ ……..؟	x :الإحداثي السيني لمركز الدائرة. y: الإحداثي الصادي لمركز الدائرة. width: عرض الدائرة. height: ارتفاع الدائرة. start: الزاوية التي يبدأ منها رسم القوس أو القطاع من (5 إلى 360). angle: قيمة الزاوية المركزية واتجاه رسم القوس أو القطاع (من 360- إلى 360)

يوضح لنا هذا المثال كيفية استخدام دوال class Graphics لرسم وتلوين الأشكال الهندسية المنتظمة. هناك دوال أخرى تستخدم لرسم الأشكال الأكثر تعقيدا مثل المضلعات المختلفة والنجوم وأي خطوط متصلة لتكون شكلاً مفتوحاً أو مغلقاً. تستخدم الدالة drawPolygon لرسم الأشكال المغلقة كثيرة الأضلاع وتستقبل مصفوفة بالإحداثي السيني لجميع النقاط المكونة للشكل، ومصفوفة ثانية بالإحداثي الصادي لجميع النقاط المكونة للشكل، ثم عدد هذه النقاط. تقوم الدالة بتوصيل النقاط مع بعضها البعض بخطوط مستقيمة حسب ترتيب النقاط، ثم وصل النقطة الأخيرة مع الأولى للحصول على الشكل المغلق، تستخدم الدالة fillPolygon لملئ الشكل المغلق باللون المعين، هناك أيضا الدالة drawPolyline، و يشبه الـ polyline الـ polygon إلا أن الـ polyline هو شكل مفتوح أي أنه مجموعة من النقاط المتصلة يبعضها عن طريق خطوط مستقيمة وتستقبل الدالة كذلك مصفوفة الإحداثي السيني للنقاط ومصفوفة الإحداثي الصادي للنقاط وعددها، ولا تقوم بإغلاق الشكل في النهاية، يوضح المثال التالي استخدام الدالة drawPolygon و drawPolyline.

```
import java.awt.*;
import javax.swing.*;
public class PolygonsAndPolylines extends JFrame{
        public PolygonsAndPolylines( ){
                super("Polygons and Polylines");
                setSize(300, 200);
                setVisible(true);
        }
        public void paint(Graphics g){
                int  x1[ ] = {50, 125, 100, 75, 150};
                int  y1[ ] = {70, 200, 35, 200, 70};
                g.setColor(Color.BLACK);
```

```
                g.drawPolygon(x1, y1, 5); // star

                int  x2[ ] = {150, 240, 205, 140, 260};
                int  y2[ ] = {90, 200, 35, 200, 90};
                g.setColor(Color.YELLOW);
                g.fillPolygon(x2, y2, 6); // filled star

                int x3[ ] = {50, 50, 125, 200, 200};
                int y3[ ] = {400, 300, 350, 300, 400};
                g.setColor(Color.RED);
                g.drawPolyline(x3, y3, 7);  // M
        }
        public static void main(String args[ ]){
                new PolygonsAndPolylines( );
        }
}
```

برنامج 3-1-4



مخرجات برنامج3-1-3

لاحظ الفرق بين الـــ polygon و الـــ polyline، حيث لا يتم إغلاق الشكل عنـــد رســـم polyline عن طريق وصل النقطة الأخيرة مع الأولى كما في polygon.

باستخدام دوال الرسم مع المدى الواسع للألوان التي توفرها جافـــا، يمكـــن أن تصـــمم مختلف الشاشات واللوحات، تستخدم دوال class Graphics التي تطرقنـــا إليهـــا لرســـم الأشكال البسيطة والتصاميم المختصرة، وتوجد دوال أخرى تســـتخدم لرســـم وتصـــميم الشاشات المعقدة مثل استخدام التظليل واختلاف سمك فرشاة الرسم وتصـــميم وحـــدات صغيرة يتم تكرارها للتلوين بدلا عن استخدام لون واحد بسيط. جميع هـــذه المميـــزات موجودة في class Graphics2D أو الرسم ببعدينtow dimensional، وتحتوي على كـــل الأدوات المناسبة للرسام المتمكن لتطبيق فنه على شاشات برامج جافا.

أسئلة تقويم ذاتي

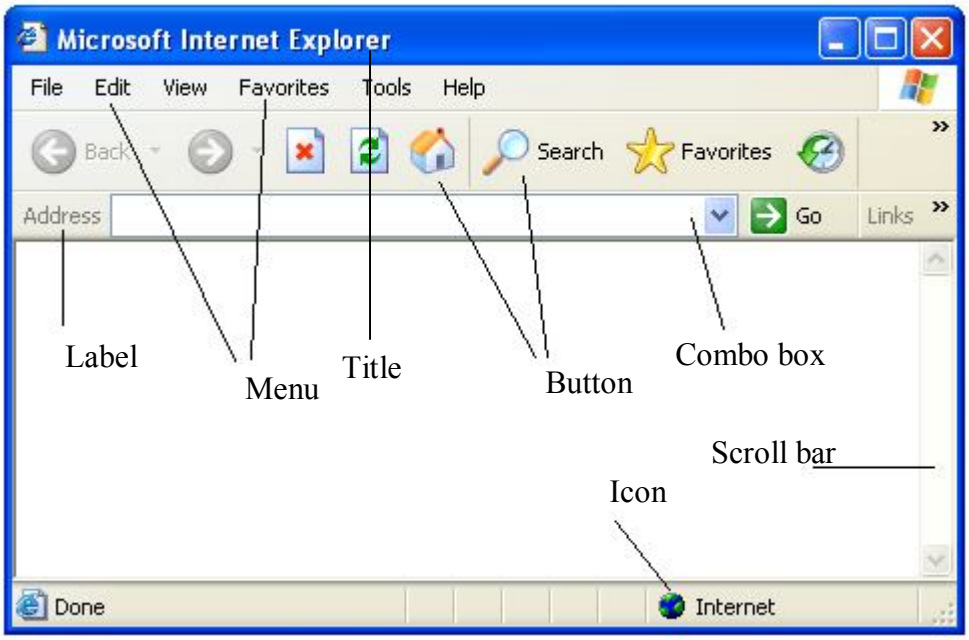


1 – وضح الفرق بين الـــ polygon و الـــ polyline ؟

3. واجهة المستخدم الرسومية Graphical User Interface (GUI)

تعتبر واجهة المستخدم الرسومية من أهم الأدوات والأكثر استخداماً في لغات البرمجـــة لجميع التطبيقات، فهي تمكن المبرمج من تصميم واجهة رسومية للبرنامج تسهل وتبسط من تعامل المستخدم مع البرنامج. تحتوي أغلب برامج الحاسوب التي نراها اليوم علـــى واجهات رسومية، وهي أي شاشة أو نافذة نراها عند تشغيل برنامج معين، وقد يحتـــوي البرنامج الواحد على عدد من الواجهات أو الشاشات، ويمكن أن تحتوي الواجهة علـــى

معلومات عن البرنامج أو يمكن أن تحتوي على منطقة لإدخال البيانات، ويتم تصميم البرنامج بحيث تظهر كل خطواته ومعالجته على الواجهة ليتمكن المستخدم من متابعة البرنامج بدراية تامة ودون ملل. فمتصفح الإنترنت يحتوي على واجهة تسمح للمستخدم بإدخال العنوان المطلوب، وتحتوي على أزرار متعددة الاستخدامات، والمنطقة التي تظهر فيها الصفحة هي منطقة المخرجات ويمكن للمبرمج متابعة تحميل الصفحة بمراقبة شريط التحميل أسفل الصفحة.



شكل 3-2-1: مثال لواجهة رسومية للمستخدم: متصفح الإنترنت

توفر لغة جافا مجموعة كبيرة وغنية من الفئات والدوال لتصميم الواجهات الرسومية ولتفاعل البرامج مع المستخدم وهي موجودة في package java.awt و package javax.swing. في القسم السابق تعرفنا على class JFrame وكيف أنه يظهر على الشاشة ويسمح للمبرمج بتصميم الأشكال عليه. يستخدم class JFrame كأساس لتصميم واجهات المستخدم الرسومية وتضاف جميع مكونات الواجهة إليه، نتعرف خلال هذا القسم على أهم هذه المكونات واستخداماتها وكيف يتعامل معها المستخدم.

يعتبر class Container والموجود في package java.awt من أهم الـــ classes التـــي تستخدم إلى جانب class JFrame، وتكمن أهميته في أنه يتم ربطه بالـــJFrame ثم تـــتم إضافة جميع مكونات النافذة (componets) إليه.

في المثال التالي نوضح كيفية إعداد شاشة تحتوي على عبارات وصور باستخدام class JLabel والذي يستخدم لكتابة نص أو إظهار صور على نافذة البرنامج.

```
import java.awt.*;
import javax.swing.*;
public class LabelTest extends JFrame {
        public LabelTest( ) {
                super ("Label Test");
                Container container = getContentPane ( );
                container.setLayout(new FlowLayout( ));
                JLabel label1 = new JLabel ("This is a label");
                container.add(label1);
                Icon image = new ImageIcon ("image.gif");
                JLabel label2 = new JLabel (image);
                container.add (label2);
                JLabel label3 = new JLabel ( );
                Label3.setText ("This is another label");
                container.add (label3);
                setSize (200, 100);
                setVisible (true);
        }
        public static void main ( String args[ ] ) {

                new LabelTest ( );

        }

}
```

برنامج 3-2-1

يبدأ الـــ constructor بتحديد عنوان النافذة، ثم تعريف الوعاء container الذي ستضاف إليه المكونات فيما بعد وربطها بالـــ JFrame باستخدام الدالـــة getContentPane. يـــتم

تحديد طريقة تنظيم المكونات على النافذة بواسطة الدالة setLayout، وسنتعرف على الطرق المختلفة لتنظيم الشاشة في نهاية الفصل. بعد ذلك يتم تعريف المكونات المناسبة التي نرغب في إضافتها إلى النافذة. وفي هذا المثال تم تعريف JLabel لطباعة نص على الشاشة، حيث نقوم بتمرير هذا النص إلى الـ constructor الخاص بـ JLabel class ثم يضاف إلى الـ container باستخدام الدالة add. ولعرض صورة على الشاشة، نقوم بتجهيزها أولاً باستخدام reference من class Icon و object من class ImageIcon، مع توضيح اسم الملف الذي يحتوي على الصورة المطلوبة لحظة إنشاء الـ object، وهو في هذا المثال الملف image.gif والذي نفترض وجوده في نفس المسار الذي يوجد فيه هذا البرنامج. بعدها يتم إنشاء JLabel باستخدام الكائن image الذي يحتوي على الصورة، ثم إضافة الـ JLabel إلى الـ container. وأخيراً يتم تحديد حجم الـ JFrame باستخدام الدالة setSize وعرضه عن طريق تمرير القيمة true إلى الدالة setVisible.



مخرجات برنامج 3-2-1

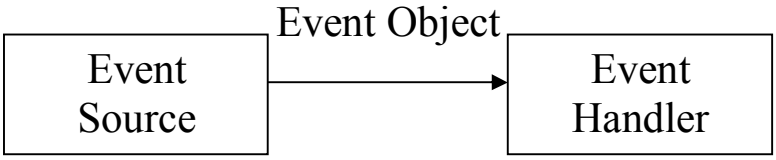
نلاحظ من المثال أن هناك عدد من الطرق يمكن بواسطتها إنشاء labels عن طريق استخدام الـ constructors المختلفة. فهناك constructor يستقبل نصاً ليظهر على الشاشة، وآخر يستقبل icon لعرضها بواسطة JLabel، وثالث يستقبل نصاً وصورة معاً. وهناك الخيار لتعريف JLabel باستخدام الدالة setText و setIcon اللتين تستقبلان نصاً و icon على التوالي.

نستخدم الـــ labels عموماً لكتابة العناوين والتعليمات على نافذة البرنامج، وقـــد تكـــون إحدي الوسائل المستخدمة لعرض مخرجات البرامج. يستخدم حقـــل الـــنص text filed لاستقبال المدخلات من المستخدم، فهي تسمح لـــه باســـتخدام لوحـــة المفـــاتيح وكتابـــة النصوص المختلفة بداخلها. يوفر class JTextField إمكانيـــة تخاطـــب المســـتخدم مـــع البرنامج، حيث يسمح له بإدخال القيم المختلفة ليتعامل معها البرنامج. وقبـــل أن نشـــرح برنامجاً يتعامل مع class JTextField، نريد أن نوضح طريقـــة تفاعـــل البرنـــامج مـــع الأحداث الصادرة من المستخدم، والتي تعرف بمعالجة الأحداثEvent Handling. الفكرة في معالج الأحداث هي أن يقوم المستخدم بعمل حدث معين في البرنامج مثـــل ضـــغطة زر أو طباعة حرف أو ضغط Enter أو تحريك الـــ mouse أو أي حدث آخر، فيقوم البرنامج بتنفيذ رد فعل محدد رداً على فعل المستخدم. ولعملية معالجة الأحـــداث ثلاثـــة مكونات:

1- مصدر الحدث event source: ويمكن أن يكون أي مكون component فـــي نافـــذة البرنامج أومفتاح في لوحة المفاتيح أو الـــ mouse.

2- كائن الحدث event object: كل حدث معين في البرنامج ينشأ أو يتولد عنـــه object محدد، يبحث هذا الـــ object عن المنطقة في البرنامج التي ستعالج الحدث وتقوم بـــرد الفعل المناسب.

3- معالجة الحدث event hander: وهو جزء من البرنامج – دالة عادة – يكـــون فـــي حالة استماع دائم لنوع محدد من الأحداث يعالجها و يقوم برد فعل محدد، ولذلك يعرف أيضا بالمستمع listener. يحدد المخطط أدناه علاقة هذه المكونات:



الشكل 3-2-2: Event Handling process

هنالك عدة أنواع من الأحداث Event مرتبطة بـــ component بعينه، وخـــلال الأمثلـــة التالية نتعرف على هذه الأحداث إلى جانب الـــ components المرتبطة بهـــا. أول هـــذه الأحداث هو Action Event، وإحدي أسباب تولده هو عند الضغط علـــى مفتـــاح enter بينما مؤشر الكتابة موجود بداخل text field. والمثال التالي يشرح ذلك.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TextFieldTest extends JFrame {
        Handler handler  = new Handler();
        JTextField text;
        public TextFieldTest() {
                super("TextField Test");
                Container container  = getContentPane( );
              container.setLayout( new FlowLayout( )),
                container.add(new JLabel("Enter your name and press enter"));
                text = new JTextField(20);
        text.addActionListener( handler );
        container.add( text );
        setSize(250, 100);
        setVisible(true) ;
        }
      public static void main(String args [ ] ) {
            new TextFieldTest();
      }
}
class Handler implements ActionListener {
        public void actionPerformed (ActionEvent event ){
         JOptionPane.showMessageDialog(null,  "Hello  "  +
event.getActionCommand());
        }
}
```

برنامج 3-2-2

يتم تعريف text field عن طريق إنشاء object من JTextFiled class والموجود في package javax.swing بالطول 20 ثم إضافته إلى الـ container باستخدام الدالة، add ويكون ذلك بعد أن يتم تحديد معالجة الحدث event handler وهو object من الـ class الذي يحتوي على الدالة التي ستنفذ تلقائياً عند حدوث الحدث. وفي حالة حدوث event من نوع ActionEvent يتم تنفيذ الدالة actionPerformed تلقائياً عند تولد هذا الحدث بشرط أن يتم ربط الـ handler مسبقاً مع الـ component، أي تحديد أن هذا المستمع ينتظر تولد الأحداث في هذا component، وذلك باستخدام الدالة addActionListener. العبارة:

```
text.addActionListener(handler);
```

تعني أن الـ object المسمى handler مشتق من class يحتوي على دالة actionPerformed تنتظر تولد أحداث من النوع ActionEvent من الـ JTextFiled المسمى text. ولكي يكون الـ object قادراً على أن يكون handler لأحداث من نوع ActionEvent يجب أن يكون implements ActionListener ويجب أن يحتوي على تعريف الدالة actionPerformed الموجودة في interface ActionListener والموجود بدوره في package java.awt.event بنفس الطريقة الموضحة. وبعد هذا، فإن الـ object handler يكون في حاله استماع دائم لأي حدث من نوع ActionEvent ينشأ من الـ text filed عن طريق الضغط على مفتاح enter بداخله، ومتى ما حدث ذلك، يتم تنفيذ الدالة actionPerformed تلقائياً.

كلما تم الضغط على مفتاح enter في وجود المؤشر داخل حقل النص text يتم تنفيذ الدالة actionPerformed الموجودة في class Handler، لأنه تم تعيينه كمستمع للأحداث من نوع ActionEvent الصادرة من الكائن text. عند حدوث الحدث، يتولد object من نوع ActionEvent وينتقل إلى الدالة actionPerformed التي يتم تنفيذ خطواتها. أي أننا

نكتب بداخل الدالة actionPerformed التعليمات التي نرغب في تنفيذها عند الضغط على enter من داخل حقل النص text. وفي هذا البرنامج نستخدم الدالة showMessageDialog التابعة لـ class JOptionPane والموجود في package javax.swing، وهي تقوم بإظهار رسالة على الشاشة تستقبلها كنص String، هذا النص عبارة عن Hello إي إضافة الاسم المكتوب بداخل حقل النص text والذي حصلنا عليه من الكائن event باستخدام الدالة getActionCommand، والتي تستخدم عموماً للحصول على النص المكتوب بداخل الـ component الذي حدث منه الحدث. نلاحظ وجود المستمع Handler في class مستقل عن بقية البرنامج، وهو ليس لازماً كما سنرى في الأمثلة القادمة.



مخرجات برنامج 3-2-2

أسئلة تقويم ذاتي



1- يعتبر class Container الموجود في package java.awt من أهم الـ classes التي تستخدم إلى جانب class Jframe ، وضح هذه الاهمية ؟

2- لعملية معالجة الأحداث ثلاثة مكونات ما هي ؟

3- في البرنامج 3-2-2 ماذا تعني العبارة أدناه :

text.addActionListener(handler); ?

4- وضح الفكرة في معالج الأحداث Event Handling ؟

1.3 الأزرار Buttons

من مكونات البرامج الأكثر أهمية واستخداماً في البرنامج الأزرار buttons، فهي تستخدم في البرامج لحث المستخدم لبداية عملية معينة. مثلاً في متصفح الإنترنت، نجد الزر Back للعودة إلى الصفحة السابقة والزر Stop لوقف التحميل، والزر Refresh لإعادة التحميل. وفي برنامج الآلة الحاسبة نجد زر يمثل كل رقم وكل عملية حسابية. إذن الزر هو مشغل لعملية معينة يضغط عليه المستخدم عندما يرغب في تنفيذ العملية. يستخدم class JButton لإضافة أزرار إلى الشاشة للقيام بعمليات محددة. يتم إنشاؤها ووضع اسم عليها باستخدام الـ constructor ثم إضافتها إلى الـ container. عند الضغط على زر معين، يتولد حدث من نوع ActionEvent، ولذلك لابد من وجود مستمع ينتظر حدوث الحدث للرد عليه. ولذلك يتم كتابة العبارات المطلوب تنفيذها عند الضغط على الزر بداخل الدالة actionPerformed. يوضح المثال التالي عدداً من الأزرار تقوم بوظائف مختلفة.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
```

```
public class ButtonTest extends JFrame  implements ActionListener {
        JPasswordField password;
        JButton ok, clear, exit;
        public ButtonTest() {
                super("Button Test");
                Container c = getContentPane( );
                c.setLayout(new FlowLayout( ));
                c.add(new JLabel("Enter Password ");
                password = new JPasswordField( );
                c.add(password);
                ok = new JButton("OK");
                ok.addActionListener(this);
                c.add(ok);
                clear = new JButton("Clear");
                clear.addActionListener(this);
                c.add(clear);
                exit = new JButton("Exit");
                exit.addActionListener(this);
                c.add(exit);
                setSize(350, 100);
                setVisible(true);
        }
        public void actionPerformed(ActionEvent event ) {
                if (event.getSource() = = exit)
                        System.exit(0);
                else
                if (event.getSource() = =clear)
                        password.setText(" ");
                else {    // ok
                        if (password.getText( ).equals("Sudan"))
                                JOptionPane.showMessageDialog(this,   "Password
                        is correct");
                        else
```

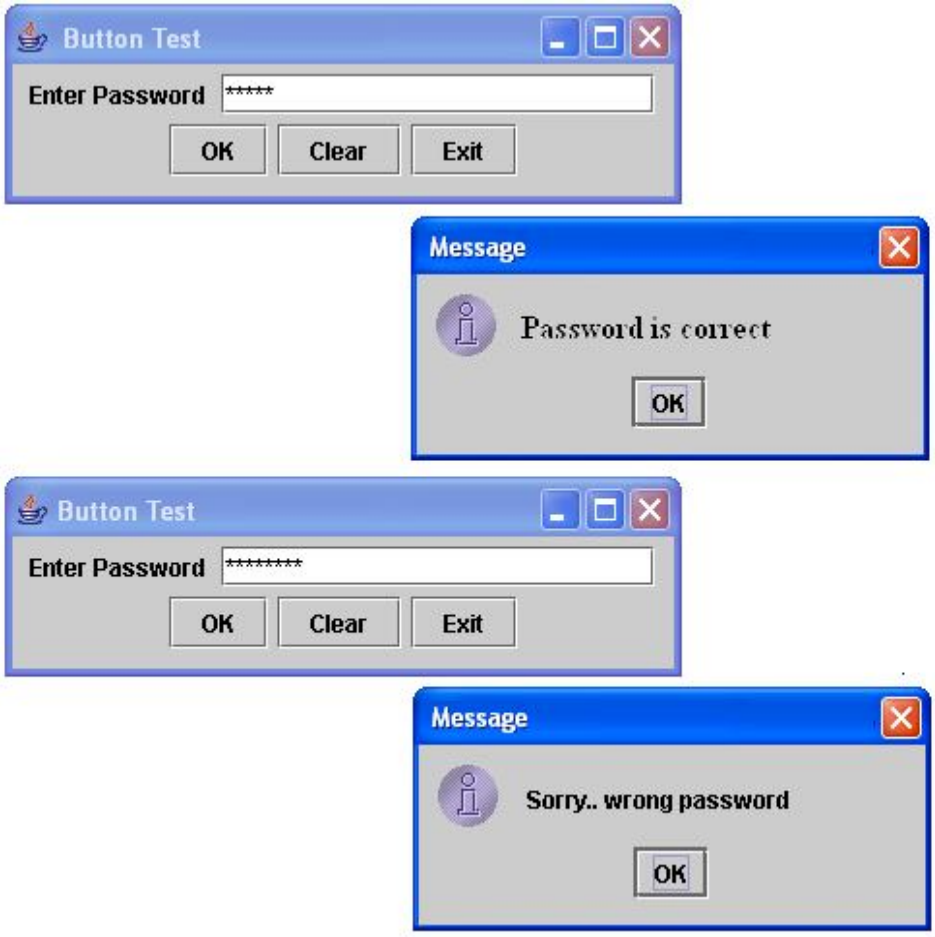
```
                    JOptionPane.showMessageDialog(this,    "Sorry..
                wrong password");
            }
        }
        public static void main(String args[]) {
            new ButtonTest();
        }
}
```

برنامج 3-2-3

عند الضغط على الزر exit، يتم إغلاق النافذة وإنهاء تنفيذ البرنامج باستخدام الدالة System.exit. وعند الضغط على الزر clear يتم إخلاء محتويات حقل النص text باستخدام الدالة setText التي تظهر النص الذي يتم تمريره إليها في حقل النص. عند الضغط على الزر ok يتم فحص كلمة السر التي أدخلها المستخدم، إذا كانت الكلمة هي sudan تظهر الرسالة password is correct، وإذ كانت غير ذلك تظهر الرسالة Sorry.. wrong password.

لاحظ أن JPasswordField هو class يشبه JTextField. إلا أن الحروف التي يطبعها بداخله المستخدم لا تظهر على الشاشة، بل تكون مخفية على شكل نجوم، وهو أمر مرغوب عند إدخال كلمات السر والبيانات الخاصة. تم ربط الأزرار الثلاثة مع مستمع واحد، أي أن نفس الدالة actionPerformed ستنفذ عند الضغط على أيٍ من الأزرار، ولذلك يجب التفريق بين الأزرار لمعرفة أيها تم الضغط عليها ليتم تنفيذ رد الفعل المناسب. تستخدم الدالة getSource التابعة لـ ActionEvent class للحصول على اسم الـ object الذي حدث فيه الحدث، أو اسم الزر الذي تم الضغط عليه. في المثال أعلاه تم استخدام الدالة getSource في عبارات if للتعرف على الزر الذي تم الضغط عليه. لاحظ أن الـ class المستمع في هذا المثال هو نفسه class البرنامج الرئيسي، ولذلك تم تعريفه على أنه ActionListener implements. وعند ربط الأزرار مع

المستمع تم استخدام المؤشر this، لأن الـــ object من هذا الـــ class هو نفسه المســـتمع الذي يحتوي على الدالة actionPerformed.



مخرجات برنامج 3-2-3

تعتبر الأزرار buttons وحقول النص text fields من أهم المكونات الرئيســـية في الواجهات الرسمية لأنها تمثل الأساس في عملية الإدخال والمعالجة والإخراج، ويمكن أن تستخدم إلى جانب المكونات الأخرى. توفر جافا مجموعة كبيرة من المكونات التي سنتطرق لأكثرها استخداماً خلال الأمثلة التالية. أحد هذه المكونات هو أزرار الخيار

radio button، وهي مجموعة من الخيارات يقوم المستخدم بإدخال أحدها بالنقر على الخيار بالفأرة، ولا يمكن اختيـــار أكثر من خيار واحد. ويمكن أن نجد هذا النوع من الأزرار مثلاً عند اختيار درجـــة صعوبة اللعبـــة فيكون الاختيار بين مبتدئ ومتوســـط ومتقدم، أو عند ملء استمارة معيـــنة ببيـــانات شـــخصـــية يتم اختيـــار النوع إما ذكر أو أنثـــى وهكذا. توفر جـــافا class JRadioButton ليستخدم مع الخيـــارات التي لا تســـمح فيها باختيار أكثر من خيـــار في نفس الوقت. يتم تعريف عدد من الكائنات من class JRadioButton بعدد الخيارات الموجودة، ويتم تعريف كائن من class ButtonGroup وإضـــافة جمـــيع الـــ radio buttons إليه للتوضـــيح بأن هذه الأزرار المضافة تابعة لنفس المجموعة ولا يمكن اختيـــار أكثر من خيـــار واحد بينها. عندمـــا يتم اختيـــار radio button معين ضمـــن مجموعة ينشـــأ حدث من نوع ItemEvent، ولذلك فإن الكائن الذي يســـتمع لهذا النوع من الأحداث يجب أن يكون implements ItemListener، وهو inferance يحتوي على دالة واحدة فقط هي itemStateChanged، وبالتالي يجب تعريفها ويتم وضـــع العبارات المطلوب تنفيذها عندمـــا يختـــار المستخدم خيـــاراً معيناً ضمـــن الخيارات الموجودة بداخلها. المثال التالي يشـــرح ذلك.

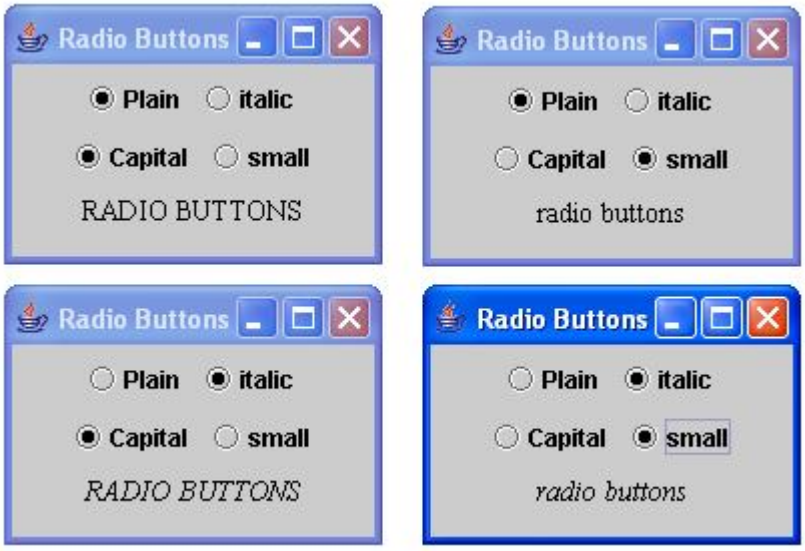
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class RadioButtonTest extends JFrame implements ItemListener {
        JLabel text;
        JRadioButton plain, bold, italic, capital, small;
        ButtonGroup styleGroup , caseGroup;
        public RadioButtonTest( ){
                super("Radio Buttons");
                Container c= getContentPane( );
                c.setLayout(new FlowLayout());
                styleGroup = new ButtonGroup();
                plain = new JRadioButton("Plain", true);
```

```
        plain. addItemListener(this);
        styleGroup.add(plain);
        c.add(plain);
        bold = new JRadioButton("Bold", false);
        bold.addItemListener(this);
        styleGroup.add(bold);
        italic = new JRadioButton("italic", false);
        italic.addItemListener(this);
        styleGroup.add(italic);
        c.add(italic);
        caseGroup = new ButtonGroup();
        capital = new JRadioButton("Capital", true);
        capital.addItemListener(this);
        caseGroup.add(capital);
        c.add(capital);
        small = new JRadioButton("small", false);
        small.addItemListener(this);
        caseGroup.add(small);
        c.add(small);
        text = new JLabel("Radio Buttons");
        c.add(text);
        setSize(180, 140);
        setVisible(true);
}
public void itemStateChanged(ItemEvent event) {
        if (event.getSource() = = plain)
                text.setFont(new Font("serif", Font.PLAIN, 14));
        if (event.getSource() = = bold)
                text.setFont(new Font("serif", Font.BOLD, 14));
        if (event.getSource() = = italic)
                text.setFont(new Font("serif", Font.ITALIC, 14));
        if (event.getSource() = = capital)
                text.setText("RADIO BUTTONS");
```

```
            if (event.getSource() = = small)
                text.setText("radio buttons");
        }
        public static void main(String args[]){
            new RadioButtonTest();
        }
}
```

برنامج 3-2-4

يتم جمع أزرار الاختيار bold و plain و italic ضمن مجموعة واحدة لكي يتم اختيار خيار واحد على الأكثر من بينها، حيث يتم تعديل خط الكتابة بناء على هذا الاختيار. أما الأزرار small و capital فتستخدم لتحويل حالة الحروف إلى كبيرة أو صغيرة حسب اختيار المستخدم. وإذا تم إضافتها لنفس المجموعة السابقة لن نستطيع أن نختار حالة الحروف إلى جانب خط الكتابة، لأننا لن نستطيع اختيار أكثر من خيار واحد ضمن المجموعة، يتم تعريف زر الاختيار بتحديد النص الذي نرغب بظهوره إلى جانب الزر، كما يتم تحديد الخيار الابتدائي للمجموعة، فالزر الذي يحمل القيمة true يمثل الخيار الابتدائي، ويكون الزر الذي يحمل القيمة false غير مظلل ابتدائياً. عند اختيار زر معين، يتولد حدث من النوع ItemEvent وبالتالي يتم نداء الدالة itemStateChange تلقائياً، وتفرق بين الأزرار المختلفة باستخدام الدالة getSource.



مخرجـــات برنامج 3-2-4

توفر جـــافا JCheckBox للاختبـــارات المتعددة، فهي تشبه JRadioButton في الشـــكل والاستخدام، إلا أنه عند استخدام check box يمكن أن نختـــار عدداً من الخيـــارات في المرة الواحدة، وبالتالي لا نحتاج لتقســـيمها إلى مجموعات، يتولد عند الضـــغط على JCheckBox حدث من نوع ItemEvent أيضـــاً.

2.3 القوائم JComboBox وJList

ومن الأدوات الهامة لتصميم الشـــاشات القوائم، وتوفر جـــافا نوعين من القوائم هي JList و JComboBox، ونســـبة للشـــبة الكبير بينهما نتطرق هنا إلى إحداهما وهي JComboBox. ويوضـــح البرنامج التالي كيـــفية تعريف القائمة والتعامل معها:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class ComboBoxTest extends JFrame implements ItemListener {
```

```
        JLabel codeLable;
        String[] countries = {"Egypt", "Jordan", "Saudi Arabia", "Sudan", "United
        Arab Emirates"};
        String[] codes = {"20", "962", "966", "249", "971"};
        JComboBox countryCodes;
        public ComboBoxTest(){
                super("Combo Box Test");
                Container c = getContentPane();
                c.setLayout(new FlowLayout());
                countryCodes = new JComboBox(countries);
                countryCodes.setMaximumRowCount(3);
                countryCodes.addItemListener(this);
                c.add(countryCodes);
                c.add(new JLabel("Telephone code is :"));
                codeLabel = new JLabel();
                c.add(codeLabel);
                setSize(200, 100);
                setVisible(true);
        }
        public void itemStateChanged(ItemEvent e) {
                int index = countryCodes.getSelectedIndex();
                codeLabel.setText(codes[index]);
        }
        public static void main(String args[]) {
                new ComboBoxTest();
        }
}
```

برنامج 3-2-5

يتم تعريف مصفوفة من String تستخدم في إنشاء القائمـــة، وهي في هذا البرنامج المصفوفة countries حيث تضـــع أسماء خمـــس دول في قائمـــة، وتســـمح باختيـــار دولة بالنقر على القائمة لفتحـــها. عندمـــا تفتح القائمـــة يكون طولها مقدار ثلاثة

أســطر، والتي تم تحديدها باستخدام الدالة setMaximumRowCount. عند اختيار دولة يتولد حدث من نوع ItemEvent ومن ثم يتم نداء الدالة itemStateChanged تلقائــياً، ولمعرفة العنصر الذي تم اختياره من القائمة تستخدم الدالة getSelectedIndex والتي ترجــع قيمــة integer تمثل ترتيب العنصر الذي تم اختياره. في هذا المثــال يترتب على اختيــار دولة معينة عرض مفتاح الدولة، حيث تم وضــع المفاتيح في المصفوفة codes بنفس ترتيب الدولة حتى يكون ترتيب المفتاح في مصفوفة المفاتيح هو نفسه ترتيب الدول في القائمــة. مثلاً عند اختيار Sudan من القائمــة تعيد الدالة getSelectedIndex القيمــة 3، وهو ترتيب Sudan في القائمــة، ويتم عرض العنصر رقم 3 في المصفوفة codes باستخدام codeLable وهو 249 والذي يمثل مفتاح السودان.



مخرجــات برنامج 3-2-5

يكمن الفرق بين القائمة من نوع JComboBox والقائمــة من نوع JList في أن JComboBox يتم النقر عليها وفتحها لاختيــار عنصر ثم تغلق مجدداً، بينما القائمــة من النوع JList هي قائمة مفتوحة دائماً، وتظهر عناصرها عل الشاشة. كذلك تسمح

القائمة من نوع JList باختيـــار عدة عناصر في المرة الواحدة إذا تم إعدادها لذلك، بينما لا يمكن اختيار أكثر من عنصر من عناصر القائمـــة من نوع JComboBox.

توفر جافا مجموعة كبيرة من الأدوات المستخدمـــة لتنســـيق الشـــاشات ودعم عمليات الإدخال والإخراج في البرنامج. و تقوم جمـــيعها على مبدأ تعريف وإعداد الأداة لاستخدامها، ثم انتظار المستخدم ليتعامل معها، ثم الاستجـــابة لأفعاله عن طريق معرفة نوع الحدث الناتج عن التعامل مع الأداة المعيـــنة. ومن هذه الأدوات مســـاحـــات النصوص JTextArea والتي تســـمح بطـــباعة عدد من السطور على الشـــاشة. هناك أيضـــاً القائمـــة الرئـــيسية JMenu لإنشاء القوائم الرئـــيســـية للشـــاشات. يمكن كذلك التعامل مع الجداول باستخدام JTable إضافة إلى الكثير من الأدوات المختلفة والتي لا يتســـع المقـــام لذكرها.

كثيراً ما تتفاعل البرامج والشـــاشات مع لوحـــة المفاتيح أي أن هناك أحداث نرغب في التحكم بها بواســـطة لوحـــة المفاتيح. فمثلاً نلاحظ في كثير من البرامج العبارة: (للمتابعة اضـــغط على أي مفـــتاح). في لغة جافا يستخدم interface KeyListener لمتابعة مدخـــلات لوحـــة المفاتيح، ويحتوي على ثلاث دوال هي:

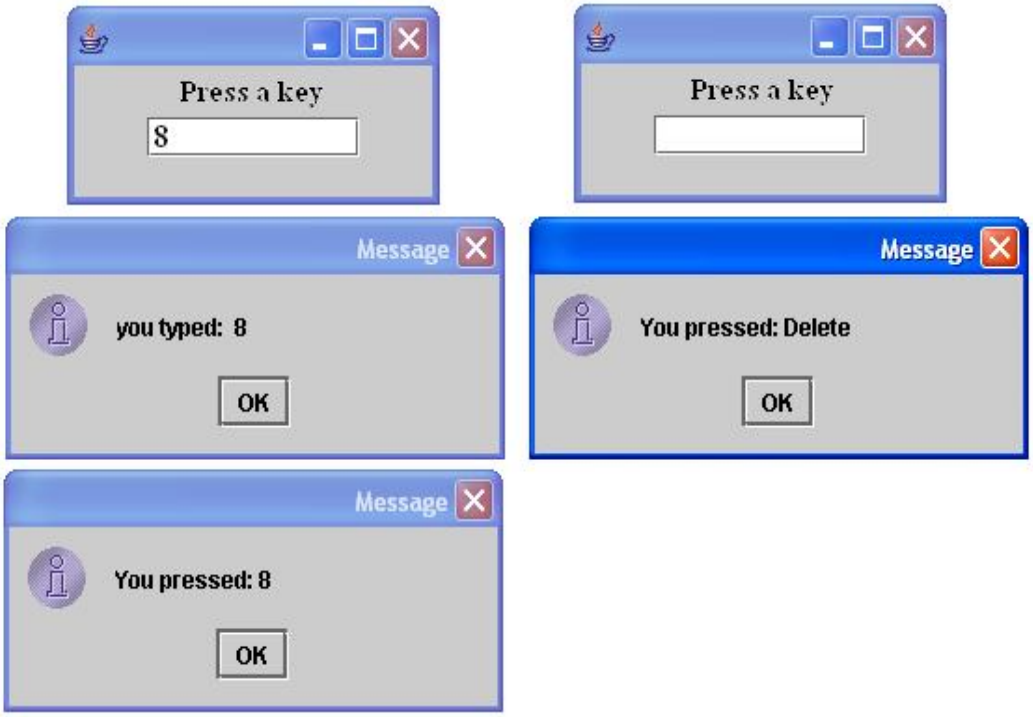
1. keyPressed: يتم نداؤها تلقائـــياً عند الضـــغط على أي مفتاح في لوحة المفاتيح مثـــل: A, Enter, ALT, 2, * وغيرها.
2. keyTyped: ويتم نداؤها تلقائـــياً عند الضـــغط على أي مفتاح يظهر على الشـــاشة مثـــل: A, 2, *, S وغيرها.
3. keyRelased: ويتم نداؤها تلقائـــياً عند إنهاء الضـــغط على المفتاح أو تركـــه بعد الضـــغط عليه.

يتم تعريف هذه الدوال حتى وإن لم نرغب في استخدامها جمـــيعاً، ويجب أن يتم تســـجـــيل العنصر المعيـــن بواســـطـــة الدالة addKeyListener لتتفاعل الدوال مع أحداث لوحـــة المفاتيح التي تحدث من ذلك العنصر. يوضـــح المثـــال التالي كـــيفية التعامل مع هذه الدوال.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KeyboardApplication extends JFrame implements KeyListener {
        JTextField text;
        public KeyboardApplication() {
                Container c= getContentPane();
                c.setLayout(new FlowLayout());
                c.add(new JLabel("Press a key");
                text = new JTextField(10);
                text.addKeyListener(this);
                c.add(text);
                setSize(200, 100);
                set.Visible(true);
        }
        public void keyTyped(KeyEvent e) {
                JOptionPane.showMessageDialog(this, "you typed: " +
e.getKeyChar());
        }
        public void keyPressed(KeyEvent e) {
                JOptionPane.showMessageDialog (this, "You pressed: " +
e.getKeyText(e.getKeyCode()));
        }
        public void keyReleased(KeyEvent e) {
                JOptionPane.showMessageDialog(this , "You released: " +
e.getKeyText(e.getKeyCode()));
        }
        public static void main(String args[]) {
                new KeyboardApplication();
        }
}
```

برنامج 3-2-6

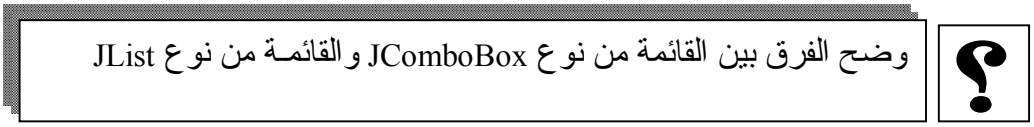
لدى class KeyEvent عدة دوال تستخدم للتفاعل مع الأحداث الصادرة عن لوحة المفاتيح. تستخدم الدالة getKeyChar للحصول على الحرف الذي تم الضغط على مفتاحه، وذلك عندما يكون المفتاح خاصاً بحرف يظهر على الشاشة مثل 9 و w وغيرها. وعند الضغط على المفاتيح الأخرى مثل Shift أو F1 أو غيرها، يتم استخدام الدالة getKeyCode للحصول على الرمز العددي الممثل لهذا المفتاح، ثم تحويل هذا الرمز إلى نص باستخدام الدالة getKeyText والتي تستقبل هذا الرمز، ومن ثم طباعته على الشاشة.



مخرجـــات برنامج 3-2-6

يوضح هذا البرنامج كـــيف أن لغة جـــافا تدعم برمجـــة التطـــبيقات التي تتفاعل مع مدخـــلات لوحـــة المفاتيح. توفر جـــافا كذلك طرقاً لتفاعل البرامج مع الأحداث التي تصدر عن الفأرة mouse مثل الضـــغط على زر الفأرة (النقر) أو تركه، ودخول الفارة إلى منطـــقة معـــيـــنة أو خروجـــها منه. وكلها عبارة عن دوال تابعة لـــ interface MouseListener و interface MouseMotionListener يمكن استخدامها للتفاعل مع حركات الفأرة.

أسئلة تقويم ذاتي



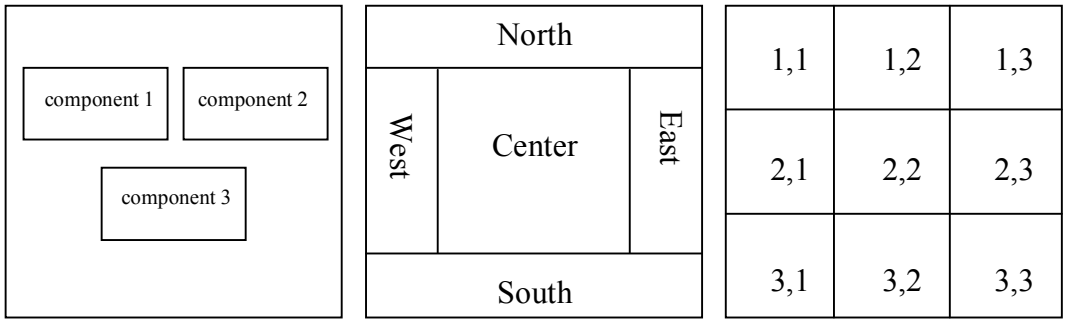
Panel 3.3

هناك أيضا مفهوم الـــ Panel وهو عبارة عن وعـــاء يمكن أن يحتوي على عدد من المكونات بداخـــله ومن ثم إضـــافته للـــ container الأساسي لنافذة البرنامج، ويتم إنشـــاء panel باستخدام class JPanel. وتكمـــن أهميته في إضفاء النظـــام والترتيب على مكونـــات النافذة، واستخدام طرق مختلفـــة للترتيب وتنســـيق المكونـــات داخـــل النافذة. وقـــبل أن نوضـــح استخدام class JPanel في مثـــال، نريد أن نقدم بعض الطرق المستخدمـــة لتصـــميم وتنظـــيم المكونـــات داخل النافذة Layout. هناك عدد كبير من الطرق التي يمكن بها تنظيـــم وعرض مكونـــات الشـــاشة ونتناول هنا أشهر ثلاث طرق وهي:

- Flow Layout: وهو يقوم بوضـــع المكونـــات داخل الشـــاشة بالترتيب الذي تمت إضافتهم به، حتى تمتلئ النافذة عرضـــياً، وحينها يقوم بمتابعة وضـــع المكونـــات في السطر التالي وهكذا.

●Border Layout: ويقسم الشاشة إلى خمسة أجزاء وهي: الوسط الذي يخصص له أكبر مساحة، إضافة إلى الشرق والغرب والشمال والجنوب، ويجب تحديد القسم الذي نرغب في إضافة العنصر إليه في عبارة الإضافة add.

●Grid Layout: وهو يقسم النافذة إلى شبكة من الخانات متساوية الحجم يحدد عددها (طولها وعرضها) المبرمج. يتم إضافة أول عنصر إلى الخانة أعلى شمال النافذة، وتتواصل إضافة العناصر من الشمال إلى اليمين بالصف الأول حتى يمتلئ، ليتم الانتقال إلى الصف الثاني وهكذا.



Flow Layout **Border Layout** **Grid Layout**

شكل 3-2-3: أشهر طرق تنظيم نافذة البرنامج

في جميع الأمثلة السابقة تم استخدام طريقة Flow Layout لتنظيم مكونات النافذة للبرنامج، حيث تظهر بنفس الترتيب الذي تمت إضافته به، ويأخذ كل عنصر حجمه العادي. خلال الأمثلة التالية نستخدم طريقة Border Layout وGrid Layout للتحكم في شكل النهائي لنافذة البرنامج إلى جانب استخدام JPanel للتعامل مع عدة Layouts في نفس النافذة.

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutTest extends JFrame {
        JButton b[];
        public GridLayoutTest(){
```

```
                super("Testing grid layout");
                Container c = getContentPane();
                c.setLayout(new GridLayout(4,4));
                b = new JButton[16];
                for (int i = 0; i <16; i ++) {
                        b[i] = new JButton(String.valueOf(i));
                        c.add(b[i]);
                }
                setSize(300, 300);
                setVisible (true);
        }
        public static void main(String args[]) {
                new GridLayoutTest();
        }
}
```

برنامج 3-2-7

في هذا المثـــال تم استخدام حلقة for لإنشاء الأزرار وإضافتها، حيث إن النص الذي يظـــهر فوق كل زر هو عبارة عن ترتيب ذلك الزر في المصفوفة. لاحظ أن هذه الأزرار ليس لها وظـــيفة معيـــنة وإنما تم تصميم الشـــاشة بغرض توضـــيح شـــكل نافذة البرنامج عند استخدام طريقة التصميم Grid Layout ومقارنتها بطريقة Flow Layout.



مخرجـــات برنامج 3-2-7

```
import java.awt.*;
import javax.swing.*;
public class BorderLayoutTest extends JFrame {
        JLabel title, nameLabel, passwordLabel;
        JTextField name, password;
        JButton signUp, login, clear, exit;
        JPanel textPanel, buttonPanel;
        public BorderLayoutTest() {
                super("Testing Border Layout");
                Container c = getContentPane( );
                c.setLayout(new BorderLayout());
                title = new JLabel ("Login page");
                c.add(title, BorderLayout.NORTH);
                nameLabel = new JLabel("Name");
                passwordLabel = new JLabel("Password");
                name = new JTextField(20);
                password = new JTextFiled(20);
                textPanel = new JPanel();
                textPanel.setLayout(new FlowLayout());
                textPanel.add(nameLabel);
                textPanel.add(name);
```

```
            textPanel.add(passwordLabel);

            textPanel.add(password);
            c.add(textPanel, BorderLayout.CENTER);
            signUp = new JButton("Sign Up");
            login = new JButton ("Login");
            clear = new JButton(" Clear");
            exit = new JButton("Exit");
            buttonPanel = new JPanel();
            buttonPanel.setLayout(new GridLayout(1, 4));
            buttonPanel.add(signUp);
            buttonPanel.add(login);
            buttonPanel.add(clear);
            buttonPanel.add(exit);
            c.add(buttonPanel, BorderLayout.SOUTH);
            setSize(320, 150);
            setVisible(true);
        }
        public static void main(String args[]){
            new BorderLayoutTest();
        }
}
```

برنامج 3-2-8

في هذا البرنامج تم تصميم نافذة البرنامج باستخدام Border Layout، وتم أولاً إضافة JLabel إلى المنطقة الشمالية بالشاشة، وكما نعلم فإن هذا النوع له خمس مناطق فقط، وبالتالي يسمح بإضافة خمسة عناصر فقط. لكن البرنامج يحتوي على مكونات كثيرة، فمثلاً هناك أربعة أزرار نرغب في إضافتها جميعاً بجنوب النافذة، ولذلك نضيفهم جميعاً إلى JPanel منظم بالطريقة المناسبة (في هذا المثال Grid Layout)، وفي النهاية يضاف الـ JPanel جنوب النافذة. ونلاحظ أن طريقة تصميم الـ JPanel تكون مستقلة تماماً

عن الطريقة المستخدمة في النافذة الأساسية. بنفس الطريقة نعرف JPanel ونصممه بطريقة Flow Layout ونضيف الـ JLabels والـ TextFileds والتي نرغب بإضافتها في وسط النافذة، وأخيراً نضيف الـ JPanel إلى وسط النافذة. نلاحظ أنه يتم تحديد أقسام النافذة الخمسة التي يوفرها Border Layout باستخدام ثوابت موجودة في class BorderLayout وهي CENTER , WEST , EAST, SOUTH, NORTH.



مخرجات برنامج 3-2-8

في الأمثلة السابقة استعنا بـ JPanel لاستخدام طرق تصميم مختلفة عن التي تم استخدامها مع نافذة البرنامج الأساسية. يمكن أن تستخدم الـ JPanels أيضاً لفصل تنسيق بعض العناصر عن بقية عناصر الشاشة، ومن أمثلة هذا التنسيق لون الخلفية ومواصفات الخط المستخدم للكتابة والمسافات بين العناصر داخل المنطقة المحددة.

توفر جافا طرقاً أخرى لتصميم شاشات البرامج، ومنها GridBagLayout و CardLayout و BoxLayout ولكل منها شكله العام وطريقة استخدامه. عندما لا نحدد layout معين للنافذة، يتم افتراضه BorderLayout وهو طريقة التصميم الافتراضية لأي JFrame، بينما طريقة التصميم الإفتراضية لأي JApplet هي FlowLayout. يمكن في بعض البرامج تحديد نوع التصميم null عند استخدام الدالة

setLayout، وعندها لن يتم تقسيم نافذة البرنامج بأي طريقة، وإنما يمكن تحديد أماكن إضافة مكونات النافذة باستخدام الدالة setBounds لكل العناصر، حيث تستقبل الإحداثي السيني والصادي لنقطة بداية رسم الركن الشمالي الأعلى من العنصر، وطول وعرض العنصر. ويعتبر استخدام هذه الطريقة صعباً ومعقداً ولا يتم اللجوء إليه إلا للضرورة، لأن تصميم نافذة أي برنامج لا تخرج عادة عن الطرق التي أعدتها وهيأتها لغة جافا مسبقاً للتسهيل على المبرمج، فيتم التعامل مع الدوال والثوابت بأقل وقت وأقل جهد.

أسئلة تقويم ذاتي



ما هو الـ Panel ؟

حدد طريقة التصميم الافتراضية لكل من JFrame و JApplet؟

الخلاصة

بينت الوحدة أن لغة جافا توفر مجموعة كبيرة وغنية من أدوات الرسم والتلوين موجودة في Package Java.awt، وذكرت أن أهمية الـــ Class بالنسبة للرسم في أنه يمثل اللوحة التي سيتم عليها الرسم والتلوين.

وضحت الوحدة أنه يمكن إعداد لون معين لاستخدامه في الرسم بإحدى طرقتين:

باستخدام الألوان التي تحتوي عليها Class Color أصلاً على شكل ثوابت

```
Color  c = Color .RED
```

باستخدام نظام RGB(Red-Green-Blue) للألوان.

كذلك وضحت الوحدة أن Class Font يساعد على اختيار نوع وحجم وشكل خط الرسم على الشاشة .

عددت الوحدة عدداً من الدوال الموجودة في Class Graphics منها الدالة Set Color التي تستخدم لتحديد اللون الذي سيتم استخدامه للرسم على النافذة ، وكذلك الدالة Set Font لتحديد خصائص الخط الذي ستظهر به الكتابة.

لمحة مسبقة عن الوحدة التالية

الوحدة التالية بعنوان موضوعات متقدمة في لغة جافا ومن خلالها نتناول عـــدداً من الموضوعات حيث نجد فيها البرمجيات applets، وأيضاً تتناول الوحدة الاســـتثناءات exceptions ومعالجة الاستثناءات Exception Handling ، كما تتناول الوحـــدة الملفـــات باعتبارها إحدى وسائل تخزين البيانات الهامة في الحاســـوب، كمـــا تتنـــاول البرمجـــة المتعددة حيث سنتعرف على أهمية البرمجة المتعددة، والدور الذي باتت تؤديه الـــ Multi processing - مع انتشار الحاسوب اليوم، والعدد الهائل للتطبيقـــات والخـــدمات التـــي يوفرها الحاسوب لعدد ضخم من المستخدمين.

مسرد المصطلحات

نظام RGB (Red- Green- Blue) للألوان

ينص على أن كل لون هو ناتج مزج نسبة معينة من اللون الأحمر والأخضر والأزرق، تتراوح هذه النسبة بين الصفر و 225 لكل لون.

معالج الأحداث Event Handling

هي أن يقوم المستخدم بعمل حدث معين في البرنامج مثل ضغطة زر أو طباعة حرف أو ضغط Enter أو تحريك الـــ mouse أو أي حدث آخر، فيقوم البرنامج بتنفيذ رد فعل محدد رداً على فعل المستخدم.

مصدر الحدث event source

ويمكن أن يكون أي مكون component في نافذة البرنامج أومفتاح في لوحة المفاتيح أو الـــ mouse.

كائن الحدث event object

كل حدث معين في البرنامج ينشأ أو يتولد عنه object محدد، يبحث هذا الـــ object عن المنطقة في البرنامج التي ستعالج الحدث وتقوم برد الفعل المناسب.

معالجة الحدث event hander:

وهو جزء من البرنامج – دالة عادة – يكون في حالة استماع دائم لنوع محدد من الأحداث يعالجها و يقوم برد فعل محدد، ولذلك يعرف أيضا بالمستمعlistener.

أزرار الخيار radio button

وهي مجموعة من الخيارات يقوم المستخدم بإدخال أحدها بالنقر على الخيار بالفأرة، ولا يمكن اختيـــار أكثر من خيار واحد.

Panel

وهو عبارة عن وعـــاء يمكن أن يحتوي على عدد من المكونات بداخـــله ومن ثم إضـــافته للـــ container الأساسي لنافذة البرنامج، ويتم إنشـــاء panel باستخدام class. JPanel

Flow Layout

إحدى طرق تنظيم وعرض مكونات الشاشة، وهو يقوم بوضـــع المكونـــات داخل الشـــاشة بالترتيب الذي تمت إضافتها به، حتى تمتلئ النافذة عرضـــياً، وحينها يقوم بمتابعة وضـــع المكونـــات في السطر التالي وهكذا.

Border Layout

إحدى طرق تنظيم وعرض مكونات الشاشة ويقســـم الشـــاشة إلى خمسة أجزاء وهي الوســـط الذي يخصص له أكبر مســـاحة، إضافة إلى الشرق والغرب والشمـــال والجنوب، ويجب تحديد القســـم الذي نرغب في إضافة العنصر إليه في عبارة الإضافة add.

Grid Layout

إحدى طرق تنظيم وعرض مكونات الشاشة وهو يقســـم النافذة إلى شـــبكة من الخانات متســـاوية الحجم يحدد عددها (طولها وعرضـــها) المبرمج. يتم إضـــافة أول عنصر إلى الخانة أعلى شمـــال النافذة، وتتواصل إضافة العناصر من الشـــمال إلى اليمين بالصـــف الأول حتى يمتلئ، ليتم الانتقال إلى الصـــف الثاني وهكذا.

المراجع

1. Y. D. Liang
   Introduction to Java Programming, Third Edition
   ISBN: 0-13-031997-X
   Prentice Hall 2001
2. Deitel and Deitel
   Java How to Program, Third Edition
   ISBN: 0-13-012507-5
   Deitel 2000
3. Deitel and Deitel
   Java How to Program, Fifth Edition
   ISBN: 0-13-101621-0
   Deitel 2003
4. Stephan J. Chapman
   Introduction to Java
   ISBN: 0-13-919416-9
   Prentice Hall 1999
5. نعمة عزالدين محمد عثمان

   البرمجة بلغة جافا للمبتدئين

   الإصدارة الثانية 2007م


الوحدة الرابعة
موضوعات متقدمة في لغة جافا

محتويات الوحدة

الموضوع	رقم الصفحة
المقدمة	232
تمهيد	232
أهداف الوحدة	233
1. البريمجات	234
2. الاستثناءات ومعالجة الاستثناءات	242
2 .1 الاستثناءات Exceptions	242
2.2 معالجة الاستثناءات Exception Handling	244
3 . الملفات Files and Streams	258
3 . 1 أهمية الملفات	258
3 . 2 التعامل مع الملفات	258
4. البرمجة المتعددة	265
1.4 أهمية البرمجة المتعددة	265
2.4 دورة حياة الـــ Thread (Thread life–cycle)	267
3.4 أولوية تنفيذ الـــ Threads	269
4.4 التشارك في المصادر	274
5 .الشبكات Networking	280
5 .1 أهمية الشبكات	280
2.5 برمجة Server و Client باستخدام Stream Socket Connection	282
3.5 برمجة Serves و Client باستخدام Connectionless Datagram Sockets	288

الخلاصة	295
مسرد المصطلحات	300
المراجع	302

المقدمة

تمهيد

عزيزي الدارس،

مرحباً بك عزيزي الدارس في الوحدة الرابعة من مقرر **"البرمجة بلغة جافا"** والتي تحمل عنوان "موضوعات متقدمة في لغة جافا ". وتحتوي هذه الوحدة على خمسة أقسام، حيث يتناول القسم الأول البريمجات Applets، والـــ Applet هي برنامج جافا مكتوب بصورة معينة تجعله قابلاً للتضمين داخل برنامج HTML (Hyper Text Markup Language)

أما القسم الثاني من الوحدة فيتناول الاستثناءات ومعالجة الاستثناءات، والقسم الثالث يتناول الملفات، في القسم الرابع نتناول أهمية البرمجة المتعددة، وفي هذا الجانب نوضح أهمية المعالجة المتعددة والدور الذي يقوم به المعالج processor كما سنشرح كيفية عمل هذا المعالج .

كما نتناول في هذا القسم دورة حياة الـــ Thread (Thread life–cycle)، وأيضاً نتناول في هذا القسم حالات البرمجة المتعددة والمتمثلة في ثلاث حالات.

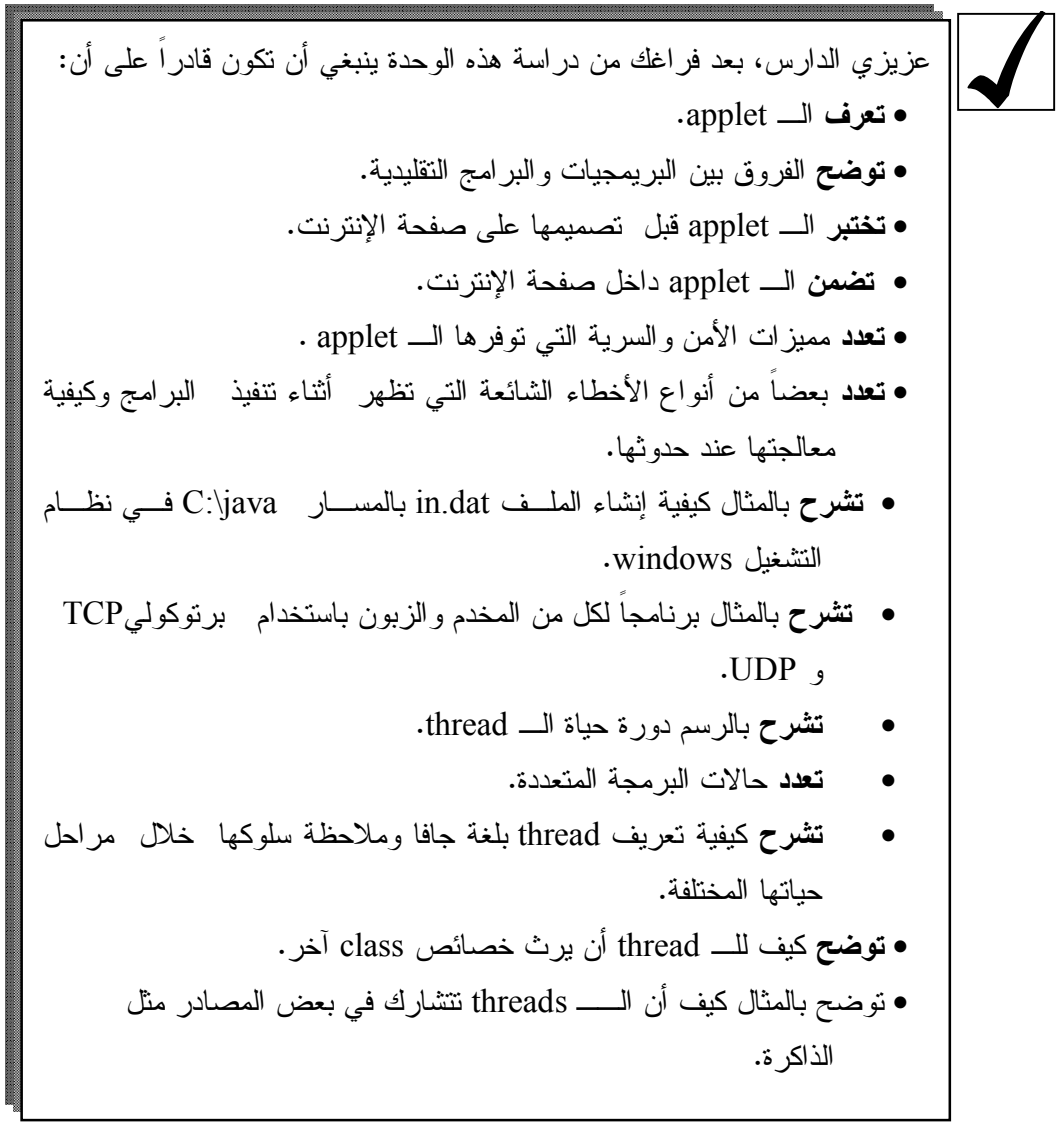
كذلك تجد في هذا القسم أولوية تنفيذ الـــ Thread . ونقدم في هذا القسم برنامج 1.4.4 والذي نشرخ فيه كيفيه تعريف الـــ Thread بلغة جافا وملاحظة سلوكها خلال مراحل حياتها المختلفة، ونتناول أيضاً البرنامج 2.4.4 الذي نوضح فيه كيفيه تعريف وتشغيل Thread باستخدام interface Runnable.

وأيضاً نتناول البرنامج 3.4.4 والذي من خلاله نوضح عملية التشارك في المصادر، وفيه نتعرف على مشكلة الأخطاء في البيانات الناجمة عن تعامل أكثر من برنامج مع نفس موقع الذاكرة ... ثم على ضوء البرنامج السابق نتعرف على كيفية منع تعامل أكثر من برنامج أو Thread واحد مع نفس مواقع الذاكرة طالما أن هناك Thread أخرى

تتعامل مع هذا الموقع وذلك من خلال برنامج 4.4.4 الذي يمثل تعديلاً للبرنامج السابق class Account.

وفي القسم الخامس نتناول الشبكات. وفي هذا القسم نقدم شرحاً وافياً لبرمجة كل من المخدم والزبون باستخدام برتوكولي TCP و UDP.

أهداف الوحدة



عزيزي الدارس، بعد فراغك من دراسة هذه الوحدة ينبغي أن تكون قادراً على أن:

- **تعرف** الـ applet.
- **توضح** الفروق بين البريمجيات والبرامج التقليدية.
- **تختبر** الـ applet قبل تصميمها على صفحة الإنترنت.
- **تضمن** الـ applet داخل صفحة الإنترنت.
- **تعدد** مميزات الأمن والسرية التي توفرها الـ applet .
- **تعدد** بعضاً من أنواع الأخطاء الشائعة التي تظهر أثناء تنفيذ البرامج وكيفية معالجتها عند حدوثها.
- **تشرح** بالمثال كيفية إنشاء الملف in.dat بالمسار C:\java في نظام التشغيل windows.
- **تشرح** بالمثال برنامجاً لكل من المخدم والزبون باستخدام برتوكوليTCP و UDP.
- **تشرح** بالرسم دورة حياة الـ thread.
- **تعدد** حالات البرمجة المتعددة.
- **تشرح** كيفية تعريف thread بلغة جافا وملاحظة سلوكها خلال مراحل حياتها المختلفة.
- **توضح** كيف للـ thread أن يرث خصائص class آخر.
- توضح بالمثال كيف أن الـ threads تتشارك في بعض المصادر مثل الذاكرة.

1. البريمجات Applets

تزامن تصميم ووضع بنود لغة جافا في بداية التسعينات مع ظهور وانتشار الإنترنت. ولذلك فقد أولى مصممو لغة جافا اهتماماً كبيراً لبرمجة صفحات الإنترنت، ومحاولة الاستفادة من أكبر قدر من خصائص هذه اللغة الجديدة في تطبيقات الإنترنت. مما ميزها عن باقي اللغات عالية المستوى.

تسمى جميع البرامج التي تمت كتابتها خلال الفصول الماضية بتطبيقات جافا java applications، واهم ما يميزها هو بداية تنفيذها عند الدالة الرئيسية main. توجد طريقة أخرى لكتابة برامج جافا لكي تكون ملائمة للتحميل والتشغيل على صفحات الإنترنت، وتسمى البريمجات applets. الـــــــ applet هي برنامج جافا مكتوب بصورة معينة تجعله قابلاً للتضمين داخل برنامج HTML (Hyper Text Markup Language) وهي إحدى أشهر وأسهل اللغات المستخدمة لبرمجة صفحات الإنترنت. عند تنفيذ برنامج HTML يتم تحميل الـــــــ applet على الصفحة وتشغيلها، مما يمكن الإنترنت من الاستفادة من جميع مزايا لغة جافا ويسهل تداول وانتشار برامج جافا بين عدد أكبر من المستخدمين.

لا تختلف البرمجة بواسطة البريمجات applets عما تطرقنا إليه حتى الآن، فهي تتمتع بنفس خواص لغة جافا ويمكن أن تحتوي على معظم ما توفره اللغة من تقنيات. يأتي الفرق بين البريمجات والبرامج التقليدية في طريقة التنفيذ والصفات الإضافية التي تتمتع بها لكونها معدة للتحميل والتشغيل على صفحات الإنترنت.

لكي يطلق على برنامج معين applet، يجب أن يتم تعريف class يرث خصائص JApplet، وهو class موجود في package javax.swing.هذا الـــــــ class يحتوي على أربع دوال ذات وظائف معينة يمكن إعادة تعريفها اختياريا بداخل البرنامج. فيما يلي بيان لهذه الدوال واستخدامها.

init( )

هي أول دالة يتم تنفيذها تلقائيا في البرنامج، حيث لا تحتوي الـــــــ applet على constructor أو دالة main. تستخدم عادة لتعريف الـ objects التي يحتاجها البرنامج، ووضع القيم الابتدائية وإعداد الشاشة وغيرها.

start( )

يبدأ تنفيذها تلقائيا بعد الدالة ( )init، وتحتوي على التعليمات الضرورية لتشغيل البرنامج الذي تم إعداده وتجهيزه في الدالة ( )init. كما يتكرر نداؤها أيضاً إذا انتقل المستخدم من صفحة الإنترنت التي تحتوي على الـــــــ applet إلى صفحة أخرى عن طريق رابط link لفترة ثم عاد إليها مرة أخرى.

paint( )

ولها نفس الاستخدام الذي تطرقنا له في الباب السابق، حيث تستخدم للوصول إلى جميع دوال Graphics الموجودة في package java.awt لإعداد الرسوم والألوان على الشاشة. يتم نداء هذه الدالة تلقائيا بعد الدالة ( )start.

stop( )

عندما يضغط المستخدم على رابط link في الصفحة، ويتم تنفيذ ما بداخل هذه الدالة إذا كان هناك ما نريد إيقاف تشغيله لحين عودة المستخدم إلى صفحة الـــــــ applet مرة أخرى. مثلا أن نوقف تشغيل ملف صوت عند الانتقال إلى صفحة أخرى ونواصل تشغيله عند عودة المستخدم إلى الصفحة عن طريق الدالة ( )start.

destroy( )

ونكتب بداخلها ما نريد تنفيذه عند إغلاق الصفحة، أي عند توقف عمل الـــــــ applet نهائيا، مثل إغلاق الملفات التي قام بفتحها البرنامج وغيره.

تحتوي الـــــــ appletعلى جميع خصائص ودوال JFrame، ذلك إن الغرض من الـــــــ applet هو الظهور على صفحة الإنترنت، لذلك فهي مصممة أن تكون JFrame دائماً، والذي أكسبها الكثير من الخصائص والدوال.

فيما يلي مثال لـــــــ applet وشرح لكيفية ترجمتها وتضمينها في صفحة انترنت بسيطة:

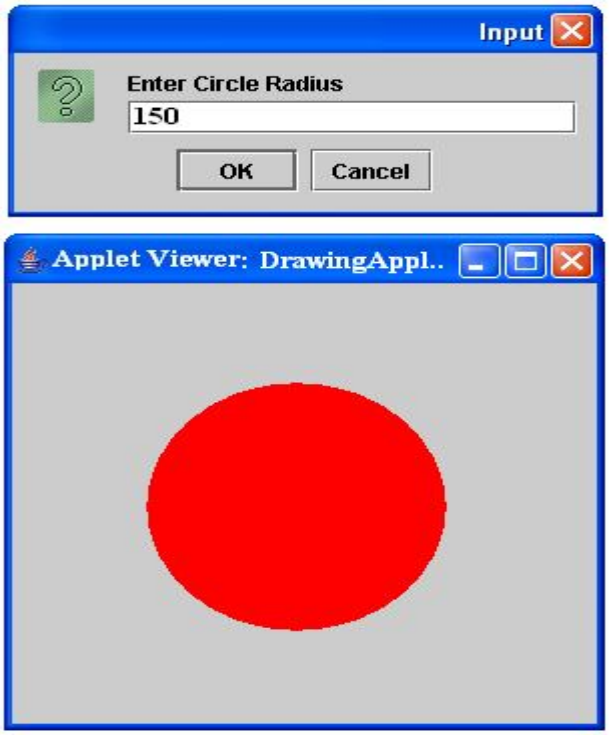
```
import  javax.swing.*;
import  java.awt.*;
public class DrawingApplet extends JApplet {
        int radius = 100;
        public void start( ) {
                String  s  =  JOptionPane.showInputDialog(this,  "Enter  Circle
Radius");
                radius = Integer.parseInt(s);
        }
        public void paint(Graphics  g) {
                g.setColor(Color.RED);
                g.fillArc(100, 100, radius, radius, 0, 360);
        }
}
```

برنامج 4-1-1

يبدأ تنفيذ البرنامج بنداء الدالة start تلقائياً ليقوم المستخدم بإدخال نصـــف قطـــر الدائرة، ومن ثم تنفيذ الدالة paint لرسم هذه الدائرة على نافذة البرنـــامج. بعـــد ترجمـــة البرنامج للحصول على الملف DrawingApplet.class. يمكـــن تنفيـــذ البرنـــامج لرؤيـــة المخرجات بإحدى الطريقتين التاليتين: 00966508249930

1- باستخدام البرنامج applet viewer والموجود في نفس المسار الذي يوجد فيه مترجم جافا Javac.exe ومفسر جافا Java.exe، وذلك بكتابة العبارة:

```
appletviewer DrawingApplet
```



مخرجات برنامج 4-1-1

ويستخدم البرنامج applet viewer لتجريب واختبار الـ applet قبل تضمينها على صفحة الإنترنت.

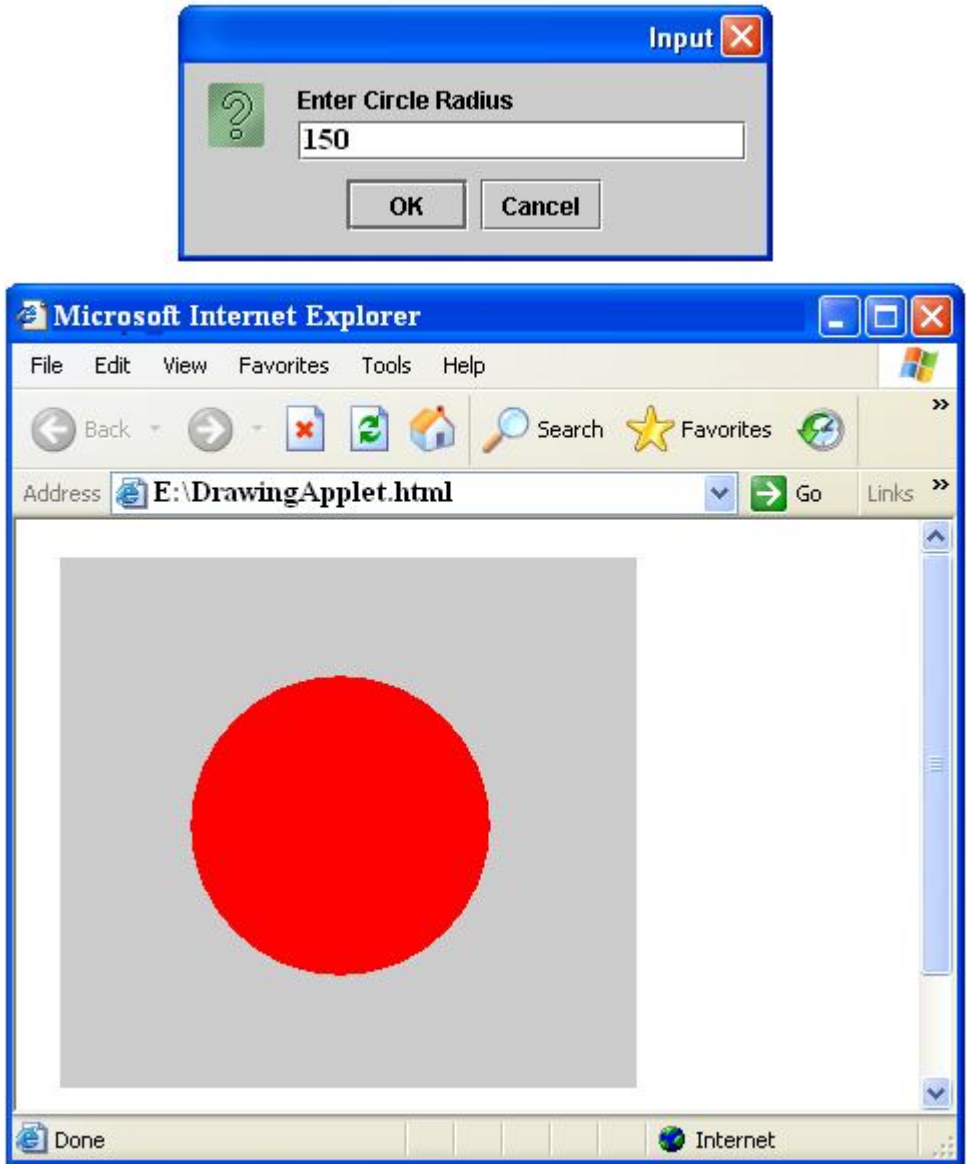
2- عن طريق تضمين الـــ applet داخل صفحة إنترنت، وفيما يلي نوضح أبسط برنامج مكتوب بلغة HTML لإظهار applet على صفحة انترنت:

```
< applet
        code = "DrawingApplet.class"
        width = "500"
        height = "500" >
< / applet>
```

برنامج 4-1-2

المعلومات التي نحتاجها عن الـــ applet ليتم كتابتها في برنامج HTML هو اسم الملف بالامتداد class، والذي يحتوي على الـــ applet بعد الترجمة. كذلك نحتاج

إلى طول وعرض النافذة التي نرغب في تخصيصها للـــ applet على الصفحة. تحفظ العبارات الموضحة في البرنامج 4-1-2 في ملف بالامتداد html أو htm، ويتم فتحه بواسطة متصفح انترنت.



مخرجات برنامج 4-1-2

أحيانا قد يحتاج الـــ applet المضمن داخل صفحة انترنت إلى بعض المعلومات من تلك الصفحة، مثلا قد تكون الـــ applet مستخدمة في صفحة تغير من مظهرها باستمرار، فترغب الـــ applet في معرفة الحجم الحالي للصفحة ونوع خط الكتابة المستخدم أو أي معلومة أخرى. توفر الـ applet طريقة لتبادل الوسائط parameters بينها وبين برنامج HTML حيث تقوم عبارات HTML معنية بتحديد اسم وقيمة كل وسيط، وتحصل الـ applet على قيم هذه الوسائط باستخدام الدالة ( )getParameter، كما هو موضح في المثال التالي:

```
import  java.awt.*;
import  javax.swing.*;
public  class  ParameterTest  extends  JApplet {
        int  edge = 50;
        int  max = 1;
        public void init( ) {
                edge = Integer.parseInt(getParameter("edge"));
                max = Integer.parseInt(getParameter("maximum"));
        }
        public  void  paint(Graphics g) {
                for (int i = 1;  i< = max;  i++)
                        g.drawRect(i * 50,  i * 50, edge, edge);
        }
}
```

برنامج 4-1-3

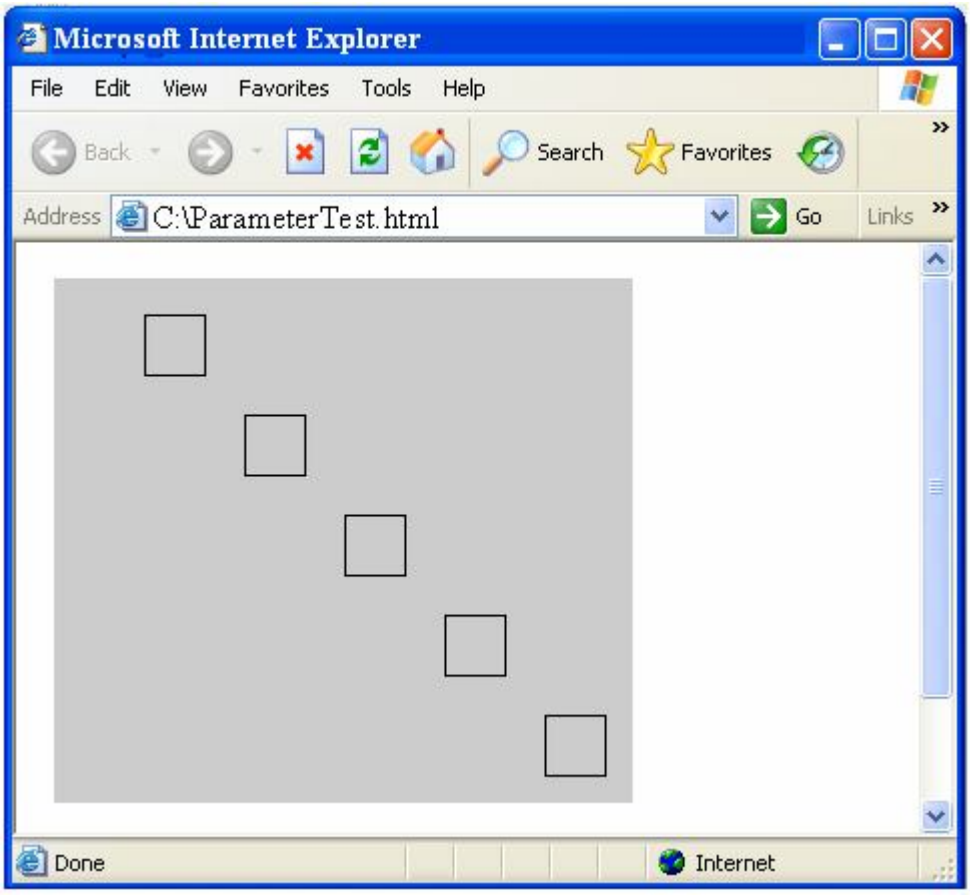
هذا البرنامج هو مثال لـ applet تستقبل قيماً لوسائط معينة من برنامج HTML، ثم تستخدمها الـ applet في تنفيذ عباراتها. الدالة ()getParameter تسترجع قيمة الوسيط المحدد بين قوسيها حسب ما هو موجود في برنامج HTML. لنفرض أن جزء برنامج HTML المتضمن للـ applet مكتوب كما يلي:

```
< applet
        code = "ParameterTest.class"
        width = "500"
        height = "500" >
        < param name = " edge " value = "30 " >
```

```
        < param name = " maximum " value = " 5 " >
< / applet >
```

برنامج 4-1-4

عند تنفيذ هذا البرنامج يتم تمرير قيم الوسائط edge و maximum للـــــ applet، وتستخدم القيم في تحديد عدد وأبعاد المربعات الناتجة، كما في مخرجات البرنامج أدناه.



مخرجات البرنامج 4-1-4

تستخدم الـــ java applets كثيراً في تطبيقات الإنترنت خصوصا في مجال الترفيه، مثل الألعاب وتطبيقات الصور والأصوات والرسوم المتحركة. فالـــ java applets تتميز بجميع مميزات لغة جافا، مما يجعلها غنية بالعبارات الملائمة لكثير من التطبيقات، كما توفر درجة عالية من الأمن لضمان سرية وسلامة جهاز المستخدم. والسبب في زيادة

درجة الأمن التي توفرها الـــ applet أكثر من غيرها هو أنه يتم تحميلها من جهاز مقدم خدمة الإنترنت إلى أجهزة المستخدمين بخلاف غيرها من التطبيقات التي عادة ما تعمل على جهاز معين أو عدد محدود من الأجهزة. يمكن تلخيص مميزات الأمن والسرية التي توفرها الـــــ applets في النقاط التالية:

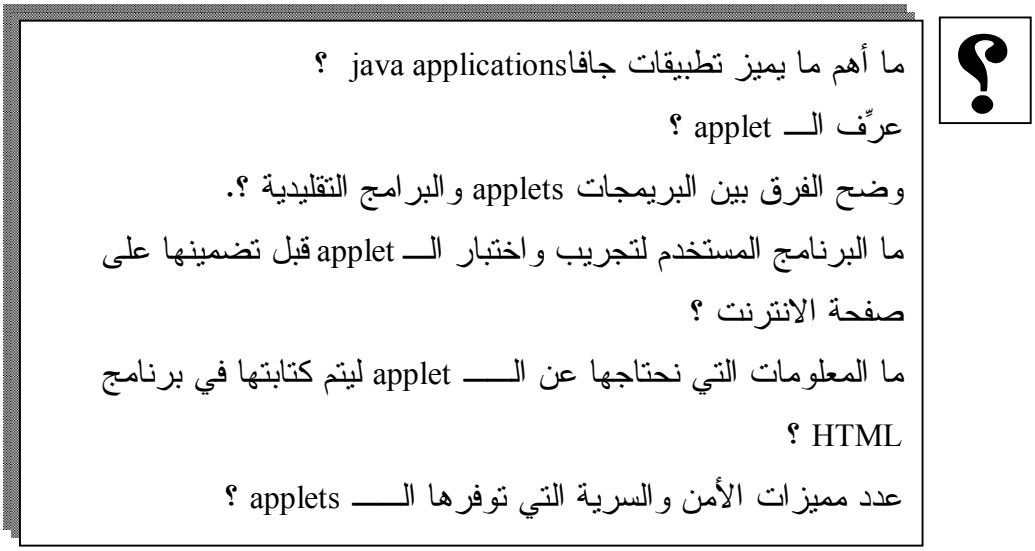
1- لا يسمح للـــ applet بالدخول إلى نظام ملفات الجهاز التي تم تحميلها عليه لا قراءة ولا كتابة.

2- لا يسمح للـــ applet بتشغيل برنامج آخر على جهاز المستخدم.

3- لا يسمح للـــ applet بتوصيل جهاز المستخدم مع أي جهاز آخر.

تمثل النقاط السابقة الفرق بين برامج الـــ applet وبرامج الـــ application. يمكن أن نحوّل أي application إلى applet مادام أنه يخضع لقوانين أمن الـــ applets السابقة، أما الـــ applets فيمكن أن تتحول إلى application بدون شـــروط.

أسئلة تقويم ذاتي



ما أهم ما يميز تطبيقات جافاjava applications ؟

عرِّف الـــ applet ؟

وضح الفرق بين البريمجات applets والبرامج التقليدية ؟.

ما البرنامج المستخدم لتجريب واختبار الـــ applet قبل تضمينها على صفحة الانترنت ؟

ما المعلومات التي نحتاجها عن الـــــ applet ليتم كتابتها في برنامج HTML ؟

عدد مميزات الأمن والسرية التي توفرها الـــــ applets ؟

2. الاستثناءات ومعالجة الاستثناءات

2 .1 الاستثناءات Exceptions

عند تنفيذ برنامج معين على جهاز حاسوب، هناك بعض الحالات غير المرغوبة التي قد تحدث أثناء تنفيذ البرنامج تؤدي إلى الحصول على نتائج غير صحيحة أو إلى انقطاع تنفيذ البرنامج. تعرف هذه الحالات عموما بأخطاء زمن التنفيذ run time errors وفي لغة جافا بالاستثناءات exceptions. تنقسم الاستثناءات من حيث أسباب حدوثها إلى ثلاثة أقسام:

أ- استثناءات لأسباب خارجية:

وهي أسباب تحدث بسببٍ لا علاقة له بالبرنامج نفسه، بل ببرنامج آخر أو نظام التشغيل أو جهاز آخر. مثال لذلك أن يحاول المستخدم تشغيل البرنامج، ولكن نظام التشغيل لا يستطيع توفير الذاكرة اللازمة لتشغيل البرنامج. أو أن يحاول البرنامج الوصول إلى ملف أو جهاز آخر، ولكنه مشغول أو غير جاهز للاستخدام لأي سبب. من الصعب التنبؤ بحدوث هذا النوع من الأخطاء لكونه خارجاً عن يد المبرمج تماماً.

ب- استثناءات لأسباب تتعلق بكتابة البرنامج:

وهذه الأخطاء صادرة عن المبرمج نفسه، حيث لا ينتبه إلى بعض العبارات في البرنامج والتي تكون صحيحة لغوياً فلا يعترض عليها المترجم، لكنها تؤدي إلى مشاكل أثناء تنفيذ البرنامج. من أمثلة هذا النوع من الأخطاء أن يقوم المبرمج بتعريف reference دون تعريف object، ثم يحاول مخاطبة الـــــ object الذي لا وجود له.

```
JButton button;
button.addActionListener(this);
```

يعتبر المترجم هذه العبارات صحيحة لأنها لا تخرق قواعد لغة جافا. ولكن عند تنفيذ العبارات يحدث exception نوعه null pointer بسبب أن المرجع button يشير إلى null، ويؤدي هذا الخطأ إلى انقطاع تنفيذ البرنامج.

مثال آخر لخطأ شائع من هذا النوع هو تعريف مصفوفة بطول معين، ثم مخاطبة حجرة خارج نطاق المصفوفة.

```
int  array[] = new  int[10];
```

فالمصفوفة array بها عشرة عناصر من عنصر رقم 0 إلى عنصر رقم 9. ويقع الكثيرون في الخطأ عندما يحتوي برنامجهم على عبارة مثل:

```
for(int  i=0;  i<=10;  i++ )
    array[i]=0;
```

فأثناء تنفيذ الحلقة، وعندما تكون قيمة المتغير i هي 10، يحاول البرنامج تنفيذ العبارة:

```
array[10]=0
```

ولأن القيمة 10 تقع خارج نطاق مدى فهرس المصفوفة array ينقطع تنفيذ البرنامج بسبب exception يعرف بـــــ array index out of bounds.

يعتبر هذا النوع من الأخطاء الأسهل اكتشافا وتصحيحا، وذلك يعتمد على براعة المبرمج والدقة في تجريب البرنامج قبل تسليمه للمستخدم.

ج- استثناءات تتعلق بمستخدم البرنامج:

وهذا النوع يتعلق بالبيانات التي يدخلها المستخدم للبرنامج، فمثلا قد يطلب البرنامج من المستخدم إدخال عمره بواسطة text field اسمه ageField ووضع هذه القيمة في متغير من نوع String بواسطة الدالة ()getText، ثم تحويل القيمة من String الى int بواسطة الدالة ()parseInt لاستخدامها لاحقا بالبرنامج.

```
String   s = ageField.getText();
int age = integer.parseInt(s);
```

وإذا قام المستخدم مثلا بإدخال القيمة "twenty " إلى البرنامج بدلا عن إدخال عمره كعدد، تفشل الدالة ()parseInt في محاولة التحويل من نص إلى عدد ويؤدي إلى حدوث خطأ يعرف بخطأ صياغة العدد number format exception وينقطع تنفيذ البرنامج .

تعتبر الأخطاء الرياضية أيضاً من أمثلة هذا النوع من الأخطاء. فإذا كان هناك برنامج يقوم بقسمة عددين، وقام المستخدم بإدخال القيمة صفر للمقام، يحدث خطأ أثناء إجراء عملية القسمة على الصفر يسمى arithmetic exception.

تعتبر من مهمة المبرمج التأكد من عدم حدوث هذا النوع من الأخطاء أيضاً عن طريق تجريب البرنامج وتوقع كل الاحتمالات للمدخلات والعمليات التي يجريها مستخدم البرنامج.

2.2 معالجة الاستثناءات Exception Handling

تعتبر الأخطاء المختلفة التي سبق ذكرها شائعة الحدوث عند تنفيذ البرامج المختلفة. وقد صممت لغة جافا عدداً من الـــ classes التي تعبر عن هذه الأخطاء، ووضعت الطرق الملائمة لمعالجة هذه الأخطاء عند حدوثها والتعامل معها. تعرف هذه الطرق بالـــ exception handling، والغرض الأساسي منها هو ألا تتأثر صحة واستمرارية البرنامج بحدوث الأخطاء.

تحتوي java.lang package على class اسمه Exception، يعبر هذا الـــ class عن خطأ من أي نوع يحدث أثناء تنفيذ البرنامج. هناك عدد كبير من الـــ classes تعرفه لغة جافا لتمثيل الأخطاء المختلفة، وجميعها ترث خصائص الـــ Exception class ، وسنتعرض إلى بعضها خلال الأمثلة.

توفر جافا آلية لمعالجة أخطاء زمن التنفيذ عن طريق مراقبة العبارات المتوقع حصول الخطأ أثناء تنفيذها والاستجابة لهذه الأخطاء في حال حدوثها. البرنامج التالي يوضح هذه الآلية:

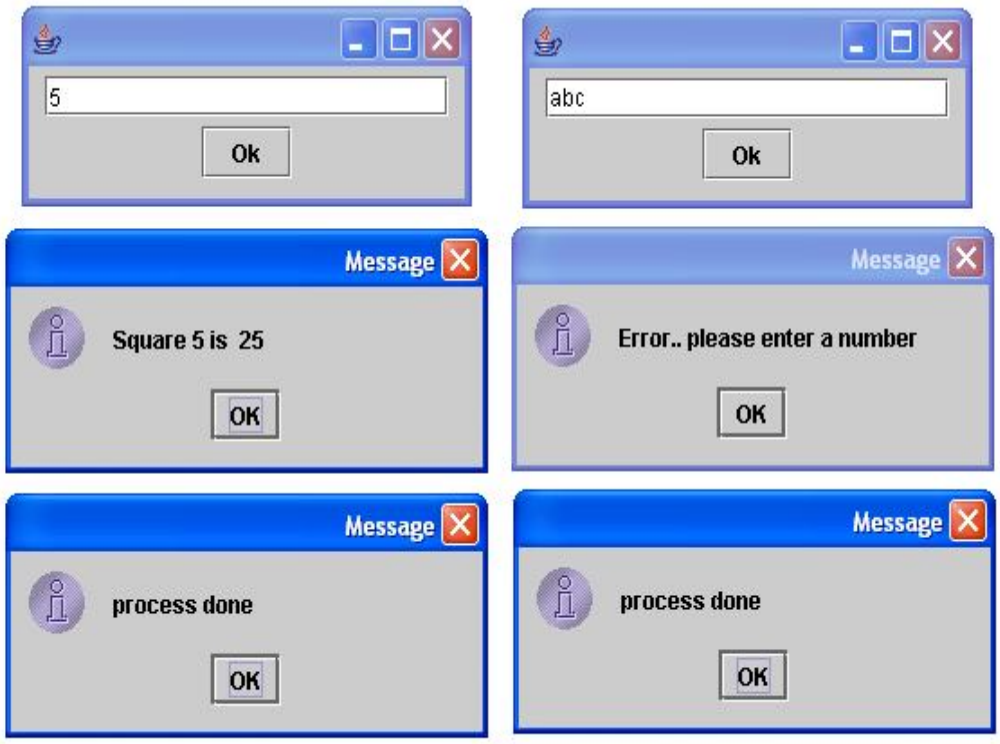
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public  class ExceptionTest1 extends JFrame implements ActionListener{
        JTextField input;
        JButton  ok;
        public ExceptionTest1(){
                Container  c = getContentPane();
                c. setLayout(new FlowLayout());
                input = new  JTextField ("Enter  Number",20);
```

```
                c.add(input);
                ok = new JButton("Ok");
                ok.addActionListener(this);
                c.add(ok);
                setSize(250,100);
                setVisible(true);
        }
        public  void  actionPerformed(ActionEvent  e){
                String s = input.getText();
                try{
                        int  number = Integer.parseInt(s);
                        int  square = number * number;
                        JOptionPane.showMessageDialog(this,"Square      "      +
number + " is  "  + square);
                }
                catch (NumberFormatException  ex) {
                        JOptionPane.showMessageDialog(this,     "Error..  please
enter a number");
                }
                finally {
                        JOptionPane.showMessageDialog(this, "process done");
                }


        }
        public  static  void  main(String  args[]){
                new  ExceptionTest1();
        }
}
```

برنامج 4-2-1

الغرض من البرنامج أعلاه أن يقوم المستخدم بكتابة عدد معين داخل الـــــ text field والضغط على الزر Ok للحصول على مربع العدد المعين. لكن المبرمج لا يضمن أن يقوم المستخدم بإدخال قيمة صحيحة، أي عددية. لذلك من المتوقع أن يحدث خطأ أثناء محاولة تنفيذ الدالة parseInt(). يتم وضع أي عبارات نتوقع حدوث خطأ فيها أثناء التنفيذ بداخل منطقة محصورة بين قوسين تبدأ بالكلمة المحجوزة try، تعرف هذه

المنطقة بـــــ try block. يعتبر تنفيذ ما بداخل هذه المنطقة بالكامل هو المرغوب، حيث يكون قد اكتمل تنفيذ جميع العبارات دون حدوث exception. أما إذا حدث exception، فإنه ينتج عن ذلك توليد object من نوع الـــ exception الذي حدث في هذا المثال يتولد object من نوع NumberFormatException وينقطع تنفيذ منطقة try لينتقل التحكم بعدها إلى منطقة catch والمتخصصة بالإمساك بهذه الـــــ exception التي تم توليدها، ولذلك نجد أن catch block تستقبل بين قوسيها object من نوع exception محدد، وهو في هذا المثال NumberFormatException. عندما يتولد object من نوع exception محدد، فإنه يبحث عن منطقة catch مناسبة لاستقباله ومعالجته. في المثال أعلاه إذا أدخل المستخدم عبارة غير عدديه سيحدث خطأ عند محاولة تنفيذ الدالة ()parseInt ويتولد object من نوع NumberFormatException، ينقطع تنفيذ منطقة try وينتقل التحكم إلى منطقة catch، وبما أن منطقة catch تستقبل object من نفس نوع الـــ object الذي تم توليده عند حدوث الخطأ، يتم تنفيذ منطقة catch وطباعة الرسالة Error..Please enter a number . يمكن أن توجد أكثر من منطقة catch واحدة لاستقبال exceptions من عدة أنواع مقابل منطقة try واحدة. في هذه الحالة عند حدوث خطأ، يتم البحث عن أول منطقة catch مناسبة لاستقبال نوع الخطأ الذي حدث، ويتم تنفيذها لوحدها ولا يتم تنفيذ أكثر من منطقة catch حتى إذا كان هناك أكثر من catch block واحدة ملائمة لاستقبال الخطأ الذي حدث. نلاحظ انه إذا تم تنفيذ منطقة try بنجاح، فإنه لا يتم تنفيذ أي منطقة catch نسبة لعدم حدوث أي خطأ. أما منطقة finally، فهي منطقة اختيارية يتم تعريفها إذا كانت هناك عبارات نرغب في تنفيذها في حال حدث خطأ أو لم يحدث. أي أنه إذا اكتمل تنفيذ البرنامج دون أخطاء، فإنه يتم تنفيذ منطقة try بالكامل إضافة إلى منطقة finally، وإذا حدث خطأ أثناء تنفيذ منطقة try، ينقطع تنفيذها ويتم تنفيذ منطقة catch المناسبة – إن وجدت – ثم تنفيذ منطقة finally. إذا حدث خطأ بالبرنامج ولم توجد منطقة catch مناسبة لمعالجة الخطأ. يضطر البرنامج إلى قطع التنفيذ والخروج.



مخرجات برنامج 4-2-1

من المثال السابق علمنا أن:

1- تحتوي منطقة try على العبارات التي نرغب في تنفيذها بصورة صحيحة، وقد يؤدي تنفيذها إلى حدوث exception من نوع معين.

2- تحتوي منطقة catch على عبارات نرغب في تنفيذها عند حدوث خطأ معين في منطقة try، لكي لا ينقطع تنفيذ البرنامج. لا يتم تنفيذ هذه المنطقة في حال عدم حدوث خطأ.

3- تحتوي منطقة finally على عبارات نرغب في تنفيذها في جميع الأحوال، أي إذا حدث أو لم يحدث خطأ.

يعتبر class Exception هو suber class لجميع الـــــ exceptions الأخرى، وجميعها

ترث صفات ومقدرات class exception. العبارة catch(Exception ex) لها المقدرة على معالجة أي خطأ يحدث بالبرنامج، حتى إذا كان من نوع number format أو أي نوع خاص من الأخطاء، حيث إن عبارة catch(Exception ex) تستطيع الإمساك بخطأ من نوع Exception أو أي exception وارث من class Exception. وحيث إن جميع الـــــ exceptions في جافا وارثة من class Exception لذلك فإن عبارة catch( ) Exception تستطيع معالجة أي خطأ يحدث في منطقة try في البرنامج. ويستحسن استخدامها إذا كنا غير متأكدين من نوع الخطأ الذي قد يحدث عند تنفيذ البرنامج.

تسمح لغة جافا بوجود أكثر من منطقة catch واحدة لكل منطقة try، حيث يمكن أن نتوقع حدوث أخطاء من عدة أنواع عند محاولة تنفيذ منطقة try. عند حدوث خطأ، يتم البحث عن أول عبارة catch مناسبة لمعالجة الخطأ حسب نوع الخطأ الذي تعالجه عبارة catch، وإذا لم توجد أي عبارة catch مناسبة، ينقطع تنفيذ البرنامج. المثال التالي يوضح كيفية تعامل البرنامج مع عبارات catch المتعددة.

```
import  javax.swing.*;
import  java.awt.*;
import  java.awt.event.*;
public class ExceptionTest2 extends JFrame implements ActionListener{
        String  names[] = { "Null Pointer Exception",  "Number Format
Exception",
        "Arithmetic Exception", "Class Not Found Exception"};
        JButton  buttons[];
        public  ExceptionTest2( ){
                Container  c = getContentPane( );
                c.setLayout(new  FlowLayout( ));
                buttons = new JButton[4];
                for (int i = 0; i<=3;  i++) {
                        buttons[ i ]= new JButton(names[ i ]);
                        buttons[ i ].addActionListener(this);
                        c.add(buttons[ i ]);
                }
                setSize(250, 200);
                setVisible(true );
```

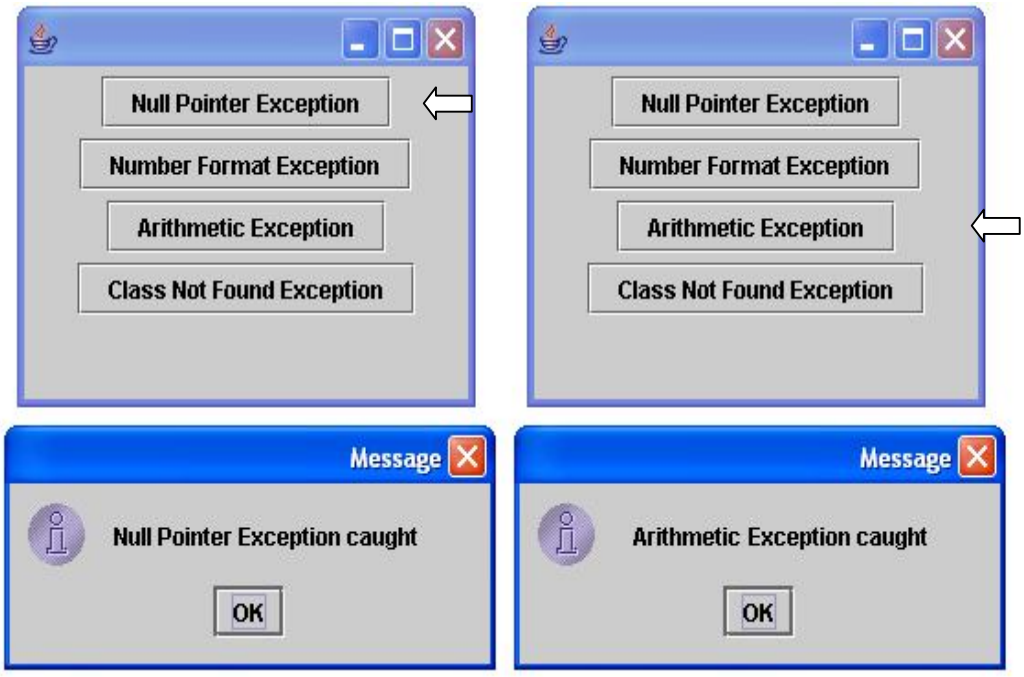
```
        }
        public  void  actionPerformed(ActionEvent  e) {
         try{
                if  (e.getActionCommand( ).equals ("Null Pointer Exception" ) )
                        throw new NullPointerException( );
                else if(e.getActionCommand( ).equals("Number Format
Exception") )
                        throw new NumberFormatException( );
                else if (e.getActionCommand( ).equals("Arithmetic Exception") )
                        throw new ArithmeticException( );
                else if (e.getActionCommand( ).equals("Class Not Found
Exception") )
                        throw new ClassNotFoundException( );
         }
         catch(NullPointerException  ex) {
                JOptionPane.showMessageDialog(this, "Null Pointer Exception
caught");
         }
         catch( NumberFormatException  ex) {
                JOptionPane.showMessageDialog( this, "Number Format
Exception caught");
         }
         catch(ArithmeticException  ex ) {
                JOptionPane.showMessageDialog(this, "Arithmetic Exception
caught");
         }
         catch(Exception  ex) {
                JOptionPane.showMessageDialog(this, "General Exception
caught");
         }
        }
        public  static  void  main(String  args[ ]){
                new  ExceptionTest2( );
        }
}
```

برنامج 4-2-2

عند الضغط على أحد الأزرار بالبرنامج، يتم توليد exception من نوع معين. لكن في هذا المثال، لا نعتمد على التوليد التلقائي لهذه الأخطاء عن طريق حدوثها، وإنما نستخدم

كلمة صريحة لنتعمد توليد هذه الأخطاء بواسطة الكلمة المحجوزة throw. تسمح جافا بالتحكم في توليد وإلقاء الـــــ objects من نوع exception خلال تنفيذ البرنامج، أي أنه ليس بالضرورة أن يحدث الخطأ الفعلي ليتولد object وإنما يمكن للمبرمج أن يختار متى يتم توليد أو إلغاء خطأ معين. تستخدم هذه الطريقة خاصة عندما يقوم المبرمج بتصميم الـــ exceptions الخاصة به كما سنرى لاحقا.

يحتوي المثال السابق على عدة مناطق catch تتنفذ واحدة منها فقط عند توليد والقاء exception بالبرنامج. فمثلا، عند الضغط على زر Null pointer Exceptionيتم توليد exceptionنوعه Null pointer Exception وينقطع تنفيذ منطقة tryويبدأ التحكم بالبحث عن منطقة catch مناسبة لاستقبال exception من نفس النوع، وهي عبارة catch الأولى فيتم طباعة العبارةNull pointer Exception caught، ولا يواصل التحكم البحث خلال عبارات catch الأخرى، بل ينتقل مباشرة إلى ما بعد جميع عبارات catch ومواصلة التنفيذ. ويحدث نفس الأمر عند الضغط على Number Format Exception أو Arithmetic Exception، حيث تتولد exceptions من الأنواع المعنية، وينتقل التحكم للبحث عن أول عبارة catchمناسبة، ويتم اختيار العبارة الثانية في حالة Number Format Exception والثالثة عند الضغط على زر Arithmetic Exception، ويتم طباعة العبارة الموجودة داخل منطقة catch. عند الضغط على الزر Class Not Found Exception، يتولد object من نوع ClassNotFoundException، وعند البحث عن عبارة catchمناسبة لمعالجة هذا النوع من الأخطاء، لا يتم إيجاد عبارة catch تستقبل object من نوع ClassNotFoundException. لكن آخر عبارة catchهي (catch(Exception e، وكما ذكرنا فإنها تستطيع استقبال object من أي نوع exception بما في ذلك ClassNotFoundException، وعليه يتم تنفيذ عبارة catch الأخيرة وطباعة العبارة General Exception caught. وفي حال لم توجد (catch (Exception e، لن يتم إيجاد عبارة catch مناسبة لاستقبال هذا النوع من الأخطاء، مما يؤدي إلى انقطاع التنفيذ.



مخرجات برنامج 4-2-2

توفر لغة جافا مجموعة كبيرة من الـــــ exceptions المختلفة للتعامل معها في البرامج حسب الحاجة، وسيتم التطرق للعديد منها خلال الفصول القادمة من هذا الكتاب. تسمح جافا كذلك للمبرمج أن يقوم بكتابة الـــــ exceptions الخاصة به في حال لم توفرها لغة جافا ليستخدمها عند الحاجة. ولكتابة exception جديد نوجز ثلاث خطوات يتبعها مثال لترجمة هذه الخطوات إلى برنامج:

1- تعريف class الـــــ exception الجديد: ويجب أن يرث الـــــ class خصائص الـــ Exception class.

2- توضيح شرط حدوث هذا الخطأ في البرنامج الذي يتعامل مع الخطأ الجديد.

3- نداء صريح لتوليد والغاء object من نوع الخطأ الجديد، حيث لن يكون هناك نداء تلقائي لهذا النوع.

في المثال التالي نريد أن نصمم أنواعاً جديدة من الأخطاء هي:

- PositiveException: ويتم تولده إذا أدخل مستخدم البرنامج عدداً موجباً.
- NegativeException: ويحدث هذا النوع إذا أدخل المستخدم عدد سالب.
- Zero Exception: إذا كان العدد الذي أدخله المستخدم هو الصفر، يتم توليد خطأ من هذا النوع.

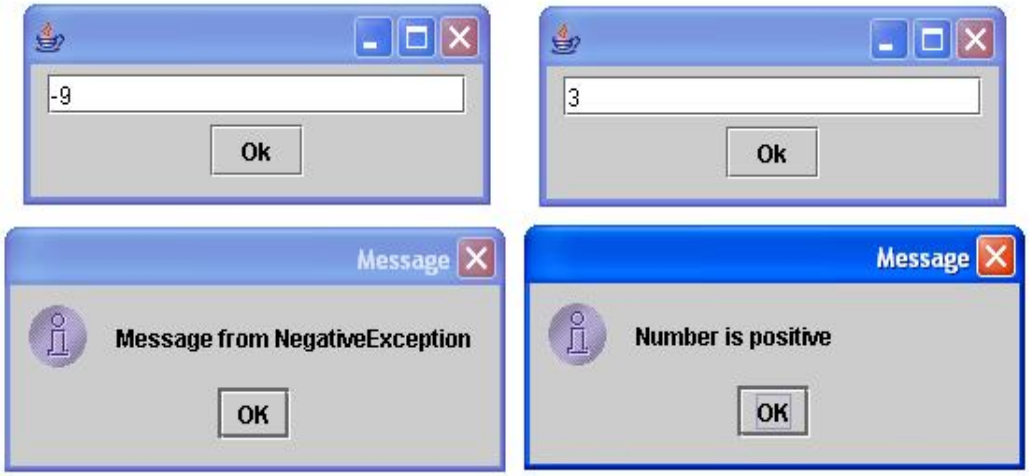
```
import  javax.swing.*;
import  java.awt.*;
import  java.awt.event.*;
public  class  ExceptionTest3 extends JFrame implements ActionListener{
        JTextField input;
        JButton  ok;
        public  ExceptionTest3(){
                Container   c = getContentPane();
                c.setLayout(new FlowLayout());
                input = new JTextField("Enter Number" , 20);
                c.add(input);
                ok = new JButton("Ok");
                ok.addActionListener(this);
                c.add(ok);
                setSize(250,100);
                setVisible(true);
        }
        public void actionPerformed(ActionEvent  e){
                String  s = input.getText();
                try{
                        int  number = Integer.parseInt(s);
                        if(number > 0)
                                throw  new  PositiveException();
                        if(number < 0)
                                throw  new  NegativeException();
                        if (number == 0)
                                throw  new  ZeroException();
                }
                catch(NumberFormatException  ex){
                        JOptionPane.showMessageDialog(null, ex.getMessage());
                }
                catch (PositiveException  ex){
                        JOptionPane.showMessageDialog(this , ex.getMessage());
```

```
                }
                catch(NegativeException ex){
                        // message in class NegativeException
                }
                catch(ZeroException ex){
                        JOptionPane.showMessageDialog(this, ex.getMessage());
                }
        }
        public static void main (String args[]){
                new ExceptionTest3();
        }
}

class PositiveException extends Exception{
        public PositiveException(){
                super("Number is positive");
        }
}

class NegativeException extends Exception{
        public NegativeException(){
                JOptionPane.showMessageDialog(null, "Message from
NegativeException");
        }
}

class ZeroException extends Exception{
}
```

برنامج 4-2-3

في المثال أعلاه، قمنا بتصميم ثلاثة classes تعبر عن أنواع الأخطاء الجديدة التي نرغب في استخدامها بالبرنامج. ولكي تكون الـــ objects من هذه الـــ classes قادرة على أن يتم إلغاؤها من داخل منطقة try معينة ويتم إمساكها في منطقة catch معينة، يجب أن ترث هذه الـــ classes خصائص class Exception. وفي هذا المثال كانت محتويات هذه الـــ classes مختلفة. نلاحظ استخدام دالة جديدة في منطقة catch، وهي

تابعة لـــ class Exception. تقوم الدالة getMessage() بإحضار الرسالة المخزنة داخل الـــ exception استعداداً لطباعتها للمستخدم ليتسنى له معرفة اسم ونوع الخطأ الذي حدث لتفادي حدوثه في المستقبل. كل exception يحتوي على رسالة ابتدائية، ولكن يمكننا وضع الرسالة الخاصة بنا من خلال نداء المشيد الخاص بالـــ class Exception باستخدام المؤشر super كما في class PositiveException. عند تنفيذ هذا البرنامج سيتم توليد exception معين عند الضغط على ok حسب القيمة المدخلة، فهي إما أن تكون عدداً موجباً أو سالباً أو صفراً أو قيمةً غير عددية. وبناء على ذلك يتم توليد الخطأ المعني وتنفيذ عبارة catch المناسبة.



مخرجات برنامج 4-2-3

خلال جميع الأمثلة السابقة، تمت معالجة الخطأ في نفس المكان الذي نشأ فيه. أي أن الخطأ الذي يتولد في منطقة try معينة تتم معالجته بواسطة عبارة catch الموجودة أسفل منطقة try. وهذا ليس ضرورياً، بل يمكن أن يحدث الخطأ في منطقة أو دالة أو class محدد، وتتم معالجته في مكان بعيد عن مكان حدوث الخطأ. وهذه ميزة مهمة في معالجة الأخطاء في لغة جافا، حيث يمكن أن نعالج أخطاءً قد تحدث في برامج بعيدة دون

الاضطرار إلى معرفة تفاصيلها، كما يمكننا كتابة تفاصيل معالجة الأخطاء التي يمكن أن تكون معقدة في مكان منفصل بعيداً عن عبارات البرنامج الأساسية، مما يزيد من سهولة ووضوح البرنامج. والمثال التالي يشرح هذه الخاصية.

```
public class ExceptionTest4 {
      public static void main(String args[]) {
                ExceptionTest4 ex4 = new ExceptionTest4();
           try {
                ex4.first();
                System.out.println("No Exceptions");
           }
           catch(Exception e) {
                e.printStackTrace();
           }
      }
      public void first() throws Exception {
           System.out.println("Method first does not catch the exception");
           second();
      }
      public void second() throws Exception {
           System.out.println("Throwing exception from method second");
           throw new Exception();
      }
}
```

برنامج 4-2-4

يبدأ تنفيذ البرنامج من الدالة main بنداء الدالة first، فينتقل التحكم إلى الدالة firstحيث يتم طباعة الرسالة على الشاشة، ثم ينتقل التحكم لتنفيذ الدالة second. تحتوي الدالة second على عبارتين، عبارة الطباعة ثم عبارة لتوليد exception من نوع Exepction، والملاحظ أنه لا تتم معالجة هذا الخطأ في الدالة second، حيث إنه لا وجود لمنطقة try أو catch بالدالة. وبالتالي فإن كل برنامج ينادي هذه الدالة second سيعاني من احتمال تولد خطأ من نوع Exception. وبما أن الدالة second لا تعالج هذا الخطأ، فسينتقل الخطأ منها إلى البرنامج الذي ينادي الدالة second، وهو في هذا المثال

الدالة first. لذلك من اللائق لمن يكتب دالة قد يؤدي تنفيذها إلى حدوث خطأ لا تتم معالجته داخليا الإشارة إلي وجود هذا الخطأ لكي يتنبه مستخدم هذه الدالة إلى المشكلة ويراعي حلها. هذه الإشارة تتم عن طريق استخدام الكلمة المحجوزة throws، التي تظهر في ترويسة الدالة للإشارة إلى نوع الخطأ الذي قد ينتج عند نداء هذه الدالة ولم يتم علاجه بداخلها. في ترويسة الدالة second نجد العبارة throw Exception، وهي تخبر منادي هذه الدالة - الدالة first – بأن الدالة second تحتوي على خطأ من نوع Exception غير معالج داخليا، وعليه على كل من ينادي هذه الدالة second أما أن يعالج هو هذا الخطأ، أو يصرح إلى غيره بوجود الخطأ، (ولن يسمح له المترجم بغير ذلك). وهذا ما حدث في الدالة first، حيث تنادي الدالة second والتي تحتوي على خطأ من نوع Exception، ولم تعالج الدالة first هذا الخطأ بداخلها، لذلك سينتقل هذا الخطأ إلى البرنامج الذي ينادي الدالة first، وهو الدالة الرئيسية main. وبما أن الدالة first لم تعالج الخطأ القادم إليها من الدالة second، وجب عليها أن تشير إلى وجود هذا الخطأ في ترويستها لتنبيه كل من ينادي هذه الدالة. نلاحظ أن الدالة first لا تحتوي خطأ محتملاً حدوثه في عباراتها، لكنها تنادي دالة بها خطأ محتمل حدوثه، وهي الدالة second، مما يجعل الدالة first أيضاً تحتوي على خطأ محتمل. نجد العبارة throws Exception في ترويسة الدالة firstتدل على وجود خطأ قد يحدث عن نداء الدالة ولم تتم معالجته. نلاحظ أن الدالة main قد عالجت الخطأ عن طريق نداء الدالة first داخل منطقة try بوجود العبارة (catch (Exception e، وإذا لم تتم هذه المعالجة ينقطع تنفيذ البرنامج. خطوات تنفيذ البرنامج هي:

1. البداية من الدالة main بنداء الدالة first.
2. انتقال التحكم للدالة first حيث يتم طباعة الرسالة ثم نداء الدالة second.
3. تنفيذ الدالة second بطباعة الرسالة ثم توليد object من نوع Exception.
4. يحاول الخطأ إيجاد المكان المناسب لمعالجته داخل الدالة second وعندما لا يجد ذلك ينتقل إلى البرنامج الذي قام بنداء الدالة second وهو الدالة first.

5. بنفس الطريقة لا يجد الخطأ المتجول عبارات مناسبة داخل الدالة first لاستقباله، فينتقل إلى الدالة main.
6. بما أنه حدث exceptin أثناء تنفيذ الدالة main فإنه ينقطع تنفيذ المنطقة try بعد نداء الدالة first مباشرة وينتقل التحكم إلى منطقة catch.
7. يتم الإمساك بالخطأ ومعالجته داخل المنطقة catch، وينتهي تنفيذ البرنامج بصورة طبيعية.

نلاحظ وجود الدالة ()PrintStackTrace التابعة لـــ Exception class والتي تقوم بطباعة معلومات عن الخطأ مثل نوعه ورقم السطر واسم الدالة التي حدث فيها الخطأ، إضافةً إلى متابعة انتقال الخطأ بين الدوال المختلفة إلى أن يتم الإمساك به. وتشبه الرسالة التي تطبعها الدالة تلك التي تظهر على الشاشة عند انقطاع البرنامج بسبب حدوث exception دون معالجة، وهي رسالة لا نرغب بظهورها لأنها تدل على وجود خطأ ما موجود بين عبارات البرنامج يظهر عند التنفيذ.

```
Method first does not catch the exception
Throwing exception from method second
java.lang.Exception
(at ExceptionTest4.second(ExceptionTest4.java:19
(at ExceptionTest4.first(ExceptionTest4.java:15
(at ExceptionTest4.main(ExceptionTest4.java:5
```

مخرجات برنامج 4-2-4

تعرضنا خلال هذا القسم لبعض الأنواع الشائعة من الأخطاء التي تظهر أثناء تنفيذ البرامج وكيفية معالجتها عند حدوثها. تعلمنا كيف نقوم بتعريف الأخطاء وتوليدها ومعالجتها وعملية انتقالها بين مناطق البرامج المختلفة. خلال الأقسام القادمة سنتعرض – بإذن الله – للمزيد من أنواع الأخطاء التي تعرفها جافا، وبعض الدوال التي تلزمنا بمعالجة الأخطاء التي قد تحدث أثناء تنفيذها.

3 . الملفات Files and Streams

1.3 أهمية الملفات

الملفات هي إحدى وسائل تخزين البيانات الهامة في الحاسوب. وتكمن أهمية الملفات للغات البرمجة في إمكانية تخزين البيانات الخاصة بالبرنامج والاحتفاظ بها حتى بعد تنفيذ البرنامج، مع إمكانية الوصول إليها واستخدامها عند إعادة تشغيل البرنامج أو بواسطة برامج أخرى.

2.3 التعامل مع الملفات

توفر جافا عدداً كبيراً من الـــ classes والموجودة في package java.io، ويمكن بواسطتها تعريف الملفات وكتابة البيانات المختلفة فيها وقراءة البيانات الموجودة بها. تعامل جافا البيانات الداخلة إلى الملفات والخارجة منها على أنها Stream من البيانات. الـــ stream هو مجرى لتدفق البيانات في اتجاه واحد، من الملف إلى البرنامج خلال عملية القراءة من الملف، أو من البرنامج إلى الملف خلال عملية الكتابة. يمكن التعامل مع الملفات بأنواعها باستخدام لغة جافا، حيث يمكن قراءة وكتابة أنواع البيانات المختلفة بما في ذلك الكائنات. في هذا الفصل نتطرق إلى أكثر الطرق شيوعاً للتعامل مع الملفات مع الإشارة لبعض الطرق الأخرى.

يمكن تعريف ملف في لغة جافا باستخدام الفئة File استعداداً لاستخدامه في البرنامج، وذلك بتحديد اسم الملف عند إنشاء الكائن. ومثال لإنشاء الملف in.dat بالمسار C:\java في نظام التشغيل windows:

```
File file = new File("C:\\java\\in.dat");
```

ومثال لإنشاء ملف باسم in.dat في المسار /java/ في نظام التشغيل Unix:

```
File file = new File("java/in.dat");
```

يمكن معرفة ما إذا كان الملف in.dat موجوداً مسبقاً أو يتم إنشاؤه لأول مرة باستخدام الدالة exists، كما يمكن الحصول على مسار الملف الكامل باستخدام الدالة getPath كالآتي:

```
if (file.exists( ) )
    System.out.println("The file path is: " + file.getPath( ));
```

ونستفيد من الدالة exists في التأكد من أن الملف المحدد موجود في حال أردنا القراءة منه، ومن أنه غير موجود عندما نرغب في إنشاء ملف جديد بهذا الاسم، حتى لا يتم مسح بيانات ملف عن طريق الخطأ.

لقراءة بيانات محفوظة في ملف معين، يتم تعريف كائن من نوع FileInputStream عن طريق تحديد اسم الملف الذي يحتوي على البيانات. يسمح الكائن من هذا النوع بفتح الملف للقراءة منه واستخدام البيانات في البرنامج. عند تعريف الملف، يتم التأكد من وجود ملف بالاسم المحدد، وإذا لم يوجد، يتولد exception من نوع FileNotFoundException، لأن فتح الملف لقراءة البيانات يعني أن الملف موجود سلفاً. ولكتابة البيانات من البرنامج وحفظها في ملف، يتم استخدام كائن من نوع FileOutputStream مع تحديد اسم الملف الذي نرغب بحفظ البيانات فيه. إذا لم يكن هذا الملف موجوداً مسبقاً، يتم إنشاؤه وحفظه في المجلد المحدد، وإذا لم يتم تحديد هذا المجلد، يتم حفظ الملف في المجلد الموجود فيه البرنامج. تستخدم الدالة read لقراءة byte واحد من الملف المفتوح للقراءة، والدالة write لكتابة byte واحد في الملف المفتوح للكتابة. فيما يلي مثال يوضح هذه العملية.

```
import java.io.*;
import java.awt.*;
import javax.swing.*;
public class FileTest extends JFrame{
        FileInputStream input;
        FileOutputStream output;
        String inFileName, outFileName;
        public FileTest(){
                super("Testing Files...");
```

```
                setSize(200, 200);
                setVisible(true);
                inFileName = JOptionPane.showInputDialog(this, "Enter the
name of the source file");
                outFileName = JOptionPane.showInputDialog(this, "Enter the
name of the destination file");
                try{
                        input = new FileInputStream(inFileName);
                        output = new FileOutputStream(outFileName);
                        byte b;
                        while ((b = (byte) input.read())!= -1){
                                output.write(b);
                        }
                }
                catch(FileNotFoundException e){
                        JOptionPane.showMessageDialog(this, "This file does not
exist");
                }
                catch(IOException e){
                        JOptionPane.showMessageDialog(this,"IO Exception");
                }
                finally{
                        try{
                                input.close();
                                output.close();
                        }
                        catch(IOException e){
                                e.printStackTrace();
                        }
                }
        }
        public static void main(String args[]){
                new FileTest();
        }
}
```

برنامج 4-3-1

في هذا البرنامج، وعند تعريف المتغير input، يصبح الملف الذي أدخل المستخدم اسمه جاهزاً ليقوم البرنامج بقراءة البيانات منه باستخدام الدالة read التي يوفرها class

FileInputStream. تتم قراءة محتويات الملف byte تلو الآخر بالمتغير b ثم تخزين هذه القيمة بالملف الثاني والذي حدده المستخدم لإجراء عملية الكتابة عليه. تبدأ القراءة من الملف من بدايته، وينتقل مؤشر القراءة أثناء عملية القراءة حتى يصل الملف إلى نهايته، وعندها تعيد الدالة read القيمة 1- لتتوقف حينها حلقة القراءة while. بعد الفراغ من التعامل مع أي ملف، يجب أن يتم إغلاقه باستخدام الدالة close. ينتج عن تنفيذ هذا البرنامج نسخ محتويات الملف الأول المحدد عند الإدخال في الملف الثاني.

استخدمنا خلال البرامج السابقة العبارة System.out.print لطباعة النصوص على الشاشة. تحتوي لغة جافا على class System.in لقراءة البيانات من الشاشة، والتي استخدمناها في كثير من البرامج السابقة. ولكن استخدام System.in ليس مباشراً كما هو الحال عند استخدام System.out، بل يجب التعامل معها عن طريق تعريف كائن من النوع DataInputStream، وهو يشبه FileInputStream المستخدم في المثال السابق، إلا أنه يتميز بإمكانية قراءة نص كامل في عبارة قراءة واحدة بدلاً عن byte. المثال البسيط التالي يبين استخدام System.in لقراءة نص من المستخدم.

```
class InputTest{
	public static void main(String args[]){
		DataInputStream in = new DataInputStream(System.in);
		String s = " ";
		System.out.print("Enter your name: ");
		try
		{
			s=in.readLine();
		}
		catch(Exception e)
		{
			System.out.println("ERROR");
		}
		System.out.println(s + ", you have entered data.");
	}
}
```

برنامج 4-3-2

يتم تحديد الملف – أو الوسيط – الذي سيستخدم للحصول على المدخلات النصية، وهو في هذا المثال System.in، ويمثل شاشة التنفيذ التي تعامل مثل معاملة الملفات. تستخدم الدالة readLine والتي يوفرها class DataInputStream لقراءة نص يدخله مستخدم البرنامج.

مثال للتنفيذ:

```
Enter your name:  Ali
Ali, you have entered data.
```

مخرجات برنامج 4-3-2

يمكن تطوير البرنامج السابق لعمل وحدة كاملة لقراءة جميع أنواع البيانات الأساسية، وذلك عن طريق تحويل النص الذي حصلنا عليه باستخدام الدالة readLine إلى أنواع البيانات الأخرى. تم استخدام البرنامج التالي لعملية الإدخال خلال الأمثلة السابقة.

```
import java.io.*;
class Input
{
	public static String readS()
	{
		DataInputStream in=new DataInputStream(System.in);
		String s=new String();
		try
		{
			s=in.readLine();
		}
		catch (Exception e)
		{
			System.out.println("ERROR");
		}
		return s;
	}
```

```
public static int readInt()
{
	String s=readS();
	int i=0;
	i=Integer.parseInt(s);
	return i;
}


public static char readChar()
{
	String s=readS();
	char c=' ';
	c=s.charAt(0);
	return c;
}



public static double readDouble()
{
	String s=readS();
	double d=0.0;
	d=Double.parseDouble(s);
	return d;
}


public static String readString()
{
	String s="";
	s=String.valueOf(readS());
	return s;
}


public static boolean readBoolean()
{
	String s=readS();
	boolean b;
	if (s.equals("true"))
```

```
            b=true;
        else
            b=false;
        return b;
    }

}
```

برنامج 4-3-3

تقوم الدوال readChar, readInt, readDouble, readString, readBoolean بتحويل القيمة التي تمت قراءتها باستخدام الدالة readLine إلى char, int, String, Boolean على التوالي. ويمكن استخدام هذا البرنامج للحصول على بيانات المستخدم من الشاشة كما شاهدنا خلال الكثير من الأمثلة.

توفر لغة جافا الكثير من الطرق للتعامل مع الملفات، مثل إمكانية قراءة الكائنات من الملفات باستخدام ObjectInputStream، وحفظ الكائنات في الملفات باستخدام ObjectOutputStream. والتعامل مع الملفات ذات الوصول العشوائي random access files وغيرها.

أسئلة تقويم ذاتي



وضح أهمية الملفات للغات البرمجة .

عرِّف الـــ stream.

يمكن معرفة ما إذا كان الملف موجوداً مسبقاً أو يتم إنشاؤه لأول مرة باستخدام الدالة .....

يمكن الحصول على مسار الملف الكامل باستخدام الدالة .....

بماذا يتميز الكائن من النوع DataInputStream ؟

4. البرمجة المتعددة

1.4 أهمية البرمجة المتعددة

في بداية عصر الحاسوب، كانت مقدرات أجهزة الحاسوب محدودة للغاية، فقد كان بإمكانه تشغيل برنامج واحد في اللحظة الواحدة. وربما لم تكن هذه الظاهرة تمثل عيباً في أجهزة الحاسوب في ذلك الوقت نظراً لمحدودية انتشاره وتطبيقاته. لكن مع انتشار الحاسوب اليوم، والعدد الهائل للتطبيقات والخدمات التي يوفرها الحاسوب لعدد ضخم من المستخدمين، كان لابد من طريقة تسمح له بتنفيذ أكثر من برنامج واحد في آن واحد. ونلاحظ هذه الظاهرة كثيرا عند استخدام الحاسوب. فمثلاً، قد نرغب في تحميل صفحة من الإنترنت في نفس الوقت الذي نستمع فيه لملف صوتي آنياً أثناء طباعة بعض الأوراق. إذا كان جهاز الحاسوب لا يسمح لنا إلا بتنفيذ برنامج واحد في اللحظة الواحدة، يجب أن يكتمل تحميل الصفحة أولا ثم انتهاء ملف الصوت قبل أن يسمح بمباشرة الطباعة، مما يضيع الوقت ويبعث ملل المستخدم من انتظار انتهاء بعض الإجراءات ليتمكن من مزاولة عمله. هناك أيضا بعض البرامج الخاصة بجهاز الحاسوب والتي أصبح من الممكن إن تعمل أثناء تنفيذ البرامج الأخرى، ومثال لهذه البرامج مضادات الفيروسات anti-viruses. تعرف مقدرة جهاز الحاسوب على تنفيذ برامج متعددة آنياً بالمعالجة المتعددة multi-processing. والمعالج processor هو الذي يحدد توفر هذه الخاصية بجهاز الحاسوب أو انعدامها.

نعلم أن المعالج هو المسئول عن تنفيذ أوامر البرامج المختلفة، وهو ينفذ تعليمة واحدة في اللحظة الواحدة. كيف إذاً يقوم المعالج بتنفيذ أكثر من برنامج واحد في اللحظة الواحدة؟ حقيقة الأمر أن المعالج لا يقوم بتنفيذ عدد من البرامج في اللحظة الواحدة، وإنما يقوم بتقسيم الوقت بين البرامج التي نرغب في تنفيذها فيما يعرف بـــ time sharing. يبدأ المعالج بتنفيذ تعليمات البرنامج الأول لفترة زمنية محددة سلفاً، ثم ينتقل

إلى البرنامج التالي وينفذ تعليماته لنفس الفترة، ثم ينتقل للبرنامج الذي يليه، وهكذا حتى يمر على جميع البرامج التي تنتظر التنفيذ، ثم يعود لمواصلة تنفيذ البرنامج الأول وهكذا إلى إن يكتمل تنفيذ جميع البرامج. ولذلك يبدو للمستخدم وكأنه يتم تنفيذ البرامج كلها في نفس الوقت.

إن أهم ميزة للمعالجة المتعددة هي الاستفادة القصوى من زمن المعالج واستغلاله في تنفيذ تعليمات البرامج المختلفة، مما يحقق سرعة أكبر في تنفيذ البرامج. فالبرنامج الذي يتعرض لتأخير في تنفيذه لأسباب مثل عمليات الإدخال والإخراج والتي تتطلب زمناً طويلاً، لا يعطل البرامج الأخرى أو يبقيها في حالة انتظار لاكتمال تنفيذه، بل يتحول المعالج إلى تنفيذ البرنامج التالي ريثما يكمل الأول عملياته الخارجية ليعود بعدها إلى صف البرامج التي تنتظر دورها في المعالجة.

تعرف البرامج التي يتم تنفيذها على التوازي مع برامج أخرى بالـــ process أوthread، كما تعرف برمجة هذا النوع من البرامج بالبرمجة المتعددة multi-programming أو multi-threading. تدعم لغة جافا هذا النوع من البرمجة، وقبل أن نباشر بكتابة البرامج.

أسئلة تقويم ذاتي



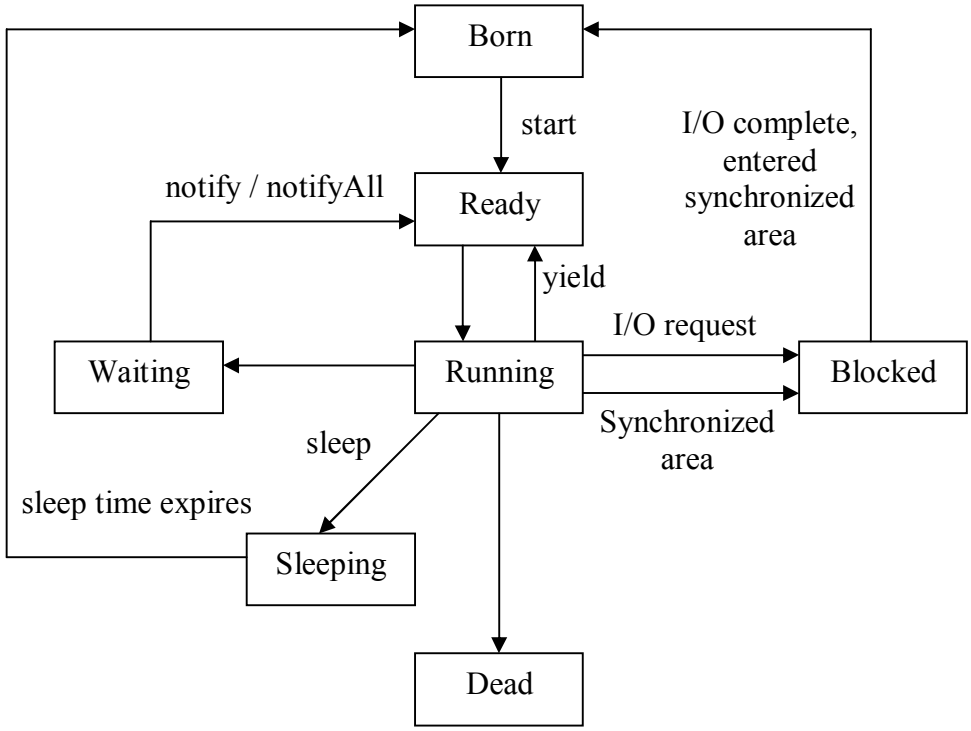
1- قدم مثالاً لبرامج تعمل أثناء تنفيذ البرامج الأخرى .

2 – ما أهم ميزة للمعالجة المتعددة ؟

3 – ما الذي يحدد توفر خاصية الـــ multi-processing بجهاز الحاسوب أو انعدامها ؟

4 – تعرف البرامج التي يتم تنفيذها على التوازي مع برامج أخرى بالـــ .....، كما تعرف برمجة هذا النوع من البرامج بال..... أو .....

2.4 دورة حياة الـــ Thread (Thread life – cycle)



شكل 4-4-1: دورة حياة الـــ thread

(1) Born : تبدأ دورة حياة الـــ thread بإعدادها للعمل، حيث يتم تحديد المطلوب منها إنجازه، وذلك يتمثل في تعريفها وإنشائها في البرنامج.

(2) Ready : بعد إنشاء الـــ thread يبدأ تشغيلها بواسطة الدالة start، حيث تنتقل إلى حالة ready، وهي تعبر عن البرامج التي تنتظر دورها في المعالجة، ومتى ما جاء دورها وتوفر المعالج، يبدأ تشغيلها وتصبح في حالة running.

(3) Running: تكون الـــ thread في حالة running حينما تكون تعليماتها قيد التنفيذ بواسطة المعالج، وتكون الـــ thread في هذه الحالة لفترة زمنية محددة، وبعدها ينتقل المعالج لتنفيذ thread أخرى، وتعود حينها هذه الـــ thread إلى حالة ready. يمكن أن

نتعمد قطع تنفيذ المعالج لـــ thread محددة باستخدام الدالة yield، والتي تحول الـــ thread إلى حالة ready، ويقوم المعالج بتنفيذ thread أخرى موجودة في حالة ready.

(4) Blocked : يمكن إن يتم إيقاف تنفيذ الـــ thread لفترة، وذلك بسبب عمليات إدخال وإخراج أو لوجود synchronized code كما سنعرف لاحقا. وتعود الـــ thread إلى حالة ready بعد الفراغ من عمليات الإدخال والإخراج أو التصريح بدخول المنطقة المعرفة synchronized.

(5) Sleeping : يمكن إن يتم قطع تنفيذ thread معينة لفترة زمنية محدودة باستخدام الدالة sleep، والتي نحدد لها الفترة الزمنية التي نرغب بانقطاع التنفيذ خلالها. تعود الـــــ thread إلى حالة ready عند نهاية تلك الفترة.

(6) Waiting : قد نرغب بأن يتوقف تنفيذ thread معينة لفترة غير محددة. يمكن ذلك باستخدام الدالة wait، وتبقى الـــ thread في حالة توقف عن التنفيذ حتى تقوم thread أخرى بتشغيلها عن طريق الدالة notify والتي تنقل thread واحدة من حالة waiting إلى حالة ready، أو باستخدام الدالة notifyAll لتنتقل جميع الـــ treads الموجودة في الحالة waiting إلى الحالة ready.

(7) Dead : عند انتهاء تنفيذ الـــ thread نهائيا، تكون قد أدت واجبها وأكملت المطلوب منها، فتصل إلى آخر حالة في دورة حياتها ويتوقف تنفيذها.

للبرمجة المتعددة ثلاث حالات:

أ- أن تكون البرامج مستقلة ويتم تنفيذها بالكامل في نفس الوقت، مثال لذلك تشغيل محتويات قرص مدمج CD أثناء تصفح الإنترنت.

ب- أن تكون البرامج مرتبطة أو معتمدة على بعضها البعض. أي أن تكون هناك قيود على ترتيب تنفيذها، مثلا المخرجات من برنامج معين هي المدخلات لبرنامج ثاني. في هذه الحالة يجب التأكد من انتهاء تنفيذ البرنامج الأول قبل بداية تنفيذ البرنامج الثاني.

ج- أن تكون البرامج عبارة عن نسخ متعددة من نفس البرنامج، مثلا عدة threads تبحث عن رقم معين موجود بين مليون رقم.

أسئلة تقويم ذاتي



1 - يمكن أن نتعمد قطع تنفيذ المعالج لـــ thread محددة باستخدام الدالة .....

2 - يمكن إن يتم قطع تنفيذ thread معينة لفترة زمنية محدودة باستخدام الدالة .....

3 - عندما نرغب بأن يتوقف تنفيذ thread معينة لفترة غير محددة نستخدم الدالة .....

4 - الدالة ..... تنقل جميع الـــ treads الموجودة في الحالة waiting إلى الحالة ready.

5 - للبرمجة المتعددة ثلاث حالات. اذكرها ؟.

3.4 أولوية تنفيذ الـــ Threads

يمكن أن تختلف الـــــ threads من حيث أولوية التنفيذ، كأن يكون تنفيذ أحدها أهم من الآخر. مثلا اكتشاف مضاد الفيروسات لفيروس في ملف هو أمر طارئ يمكن أن يقطع لأجله برنامجاً آخر لإخطار المستخدم بوجوده وإجراء اللازم للتخلص منه. بينما تشغيل برنامج جامع النفايات garbage collector لتحرير خانات الذاكرة غير المستغلة بواسطة البرامج لا يعتبر أمرا مهما يقطع لأجله برنامج المستخدم. لذلك نجد أن thread من نوع البرنامج الأول ذات الوظيفة العاجلة ستكون لها أولوية أعلى من برامج المستخدم ذات الطبيعة العادية، بينما thread من نوع البرنامج الثاني والتي يمكن تأجيل تنفيذها لحين فراغ المستخدم من تنفيذ برامجه تكون لها أولوية أقل من برامج المستخدم. يتم تحديد

أولوية الـــ thread بواسطة الدالة set Priority. إذا كان هناك عدد من الـــــ threads بأولويات مختلفة جاهزة للتنفيذ، يقوم المعالج بتنفيذ الـــ thread ذات الأولوية الأعلى حتى تنتهي، ثم يبدأ في تنفيذ الـــ thread ذات الأولوية الأقل. إذا كان هناك أكثر من thread تشترك في الأولوية، يقسم المعالج زمن التنفيذ عليها بالتساوي كما سبق شرحه، وبعد اكتمال تنفيذها جميعا ينتقل للـــ threads ذات الأولويات الأقل. تعتبر الأولوية 1 هي أقل أولوية للـــــ thread في لغة جافا، وأعلى أولوية ممكنة هي 0 . وإذا لم يتم تحديد أولوية معينة للـــ thread، تعطى أولوية عادية 5. برنامج جامع النفايات garbage collector هو thread لها أولوية منخفضة، لأنه مصمم للعمل عندما لا يحتاج برنامج المستخدم إلى المعالج، فوظيفته مساعدة برامج المستخدم وزيادة كفاءتها وليس تعطيلها وتأخيرها. فيما يلي برنامج يشرح كيفية تعريف thread بلغة جافا، وملاحظة سلوكها خلال مراحل حياتها المختلفة.

```
public class MyThread extends Thread{
        int  sleepTime;
        public MyThread(String name){
                super (name);
        }
        public void run( ) {
                System.out.print  ("Starting thread : " + getName( ) );
                try{
                        sleepTime = (int) (Math.random( ) * 10000);
                        System.out.println (", sleeping for " + sleepTime + "
milliseconds");
                        Thread.sleep(sleepTime);
                }
                catch(InterruptedException  e){
                        System.out.println(e.getMessage());
                }
                System.out.println(getName( ) + " finished");
        }
        public  static  void  main(String  args[ ]){
                MyThread  first =  new MyThread( "First");
```

```
                MyThread  second = new MyThread( "Second" );
                MyThread  third =  new MyThread( " Third " );
                first.start( );
                second.start( );
                third.start( );
                System.out.println( "Main method done.. " );
        }
}
```

برنامج 4-4-1

لكي يكون البرنامج عبارة عن thread، يجب أن يرث الـــ class المعنـــي class thread، وهو الذي يعطيه جميع الخصائص التي تجعله قادراً على التنفيذ آنياً مع برامج أخـــرى وتقاسم زمن المعالج فيما بينها. يوجد class thread فـــي package java.lang، ويحتـــوي على constructor يستقبل string تستخدم كاسم يمكن أن يستخدم للتفريـــق بـــين الـــــــ threadsالمختلفة، خاصة إذا كانت متشابهة كما في هذا المثال.

يمكن الوصول إلى هذا الاسم فيما بعد باستخدام الدالة getName. نضع كل ما نرغب أن تقوم به الـــ thread عند تشغيلها بداخل الدالة run، والتي يتم تنفيذ عباراتها تلقائياً عند نداء الدالة start. قد ينتهي تنفيذ جميع عبارات الدالة run أو قد ينقطع تنفيذها بسبب أحد الأسباب التي وردت سابقاً، والتي تؤدي بالـــ thread إلى الانتقال إلى حالة أخرى لفترة معينة قبل إن تعود إلى حالة ready لتصبح جاهزة لمواصلة التنفيذ. وعندما يحين دورها في المعالجة، ستواصل الدالة run تنفيذ عباراتها ابتداء من المكان الذي انقطع عنده التنفيذ. قد يتوقف تنفيذ run أيضا إذا ظهرت thread ذات أولوية أعلى، لتستمر بعد نهاية تنفيذ تلك الـــthread. تقوم الـــ thread في هذا المثال بطباعة عبارة :Starting thread يليها اسم الـــــ thread المعينة والزمن الذي ستتوقف خلاله عن التنفيذ. بعدها تنتقل إلى الحالة sleeping عن طريق نداء الدالة sleep لزمن عشوائي تم توليده باستخدام الدالة ( )Math.random والتي تولد رقماً عشوائياً بين صفر وواحد. لذلك ينتج عن العبارة 10000 * ( )Math.random عدد عشوائي بين 0 و 10000. ولأن الدالة sleep تستقبل عدداً صحيحاً يمثل زمن توقف تنفيذ الـــــ thread عن التنفيذ بالملي ثانية millisecond،

لذا يتوقف عمل الـــــ thread لزمن عشوائي بين 0 و10 ثواني يمكن خلالها تنفيذ threads أخرى. وبعد أن تعاود الـــــ thread التنفيذ، تقوم بطباعة العبارة finished مع توضيح اسم الـــــthread. يحتوي البرنامج على الدالة main وفيها يتم توليد ثلاثة threads بالأسماء first , second, third. بعد ذلك يتم تشغيلها بتنفيذ الدالة start والتي تقوم تلقائيا بمناداة الدالة run وبداية تنفيذ الـــthread. ننوه إلى أن الدالة main هي نفسها عبارة عن thread يكون تنفيذها مستقلاً عن بقية الـــــ threads وتتنافس معها على زمن المعالج. يصبح البرنامج أعلاه عبارة عن أربعة threads يتم تنفيذها آنيا. وبما أن لجميعها نفس الأولوية، يمكن لأي منها أن يستهل التنفيذ، ويعتمد إنهاء التنفيذ على قيمة المتغير sleepTime لكل thread، ويختلف ترتيب وزمن تنفيذ الـــــ threads كل مرة يتم فيها تشغيل البرنامج.

مثال للتنفيذ:

```
Starting thread: First, sleeping  for 6321 milliseconds
Starting thread: Second, sleeping  for 1178 milliseconds
Starting thread: Third, sleeping  for 5304 milliseconds
Main method done ..
Second Finished
Third Finished
First Finished
```

مثال آخر للتنفيذ :

```
Starting thread: First, sleeping for  162  millisecond
Main method done ..
Starting thread: Second, sleeping for  2714  millisecond
First Finished
Starting thread: Third, sleeping for  4592  millisecond
Second Finished
Third Finished
```

مخرجات برنامج 4-4-1

لاحظ في هذا التنفيذ أن الدالة main أكملت تنفيذ خطواتها قبل أن تبدأ second و third بالتنفيذ.

إن الوراثة من class thread ليست هي الطريقة الوحيدة لجعل البرنامج thread، وقد يكون البرنامج وارثاً أساسا من JFrame أو JApplet أو أي class آخر، ومن المتوقع غالبا أن تكون الـــ thread وارثة من class آخر. Runnable هو interface يمكن للبرنامج أن يقوم بتعريفه فيصبح thread مع إتاحة الفرصة للـ thread أن يرث خصائص class آخر. يحتوي interface Runnable على دالة واحدة هي run يجب تعريفها. فيما يلي مثال لكيفية تعريف وتشغيل thread باستخدامinterface Runnable:

```
class MyThread implements Runnable {
        public void run ( ) {
            // any code
        }
        public static void main(String args[ ]) {
            MyThread runnable = new MyThread ( );
            Thread  task = new Thread(runnable);
             task.start( );
        }
}
```

برنامج 4-4-2

نلاحظ هنا أن تعريف الـ object لن يكون مباشراً كما في المثال السابق. فالـ runnable object ليس thread تحديداً، ولتحويله إلى thread يتم إنشاء object من thread class وتمرير runnable كوسيط، ليصبح المتغير task هو الـــ thread التي يبدأ تشغيلها عند تنفيذ العبارة task.start( ).

أسئلة تقويم ذاتي



1 - ما الدالة التي يتم بواسطتها تحديد أولوية الـــ thread ؟

2 – اشرح كيف يقوم المعالج بتنفيذ عدد من الـــ threads بأولويات مختلفة جاهزة للتنفيذ.

4.4 التشارك في المصادر

كما علمنا يمكن أن تكون الـــــ threads عبارة عن برامج مستقلة تعمل على حدة، ويمكن أن تكون أجزاء من نفس النظام. إذا كان هناك عدد من الـــــ threads تنجز وظائف في نظام واحد، فمن المحتمل أن تتشارك في بعض المصادر مثل الذاكرة. وأن تعامل أكثر من جهة مع نفس مواقع الذاكرة قد يؤدي إلى أخطاء في البيانات إن لم تكن هناك إدارة جيدة للذاكرة تتأكد من صحة البيانات والعمليات عليها في كل الأوقات.

تمثل الـــ threads مثالاً لهذا التشارك في المصادر. ولتوضيح المسالة أكثر، لنفرض أن لشركةٍ ما حساب في أحد البنوك به مبلغ من المال. ومسموح لكل من مدير ونائب مدير الشركة بسحب المال من هذا الحساب. ولنفرض أن مدير الشركة أراد أن يسحب مبلغ 2000 جنيه من الحساب الذي يحتوي على 8000 جنيه. يتكون برنامج سحب المال في هذا البنك من ثلاث خطوات:

1- قراءة القيمة الحالية للرصيد.

2- الإجراءات المكتبية ومطابقة التوقيع.

3- كتابة الرصيد الجديد.

جلس مدير الشركة مع موظف البنك الذي شرع في تنفيذ مراحل البرنامج الثلاث. فقرأ القيمة الحالية للرصيد وهي 8000، ثم شرع في إكمال الإجراءات المكتبية. وفي هذا الأثناء، ذهب نائب رئيس الشركة إلى فرع آخر للبنك لسحب مبلغ 1000 جنيه، وجلس مع الموظف الذي قرأ قيمة الرصيد الحالي 8000 جنيه، وأخذ يكمل الإجراءات المكتبية

أيضاً. فرغ الموظف الأول من إجراءات المدير وأعطاه 2000 جنيه وأدخل القيمة الجديدة للرصيد وهي 8000 ـــ 2000 = 6000 جنيه، ورحل المدير. ثم انتهى الموظف الثاني وأعطى نائب المدير 1000 جنيه، وسجل القيمة الجديدة للرصيد وهي 8000 ـــ 1000 = 7000. من هنا تظهر المشكلة في رصيد البنك، حيث من المفترض أن يكون الرصيد النهائي هو 8000 ـــ 2000 ـــ 1000 = 5000 جنيه، بينما قيمة الرصيد الموجود في نظام البنك هو 7000. تظهر أخطاء البيانات بسبب استخدام أكثر من برنامج لهذه البيانات في نفس الوقت، عندما يكون هناك فاصل زمني بين قراءة البيانات وكتابتها، مثل الإجراءات المكتبية في هذا المثال. لترجمة السيناريو السابق إلى برنامج جافا، سنقوم بتعريف thread لكلٍ من المدير ونائب المدير، إلى جانب objectمشترك بينهما يعبر عن الحساب، والذي يحتوي على دالة لسحب المال من الحساب.

```
class  Client extends  Thread {
        Account account;
        int money;
        public Client(Account  a,  int  i) {
                account = a;
                money = i;
        }
        public void run( ) {
                System.out.println("Taking " + money + " from account");
                account.withdraw(money);
                System.out.println("Complete withdraw of " + money);
        }
}
class Account{
        int balance;
        public Account(int i){
                balance = i;
        }
        public void withdraw(int money){
                System.out.println("Old balance = "+ balance);
                int temp = balance;  // read the initial balance
                try {
```

```
                Thread.sleep(5000);     // desk procedures
            }
            catch(InterruptedException e){
            }
            temp -= money;
            balance = temp;   // write the new balance
            System.out.println("New balance = " + balance);
        }
}
public class MainApp{
        public static void main(String args[ ]) {
            Account a = new Account(8000);
            Client client1 = new Client(a, 2000);
            Client client2 = new Client(a, 1000);
            client1.start();
            client2.start();
        }
}
```

برنامج 4-4-3

يبدأ تنفيذ البرنامج من الدالة main حيث يتم إعداد رصيد الحساب بالقيمة الابتدائية 8000 ثم إنشاء thread لسحب 2000 جنيه من الحساب a و thread آخر لسحب مبلغ 1000 جنيه من نفس الحساب. عندما يتم نداء الدالة run، تقوم بنداء الدالة withdraw والتي تقوم بعملية السحب من الحساب في عدة خطوات. تبدأ العملية بقراءة الرصيد الابتدائي وتضع قيمته في المتغير المحلي temp. تستخدم الدالة sleep لتمثيل الإجراءات المكتبية، فتقوم بتعطيل التنفيذ لفترة خمس ثواني قبل أن تكتب قيمة الرصيد الجديد، ولتعطي فرصة للزبون الآخر بالسحب من الرصيد. يمكن استبدال الدالة sleep بأي عبارات يستغرق تنفيذها وقتاً طويلاً نسبياً مثل عبارة لقراءة المبلغ المطلوب سحبه من الحساب، وذلك لكي يقوم المعالج بإيقاف تشغيل الـ thread واستبدالها بـ thread أخرى تنتظر المعالج كي تظهر مشكلة الخطأ في البيانات، لأن كلاً من البرنامجين يتعاملان مع نفس موقع الذاكرة account . بعد انتهاء فترة نوم الـ thread، تقوم بطرح المبلغ المطلوب

سحبه من المبلغ الابتدائي ثم حفظ المبلغ الجديد بالحساب، ثم تطبع على الشاشة القيمة المتبقية من الرصيد.

مثال للتنفيذ :

```
Taking 2000 from account
Old balance = 8000
Taking 1000 from account
Old balance = 8000
New balance = 6000
Complete withdraw of 2000
New balance = 7000
Complete withdraw of 1000
```

مخرجات برنامج 4-4-3

نلاحظ هنا أن القيمة النهائية للرصيد هي 7000، أي أكثر بألفين من القيمة الصحيحة له. والطريقة الوحيدة لضمان صحة البيانات في مثل هذه الحالات هو أن نمنع تعامل أكثر من برنامج أو thread واحد مع نفس مواقع الذاكرة، أي أن نحظر دخول thread محددة إلى موقع ذاكرة معين طالما أن هناك thread أخرى تتعامل مع هذا الموقع. تستخدم الكلمة المحجوزة synchronized في ترويسة الدالة لكي تمنع تنفيذ أكثر من thread واحدة للدالة. عندما تحاول thread تنفيذ دالة synchronized، إذا كانت الدالة غير مستخدمة من قبل أي برنامج آخر يتم تنفيذ الدالة، لكن إذا كان هناك أي برنامج يقوم بتنفيذ هذه الدالة يتم وضع الـــــ thread في حالة blocked إلى أن ينتهي البرنامج من تنفيذ الدالة، فيتم إعطاء الإذن للـــــ thread بالتنفيذ، فتنتقل من حالة blocked إلى حالة ready ومن ثم إلى حالة running بمجرد أن يأتي دورها في المعالجة.

فيما يلي تعديل لـــ class Account ليوضح حل المشكلة:

```
class Account{
        int balance;
        public Account(int i){
                balance = i;
        }
```

```
        public synchronized void withdraw(int money){
                System.out.println("Old balance = "+ balance);
                int temp = balance;  // read the initial balance
                try {
                        Thread.sleep(5000);     // desk procedures
                }
                catch(InterruptedException e){
                }
                temp - = money;
                balance = temp;   // write the new balance
                System.out.println("New balance = " + balance);
        }
}
```

برنامج 4-4-4

ليكون تنفيذ البرنامج الجديد هو:

```
Taking 2000 from account
Old balance = 8000
Taking 1000 from account
New balance = 6000
Complete withdraw of 2000
Old balance = 6000
New balance = 5000
Complete withdraw of 1000
```

مخرجات برنامج 4-4-4

في هذه الحالة، لا مجال لحدوث خطأ في الحساب، لأنه لن يسمح بتشغيل الـــــ thread الثانية للدالة withdraw حتى تفرغ الـــــ thread الأولى من تنفيذها. ورغم أن التنفيذ سيستغرق فترة أطول، إلا أن صحة البيانات تعتبر أهم من سرعة البرنامج في مثل هذه التطبيقات.



يعتبر الـــ multi threading ذا أهمية بالغة في أنظمة التشغيل والبرمجة المتقدمة. ويفضل استخدامها في البرامج ما أمكن وحيثما توجد نسخ متعددة من البرامج أو الدوال، للاستفادة القصوى من زمن المعالج ومن ثم زيادة سرعة وكفاءة الأنظمة. ويجب دائماً الانتباه إلى مصادر نظام التشغيل المشتركة بين هذه البرامج.

أسئلة تقويم ذاتي



1 – وضح كيف نحظر دخول thread محددة إلى موقع ذاكرة معين طالما أن هناك thread أخرى تتعامل مع هذا الموقع.

5 .الشبكات Networking

1 .5 أهمية الشبكات

تعتبر شبكات الحاسوب ذات أهمية بالغة في تطبيقات الحاسوب، فهي الطريقة الأساسية لتشارك البيانات والمصادر والأجهزة المختلفة، وهي وسيلة الاتصال الرئيسية بين أجهزة النظام المختلفة. ويمكن أن نرى أثر الشبكات واضحا في مختلف أنواع الأنظمة. فالإنترنت هي مثال لأكبر وأشهر شبكة نتعامل معها من أي مكان. وبالشبكات أصبح من الممكن وجود آلة واحدة للطباعة يتشارك فيها عدد كبير من الأجهزة. وتطبيقات الشبكات لا حصر له، ففي مستشفى مثلا، يمكن للطبيب المتخصص أن يطالع تشخيص الطبيب العمومي لمتابعة حالة المريض، ويستطيع موظف الاستقبال أن يحدد مواعيد المقابلات حسب رغبة الطبيب، وأن يحدد ما إذا كان الدواء الذي يطلبه المريض متوفراً بالصيدلية أو لا، ويمكن للموردين الإطلاع على بيانات المخازن والمعامل والصيدلية لطلب المواد الضرورية في الوقت المناسب. تقوم أجهزة الحاسوب بتسهيل وتبسيط جميع هذه العمليات عن طريق الشبكات، ويمكن أن نتصور مقدار الجهد والوقت والمال الذي سيتم إنفاقه لإجراء هذه العمليات بدون وجود الشبكات. لكل هذه الأسباب، أولت أنظمة التشغيل ولغات البرمجة اهتماما كبيرا بالشبكات وتطبيقاتها.

لكي يتسنى تنفيذ برنامج عبر شبكة، يجب أن يتكون من برنامجين على الأقل: برنامج المخدم Server، وهو الذي يقدم الخدمة أو يمتلك المعلومة التي يرغب بها الزبون، وبرنامج الزبون Client، وهو الذي يطلب الخدمة التي يوفرها المخدم. توفر لغة جافا package لبرمجة تطبيقات الشبكات وهي java.net. وفي هذا القسم نقدم شرحاً وافياً لبرمجة كل من المخدم والزبون باستخدام برتوكولي TCP و UDP. ولكن قبل ذلك لابد لنا من التأكيد على معاني بعض المصطلحات المستخدمة في برمجة تطبيقات الشبكات.

Socket

تسمح بتخاطب الزبون والمخدم، وتبسط للمبرمج تفاصيل البرمجة الخاصة بالشبكة، فتجعل عملية التخاطب مع الشبكة أشبه بالتخاطب مع الملفات.

Port

هو رقم يحدده المخدم كي يستطيع الزبون معرفة مكان موقع برنامج المخدم بجهاز الحاسوب.

Datagram

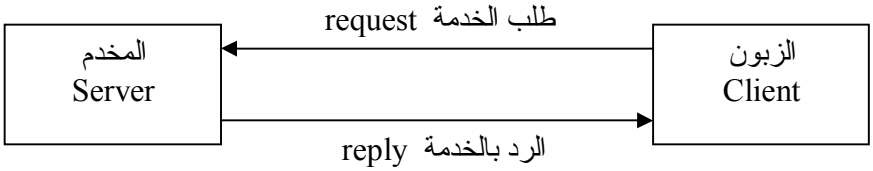
وحدات ذات أحجام معينة تحمل البيانات التي يرغب المرسل بإرسالها إضافة إلى عنوان المستقبل. يعتمد عدد الوحدات على حجم البيانات المرسلة.

TCP (Transmission Control Protocol)

من أشهر بروتوكولات تخاطب الشبكات، وفيها يتم فتح قناة اتصال بين الجهازين ثم إرسال جميع البيانات المرغوب إرسالها على شكل stream، ثم إغلاق قناة الاتصال. يتميز بالاعتمادية reliability.

UDP (User Datagram Protocol)

هو بروتوكول آخر للشبكات. وفيه يتم إرسال البيانات عن طريق datagrams مستقلة عن بعضها دون فتح قناة اتصال ثابتة، يتميز بالسرعة.



شكل 4-5-1: عملية الاتصال بين المخدم والزبون

أسئلة تقويم ذاتي



ما متطلبات تشغيل برنامج عبر شبكة ؟

ما معاني المصطلحات التالية :

Socket – Port – Datagram ؟

بماذا يتميز برتوكول TCP ؟

بماذا يتميز برتوكول UDP ؟

2.5 برمجـــة Server و Client باســـتخدام Stream Socket Connection

يستخدم هذا النوع بروتوكول TCP، حيث يتم فتح قناة اتصـــال بـــين برنـــامج المخـــدم وبرنامج الزبون، وقد يكون البرنامجان على نفس جهاز الحاســـوب أو علـــى جهـــازين مختلفين.

1.2.5 برنامج المخدم Server

```
import  java. io.*;
import java.net.*;
public  class  Server{
    public static void main(String args [ ] ){
        try{
                ServerSocket  serversocket = new ServerSocket(7000);
                Socket  socket = ServerSocket.accept( ) ;
                DataInputStream                fromClient        =        new
DataInputStream(socket.getInputStream( ));
        DataOutputStream        toClient        =        new        DataOutputStream
(socket.getOutputStream( ));
                while(true) {
                        double number = fromClient.readDouble( );
                        System.out.println("Client sent : " + number);
                        double  squareRoot = Math.sqrt(number);
```

```
                    toClient.writeDouble(squareRoot);
                    toClient.flush( );
                    System.out.println("The square root = " + squareRoot);
                }
            }
            catch(IOException e){
                e.printStackTrace( );
            }
        }
    }
```

برنامج 4-5-1

البرنامج أعلاه هو مثال لمخدم بسيط. يتم أولاً إنشاء object من class ServerSocket للإعلان عن وجود server وتحديد رقم الـ port الذي سيتخاطب البرنامج عبره مع الزبائن. بعدها ينتظر المخدم قدوم طلب خدمة من الزبون، وعند حدوث ذلك يتم إنشاء object من Socket، وهو الذي يحمل معلومات قناة الاتصال مع الزبون. ومن خلاله يمكن إنشاء data input stream لاستلام البيانات من الزبون باستخدام الدالة getInputStream. يختلف باقي البرنامج باختلاف الخدمة التي يقدمها المخدم للزبون. وفي هذا البرنامج يستقبل المخدم من الزبون عدداً double ويحسب الجزر التربيعي للعدد، ويعيد الجزر التربيعي إلى الزبون، في مثال لخدمة بسيطة يقدمها المخدم للزبائن. تستمر هذه العملية بواسطة while(true) إلى أن ينهي برنامج المخدم أو الزبون برنامجه باستخدام Ctrl+C. لاحظ أن هذا المخدم يستطيع خدمة زبون واحد فقط في المرة الواحدة، وسنعرف لاحقا كيف يمكن للمخدم التخاطب مع أكثر من زبون.

2.2.5 برنامج الزبون Client

البرنامج التالي هو برنامج الزبون، وفيه يجري الاتصال ببرنامج المخدم الذي من المفروض أن يتم تشغيله أولاً لكي يستجيب عند محاولة اتصال الزبون به.

```
import  java.io.*;
import  java.net.*;
public class  Client{
        public static void main(String args[]) {
            try{
                Socket  socket = new Socket("localhost", 7000);
                DataInputStream fromServer = new
        DataInputStream(socket.getInputStream( ));
                DataOutputStream toServer = new
        DataOutputStream(socket.getOutputStream( ));
                while(true) {
                        System.out.print("Enter  a  number: ");
                        double number = readDouble( );
                        toServer.flush( );
                        double squareRoot = fromServer.readDouble( );
                        System.out.println("Square Root = " + squareRoot);
                }
            }
            catch(IOException  e){
                        e.printStackTrace( );
            }
        }
        public static double  readDouble( ){
                DataInputStream in = new DataInputStream(System.in);
                try{
                        double d = Double.parseDouble(in.readLine( ));
                        return  d ;
                }
                catch(Exception  e){
                        System.out.println(e);
                        return 0;
                }
        }
}
```

برنامج 4-5-2

يبدأ برنامج الزبون بإنشاء socket للاتصال ببرنامج المخدم الموجود على الجهاز localhost، والتي تشير إلى نفس البرنامج الذي يحتوي على برنامج الزبون. وفي حال لم يكن في نفس الجهاز، يتم استبدال العبارة localhost برقم الجهاز الذي يحتوي على برنامج المخدم، مثلاً "160.251.44.2". يتم كذلك توضيح رقم الـ port المخصص للتخاطب بين البرنامجين، وهو في هذا المثال 7000. يتم إعداد data input stream و data output stream لاستلام وإرسال البيانات من وإلى المخدم بنفس الطريقة التي تم إنشاؤها في برنامج المخدم. يطلب برنامج الزبون من المستخدم إدخال عدد يتم قراءته من الشاشة باستخدام الدالة readDouble المعرفة داخل الـ class، والتي تستخدم System.in لقراءة سطر من شاشة الإدخال ثم تحوله إلى double. يتم إرسال العدد إلى برنامج المخدم والذي كما نعلم يقوم بحساب الجذر التربيعي للعدد، ومن ثم استقبال الجذر التربيعي وطباعته على شاشة برنامج الزبون.

يشرح هذا المثال المكونات الرئيسية لبرنامجي المخدم والزبون اللازمة لعملية الاتصال بين البرنامجين، وبعد معرفة الـ class والدوال الأساسية، يمكن تطبيق أي نظام جافا على شبكة وإضافة GUI components لتحسين مظهر البرنامج.

مثال لتنفيذ البرنامج:

Client screen:

```
Enter a number: 5
Square Root = 2.23606797749979
Enter a number: 9
Square Root = 3.0
Enter a number: 625
Square Root = 25.0
Enter a number:
```

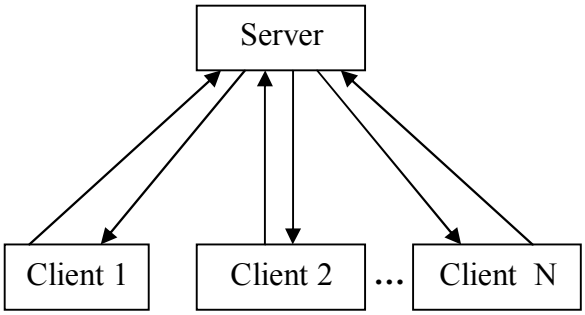
Server screen:

```
Client sent : 5.0
The square root =
2.23606797749979
Client sent : 9.0
The square root = 3.0
Client sent : 625.0
The square root = 25.0
```

مخرجات برنامج 4-5-2

3.2.5 التخاطب مع أكثر من زبون

فيما يلي تعديل على برنامج المخدم ليكون قادراً على التخاطب مع عدد من الزبائن في نفس الوقت، كما نتوقع من أغلب المخدمات، حيث يكون هناك عدد من الزبائن الذين يحاولون الاتصال بمخدم واحد لطلب الخدمة. يتم تعديل المخدم بحيث يقوم باستقبال طلب الخدمة من زبون محدد، ويقوم بإنشاء thread تتولى عملية التخاطب مع هذا الزبون حتى تكتمل عملية الاتصال، وبالتالي يستطيع المخدم التخاطب مع عدد كبير من الزبائن.



شكل 4-5-2: عملية الاتصال بين المخدم وعدد من الزبائن

```
import  java.io.*;
import  java.net.*;
public  class  Server{
        public static void main(String args[ ]){
                try{
                        int clientID = 1;
                        ServerSocket  serverSocket  =  new  ServerSocket(7000,
10);
                        while(true){
                                Socket socket = serverSocket.accept( );
                                System.out.println("Connection  established  For
client: " + clientID);
                                InetAddress  clientAddress  =
socket.getInetAddress( );
                                System.out.println("Client  IP  address:  "  +
clientAddress.getHostAddress( ));
```

```
                    ClientThread clientThread = new
ClientThread(socket, clientID);
                    clientThread.start( );
                    clientID+ + ;
                }
            }
            catch(IOException e){
                e.printStackTrace( );
            }
        }
}
class ClientThread extends Thread {
        Socket socket;
        int clientID;
        public ClientThread(Socket s, int id){
            socket = s;
            clientID = id;
        }
        public void run( ){
            try {
                DataInputStream fromClient = new
DataInputStream(socket.getInputStream( ) );
                DataOutputStream toClient = new
DataOutputStream(socket.getOutputStream( ) );
                while(true) {
                    double number = fromClient.readDouble( );
                    System.out.println("Client " + clientID + " sent: "
+ number);
                    double squareRoot = Math.sqrt(number);
                    toClient.writeDouble(squareRoot);
                    toClient.flush( );
                    System.out.println("The square root = " +
squareRoot);
                }
            }
            catch(IOException e) {
                e.printStackTrace( );
            }
        }
}
```

برنامج 4-5-3

عند إنشاء object من ServerSocket، يمكن أن نحدد العدد الأقصى للزبائن الذين يمكن للمخدم التخاطب معهم في اللحظة الواحدة، وهو في هذا البرنامج 10. ونختار هذا العدد بناء على نوع التطبيق وحجم المصادر التي سيقوم المخدم بحجزها لكل طالب خدمة، فالهدف هو ألاَّ يزدحم المخدم فوق طاقته وألاَّ تحدث مشاكل بسبب عدد الزبائن الكبير. يوفر class InetAddress معلومات عن الزبون مثل عنوان الجهاز في الشبكة IP Address عن طريق الدالة getHostAddress أو اسم الجهاز باستخدام الدالة getHostName. يتم تخصيص وتشغيل thread لكل طالب خدمة بعد مده بالـ socket التابع لهذا الزبون ورقمه الذي يتابع به المخدم الزبائن clientID. يتابع الـ thread عمل المخدم طوال فترة اتصال الزبون. لا يكون هناك أي تعديل على برنامج الزبون، ويتم التفاعل بين البرنامجين تماما كما في المثال السابق.

أسئلة تقويم ذاتي



كيف يتم إعداد data input stream و data output stream في كل من برنامجي المخدم والزبون ؟

3.5 برمجة Serves و Client باستخدام Connectionless Datagram Sockets

يستخدم هذا النوع بروتوكول UDP، وسمي بـ connectionless لعدم وجود قناة اتصال ثابتة بين المخدم والزبون كما في TCP، بل يتم إرسال البيانات على شكل وحدة datagram ذات حجم معين. ولذلك فإن كل وحدة يجب أن تحتوي على عنوان الجهاز الذي سيتم إرسالها إليه. فيما يلي برنامج لمخدم بسيط وزبون وشرح لعملية إرسال واستقبال الوحدات بينهما.

```
// Server that receives and sends packets from/to a client.
import java.io.*;
```

```
import java.net.*;

public class Server {
        private DatagramSocket socket;
        public Server()
        {
      try {
                socket = new DatagramSocket( 5000 );
    }
     catch( SocketException socketException ) {
                socketException.printStackTrace();
                System.exit( 1 );
    }
    }
    private void waitForPackets()
     {
      while ( true ) {
                try {
                        byte data[] = new byte[ 100 ];
                        DatagramPacket receivePacket = new
DatagramPacket( data, data.length );
                        socket.receive( receivePacket );
                        System.out.println( "\nPacket received:" +
                "\nFrom host: " + receivePacket.getAddress() +
                "\nHost port: " + receivePacket.getPort() +
                "\nLength: " + receivePacket.getLength() +
                "\nContaining:\n\t" + new String(
receivePacket.getData(),
                0, receivePacket.getLength() ) );


                        sendPacketToClient( receivePacket );
                    }
                    catch( IOException ioException ) {
                        System.out.println( ioException.toString() + "\n"
);
                        ioException.printStackTrace();
                    }
          }
        }
```

```
        private void sendPacketToClient( DatagramPacket receivePacket )
throws IOException
        {
                System.out.println( "\n\nEcho data to client..." );
        DatagramPacket sendPacket = new DatagramPacket(
                receivePacket.getData(), receivePacket.getLength(),
                receivePacket.getAddress(), receivePacket.getPort() );
                socket.send( sendPacket ); // send packet
        System.out.println( "Packet sent\n" );
        }


        public static void main(String args[] )
        {
                Server application = new Server();
                application.waitForPackets();
        }
 }
```

برنامج 4-5-4

يستخدم DatagramSocket class لتحديد port للتخاطب بين المخدم والزبون. Datagrampacket هو class يكون الـ object منه عبارة عن الوحدة datagram المطلوب إرسالها عبر الشبكة. ولإرسال datagram، يجب أن تحتوي بداخلها على بعض المعلومات وهي:

1- عنوان الجهاز الذي نرغب في إرسال الوحدة إليه.

2- رقم الـ port المخصص للتخاطب.

3- البيانات المطلوب إرسالها.

4- طول البيانات المرسلة.

عند إرسال وحدة معينة إلى الشبكة، يتم تضمين هذه المعلومات بداخل الوحدة عن طريق الـ constructor الخاص بـ DatagramPacket عند إنشاء الوحدة. وعند استقبال وحدة من الشبكة، يتم استخلاص المعلومات التالية منها:

1- عنوان الجهاز الذي تم إرسالها منه.

2- رقم الـــــــ port المخصص للتخاطب.

3- البيانات التي أرسلها المرسل.

4- طول البيانات الواردة.

فيما يلي برنامج الزبون client الذي يتخاطب مع البرنامج السابق باستخدامdatagrams .

```
import java.io.*;
import java.net.*;
import javax.swing.*;
public class Client {
      private DatagramSocket socket;
      private BufferedReader br;
      public Client3()
      {
                  try {
                              socket = new DatagramSocket();
                              br = new BufferedReader(new
InputStreamReader(System.in));
            }
            catch( SocketException socketException ) {
                              socketException.printStackTrace();
                              System.exit( 1 );
            }
            }
            private void waitForPackets()
            {
            while ( true ) {
                              try{
                                          String message = br.readLine();
                                          byte data[] = message.getBytes();
                                          DatagramPacket sendPacket = new
DatagramPacket( data,
                                          data.length, InetAddress.getLocalHost(), 5000 );
                                          socket.send( sendPacket ); // send packet
                                          System.out.println( "Packet sent\n" );
                              }
                              catch ( IOException ioException ) {
                                          System.out.println( ioException.toString() + "\n"
);
```

```
                    ioException.printStackTrace();
                }
                try {
                    byte data[] = new byte[ 100 ];
                    DatagramPacket receivePacket = new
DatagramPacket(data, data.length );
                    socket.receive( receivePacket ); // wait for packet
                    System.out.println( "\nPacket received:" +
                "\nFrom host: " + receivePacket.getAddress() +
                "\nHost port: " + receivePacket.getPort() +
                "\nLength: " + receivePacket.getLength() +
                "\nContaining:\n\t" + new String(
receivePacket.getData(),
                0, receivePacket.getLength() ) );
                }
                catch( IOException exception ) {
                    System.out.println( exception.toString() + "\n" );
                    exception.printStackTrace();
                }
            }
        }

        public static void main( String args[] )
        {
            Client application = new Client();
            application.waitForPackets();
        }

}
```

برنامج 4-5-5

تستخدم BufferedReader لقراءة بيانات الزبون من الشاشة. تقوم الدالة getLocalHost التابعة لـــــ class InetAddress بتوفير عنوان جهاز المخدم المطلوب إرسال البيانـــات إليه، وهو في هذا المثال نفس الجهاز الذي يوجد به برنامج الزبـــون. localhost تعبـــر

عن "هذا الجهاز"، وإذا كان برنامج المخدم موجوداً في جهاز آخر – وهو الحال في أغلب البرامج – يجب تعديل العنوان إلى عنوان الجهاز الذي يوجد به برنامج المخدم.

مثال لتنفيذ البرنامج:

Client screen:

```
Packet received:
From host: /127.0.0.1
Host port: 1049
Length: 25
Containing:
This is a datagram packet
```

Server screen:

```
This is a datagram packet
Packet sent
Packet received:
From host: /127.0.0.1
Host port: 5000
Length: 25
Containing:
This is a datagram packet
```

مخرجات برنامج 4-5-5

قدَّم هذا القسم شرحاً لعملية الاتصال بين برامج جافا عبر شبكات الحاسوب. وتوفر جافا المزيد من وسائل الاتصال المختلفة باختلاف التطبيق مثل الاتصال بمخدمات الإنترنت وتحميل الملفات وعرض الصفحات باستخدام الـــapplets. كذلك من طرق الاتصال نداء الدوال الموجودة في أجهزة وبرامج بعيدة باستخدام تقنية Remote Method Invocation RMI، أو التعامل مع برامج من لغات أخرى باستخدام Common Object Request Broker Architecture CORBA، أو الاتصال بأجهزة ليست بالضرورة أجهزة حاسوب باستخدام Jini. لكل هذه التقنيات الـــ packages و الـــ classes الخاصة بها، ولا يتسع المقام هنا لذكرها، ولكنها متوفرة في معظم المراجع ومواقع الإنترنت الخاصة بلغة جافا.

أسئلة تقويم ذاتي



لماذا يسمى الـ Connectionless Datagram Socket بـــ connectionless ؟

لإرسال datagram، يجب أن تحتوي بداخلها على بعض المعلومات مـــاهي هذه المعلومات ؟

عند استقبال وحدة datagram من الشبكة، ما هـــي المعلومـــات التـــي يـــتم استخلاصها منها ؟

الخلاصة

تناولت الوحدة البريمجات applets. و الـــــ applet هي برنـــامج جافـــا مكتـــوب بصورة معينة تجعله قابلاً للتضمين داخل برنامج HTML (Hyper Text Markup Language) وهي احد اشهر وأسهل اللغات المستخدمة لبرمجة صفحات الانترنت.

وأوضحت أن الفرق بين البريمجات والبرامج التقليدية يأتي في طريقة التنفيذ والصـــفات الإضافية التي تتمتع بها لكونها معدة للتحميل والتشغيل على صفحات الانترنت ، ولكـــي يطلق على برنامج معين applet، يجـــب أن يـــتم تعريـــف class يـــرث خصـــائص JApplet، وهو class موجود فـــي javax.swing package.هـــذا الـــــــ class يحتوي على أربع دوال ذات وظائف معينة يمكن إعادة تعريفها اختياريا بداخل البرنامج. والدوال هي :

init( ) – start( ) – paint( ) – stop( ) – destroy( ) ، وقد أوضـــحت الوحـــدة استخدامات هذه الدوال .

من خلال برنامج 4-1-2 ، ذكرت الوحدة أن المعلومات التي نحتاجهـــا عـــن الـــــــ applet ليتم كتابتها في برنامج HTML هو اســـم الملـــف بالامتـــداد class- طـــول وعرض النافذة التي نرغب في تخصيصها للـــ applet على الصفحة. يحفظ البرنـــامج في ملف بالامتداد html أو htm، ويتم فتحه بواسطة متصفح انترنت ، كما ذكـــرت أن الـــ applet توفر طريقة لتبادل الوسائط parameters بينها وبـــين برنـــامج HTML حيث تقوم عبارات HTML معنية بتحديد اسم وقيمة كل وسيط ، وتحصل الـــ applet على قيم هذه الوسائط باستخدام الدالة( )getParameter .

كما لخصت الوحدة مميزات الأمن والسرية التي توفرها الـــــــ applets فـــي النقـــاط التالية:

- لا يسمح للـــ applet بالدخول إلى نظام ملفات الجهاز التي تم تحميلها عليه لا قراءة ولا كتابة.

- لا يسمح للـ applet بتشغيل برنامج أخر على جهاز المستخدم.
- لا يسمح للـ applet بتوصيل جهاز المستخدم مع أي جهاز آخر.

يمكن أن نحوّل أي application إلى applet مادام انه يخضع لقوانين امن الـ applets ، أما الـ applets فيمكن أن تتحول إلى application بدون شروط

قدمت الوحدة الاستثناءات ومعالجة الاستثناءات حيث تناولت أولاً الاستثناءات وهي تنقسم من حيث أسباب حدوثها إلى :

- استثناءات لأسباب خارجية - استثناءات لأسباب تتعلق بكتابة البرنامج - استثناءات تتعلق بمستخدم البرنامج ، ثم تناولت الوحدة معالجة هذه الاستثناءات.

وبينت أن لغة جافا صممت عدد من الـ classes التي تعبر عن هذه الأخطاء، ووضعت الطرق الملائمة لمعالجة والتعامل مع هذه الأخطاء عند حدوثها ، وتحتوي java.lang package على class اسمه Exception، يعبر هذا الـ class عن خطأ من أي نوع يحدث أثناء تنفيذ البرنامج. هناك عدد كبير من الـ classes تعرفه لغة جافا لتمثيل الأخطاء المختلفة، وجميعها ترث خصائص الـ Exception class.

تناولت الوحدة أيضا الملفات وأوضحنا أنه يمكن التعامل مع الملفات بأنواعها باستخدام لغة جافا، حيث يمكن قراءة وكتابة أنواع البيانات المختلفة بما في ذلك الكائنات، وذكرنا انه يمكن تعريف ملف بلغة جافا باستخدام الفئة file استعداداً لاستخدامه في البرنامج، وذلك بتحديد اسم الملف عند إنشاء الكائن . وقدمنا مثال لإنشاء الملف in.dat بالمسار C:\java في نظام التشغيل windows:

```
File file = new File("C:\\java\\in.dat");
```

ومثال لإنشاء ملف باسم in.dat في المسار /java/ في نظام التشغيل Unix:

```
File file = new File("java/in.dat");
```

وذكرنا أنه يمكن معرفة ما إذا كان الملف in.dat موجود مسبقاً أو يتم انشاؤه لأول مرة باستخدام الدالة exists، كما يمكن الحصول على مسار الملف الكامل باستخدام الدالة

getPath . كما بينت الوحدة أنه لقراءة بيانات محفوظة في ملف معين، يتم تعريف كائن من نوع FileInputStream عن طريق تحديد اسم الملف الذي يحتوي على البيانات. وعند تعريف الملف، يتم التأكد من وجود ملف بالاسم المحدد، وإذا لم يوجد، يتولد exception من نوع FileNotFoundException، ولكتابة البيانات من البرنامج وحفظها في ملف، يتم استخدام كائن من نوع FileOutputStream مع تحديد اسم الملف الذي نرغب بحفظ البيانات فيه. إذا لم يكن هذا الملف موجود مسبقاً، يتم إنشاؤه وحفظه في المجلد المحدد، وإذا لم يتم تحديد هذا المجلد، يتم حفظ الملف في المجلد الموجود فيه البرنامج.، كما ذكرنا أن الدالة read تستخدم لقراءة byte واحد من الملف المفتوح للقراءة، والدالة write لكتابة byte واحد في الملف المفتوح للكتابة.

تناولت الوحدة أهمية البرمجة المتعددة ، وعرفت الـ multi-processing أو المعالجة المتعددة وهي مقدرة جهاز الحاسوب على تنفيذ برامج متعددة آنياً ، وان الذي يحدد توفر هذه الخاصية بجهاز الحاسوب أو انعدامها هو المعالج processor ذلك باعتباره هو المسئول عن تنفيذ أوامر البرامج المختلفة، وهو ينفذ تعليمة واحدة في اللحظة الواحدة ، وشرحت الوحدة كيف أن المعالج لا يقوم بتنفيذ عدد من البرامج في اللحظة الواحدة، وإنما يقوم بتقسيم الوقت بين البرامج التي نرغب في تنفيذها فيما يعرف بـ time sharing . كما تناولت دورة حياة الـ thread ونلخص حالاتها في :

(1) Born (2) Ready (3) Running (4) Blocked (5) Sleeping (6) Waiting (7) Dead

أوضحت الوحدة أن للبرمجة المتعددة ثلاث حالات هي :-

- إن تكون البرامج مستقلة ويتم تنفيذها بالكامل في نفس الوقت - إن تكون البرامج مرتبطة أو معتمدة على بعضها البعض. أي أن تكون هناك قيود على ترتيب تنفيذها .- أن تكون البرامج عبارة عن نسخ متعددة من نفس البرنامج .

كذلك تناولت أولوية تنفيذ الـ Threads وذكرت ان الـ Threads يمكن ان تختلف من حيث أولوية التنفيذ ..... وان أولوية الـ Threads يتم تحديدها بواسطة الدالة set Priority ، وذكرت أن الأولوية 1 هي اقل أولوية للـ thread في لغة جافا، وأعلى أولوية ممكنة هي 0 . وإذا لم يتم تحديد أولوية معينة للـ thread، تعطى أولوية عادية 5.

تناولت الوحدة الـ Runnable وهو interface يمكن للبرنامج أن يقوم بتعريفه فيصبح thread مع اتاحة الفرصة للـ thread أن يرث خصائص class آخر. قدمت الوحدة مثال يوضح عملية التشارك في المصادر حيث نجد في هذا المثال برنامجين يتعاملان مع نفس موقع الذاكرة حيث تظهر أخطاء البيانات بسبب استخدام أكثر من برنامج لهذه البيانات في نفس الوقت .... ثم تعرفنا على كيفية منع تعامل أكثر من برنامج أو thread واحد مع نفس مواقع الذاكرة وذلك باستخدام الكلمة المحجوزة synchronized في ترويسة الدالة لكي تمنع تنفيذ أكثر من thread واحدة للدالة.

أوضحت الوحدة أهمية الشبكات البالغة في تطبيقات الحاسوب باعتبارها الطريقة الأساسية لتشارك البيانات والمصادر والأجهزة المختلفة وهي الوسيلة الرئيسية بين أجهزة النظام المختلفة .

كما تناولت معاني بعض المصطلحات المستخدمة في برمجة تطبيقات الشبكات نذكر منها:

Socket : تسمح بتخاطب الزبون والمخدم، وتبسط للمبرمج تفاصيل البرمجة الخاصة بالشبكة، فتجعل عملية التخاطب مع الشبكة أشبه بالتخاطب مع الملفات.

Port : هو رقم يحدده المخدم كي يستطيع الزبون معرفة مكان موقع برنامج المخدم بجهاز الحاسوب.

Datagram : وحدات ذات أحجام معينة تحمل البيانات التي يرغب المرسل بإرسالها إضافة إلى عنوان المستقبل. يعتمد عدد الوحدات على حجم البيانات المرسلة. كما عرفنا بروتوكول TCPوبروتوكولUDP .

قدمت الوحدة برمجة Server و Client باستخدام Stream Socket Connection وذكرت أن هذا النوع يستخدم بروتوكول TCP، وقدمت برنامج مثال لمخدم بسيط ويستطيع خدمة زبون واحد فقط في المرة الواحدة ،في هذا البرنامج يتم أولا إنشاء object من class ServerSocket للإعلان عن وجود server وتحديد رقم الـ port الذي سيتخاطب البرنامج عبره مع الزبائن. بعدها ينتظر المخدم قدوم طلب خدمة من الزبون، وعند حدوث ذلك يتم إنشاء object من Socket، وهو الذي يحمل معلومات قناة الاتصال مع الزبون. ومن خلاله يمكن إنشاء data input stream لاستلام البيانات من الزبون باستخدام الدالة getInputStream . كما قدمت الوحدة مثال برنامج الزبون ويبدأ برنامج الزبون بإنشاء socket للاتصال ببرنامج المخدم الموجود على الجهاز localhost، والتي تشير إلى نفس البرنامج الذي يحتوي على برنامج الزبون. وفي حال لم يكن في نفس الجهاز، يتم استبدال العبارة localhost برقم الجهاز الذي يحتوي على برنامج المخدم ، يتم كذلك توضيح رقم الـ port المخصص للتخاطب بين البرنامجين ،و يتم إعداد data input stream و data output stream لاستلام وإرسال البيانات من وإلى المخدم بنفس الطريقة التي تم إنشاؤها في برنامج المخدم ، كما أوضحنا في هذا القسم كيف يمكن للمخدم التخاطب مع أكثر من زبون .

مسرد المصطلحات

Socket

تسمح بتخاطب الزبون والمخدم، وتبسط للمبرمج تفاصيل البرمجة الخاصة بالشبكة، فتجعل عملية التخاطب مع الشبكة أشبه بالتخاطب مع الملفات.

Port

هو رقم يحدده المخدم كي يستطيع الزبون معرفة مكان موقع برنامج المخدم بجهاز الحاسوب.

Datagram

وحدات ذات أحجام معينة تحمل البيانات التي يرغب المرسل بإرسالها إضافة إلى عنوان المستقبل. يعتمد عدد الوحدات على حجم البيانات المرسلة.

TCP (Transmission Control Protocol)

من أشهر بروتوكولات تخاطب الشبكات، وفيها يتم فتح قناة اتصال بين الجهازين ثم إرسال جميع البيانات المرغوب إرسالها على شكل stream، ثم إغلاق قناة الاتصال. يتميز بالاعتمادية reliability.

UDP (User Datagram Protocol)

هو بروتوكول آخر للشبكات. وفيه يتم إرسال البيانات عن طريق datagrams مستقلة عن بعضها دون فتح قناة اتصال ثابتة، ويتميز بالسرعة.

جامع النفايات garbage collector

برنامج لتحرير خانات الذاكرة غير المستغلة بواسطة البرامج.

Threads

البرامج التي يتم تنفيذها على التوازي مع برامج أخرى.

المعالج processor

هو المسئول عن تنفيذ أوامر البرامج المختلفة، وهو ينفذ تعليمة واحدة في اللحظة الواحدة.

millisecond

الملي ثانية ويمثل $\frac{1}{1000}$ من الثانية .

المراجع

1. Y. D. Liang
   Introduction to Java Programming, Third Edition
   ISBN: 0-13-031997-X
   Prentice Hall 2001
2. Deitel and Deitel
   Java How to Program, Third Edition
   ISBN: 0-13-012507-5
   Deitel 2000
3. Deitel and Deitel
   Java How to Program, Fifth Edition
   ISBN: 0-13-101621-0
   Deitel 2003
4. Stephan J. Chapman
   Introduction to Java
   ISBN: 0-13-919416-9
   Prentice Hall 1999
5. نعمة عزالدين محمد عثمان

   البرمجة بلغة جافا للمبتدئين

   الإصدارة الثانية 2007م

الملحق الأول

كلمات جافا المحجوزة

الكلمات المحجوزة هي كلمات عرّفتها اللغة وحجزتها كي تستخدم في البـــرامج فقط لما عرّفتها له. كل كلمة لها معنى محدد تميزه اللغة بمجرد ظهور هذه الكلمة فـــي البرنامج. ولا يسمح باستخدام هذه الكلمات كأسماء متغيرات أو لأغراض غير المعنـــى الذي تحمله.

تحتوي لغة **جافا** على الكثير من الكلمات المحجوزة. ولـــيس مـــن الضـــروري الإلمام بجميع هذه المعاني، لأن كثيراً منها يختص بمواضيع متقدمة في لغة **جافا**، علـــى الرغم من ذلك، لا بأس من أن نلقي عليها نظرة عابرة، حتى لا نخطئ في اختيار أحدها كاسم لمتغير.

كلمات جافا المحجوزة هي:

abstract	**double**	**int**	**Super**
boolean	**else**	**interface**	**switch**
break	**extends**	**long**	**synchronized**
byte	**false**	**native**	**this**
byvalue	**final**	**new**	**threadsafe**
case	**finally**	**null**	**throw**
catch	**float**	**package**	**transient**
char	**for**	**private**	**true**
class	**goto**	**protected**	**try**
const	**if**	**public**	**void**
continue	**implements**	**return**	**while**
default	**import**	**short**	
do	**instanceof**	**static**	

الملحق الثاني

الشفرة الموحدة Unicode

تتعامل أجهزة الحاسوب مع الأرقام فقط، ولذلك لتخزين الحروف، تقوم بإعطاء كل حرف رقماً معينا فريداً لكل واحدٍ منها، ويتم تخزين هذا العدد. وقبل اختراع الشفرة الموحدة، كان هناك مئات الأنظمة للتشفير الخاصة بتحديد هذه الأرقام للحروف، ولم يوجد من بينها نظام تشفير يحتوي على جميع الحروف الشاملة لكل اللغات. فعلى سبيل المثال، فإن الاتحاد الأوروبي وحده احتوى العديد من الشفرات المختلفة ليغطي جميع اللغات المستخدمة في الاتحاد. وحتى لو اعتبرنا لغة واحدة، كاللغة الإنجليزية، لم يكن هناك نظام تشفير يشمل جميع الحروف وعلامات الترقيم والرموز الفنية والعلمية الشائعة الاستخدام باللغة الإنجليزية.

الشفرة الموحدة Unicode هي عبارة عن جدول ضخم يخصص رقماً فريداً لكل حرف في جميع اللغات العالمية، وذلك بغض النظر عن نوع الحاسوب أو البرامج المستخدمة. وقد تم تبني مواصفات الشفرة العالمية من قبل قادة صناع أنظمة الحاسوب في العالم، لتتلاءم معها جميعاً. وتشمل الشفرة جميع حروف اللغات العالمية واللهجات المحلية والرموز العلمية في جميع هذه اللغات.

مدى حروف بعض اللغات في الشفرة الموحدة Unicode:

- الحروف العربية: 1575 – 1610

  هذا يعني أن الحرف أ يقابل العدد 1575، والحرف ي يقابل العدد 1610

- الحروف الإنجليزية الكبيرة capital: 65 – 90

  هذا يعني أن الحرف A يقابل العدد 65، والحرف Z يقابل العدد 90

- الحروف الإنجليزية الصغيرة small: 97 – 122

  هذا يعني أن الحرف a يقابل العدد 97، والحرف z يقابل العدد 122