

بسم الله الرحمن الرحيم

اساسيات البرمجة بلغة الجافا

الجزء الثاني

محمد محمود ابراهيم

جامعة الزعيم الازهري

كلية علوم الحاسوب وتقانة المعلومات

2011

اهداء : –

الي طلاب كلية علوم الحاسوب وتقانة المعلومات جامعة الزعيم الازهري الي
الدفعة 16 الي كل من ساهم في وصول هذا العمل الي هذا الشكل وارجوا ان ينال
رضاءكم .

البرمجة بالكائنات :-

تعتبر لغة جافا من أشهر لغات 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.

الصف 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 الصف مرة واحدة ثم اشتقاق أي عدد من الكائنات من هذا الصف. 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 لصف وكيفية استخدامه لاشتقاق عدد من الكائنات والتعامل معها.

مثال : -

```

1  class student
2  {
3      public String name;
4      public void printname ()
5      {
6          System.out.println(name) ;
7      }
8  }
9  class classtest
10 {
11     public static void main(String args[])
12     {
13         student std = new student ();
14         std.name = "Mohammed Mahmoud";
15         std.printname ();
16     }
17 }

```

الخرج من البرنامج

```

C:\WINDOWS\system32\cmd.exe
Mohammed Mahmoud
Press any key to continue . . .

```

يعرف هذا البرنامج الفئة student والتي تحتوى على عضو بياني واحد من النوع

String ودالة لطباعة هذا الاسم

في الدالة main في السطر رقم 13 قمنا بانشاء كائن من الفئة student يحمل الاسم std

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، في السطر رقم 14 وضعنا قيمة في المتغير name وقمنا بتفيذ الدالة printName للكائن std .

محددات الوصول :-

- Public عندما يتم الاعلان عن محدد الوصول عام فان هذا العضو يمكن الوصول اليه من جميع الفئات الاخرى
- Private عندما يعن عن محدد الوصول خاص فان هذا العضو يستخدم داخل الفئة فقط ولاستطيع الفئات الاخرى استخدامه
- Protected عندما يكون محدد الوصول محمي فان هذا العضو يستخدم داخل الفئة والفئات المشتقة فقط

المشيدات Constructor :-

1. هي دالة تحمل نفس اسم الفئة
2. يتم تنفيذها تلقائيا عند انشاء كائن من الفئة
3. تستخدم هذه الدالة لإجراء العمليات التي نرغب في تنفيذها ابتداءً لحظة إنشاء الكائن وقبل تعامل أي جهة مع هذا الكائن
4. يمكن للمشيدات ان تحمل وسائط لكنها لا ترجع اي قيمة حتى void .

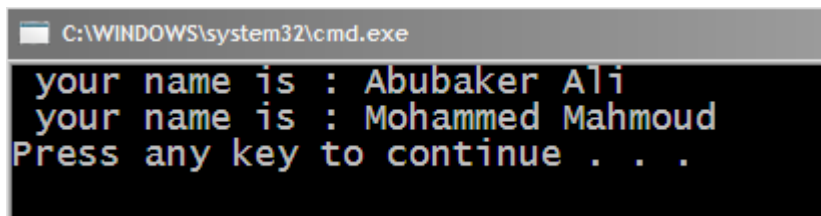
الصيغة العامة لتعريف مشيد

```
access classname(parameters)
{
    statement ;
}
```

مثال : -

```
1 class student
2 {
3     public String name;
4     public student()
5     {
6         name = "Abubaker Ali";
7     }
8     public void printname()
9     {
10        System.out.println(" your name is : " + name);
11    }
12 }
13 class Constuctor
14 {
15     public static void main(String args[])
16     {
17         student std = new student();
18         std.printname();
19         std.name = "Mohammed Mahmoud";
20         std.printname();
21     }
22 }
23
```

الخرج من البرنامج



```
C:\WINDOWS\system32\cmd.exe
your name is : Abubaker Ali
your name is : Mohammed Mahmoud
Press any key to continue . . .
```

السطور من 1 الى 12 تم تعريف الفئة student والذي يحتوى على الخاصية name و الدالة printname() والمشيء student() والذي يتم فيه اعطاء قيمه اولية للخاصية name .
في السطر 17 تم انشاء كائن من الفئة student بالاسم std .
ويمكن ان تحتوى الفئة على اكثر من مشيء تختلف في عدد ونوع بيانات الوسائط وفي هذه الحالة يعرف بالتحميل الزائد للمشيءات constructor overloading .

المؤشر this :-

يستطيع كل class أن يشير إلى محتوياته من متغيرات ودوال باستخدام المؤشر this
والمثال التالي يوضح كيف استخدام المؤشر this

```
1 public class Circle
2 {
3     int radius ;
4     public Circle()
5     {
6         radius = 1;
7     }
8     public Circle(int radius)
9     {
10        this.radius = radius;
11    }
12    public double getArea()
13    {
14        return 3.14 * radius * radius ;
15    }
16    public static void main(String args[]) // Main of program
17    {
18        Circle c = new Circle();
19        System.out.println(" Area is : " + c.getArea());
20    }
21 }
```

وهذا الخرج من البرنامج :-

```
C:\WINDOWS\system32\cmd.exe
Area is : 3.14
Press any key to continue . . .
```

في السطر رقم 10 العبارة `this.radius` تشير الي العضو البياني `radius` في الفئة `Circle` وليس الوسيط `radius` الممرر للمشيء `radius`.

الوراثة : -

تعتبر الوراثة واحدة من أهم المميزات التي توفرها لغات البرمجة بالكائنات، وتسمح بالاستفادة من خصائص ودوال الفئات مسبقا لتعريف فئات جديدة، بحيث لا يضطر المبرمج إلى إعادة كتابة تلك الخصائص مرة ثانية، ولعمل علاقات جديدة تربط بين الكائنات. عندما ترث فئة معينة خصائص فئة أخرى تسمى الفئة الوارثة بالابن `subclass` وتسمى الفئة الموروثة بالأب `superclass`. 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،

الشكل العام لعملية الوراثة :


```

class superclass
{
    attributes ;
    .
    .
    method ;
    .
    .
}
class subclass extends superclass
{
    attributes ;
    .
    .
    method ;
    .
    .
}

```

الشكل اعلاه يوضح عملية الوراثة

محددات الوصول في 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.

مثال : -

```
1 import java.util.*;
2 class person
3 {
4     protected String name ;
5     protected int age ;
6     public void set()
7     {
8         Scanner sc = new Scanner(System.in);
9         System.out.print(" Enter Name : ");
10        name = sc.next();
11        System.out.print(" Enter age : ");
12        age = sc.nextInt();
13    }
14 }
15 class student extends person
16 {
17     double degree = 0.0;
18     public void display()
19     {
20         System.out.println(" Name " + "\t age" + "\t degree");
21         System.out.println(name + "\t" + age + "\t" + degree);
22     }
23 }
24 public class Inher
25 {
26     public static void main(String args[])
27     {
28         student std = new student();
29         std.set();
30         std.degree = 97;
31         std.display();
32     }
33 }
```

الخروج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Enter Name : kojo
Enter age : 20
Name      age      degree
kojo     20      97.0
Press any key to continue . . .
```

في المثال السابق الفئة student ترث الفئة person ونلاحظ على الرغم من ان الدالة set() معرفة في الفئة person الا اننا استطعنا استخدامها بواسطة الكائن std التابع للفئة student وذلك لان الفئة student ورثت متغيرات ودوال الفئة person .

التجريد Abstraction :-

الفئة مجردة هي فئة اب super class لا يمكن انشاء كائن من الفئة المجردة لان الفئة المجردة تحتوي على الاقل على دالة واحدة غير مكتملة incomplete . ولعمل فئة مجردة نستخدم الكلمة المحجوزة abstract تحتوي على الاقل على دالة واحدة مجردة abstract ويتم عمل تحميل override عليها . وتعني override امكانية تعديل الدالة في الفئات التي ترث الفئة المجردة .

المثال التالي يوضح مفهوم التجريد :-

```

1  abstract class B // abstract class
2  {
3      protected int num ;
4      public abstract void show(); // abstract function
5  }
6  class A extends B
7  {
8      public void show()
9      {
10         num = 10;
11         System.out.println(" Number is : " + num);
12     }
13 }
14 class C extends B
15 {
16     public void show()
17     {
18         num = 12;
19         System.out.println(" Numser is : " + num);
20     }
21 }
22 public class Abstract
23 {
24     public static void main(String args[])
25     {
26         A a = new A();
27         a.show();
28         C b = new C();
29         b.show();
30     }
31 }

```

الخروج من البرنامج :-

```

C:\WINDOWS\system32\cmd.exe
Number is : 10
Numser is : 12
Press any key to continue . . .

```

الفئات والدوال الثابتة :

الدوال الثابتة final method لا يمكن عمل override عليها اي دالة معرفة static هي ثابتة واي دالة معرفة private هي ايضا ثابتة .

الفئة الثابتة final class لا يمكن ان تكون فئة اب superclass واي دالة معرفة داخل الفئة الثابتة تكون ثابتة .

الواجهات :-

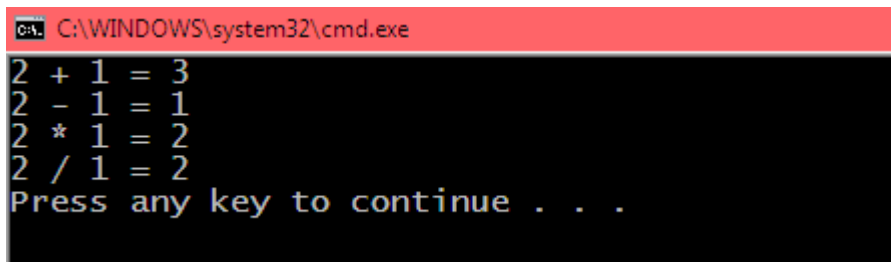
تعتبر لغة جافا من اللغات وحيده الوراثة single inheritance languages ، وتعني انه يمكن للفئة أن ترث خصائص فئة واحدة على الأكثر. من مميزات هذا النوع سهولة إدارة الكائنات وتحديد علاقتها وصلاحيتها، وله عيب هو عدم التمكن من وراثة خصائص معرفة في عدد من الفئات. تدعم بعض اللغات الوراثة المتعددة multiple inheritance ، وفيها يمكن للفئة أن ترث خصائص أكثر من فئة واحدة. وميزة هذا النوع من الوراثة هو الاستفادة من خصائص عدد أكبر من الفئات، وعيبيها صعوبة إدارة الكائنات وتحديد صلاحياتها. وضعت لغة جافا حلا لتلافي عيب الوراثة الوحيدة باستخدام الواجهات Interface ، وهو هيكل يشبه في تركيبه الفئة، إلا أن جميع دواله

خالية من التعليمات وجميع بياناته ثوابت. يمكن للفئة أن تستفيد من الخصائص المعرفة بال interfaces ولكن لا يطلق عليها وراثه بل تسمى تعريفاً implementation .

المثال التالي يوضح تعريف واجهة وكيفية استخدام الفئة لهذه الواجهة :

```
1 interface operations // define interface operations
2 {
3     public int sum(int x , int y);
4     public int sub(int x , int y);
5     public int mult(int x , int y);
6     public int div(int x , int y);
7 } // end of interface
8 public class Interface implements operations // define class Interface use interface operation
9 {
10     public int sum(int x , int y)
11     {
12         return (x + y);
13     }
14     public int sub(int x , int y)
15     {
16         return (x - y);
17     }
18     public int mult(int x , int y)
19     {
20         return (x * y);
21     }
22     public int div(int x , int y)
23     {
24         return (x / y);
25     }
26     public static void main(String args[]) // main program
27     {
28         Interface i = new Interface();
29         int a = 2 ;
30         int b = 1;
31         System.out.println(a + " + " + b + " = " + i.sum(a , b));
32         System.out.println(a + " - " + b + " = " + i.sub(a , b));
33         System.out.println(a + " * " + b + " = " + i.mult(a , b));
34         System.out.println(a + " / " + b + " = " + i.div(a , b));
35     }
36 }
```

الخرج من البرنامج :-



```
C:\WINDOWS\system32\cmd.exe
2 + 1 = 3
2 - 1 = 1
2 * 1 = 2
2 / 1 = 2
Press any key to continue . . .
```

في هذا البرنامج تعريف للواجهة Operations والتي تقوم بتعريفها الفئة Interface باستخدام الكلمة المحجوزة implements . 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 لا بد 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 والصيغة التالية يوضح ذلك :

```
public class Myclass implements interfaced1 , interfaced2
{
    :
    :
    :
}
```

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. والمثال التالي يوضح ذلك :-

```
1 import java.util.*;
2 import java.io.*;
3 interface Student
4 {
5     public void displayInfo();
6 }
7 interface Teacher
8 {
9     public void displayInfo();
10    public void setInfo();
11 }
12 public class StudentData implements Student , Teacher
13 {
14     private String name ;
15     private int marks[];
16     public StudentData(String name)
17     {
18         this.name = name ;
19         marks = new int[3];
20     }
21     public void setInfo()
22     {
23         Scanner in = new Scanner(System.in);
24         for(int i = 0;i < 3;i++)
25         {
26             System.out.print(" Enter Subject [" + (i + 1) + "] mark : ");
27             marks[i] = in.nextInt();
28         }
29     }
30     public void displayInfo()
31     {
32         System.out.println(name);
33         for(int i = 0;i < 3;i++)
34         {
35             System.out.println( "\t Subject ["+ (i + 1) + "] \t " + marks[i]);
36         }
37     }
38     public static void main(String[] args) throws IOException
39     {
40         StudentData std = new StudentData("Mohammed");
41         Student stu = std;
42         Teacher tch = std;
43         tch.setInfo();
44         std.displayInfo();
45     }
46 }
```


الخروج من البرنامج :-

```
C:\WINDOWS\system32\cmd.exe
Enter Subject [1] mark : 90
Enter Subject [2] mark : 88
Enter Subject [3] mark :
```

```
C:\WINDOWS\system32\cmd.exe
Enter Subject [1] mark : 90
Enter Subject [2] mark : 88
Enter Subject [3] mark : 87
Mohammed
      Subject [1]      90
      Subject [2]      88
      Subject [3]      87
Press any key to continue . . .
```

في هذا البرنامج تعريف واجهتان 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 std ، وبدلاً عن ذلك قمنا بتعريف مؤشرين references من أنواع الواجهتين، ثم جعلناها تشير إلى الكائن الذي يحتوي على البيانات. يتعامل المؤشر من نوع Student حسب تعريفه مع دالة الطباعة فقط، ولذلك لا يستطيع نداء دالة ادخال المعلومات، لأنه يتعامل مع البيانات عبر واجهة مخصصة للطالب. بالنسبة

للأستاذ فيستطيع التعامل مع الدرجات قراءةً وكتابةً، لأن واجهة الأستاذ تسمح له بذلك،

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.

تعدد الاشكال :-

هناك خاصية هامة جداً توفرها بعض لغات برمجة الكائنات، تضيء 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.

المثال التالي يوضح تعدد الاشكال :

```

1 interface Shapes // interface shapes
2 {
3     public void printArea();
4 } // end of interface
5 class Circle implements Shapes // class circle use interface shapes
6 {
7     int radius ;
8     public Circle(int radius)
9     {
10         this.radius = radius ;
11     }
12     public void printArea()
13     {
14         System.out.println(" the area of circle : " + (3.14 * radius * radius));
15     }
16 } // end of class circle
17 class Square implements Shapes // class square use interface shapes
18 {
19     int side ;
20     public Square(int side)
21     {
22         this.side = side ;
23     }
24     public void printArea()
25     {
26         System.out.println(" the area of square : " + (side * side));
27     }
28 } // end of class square
29 public class Main // the main class of program
30 {
31     public static void main(String[] args)
32     {
33         Shapes s[] = new Shapes[2];
34         s[0] = new Circle(2);
35         s[0].printArea();
36         s[1] = new Square(2);
37         s[1].printArea();
38     } // end of main method
39 } // end of main class

```

الخروج من البرنامج :

```

C:\WINDOWS\system32\cmd.exe
the area of circle : 12.56
the area of square : 4
Press any key to continue . . .

```

يحتوي هذا المثال على نوعين من الاشكال Shapes الدائرة Circle والمربع Square

تعرف كل من هذه الفئات الواجهة Shapes ومن ثمّ تعرف كل فئة محتويات الدالة printArea المستخدمة المساحة ونلاحظ أنّ لكل فئة طريقته الخاصة في حساب المساحة.

تظهر خاصية تعدد الأشكال في الدالة الرئيسية، حيث تم تعريف مصفوفة من نوع الواجهة Shapes ، وهو ممكن بالنسبة للواجهات والفئات المعرفة abstract ، لأنّه يمكن تعريف مؤشرات من هذه الأنواع ولكن لا يمكن تعريف كائنات من نوعها.

الحزم Packages :-

توضع مجموعة الفئات والواجهات التي تنتمي لنفس التطبيق في وحدة تسمى package وتحفظ في ملف يحمل اسم ال package التي تنتمي إليها فئة معينة. يتم كتابة الكلمة المحجوزة package في بداية الملف الذي يحتوي على الفئة ثمّ كتابة اسم ال package .
مثلا :

```
package Application ;
```

ويلي ذلك تعريف الفئة أو الواجهة بالطريقة المعروفة. يوجد عدد كبير من ال 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 ال packages الأخرى فيجب تضمينها في البرنامج بعبارة صريحة للتمكن من استخدامها .
وذلك كالآتي :

```
package Application ;  
import java.util.*;  
public class Main  
{  
|  
}  
}
```

وعلامة * تعني تضمين جميع محتويات ال package في البرنامج، أما لتضمين فئة أو واجهة معينة فيتم كتابة اسمها .

الاستثناءات : –

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 ال object الذي لا وجود له.

ج- استثناءات تتعلق بمستخدم البرنامج:

وهذا النوع يتعلق بالبيانات التي يدخلها المستخدم للبرنامج.

معالجة الاستثناء :-

صممت لغة جافا عدداً من ال classes التي تعبر عن الأخطاء، ووضعت الطرق الملائمة لمعالجة هذه الأخطاء عند حدوثها والتعامل معها. تعرف هذه الطرق بال exception

handling ، والغرض الأساسي منها هو ألا تتأثر صحة واستمرارية البرنامج بحدوث الأخطاء.

تحتوي java.lang package على class اسمه Exception ، يعبر هذا ال class عن خطأ من أي نوع يحدث أثناء تنفيذ البرنامج. هناك عدد كبير من ال classes تعرفه لغة جافا لتمثيل الأخطاء المختلفة، وجميعها ترث خصائص ال Exception class . توفر جافا آلية لمعالجة أخطاء زمن التنفيذ عن طريق مراقبة العبارات المتوقع حصول الخطأ أثناء تنفيذها والاستجابة لهذه الأخطاء في حال حدوثها.

الصيغة العامة لرمي الاستثناء :

```
try
{
    // block of code to monitor for errors
}
catch (ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
    // exception handler for ExceptionType2
}
finally
{
    // block of code to be executed before try block ends
}
```

يتم وضع أي عبارات نتوقع حدوث خطأ فيها أثناء التنفيذ بداخل منطقة محصورة بين قوسين تبدأ بالكلمة المحجوزة try ، تعرف هذه المنطقة بـ block try (كتلة المحاولة) . 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 ال exception وينقطع تنفيذ منطقة try لينتقل التحكم بعدها إلى منطقة catch والمتخصصة بالإمساك بهذه ال exception التي تم توليدها، ولذلك نجد أن catch block (كتلة الالتقاط) تستقبل بين قوسيهما object من نوع exception محدد وعندما يتولد object من نوع exception محدد، فإنه يبحث عن منطقة catch مناسبة لاستقباله ومعالجته.

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.

يعتبر Exception class هو suber class لجميع الـ exceptions الأخرى، وجميعها ترث صفات ومقدرات class exception . العبارة (Exception ex).catch لها المقدر على معالجة أي خطأ يحدث بالبرنامج .

مثال على الاستثناء :-

```
1 public class Testor
2 {
3     public static void main(String args[])
4     {
5         int a = 10;
6         int b = 0;
7         try                // try block
8         {
9             int c = a / b;
10            System.out.println(" Result is : " + c);
11        }
12        catch(ArithmeticException e)    // catch block
13        {
14            System.out.println(" Division By Zero ");
15        }
16        finally                // finally block
17        {
18            System.out.println(" End Program ");
19        }
20    }
21 }
```

المخرج من البرنامج :

```
C:\WINDOWS\system32\cmd.exe
Division By Zero
End Program
Press any key to continue . . .
```

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 .

توفر جافا عدداً كبيراً من الـ 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. الـ 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.

يمكن تعريف ملف في لغة جافا باستخدام الفئة File استعداداً لاستخدامه في البرنامج، وذلك بتحديد اسم الملف عند إنشاء الكائن.

التعامل مع الملفات في لغة الجافا : -

- نستفيد من الدالة exists في التأكد من أن الملف المحدد موجود
- يمكن الحصول على مسار الملف الكامل باستخدام الدالة getPath
- 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 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. المثال التالي يوضح هذه العملية.

```

1  import java.io.*;
2
3  public class Main
4  {
5      public static void main(String args[]) throws IOException
6      {
7          FileInputStream fi = new FileInputStream("welcome.java");
8          FileOutputStream fo = new FileOutputStream("welcome.txt");
9          int g = 0 ;
10         while(g != -1)
11         {
12             g = fi.read();
13             if(g != -1)
14             {
15                 fo.write((char)g);
16             }
17         }
18         fi.close(); |
19         fo.close();
20     }
21 }
22

```

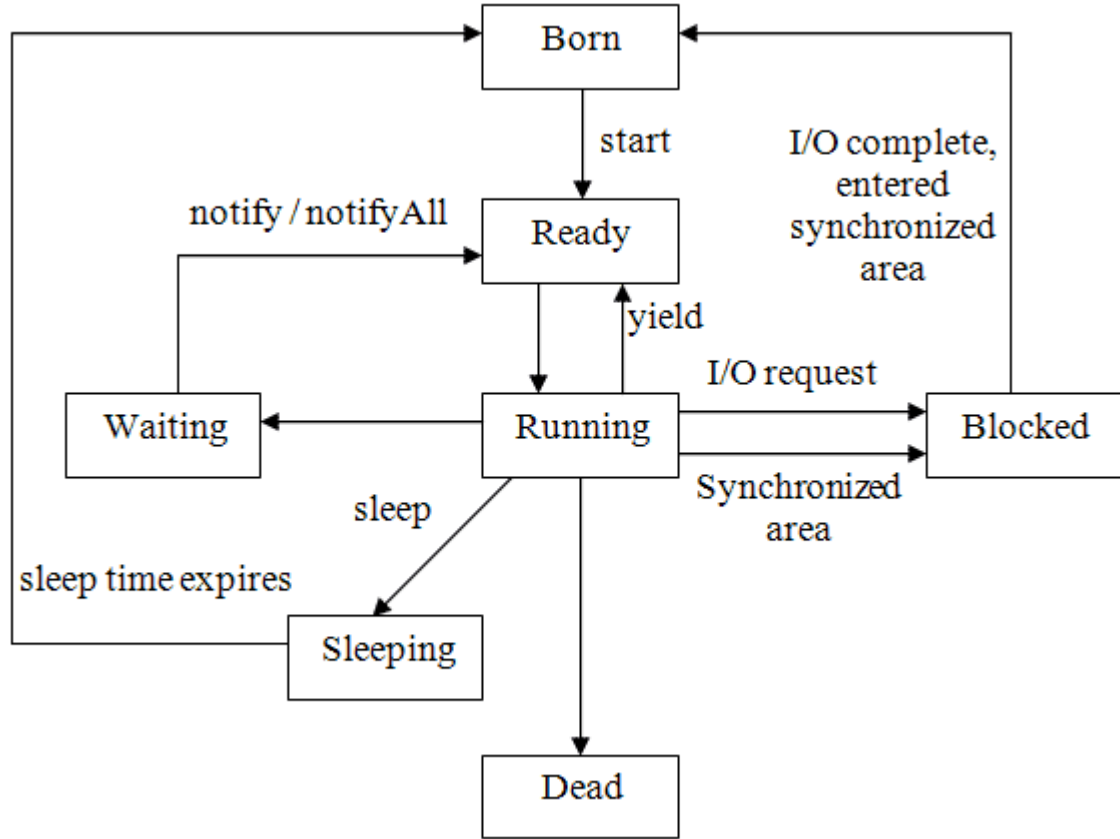
في هذا البرنامج عند تعريف المتغير fi يصبح الملف welcome.java جاهزاً ليقوم البرنامج بقراءة البيانات منه باستخدام الدالة read التي يوفرها FileInputStream class . تتم قراءة محتويات الملف byte تلو الآخر بالمتغير g ثم تخزين هذه القيمة بالملف welcome.txt تبدأ القراءة من الملف من بدايته، وينتقل مؤشر القراءة 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 welcome.java في الملف .welcome.txt

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.

البرمجة المتعددة `Multi threads` :-

تعرف البرامج التي يتم تنفيذها على التوازي مع برامج أخرى بالـ `process` أو `thread` ، كما تعرف برمجة هذا النوع من البرامج بالبرمجة المتعددة `multi-programming` أو `multi-threading` . تدعم لغة جافا هذا النوع من البرمجة .

دورة حياة ال `Thread` :-



الشكل يوضح دورة حياة ال Thread

1. Born : تبدأ دورة حياة ال thread بإعدادها للعمل، حيث يتم تحديد المطلوب منها إنجازه، وذلك يتمثل في تعريفها وإنشائها في البرنامج.

2. Ready : بعد إنشاء ال 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 ال thread في حالة running حينما تكون تعليماتها قيد التنفيذ

بواسطة المعالج، وتكون ال thread في هذه الحالة لفترة زمنية محددة، وبعدها ينتقل

المعالج لتنفيذ thread أخرى، وتعود حينها هذه ال thread إلى حالة ready . يمكن أن نتعمد قطع تنفيذ المعالج ل thread محددة باستخدام الدالة yield ، والتي تحول ال thread إلى حالة ready ، ويقوم المعالج بتنفيذ thread أخرى موجودة في حالة ready .

4. Blocked : يمكن إن يتم إيقاف تنفيذ ال thread لفترة، وذلك بسبب عمليات إدخال وإخراج أو لوجود synchronized code كما سنعرف لاحقا. وتعود ال 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 ال 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 ال 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 لنتنقل جميع ال threads الموجودة في الحالة waiting إلى الحالة ready .

7. Dead : عند انتهاء تنفيذ ال thread نهائياً، تكون قد أدت واجبها وأكملت المطلوب

منها، فتصل إلى آخر حالة في دورة حياتها ويتوقف تنفيذها.

للبرمجة المتعددة ثلاث حالات:

1. أن تكون البرامج مستقلة ويتم تنفيذها بالكامل في نفس الوقت، مثال لذلك تشغيل محتويات قرص مدمج CD أثناء تصفح الإنترنت.
2. أن تكون البرامج مرتبطة أو معتمدة على بعضها البعض. أي أن تكون هناك قيود على ترتيب تنفيذها، مثلاً المخرجات من برنامج معين هي المدخلات لبرنامج ثاني. في هذه الحالة يجب التأكد من انتهاء تنفيذ البرنامج الأول قبل بداية تنفيذ البرنامج الثاني.
3. أن تكون البرامج عبارة عن نسخ متعددة من نفس البرنامج، مثلاً عدة threads تبحث عن رقم معين موجود بين مليون رقم.

أولوية تنفيذ ال Threads

يمكن أن تختلف ال threads من حيث أولوية التنفيذ، كأن يكون تنفيذ أحدها أهم من الآخر. مثلاً 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 برامجها تكون لها أولوية أقل من برامج المستخدم. يتم تحديد أولوية ال thread بواسطة الدالة set Priority . إذا كان هناك عدد من ال threads بأولويات مختلفة جاهزة للتنفيذ، يقوم المعالج بتنفيذ ال thread ذات الأولوية الأعلى حتى تنتهي، ثم يبدأ في تنفيذ ال 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 لل threads ذات الأولويات الأقل. تعتبر الأولوية (1) هي أقل أولوية لل thread في لغة جافا، وأعلى أولوية ممكنة هي (0) . وإذا لم يتم تحديد أولوية معينة لل thread ، تعطى أولوية عادية (0,5) . 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.

```

1 public class MyThread extends Thread
2 {
3     int sleepTime ;
4     public MyThread(String name)
5     {
6         super(name);
7     }
8     public void run()
9     {
10        System.out.println(" Stating Threads " + getName());
11        try
12        {
13            sleepTime = (int) (Math.random() * 10000);
14            System.out.println(" Sleeping for " + sleepTime + " milliSecond ");
15            Thread.sleep(sleepTime);
16        }
17        catch (InterruptedException e)
18        {
19            System.out.println(e.getMessage());
20        }
21        System.out.println(getName() + " Finished ");
22    }
23    public static void main(String[] args)
24    {
25        MyThread t1 = new MyThread(" first ");
26        MyThread t2 = new MyThread(" Second ");
27        MyThread t3 = new MyThread(" third ");
28        t1.start();
29        t2.start();
30        t3.start();
31        System.out.println(" main method is done");
32    }
33 }
34 }

```

لكي يكون البرنامج عبارة عن thread ، يجب أن يرث الـ class المعني class thread ، وهو الذي يعطيه جميع الخصائص التي تجعله قادراً على التنفيذ آنياً مع برامج أخرى وتقاسم زمن المعالج فيما بينها. يوجد class thread في package java.lang ، ويحتوي على constructor يستقبل string تستخدم كاسم يمكن أن يستخدم للتفريق بين الـ threads المختلفة، 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 ال `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 بال `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 ابتداءً من المكان الذي انقطع عنده التنفيذ. قد يتوقف تنفيذ `run` أيضاً إذا ظهرت `thread` ذات أولوية أعلى، لتستمر بعد نهاية تنفيذ تلك ال `thread` . تقوم ال `thread` في هذا المثال بطباعة عبارة `Starting thread`: يليها اسم ال `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 `Math.random * 10000` عدد عشوائي بين 0 و 10000 . ولأن الدالة `sleep` تستقبل عدداً صحيحاً يمثل زمن توقف تنفيذ ال `thread` عن التنفيذ بالمللي ثانية `millisecond` ، لذا يتوقف عمل ال `thread` لزمن عشوائي بين 0 و 10 ثواني يمكن خلالها تنفيذ `threads` أخرى. وبعد أن تعاود ال `thread` التنفيذ، تقوم بطباعة العبارة `finished` مع توضيح اسم ال `thread` . يحتوي البرنامج على الدالة `main` وفيها يتم توليد ثلاثة `threads` بالأسماء `first` , `second` , `third` . بعد ذلك يتم تشغيلها

بتنفيذ الدالة start والتي تقوم تلقائياً بمناداة الدالة run وبداية تنفيذ الـ thread . ننوه إلى أن الدالة main هي نفسها عبارة عن thread يكون تنفيذها مستقلاً عن بقية الـ threads وتتنافس معها على زمن المعالج. يصبح البرنامج أعلاه عبارة عن أربعة threads يتم تنفيذها آنياً. وبما أن لجميعها نفس الأولوية، يمكن لأي منها أن يستهل التنفيذ، ويعتمد إنهاء التنفيذ على قيمة المتغير sleepTime لكل thread ، ويختلف ترتيب وزمن تنفيذ الـ threads كل مرة يتم فيها تشغيل البرنامج.

مخرجات البرنامج :

```
C:\WINDOWS\system32\cmd.exe
main method is done
Stating Threads first
Sleeping for 2394 milliSecond
Stating Threads Second
Sleeping for 1894 milliSecond
Stating Threads third
Sleeping for 5970 milliSecond
Second Finished
first Finished
third Finished
Press any key to continue . . .
```

عند التنفيذ مرة اخرى :

C:\WINDOWS\system32\cmd.exe

```
main method is done
Starting Threads first
Sleeping for 6944 milliSecond
Starting Threads Second
Sleeping for 3456 milliSecond
Starting Threads third
Sleeping for 7430 milliSecond
Second Finished
first Finished
third Finished
Press any key to continue . . .
```

إن الوراثة من class thread ليست هي الطريقة الوحيدة لجعل البرنامج thread ، وقد يكون البرنامج وراثياً أساساً من JFrame أو JApplet أو أي class آخر، ومن المتوقع غالباً أن تكون الـ thread وراثية 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 .

المراجع :-

- البرمجة بلغة جافا - جامعة السودان المفتوحة
- Java how to program 7th edition

الفهرس

رقم الصفحة	الموضوع	الرقم
3	مقدمة في البرمجة بالكائنات	1
5	معدات الوصول	2
5	المشيدات	3
7	المؤشر this	4
8	الوراثة	5
9	محددات الوصول في الوراثة	6
11	التجريد Abstract	7
13	الواجهات interfaces	8
18	تعدد الاشكال	9
20	الحزم Packages	10
21	الاستثناءات Exceptions	11
26	الملفات Files	12
29	البرمجة المتعددة Multi Threads	13
38	المراجع	14