



## أساسيات البرمجة بلغة الجافا

---

### كتاب أساسيات البرمجة بلغة الجافا

يتكون الكتاب من سبعة فصول تشرح الجوانب الأساسية في لغة الجافا، في الفصل الأول من الكتاب نشرح مميزات لغة الجافا والمتغيرات، في الفصل الثاني نوضح عبارات التحكم والحلقات التكرارية، في الفصل الثالث نتحدث عن المصفوفات وأنواعها، في الفصل الرابع نتحدث عن الدوال، في الفصل الخامس ندخل الي مفاهيم البرمجة بالكائنات في الفصل السادس من الكتاب نتحدث عن الوراثة وتعدد الاشكال في الفصل السابع من الكتاب نوضح الاستثناءات والتعامل مع الملفات بالإضافة الي مفهوم البرمجة المتعددة ( Multithreads )

بسم الله الرحمن الرحيم



# أساسيات البرمجة بلغة الجافا

محمد محمود إبراهيم موسى

جامعة الأزهر الأزهرى

كلية علوم الحاسوب وتقنية المعلومات

5	مقدمة:
6	الفصل الاول: مدخل الي لغة البرمجة جافا
6	مميزات لغة الجافا:
7	انواع البيانات في الجافا:
7	المتغيرات:
8	ربط السلاسل نصية:
9	تعريف الثوابت:
10	العمليات الرياضية في الجافا:
10	العبارات المنطقية:
11	قراءة البيانات من المستخدم:
14	الفصل الثاني: جمل التحكم والحلقات التكرارية
14	عبارات المقارنة:
14	جمل الشرط:
21	الحلقات التكرارية:
25	ملاحظات حول الحلقات التكرارية :
27	الفصل الثالث: المصفوفات
27	المصفوفات Arrays:
27	المصفوفات احادية البعد:
29	المصفوفات متعددة البعد:
31	الفصل الرابع: الدوال في الجافا
31	الدوال في الجافا:
31	الدوال الجاهزة:
36	الدوال المعرفة بواسطة المستخدم :
38	استدعاء الدوال:

39	النداء الذاتي:
41	تحميل الدوال بشكل زائد :
42	الفصل الخامس : البرمجة بالكائنات
42	البرمجة بالكائنات:
44	محددات الوصول:
44	المشيدات Constructor:
46	المؤشر this:
47	الفصل السادس: الوراثة وتعدد الأشكال
47	الوراثة:
50	التجريد Abstraction:
52	الفئات والدوال الثابتة:
52	الواجهات:
56	تعدد الأشكال:
58	الحزم Packages:
59	الفصل السابع: الاستثناءات والملفات
59	الاستثناءات:
59	معالجة الاستثناء:
62	الملفات:
63	البرمجة المتعددة Multi threads:
70	المراجع:

**اهداء:**

إلى طلاب كلية علوم الحاسوب وتقانة المعلومات جامعة الزعيم الأزهري إلى الدفعة 16 إلى كل من ساهم في وصول هذا العمل إلى هذا الشكل وأرجوا أن ينال رضائكم.

الحمد لله الواحد المعبود، عم بحكمته الوجود، وشملت رحمته كل موجود، أحمده سبحانه وأشكره وهو بكل لسان محمود، وأشهد أن لا إله إلا الله وحده لا شريك له الغفور الودود، وعد من أطاعه بالعزة والخلود، وتوعد من عصاه بالنار ذات الوقود، وأشهد أن نبينا محمداً عبد الله ورسوله، صاحب المقام المحمود، واللواء المعقود، والحوض المورود، صلى الله عليه وعلى آله وأصحابه، الركع السجود، والتابعين ومن تبعهم من المؤمنين الشهود، وسلم تسليماً كثيراً إلى اليوم الموعود.

تعتبر لغة جافا من اللغات الحديثة جداً في عالم البرمجة، حيث ظهرت بصورة رسمية عام 1990م 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 والحواسيب الدفترية وغيرها.

## الفصل الاول: مدخل الي لغة البرمجة جافا

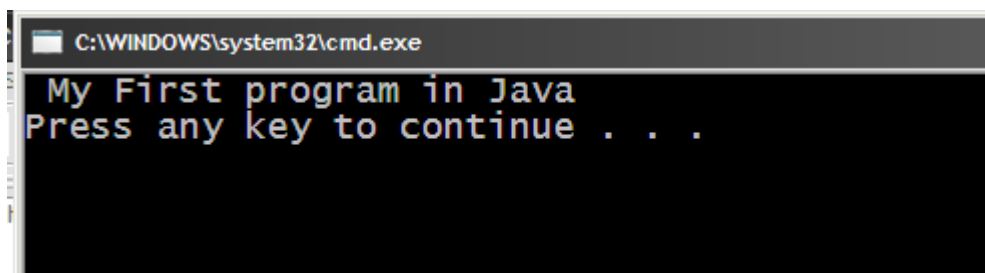
### مميزات لغة الجافا:

- 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) ، ويعتبر هذا الصنف من لغات البرمجة من أوسعها انتشاراً وأكثرها استخداماً 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.

وهذا اول برنامج لنتعرف على محتويات برنامج جافا

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



يبدأ برنامج جافا بالكلمة المحجوزة class يليها اسم البرنامج الذي اختاره المبرمج وهنا first ويجب حفظ الملف بنفس الاسم ويحتوي ال class على الدالة (public static void main(String args[])) ويبدأ تنفيذ البرنامج من هذه الجملة

### انواع البيانات في الجافا:

#### الاعداد الصحيحة:

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

#### الاعداد الحقيقية:

النوع	طوله في الذاكرة
Float	4 byte
Double	8 byte

#### النوع المنطقي:

Boolean ويشمل القيم true او false

#### النوع String:

هذا النوع شائع الاستخدام على الرغم من انه من انواع البيانات غير الاساسية ويستخدم لتعريف النصوص

#### المتغيرات:

#### تعريف المتغيرات:

```
int number1;  
int number1 , number2;  
double num;  
boolean test;  
char ch;  
String text;
```



وضع قيمة للمتغير:

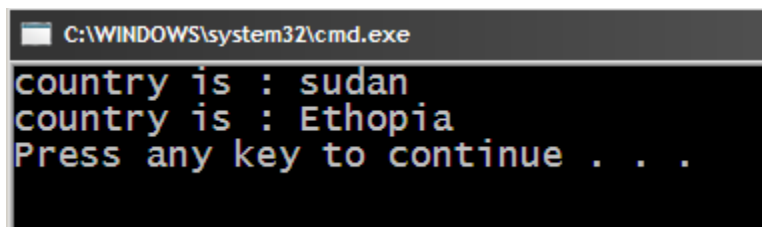
```
number1 = 6;  
num = 6.8;  
text = "Sudan";  
test = true;  
ch = 'a';
```

وهذا يعني ان قيمة المتغير number1 هي 6 وقيمة المتغير test هي true وهكذا لبقية المتغيرات

مثال:

```
1  class country  
2  {  
3      public static void main(String args[])  
4      {  
5          String count;  
6          count = "sudan";  
7          System.out.println("country is : " + count);  
8          count = "Ethopia";  
9          System.out.println("country is : " + count);  
10         }  
11     }
```

الخرج من البرنامج



```
C:\WINDOWS\system32\cmd.exe  
country is : sudan  
country is : Ethopia  
Press any key to continue . . .
```

ربط السلاسل نصية:

لربط السلاسل النصية نستخدم المعامل (+)

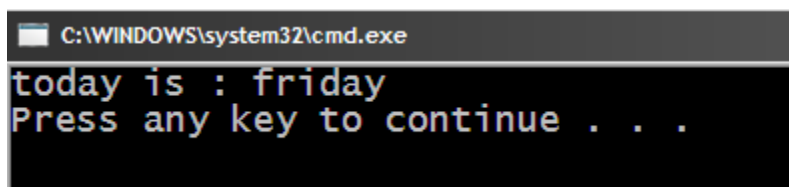
والمثال التالي يوضح ذلك

```

1  class today
2  {
3      public static void main(String args[])
4      {
5          -----
6              String text = "today is : ";
7              String day = "friday";
8              String output = text + day;
9              System.out.println(output);
10
11      }
12  }

```

الخرج من البرنامج



C:\WINDOWS\system32\cmd.exe  
today is : friday  
Press any key to continue . . .

الدالة `print` و `println` تقوم هذه الدوال بعملية الطباعة على الشاشة ولكن الدالة `println` بعد الفراغ من الطباعة تنتقل الى سطر جديد

### تعريف الثوابت:

الثابت هو متغير لا يمكن تغيير قيمته في البرنامج ولكننا 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.

```

final char plus = '+';
final double pi = 3.14;

```

## العمليات الرياضية في الجافا:

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

## العبارات المنطقية:

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

مثال :

```

1 class math
2 {
3     public static void main(String args[])
4     {
5         int num1 = 3, num2 = 4;
6         int sum = num1 + num2;
7         System.out.println("Sum is : " + sum);
8     }
9 }
10
```

الخروج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Sum is : 7
Press any key to continue . . .
```

### قراءة البيانات من المستخدم:

لقراءة البيانات من المستخدم نستخدم الكائن `BufferedReader` الموجود بالحزمة `java.io` والبرنامج التالي يوضح ذلك

```
1 import java.io.*;
2 // to use BufferedReader
3 class input
4 {
5     public static void main(String args[]) throws IOException
6     {
7         String text;
8         char ch;
9         BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
10        System.out.print(" Enter Text : ");
11        text = in.readLine();
12        System.out.print(" Enter Character : ");
13        ch = (char)in.read();
14        System.out.println(text + " - " + ch);
15    }
16 }
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Enter Text :

C:\WINDOWS\system32\cmd.exe
Enter Text : Mohammed
```

```
C:\WINDOWS\system32\cmd.exe
Enter Text : Mohammed
Enter Character : M

C:\WINDOWS\system32\cmd.exe
Enter Text : Mohammed
Enter Character : M
Mohammed - M
Press any key to continue . . .
```

طريقة اخرى لقراءة البيانات من المستخدم:

نستخدم الكائن in الموجود في الحزمة System ولعمل ذلك نستخدم الفئة Scanner الموجودة في

import java.util.scanner

والمثال التالي يوضح ذلك

```
1 import java.util.Scanner;
2 // to import scanner class
3 class input
4 {
5     public static void main(String args[])
6     {
7         Scanner in = new Scanner(System.in);
8         System.out.print(" Enter number : ");
9         int m = in.nextInt();
10        System.out.println(" Number is : " + m);
11    }
12 }
```

الخرج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Enter number : 6
Number is : 6
Press any key to continue . . .
```

ونلاحظ ان هذه الطريقة اسهل في الادخال

لادخال قيمة من النوع char نستخدم `in.nextChar()` و `in.nextString()` للسلاسل

## الفصل الثاني: جمل التحكم والحلقات التكرارية

---

### عبارات المقارنة:

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

### جمل الشرط:

#### عبارة الشرط **if**:

الصيغة العامة لعبارة if

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

والمثال التالي يوضح ذلك

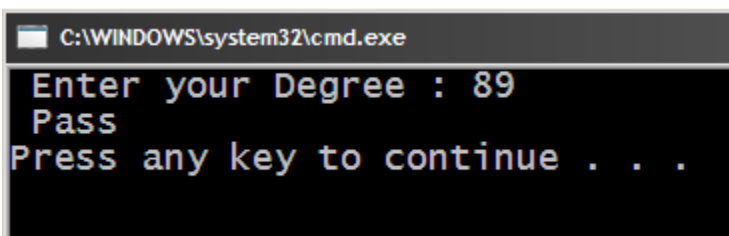
```

1 // if statement program
2 import java.util.*;
3
4 class IfStatement
5 {
6     public static void main(String args[])
7     {
8         int degree;
9         Scanner input = new Scanner(System.in);
10        System.out.print(" Enter your Degree : ");
11        degree = input.nextInt();
12
13        if(degree >= 50)
14        {
15            System.out.println(" Pass ");
16        }
17    }
18 }
19 }

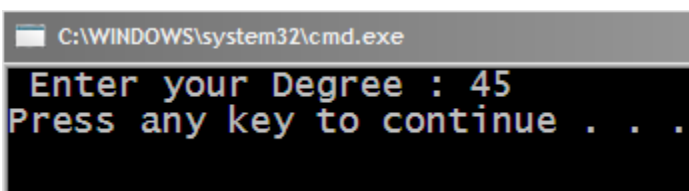
```

الخرج من البرنامج

- القيمة المدخلة اكبر من 50



- القيمة المدخلة اقل من 50



العبارة if else :



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

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

الآن سنقوم بتعديل المثال السابق

```
1 // if statement program
2 import java.util.*;
3
4 class IfStatement
5 {
6     public static void main(String args[])
7     {
8         int degree;
9         Scanner input = new Scanner(System.in);
10        System.out.print(" Enter your Degree : ");
11        degree = input.nextInt();
12
13        if(degree >= 50)
14            System.out.println(" Pass ");
15        else
16            System.out.println(" Fail ");
17    }
18 }
```

الخرج من البرنامج عندما يدخل المستخدم قيمة اكبر من 50

```
C:\WINDOWS\system32\cmd.exe
Enter your Degree : 78
Pass
Press any key to continue . . .
```

وعندما يدخل قيمة اقل من 50

```
C:\WINDOWS\system32\cmd.exe
Enter your Degree : 47
Fail
Press any key to continue . . .
```

مثال:

برنامج يحدد اذا كان العدد يقبل القسمة على 6

```
1 // program to known number is devisable by six
2 import java.util.*;
3 class test1
4 {
5     public static void main(String args[])
6     {
7         Scanner in = new Scanner(System.in);
8         int number1;
9         System.out.print(" Enter number : ");
10        number1 = in.nextInt();
11        if(((number1%2)== 0)&&((number1%3)==0))
12            System.out.println(" number is devisable by 6 ");
13        else
14            System.out.println(" number is not devisable by 6 ");
15    }
16 }
```

الخرج من البرنامج عندما ادخل المستخدم الرقم 24

```
C:\WINDOWS\system32\cmd.exe
Enter number : 24
number is devisable by 6
Press any key to continue . . .
```

عندما ادخل المستخدم الرقم 56

```
C:\WINDOWS\system32\cmd.exe
Enter number : 56
number is not devisable by 6
Press any key to continue . . .
```

ويمكن التعبير عن ال if else بـ :

**max = (number1 < numner2)?number1:number2;**

المثال التالي يوضح ذلك

```
1  import java.util.*;
2  class test2
3  {
4      public static void main(String args[])
5      {
6          Scanner in = new Scanner(System.in);
7          int number1 , number2;
8          System.out.print(" Enter number 1 : ");
9          number1 = in.nextInt();
10         System.out.print(" Enter number 2 : ");
11         number2 = in.nextInt();
12         int max = (number1 > number2) ? number1 : number2;
13         System.out.println(" Maximum number is " + max );
14     }
15 }
```

الخروج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Enter number 1 : 78
Enter number 2 : 90
Maximum number is 90
Press any key to continue . . .
```

العبارة : if else if else..... :

```

if(condition)
    statement;
else if(condition)
    statment;
else
    statement;

```

العبارة **switch**:

الصيغة العامة للعبارة switch

```

switch(variable)
{
    case value1:
        statement;
    break;
    case value2:
        statement;
    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:

```

1  import java.util.*;
2  class switchstatement
3  {
4      public static void main(String args[])
5      {
6          Scanner in = new Scanner(System.in);
7          int digit;
8          String number;
9          System.out.print(" Enter number  : ");
10         digit = in.nextInt();
11         switch(digit)
12         {
13             case 1:
14                 number = "one";
15                 break;
16             case 2:
17                 number = "two";
18                 break;
19             case 3:
20                 number = "three";
21             break;
22             default:
23                 number = " number is " + digit;
24         }
25         System.out.println(number);
26     }
27 }

```

الخرج من البرنامج

```

C:\WINDOWS\system32\cmd.exe
Enter number  : 1
one
Press any key to continue . . .

```

```

C:\WINDOWS\system32\cmd.exe
Enter number  : 4
number is 4
Press any key to continue . . .

```

## الحلقات التكرارية:

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.

### الحلقة **while**:

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

الصيغة العامة للعبارة **while**

```
while(condition)
{
    Statement;
}
```

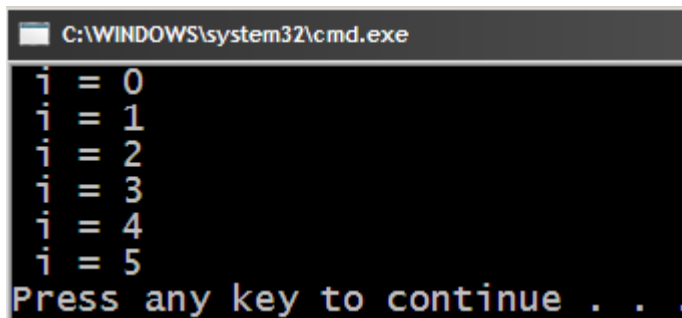
مثال:

```

1  // use while statement
2  class whileloop
3  {
4      public static void main(String args[])
5      {
6          int i = 0;
7          while(i <= 5)
8          {
9              System.out.println(" i = " + i);
10             i++;
11         }
12     }
13 }

```

الخرج من البرنامج



```

C:\WINDOWS\system32\cmd.exe
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
Press any key to continue . . .

```

الحلقة do while:

هي 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.

الصيغة العامة للحلقة do while

```

do
{
    statement;
}while(condition);

```

مثال:

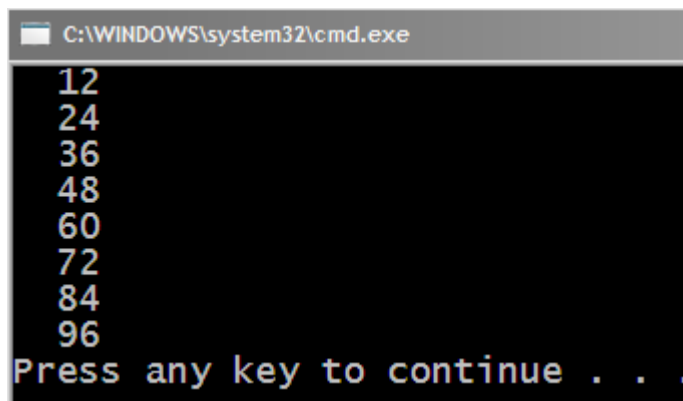
برنامج يقوم بطباعة مضاعفات العدد 12 ال 100

```

1 // use do while statement
2 class dowhileloop
3 {
4     public static void main(String args[])
5     {
6         int i = 12;
7         do
8         {
9             System.out.println(" " + i);
10            i += 12;
11        }while(i <= 100);
12    }
13 }

```

الخرج من البرنامج



```

C:\WINDOWS\system32\cmd.exe
12
24
36
48
60
72
84
96
Press any key to continue . . .

```

### الحلقة for:

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 معرف في incrementExpression حتى يصل إلى القيمة النهائية finalValue ، وعندها يتوقف التكرار.

الصيغ العامة للحلقة for

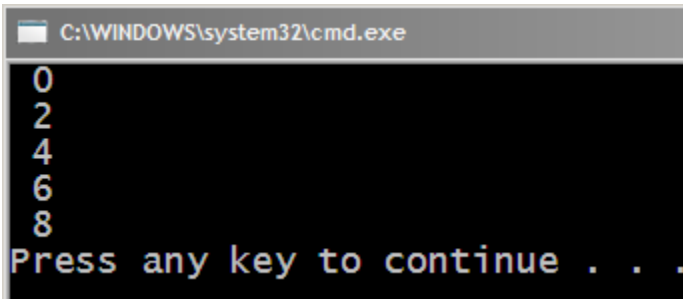


```
for(start;end;frequency)
    statement ;
```

مثال:

```
1 // use for statement
2 class forloop
3 {
4     public static void main(String args[])
5     {
6         for(int i = 0; i < 10; i+=2)
7             System.out.println(" " + i);
8     }
9 }
```

الخرج من البرنامج



```
C:\WINDOWS\system32\cmd.exe
0
2
4
6
8
Press any key to continue . . .
```

الحلقة for المتداخلة:

في هذا المثال نستخدم حلقة for داخل حلقة for اخرى

```

1  // use nested for statement
2  class forloop
3  {
4      public static void main(String args[])
5      {
6          for(int i = 0; i < 5; i++)
7          {
8              for(int j = 0; j < 5; j++)
9                  System.out.print(" * ");
10                 System.out.println();
11             }
12         }
13     }

```

الخرج من البرنامج

```

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

```

### ملاحظات حول الحلقات التكرارية :

- 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 **while do**.
- إذا كنا نحتاج لعدد لمعرفة رقم التكرار أو استخدام قيمته في البرنامج يمكن استخدام حلقة **for** للاستفادة من عددها، 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 **while do**.
- عموماً عند استخدام لغة **Java** يمكن أن نعبر عن أي فكرة بها تكرار بأي 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.

## الفصل الثالث: المصفوفات

---

### المصفوفات Arrays:

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

### المصفوفات احادية البعد:

الصيغة العامة لتعريف المصفوفة احادية البعد

```
DataType[] VariableName = new DataType[Number];
```

او

```
DataType VariableName[] = new DataType[Number];
```

والمثال التالي يوضح تعريف مصفوفة من النوع int

```
int array[] = new int[5];
```

وضع قيم ابتدائية لعناصر المصفوفة:

الشكل التالي يوضح كيفية وضع قيم اولية للمصفوفة array

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

تعريف المصفوفة واعطاءها قيم اولية

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

طباعة عنصر من المصفوفة

```
System.out.println(array[3]);
```

لطباعة كل عناصر المصفوفة نستخدم حلقة for

```
for(int i = 0; i < array.length; i++)  
    System.out.print(" " + array[i]);
```

مثال على المصفوفات احادية البعد

```
1 import java.util.Scanner;  
2 class student  
3 {  
4     public static void main(String args[])  
5     {  
6         Scanner in = new Scanner(System.in);  
7         String name[] = new String[3];  
8         int degree[] = new int[3];  
9         for(int i = 0; i < 3; i++)  
10        {  
11            System.out.print(" Enter Name : ");  
12            name[i] = in.next();  
13            System.out.print(" Enter Degree : ");  
            degree[i] = in.nextInt();  
        }  
        System.out.println(" Name " + " *** " + " Degree ");  
        for(int j = 0; j < name.length; j++)  
        {  
            System.out.println(name[j] + " *** " + degree[j]);  
        }  
    }  
}
```

الخروج من البرنامج

```
C:\WINDOWS\system32\cmd.exe
Enter Name : md
Enter Degree : 78
Enter Name : ea
Enter Degree : 88
Enter Name : mn
Enter Degree : 77
Name *** Degree
md *** 78
ea *** 88
mn *** 77
Press any key to continue . . .
```

### المصفوفات متعددة البعد:

الصيغة العامة لتعريف مصفوفة متعددة البعد

```
int arr2[][] = new int[2][3];
```

تعريف ووضع قيم أولية لمصفوفة متعددة الأبعاد

```
int arr2[][] = new int[][] {{1,2,3},{4,5,6}};
```

طباعة عنصر محدد من المصفوفة متعددة الأبعاد

```
System.out.print(arr2[0][1]);
```

لطباعة كل عناصر المصفوفة نستخدم حلقات for متداخلة

```

for(int i = 0;i < arr2.length;i++)
{
    for(int j = 0;j < arr2[i].length;j++)
    {
        System.out.print(arr2[i][j] + " ");
    }
    System.out.println();
}

```

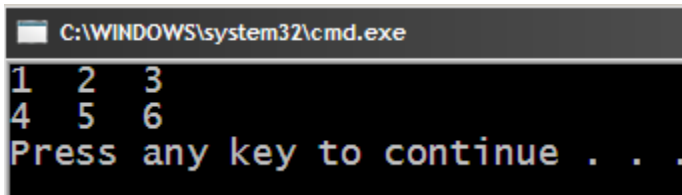
مثال على المصفوفة متعددة البعد

```

1  import java.util.*;
2
3  class arraytest1
4  {
5      public static void main(String args[])
6      {
7          int arr2[][] = new int[][] {{1,2,3},{4,5,6}};
8          for(int i = 0;i < arr2.length;i++)
9          {
10             for(int j = 0;j < arr2[i].length;j++)
11             {
12                 System.out.print(arr2[i][j] + " ");
13             }
14             System.out.println();
15         }
16     }
17 }

```

الخروج من البرنامج



```

C:\WINDOWS\system32\cmd.exe
1 2 3
4 5 6
Press any key to continue . . .

```

## الفصل الرابع: الدوال في الجافا

---

### الدوال في الجافا:

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 والمعروفة أصلاً في لغة **Java** والتي يمكن أن نستخدمها عندما نريد. من أكثر دوال لغة **Java** التي استخدمناها خلال الأمثلة الدالة `print` والدالة `println`، وهما دالتان الغرض منهما الطباعة على الشاشة.

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. ولاستخدام هذه الدوال لا بد من معرفة طريقة كتابتها ونوع وعدد الوسائط التي تأخذها.

### الدوال الجاهزة:

دوال ال `math class`:



الطريقة	وصف الطريقة	مثال
<code>abs (x)</code>	القيمة المطلقة لـ $x$ .	<code>Math.abs (6.2) → 6.2</code> <code>Math.abs (-2.4) → 2.4</code>
<code>ceil (x)</code>	تقرب $x$ إلى أقل عدد صحيح ليس أقل من $x$ .	<code>Math.ceil (5.1) → 6</code> <code>Math.ceil (-5.1) → -5</code>
<code>floor (x)</code>	تقرب $x$ إلى أكبر عدد صحيح ليس أكبر من $x$ .	<code>Math.floor (5.1) → 5</code> <code>Math.floor (-5.1) → -6</code>
<code>max (x, y)</code>	أكبر قيمة من $x$ و $y$ .	<code>Math.max (7, 6) → 7</code>
<code>min (x, y)</code>	أقل قيمة من $x$ و $y$ .	<code>Math.min (-7, -8) → -8</code>
<code>pow (x, y)</code>	$x$ مرفوعة للأس $y$ .	<code>Math.pow (6, 2) → 6<sup>2</sup> → 36</code>
<code>sqrt (x)</code>	الجذر التربيعي لـ $x$ .	<code>Math.sqrt (9) → <math>\sqrt{9}</math> → 3</code>
<code>random ()</code>	تكوّن رقم عشوائي بين الصفر والواحد.	<code>Math.random () → 0.23121</code>

هذه بعض الدوال الموجودة في الفئة `math`

الدوال الخاصة بالسلاسل:

## إيجاد طول السلسلة الرمزية:

ترجع الطريقة <code>length()</code> طول السلسلة الرمزية <code>s</code> .	<code>s.length()</code>
عمليات المقارنة بين سلسلتين رمزيتين (ملاحظة: لا تستخدم <code>==</code> و <code>!=</code> ).	
تقوم الطريقة بمقارنة السلسلة الرمزية <code>s</code> مع السلسلة الرمزية <code>t</code> وتعيد رقم سالب إذا كانت <code>s</code> أقل من <code>t</code> وتعيد صفر إذا كانت <code>s</code> تساوي <code>t</code> وتعيد رقم موجب إذا كانت <code>s</code> أكبر من <code>t</code> .	<code>s.compareTo(t)</code>
تعمل هذه الطريقة بنفس عمل الطريقة <code>compareTo()</code> ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).	<code>s.compareToIgnoreCase(t)</code>
تعيد <code>true</code> إذا كان <code>s</code> يساوي <code>t</code> .	<code>s.equals(t)</code>
تعمل هذه الطريقة بنفس عمل الطريقة <code>equals()</code> ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).	<code>s.equalsIgnoreCase(t)</code>
تعيد <code>true</code> إذا كان <code>s</code> يبدأ بالسلسلة الرمزية <code>t</code> .	<code>s.startsWith(t)</code>
تعيد <code>true</code> إذا كانت السلسلة الرمزية <code>t</code> موجودة في <code>s</code> بدءاً من الموقع <code>i</code> .	<code>s.startsWith(t, i)</code>
تعيد <code>true</code> إذا كان <code>s</code> تنتهي بـ <code>t</code> .	<code>s.endsWith(t)</code>

### عمليات البحث:

كل طرق `indexOf()` تقوم بإرجاع -1 إذا كان العنصر المراد البحث عنه غير موجود، ويمكن للعنصر المراد البحث عنه أن يكون حرف أو سلسلة رمزية.

ترجع موقع أول مكان توجد فيه <code>t</code> داخل السلسلة الرمزية <code>s</code> .	<code>s.indexOf(t)</code>
ترجع موقع أول مكان توجد فيه <code>t</code> داخل السلسلة الرمزية <code>s</code> بعد الموقع <code>i</code> .	<code>s.indexOf(t, i)</code>
ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير <code>c</code> داخل السلسلة الرمزية <code>s</code> .	<code>s.indexOf(c)</code>
ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير <code>c</code> داخل السلسلة الرمزية <code>s</code> بعد الموقع <code>i</code> .	<code>s.indexOf(c, i)</code>
ترجع موقع آخر مكان يوجد فيه الحرف المخزن في المتغير <code>c</code> داخل السلسلة الرمزية <code>s</code> .	<code>s.lastIndexOf(c)</code>
ترجع موقع آخر مكان توجد فيه السلسلة الرمزية <code>t</code> داخل السلسلة الرمزية <code>s</code> .	<code>s.lastIndexOf(t)</code>

عمليات أخذ جزء من السلسلة الرمزية <code>string</code> .	
ترجع الحرف الموجود في الموقع <code>i</code> داخل السلسلة الرمزية <code>s</code> .	<code>s.charAt(i)</code>
ترجع جزء من السلسلة الرمزية <code>s</code> بدءاً من الموقع <code>i</code> وحتى النهاية.	<code>s.substring(i)</code>
ترجع جزء من السلسلة الرمزية <code>s</code> بدءاً من الموقع <code>i</code> وحتى الموقع <code>j-1</code> .	<code>s.substring(i, j)</code>
عمليات التعديل على السلسلة الرمزية <code>string</code> وإنشاء سلسلة رمزية جديدة.	
إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية <code>s</code> بعد تحويل كل الحروف إلى حروف صغيرة.	<code>s.toLowerCase()</code>
إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية <code>s</code> بعد تحويل كل الحروف إلى حروف كبيرة.	<code>s.toUpperCase()</code>
إنشاء سلسلة رمزية جديدة من السلسلة الرمزية <code>s</code> بعد الفارغ من البداية والنهاية.	<code>s.trim()</code>
إنشاء سلسلة رمزية جديدة من السلسلة الرمزية <code>s</code> بعد تبديل كل <code>c1</code> بـ <code>c2</code> ، وهما من نوع <code>char</code> .	<code>s.replace(c1, c2)</code>

عمليات أخرى على السلاسل الرمزية <code>string</code> .	
<code>s.matches(regexStr)</code>	ترجع هذه الطريقة <code>true</code> إذا كانت السلسلة الرمزية <code>regexStr</code> تطابق السلسلة الرمزية <code>s</code> كاملة.
<code>s.replaceAll(regexStr, t)</code>	إنشاء سلسلة رمزية <code>string</code> جديدة بعد تبديل كل <code>regexStr</code> بـ <code>t</code> .
<code>s.replaceFirst(regexStr, t)</code>	إنشاء سلسلة رمزية <code>string</code> جديدة بعد تبديل أول <code>regexStr</code> بـ <code>t</code> .
<code>s.split(regexStr)</code>	إنشاء مصفوفة تحتوي على أجزاء من السلسلة الرمزية <code>s</code> مقسمة حسب ظهور <code>regexStr</code> .
<code>s.split(regexStr, count)</code>	كما في الطريقة <code>split(regexStr)</code> لكن مع تحديد عدد مرات التقسيم.

الدوال المعرفة بواسطة المستخدم :

الشكل العام لتعريف الدالة

```

access static return_type method_name(parameters)
{
    statement1;
    statement2;
    .
    .
    .
}

```

Access محدد الوصول ويكون `public` أو `private` أو `protected`  
`Static` تستخدم لتعريف الدالة ليتم استخدامها داخل الصنف الذي عرفت فيه فقط.

Return\_type يحدد نوع القيم التي تعيدها الدالة

Method\_name اسم الدالة

Parameters هي المعاملات . وعند تعريف الدالة تسمى هذه المعاملات بالمعاملات الشكلية ( Formal

Parameters) وعند استدعاء الدالة تسمى بالمعاملات الفعلية (Actual Parameters)

مثال :

```
public static void university()  
{  
    System.out.println(" Alzaim Alazhari University ");  
}
```

دالة لا تعيد قيمة ولا تحمل وسائط هذه الدالة تقوم بطباعة النص Alzaim Alazhari University

اشكال الدوال:

- دالة لا تأخذ وسائط ولا تعيد قيمة
- دالة تأخذ وسائط ولا تعيد قيمة
- دالة تأخذ وسائط وتعيد قيمة
- دالة لا تأخذ وسائط وتعيد قيمة

مثال يوضح اشكال الدوال :

```

public static void method1 ()
{
    System.out.println("method 1");
}
public static void method2 (String method2)
{
    method2 = "method 2";
    System.out.println(method2);
}
public static int method3 (int x)
{
    return x * x;
}
public static int method4 ()
{
    int x, pi = 3.14;
    return x * pi;
}

```

**استدعاء الدوال:**

يتم استدعاء الدالة باسمها كما في الشكل التالي

```

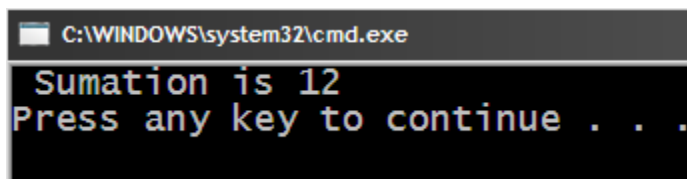
university ();

```

مثال:

```
1  class method
2  {
3      static int sum(int a, int b)
4      {
5          return a * b ;
6      }
7      public static void main(String[] args)
8      {
9
10         System.out.println(" Sumation is " + sum(3,4)) ;
11     }
12 }
```

الخرج من البرنامج



C:\WINDOWS\system32\cmd.exe  
Sumation is 12  
Press any key to continue . . .

**النداء الذاتي:**

هو ان تقوم الدالة باستدعاء نفسها بنفسها . البرنامج التالي يقوم بايجاد مضروب العدد n باستخدام النداء الذاتي



```

1  import java.util.*;
2  class recursion
3  {
4      static int fact(int a)
5      {
6          if(a == 1 || a == 0)
7              return 1;
8          else
9              return a * fact(a - 1);
10     }
11     public static void main(String args[])
12     {
13         Scanner in = new Scanner(System.in);
14         int num;
15         System.out.print(" Enter Number : ");
16         num = in.nextInt();
17         System.out.println(" Factorial : " + fact(num));
18     }
19 }
20

```

الخرج من البرنامج

```

C:\WINDOWS\system32\cmd.exe
Enter Number : 5
Factorial : 120
Press any key to continue . . .

```

ملاحظات:

- 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 .
- 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.

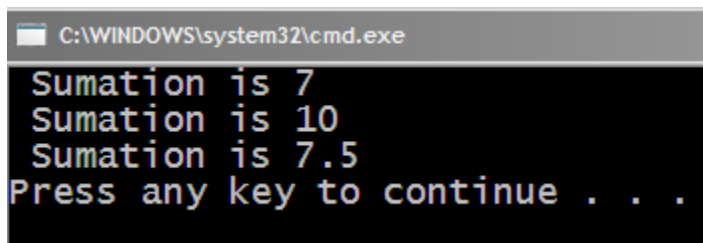
## تحميل الدوال بشكل زائد :

تتم عملية تحميل الدوال بشكل زائد عندما تكون هناك اكثر من دالة تحمل نفس الاسم في نفس الفئة ويتم التمييز بين هذه الدوال من خلال عدد المعاملات التي تحملها وانواعها

مثال :

```
1  class method
2  {
3      static int sum(int a, int b)
4      {
5          return a + b ;
6      }
7      static int sum(int a,int b,int c)
8      {
9          return a + b + c;
10     }
11     static double sum(double a,double b)
12     {
13         return a + b;
14     }
15     public static void main(String[] args)
16     {
17
18         System.out.println(" Sumation is " + sum(3,4));
19         System.out.println(" Sumation is " + sum(3,4,3));
20         System.out.println(" Sumation is " + sum(3.2,4.3));
21     }
22 }
```

الخرج من البرنامج



```
C:\WINDOWS\system32\cmd.exe
Sumation is 7
Sumation is 10
Sumation is 7.5
Press any key to continue . . .
```

### 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 سلفاً 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.

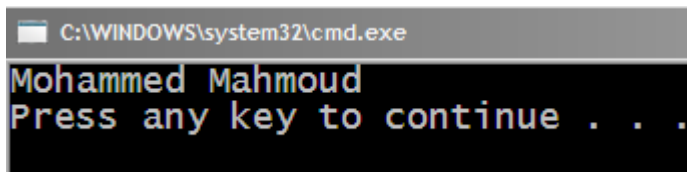
**مثال:**

```

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 بتنفيذ 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 ولا تستطيع 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 تلقائياً 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() والمشييد 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 مشيد تختلف في عدد ونوع بيانات الوسائط وفي هذه الحالة يعرف بالتحميل الزائد للمشييدات constructor overloading .

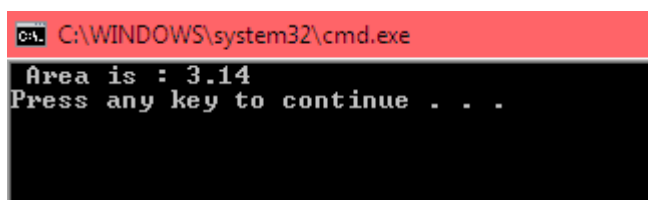
## المؤشر this:

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:



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

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 }

```

الخرج 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 تحتوى على الاقل على دالة واحدة غير مكتملة incomplete . ولعمل فئة مجردة نستخدم الكلمة المحجوزة abstract تحتوى على الاقل على دالة واحدة مجردة abstract ويتم عمل تحميل override عليها . وتعني override امكانية تعديل الدالة في الفئات التي ترث الفئة المجردة .

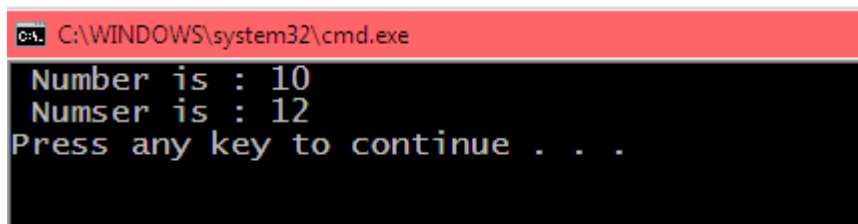
المثال التالي يوضح مفهوم التجريد :-

```

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 }

```

الخرج 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 وحيدة الوراثة single inheritance languages ، وتعني انه يمكن للفئة أن تترث خصائص فئة واحدة على الأكثر. من مميزات هذا النوع سهولة إدارة الكائنات وتحديد علاقتها وصلاحياتها، وله عيب هو عدم التمكن من وراثة خصائص معرفة في عدد من الفئات. تدعم بعض اللغات الوراثة المتعددة multiple inheritance ، وفيها يمكن للفئة أن تترث خصائص أكثر من فئة واحدة. وميزة هذا النوع من الوراثة هو الاستفادة من خصائص عدد أكبر من الفئات، وعيوبها صعوبة إدارة الكائنات وتحديد صلاحياتها. وضعت لغة جافا حلاً 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 وراثة 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 توضح ذلك :

```
public class MyClass implements interface1 , interface2
{
    .
    .
    .
}
```

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 }

```

الخرج 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، تضيف 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

```

الخرج 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 ثم تعرف كل فئة محتويات الدالة printArea المستخدمة المساحة ونلاحظ أنَّ لكل فئة طريقتها 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 ، لأنه 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 الـ package التي تنتمي إليها فئة معينة. يتم كتابة الكلمة المحجوزة package في بداية الملف الذي يحتوي على الفئة ثم كتابة اسم الـ package . مثلا :

```
package Application ;
```

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 . وذلك كالآتي :

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

وعلامة \* تعني تضمين جميع محتويات الـ 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 عموماً 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 بسبب 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 الـ 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 الـ classes التي تعبر عن الأخطاء، ووضعت الطرق الملائمة لمعالجة هذه الأخطاء عند حدوثها والتعامل معها. تعرف هذه الطرق بالـ handling exception ، والغرض الأساسي منها هو ألا تتأثر صحة واستمرارية البرنامج بحدوث الأخطاء.

تحتوي 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

تُرت خصائص الـ 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 try block (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 الـ exception وينقطع تنفيذ منطقة try لينتقل التحكم بعدها إلى منطقة catch والمتخصصة بالإمساك بهذه الـ exception التي تم توليدها، ولذلك نجد أن catch block (كتلة الالتقاط) تستقبل بين قوسيه 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 أنه 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.

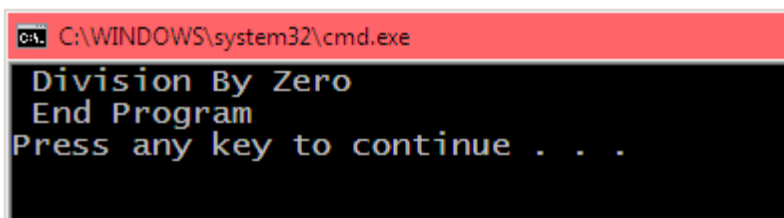
يعتبر Exception class هو suber class لجميع الـ exceptions الأخرى، وجميعها تراث صفات ومقدرات class exception . العبارة catch-(Exception ex) لها المقدرة على معالجة أي خطأ يحدث بالبرنامج .

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 }
```

الخروج من البرنامج :



## 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

```

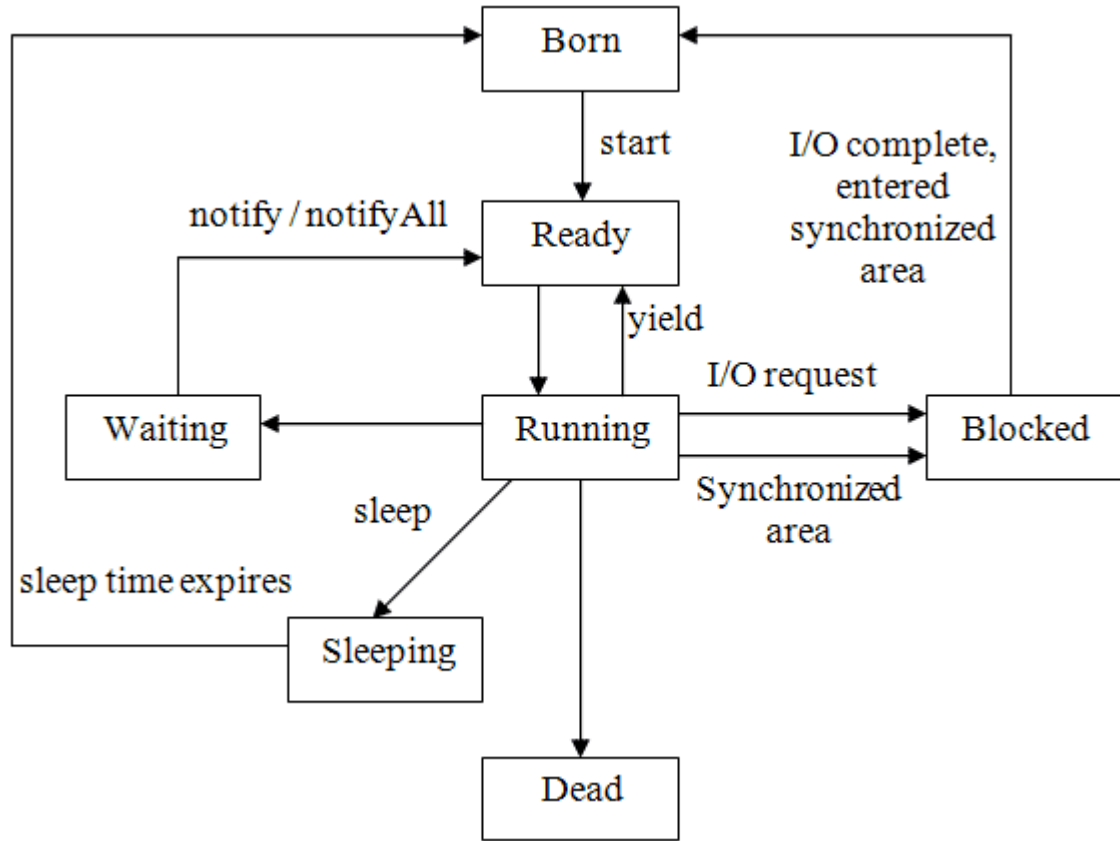
في هذا البرنامج عند تعريف المتغير fi يصبح الملف welcome.java جاهزاً ليقوم البرنامج بقراءة البيانات منه باستخدام الدالة read التي يوفرها FileInputStream class . تتم قراءة محتويات الملف byte تلو الآخر بالمتغير g ثم تخزين هذه القيمة بالملف welcome.txt تبدأ القراءة من الملف من بدايته، وينتقل مؤشر القراءة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





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 نتمتع قطع تنفيذ المعالج لـ thread محددة باستخدام الدالة

yield ، والتي تحول الـ thread إلى حالة ready ، ويقوم المعالج بتنفيذ thread أخرى موجودة في حالة ready .

4. Blocked : يمكن أن يتم إيقاف تنفيذ الـ thread لفترة، وذلك بسبب عمليات إدخال وإخراج أو لوجود synchronized code كما سنعرف لاحقاً. وتعود الـ thread إلى حالة ready بعد الفراغ من عمليات الإدخال والإخراج أو التصريح بدخول المنطقة المعرفة synchronized .

5. Sleeping : يمكن أن يتم قطع تنفيذ thread معينة لفترة زمنية محدودة باستخدام الدالة sleep ، والتي تحدد لها الفترة الزمنية التي نرغب بانقطاع التنفيذ خلالها. تعود الـ 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 الـ 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 لتنقل جميع الـ threads الموجودة في الحالة waiting إلى الحالة ready .

7. Dead : عند انتهاء تنفيذ الـ 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 الـ Threads

يمكن أن تختلف الـ 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 أمراً مهماً 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 الـ thread بواسطة الدالة set Priority. إذا كان هناك عدد من الـ threads بأولويات مختلفة جاهزة للتنفيذ، يقوم المعالج بتنفيذ الـ thread ذات الأولوية الأعلى حتى تنتهي، ثم يبدأ في تنفيذ الـ 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 جميعاً ينتقل للـ threads ذات الأولويات الأقل. تعتبر الأولوية (1) هي أقل أولوية للـ thread في لغة جافا، وأعلى أولوية ممكنة هي (0). وإذا لم يتم تحديد أولوية معينة للـ 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 في 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 الـ 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 بالـ thread إلى الانتقال إلى حالة أخرى لفترة معينة قبل أن 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 الـ thread . تقوم الـ thread في هذا المثال بطباعة عبارة Starting thread: يليها اسم الـ 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 الـ thread عن التنفيذ بالمللي ثانية millisecond ، لذا يتوقف عمل الـ thread لزمن عشوائي بين 0 و 10 ثواني يمكن خلالها تنفيذ threads أخرى. وبعد أن تعاود الـ thread التنفيذ، تقوم بطباعة العبارة finished مع توضيح اسم الـ 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 .

- البرمجة بلغة الجافا – جامعة السودان المفتوحة
- برمجة الحاسب – الإدارة العامة لتصميم وتطوير المناهج المملكة العربية السعودية
- Java How to Program 7<sup>th</sup> Edition