

مقدمة : -

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 والبيجر والحواسيب الدفترية وغيرها.

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.

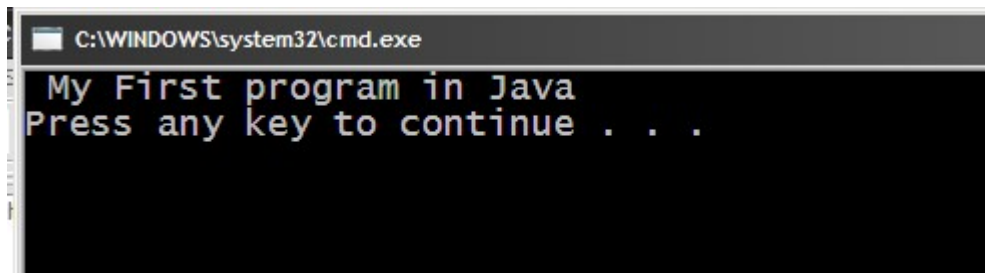
• تعتبر لغة جافا لغة برمجة بالكائنات (Object Oriented Programming)

(Language) ، ويعتبر هذا الصنف من لغات البرمجة من أوسعها انتشاراً وأكثرها استخداماً اليوم.

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

وهذا اول برنامج لتتعرف 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
```



C:\WINDOWS\system32\cmd.exe

```
My First program in Java
Press any key to continue . . .
```

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 : -

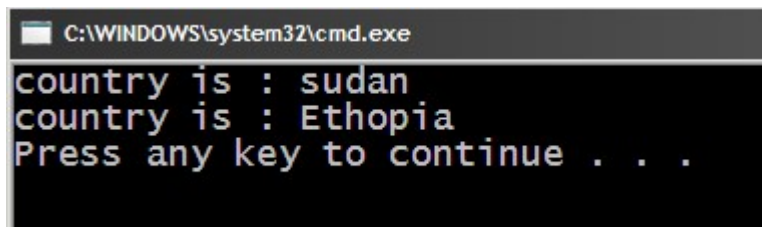
```
number1 = 6;  
num = 6.8;  
text = "Sudan";  
test = true;  
ch = 'a';
```

وهذا يعني ان قيمة المتغير number1 هي 6 و قيمة 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 }
```

الخروج من البرنامج



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

ربط سلاسل 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 }

```

الخروج 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 الي سطر جديد

تعريف الثوابت : -

الثابت هو متغير لا يمكن تغير قيمته في البرنامج ولكننا فقط نقوم بتعريفه ووضع قيمة

ابتدائية له لحظة التعريف، وتظل هذه القيمة ثابتة طوال البرنامج. تعريف الثوابت لا

يختلف عن المتغيرات إلا في الكلمة المحجوزة `final` والتي نكتبها أمام التعريف لنستدل

بها على أنه ثابت.

```
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	%	$a \% b$

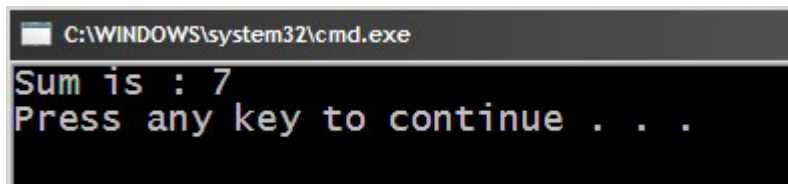
العبارات المنطقية : -

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 }

```

الخروج من البرنامج

The image shows two screenshots of a Windows command prompt window. The first screenshot shows the prompt "Enter Text :". The second screenshot shows the prompt "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 }
```

الخروج 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

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

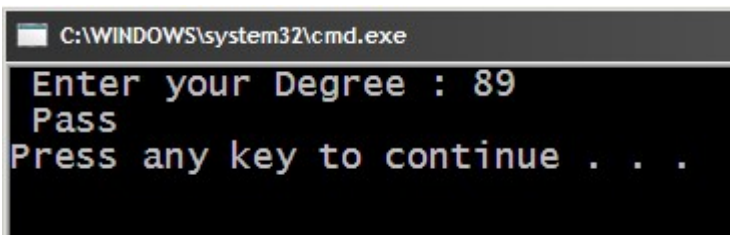
```
if(condition)
{
    statement ;
}
```

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 }
```

الخروج من البرنامج

- القيمة المدخلة اكبر من 50



```
C:\WINDOWS\system32\cmd.exe
Enter your Degree : 89
Pass
Press any key to continue . . .
```

- القيمة المدخلة اقل من 50

```
C:\WINDOWS\system32\cmd.exe
Enter your Degree : 45
Press any key to continue . . .
```

العبارة if else : -

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

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

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 }

```

الخروج 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 . . .

```

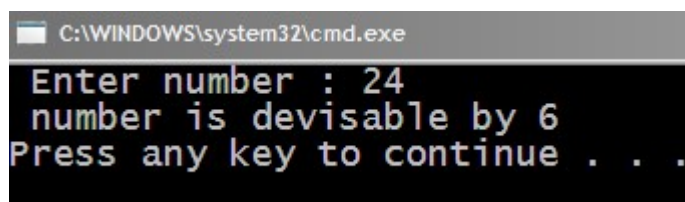
مثال : -

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 }
```

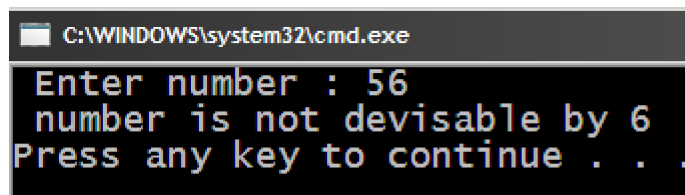
الخروج من البرنامج

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 . . .
```

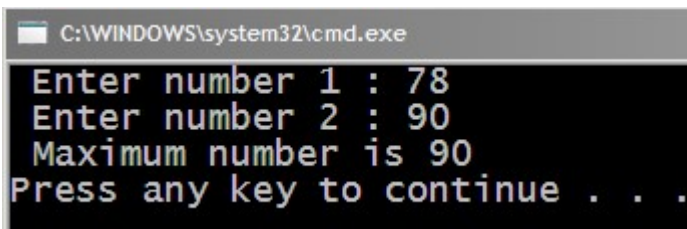
ويمكن التعبير عن ال if else ب

`max = (number1 < number2)?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(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;
                default:
                    number = " number is " + digit;
            }
            System.out.println(number);
        }
    }
```

الخروج 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 }
```

الخروج 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 }

```

الخروج 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.

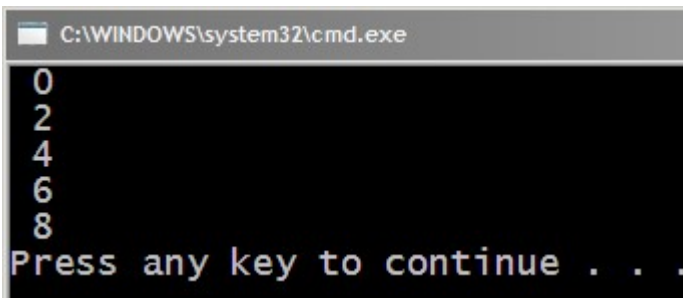
الصيغ العامة للحلقة **for**

```
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 }
```

الخروج من البرنامج



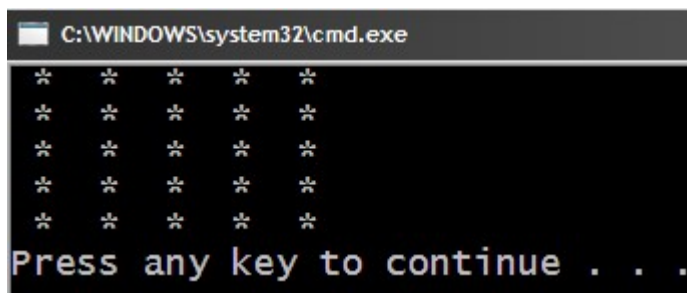
```
C:\WINDOWS\system32\cmd.exe
0
2
4
6
8
Press any key to continue . . .
```

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    }
```

الخروج من البرنامج



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

ملاحظة 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 **do while**.
- إذا كنا نحتاج لعداد لمعرفة رقم التكرار أو استخدام قيمته في البرنامج يمكن استخدام حلقة **for** للاستفادة من عدادها، حيث أن قيمته تبين رقم التكرار.
- إذا كان من الممكن ألا يتم تنفيذ الحلقة أصلاً، فالأصح استخدام حلقة **while**، أما إن كان تنفيذ الحلقة يكتمل للمرة الأولى في كل الأحوال، يتساوى حينها استخدام **while do** و **while**.
- 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.

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 اولية لمصفوفة متعددة الابعاد

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

طباعة عنصر محدد من المصفوفة متعددة الابعاد

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

طباعة 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 . . .

```

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) → 6² → 36</code>
<code>sqrt (x)</code>	الجذر التربيعي لـ x .	<code>Math.sqrt (9) → $\sqrt{9}$ → 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 `string`.

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 `string` 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` او `private` او `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 تاخذ وسائط ولا تعيد قيمة
- دالة تاخذ وسائط ولا تعيد قيمة
- دالة تاخذ 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 }

```

الخروج 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 . . .

```

ملاحظة : -

- 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 : -

تم 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 }

```

الخروج من البرنامج

```

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

```