

JAVA LANGUAGE PROGRAMMING

لغة البرمجة جافا

إعداد / م. إبراهيم الكولي . المعهد التقني التجاري - ذمار .

1-1 ميزات لغة جافا (Java characteristics) :

- . Object Oriented Programming (البرمجة الشيئية (البرمجة بالكائنات))
- . لغة مبنية على لغة البرمجة C/C++
- . هي امكانية تشغيل برامج الجافا على اي منصة تشغيل مثل Platform independent .3 (Windows,Unix,MacOS)
- . تملك الجافا ما يسمى ب Java Virtual Machine (JVM) والتي تعمل كآلية افتراضية لتنفيذ برامج الجافا .

للبدء ، أنت بحاجة الى شيئين مهمين :

1. بيئة تطوير (Integrated Development Environment IDE) وذلك لكي تتمكن من كتابة برامجك مثل (JBuilder,Eclips,NetBeans,...).
2. Java Development Kit (JDK) وهو يعمل كمترجم (Compiler) ومفسر (Interpreter) لكي يقوم بتحويل البرامج وترجمتها الى صيغة يمكن تنفيذها على الكمبيوتر .

1-2 المتغيرات (Variables)

وُتستخدم في حجز حيز في ذاكرة الكمبيوتر ، ويمكن أن تتغير القيمة الموجودة في هذا المتغير.

أنواع المتغيرات :

- الصحيحة (Integers) : لتخزين عدد صحيح

نوع المتغير	الحجم
byte	8 بت
short	16 بت
int	32 بت
long	64 بت

- الحقيقة (float) : لتخزين عدد حقيقي (كسرى)

نوع المتغير	الحجم
float	32 بت
double	64 بت

- الحرفية (characters) : لتخزين حرف واحد او رمز واحد .

نوع المتغير	الحجم
char	16 بت

- المنطقية (Logical) : لتخزين قيمة منطقية إما True أو False

نوع المتغير	الحجم
Boolean	

أمثلة لتعريف متغيرات :

```
int salary ;  
int salary=5000;  
byte x=0;  
float cube=10.5f;  
float vol=10.5;  
char c;  
char k ='*';  
boolean male=false;
```

يتم تعريف المتغير بكتابة نوعه ثم الاسم الذي تريده ثم علامة الفارزة المنقوطة (;) ، ويمكن اسناد قيمة للمتغير مباشرة كما في الامثلة الموضحة اعلاه .

ويجب أن يكون اسم المتغير ذو دلالة على استخدامه وان لا يكون كلمة محجوزة في لغة الجافا .

وعند استخدام المتغير الحرفي يجب ان تكون القيمة مكتوبة بين الرمزين ' ' .

3-1 الثوابت (Constant) :

وتستخدم لتعريف قيمة ثابتة لا تتغير حتى نهاية البرنامج ، نضيف الكلمة المحجوزة final .

مثال :

```
Final float pi =3.14;
```

```
Final int x=7;
```

4-31 التعليقات (Comments) :

وهي عبارة عن جمل لا يتم تنفيذها ، وستستخدم لشرح وتوضيح الكود.

— لسطر واحد نستخدم في بداية السطر // .

مثال :

```
// this method to calculate the average
```

— لعدة أسطر نقوم بكتابه التعليق بين /* . */

مثال :

```
/* this method to calculate the average of student in the second level in  
Dhamar University */
```

5-1 السلسل الحرفية (Strings) :

لتخزين سلسلة من الحروف والرموز مثل اسم الطالب والعنوان .

مثال :

```
String name = " Ahmed ";
```

```
String address= " Sanaa_st ";
```

عند اسناد قيمة للسلسلة نكتبه بين الرموز " " .

ويكن ان يتم تعريف السلسلة على أنها كائن object من الصنف

```
String name = new String ();
```

1-6 المعاملات : (Operators)

1-1 المعاملات الرياضية (Arithmetric operators)

الرمز	اسم المعامل
*	الضرب
/	القسمة
+	الجمع
-	الطرح
%	باقي القسمة
++	الزيادة (التزايد)
--	التناقص

أمثلة :

```
int k=11+3; // k=14
```

```
int k=10*3; // k=30
```

```
int k=10%3; // k=1
```

```
x++; // increasing by one.
```

```
x=x+5; // the same x+=5;
```

```
x=x*5; // the same x*=5;
```

عملية casting :

وهي عملية تجعل المترجم يتعامل مع نوع البيانات على أنه نوع آخر .

مثال :

```
int r=0;
```

```
r= (int ) ( 10-10.5f);
```

ملحوظة : المتغير ذو النطاق الأكبر يستطيع احتواء الأصغر بدون الحاجة لعملية casting

مثال :

```
Byte x=50;
```

```
Short z=x;
```

- الاولوية في تنفيذ العمليات الحسابية :

العملية	الاولوية
()	1
-- ، ++	2
/ ، *	3
%	4
- ، +	5
=	6

1-6-2 المعاملات العلائقية (Relational operators) :

الرمز	اسم المعامل
>	أكبر من
\geq	أكبر من أو يساوي
<	أصغر من
\leq	أصغر من أو يساوي
\neq	لا يساوي
\equiv	يساوي (منطقية) if $x==0$

1-6-3 المعاملات المنطقية (logical operators) :

وتستخدم للمقارنة بين جمل منطقية للبحث عن الصواب أو الخطأ لها .

الرمز	اسم المعامل
$\&\&$	العملية المنطقية (and)
\parallel	العملية المنطقية (or)
\wedge	العملية المنطقية (XOR)
!	العملية المنطقية (not)

الأولوية في تنفيذ المعاملات المنطقية والعلائقية :

العملية	الأولوية
NOT	1
Relational ($<,<=,>,>=$)	2
Equality ($\equiv,=$)	3
$\&\&$	4
\parallel	5

4-6-4 معملات على مستوى البت (bit-wise operators) :

و يتم تفزيذها على مستوى البت :

الرمز	اسم المعامل
&	And
^	Or
!	Not
>>	ازاحة للبين (right shifting)
<<	ازاحة للشمال (left shifting)

مثال :

```
int x=8;
```

```
int y,z;
```

```
y=x>>1;
```

```
x=8;
```

```
z=x<<2;
```

هنا ستكون قيمة المتغير (y=4) لانه قمت ازاحة المتغير (x) بمقدار بت واحد للبين ، وستكون قيمة المتغير (z=32) لانه قمت ازاحة المتغير (x) بمقدار اثنين بت للشمال .

الصيغة العامة لبرنامج المايكروسوفت :

Package package name;

Class class_name {

Variable_type variable1;

Variable_type variable2;

.

Method_name () {

Method body ;

}

}

7-1 برنامج ترحيب :

public class first {

public static void main (String args[])

{

System.out.println(" hello for all student");

}

}

- السطر الاول لتعريف صنف (class) first (public class first) باسم first . وفي برنامج الجافا يوجد صنف واحد على الأقل .

- السطر الثاني كتابة الدالة الرئيسية والتي من خلالها نستدعي كل الدوال في البرنامج .

- السطر الثالث ; System.out.println(" hello for all student") ; لاخراج جملة الترحيب وطباعتها على الشاشة ، ويوجد العديد من اوامر الاخراج في لغة الجافا .

- يجب أن يكون الصنف (class) الذي يحتوي الدالة الرئيسية من النوع public

8-الادخال في الجافا :

توجد عدة طرق لاستقبال المدخلات من المستخدمين ، وهنامنطريق لطريقة واحدة فقط وهي
لستخدام الصنف : scanner
مثال :

```
import java.util.Scanner;  
  
class input {  
  
    public static void main(String args[]){  
  
        Scanner in= new Scanner(System.in);/  
  
        System.out.println(" enter the number ");  
  
        int m = in.nextInt();  
  
        System.out.println(" Number is : "+m);  
  
    } }  
  
Scanner in= new Scanner(System.in);/  
  
int m = in.nextInt();  
  
وتم تعيين متغير لكي يتم تخزين القيمة التي ادخلها المستخدم ;
```

وهي من نوع عدد صحيح .

واخير طباعة الرقم الذي ادخله المستخدم على الشاشة .

ملحوظة : يمكن ادخال انواع كثيرة من البيانات عن طريق تغيير :

nextByte(), nextShort(),nextBoolean(),nextLine () ,next() ,...etc

مثال اخر :

```
import java.util.scanner;  
  
class hello{  
  
public static void main(String args[]){  
  
{  
  
System.out.println(" what is your name ? ");  
  
Scanner name = new Scanner(System.in);  
  
System.out.println(" Hey there "+ name.nextLine());  
  
}}}
```

لقراءة سطر استخدمنا (nextline()

مثال اخر:

```
import java.util.scanner;  
  
class age {  
  
public static void main(String args[]){  
  
System.out.println(" what is your age ? ");  
  
Scanner age = new Scanner(System.in);  
  
System.out.println(" Your age is :" + age.nextLine());  }}
```

برنامـج لــلة حــسبة مــبسـطة :

```
public class Calulator {  
    public static void main(String[] args)  
    {  
        int a=10 ;  
        int b=20 ;  
        int c=25 ;  
        int d=25 ;  
  
        System.out.println("a="+a);  
        System.out.println("b="+b);  
        System.out.println("a+b="+(a+b));  
        System.out.println("a-b="+(a-b));  
        System.out.println("a*b="+(a*b));  
        System.out.println("b/a="+(b/a));}  
}
```

9- جمل التحكم (Control Statement) :

If ●

تقوم بفحص شرط معين فإذا تحقق يتم تنفيذ الجملة (أو الجمل في حالة أنها كانت محصورة بين { }) التي تليها .

مثال :

```
If ( x>50)  
{ System.out.println(" pass ");}
```

If .. else ●

تقوم بفحص الشرط فإذا تتحقق يتم تنفيذ الجملة التي بعد if ، وإذا لم يتحقق يتم تنفيذ الجملة التي بعد else .

مثال :

```
If ( x>50)  
{ System.out.println (" pass");}  
else  
{ System.out.println(" fail"); }
```

If .. else if .. else ●

تقوم بفحص الشرط فإذا تحقق يتم تنفيذ الجملة التي بعد if ، ولا يتم فحص شرط جديد .

مثال :

```
If (x<50)
```

```
System.out.println (" fail");
```

Else if (x=50)

```
System.out.println (" pass");
```

Else

```
System.out.println (" good");
```

(Nested If) ●

: مثال

```
if (x>50)
```

```
If (x<100)
```

```
{ System.out.println ("pass");}
```

Switch... case ●

يقوم بفحص الشرط بمجموعة من الخيارات :

```
Switch (X) {
```

Case 90:

```
System.out.println( " excellent");
```

Break;

Case 80:

```
System.out.println( " very good");
```

Break;

Case 70:

```
System.out.println( " good");
```

```
Break;
```

Case 50:

```
System.out.println( " pass");
```

```
Break;
```

Default

```
System.out.println( " fail");
```

10-1 التكرار : (loop)

وهو عبارة للتكرار تفيذ جمل معينة بعدد محدد بشرط .

: For ●

: مثل

```
For (int i=0; i<5; i++)
```

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

: While ●

: مثل

```
While ( x<5 ) {
```

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

```
x-=1; }
```

: Do .. while ●

: مثل

```
Do{  
    System.out.println(i);  
    x--;  
} while (x<5);
```

- استخدام تعليمة break للخروج من التكرار :
مثال :

```
Class breaktest {  
    Public static void main (String args[]) {  
        int n [ ] ={10,20,30,40,50};  
        int i=0;  
        while ( n[i] <50)  
        {  
            System.out.println(n[i]);  
            If (n[i]==20)  
                break;  
            I++;  
        }  
    }}
```

- استخدام تعليمة continue للاستمرار في التكرار :
مثال :

```
public class Continue_ex{  
    public static void main(String[] args) {  
        for(int i=0; i<n.length; i++)
```

```
{  
if(n[i]==30) {continue; }  
  
System.out.println (n[i]) ; }  
}}
```

مثال : تطوير لبرنامج الحاسبة بحيث يطلب المدخلات من المستخدم وكذلك العملية الحسابية .

```
package calculator;  
  
import java.util.Scanner;  
  
public class Calculator {  
  
    public static void main(String[] args){  
  
        int no1;  
  
        int no2;  
  
        String op ;  
  
        System.out.println("please insert the first number ");  
  
        Scanner a=new Scanner(System.in);  
  
        no1 =a.nextInt();  
  
        System.out.println("please insert the second number ");  
  
        Scanner b=new Scanner(System.in);  
  
        no2=b.nextInt();  
  
        System.out.println("please insert the arithmetic operation ");  
  
        Scanner c=new Scanner(System.in);  
  
        op=c.next();  
  
        switch (op){
```

```
case "+":  
    System.out.println("sum : = "+ (no1+no2) );  
    break;  
  
case "*":  
    System.out.println("mul : = "+ (no1*no2) );  
    break;  
  
case "/":  
    System.out.println("div : = "+ (no1/no2) );  
    break;  
  
case "-":  
    System.out.println("sub : = "+ (no1-no2) );  
    break;  
  
default:  
    System.out.println(" bad insert ");  
}
```

11-1 المصفوفات (Arrays)

وهي عبارة عن مجموعة من العناصر من نفس النوع وللتعامل معها نحتاج الى ثلاثة خطوات :

1- تعريف المصفوفة (declaring the array variable)

2- انشاء كائن المصفوفة (create array object)

3- حزن القيم في المصفوفة (store the value into array)

مثال :

```
String name[];
```

```
int code[];
```

ويمكن تعريفها كالتالي :

```
String [] name;
```

```
int [] code;
```

ويمكن تعريف المصفوفة مع إعطائها قيم إبتدائية :

```
String [] gobs = {"Eng","Doc","Tech"};
```

ولإنشاء كائن المصفوفة نستعمل الكلمة المحوزة new ، فمثلا لانشاء مصفوفة من الاعداد الصحيحة

بأربعة عناصر :

```
int x[] = new int[4];
```

ولاستناد القيمة مثلا الى الموقع [2]x :

```
X[2]=10;
```

مثلا لانشاء مصفوفة من ثلاثة عناصر من الاسماء :

```
String name=new String [3];
```

وهي تكفي الجملتين التاليتين :

```
String name[];
```

```
name=new String [3];
```

مثال : برنامج للتعامل مع مصفوفة :

```
public class Simplearray{
```

```
String [] name={"ali","ahmed","Hani"} ;
```

```
void print(){
```

```
for (int i=0; i<name. Length; i++)
```

```
System.out.println(name[i]); }
```

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

```
Simplearray ar=new Simplearray ();
```

```
ar.print();}}
```

2-11-1 : (Arrays Types) أنواع المصفوفات

● المصفوفات ذات البعد الواحد (one dimension arrays) :

مثال : int x [] ;

● المصفوفات ذات البعدين (Two dimension arrays) :

مثال :

```
int x [ ][ ] = new int [4][3];
```

ولاسناد قيمة : . x [2][3]=20;

و للوصول الى قيمة مخزنة : System.out.println (x[2][3]);

ويستحسن للتعامل مع هذه المصفوفات استخدام تكرارين FOR

```
For (int i=0; i<5;i++)
```

```
    For (j=0;j<5;j++)
```

```
{      }
```

مثال : برنامج لجمع عناصر مصفوفة ذات بعد واحد :

```
Class prog3 {
```

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

```
        int [ ] array1={8,10,50,6,9};
```

```
        int total=0;
```

```
        for (int i=0; i<array1.length; i++)
```

```
            total+=array1[i];
```

```
        System.out.println("total sum of array : "+total );
```

```
    }}
```

(2)

Classes and Objects

تعريف الصنف (class) تقوم بتخصيص وتحديد البيانات التي سيحتويها الصنف وايضاً تقوم بكتابة الكود الذي سيتم تنفيذه على هذه البيانات .

والصيغة العامة لتعريف الصنف :

```
Class className{  
    Type instanceVariable1;  
    Type instanceVariable 2;  
    Type methodName(){  
        //body bo method;  
    }  
}
```

ولإنشاء كائن (object) من هذا الصنف :

```
ClassName Object Name= new ClassName();
```

وللوصول الى المتغيرات او الدوال (methods) والتعامل معها :

```
Object Name. instanceVariable1= VALUE;
```

```
Object Name. methodName();
```

مثال :

تعريف صنف باسم `box` يحتوي على ثلاثة خصائص (متغيرات) وهي الارتفاع (`height`) والعرض (`width`) والعمق (`depth`) ولا يحاجد الحجم (`VOLUME`) فهو حاصل ناتج ضرب (الارتفاع * العرض * العمق) :

```
Class Box{
```

```
    double width;
```

```
    double height;
```

```
    double depth;
```

```
}
```

```
Class boxdemo{
```

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

```
        Box mybox = new Box();
```

```
        double vol ;
```

```
        mybox.width=20;
```

```
        mybox.height=15;
```

```
        mybox.depth=30;
```

```
        vol=mybox.width*mybox.height*mybox.depth;
```

```
        System.out.println("volume is : "+vol); }}
```

يمكنك تعريف أكثر من كائن ، مثلا في المثال السابق :

```
Box mybox1 = new Box();
```

```
Box mybox2 = new Box();
```

ويمكنك التعامل مع محتويات الصنف من بيانات ودوال واسناد القيم الخاصة بكل كائن :

```
mybox2.width=7;
```

```
mybox2.height=8;
```

```
mybox2.depth=9;
```

2-2 الدوال (Methods)

عرفنا سابقاً أن الكائن (object) يملك صفات ويتصرف بشكل معين ، وعرفنا أن الكائن هو عبارة عن تمثيل (instance) للصنف الذي ينتمي إليه .

ف عند بناء الصنف (class) فالصفات يتم تعريفها على أنها البيانات (متغيرات ، ثوابت ،....) والسلوك أو التصرف يتم تعريفها بواسطة الدوال (Methods) التي تنفذ على هذه البيانات.

أي أن الصنف غالباً يحتوي على (instance variable) و ال (Methods).

والصيغة العامة لتعريف الدالة هي :

```
Type Method_name (parameters list) {
```

```
//Body of method.
```

```
}
```

2-3 أنواع الدوال (Method Types)

2-3-1 دالة لا تأخذ وسائط (parameters) ولا تقوم بإرجاع قيمة :

مثال : لنأخذ نفس المثال السابق ببرنامج لا يجاد الحجم :

```
Class B V ox2{
```

```
double w;
```

```
double h;
```

```
double d;
```

```
void volume(){
```

```
System.out.println("volume is :");
```

```
System.out.println( w*h*d);
```

```
}
```

```
Class boxdemo{
```

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

```
    Box2 mybox = new Box2();
```

```
    mybox.w=5;
```

```
    mybox.h=9;
```

```
    mybox.d=10;
```

```
    mybox.volume();
```

```
}
```

2-3 دالة لا تأخذ وسائط ولكنها ترجع قيمة :

هنا نقوم باستخدام الكلمة المحفوظة `return`

: مثال

```
Class box3{
```

```
    double w;
```

```
    double h;
```

```

double d;

double volume (){
    return w*h*d;
}

}

Class boxdemo{

Public static void main(String args []){
    Box3 mybox = new Box3();

    double vol;

    mybox.w=5;

    mybox.h=7;

    mybox.d=12;

    vol=mybox.volume();

    System.out.println("volume is :" +vol);
}
}

```

3-3 دالة تأخذ وسائط وتقوم بإرجاع قيمة :

هنا سنقوم بتمرير الوسائط الى الدالة عند استدعائهما .

مثال :

Class Box4

double w;

double h;

double d;

double volume(double a ,double b, double c){

return a*b*c;

}

}

Class boxdemo{

Public static void main(String args []){

Box4 mybox = new Box4();

double vol;

vol =mybox.volume(5 ,10, 12);

System.out.println(" volume is :" +vol);

}

هنا قمنا بتعريف الدالة على انها تستقبل ثلاثة من الوسائط من نوع double وايضاً ترجع قيمة

vol =mybox.volume(5 ,10, 12) ، وعند استدعاء الدالة قمنا بتمرير القيم لها;

مثال : لبرنامج يطلب من المستخدم ادخال رقم ومن ثم يقوم البرنامج بطباعة مربع هذا الرقم
ومكعبه :

```
import java.util.scanner;

class prg{

int sqr(int y){

int x=0;

x=y*y;

return x;

}

int cubic(int a) {

return a*a*a; }

public static void main (String args [])

prg ob= new prg ();

Scanner in = new Scanner(System.in);

System.out.println(" enter the Number " );

int m= in.nextInt();

System.out.println(" square of "+m+" ="+ prg.sqr(m));

System.out.println(" cubic of "+m+" ="+ prg.cubic(m)); {}}
```

- ملاحظة : استخدام دوال تأخذ وسائل تحمل الدالة أكثر عمومية واستخداماً .

الاستدعاء الذاتي (Recursion) :

وهو استدعاء الدالة لنفسها .

مثال : برنامج يقوم بحساب مضروب عدد صحيح :

```
import java.util.Scanner;

/**
 *
 * @author Eng.Ebraheem
 */

public class FACTORIAL {

    static double fact(double a){

        if ((a==1)|| (a==0))

            return 1;

        else

            return a* fact(a-1);

    }

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

```
double x;
```

```
System.out.println("ادخل الرقم الذي تريد ايجاد المضروب له");
```

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

```
x=n.nextInt();
```

```
System.out.println(" هو " + "" + x + " مضروب العدد " + fact(x));
```

```
}
```

4-2 دوال البناء (Constructors) :

لتهيئة (initialize) كل المتغيرات في الصنف في اي وقت يتم إنشاء كائن من هذا الصنف ، في لغة جافا تسمح للكائنات بإن تهئ نفسها بالقيم الإبتدائية عندما يتم إنشائها ، هذا يتم بإستخدام البيانات (دوال البناء Constructors).

4-2-1 خصائص البيانات (دوال البناء):

- دوال البناء تقوم بتهيئة أي كائن بقيم إبتدائية مباشرةً وقت الإنشاء.
- يكون اسمها بنفس إسم الصنف (وتشبه بنية الدالة Method).
- يتم إستدعاء دالة البناء تلقائياً عند تعلية new .
- لا يمكن أن ترجع قيمة (لا يمكن ان نستخدم تعلية return).

- يمكن للصنف أن يكون له أكثر من دالة بناء .
- اذا لم تقم بكتابة دالة بناء في الصنف ، فإن مترجم الجافا ينشئ دالة بناء افتراضية تلقائياً.

2-4-2 أنواع دوال البناء (Constructors Types) :

1. دوال بناء لا تأخذ وسائط وتسمى بالبنيان الافتراضي (default constructor)
2. دوال بناء تأخذ وسائط وهذا النوع هو الأكثر استخداماً .

مثال : دالة بناء لا تأخذ وسائط :

```
Class box5{
```

```
double w;
```

```
double h;
```

```
double d;
```

```
box5() {
```

```
System.out.println(" contructing box5);
```

```
w=10;
```

```
h=20;
```

```
d=15;
```

```
}
```

```

double volume(){

return w*h*d;

}

Class test{

public static void main (String args []) {

box5 ob = new box5();

double vol;

vol=ob.volume();

System.out.println(" volume is :" + vol);

}}

```

مثال : دالة بناء تأخذ وسائط :

```

class box6 {

int width ;

int height ;

int depth ;

box( int w ,int h, int d) {

width=w;

```

```

hieght=h ;
depth=d ; }

double volume (){
return width*hieght*depth ;

} }

public class Construct {

public static void main(String[] args) {

box6 ob = new box6();

:System.out.println(" volume is " + ob.volume())

} }

```

: مثال :

```

Class Bike{

Bike(){

System.out.println(" bike is created ");

}

Public static void main(String args [ ] ){


```

```
Bike b =new Bike();
```

```
} } ,
```

: مثال

```
Class student {
```

```
int id;
```

```
String name;
```

```
Student (int a , String b){
```

```
id = a;
```

```
name=b;
```

```
}
```

```
void display (){
```

```
System.out.println("id+" "+name);
```

```
}
```

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

```
Student s1=new student (1,"ali");
```

```
Student s2 = new student (2"ahmed");
```

```
}}
```

5-2 تعلیمة this keyword (this)

هناك استخدامات عديدة للكلمة المحجزة this في لغة الجافا ، يعتبر كمرجع يشير الى الكائن الحالي ، سنأخذ الان استخدامين فقط هما :

1. استخدام تعلیمة this كمرجع إلى الكائن الحالي و الوصول الى أعضائه و متغيراته الحالية للصنف (instance variables).
2. () This تستخدم لاستدعاء دالة البناء الخاصة بالصنف .

1-5-2 الاستخدام الاول ل this :

مثال :

```
class student {  
  
    int id;  
  
    String name;  
  
    student(int id,String name){  
  
        this.id=id;  
  
        this.name=name;  
  
    }  
  
    void display(){  
  
        System.out.println(id +"" +name);  
    }  
}
```

```
}
```

```
public class this_ex{  
    public static void main(String args[]){  
        student s1=new student(6,"saleh');  
        student s2=new student(7,"Mohammed');  
        s1.display();  
        s2.display();  
    }  
}
```

2-5-2 الاستخدام الثاني ل this :

هذا الاسلوب مفيد جدا في حالة وجود أكثر من دالة بناء في الصنف ، وتريد أن تستخدم دالة محددة (يتم تحديد الدالة عن طريق الوسائط parameters).

مثال :

```
class student1{  
    int id;  
    String name;  
    student1(){  
        System.out.println(" default constructor is called );  
    }  
}
```

```
student1(int id, String name){  
    this();  
    this.id=id;  
    this.name=name;  
}  
  
void display (){  
    System.out.println(id+ " " + name);  
}  
  
Public class this_ex2{  
    student1 s1=new student (10,"Mosaab Alguraishi");  
    student1 s2=new student (10,"Ebraheem Alkoli");  
    s1.display();  
    s2.display();  
}
```

2- محددات الوصول (Access Control) :

كما عرفت سابقاً فإن من مفاهيم البرمجة بالكائنات (OOP) هو مفهوم التغليف (Encapsulation) والذي يقصد به وضع البيانات مع الدوال في كيان واحد والتحكم بالوصول إليها مع وضع قواعد لمنع أو السماح بقراءة أو تعديل المتغيرات الموجودة في الصنف من صنف آخر .. تسمى هذه القواعد بمحددات الوصول

- Public : يعني أنه يمكن الوصول إليه من أي صنف (class) ومن أي حزمة (package).
- Private : لا يمكن الوصول إليه إلا من الصنف (class) الذي تم التعريف عنه فيه، ولا يمكن الوصول إليه من أي صنف آخر.
- Protected : لا يمكن الوصول إليه إلا من خلال الصنف نفسه أو من أي صنف فرعية (subclass) أو من أي صنف في نفس الحزمة.
- إذا لم تعرف هذه المحددات فستكون من النوع default ، أي يمكن الوصول للمتغير من أي صنف في نفس الحزمة .

7-2 الحزم (Packages) :

هي عبارة عن مجلدات تنظم الاصناف بحيث ان كل مجموعة من الاصناف لها مجموعة مشتركة من الوظائف توضع في حزمة واحدة.

- لإنشاء حزمة :

Package packageName;

- لاستخدام الاصناف الموجودة في حزمة معينة توضع الجملة التالية في بداية البرنامج :

import packageName.*;

(3)

Inheritance

and

polymorphism

1-3 الوراثة (Inheritance) :-

تعتبر الوراثة حجز الزاوية في البرمجة بالكائنات (OOP) واهم مميزاتها ، وتسمح بالتصنيفات الهيكلية ، فبواسطتها يمكنك من إنشاء صنف عام (Common Class) يعرف الميزات العامة لمجموعة من العناصر المرتبطة مع بعضها .

هذا الصنف العام يمكن أن يتم توريثه لأصناف أخرى ، كل من هذه الأصناف تضيف الأشياء الخاصة بها بالإضافة إلى الأشياء التي ورثتها من الصنف العام .

في لغة الجافا الصنف العام التي سيتم التوريث منه يسمى Super class ، والأنواع الوراثة تسمى Sub classes .

إذن الصنف الفرعي (sub class) هو عبارة عن نسخة من الصنف العام (super class) أي أن الصنف الفرعي سيرث كل المتغيرات (instance variable) والدوال (Methods) المعروفة في الصنف العام وسيضيف إليها المتغيرات والدوال الخاصة به .

ومن أهم فوائد الوراثة هي عملية إعادة استخدام الكود (Code-reuse) مما وفر كثير من الوقت والجهد والمال . وأيضاً من فوائدها هو القيام بعملية (Method overriding) .

في لغة الجافا ، إذا أردت وراثة صنف معين قم بإستخدام الكلمة المحوزة (extends) في الصيغة العامة :

```
Class sub_className extends Super_className{ }
```

مثال :

```
class A {
```

```
int i , j;
```

```
void showij(){
```

```
System.out.println("I and j :" + i + "" + j);  
}}
```

```
Class B extends A{
```

```
    int k;
```

```
    void showk(){
```

```
        System.out.println("k :" + k);
```

```
}
```

```
    void sum (){
```

```
        System.out.println("i+j+k" +(i+j+k));
```

```
    }}
```

```
public class inher{
```

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

```
        A supe = new A();
```

```
        B sub = new B();
```

```
        supe.i=100;
```

```
        supe.j=200;
```

```
        System.out.println(" contents of super object ");
```

```
        supe.showij();
```

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

```
sub.i=8;  
sub.j=5;  
sub.l=9;  
  
System.out.println("content of the sub object ");  
  
sub.showij();  
  
sub.showk();  
  
System.out.println();  
  
System.out.println("sum of I ,j and k in the sub :");  
  
sub.sum();  
  
}}
```

مثال اخر :

```
class Employee {  
  
float salasy=4000;  
  
}  
  
class programmer extends Employee{  
  
int bouns=2000;  
  
public static void main (String args[]){  
  
programmer p =new programmer ();  
  
System.out.println("programmer salary :+p.salary);
```

```
System.out.println("programmer bouns :+p.bouns);  
System.out.println("total amount for programmer :+(p.salary+p.  
bouns)); }}
```

1-1-3 الوراثة ودوال البناء : (Inheritance with Constructors)

عند استخدام دوال البناء في الوراثة ، يتم تنفيذ أولا دالة البناء الخاصة بال super class ثم دالة البناء الموجودة في ال sub class

مثال :

```
class a{  
    a(){  
        System.out.println(" a,s construtor");  
    }  
}  
  
class b extends a {  
    b(){  
        System.out.println("b ,s construtors");  
    }  
}
```

```
public class subclassconstruct {  
    public static void main (String args[]){  
        b sub=new b();}}}
```

2-3 الكلمة المحوزة : Super

سنستخدمها هنا لأمرین :

- للوصول الى اعضاء الصنف العام super class من متغيرات ودوال ، وذلك من الصنف الفرعی . sub class

Super.member;

Member may be instance variable or method

- لإستدعاء دوال البناء الخاصة بالصنف العام (super class) .

Super (parameters list);

2-3 تعدد الأشكال (Polymorphism) :

هو مفهوم من مفاهيم البرمجة بالكائنات (OOP) والذي بواسطته يمكن عمل الحدث المفرد بعده طرق .

وكلمة Polymorphism هي مشتقة من كلمتين هما POLY وتعني متعدد وكلمة MORPHS وتعني صيغ أو أشكال .

ويستخدمها يكون البرنامج أكثر عموميةً ،

لغة الجافا توفر طريقتين لتنفيذ هذا المفهوم هما :

.Method Overloading •

.Method Overriding •

: Method Overloading 1-2-3 •

شكل مختصر هو عبارة وجود أكثر من دالة في نفس الصنف ولها نفس الإسم ، لكن مع اختلاف الوسائل (اما بالنوع أو بالعدد).

: مثال :

```
class over{
```

```
    void test(){
```

```
        System.out.println(" no parameters");
```

```
}
```

```
    void test(int a){
```

```
        System.out.println(" a is :" + a);
```

```
}
```

```
    void test(int a , int b){
```

```
        System.out.println(" a and b is :" +a+ " " +b);
```

```
}
```

```
    double test( double b){
```

```

System.out.println(" double a "+b);

return b*b;

} }

public class Overload3 {

public static void main(String[] args) {

over ob=new over();

double r ;

ob.test();

ob.test(10);

ob.test(10, 20);

r=ob.test(2.5);

System.out.println("Results of ob (2.5)= "+r); } }

```

مثال اخر :

```

public class Overloading1 {

void sum (int a,int b){

System.out.println("a+b ="+ " +(a+b));

```

```
}
```

```
void sum (int a, int b , int c){  
  
    System.out.println("a+b+c = "+" "+(a+b+c));  
  
}  
  
public static void main(String[ ] args) {  
  
    Overloading1 ob=new Overloading1();  
  
    ob.sum(10, 20);  
  
    ob.sum(20, 30, 40);} }
```

: Overloading Constructors -1-2-3

يمكنك ان تنشئ أكثر من دالة بناء لنفس الصنف .

: مثال :

```
class box{
```

```
    double width;
```

```
    double height;
```

```
    double depth;
```

```
    box(){
```

```
width=-1;  
  
height=-1;  
  
depth=-1;  
  
}  
  
box(double w, double h, double d){  
  
width=w;  
  
height=h;  
  
depth=d;  
  
}  
  
box(double len){  
  
width=height=depth=len;  
  
}  
  
double volume(){  
  
return width*height*depth;  
  
}  
  
public class Overload {  
  
public static void main(String[] args) {
```

```

box mybox1=new box();
box mybox2=new box(10,20,15);
box mybox3=new box(7);
double vol;
vol=mybox1.volume();
System.out.println(" volume of my box1 is "+vol );
vol=mybox2.volume();
System.out.println(" volume of my box2 is "+vol );
vol=mybox3.volume();
System.out.println(" volume of my box3 is "+vol );
}

```

: Method Overriding -2-2-3

في هيكلة الأصناف عندما تكون الدالة في الصنف الفرعى بنفس الإسم في الصنف العام ونفس الوسائل تكون لدينا عملية .Method Overriding

عندما يتم إستدعاء الدالة في الصنف الفرعى ، ف يتم الاستدعاء للدالة الموجودة في الصنف الفرعى وليس الموجودة في الصنف العام .

أى أن الدالة الموجودة في الصنف العام سيتم إخفائها ، إلا إذا أشرنا إليها بإستخدام الكلمة .super . المحوza

مثال :

```
public class Vechile {  
    // overriding  
    void run (){  
        System.out.println("vechile is running ");  
    }  
  
    class bike extends Vechile {  
        void run (){  
            System.out.println(" bike is running ");  
            super.run();  
        }  
  
        public static void main(String[] args) {  
            bike obj=new bike();  
            obj.run();}}}
```

مثال اخر :

```
class AA{  
    int i,j;
```

```
AA(int a,int b){  
    i=a;  
    j=b;  
}  
  
void show(){  
    System.out.println("i and j :" +i+ " "+j);  
}  
  
}  
  
class b extends AA{  
    int k;  
  
    b(int a, int b, int c ) {  
        super(a, b);  
        k=c;  
    }  
  
    void show(){  
        super.show();  
        System.out.println(" k is "+k);  
    }  
}
```

}

```
public class overid{  
    public static void main (String args[]){  
        b ob=new b(2,4,6);  
        ob.show();}}}
```

(4)

Exception Handling

- ما هو الاستثناء (Exception) ؟

في لغة الجافا : هو عبارة عن حدث يعرقل التساقط (التدفق) الطبيعي للبرناموج ، وبشكل أوضح هو عبارة عن كائن يقوم باعطاء وصف عن الاستثناء (الخطأ) .

عندما يحدث الاستثناء

- معالجة الاستثناء (Exception Handling)

معالجة الاستثناء في لغة الجافا هي واحدة من التقنيات المهمة للتعامل مع الأخطاء التي تحصل في وقت التنفيذ (runtime errors) ومعالجة هذه الأخطاء .

- الفوائد من معالجة الاستثناء :

الميزة الجوهرية من مزايا معالجة الاستثناء هو معالجة التدفق (flow) الغير طبيعي للبرناموج . فعلى سبيل المثال اذا كان برناموج يحتوي على 100 جملة برمجية وحصل خطأ في الجملة رقم 5 ، فإنه سيتوقف التنفيذ عند الجملة رقم 5 وبقية جمل البرنامج من الجملة 6 حتى الجملة 100 لن تتنفذ .

لكن اذا قمت ب مهمة معالجة الاستثناء في برناموجك ، فإن بقية جمل البرنامج من 6 الى 100 سوف يتم تنفيذها ، والجملة رقم 5 فقط هي التي لن تتنفذ .

2-4 أنواع الإستثناءات (Exception Types) :

: Checked Exceptions -1

وهي التي يتم فحصها في وقت ترجمة البرنامج (Compile Time) مثل :
IOException,SQLException

: Unchecked Exception -2

وهي التي يتم فحصها في وقت تنفيذ البرنامج (Run Time) مثل :
. ArithmeticException

VirtualMachineError : وهي أخطاء غير لا يمكن استردادها مثل : Errors -3

أمثلة عن بعض الإستثناءات من نوع : Unchecked Exception

1) **ArithmaticException :**

If we divide any number by zero.

Example :

```
int a = 50/0;
```

2) **NullPointerException :**

If we have null value in any variable, performing any operation by the variable.

Example:

```
String s = null;
```

```
System.out.println(s.length);
```

3) **ArrayIndexOutOfBoundsException:**

If you are inserting value in the wrong index .

Example :

```
int a[ ] =new int [6];
```

```
a[10]=55;
```

3-4 الكلمات المحوزة في لغة الجافا للتعامل مع الاستثناءات :

توفر الجافا خمس تعليمات للتعامل مع الاستثناءات وهي:

try -1

catch -2

throw -3

throws -4

finally -5

تعالج جافا الاستثناءات بهذه الكلمات ،

باختصار ، جمل البرنامج التي تريده مراقبتها تكون محتواه داخل { } اي ان اي خطأ سيحدث داخل هذه ال { } سوف يتم تربيه ويقوم الكود بالإمساك به بإستخدام تعليمة catch ، ويتم معالجته بإسلوب مناسب .

الاستثناءات المولدة من النظام يتم تربيتها تلقائياً بواسطة Java run-time system ، ولكن تمرر الاستثناء يدوياً قم بإستخدام تعليمة throw ، وأي إستثناء يتم تربيه خارج ال Method يجب أن تخصصه بإستخدام . throws

إذا أردت تففيذ جمل برمجية سواءً حصل استثناء او لم يحصل قم بوضعها في { } finally .

الصيغة العامة :

```
try {  
  
// the code you want to monitor from error  
  
}catch (ExceptionType1 exob1){// }  
  
}catch (ExceptionType2 exob2){// }  
  
}finally {// }
```

قبل أن نبدأ ، لنأخذ البرنامج التالي وننظر ما الذي يحصل إذا لم نقم بمعالجة الاستثناء :

```
class exep1{  
    public static void main(String args []){  
        int a=0;  
        int b=5;  
        System.out.println(" exception example");  
        int c=a/b;  
        System.out.println(c);  
        System.out.println("good bye ");  
    }  
}
```

هنا سيكتشف System.java run-time محاولة القسمة على صفر، وسيتم إنشاء كائن استثناء تلقائياً ويتم تجاهله هذا الاستثناء وسيتوقف تنفيذ البرنامج.

في هذا المثال نحن لم نوفر اي شيء لمعالجة الاستثناء ، لذلك الاستثناء السابق سيتم الامساك به من قبل default handler الذي توفره الجافا.

معنى آخر اي استثناء لا يقوم بمعالجته سوف يتم معالجته من قبل default . handler

في هذا المثال ستنتهي التعليمية التي تسبق الخطأ (الاستثناء) ولكن باقي التعليمات من بعد جملة الخطأ لن ت被执行 وسيتم وقف تنفيذ البرنامج كاملا. وسيكون الخرج للبرنامج :

exception handler

Exception in thread "main" java.lang.ArithemticException: / by zero

at exep1.Exep1.main(Exep1.java:22)

حيث تم تنفيذ الجملة التي سبقت الخطأ ، ثم ظهرت رسالة توصف نوع الخطأ مع رقم السطر الذي فيه الخطأ ArithmeticException: / by zero .

3-4 استخدام try و catch :

لعلاج اي مشكلة او خطأ يحصل في وقت التشغيل ،ضع الكود الذي تريد مراقبته داخل

. try { }

: مثال

```
public class Exep2 {  
    public static void main(String[] args) {  
        int d,a;  
        try{  
            d=0;  
            a=50/d;  
            System.out.println(" inside try block");  
        }catch (ArithmaticException e){  
            System.out.println(" division by zero");  
        }  
        System.out.println(" after catch statement");  
    }  
}
```

```
}
```

لاحظ تنفيذ البرنامج ستتجد أن التعليمات التي لم تتنفذ هي التي تقع داخل { } try ، وهنا استثناء واحد هو (القسمة على صفر) تم اكتشافه وتمريره وتم الخروج من { } try والدخول في catch { } وسيستمر تنفيذ بقية تعليمات البرنامج ، وسيكون خرج البرنامج :

division by zero

after catch statement

مثال اخر :

```
public class Ecep_ {  
  
    public static void main(String[] args) {  
  
        int a[] = new int[4];  
  
        try{  
  
            a[50]=7;  
  
            System.out.println(" inside try ");  
  
        }catch (ArrayIndexOutOfBoundsException e){  
  
            System.out.println(" bad insert of index of arrays ");  
  
        }  
  
        System.out.println("after catch ");  
  
    }  
}
```

وسيكون خرج البرنامج :

bad insert of index of arrays

after catch

مثال اخر :

في البرنامج التالي استخدمنا الصنف Random لكي نولد ارقام عشوائية داخل تكرار ، ننفذ عملية القسمة ، فإذا كان المقسم عليه صفر ستتولد حالة استثناء ويتم الامساك بها ومعالجتها ويستمر تنفيذ البرنامج.

```
public class Excep3 {  
  
    public static void main(String[] args) {  
  
        int a=0;  
  
        int b=0;  
  
        int c=0;  
  
        Random r= new Random();  
  
        for (int i=0; i<5;i++){  
  
            try {  
  
                b=r.nextInt();  
  
                c=r.nextInt();  
            }  
        }  
    }  
}
```

```

a=2000/(b/c);

}catch(ArithmeticException e){

System.out.println(" division by zero");

a=0;

}

System.out.println( "a =" +a);

}

}

```

- يمكن عرض الوصف الذي توفره لغة الجافا عن الاستثناء وذلك بتمرير كائن الاستثناء كسيط في الدالة : System.out.println();

```
}catch (ArithmeticException e){
```

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

: مثال

```

public class Exep4 {

public static void main(String[] args) {

int d,a;

```

```

try{
    d=0;
    a=50/d;
}catch (ArithmetricException e){
    System.out.println(e);
}

```

4-4 استثناء بتعدد تعليمات catch (Multiple catch clauses)

في بعض الحالات ، يتولد أكثر من استثناء بواسطة جزء من الكود وللتعامل مع هذه الحالة يمكن استخدام أكثر من تعليمات catch ، كل منها يتعامل مع نوع مختلف من الاستثناءات.

فعندما نقوم بتمرير الاستثناء ، فسيفحص الاستثناء اذا كان مطابق لأي من جمل catch فيتم تنفيذها ويتم تجاهل بقية جمل catch الأخرى .

```

class multicatch{
    public static void main(String args[])
    {
        try{
            int a =0;
            int b=500/a;
            int c[]={2,9,8};
            c[10]=7;
        }
    }
}

```

```

String name;

System.out.println(name.length);

}catch (ArithmaticException e ){

System.out.println("divided by zero"+e);

}catch (ArrayIndexOutOfBoundsException e){

System.out.println("index of array error "+e);

}catch (NullPointerException e){

System.out.println("null error"+e);

System.out.println(" after multi catch block");

}

}

```

في المثال السابق ثلاث من الاستثناءات سوف يتم تجربتها لكن لن ينفذ الا اول استثناء يتم الامساك به بواسطة تعليمة catch وسيكون الخرج كما يلي :

divided by zerojava.lang.ArithmaticException: / by zero

after multi catch block

وإذا قمنا بتغيير قيمة المتغير a بالرقم واحد (int a =1;) اي ان الاستثناء الاول لن يتم الامساك به لانه لا يوجد خطأ سبب لتعليق catch الاولى ، فسيكون خرج البرنامج السابق كما يلي :

index of array error java.lang.ArrayIndexOutOfBoundsException: 10

after multi catch block

5-4 : throw تعلية

حتى الان انت فقط قمت بالامساك (catching) بالاستثناء الذي قام بتمريره نظام الجافا ،
لكن باستخدام تعلية throw ستتمكن من تمرير اي استثناء .

الصيغة العامة :

throw throwable –instance

or

throw new

وكمَا تعرف فإن ال throwable هو الصنف العام (super class) لأصناف الاستثناءات .

في المثال التالي برنامج يقوم بإنشاء وتمرير استثناء ، ثم يقوم بإعادة تمرير هذا الاستثناء خارج
الدالة ليتم امساك به مرة أخرى .

```
public class Excep6 {  
  
    static void my_method(){  
  
        try{  
  
            throw new NullPointerException (" one ");  
  
        }catch (NullPointerException e){  
  
            System.out.println("cought inside my_method");  
        }  
    }  
}
```

```
        throw e;  
    }  
}  
  
public static void main(String[] args) {  
    try{  
        my_method();  
    }catch (NullPointerException e){  
        System.out.println("recaught :" +e );  
    }  
}
```

وسيكون خرج البرنامج :

```
cought inside my_method  
recaught :java.lang.NullPointerException: one
```

5-4 تعلية throws :

اذا كانت الدالة قادرة على الامساك باستثناء وغير قادرة على التعامل معه ،فيجب ان تقوم بتحديد هذا التعامل . وذلك باستخدام **throws** .

الصيغة العامة :

Type method_name(parameters_list) **throws** exception_list

: مثال

```
public class Excep_throw {  
    static void my_method() throws ArithmeticException {  
        System.out.println("inside my_method ");  
        throw new ArithmeticException (" my exception");  
    }  
  
    public static void main(String[ ] args) {  
        try{  
            my_method();  
        }catch (ArithmeticException e){  
            System.out.println("cought "+ e);  
        }  
    }  
}
```

وسيكون خرج البرنامج :

inside my_method

cought java.lang.ArithmaticException: my exception

: finally 6-4 تعلية

تستخدم في تنفيذ التعليمات البرمجية سواءً حدث استثناء أم لم يحدث ، اي ان الكود المكتوب داخل { س يتم تنفيذه دائماً .

مثال :

```
public class finalle_ex {  
    public static void main(String[] args) {  
        int d,a;  
        try{  
            d=0;  
            a=50/d;  
            System.out.println(" this will not printed");  
        }catch (ArithmaticException e){  
            System.out.println(" division by zero");  
        finally {  
    
```

```
        System.out.println(" always excuted");

    }

System.out.println(" after catch statement");

}}
```

خرج البرنامج :

division by zero

always excuted

after catch statement

الآن قم بتجربة انه لن يحصل استثناء في المثال السابق وذلك بتغيير قيمة المتغير d وقم بتعديلها (d=1;) ، سيكون خرج البرنامج كما يلي :

this will not printed

always excuted

after catch statement

اي انه تم تنفيذ { } finally في كلا الحالتين .

مثال اخر :

```
public class Exce9 {

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

```
int i=0;

try {

    if (i==0) throw new Exception ("throw an error");

    System.out.println(" main");

}catch (Exception e){

    System.out.println(e);

finally{

    System.out.println(" finally block");

}}}}
```

خرج البرنامج :

java.lang.Exception: throw an error

finally block

(5)

Multithreading

تعدد المهام (Multitasking) وهو القدرة على تنفيذ أكثر من مهمة في نفس الوقت ،
ويكن تحقيقاً مبدأ تعدد المهام بواسطة شيئاً هما :

-1 : Multiprocessing

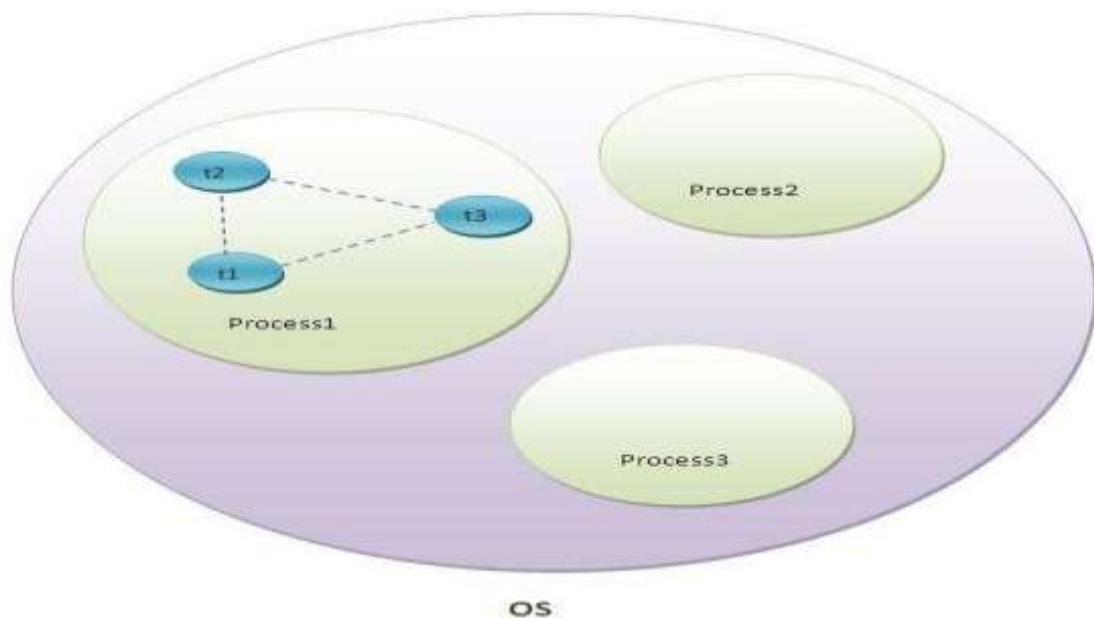
وهو تنفيذ المعالج (Processor) لعدة عمليات (أو برامج) في نفس الوقت ، لأن تقوم
بتتشغيل متعدد متى وفي نفس الوقت تستمع ملف موسيقي وتقوم بتعديل بعض
الصور بإستخدام برنامج الرسام.

-2 : Multithreading

وهو عملية تنفيذ أجزاء من البرنامج (وحدات صغيرة من الكود) بشكل متزامن (في
نفس الوقت)

- Threads هي عبارة عن عمليات فرعية (وحدات صغيرة من المعالجة) في برنامج
واحد.

ويكن القول بأنه في نظام التشغيل يمكن أن يتم تنفيذ أكثر من عملية (Process) ، والعملية
الواحدة يمكن أن يكون فيها أكثر من Thread



توفر لغة الجافا طرفيتين لإنشاء الـ Threads، وستطرق هنا لطريقة واحدة وهي الوراثة من الصنف المعرف في لغة الجافا المسمى (Thread) .

وفيما يلي بعض الدوال الموجودة في الصنف العام (Thread) والتي تُستخدم بشكل كثير:

1- Public void run ():

لاداء حدث معين للـ Thread .

2- Public void start():

لبدء تنفيذ الـ Thread ، وهنا سيتم استدعاء دالة ((run)) تلقائياً .

3- Public void sleep(long milisecond):

لوقف تنفيذ الـ Thread لفترة محددة بالملي ثانية .

4- Public int getPriority():

لاسترجاع قيمة اولوية الـ Thread

5- Public setPriority():

لتغيير قيمة الاولوية الخاصة بالـ Thread

6- Public void setName():

لتغيير اسم الـ Thread

7- Public void getName():

لارجاع اسم الـ Thread

8- Public boolean isAlive():

تفحص اذا كانت الـ Thread لم يتم ايقافها بشكل كامل .

9- Public void suspend():

تُستخدم لايقاف الـ Thread مؤقتاً

10- Public void resume():

تستخدم لاستئناف ال Thread التي تم ايقافها بشكل مؤقت .

11- Public void stop():

تستخدم لايقاف ال Thread بشكل كامل .

: مثال

```
public class Thread1 extends Thread {
```

```
    public void run(){
```

```
        System.out.println(" thread is running ");
```

```
}
```

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

```
        Thread1 t = new Thread1();
```

```
        t.start(); } }
```

: مثال اخر

```
public class test extends Thread {
```

```
    public void run(){
```

```
        System.out.println("running ");
```

```
}
```

```
public static void main(String[] args) {  
  
    test t1=new JavaApplication102();  
  
    test t2= new JavaApplication102();  
  
    t1.start();  
  
    t2.start();  
  
    System.out.println("name of t1 :" +t1.getName());  
  
    System.out.println(" name of t2 :" +t2.getName());  
  
    t1.setName("EBRAHEEM");  
  
    System.out.println("After changing the name of t1 :" +t1.getName());  
  
}  
}
```

: Thread 2-5 حالات ال

يمكن ان تكون ال Thread في عدة حالات هي :

1- New :

عند إنشاء كائن من صنف ال Thread ، لكن قبل تنفيذ دالة () start .

2- Runnable:

ال Thread جاهزة للتنفيذ بعد استدعاء دالة () start . لكن المجدول (scheduler) لم يحدد هذه ال Thread لكي يتم تنفيذها .

3- Running:

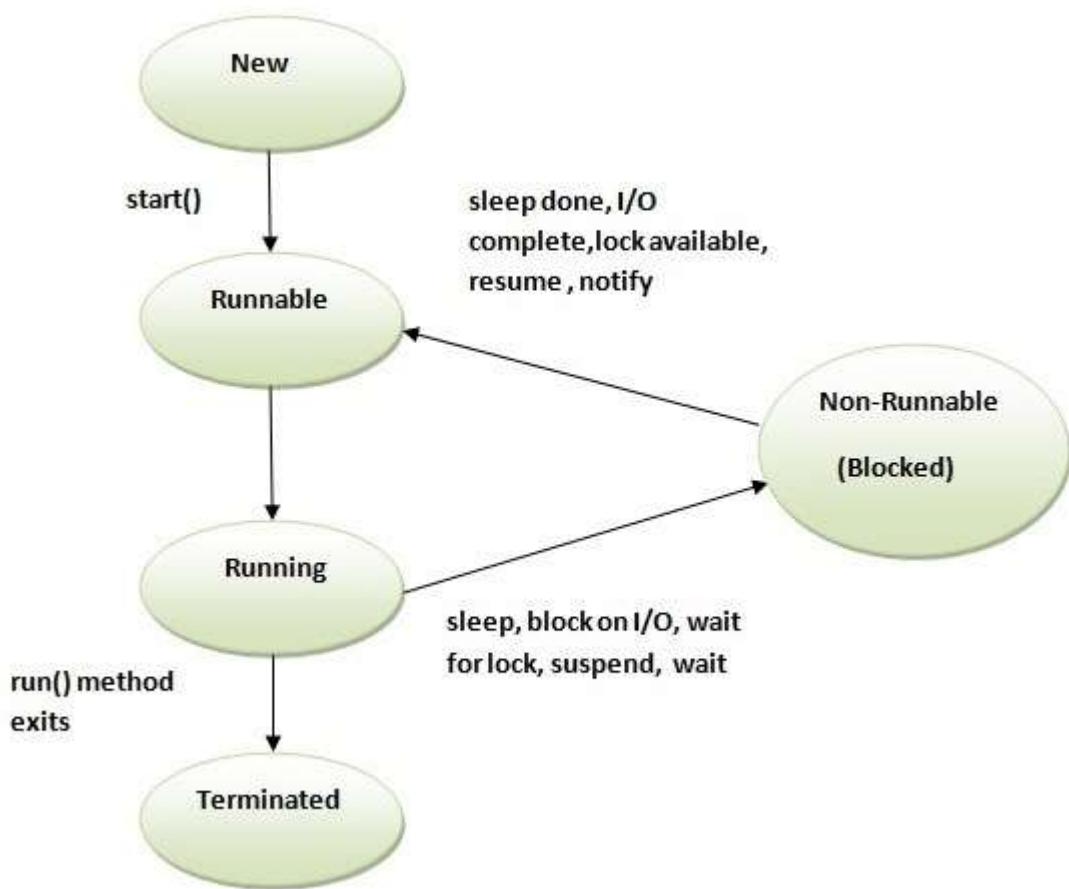
ال Thread في حالة التشغيل بعد أن قام بتحديدها المجدول .

4- Non-Runnable:

عندما ال Thread في حالة ايقاف ، لكنها مؤهلة لاستئناف تشغيلها

5- Terminated :

وهي حالة التوقف الكامل لل Thread ، عندما يتم الخروج من دالة () run .



يمكن ان تكون ال Thread في عدة حالات ، يمكن ان تكون في حالة تنفيذ (running) ، ويمكن أن تكون جاهزة للتنفيذ تنتظر اختيارها من قبل المجدول ، ايضا ال Thread التي تكون في حالة تنفيذ يمكن ان يتم ايقافها بشكل مؤقت (suspend) ، وبعد ذلك يمكن أن نستأنف تنفيذها(resume)، وبعدها يمكن ان تنتهي ال thread (terminated).

مثال :

```

public class Thread3 extends Thread {

    public void run(){

        for(int i=1; i<5;i++){
            try{

```

```

        Thread.sleep(5000);

    }catch (InterruptedException e){

        System.out.println(e);

    }

        System.out.println(i);

    }

}

public static void main(String[] args) {

    Thread3 t1 =new Thread3();

    Thread3 t2 =new Thread3();

    t1.start();

    t2.start();    }

}

```

: مثال اخر :

```

class class11 extends Thread{

private int sleeptime;

private String taskname;

public class11(String taskname ,int sleeptime){

    this.taskname=taskname;

    this.sleeptime=sleeptime;
}

```

```

    }

public void run(){

    try{

        System.out.println("going to sleeping for millisecond"+" "+taskname+"
"+sleeptime);

        Thread.sleep(sleeptime);

    }catch (InterruptedException e){

        System.out.println(e.getStackTrace());

    }

    System.out.println("done sleeping ,"+taskname);

}

}

public class THREAD4 {

    public static void main(String[] args {

        class11 task1=new class11("task1",5000);

        class11 task2=new class11("task2",500);

        class11 task3=new class11("task3",1000);

        task1.start();

        task1.setPriority(1);

```

```
System.out.println(task1.getId());  
System.out.println(task2.getId());  
System.out.println(task3.getId());  
System.out.println(task1.getPriority());  
System.out.println(task2.getPriority());  
System.out.println(task3.getPriority());  
task2.start();  
task3.start();}}
```

ملاحظات :

- لا يمكنك استدعاء الدالة () start مرتين متتاليتين لنفس ال Thread ، ولا تستحصل على خطأ.
- لا تقم بإستدعاء الدالة run بشكل مباشر ، ولكن استخدم الدالة () . start

3-5 مزامنة ال Thread : (Thread Synchronization)

عندما تشارك عدة Threads في نفس المصدر فتحاول احداها ان تقوم بتحديث المصدر بينما الثانية تريد أن تقوم بقراءته ، فايها ستقرأ القيمة الجديدة أم القيمة القديمة للمصدر ، هذه المشكلة يمكن أن يتم معالجتها بأن تعطى ل Thread واحدة الوصول الى المصدر فيما بقية ال Threads تكون في حالة إنتظار ، هذه العملية تسمى (Thread Synchronization) . و يتم تنفيذها في لغة الجافا بإستخدام الكلمة المحفوظة (synchronized) .

Synchronized(object)

{ }

(6)

Graphical User Interface

(GUI)

1-6 مقدمة :

تعطي واجهة المستخدم الرسومية شكلاً جميلاً لبرنامتك ، وترجح المستخدم وت تكون من العديد من المكونات (Components) وهي عبارة عن كائنات تظهر على الشاشة و يستطيع المستخدم التعامل معها بواسطة لوحة المفاتيح وال فأرة .

6-2 عناصر واجهة المستخدم الرسومية :

- المكونات (components) : الكائنات المرئية على الشاشة .
- الخططات (layouts) : التحكم بموقع المكونات .
- الاصدارات (events) : تمثل الاصدارات التي يقوم بها المستخدم .
- الرسوميات (graphics) : الاشكال ، الالوان ، الخطوط ، الخ .

توجد حزمتان في الجافا لدعم الواجهات الرسومية هما :

- awt (قديمة ، مكوناتها قليلة)
- Swing (حديثة ، مكوناتها كثيرة)

3-6 صناديق الحوار (dialog box) :

وهو عبارة عن واجهة بسيطة تعرض رسالة للمستخدم ، وهي من الصنف JOptionPane ولها أنواع ذكر منها :

- ShowMessageDialog();
- ShowConfirmDialog ()

- ShowInputDialog();