البرمجة بلغة جافا
الفصل الخامس
Java Classes

جافا هي لغة برمجة موجهة للكائنات **(OOP) object-oriented programming language**

تدور **البرمجة الإجرائية** حول كتابة الإجراءات أو التوابع التي تؤدي عمليات على البيانات ، **بينما تدور البرمجة الموجهة للكائنات** حول إنشاء كائنات تحتوي على كل من البيانات والتوابع.

تتميز البرمجة **الموجهة للكائنات** بالعديد من المزايا على البرمجة الإجرائية:

١- OOP أسرع وأسهل في التنفيذ.

٢- OOP توفر بنية واضحة للبرامج.

٣- OOP تساعد في الحفاظ على رمز جافا "DRY "Don't Repeat Yourself ، ويسهل الحفاظ على التعليمات البرمجية وتعديلها وتصحيحها.

٤- OOP يجعل من الممكن إنشاء تطبيقات كاملة قابلة لإعادة الاستخدام مع كود أقل ووقت تطوير أقصر.

الصفوف والكائنات: **Classes and Objects**

الصفوف والكائنات هما الجانبان الرئيسيان للبرمجة الشيئية.والجدول التالي يوضح الفرق بين الصفوف والكائنات:

Class	Objects
Student	Ahmad
	Bana
	Salem

example:

Class	objects
Car	BMW
	Lada
	Toyota

عندما يتم إنشاء الكائنات الفردية ، فإنها ترث جميع المتغيرات (الخصائص) والتوابع من الصف.

إنشاء صف: **Create a Class**

لإنشاء صف، نستخدم الكلمة المحجوزة class ثم اسم الصف (يبدأ دائمًا بحرف كبير) ثم قوسي الصف ليتم كتابة الخصائص والتوابع بداخلها كما يلي:

```
class MyClass  {   تعريف المتغيرات والتوابع  }
```

إنشاء كائن

في Java ، يتم إنشاء كائن من صف، بتحديد اسم الصف متبوعًا باسم الكائن ، ونستخدم الكلمة المحجوزة new. ونكتب ذلك في التابع main كما يلي:

```
Class_name   object_name = new   class_name();
```

ثم يتم اسناد القيم لكافة الخصائص كما يلي:

```
Object_name.var_name  = value;
```

مثال:

```
class MyClass  { int  x ;}                    //  إنشاء صف
public class Main {
  public static void main(String[] args) {
    MyClass  myObj = new MyClass();           //    إنشاء كائن
   myObj.x=5;
    System.out.println(myObj.x); } }          // out   5
```

ملاحظة هامة:

بعد إنشاء الكائن من الصف ، **فيمكن عن طريق هذا الكائن الوصول إلى المتغيرات والتوابع داخل هذا الصف**.

ففي المثال السابق الكائن myObj وصل إلى المتغير x، حيث تم اسناد قيمة له وطباعتها.

الوصول إلى الخصائص (متغيرات) والتوابع:

يمكنك الوصول إلى **الخصائص والتوابع** عن طريق إنشاء كائن من الصف وباستخدام النقطة (.) نصل لأي متغير أو تابع معرف داخل الصف.

تمرين:

اكتب برنامجا يحتوي على صف اسمه add نصرح بداخله على متغيرين a,b وتابع sum يقوم بطباعة ناتج جمع المتغيرين a,b بعد اعطائهما قيمة في البرنامج الرئيسي main.

الحل:

```
class Add {                                    // إنشاء صف
int sum(int a,int b)
{ return (a+b);}
 }
public class Main {
  public static void main(String[] args) {
  Add OPj = new Add();                         // Add انشاء كائن من الصف
  int  s=OPj.sum(5,10);                        // وصول الكائن إلى التابع والمتغيرين
  System.out.println(s);
  System.out.println(OPj.sum(4,4));            // استدعاء التابع مرة ثانية
}}
// run:  15
         8
```

تمرين: اكتب برنامجا يحتوي على صف اسمه **Person** عرف ثلاث خصائص بداخله وتابع **printInfo** يقوم بطباعة هذه الخصائص. والمطلوب:

- إنشاء ثلاث كائنات من الصف **Person** في الـ main.
- اسناد خصائص لكل كائن.
- طباعة خصائص كل كائن باستدعاء التابع **printInfo** في البرنامج الرئيسي main.

الحل:

```
class Person {
String  name;                                  // تعريف ثلاث خصائص
String  job;
int  age;

                                               // تابع يطبع محتوى كل خاصية
void printInfo() {
   System.out.print("  Name: " +name);
   System.out.print("  Job: " +job);
   System.out.println("  Age: " +age);
} }
public class Main {
  public static void main(String[] args) {
```

```
        // إنشاء كائنات من الصف Person
        Person p1 = new Person();
        Person p2 = new Person();
        Person p3 = new Person();
        // خصائص الكائن p1
        p1.name = " Ahmad ";
        p1.job  = " Engineer ";
        p1.age  = 25;
        // خصائص الكائن p2
        p2.name = " Bana ";
        p2.job  = " student ";
        p2.age  = 20;
        // خصائص الكائن p3
        p3.name = "Ali";
        p3.job  = " Teacher ";
        p3.age  = 30;
        // عرض خصائص كل كائن
        p1.printInfo();
        p2.printInfo();
        p3.printInfo();
    }   }
run:
Name:  Ahmad      Job:  Engineer      Age: 25
Name:  Bana       Job:  student       Age: 20
Name: Ali         Job:  Teacher       Age: 30
```

إعادة تسمية صف (متغير أو تابع):

يتم ذلك بالضغط على الاسم بالزر الأيمن، ومن القائمة نختار الأمر Refactor ثم نختار rename ونكتب الاسم الجديد وسيتم تحديث الاسم بالبرنامج تلقائيا.

مثال:

```
 class Person {
 String  fname = "Ahmad";
 String  lname = "Ali";
 int age = 20;}
public class Main
 public static void main(String[] args) {
   Person myObj = new Person();
   System.out.println("Name: " + myObj.fname + " " + myObj.lname);
   System.out.println("Age: " + myObj.age);
    }   }Out
Name: Ahmad Ali
 Age: 20
```

التابع البانى: constructor

هو عبارة عن **تابع يحمل نفس اسم الصف ويتم تنفيذه تلقائيا عند إنشاء كائن من هذا الصف** ويتميز بـ:

- يتم تنفيذه تلقائيا بمجرد إنشاء كائن من الصف.
- يستخدم لتهيئة المتغيرات وإعطاء قيم أولية وكتابة التوابع التي تنفذ مع بدء تنفيذ البرنامج.
- يمكن التصريح عنه بمعاملات أو بدونها.
- لا يحتوي على نوع int أو void كما في باقي التوابع ولا يعيد قيمة.
- يمكن إنشاء الباني أكثر من مرة (overload) بشرط أن يختلف في عدد البارامترات.
- يقوم مترجم جافا بإنشاء تابع باني **افتراضي** فارغ إذا كان الصف لا يحتوي على باني.
- لا يوجد destructor (الهادم)في جافا بعكس لغة c++ .

مثال 1:

```
public static class Add {
  Add()                                          // constructor
 { System.out.println("Welcome constructor");}  }
 public class Main {
 public static void main(String[] args) {
   Add OPj = new Add();
}} //run      Welcom constructor
```

لاحظ أن اسم الباني يجب أن يتطابق مع اسم الصف ، ولا يمكن أن يكون له نوع إرجاع (مثل void)
لاحظ أيضًا عند إنشاء الكائن في البرنامج الرئيسي فإنه تلقائيا يتم استدعاء الباني ويطبع الجملة .

مثال 2:

يحوي ثلاث constructors الأول بدون بارمترات ويطبع "Welcome constructor" والثاني والثالث بارمتر واحد وبارمترين على الترتيب وكل منهما يطبع قيمة x , y، بعد تمرير قيم لهما عند إنشاء الكائنات.

```
 class Add {
 int  x,y;
  Add()                                          //constructor No1
   { System.out.println("Welcome constructor");}
 Add(int a)                                       //constructor No2
   {  x=a;
    System.out.println("x=" + x ); }
 Add(int a,int b)                                 //constructor No3
   { x=a; y=b;
    System.out.println("x=" + x + " y=" + y); }    }
  public class Main {
  public static void main(String[] args) {
  Add OPj1 = new Add();
  Add OPj2 = new Add(5);
  Add OPj3 = new Add(7,8);   }  }
run:   Welcome constructor
       x=5
       x=7 y=8
```

أنواع المتغيرات في الصف:

١. Local Variables

هي المتغيرات التي يتم تعريفها بداخل أي تابع، أو باني، أو بداخل block مثل (الحلقات، الجملة switch)

٢. Instance Variables الخاصة(Global Variables)

هي المتغيرات التي يتم تعريفها بداخل الصف و خارج حدود أي تابع، أو باني، أو block .

٣. Class Variables

هي المتغيرات التي يتم تعريفها كـ static بداخل الصف و خارج حدود أي تابع، أو باني، أو block .

مثال:

```
class VarTypes {
  // المتغيرات ( a, b, c, d ) تعتبر Instance Variables لأنه تم تعريفهم بداخل الصف و خارج أي تابع أو block
   int a,b;
   double c,d;
   // static المتغير e يعتبر Class Variable لأن نوعه
   static int e;
   // المتغيرات ( x, y, z ) تعتبر Local Variables لأنه تم تعريفها بداخل التابع
   public int sum(int x, int y) {
      int z = x + y;
      return z;   }
}
public class Main {
public static void main(String[] args) {
   VarTypes OPj1 = new VarTypes();
   VarTypes OPj2 = new VarTypes();
   VarTypes OPj3 = new VarTypes();
    System.out.println(OPj1.sum(5,4));
    OPj2.a=66;
    OPj3.e=77;
    System.out.println( OPj2.a);
    System.out.println( OPj3.e); }}
```

الوراثة (Inheritance):

بواسطة الوراثة يمكن إنشاء صف عام يعرف الميزات العامة لمجموعة من العناصر المرتبطة مع بعضها، وهذا الصف العام يمكن توريثه لصفوف أخرى، كل من هذه الصفوف تضيف الأشياء الخاصة بها بالإضافة إلى الأشياء التي ورثتها من الصف العام.

الصف العام الذي سيتم التوريث منه يسمى الأب (Super class **or** parent class) ، والصف الذي يرث يسمى الابن (Sub class **or** child class).

الصف الفرعي (Sub class) هو عبارة عن نسخة من الصف العام (Super class)، وسيرث كل المتغيرات والتوابع المعرفة في الصف العام **(ما عدا البانى)** وسيضيف المتغيرات والتوابع الخاصة به.

ومن أهم فوائد الوراثة هي عملية إعادة استخدام الكود (Code-reuse) مما يوفر الكثير من الوقت والجهد.

في لغة جافا إذا أردت وراثة صف معين قم باستخدام الكلمة المحجوزة (extends).

الصيغة العامة:

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

مثال 1:

```
 class Super_class {                                    // Super class
    void method_Super()
    { System.out.println("Super class method");}   }
 class Sub_class extends Super_class  {                 //  Sub class
    void method_Sub()
    { System.out.println("Sub class method");}   }
public class Main {
public static void main(String[] args) {
 Sub_class  Ob=new Sub_class();                         //sub class  إنشاء كائن من
  Ob.method_Super();                                    // Super class استدعاء تابع
  Ob.method_Sub();}}                                    // Sub class استدعاء تابع
run:    Super class method
        Sub class method
```

نلاحظ أن الكائن Ob هو من الصف Sub class (الابن) ومن خلاله تم استدعاء التابع **method_Super** الموجود في الصف Super class (الأب)

مثال 2:

```
class Car {                             // Super class
      int size,speed;
      String color; }
class Car1 extends Car  {               // Car1  - Sub class
      String name;
      void showDetail()
           {   System.out.println(name+ " " + color + " " + size + " " + speed);      } }
public class Main {
public static void main(String[] args) {
  Car1  A=new Car1();
    A.name="Mazda";
    A.color="White";
    A.size=2000;
    A.speed=250;
      A.showDetail();
 Car1  B=new Car1();
   B.name="Nissan";
   B.color="Black";
   B.size=1600;
   B.speed=200;
    B.showDetail();   }  }run:
Mazda White 2000 250
Nissan Black 1600 200
```

أنواع الوراثة في جافا

Types of inheritance

توجد في لغة الجافا عدة أنواع من الوراثة منها:

١. الوراثة الاحادية:

صف ابن واحد يرث من صف أب والمثالين السابقين(١+٢) وراثة احادية.

٢. الوراثة متعددة المستويات:

صف ابن واحد يرث من صف أب الذي يرث من صف جد.

مثال:

```
class Super_class {                                  // صف الجد
        void method_Super()
       { System.out.println("Super class method");}    }
class Sub_class1 extends Super_class  {              //  صف الابن الأب
          void method_Sub1()
       { System.out.println("Sub class1 method");}    }
class Sub_class2 extends Sub_class1  {               // صف الابن
        void method_Sub2()
       { System.out.println("Sub class2 method");}    }
public class Main {
public static void main(String[] args) {
 Sub_class2  A=new Sub_class2();                     // كائن من الابن
   A.method_Super();                                 //استدعاء تابع الجد
   A.method_Sub1();                                  //استدعاء تابع الابن الاب
  A.method_Sub2();}}                                 //استدعاء تابع الابن
run:
Super class method
Sub  class1 method
Sub  class2 method
```

٣. الوراثة الهرمية:

أكثر من **صف ابن يرث صف أب واحد**.

مثال:

```
class Super_class {                                  // Super class
      void method_Super()
      { System.out.println("Super class method");}   }
class Sub_class1 extends Super_class  {              //  Sub class1
      void method_Sub1()
     { System.out.println("Sub1 class method");}    }
class Sub_class2 extends Super_class  {              //  Sub class2
      void method_Sub2()
     { System.out.println("Sub2 class method");}    }
class Sub_class3 extends Super_class  {              //  Sub class3
       void method_Sub3()
      { System.out.println("Sub3 class method");}    }
```

```
public class Main {
public static void main(String[] args) {
  Sub_class1  Ob1=new Sub_class1();                    //إنشاء كائن من sub class1
  Sub_class2  Ob2=new Sub_class2();                    //إنشاء كائن من sub class2
  Sub_class3  Ob3=new Sub_class3();                    //إنشاء كائن من sub class3
    Ob1.method_Super();                               // استدعاء تابعSuper class
    Ob2.method_Super();                               // استدعاء تابعSuper class
    Ob3.method_Super();                               // استدعاء تابعSuper class
    Ob1.method_Sub1();                                // استدعاء تابع Sub class1
    Ob2.method_Sub2();                                // استدعاء تابع Sub class2
    Ob3.method_Sub3();                                // استدعاء تابع Sub class3
}}run  Super class method
       Super class method
       Super class method
       Sub1 class method
       Sub2 class method
       Sub3 class method
```

الكلمة المفتاحية super :

تستخدم للوصول إلى المتغيرات والتوابع (**التابع الباني**) الموجودة في صف الأب **من داخل صف الابن** وخصوصا عند وجود تشابه بين أسماء المتغيرات وأنواعها في صفي الأب والأبن.

طريقة استخدام الكلمة super لاستدعاء متغير أو تابع من الـ Superclass

نضع الكلمة **super**، بعدها نقطة، ثم نضع اسم المتغير أو التابع الذي نريد استدعائه من الـ Superclass.

```
super.variableName
super.methodName();
```

يمكن استدعاء تابع باني موجود في الـ Superclass **من داخل تابع باني** في الـ Subclass كما يلي:

```
super()                                  //استدعاء تابع باني فارغ
super( parameter List)                   //استدعاء تابع باني يحوي بارامترات
```

مثال: عن استخدام الكلمة super لاستدعاء متغير من صف الأب.

```
class Person {                                    // صف الاب
       String name="Ahmad";                       //متغيرات صف الاب
       int id=10;    }
class Student extends Person {                    // صف الابن
       String name="Hassan";
       int id=230;
 void print()  {                                  // طباعة المتغيرات الموجود في صف الاب
       System.out.println(super.name);
       System.out.println(super.id); }}
public class Main {
public static void main(String[] args){
Student obj=new Student();                        // انشاء كائن من صف الابن
obj.print();}}                                    // استدعاء التابع الموجود في صف الابن
```

```
run:   Ahmad
       10
```

مثال: عن استخدام الكلمة super لاستدعاء تابع من صف الأب.

```
class AA {                                  // Super class
    void print()
    { System.out.println("print method from the class AA");}
   }
class BB extends AA  {                      //  Sub class
   public void print()
        {  System.out.println("print method from the class BB");}
   void PrintBoth()
        {   print();                        // BB استدعاء التابع print() الموجود في الكلاس
            super.print();                  // AA استدعاء التابع print() الموجود في الكلاس
        }   }
public class Main {
public static void main(String[] args) {
  BB  Ob=new BB();
   Ob.PrintBoth();
}}
run:
print method from the class BB
print method from the class AA
```

مثال آخر

```
class Device   {
    Device ()
          { System.out.println("default Parent Constructor");}
   Device (int x)
          {  System.out.println("Second Parent Constructor"); }
  void print()
    {   System.out.println("Method in Parent class");}    }
class Computer extends Device  {
    Computer()
       {  super.print(); }                          // استدعاء التابع print في الصف الاب
 }
public class Main {
public static void main(String[] args) {
    Computer obj=new Computer();
} }
run:
default Parent Constructor
Method in Parent class
```

مثال آخر

```
class Animal{
        void move()
          {  System.out.println("Animals can move"); }
       }
class Dog extends Animal {
        void move()
        {  super.move();                          // invokes the super class method
          System.out.println("Dogs can walk and run");
        }  }
public class Main {
public static void main(String args[]){
Animal b = new Dog();              // Animal reference but Dog object
b.move();}}                        //Runs the method in Dog class
run:
Animals can move
Dogs can walk and run
```

مثال: عن استخدام الكلمة super لاستدعاء تابع باني فارغ.

```
class AA {                                  // Super class
   public int x;
   public int y;
    // AA تعريف الباني الافتراضي للصف
   // و بما أنه لا يوجد غيره في الصف AA. سيتم تنفيذه بشكل تلقائي في أي صف يرث منه
    AA() {
      x = 5;
      y = 10;   } }
class BB extends  AA  {              // AA الصف BB يرث الصف
    // عند استدعاء الباني الافتراضي للصف BB, أي عند إنشاء كائن منه
   // سيتم استدعاء الصف الافتراضي الموجود في الصف AA حتى و إن لم نقم باستدعائه
 BB() {                  // AA عند استخدام هذا الباني لإنشاء كائن من الصف BB سيتم استدعاء باني الصف
      super();
 }  }                    // يمكن أن لا نكتب هذا الباني لأنه سيكون افتراضي وسينفذ البرنامج في الحالتين
public class Main {
public static void main(String[] args) {
  BB  Ob=new BB();
      System.out.println("x: " + Ob.x);
      System.out.println("y: " + Ob.y);}}
run:
x: 5
y: 10
```

مثال: عن استخدام الكلمة super لاستدعاء تابع باني يحوي بارامترات.

```
class AA {
    int x;
    int y;
 AA(int x,int y)
       {    System.out.println(x+y);   }
    }
class BB extends AA  {
   BB()
       {    super(10,20);    }
    }
public class Main {
public static void main(String[] args) {
 BB  Ob=new BB();}}
run:
30
```

مثال آخر:

```
class AA {
    int x;
    int y;
  AA(int x,int y)
       {
          this.x=x;
          this.y=y;
          System.out.println(x+y);
       }
 }
class BB extends AA  {
   BB()
      {   super(10,20);   }
 }
public class Main {
public static void main(String[] args) {
 BB  Ob=new BB();
 System.out.println("x: " + Ob.x);
 System.out.println("y: " + Ob.y);}}
run:
30
x: 10
y: 20
```

الكلمة this في جافا

تستخدم للإشارة إلى الـ Global Variables، و تستخدم أيضاً للإشارة إلى الكائن الحالي.
سنستخدمها للتفرقة بين المتغيرات التي تم تعريفها بداخل التوابع **Local Variables** و بين المتغيرات التي تم تعريفها بداخل الصف و خارج التوابع **Global Variables** .
عند استخدام الباني (constructor) يوجد بداخله متغيرات بنفس أسماء المتغيرات الموجودة في نفس الصف، لابد أن نستخدم كلمة this تعبيراً عن أن القيمة التي يحملها التابع هي نفسها القيمة في داخل الصف.

مثال:

في هذا المثال يوجد لدينا صف يحتوي على المتغير x والمتغير y ولدينا تابع لديه متغيرين كبارمترات بنفس اسم المتغيرات في الصف، في هذه الحالة سوف نقابل مشكلة وهي أن المفسر لن يستطيع التمييز بين ال x و y الخاص بالصف وبين ال x و y الخاص بالتابع.

```
class A {
      int  x,y;
   void setvalue(int  x,int  y)
   { x=x;
     y=y; }   }
```

لحل المشكلة نقوم باستخدام الكلمة المحجوزة this للإشارة إلى متغيرات الصف وهنا سيستطيع المفسر ان يميز بين متغيرات الصف ومتغيرات التابع ، نعدل الكود بحيث يتم تخزين قيمة ال x وقيمة ال y التي تم تمريرها إلى التابع setvalue إلى المتغيرات x و y الخاصة بمتغيرات الصف كما يلي:

```
class A {
   int  x,y;
void setvalue(int x ,int y)
 { this.x=x;
  this.y=y; } }
```

إذا كانت متغيرات التابع setvlaue تختلف عن أسماء متغيرات الصف، لا نحتاج إلى الكلمة المحجوزة this.

مثال:

اكتب برنامجا يحتوي على صف اسمه (**PersonThis**) ومتغير باسم age ضمن الصف، والمطلوب:

- إنشاء باني أول فارغ.
- وباني ثاني نصرح بدخاله عن متغير بنفس اسم متغير داخل الصف.
- استخدام كلمة this مع متغير الباني للدلالة على أنه يحمل نفس قيمة المتغير داخل الصف.
- إنشاء كائن من الصف واسناد قيمة له، ثم طباعة خاصية هذا الكائن.

الحل:

```
class PersonThis {
        String name;
   PersonThis ()
        {    }
   PersonThis (String name)
        { this.name=name;
          System.out.println("Name: "+name);
        }
 }
```

```
public class Main {
public static void main(String[] args) {
      PersonThis p1 = new PersonThis();
      p1.name = "Ahmad ";
      System.out.println("Name: " +p1.name);
      PersonThis p2 = new PersonThis("Bana");
}} run:
  Name:  Ahmad
  Name: Bana
```

تعدد الأشكال (Polymorphism):

تعدد الأشكال من مفاهيم OOP بحيث يمكن عمل الحدث المفرد بعدة طرق، لغة الجافا توفر طريقتين لتنفيذ هذا المفهوم:

- Method Overloading:

عبارة عن وجود أكثر من تابع في نفس الصف ولهم نفس الاسم، لكن مع اختلاف البارمترات اما بالنوع أو بالعدد.

مثال 1:

```
class overload1{
   void test()
                { System.out.println(" no parameters");  }
   void test(int a)
                {  System.out.println(" power a is :"+ a*a);   }
   void test(int a , int b)
                {  System.out.println(" max a and b is :" +Math.max(a,b));  }
  double test( double b)
                {  System.out.println(" double a "+b);
                   return 3*b;        }
}
public class Main {
public static void main(String[] args) {
overload1 ob=new overload1();
double r ;
ob.test();
ob.test(25);
ob.test(5, 10);
r=ob.test(2.5);
System.out.println("Results of ob (2.5)= "+r); } }
run:
 no parameters
 power a is :625
 max a and b is :10
 double a 2.5
Results of ob (2.5)= 7.5
```

مثال 2: (Overloading Constructors)

يمكن إنشاء أكثر من تابع باني بنفس الصف.

```
class box
{
        double  width , height , depth;
    box()
        {  width=height=depth=2;    }
    box(double width, double h, double d)
        {
         this.width=width;
         height=h;
        depth=d;
        }
   box(double len)
        { width=height=depth=len;  }
   double volume()
        { return width*height*depth;}
}
public class Main {
public static void main(String[] args) {
   box ob1=new box();
   box ob2=new box(5,10,15);
   box ob3=new box(5);
 double vol;
 vol=ob1.volume();
 System.out.println(" volume of my box1 is "+vol );
 vol=ob2.volume();
 System.out.println(" volume of my box2 is "+vol );
 vol=ob3.volume();
 System.out.println(" volume of my box3 is "+vol );
}}
run:
 volume of my box1 is 8.0
 volume of my box2 is 750.0
 volume of my box3 is 125.0
```

- Method Overriding:

عندما يكون التابع في الصف الفرعي بنفس الاسم في الصف العام وبنفس البارامترات فتكون لدينا عملية Method Overriding. عندما يتم استدعاء التابع في الصف الفرعي، فيتم الاستدعاء للتابع الموجود في الصف الفرعي وليس الموجود في الصف العام. أي أن التابع الموجود في الصف العام سيتم اخفائه، إلا إذا أشرنا اليه باستخدام الكلمة المحجوزة **super**.

مثال 1:

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

مثال 2:

```
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 bb extends AA
 {      int k;
        bb(int a, int b, int c )
         { super(a, b);
           k=c;     }
void show()
        { super.show();
          System.out.println(" k is "+k);}
}
public class Main {
public static void main (String args[]){
bb ob=new bb(5,10,15);
ob.show();}}
run:
i and j :5 10
 k is 15
```

الكبسلة Encapsulation:

تعني أن تضع الكود المكتوب داخل كتلة واحدة بحيث يمكن الوصول إليه، ويمكن تحديد درجات الوصول إلى البيانات(المتغيرات والتوابع) بالشكل التالي :

١. النوع **private** عند تعريف المتغير أو التابع من نوع private هذا يعني أن الوصول متاح لهذا المتغير أو التابع من **نفس الصف فقط.**

٢. النوع **protected** هذا يعني أن الوصول متاح **من نفس الصف أو من صف آخر** معرف معه في نفس الحزمة أو **نفس البرنامج.**

٣. النوع **public** يمكن الوصول إليه من نفس الصف أو من صف آخر من داخل أو خارج الحزمة.

ملاحظة: إذا لم تضع أي كلمة من الـ Access Modifiers عند تعريف الصف أو التابع أو المتغير سيتم وضع Modifier افتراضي عنك يسمى package private. و هذا يعني أنه يمكن الوصول إليه فقط من الصفوف الموجودة في نفس الـ package .

مثال:

```
class Encap
 {
     private  int x;
     protected int y;
     public int z;
 }
public class Main {
public static void main(String[] args) {
   Encap  A = new Encap();
   //A.x=33;                                  // private    لا ينفذ لأنه
   A.y=35;
   A.z=255;
    System.out.println(A.y+ "  " + A.z  );
 }      }
run:
35   255
```

تمارين الفصل الخامس

التمرين الأول:

اكتب برنامجا يقوم بإنشاء صف اسمه student وعرف بداخله أربع خصائص وتابع **print** يقوم بطباعة هذه الخصائص. والمطلوب:

- إنشاء كائن من الصف student في الـ main اسمه st1.
- تحديد خصائص لهذا كائن.
- طباعة خصائص هذا الكائن باستدعاء التابع **print** في البرنامج الرئيسي main.

```
class Student                                   //انشاء الصف Student
{
        String name;
         int No;
         String Dep;
         double avg;
 void print(String p1,int p2,String p3,double p4)
   {
        System.out.println("Student name is :" + p1 );
        System.out.println("Student No is :" + p2 );
        System.out.println("Student department is :" + p3 );
        System.out.println("Student average is :" + p4 );
} }
public class Main {
 public static void main(String args[]) {
 Student st1 = new Student();                    //اشتقاق كان من الفئة Student
 st1.name="أ حمد ";
 st1.No=2020;
 st1.Dep="معلوماتية";
 st1.avg=85;
  st1.print(st1.name,st1.No,st1.Dep,st1.avg);    // استدعاء تابع الطباعة
 } }
 run:
Student name is :أ حمد
Student No is :2020
Student department is :معلوماتية
Student average is :85.0
```

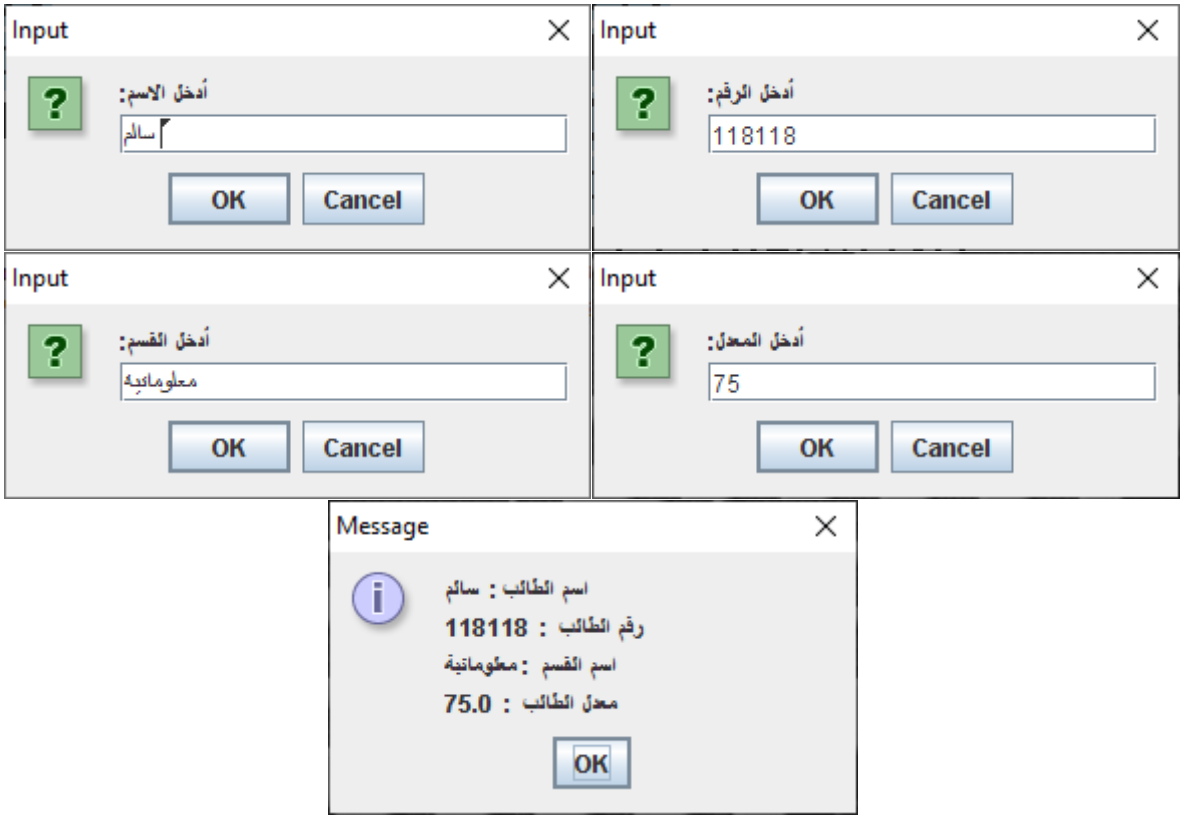
التمرين الثاني:

نفس المثال السابق إدخال الخصائص من قبل المستخدم (GUI).

```
import javax.swing.*;
class Student //Student انشاء الفئة
{
        String name;
```

```
        int No;
        String Dep;
        double avg;
 void print(String p1,int p2,String p3,double p4)
 {
        JOptionPane.showMessageDialog (null,"اسم الطالب :  " + p1
      +"\nرقم الطالب :  " + p2 +"\nاسم القسم : " + p3+"\nمعدل الطالب :  " + p4 );
}}
public class Main {
 public static void main(String args[]) {
        Student st1 = new Student();                    //Student  اشتقاق كائن من الفئة
        String Str;
        st1.name=JOptionPane.showInputDialog(null," أدخل الاسم:");
        Str=JOptionPane.showInputDialog(null,"أدخل الرقم:");
        st1.No=Integer.parseInt( Str);
        st1.Dep=JOptionPane.showInputDialog(null,"أدخل القسم:");
        Str=JOptionPane.showInputDialog(null,"أدخل المعدل:");
        st1.avg=Double.parseDouble( Str);

   st1.print(st1.name,st1.No,st1.Dep,st1.avg);          // استدعاء دالة الطباعة
}
}
```



التمرين الثالث:

ما خرج هذا البرنامج مع الشرح.

```
class Foo {
        int x = 0;                          // Instance variable
        int y = 5;
        Foo() {         }                   // Constructor
        void p()
{
        int  x = 1;                         // Local variable
        System.out.println("x = "+x);
        System.out.println("y = "+y);
} }
public class Main {
public static void main(String[] args) {
Foo p1=new Foo();
p1.p();}}
run:
x = 1
y = 5
```

التمرين الرابع:

ما خرج هذا البرنامج مع الشرح.

```
class Overloading {
    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 class Main {
public static void main(String[] args) {
  Overloading ob=new Overloading();
  ob.sum(10, 20);
  ob.sum(20, 30, 40);}}
run:
a+b = 30
a+b+c =  90
```

التمرين الخامس:

اكتب برنامجا يقوم بإنشاء صف اسمه Student وعرف بداخله ثلاث خصائص وهي رقم الطالب واسمه ودرجته، وتابع باني يقوم بإسناد قيم لهذه الخصائص. والمطلوب:

- إنشاء **مصفوفة كائنات** من الصف Student في الـ main اسمها arr.(تحوي خمس كائنات)
- تحديد خصائص لهذه الكائنات.
- طباعة خصائص هذا الكائنات في البرنامج الرئيسي main.

```
class Student
{
   public int No;
   public String name;
   public int mark;
   Student(int No, String name,int mark)
   {
      this.No = No;
      this.name = name;
      this.mark = mark;
   }
}
                // Elements of array are objects of a class Student.
public class Main {
    public static void main(String[] args) {
       // declares an Array of integers.
      Student[] arr;
                // allocating(تخصيص) memory for 5 objects of type Student.
      arr = new Student[5];
      arr[0] = new Student(1,"Ali",77);
      arr[1] = new Student(2,"Hassan",65);
      arr[2] = new Student(3,"Ahmad",80);
      arr[3] = new Student(4,"Bana",42);
      arr[4] = new Student(5,"Samir",45);
      for (int i = 0; i < arr.length; i++)
         System.out.println("Element at " + i + " :  " +
                  arr[i].No +"\t "+ arr[i].name+"\t "+arr[i].mark);}}
run
Element  at  0 :   1       Ali          77
Element  at  1 :   2       Hassan       65
Element  at  2 :   3       Ahmad        80
Element  at  3 :   4       Bana         42
Element  at  4 :   5       Samir        45
```