Inheritance: الوراثة

Reusability--building new components by utilizing existing components- is yet another important aspect of OO paradigm. It is always good "productive" if we are able to reuse something that is already exists rather than creating the same all over again. This is achieving by creating new classes, reusing the properties of existing classes.

This mechanism of deriving a new class from existing/old class is called "*inheritance*". The old class is known as "base" class, "super" class or "parent" class"; and the new class is known as "sub" class, "derived" class, or "child"class.

تعد قابلية إعادة الاستخدام - بناء مكونات جديدة من خلال استخدام المكونات الحالية - جانبًا مهمًا آخر من جوانب نموذج OO. إنه "منتج" جيد دائمًا إذا كنا قادرين على إعادة استخدام شيء موجود بالفعل بدلاً من إنشاء نفس الشيء مرة أخرى ، ويتم تحقيق ذلك من خلال إنشاء فئات جديدة ، وإعادة استخدام خصائص الفئات الموجودة.

تسمى آلية اشتقاق فئة جديدة من فئة موجودة / قديمة "الميراث". تُعرف الفئة القديمة بالفئة "الأساسية" أو الفئة "الفائقة" أو الفئة "الأصل" ؛ وتعرف الفئة الجديدة بالفئة "الفرعية" أو الفئة "المشتقة" أو الفئة "الفرعية".

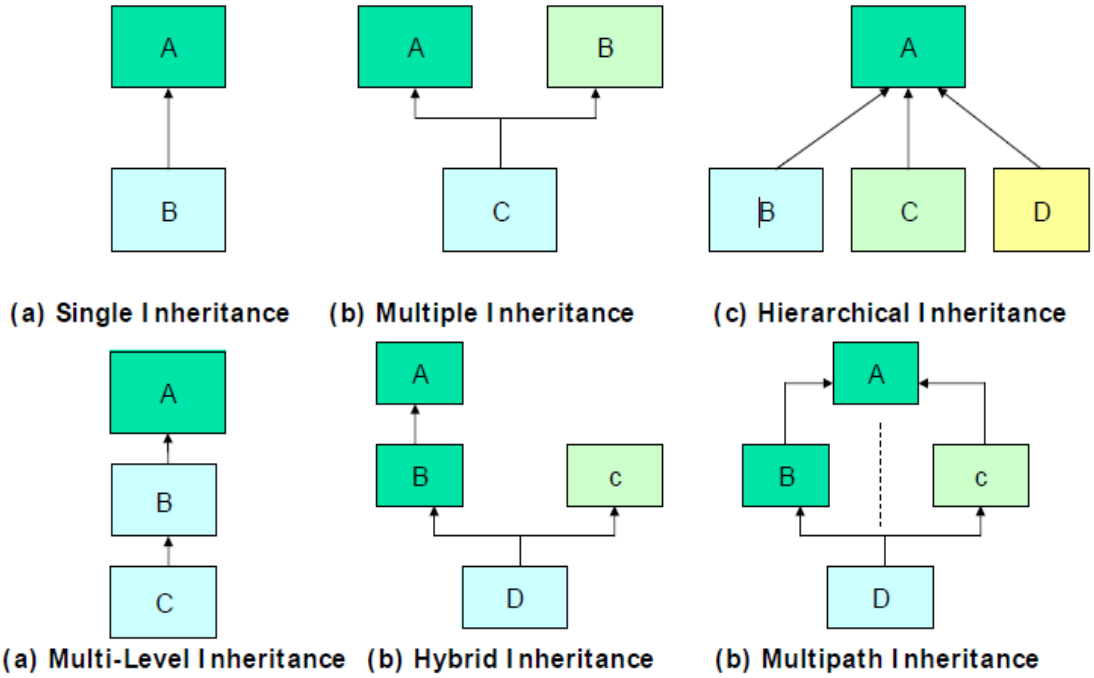
The inheritance allows subclasses to inherit all properties (variables and methods) of their parent classes. The different forms of inheritance are:

يسمح الوراثة للفئات الفرعية أن ترث جميع الخصائص (المتغيرات والطرق) لفئاتهم الأصلية. أشكال الميراث المختلفة هي:

- Single inheritance (only one super class)
- Multiple inheritance (several super classes)
- Hierarchical inheritance (one super class, many sub classes)
- Multi-Level inheritance (derived from a derived class)
- Hybrid inheritance (more than two types)
- Multi-path inheritance (inheritance of some properties from two sources).

- الميراث الفردي (فئة ممتازة واحدة فقط)
- الميراث المتعدد (عدة فئات ممتازة)
- الميراث الهرمي (فئة عليا واحدة ، العديد من الفئات الفرعية)
- الميراث متعدد المستويات (مشتق من فئة مشتقة)
- وراثة هجينة (أكثر من نوعين)
- الوراثة متعددة المسارات (وراثة بعض الخصائص من مصدرين).

Forms of Inheritance



Inheritance Basics:

Java supports inheritance by allowing one class to incorporate another class into its declaration. This is done by using the **extends** keyword. Thus, the subclass adds to (extends) the superclass.

أساسيات الميراث:

تدعم Java الوراثة من خلال السماح لفئة واحدة بدمج فئة أخرى في إعلانها. يتم ذلك باستخدام الكلمة الأساسية الموسعة. وهكذا ، فإن الفئة الفرعية تضيف (تمتد) إلى الطبقة العليا.

```
class SubClassName extends SuperClassName
{
fields declaration;إعلان الحقول
methods declaration;إعلان الطرق
}
```

The keyword “extends” signifies that the properties of super class are extended to the subclass. That means, subclass contains its own members as well of those of the super class. This kind of situation occurs when we want to enhance properties of existing class without actually modifying it.

تشير الكلمة الأساسية "تمتد" إلى أن خصائص الفئة الممتازة تمتد إلى الفئة الفرعية. هذا يعني أن الفئة الفرعية تحتوي على أعضائها وكذلك أعضاء الطبقة العليا. يحدث هذا النوع من المواقف عندما نريد تحسين خصائص فئة موجودة دون تعديلها فعليًا.

The following program creates a superclass called **TwoDShape**, which stores the width and height of a two-dimensional object, and a subclass called **Triangle**. Notice how the keyword **extends** is used to create a subclass.

يقوم البرنامج التالي بإنشاء فئة فائقة تسمى TwoDShape ، والتي تخزن عرض وارتفاع كائن ثنائي الأبعاد ، وفئة فرعية تسمى Triangle. لاحظ كيف يتم استخدام الكلمة الأساسية الممتدة لإنشاء فئة فرعية.

```
// A simple class hierarchy.تسلسل هرمي لفئة بسيطة // .
// A class for two-dimensional objects.فئة للأشياء ثنائية الأبعاد // .
class TwoDShape {
  double width;
  double height;
  void showDim() {
    System.out.println("Width and height are " +
   width + " and " + height);
    }
 }
// A subclass of TwoDShape for triangles.فئة فرعية من شكلين // D للمثلثات.
class Triangle extends TwoDShape {
  String style;
  double area() {
  return width * height / 2;
  }
  void showStyle() {
     System.out.println("Triangle is " + style); }
}
class Shapes {
public static void main(String args[]) {
  Triangle t1 = new Triangle();
  Triangle t2 = new Triangle();
  t1.width = 4.0;
  t1.height = 4.0;
  t1.style = "isosceles";
  t2.width = 8.0;
  t2.height = 12.0;
  t2.style = "right";
  System.out.println("Info for t1: ");
  t1.showStyle();
  t1.showDim();
  System.out.println("Area is " + t1.area());
```

```
  System.out.println();
  System.out.println("Info for t2: ");
  t2.showStyle();
  t2.showDim();
  System.out.println("Area is " + t2.area());
  }
}
```

The output from this program is shown here:

```
Info for t1:
Triangle is isosceles
Width and height are 4.0 and 4.0
Area is 8.0
Info for t2:
Triangle is right
Width and height are 8.0 and 12.0
Area is 48.0
```

Here, **TwoDShape** defines the attributes of a “generic” two-dimensional shape, such as a square, rectangle, triangle, and so on. The **Triangle** class creates a specific type of **TwoDShape**, in this case, a triangle. The **Triangle** class includes all of **TwoDObject** and adds the field **style**, All members of **Triangle** are available to **Triangle** objects, even those inherited from **TwoDShape**.

هنا ، يعرّف TwoDShape سمات شكل ثنائي الأبعاد "عام" ، مثل a
مربع ، مستطيل ، مثلث ، وما إلى ذلك. تنشئ فئة Triangle نوعًا معينًا من TwoDShape ، في هذه الحالة ، مثلث.
تتضمن فئة Triangle كل TwoDObject وتضيف نمط الحقل ، ويتوفر كل أعضاء Triangle لكائنات Triangle
، حتى تلك الموروثة من TwoDShape.

The method **area( )**, and the method **showStyle( )**. A description of the type of triangle is stored in **style**, **area( )** computes and returns the area of the triangle, and **showStyle( )** displays the triangle style.

منطقة الطريقة () ، والطريقة showStyle (). يتم تخزين وصف لنوع المثلث في النمط ، وتحسب المنطقة ()
وتعيد مساحة المثلث ، ويعرض showStyle () نمط المثلث.

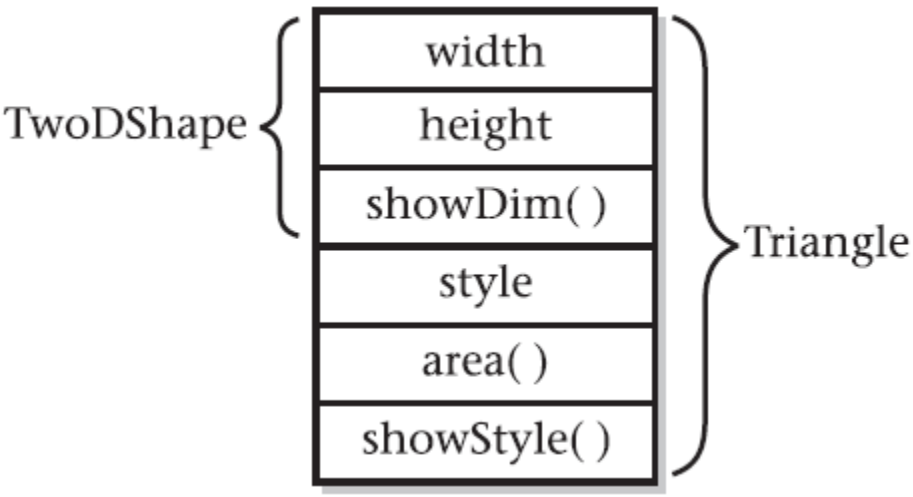
Because **Triangle** includes all of the members of its superclass, **TwoDShape**, it can access **width** and **height** inside **area( )**. Also, inside **main( )**, objects **t1** and **t2** can refer to **width** and **height** directly, as if they were part of **Triangle**. Even though **TwoDShape** is a superclass for **Triangle**, it is also a completely independent, stand-alone class. Being a superclass for a subclass does not mean that the superclass cannot be used by itself. For example, the following is perfectly valid.

نظرًا لأن Triangle يشمل جميع أعضاء فئته الفائقة ، TwoDShape ، فإنه يمكنه الوصول إلى العرض والارتفاع داخل المنطقة ().
أيضًا ، داخل main () ، يمكن أن تشير الكائنات t1 و t2 إلى العرض والارتفاع مباشرةً ، كما لو كانت جزءًا من Triangle. على الرغم من
أن TwoDShape هي فئة ممتازة لـ Triangle ، فهي أيضًا فئة مستقلة تمامًا ومستقلة. كونك فئة فائقة لفئة فرعية لا يعني أنه لا يمكن
استخدام الطبقة الفائقة بمفردها. على سبيل المثال ، ما يلي صالح تمامًا.

```
TwoDShape shape = new TwoDShape();
shape.width = 10;
shape.height = 20;
shape.showDim();
```

Of course, an object of **TwoDShape** has no knowledge of or access to any subclasses of **TwoDShape**. The fallowing Figure depicts conceptually how **TwoDShape** is incorporated into **Triangle**.

بالطبع ، كائن TwoDShape ليس لديه معرفة أو الوصول إلى أي فئات فرعية من TwoDShape. يصور الشكل المريح من الناحية المفاهيمية كيفية دمج TwoDShape في المثلث.



Member Access and Inheritance:

As you learned, often an instance variable of a class will be declared **private** to prevent its unauthorized use or tampering. Inheriting a class *does not* overrule the **private** access restriction. Thus, even though a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared **private**. For example, if, as shown here, **width** and **height** are made private in **TwoDShape**, then **Triangle** will not be able to access them.

وصول الأعضاء والميراث:

كما تعلمت ، غالبًا ما يتم الإعلان عن متغير مثيل لفئة ما خاصة لمنع الاستخدام غير المصرح به أو العبث به. وراثة فئة لا تلغي قيود الوصول الخاص. وبالتالي ، على الرغم من أن الفئة الفرعية تشمل جميع أعضاء الطبقة العليا الخاصة بها ، إلا أنها لا تستطيع الوصول إلى أعضاء الطبقة العليا الذين تم إعلانهم كخاص. على سبيل المثال ، إذا تم جعل العرض والارتفاع خاصين في TwoDShape ، كما هو موضح هنا ، فلن يتمكن Triangle من الوصول إليهما.

```
// Private members are not inherited.لا يتم توريث الأعضاء الخاصين .
// This example will not compile.لن يتم ترجمة هذا المثال .
// A class for two-dimensional objects.فئة للأشياء ثنائية الأبعاد // .
```

```
class TwoDShape {
 private double width; // these are
 private double height; // now private
 void showDim() {
  System.out.println("Width and height are " + width + " and " +
height);
}
}
// A subclass of TwoDShape for triangles.فئة فرعية من // TwoDShape للمثلثات.
class Triangle extends TwoDShape {
String style;
double area() {
return width * height / 2; // Error! can't access
}
void showStyle() {
System.out.println("Triangle is " + style);
 }
}
```

The **Triangle** class will not compile because the reference to **width** and **height** inside the **area( )** method causes an access violation. Since **width** and **height** are declared **private**, they are accessible only by other members of their own class. Subclasses have no access to them.

لن يتم ترجمة فئة المثلث لأن المرجع إلى العرض والارتفاع داخل طريقة area () يتسبب في انتهاك وصول. نظرًا لأن العرض والارتفاع يتم اعتبارهما خاصين ، فلا يمكن الوصول إليهما إلا من قبل أعضاء آخرين من فئتهم الخاصة. لا يمكن الوصول إلى الفئات الفرعية.

Remember that a class member that has been declared **private** will remain private to its class. It is not accessible by any code outside its class, including subclasses.

At first, you might think that the fact that subclasses do not have access to the private members of superclasses is a serious restriction that would prevent the use of private members in many situations. However this is not true. Java programmers typically use accessor methods to provide access to the private methods of a class. Here is a rewrite of the **TwoDShape** and **Triangle** classes that uses methods to access the private instance variables **width** and **height**.

تذكر أن عضو الفصل الذي تم إعلانه خاصًا سيظل خاصًا بفئته. لا يمكن الوصول إليه من قبل أي رمز خارج فئته ، بما في ذلك الفئات الفرعية.

في البداية ، قد تعتقد أن حقيقة أن الفئات الفرعية لا يمكنها الوصول إلى الأعضاء الخاصين من الطبقات الفائقة هو قيد خطير من شأنه أن يمنع استخدام الأعضاء الخاصين في العديد من المواقف. مهما يكن ... هذه ليست الحقيقة. يستخدم مبرمجو Java عادةً طرق الوصول لتوفير الوصول إلى الطرق الخاصة للفصل. فيما يلي إعادة كتابة لفئات TwoDShape و Triangle التي تستخدم طرقًا للوصول إلى عرض متغيرات المثيل الخاصة وارتفاعها.

```
// Use accessor methods to set and get private members.
// استخدم طرق الوصول لتعيين الأعضاء الخاصين والحصول عليهم.
// A class for two-dimensional objects.
// فئة للأشياء ثنائية الأبعاد.
class TwoDShape {
 private double width; // these are
 private double height; // now private
// Accessor methods for width and height.
 double getWidth() { return width; }
 double getHeight() { return height; }
 void setWidth(double w) { width = w; }
 void setHeight(double h) { height = h; }
 void showDim() {
   System.out.println("Width and height are " +
width + " and " + height);
 }
}
// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
String style;
double area() {
return getWidth() * getHeight() / 2;
}
void showStyle() {
System.out.println("Triangle is " + style);
}
}
class Shapes2 {
public static void main(String args[]) {
 Triangle t1 = new Triangle();
 Triangle t2 = new Triangle();
 t1.setWidth(4.0);
 t1.setHeight(4.0);
 t1.style = "isosceles";
 t2.setWidth(8.0);
 t2.setHeight(12.0);
 t2.style = "right";
 System.out.println("Info for t1: ");
 t1.showStyle();
 t1.showDim();
 System.out.println("Area is " + t1.area());
 System.out.println();
 System.out.println("Info for t2: ");
 t2.showStyle();
 t2.showDim();
 System.out.println("Area is " + t2.area());
 }
}
```