Constructors and Inheritance المنشئ و الوراثة

In a hierarchy, it is possible for both super classes and subclasses to have their own constructors. This raises an important question: what constructor is responsible for building an object of the subclass—the one in the superclass, the one in the subclass, or both? The answer is this: the constructor for the superclass constructs the superclass portion of the object, and the constructor for the subclass constructs the subclass part. This makes sense because the superclass has no knowledge of or access to any element in a subclass. Thus, their construction must be separate.

The preceding examples have relied upon the default constructors created automatically by Java, so this was not an issue. However, in practice, most classes will have explicit constructors.

في التسلسل الهرمي ، من الممكن لكل من الطبقات الفائقة والفئات الفرعية أن يكون لها صانعوها الخاصون. يثير هذا سؤالًا مهمًا: ما المُنشئ المسؤول عن بناء كائن من الفئة الفرعية - الكائن الموجود في الطبقة العليا ، أو الموجود في الفئة الفرعية ، أو كليهما؟ الإجابة هي كالتالي: يُنشئ مُنشئ الطبقة الفائقة الجزء الخاص بالفئة الفائقة من الكائن ، ويُنشئ مُنشئ الفئة الفرعية جزء الفئة الفرعية. هذا منطقي لأن الطبقة العليا ليس لديها معرفة أو الوصول إلى أي عنصر في فئة فرعية. وبالتالي ، يجب أن يكون بنائها منفصلًا.

اعتمدت الأمثلة السابقة على المُنشئ الافتراضي الذي تم إنشاؤه تلقائيًا بواسطة Java ، لذلك لم تكن هذه مشكلة. ومع ذلك ، من الناحية العملية ، سيكون لمعظم الفئات مُنشئون صريحون.

Here you will see how to handle this situation.هنا سترى كيفية التعامل مع هذا الموقف .

When only the subclass defines a constructor, the process is straightforward: simply construct the subclass object. The superclass portion of the object is constructed automatically using its default constructor. For example, here is a reworked version of **Triangle** that defines a constructor. It also makes **style** private since it is now set by the constructor.

عندما تحدد الفئة الفرعية المُنشئ فقط ، تكون العملية مباشرة: ببساطة
بناء كائن فئة فرعية. يتم إنشاء جزء الطبقة الفائقة من الكائن تلقائيًا باستخدام المُنشئ الافتراضي الخاص به. على سبيل المثال ، هنا نسخة مُعاد صياغتها من Triangle تحدد المُنشئ. كما أنه يجعل النمط خاصًا لأنه تم تعيينه الآن بواسطة المنشئ.

```
// Add a constructor to Triangle.
// A class for two-dimensional objects.
```

// إضافة مُنشئ إلى المثلث.
// 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; }
```

// طرق الوصول للعرض والارتفاع. .

```
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 {
private String style;
// Constructor
Triangle(String s, double w, double h) { //Initialize TwoDShape
portion of object.تهيئة قسمين من الشكل // D من الكائن.
setWidth(w);
setHeight(h);
style = s;
}
double area() {
return getWidth() * getHeight() / 2;
}
void showStyle() {
System.out.println("Triangle is " + style);
}
}
class Shapes3 {
public static void main(String args[]) {
Triangle t1 = new Triangle("isosceles", 4.0, 4.0);
Triangle t2 = new Triangle("right", 8.0, 12.0);
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());
```

Here, **Triangle**'s constructor initializes the members of **TwoDClass** that it inherits along with its own **style** field.

When both the superclass and the subclass define constructors, the process is a bit more complicated because both the superclass and subclass constructors must be executed.

هنا ، تقوم مُنشئ Triangle بتهيئة أعضاء TwoDClass التي ترثها جنبًا إلى جنب مع حقل النمط الخاص بها.
عندما تحدد كل من الطبقة الفائقة والفئة الفرعية المُنشئين ، تكون العملية أكثر تعقيدًا بعض الشيء لأنه يجب تنفيذ كل من مُنشِئات الطبقة الفائقة والفئة الفرعية.

Using super to Call Superclass Constructors:

A subclass can call a constructor defined by its superclass by use of the following form of **super**:

super(*parameter-list*);

Here, *parameter-list* specifies any parameters needed by the constructor in the superclass.

super( ) must always be the first statement executed inside a subclass constructor.

To see how **super( )** is used, consider the version of **TwoDShape** in the following program. It defines a constructor that initializes **width** and **height**.

استخدام Super لاستدعاء Superclass Constructors:

يمكن للفئة الفرعية استدعاء مُنشئ محدد بواسطة فئتها الفائقة باستخدام الشكل التالي من super:

سوبر (قائمة المعلمات) ؛

هنا ، تحدد قائمة المعلمات أي معلمات يحتاجها المُنشئ في الطبقة الفائقة.

يجب أن تكون super () دائمًا أول تعليمة يتم تنفيذها داخل مُنشئ فئة فرعية.

لمعرفة كيفية استخدام super () ، ضع في اعتبارك إصدار TwoDShape في البرنامج التالي. يحدد المُنشئ الذي يقوم بتهيئة العرض والارتفاع.

```
// Add constructors to TwoDShape.
class TwoDShape {
private double width;
private double height;
// Parameterized constructor.
TwoDShape(double w, double h) {
width = w;
height = h;
}
// 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 {
private String style;
Triangle(String s, double w, double h) {
super(w, h); // call superclass constructor
style = s;
}
double area() {
return getWidth() * getHeight() / 2;
}
```

```
void showStyle() {
System.out.println("Triangle is " + style);
}
}
class Shapes4 {
public static void main(String args[]) {
Triangle t1 = new Triangle("isosceles", 4.0, 4.0);
Triangle t2 = new Triangle("right", 8.0, 12.0);
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());
}
}
```

Here, **Triangle( )** calls **super( )** with the parameters **w** and **h**. This causes the **TwoDShape( )** constructor to be called, which initializes **width** and **height** using these values. **Triangle** no longer initializes these values itself. It need only initialize the value unique to it: **style**. This leaves **TwoDShape** free to construct its subobject in any manner that it so chooses. Furthermore, **TwoDShape** can add functionality about which existing subclasses have no knowledge, thus preventing existing code from breaking.

هنا ، يستدعي super () Triangle () مع المعلمات w و h. يؤدي هذا إلى استدعاء مُنشئ TwoDShape () ، والذي يقوم بتهيئة العرض والارتفاع باستخدام هذه القيم. لم يعد المثلث يقوم بتهيئة هذه القيم بنفسه. يحتاج فقط إلى تهيئة القيمة الفريدة له: النمط. هذا يترك TwoDShape حراً لبناء كائن فرعي بأي طريقة يختارها. علاوة على ذلك ، يمكن لـ TwoDShape إضافة وظائف لا تعرف الفئات الفرعية الموجودة عنها ، وبالتالي منع الكود الحالي من الانهيار.

Any form of constructor defined by the superclass can be called by **super( )**. The constructor executed will be the one that matches the arguments. For example, here are expanded versions of both **TwoDShape** and **Triangle** that include default constructors and constructors that take one argument.

يمكن استدعاء أي شكل من أشكال المُنشئ التي تحددها الطبقة الفائقة بواسطة super (). سيكون المُنشئ الذي يتم تنفيذه هو الذي يطابق الوسيطات. على سبيل المثال ، فيما يلي إصدارات موسعة من كل من TwoDShape و Triangle التي تتضمن المنشئات والمُنشئين الافتراضيين التي تأخذ وسيطة واحدة.

```
// Add more constructors to TwoDShape.
class TwoDShape {
private double width;
private double height;
// A default constructor.
```

```
TwoDShape() {
width = height = 0.0;
}
// Parameterized constructor.
TwoDShape(double w, double h) {
width = w;
height = h;
}
// Construct object with equal width and height.
TwoDShape(double x) {
width = height = x;
}
// 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 {
private String style;
// A default constructor.
Triangle() {
super();
style = "null";
}
// Constructor
Triangle(String s, double w, double h) {
super(w, h); // call superclass constructor
style = s;
}
// Construct an isosceles triangle.
Triangle(double x) {
super(x); // call superclass constructor
style = "isosceles";
}
double area() {
return getWidth() * getHeight() / 2;
}
void showStyle() {
System.out.println("Triangle is " + style);
}
}
class Shapes5 {
public static void main(String args[]) {
Triangle t1 = new Triangle();
Triangle t2 = new Triangle("right", 8.0, 12.0);
```

```
Triangle t3 = new Triangle(4.0);
t1 = t2;
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());
System.out.println();
System.out.println("Info for t3: ");
t3.showStyle();
t3.showDim();
System.out.println("Area is " + t3.area());
System.out.println();
}
}
```

Here is the output from this version.

```
Info for t1:
Triangle is right
Width and height are 8.0 and 12.0
Area is 48.0
Info for t2:
Triangle is right
Width and height are 8.0 and 12.0
Area is 48.0
Info for t3:
Triangle is isosceles
Width and height are 4.0 and 4.0
Area is 8.0
```

Using super to Access Superclass Members:

There is a second form of **super** that it always refers to the superclass of the subclass in which it is used. This usage has the following general form:

super.*member*

Here, *member* can be either a method or an instance variable.

This form of **super** is most applicable to situations in which member names of a subclass hide members by the same name in the superclass. Consider this simple class hierarchy:

استخدام super للوصول إلى أعضاء Superclass:

هناك شكل ثانٍ من الفائق يشير دائمًا إلى الطبقة العليا للفئة الفرعية التي يتم استخدامه فيها. هذا الاستخدام له الشكل العام التالي:

عضو خارق

هنا ، يمكن أن يكون العضو إما طريقة أو متغير حالة.

هذا الشكل من أشكال super هو الأكثر قابلية للتطبيق في المواقف التي تخفي فيها أسماء أعضاء فئة فرعية أعضاء بنفس الاسم في الطبقة العليا. ضع في اعتبارك هذا التسلسل الهرمي للفصل البسيط:

```
// Using super to overcome name hiding.
class A {
int i;
}
// Create a subclass by extending class A.
class B extends A {
int i; // this i hides the i in A
B(int a, int b) {
super.i = a; // i in A
i = b; // i in B
}
void show() {
System.out.println("i in superclass: " + super.i);
System.out.println("i in subclass: " + i);
}
}
class UseSuper {
public static void main(String args[]) {
B subOb = new B(1, 2);
subOb.show();
}
}
```

This program displays the following:

```
i in superclass: 1
i in subclass: 2
```

Although the instance variable **i** in **B** hides the **i** in **A**, **super** allows access to the **i** defined in the superclass. **super** can also be used to call methods that are hidden by a subclass. Here, **super.i** refers to the **i** in **A**.

Note: in multilevel inheritance in JAVA when we need to use three classes (A,B,C). We must put the main method in the Class C to get read from errors.

يعرض هذا البرنامج ما يلي:
أنا في الطبقة العليا: 1
أنا في فئة فرعية: 2
على الرغم من أن متغير المثيل i في B يخفي i في A ، فإن super يسمح بالوصول إلى i المحدد في الطبقة الفائقة.
يمكن أيضًا استخدام super لاستدعاء الطرق المخفية بواسطة فئة فرعية. هنا ، يشير super.i إلى i في A.

ملاحظة: في الوراثة متعددة المستويات في جافا عندما نحتاج إلى استخدام ثلاث فئات (أ ، ب ، ج). يجب أن نضع الطريقة الرئيسية في الفئة C للقراءة من الأخطاء.

```
public class A {
int i;
}
public class B extends A {
            int i; // this i hides the i in A
            B(int x, int y) {
            super.i = x; // i in A
            i = y; // i in B
```

```
		}
		void show() {
		System.out.println("i in Grandclass: " + super.i);
		System.out.println("i in ParentClass: " + i);
		}

}
public class C extends B {
		int i;
		C(int x, int y, int z) {
		super(x,y); // i in B and A
		i = z; // i in C
		}
		void show() {
		super.show();
		System.out.println("i in subclass: " + i);
		}
		public static void main(String[] args) {
			// TODO Auto-generated method stub
			C subOb = new C(1, 2,3);
			subOb.show();
		}

}
```

This program displays the following:

```
i in Grandclass: 1
i in ParentClass: 2
i in subclass: 3
```