An Introduction to Polymorphism in Java:مقدمة عن تعدد الأشكال في جافا :

Method Overridingأسلوب التجاوز

In a class hierarchy, when a method in a subclass has the same return type and signature as a method in its superclass, then the method in the subclass is said to *override* the method in the superclass. When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden. Consider the following:

في التسلسل الهرمي للفئة ، عندما يكون لطريقة ما في فئة فرعية نفس نوع الإرجاع والتوقيع كطريقة في صنفها الفائق ، يُقال إن الطريقة في الفئة الفرعية تتجاوز الطريقة في الفئة الفائقة. عندما يتم استدعاء طريقة متجاوزة من داخل فئة فرعية ، فإنها ستشير دائمًا إلى إصدار تلك الطريقة المحددة بواسطة الفئة الفرعية. سيتم إخفاء إصدار الطريقة المحددة بواسطة الطبقة الفائقة. ضع في اعتبارك ما يلي:

```
// Method overriding.
// أسلوب التجاوز
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}

// display i and j


void show() {
System.out.println("i and j: " + i + " " + j);
}
}
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
// display k – this overrides show() in A
void show() {
System.out.println("k: " + k);
}
}
class Override {
public static void main(String args[]) {
B subOb = new B(1, 2, 3);
subOb.show(); // this calls show() in B
}
}
```

The output produced by this program is shown here:
k: 3
When **show( )** is invoked on an object of type **B**, the version of **show( )** defined within **B** is used. That is, the version of **show( )** inside **B** overrides the version declared in **A**.
If you want to access the superclass version of an overridden method, you can do so by using **super**.

يتم عرض الإخراج الناتج عن هذا البرنامج هنا:
ك: 3
عندما يتم استدعاء show () على كائن من النوع B ، يتم استخدام إصدار show () المحدد داخل B. أي أن إصدار show () داخل B يتجاوز الإصدار المعلن في A.
إذا كنت ترغب في الوصول إلى إصدار superclass لطريقة تم تجاوزها ، فيمكنك القيام بذلك باستخدام
ممتاز.

For example, in this version of **B**, the superclass version of **show( )** is invoked within the subclass' version. This allows all instance variables to be displayed.

على سبيل المثال ، في هذا الإصدار من B ، يتم استدعاء نسخة الفئة الفائقة من show () داخل إصدار الفئة الفرعية. هذا يسمح بعرض جميع متغيرات الحالة.

```
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
super.show(); // this calls A's show()
System.out.println("k: " + k);
 }
}
```

If you substitute this version of **show( )** into the previous program, you will see the following output:

إذا قمت باستبدال هذا الإصدار من show () في البرنامج السابق ، فسترى الناتج التالي:

```
i and j: 1 2
k: 3
```

Here, **super.show( )** calls the superclass version of **show( )**.

Because signatures differ, this **show( )** simply overloads **show( )** in superclass **A**.

هنا ، تستدعي super.show () نسخة superclass من show ().

نظرًا لاختلاف التواقيع ، فإن هذا العرض () يظهر ببساطة التحميل الزائد () في الطبقة الفائقة أ.

Method overriding occurs *only* when the return types and signatures of the two methods are identical. If they are not, then the two methods are simply overloaded. For example, consider this modified version of the preceding example:

يحدث تجاوز الطريقة فقط عندما تكون أنواع الإرجاع والتوقيعات الخاصة بالطريقتين متطابقة. إذا لم تكن كذلك ، فإن الطريقتين محملتان فوق طاقتهما. على سبيل المثال ، ضع في اعتبارك هذه النسخة المعدلة من المثال السابق:

```
/* Methods with differing signatures are
overloaded and not overridden. */
```

/ * الطرق ذات التوقيعات المختلفة محملة بشكل زائد ولا يتم تجاوزها. * /

```
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
// display i and j
void show() {
System.out.println("i and j: " + i + " " + j);
}
}
```

```
// Create a subclass by extending class A.
```

// قم بإنشاء فئة فرعية بتوسيع الفئة A.

```
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c; }
//  overload  show()
void show(String msg) {
System.out.println(msg + k);
}
}
```

```
class Overload {
public static void main(String args[]) {
B subOb = new B(1, 2, 3);
subOb.show("This is k: "); // this calls show() in B
subOb.show(); // this calls show() in A// تظهر هذه المكالمات () في A
}
}
```

The output produced by this program is shown here:

```
This is k: 3
i and j: 1 2
```

The version of **show( )** in **B** takes a string parameter. This makes its signature different from the one in **A**, which takes no parameters. Therefore, no overriding (or name hiding) takes place.

إصدار show () في B يأخذ معلمة سلسلة. هذا يجعل توقيعه مختلفًا عن التوقيع في A ، والذي لا يأخذ أي معلمات. لذلك ، لا يتم التجاوز (أو إخفاء الاسم).

Using Abstract Classes

Sometimes you will want to create a superclass that defines only a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement but does not, itself, provide an implementation of one or more of these methods.

An abstract method is created by specifying the **abstract** type modifier. An abstract method contains no body and is, therefore, not implemented by the superclass. Thus, a subclass must override it—it cannot simply use the version defined in the superclass. To declare an abstract method, use this general form:

استخدام فئات مجردة

في بعض الأحيان ، قد ترغب في إنشاء فئة فائقة تحدد فقط نموذجًا معممًا ستشترك فيه جميع الفئات الفرعية ، تاركًا إياه لكل فئة فرعية لملء التفاصيل. تحدد هذه الفئة طبيعة الطرق التي يجب أن تنفذها الفئات الفرعية ولكنها لا توفر ، في حد ذاتها ، تنفيذًا لواحدة أو أكثر من هذه الطرق.

يتم إنشاء طريقة مجردة عن طريق تحديد معدل النوع المجرد. الطريقة المجردة لا تحتوي على أي جسم ، وبالتالي لا يتم تنفيذها من قبل الطبقة العليا. وبالتالي ، يجب أن تحل فئة فرعية محلها - فلا يمكنها ببساطة استخدام الإصدار المحدد في الطبقة العليا. للإعلان عن طريقة مجردة ، استخدم هذا النموذج العام:

abstract *type name(parameter-list)*;

As you can see, no method body is present. The **abstract** modifier can be used only on normal methods. It cannot be applied to **static** methods or to constructors.

اسم نوع مجردة (قائمة المعلمات) ؛

كما ترى ، لا يوجد جسم طريقة موجود. لا يمكن استخدام المعدل المجرد إلا في الطرق العادية. لا يمكن تطبيقه على الأساليب الثابتة أو المنشئات.

A class that contains one or more abstract methods must also be declared as abstract by preceding its **class** declaration with the **abstract** specifier. Since an abstract class does not define a complete implementation, there can be no objects of an abstract class. Thus, attempting to create an object of an abstract class by using **new** will result in a compile-time error.

يجب أيضًا الإعلان عن الفئة التي تحتوي على واحد أو أكثر من الطرق المجردة على أنها مجردة من خلال تقديم إعلان الفئة الخاص بها باستخدام المحدد المجرد. نظرًا لأن الفصل المجرد لا يحدد التنفيذ الكامل ، فلا يمكن أن يكون هناك كائنات لفئة مجردة. وبالتالي ، فإن محاولة إنشاء كائن من فئة مجردة باستخدام new ستؤدي إلى خطأ في وقت الترجمة.

When a subclass inherits an abstract class, it must implement all of the abstract methods in the superclass. If it doesn't, then the subclass must also be specified as **abstract**. Thus, the **abstract** attribute is inherited until such time as a complete implementation is achieved. Using an abstract class, you can improve the **TwoDShape** class. Since there is no meaningful concept of area for an undefined two-dimensional figure, the following version of the preceding program declares **area( )** as **abstract** inside **TwoDShape**, and **TwoDShape** as **abstract**. This, of course, means that all classes derived from **TwoDShape** must override **area( )**.

عندما ترث فئة فرعية فئة مجردة ، يجب أن تنفذ جميع الطرق المجردة في الطبقة العليا. إذا لم يحدث ذلك ، فيجب أيضًا تحديد الفئة الفرعية على أنها مجردة. وبالتالي ، يتم توريث السمة المجردة حتى يتم تحقيق التنفيذ الكامل. باستخدام فئة مجردة ، يمكنك تحسين فئة TwoDShape. نظرًا لعدم وجود مفهوم ذي معنى للمنطقة لشكل ثنائي الأبعاد غير محدد ، فإن الإصدار التالي من البرنامج السابق يعلن المنطقة () على أنها مجردة داخل TwoDShape ، و TwoDShape على أنها مجردة. هذا ، بالطبع ، يعني أن جميع الفئات المشتقة من TwoDShape يجب أن تتجاوز area ()

```
// Create an abstract class.// إنشاء فئة مجردة.
abstract class TwoDShape {
private double width;
private double height;
private String name;
```

```
// A default constructor.
TwoDShape() {
width = height = 0.0;
name = "null";
}
// Parameterized constructor// منشئ ذو معلمات.

TwoDShape(double w, double h, String n) {
width = w;
height = h;
name = n;
}
// Construct object with equal width and height// قم ببناء كائن بعرض متساوٍ وارتفاع.

TwoDShape(double x, String n) {
width = height = x;
name = n;
}
// Construct an object from an object.بناء كائن من كائن

TwoDShape(TwoDShape ob) {
width = ob.width;
height = ob.height;
name = ob.name;
}
// 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; }
String getName() { return name; }
void showDim() {
System.out.println("Width and height are " +
width + " and " + height);
}
// Now, area() is abstract.
abstract double area();
}
// A subclass of TwoDShape for triangles.
class Triangle extends TwoDShape {
private String style;
// A default constructor.
Triangle() {
super();
style = "null";
}
// Constructor for Triangle.
Triangle(String s, double w, double h) {
super(w, h, "triangle");
```

```
style = s;
}
// Construct an isosceles triangle.
```

```
Triangle(double x) {
super(x, "triangle"); // call superclass constructor
style = "isosceles";
}
// Construct an object from an object.
Triangle(Triangle ob) {
super(ob); // pass object to TwoDShape constructor
style = ob.style;
}
double area() {
return getWidth() * getHeight() / 2;
}
void showStyle() {
System.out.println("Triangle is " + style);
}
}
// A subclass of TwoDShape for rectangles.
class Rectangle extends TwoDShape {
// A default constructor.
Rectangle() {
super();
}
// Constructor for Rectangle.
Rectangle(double w, double h) {
super(w, h, "rectangle"); // call superclass constructor
}
// Construct a square.
Rectangle(double x) {
super(x, "rectangle"); // call superclass constructor
}
// Construct an object from an object.
Rectangle(Rectangle ob) {
super(ob); // pass object to TwoDShape constructor
}
boolean isSquare() {
if(getWidth() == getHeight()) return true;
return false;
}
double area() {
return getWidth() * getHeight();
}
}
class AbsShape {
public static void main(String args[]) {
TwoDShape shapes[] = new TwoDShape[4];
shapes[0] = new Triangle("right", 8.0, 12.0);
shapes[1] = new Rectangle(10);
shapes[2] = new Rectangle(10, 4);
shapes[3] = new Triangle(7.0);
```

```
for(int i=0; i < shapes.length; i++) {
System.out.println("object is " +
shapes[i].getName());
System.out.println("Area is " + shapes[i].area());
System.out.println();
}
}
}
```

As the program illustrates, all subclasses of **TwoDShape** *must* override **area( )**.

To prove this to yourself, try creating a subclass that does not override **area( )**. You will receive a compile-time error. Of course, it is still possible to create an object reference of type **TwoDShape**, which the program does. However, it is no longer possible to declare objects of type **TwoDShape**. Because of this, in **main( )** the **shapes** array has been shortened to 4, and a generic **TwoDShape** object is no longer created.

One last point: notice that **TwoDShape** still includes the **showDim( )** and **getName( )** methods and that these are not modified by **abstract**. It is perfectly acceptable—indeed, quite common—for an abstract class to contain concrete methods which a subclass is free to use as is. Only those methods declared as **abstract** need be overridden by subclasses.

كما يوضح البرنامج ، يجب أن تتجاوز جميع الفئات الفرعية لـ TwoDShape المنطقة ().

لإثبات ذلك لنفسك ، حاول إنشاء فئة فرعية لا تتجاوز المنطقة (). سوف تتلقى خطأ وقت الترجمة. بالطبع ، لا يزال من الممكن إنشاء مرجع كائن من النوع TwoDShape ، وهو ما يفعله البرنامج. ومع ذلك ، لم يعد من الممكن إعلان كائنات من النوع TwoDShape. لهذا السبب ، تم تقصير مجموعة الأشكال الرئيسية () إلى 4 ، ولم يعد يتم إنشاء كائن TwoDShape عام.

نقطة أخيرة: لاحظ أن TwoDShape لا يزال يتضمن showDim () و getName ()

الأساليب وأن هذه لم يتم تعديلها بواسطة الملخص. من المقبول تمامًا - في الواقع ، شائع جدًا - أن تحتوي فئة مجردة على طرق ملموسة يكون للفئة الفرعية حرية استخدامها كما هي. فقط تلك الأساليب التي تم إعلانها على أنها مجردة تحتاج إلى أن تلغيها الفئات الفرعية.