22. Inheritance

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Outline

- Superclasses and Subclasses
- Using the super Keyword
- Overriding Methods
- The Object Class
- References

Inheritance

- Object-oriented programming allows you to derive new classes from existing classes.
- This is called ***inheritance***.
- Inheritance is an important and powerful concept in Java.
- In fact, every class you define in Java is inherited from an existing class, either explicitly or implicitly.
- The classes you created in the preceding chapters were all extended implicitly from the java.lang.Object class.

Superclasses and Subclasses

Superclasses and Subclasses

- A class C1 extended from another class C2 is called a ***subclass***, and C2 is called a ***superclass***.
- A superclass is also referred to as a ***supertype***, a ***parent class***, or a ***base class***
- A subclass is also referred to as a ***subtype***, a ***child class***, an ***extended class***, or a ***derived class***.
- A subclass inherits accessible data fields and methods from its superclass, and may also add new data fields and methods.

Superclasses and Subclasses

- Suppose you want to design the classes to model geometric objects like circles and rectangles.
- Geometric objects have many common properties such as:
  - color
  - filled or unfilled
  - Date created
- And behaviors:
  - Can be drawn in a certain color
  - filled or unfilled methods
  - get and set methods
  - getDateCreated()
  - toString() method returns a string representation for the object

Superclasses and Subclasses

GeometricObject	
-color: String	The color of the object (default: white).
-filled: boolean	Indicates whether the object is filled with a color (default: false).
-dateCreated: java.util.Date	The date when the object was created.
+GeometricObject()	Creates a GeometricObject.
+getColor(): String	Returns the color.
+setColor(color: String): void	Sets a new color.
+isFilled(): boolean	Returns the filled property.
+setFilled(filled: boolean): void	Sets a new filled property.
+getDateCreated(): java.util.Date	Returns the dateCreated.
+toString(): String	Returns a string representation of this object.

Circle

-radius: double

+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double

Rectangle

-width: double
-height: double

+Rectangle()
+Rectangle(width: double, height: double)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double

Superclasses and Subclasses

- The Circle class inherits all accessible data fields and methods from the GeometricObject class.
- In addition, it has a new data field, radius, and its associated get and set methods.
- It also contains the getArea(), getPerimeter(), and getDiameter() methods for returning the area, perimeter, and diameter of the circle.

GeometricObject.java

```
1   package chapter09;
2
3   public class GeometricObject {
4      private String color = "white";
5      private boolean filled;
6      private java.util.Date dateCreated;
7
8      /** Construct a default geometric object */
9      public GeometricObject() {
10        dateCreated = new java.util.Date();
11     }
12
13     /** Return color */
14     public String getColor() {
15        return color;
16     }
17
18     /** Set a new color */
19     public void setColor(String newColor) {
20        color = newColor;
21     }
22
```

GeometricObject.java

```
23    /** Return filled. Since filled is boolean,
24     so, the get method name is isFilled */
25    public boolean isFilled() {
26       return filled;
27    }
28
29    /** Set a new filled */
30    public void setFilled(boolean newFilled) {
31       filled = newFilled;
32    }
33
34    /** Get dateCreated */
35    public java.util.Date getDateCreated() {
36       return dateCreated;
37    }
38
39    /** Return a string representation of this object */
40    public String toString() {
41       return "created on " + dateCreated + "\ncolor: " + color +
42             " and filled: " + filled;
43    }
44  }
```

Circle.java

```
1  package chapter09;
2
3  public class Circle extends GeometricObject {
4     private double radius;
5
6     public Circle() {
7     }
8
9     public Circle(double newRadius) {
10       radius = newRadius;
11    }
12
13    /** Return radius */
14    public double getRadius() {
15       return radius;
16    }
17
18    /** Set a new radius */
19    public void setRadius(double newRadius) {
20       radius = newRadius;
21    }
22
```

Circle.java

```
23     /** Return area */
24     public double getArea() {
25       return radius * radius * Math.PI;
26     }
27
28     /** Return diameter */
29     public double getDiameter() {
30       return 2 * radius;
31     }
32
33     /** Return perimeter */
34     public double getPerimeter() {
35       return 2 * radius * Math.PI;
36     }
37
38     /** Print the circle info */
39     public void printCircle() {
40       System.out.println("The circle is created " + getDateCreated() +
41           " and the radius is " + radius);
42     }
43   }
```

Rectangle.java

```
1  package chapter09;
2
3  public class Rectangle extends GeometricObject {
4     private double width;
5     private double height;
6
7     public Rectangle() {
8     }
9
10    public Rectangle(double newWidth, double newHeight) {
11       width = newWidth;
12       height = newHeight;
13    }
14
15    /** Return width */
16    public double getWidth() {
17       return width;
18    }
19
20    /** Set a new width */
21    public void setWidth(double newWidth) {
22       width = newWidth;
```

Rectangle.java

```
23     }
24
25     /** Return height */
26     public double getHeight() {
27       return height;
28     }
29
30     /** Set a new height */
31     public void setHeight(double newHeight) {
32       height = newHeight;
33     }
34
35     /** Return area */
36     public double getArea() {
37       return width * height;
38     }
39
40     /** Return perimeter */
41     public double getPerimeter() {
42       return 2 * (width + height);
43     }
44   }
```

Superclasses and Subclasses

- The classes Circle and Rectangle extend the GeometricObject class.
- The reserved word extends tells the compiler that these classes extend the GeometricObject class, thus inheriting the methods getColor, setColor, isFilled, setFilled, and toString.

TestCircleRectangle.java

```
1   package chapter09;
2
3   public class TestCircleRectangle {
4     public static void main(String[] args) {
5       Circle circle = new Circle(1);
6       System.out.println("A circle " + circle.toString());
7       System.out.println(circle.getRadius());
8       System.out.println("The radius is " + circle.getRadius());
9       System.out.println("The area is " + circle.getArea());
10      System.out.println("The diameter is " + circle.getDiameter());
11
12      Rectangle rectangle = new Rectangle(2, 4);
13      System.out.println("\nA rectanlge " + rectangle.toString());
14      System.out.println("The area is " + rectangle.getArea());
15      System.out.println("The perimeter is " +
16        rectangle.getPerimeter());
17    }
18  }
```

TestCircleRectangle.java

- Output:

```
A circle created on Tue Sep 30 22:55:31 IRST 2008
color: white and filled: false
1.0
The radius is 1.0
The area is 3.141592653589793
The diameter is 2.0

A rectangle created on Tue Sep 30 22:55:32 IRST 2008
color: white and filled: false
The area is 8.0
The perimeter is 12.0
```

Superclasses and Subclasses

- Contrary to the conventional interpretation, a subclass is not a subset of its superclass.
- In fact, a subclass usually contains more information and functions than its superclass.

Superclasses and Subclasses

- Private data fields and methods in a superclass are not accessible outside of the class.
- Therefore, they are not inherited in a subclass.

Using the super Keyword

Using the super Keyword

- A constructor is used to construct an instance of a class.
- Unlike properties and methods, a superclass's constructors are not inherited in the subclass.
- They can only be invoked from the subclasses' constructors, using the keyword super.
- If the keyword super is not explicitly used, the superclass's no-arg constructor is automatically invoked.

Using the super Keyword

- The keyword super refers to the superclass of the class in which super appears.
- It can be used in two ways:
  - To call a superclass constructor.
  - To call a superclass method.

Calling Superclass Constructors

- The syntax to call a superclass constructor is:

  super()

  super(parameters);
- The statement super() invokes the no-arg constructor of its superclass,
- The statement super(arguments) invokes the superclass constructor that matches the arguments.
- The statement super() or super(arguments) must appear in the first line of the subclass constructor and is the only way to invoke a superclass constructor.

Using the super Keyword

- A constructor may invoke an overloaded constructor or its superclass's constructor.
- If neither of them is invoked explicitly, the compiler puts super() as the first statement in the constructor.
- For example:

```
public A() {
}
```

is equivalent to

```
public A() {
  super();
}
```

```
public A(double d) {
  // some statements
}
```

is equivalent to

```
public A(double d) {
  super();
  // some statements
}
```

Using the super Keyword

- You must use the keyword super to call the superclass constructor.
- Invoking a superclass constructor's name in a subclass causes a syntax error.

Constructor Chaining

- In any case, constructing an instance of a class invokes the constructors of all the superclasses along the inheritance chain.
- A superclass's constructor is called before the subclass's constructor.
- This is called ***constructor chaining***.

Faculty.java

```
1   package chapter09;
2
3   public class Faculty extends Employee {
4     public static void main(String[] args) {
5       new Faculty();
6     }
7
8     public Faculty() {
9       System.out.println("(3) Faculty's no-arg constructor is invoked");
10    }
11  }
12
13  class Employee extends Person {
14    public Employee() {
15      System.out.println("(2) Employee's no-arg constructor is invoked");
16    }
17  }
18
19  class Person {
20    public Person() {
21      System.out.println("(1) Person's no-arg constructor is invoked");
22    }
23  }
```

Faculty.java

- The output:
  - (1) Person's no-arg constructor is invoked
  - (2) Employee's no-arg constructor is invoked
  - (3) Faculty's no-arg constructor is invoked he output:

Faculty.java

```
1  package chapter09;
2
3  public class Faculty extends Employee {
4     public static void main(String[] args) {
5        new Faculty();
6     }
7
8     public Faculty() {
9        super();
10       System.out.println("(3) Faculty's no-arg constructor is invoked");
11    }
12 }
13
14 class Employee extends Person {
15    public Employee() {
16       System.out.println("(2) Employee's no-arg constructor is invoked");
17    }
18 }
19
20 class Person {
21    public Person() {
22       System.out.println("(1) Person's no-arg constructor is invoked");
23    }
24 }
```

Constructor Chaining

- If a class is designed to be extended, it is better to provide a no-arg constructor to avoid programming errors.
- Find out the errors in the program:

```
public class Apple extends Fruit {
}
class Fruit {
   public Fruit(String name) {
   System.out.println("Fruit's constructor is invoked");
  }
}
```

Constructor Chaining

- Since no constructor is explicitly defined in Apple, Apple's default no-arg constructor is declared implicitly.
- Since Apple is a subclass of Fruit, Apple's default constructor automatically invokes Fruit's no-arg constructor.
- However, Fruit does not have a no-arg constructor because Fruit has an explicit constructor defined.
- Therefore, the program cannot be compiled.

Calling Superclass Methods

- The keyword super can also be used to reference a method in the superclass. The syntax is like this:

```
super.method(parameters);
```

- You could rewrite the printCircle() method in the Circle class as follows:

```
public void printCircle() {
    System.out.println("The circle is created " +
    super.getDateCreated() + " and the radius is " + radius);
}
```

Overriding Methods

Overriding Methods

- A subclass inherits methods from a superclass.
- Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass.
- This is referred to as ***method overriding***.

Overriding Methods

- The toString method in the GeometricObject class returns the string representation for a geometric object.
- This method can be overridden to return the string representation for a circle.
- To override it, add the following new method in Circle.java:

```
public class Circle extends GeometricObject {
    // Other methods are omitted
    /** Override the toString method defined in GeometricObject */
    public String toString() {
        return super.toString() + "\nradius is " + radius;
    }
}
```

Overriding Methods

- An instance of Circle can not invoke the toString method defined in the GeometricObject class.
- Because toString() in GeometricObject has been overridden in Circle.

Overriding Methods

- An instance method can be overridden only if it is accessible.
- Thus a private method cannot be overridden, because it is not accessible outside its own class.
- If a method defined in a subclass is private in its superclass, the two methods are completely unrelated.

Overriding Methods

- Like an instance method, a static method can be inherited.
- However, a static method cannot be overridden.
- If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

Overriding vs. Overloading

- Overloading a method is a way to provide more than one method with the same name but with different signatures to distinguish them.
- To override a method, the method must be defined in the subclass using the same signature and same return type as in its superclass.

Overriding vs. Overloading

- In (a), the method p(int i) in class A overrides the same method defined in class B. However, in (b), the method p(double i) in class A and the method p(int i) in class B are two overloaded methods. The method p(int i) in class B is inherited in A.

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

class B {
  public void p(int i) {
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

(a)

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

class B {
  public void p(int i) {
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

(b)

Overriding vs. Overloading

- When you run the Test class in (a), a.p(10) invokes the p(int i) method defined in class A, so the program displays 10.
- When you run the Test class in (b), a.p(10) invokes the p(int i) method defined in class B, so nothing is printed.

The Object Class

The Object Class

- If no inheritance is specified when a class is defined, the superclass of the class is java.lang.Object class by default.
- For example, the following two class declarations are the same:

```
public class Circle {
  ...
}
```

Equivalent

```
public class Circle extends Object {
  ...
}
```

- It is important to be familiar with the methods provided by the Object class so that you can use them in your classes.

The toString() method

- The signature of the toString() method is

  public String toString()

- The toString() method returns a string representation of the object.
- The default implementation returns a string consisting of a class name of which the object is an instance, the at sign (@), and a number representing this object.

The toString() method

- For example:

```
Loan loan = new Loan();
System.out.println(loan.toString());
```

- The code displays something like Loan@15037e5 .
- This message is not very helpful or informative.
- Usually you should override the toString method.

References

References

- Y. Daniel Liang, **Introduction to Java Programming**, Sixth Edition, Pearson Education, 2007. (Chapter 9)

The End