24. Inheritance

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

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

Superclasses and Subclasses

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.

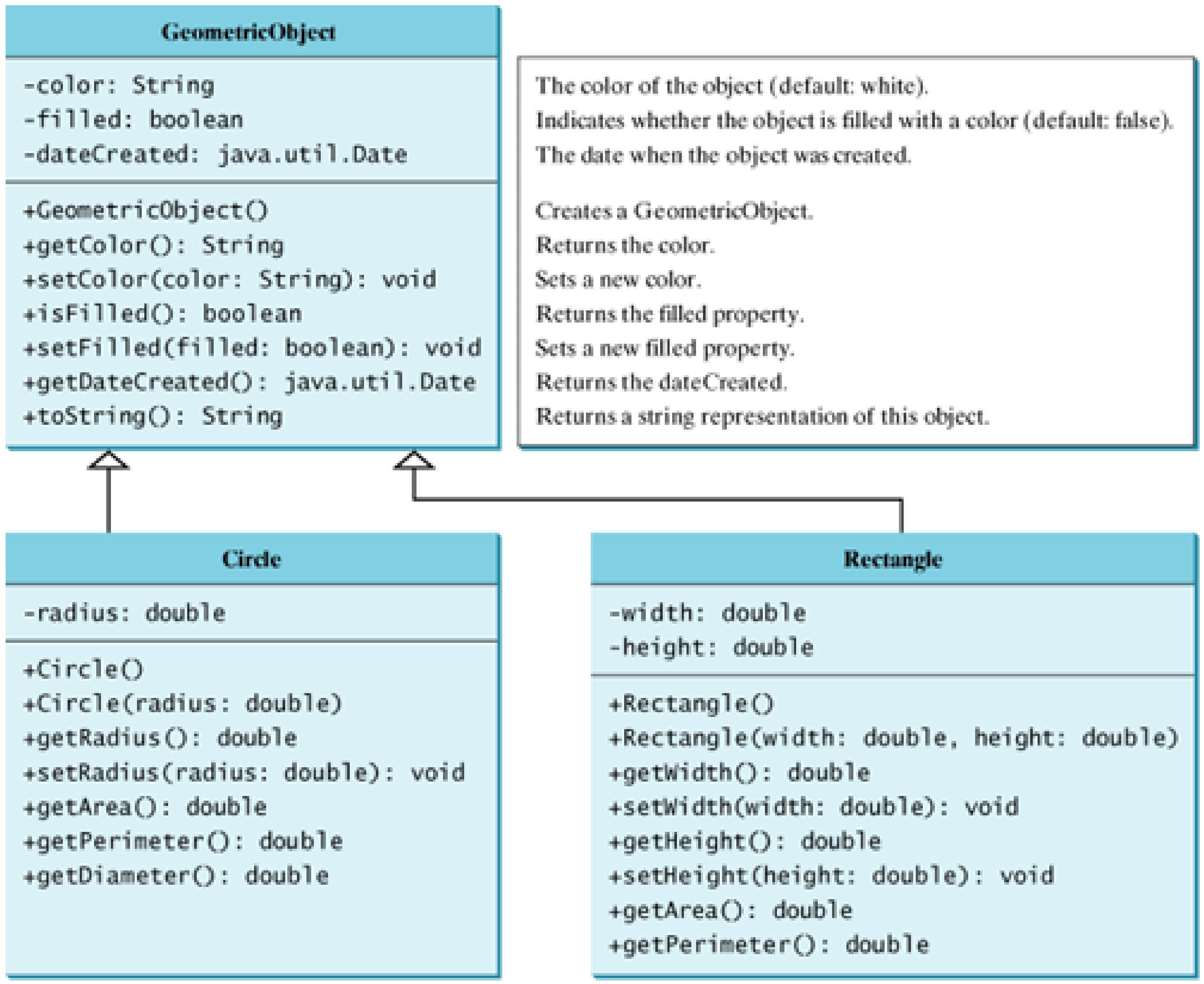
Superclasses and Subclasses

- **Subclass**
  - A class **C1** is extended from another class **C2** is called a **subclass**,
  - It is also referred to as a **subtype**, a **child class**, an **extended class**, or a **derived class**.
- **Superclass**
  - Class **C2** is called a **superclass**
  - It is also referred to as a **supertype**, a **parent class**, or a **base 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
-filled: boolean
-dateCreated: java.util.Date
+GeometricObject()
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
The color of the object (default: white).
Indicates whether the object is filled with a color (default: false).
The date when the object was created.
Creates a GeometricObject.
Returns the color.
Sets a new color.
Returns the filled property.
Sets a new filled property.
Returns the dateCreated.
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.

Superclasses and Subclasses

- The programs:
  - GeometricObject.java
  - Circle.java
  - Rectangle.java
  - TestCircleRectangle.java

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

- 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.

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 **variables** 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

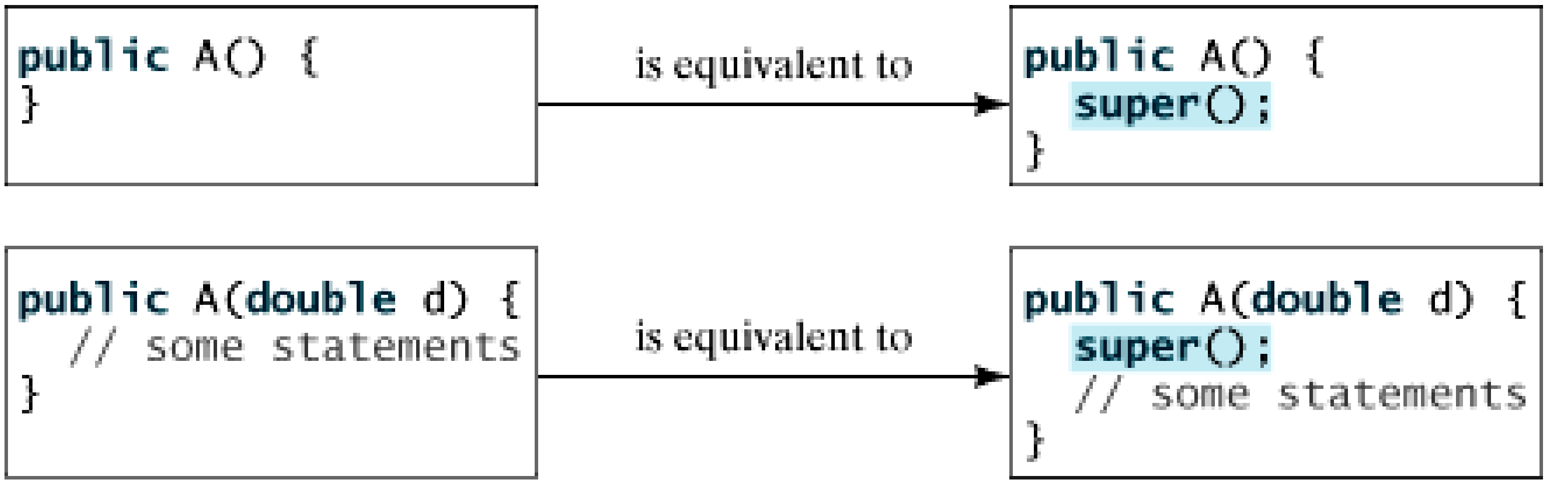
- The syntax to call a superclass constructor is:

  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

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

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

- Example:
  - TestFaculty.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:
```

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
{
    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

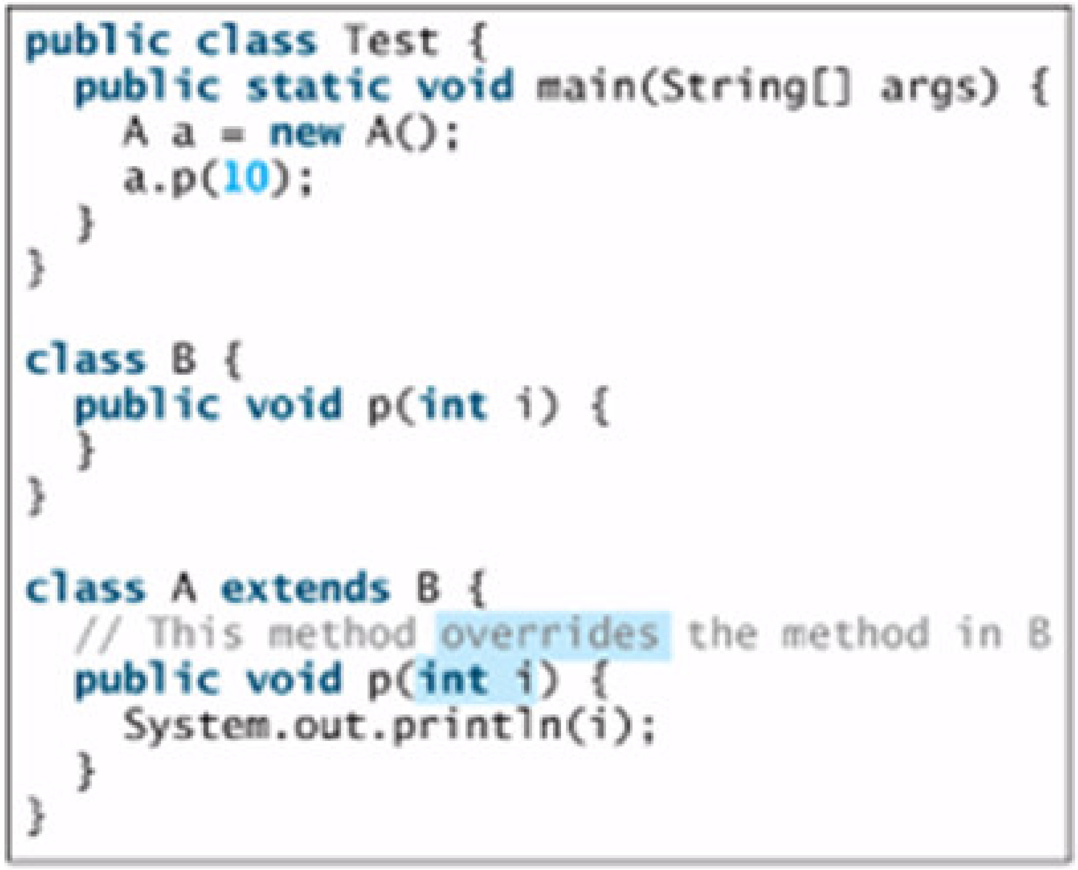
- Like an instance method, a static method can be inherited.
- However, a static method **cannot be overridden**.
- If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

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

- The method p(int i) in class A overrides the same method defined in class B.

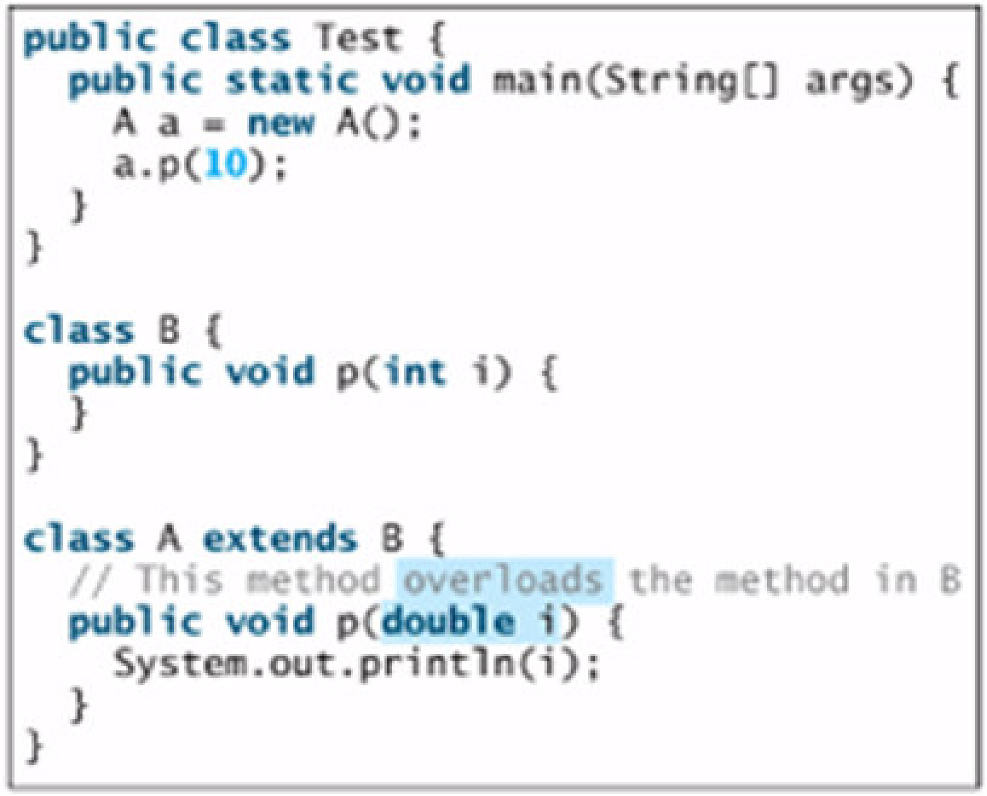


```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
  }
}

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)

Overriding vs. Overloading

- 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 overloads the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

(b)

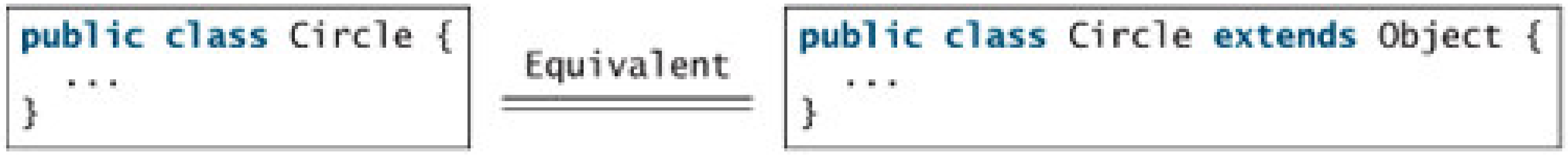
Overriding vs. Overloading

- 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 {
  ...
}
```

```
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.

Two Methods of Object Class

- equals() Method
  - Use the equals() to compare two objects for equality. This method returns true if the objects are equal, false otherwise.
- toString() Method
  - 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
  - For an object of Object class the at sign (@) and a number representing this object is returned.

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