

Inheritance and Polymorphism

- Using the super Keyword
- Calling Superclass Constructors
- Constructor Chaining
- Calling Superclass Methods

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

A subclass inherits accessible data fields and methods from its superclass. Does it inherit constructors?

No. Constructors are not inherited.

Can the superclass's constructors be invoked from a 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.

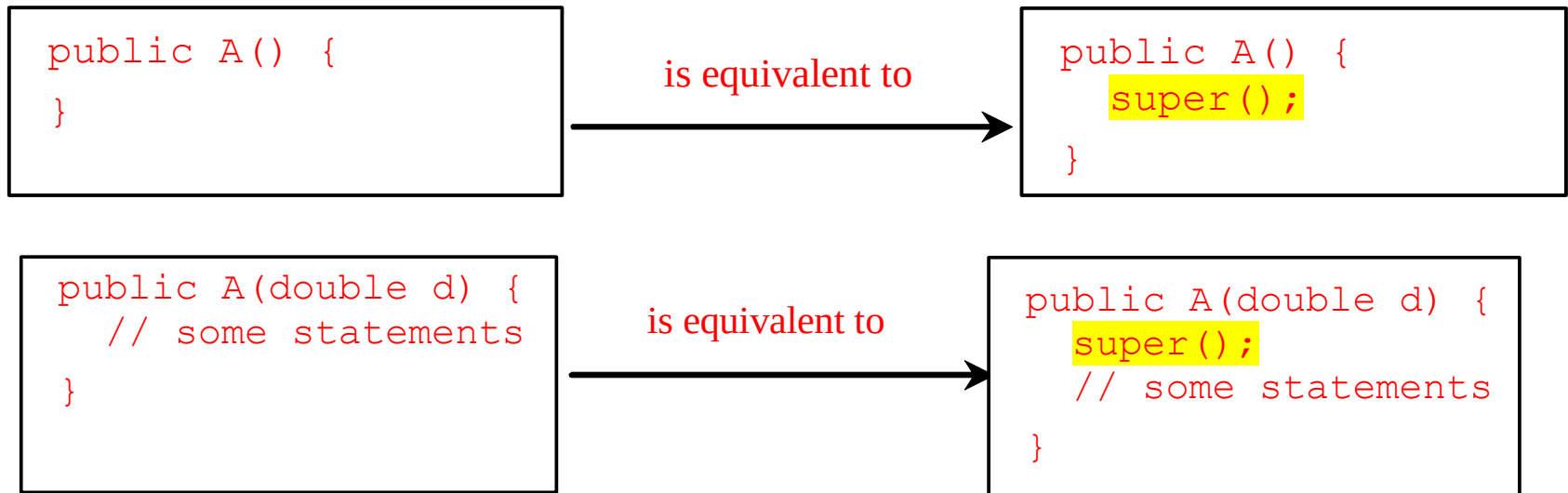
Calling Superclass Constructors

The syntax to call a superclass's constructor is: **super()**, or **super(parameters)**;

The statement **super()** invokes the no-arg constructor of its superclass, and the statement **super(arguments)** invokes the superclass constructor that matches the **arguments**.

Superclass's Constructor Is Always Invoked

If the superclass constructor is not invoked explicitly in the subclass constructor, the compiler puts super() as the first statement in the subclass constructor. This will call the no-arg constructor of the superclass if one exists. If in such case the superclass does not contain a no-arg constructor then we get a syntax error. For example,



CAUTION

- To call the superclass constructor you must use the keyword super. Invoking a superclass constructor's name in a subclass causes a syntax error.
- Java requires that the statement that uses the keyword super appear first in the constructor.

Constructor Chaining

Constructing an instance of a class invokes all the superclasses' constructors along the inheritance chain. This is called constructor chaining.

```
public class Faculty extends Employee {
    public static void main(String[] args) {
        new Faculty();
    }

    public Faculty() {
        System.out.println("(4) Faculty's no-arg constructor is invoked");
    }
}

class Employee extends Person {
    public Employee() {
        this("(2) Invoke Employee's overloaded constructor");
        System.out.println("(3) Employee's no-arg constructor is invoked");
    }

    public Employee(String s) {
        System.out.println(s);
    }
}

class Person {
    public Person() {
        System.out.println("(1) Person's no-arg constructor is invoked");
    }
}
```

Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```

1. Start from the
main method

Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```



Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```

3. Invoke Employee's no-arg constructor

Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```

4. Invoke Employee(String)
constructor

Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```

5. Invoke Person() constructor

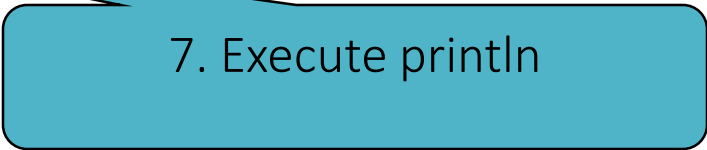
Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```

6. Execute println

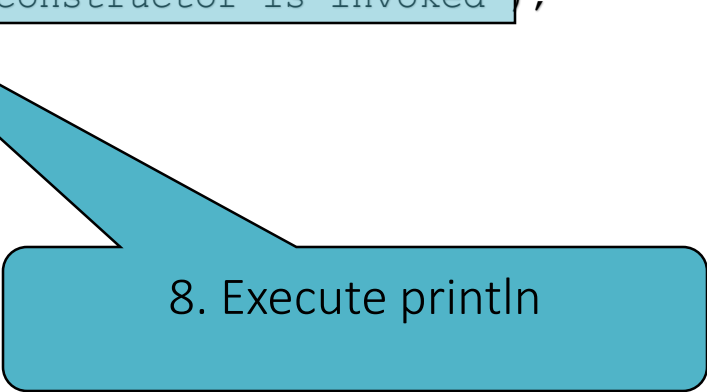
Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```



Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```



8. Execute println

Trace Execution

```
public class Faculty extends Employee {  
    public static void main(String[] args) {  
        new Faculty();  
    }  
  
    public Faculty() {  
        System.out.println("(4) Faculty's no-arg constructor is invoked");  
    }  
}  
  
class Employee extends Person {  
    public Employee() {  
        this("(2) Invoke Employee's overloaded constructor");  
        System.out.println("(3) Employee's no-arg constructor is invoked");  
    }  
  
    public Employee(String s) {  
        System.out.println(s);  
    }  
}  
  
class Person {  
    public Person() {  
        System.out.println("(1) Person's no-arg constructor is invoked");  
    }  
}
```

9. Execute println

Example on the Impact of a Superclass without no-arg Constructor

What is the error below?

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

```
public class Apple extends Fruit {  
    public Apple(String name) {  
    }  
}
```

The constructor of Apple does not contain an explicit call to the superclass Fruit constructor so it will call the no-arg constructor of Fruit which does not exist. This will cause an error.

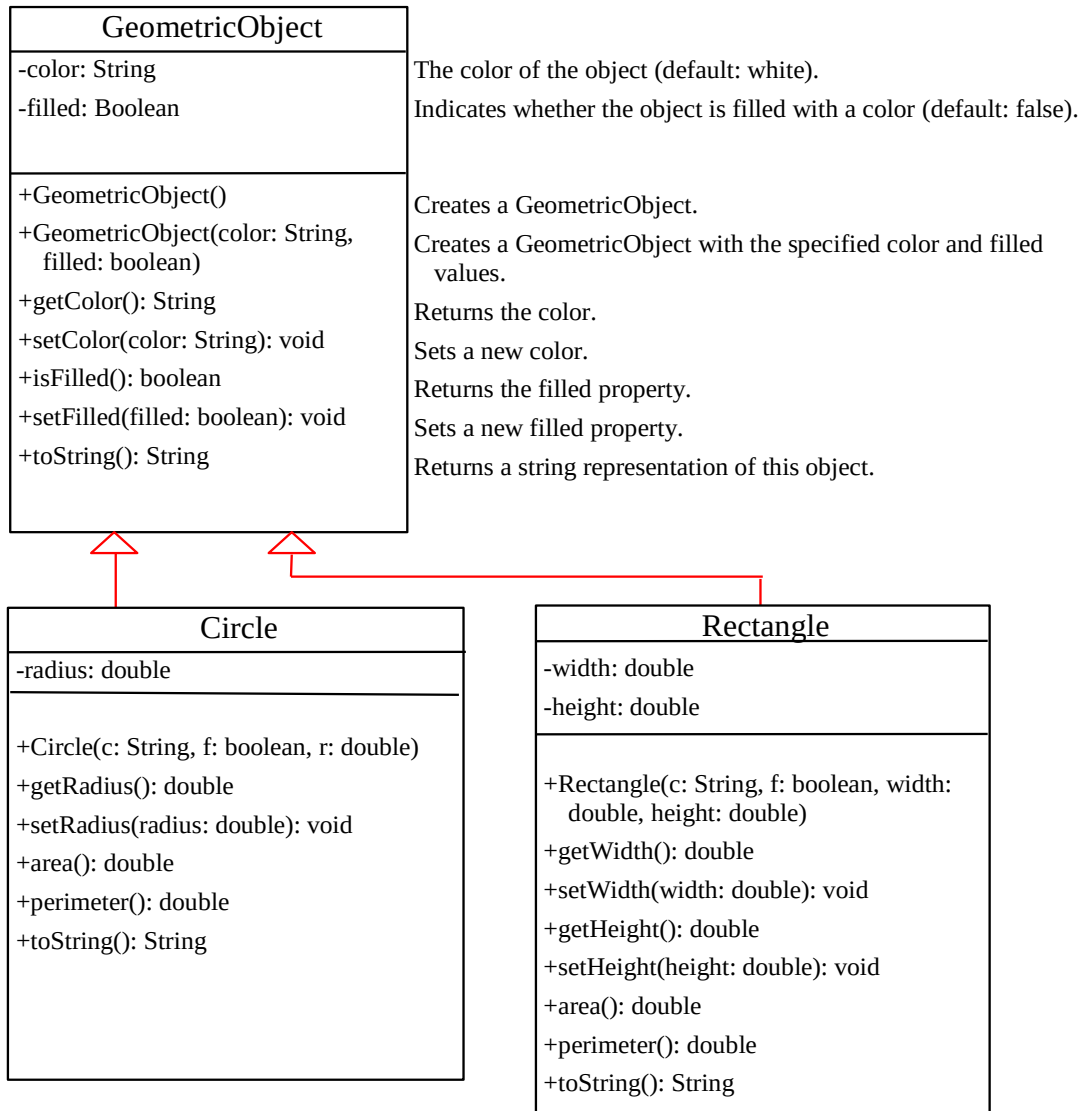
Calling Superclass Methods

You can rewrite the toString() method in the Circle class as follows:

```
public class Circle extends GeometricObject {
...
    public String toString() {
        return super.toString() + ".\nThe radius is " + radius;
        // super.toString( ) calls the toString of GeometricObject
    }
}

public class GeometricObject {
...
    public String toString() {
        String s = "The color is " + color;
        if (filled)
            s += ".\nThe geometric object is filled";
        else
            s += ".\nThe geometric object is not filled";
        return s;
    }
}
```

Rewriting the code of Circle class using the super keyword



You can download and run the java project:

“TestingInheritance using super”

You can also look at the code in the next slides.

Class GeometricObject

```
public class GeometricObject {  
  
    private String color ;  
  
    private boolean filled;  
  
    public GeometricObject() { // No arg constructor  
  
        color = "white";  
  
        filled = false;  
  
    }  
  
    public GeometricObject(String c, boolean f) {  
  
        color = c;  
  
        filled = f;  
  
    }  
  
    public String getColor() {  
  
        return color;  
  
    }  
  
    public void setColor(String c) {  
  
        color = c;  
  
    }  
  
    public boolean isFilled() {  
  
        return filled;  
  
    }  
  
    public void setFilled(boolean f) {  
  
        filled = f;  
  
    }  
}
```



```
    public String toString() {  
        String s = "The color is " + color;  
        if (filled)  
            s+= ".\nThe geometric object is filled";  
        else  
            s+= ".\nThe geometric object is not filled";  
        return s;  
    }  
} // end class GeometricObject
```

Class Circle

```
public class Circle extends GeometricObject {  
    private double radius;  
  
    public Circle(String c, boolean f, double r) {  
        super(c,f); // calls constructor of GeometricObject using the keyword super  
        radius = r;  
    }  
  
    public double getRadius() {  
        return radius;  
    }  
  
    public void setRadius(double radius) {  
        this.radius = radius;  
    }  
  
    public double area() {  
        return radius * radius * Math.PI;  
    }  
  
    public double perimeter() {  
        return 2 * radius * Math.PI;  
    }  
  
    public String toString() {  
        // call toString of GeometricObject using the keyword super  
        return super.toString() + ".\nThe radius is " + radius;  
    }  
} // end class Circle
```

Class Rectangle

```
public class Rectangle extends GeometricObject {  
    private double width;  
    private double height;  
    public Rectangle(String c, boolean f, double width, double height) {  
        super(c, f); // calls constructor of GeometricObject  
        this.width = width;  
        this.height = height;  
    }  
    public double getWidth() {  
        return width;  
    }  
    public void setWidth(double width) {  
        this.width = width;  
    }  
    public double getHeight() {  
        return height;  
    }  
    public void setHeight(double height) {  
        this.height = height;  
    }  
    public double area() {  
        return width * height;  
    }  
}
```



```
    public double perimeter() {  
        return 2 * (width + height);  
    }  
  
    public String toString() {  
        // call toString of GeometricObject  
        return super.toString() +  
            ".\nThe height is " + height +  
            ".\nThe width is " + width;  
    }  
  
} // end class Rectangle
```

Application

```
public class TestingInheritance {

    public static void main(String[] args) {

        Circle c1 = new Circle("red", true, 2);

        System.out.println(c1.getColor());

        System.out.println(c1.getRadius());

        System.out.println(c1);

        GeometricObject g1 = new GeometricObject("black", false);

        System.out.println(g1.getColor());

        // getRadius() will give an error since class GeometricObject does not
        // have a method called getRadius()

        // System.out.println(g1.getRadius());

        System.out.println(g1);

        Rectangle r1 = new Rectangle("green", true, 3,4);

        System.out.println(r1.getColor());

        System.out.println(r1.getHeight());

        System.out.println(r1);

    }

}
```