B.Sc. In Software Development.

Applications Programming.

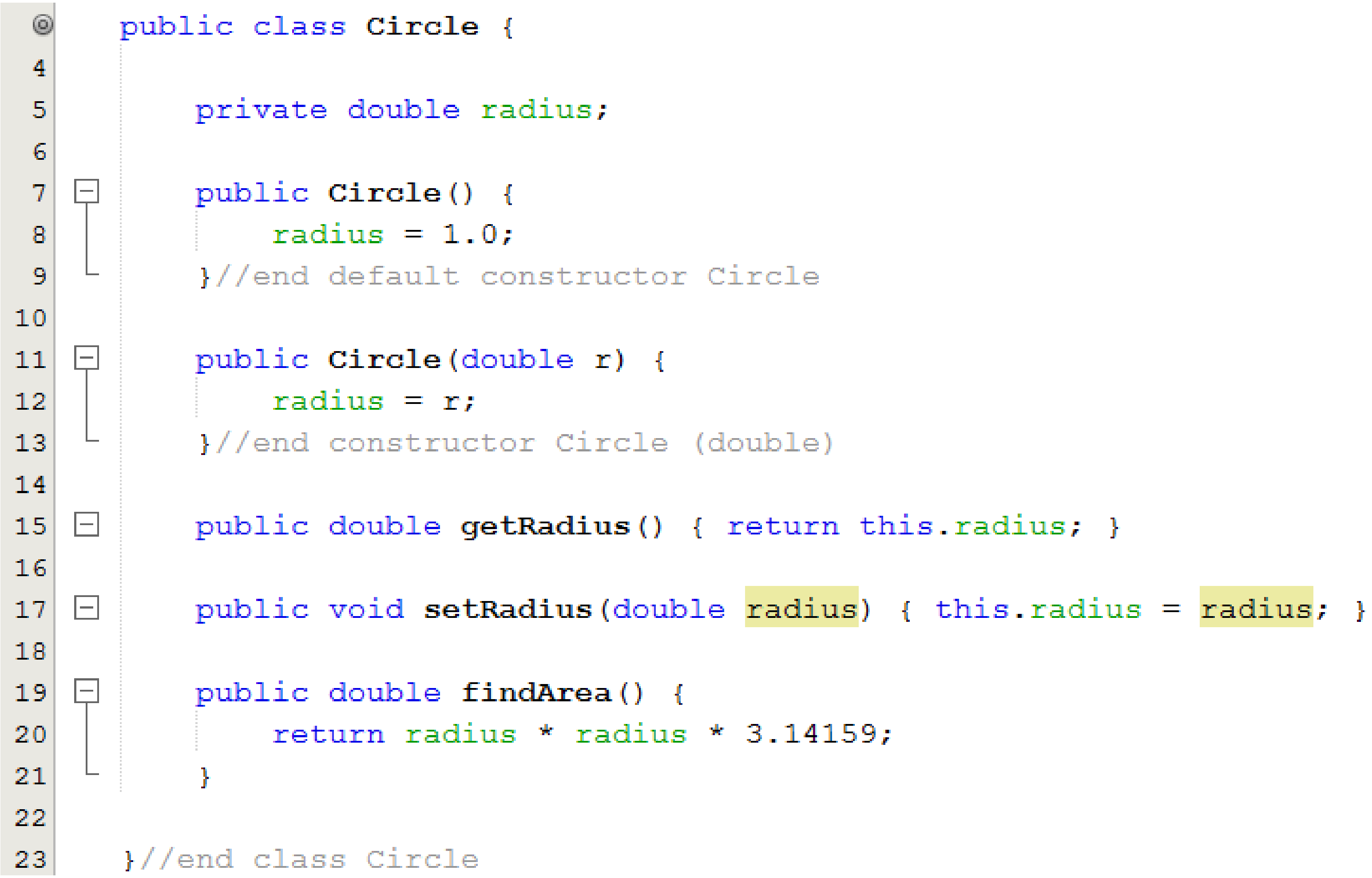
Inheritance, Polymorphism and Interfaces.

LIMERICK INSTITUTE
OF TECHNOLOGY
SCHOOL OF SCIENCE,
ENGINEERING & I.T.
Department of Information Technology

Inheritance

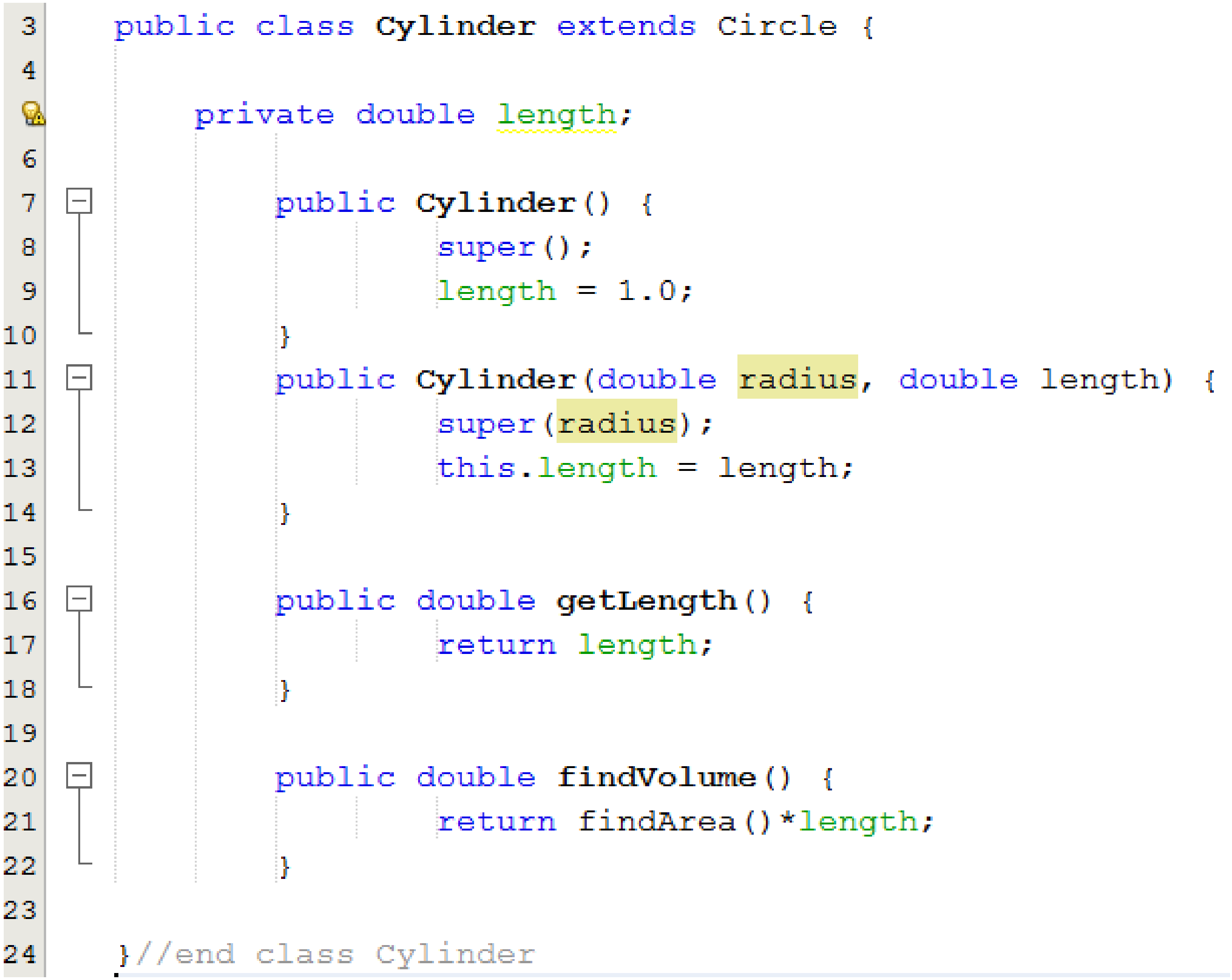
- A class c1 derived from another c2 is called a subclass, and c2 is called a superclass.
  - Sometimes a superclass is referred to as a parent class or a base class and a subclass is referred to as a child class, an extended class or a derived class.
- A subclass inherits functionality from its superclass and also creates new data and new methods.
- Subclasses have more functionality than their super classes.

Inheritance Example



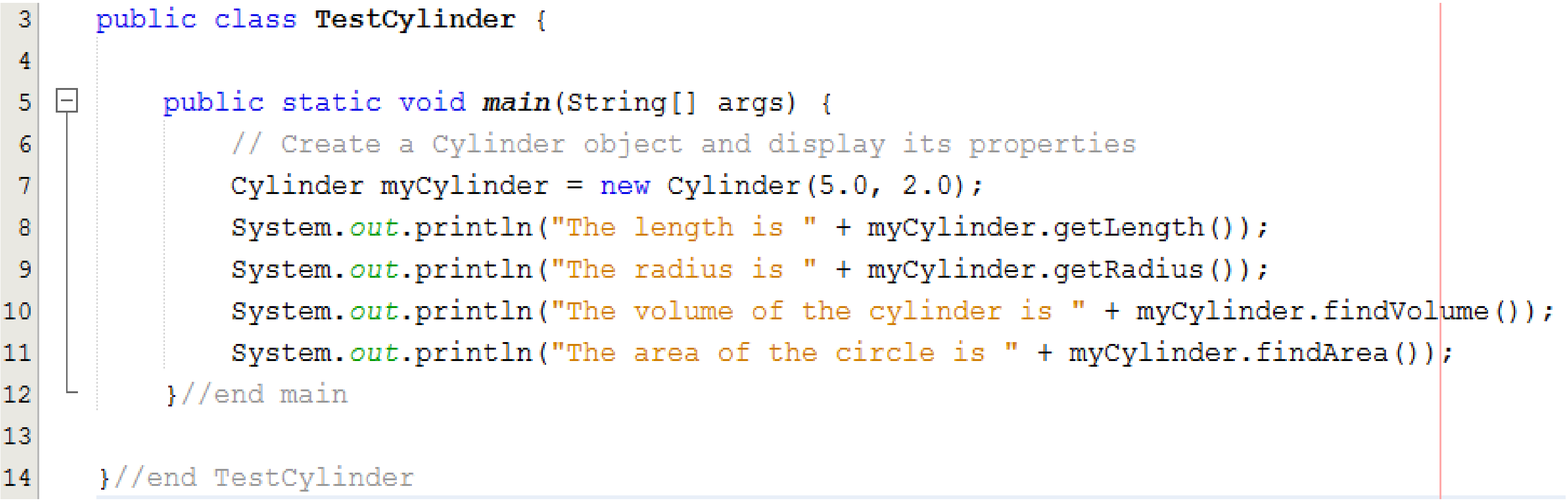
```java
public class Circle {

    private double radius;

    public Circle() {
        radius = 1.0;
    }//end default constructor Circle

    public Circle(double r) {
        radius = r;
    }//end constructor Circle (double)

    public double getRadius() { return this.radius; }

    public void setRadius(double radius) { this.radius = radius; }

    public double findArea() {
        return radius * radius * 3.14159;
    }

}//end class Circle
```

Inheritance Example

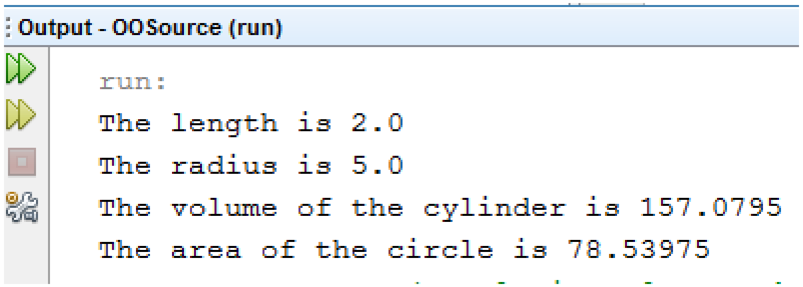


```java
public class Cylinder extends Circle {

    private double length;

        public Cylinder() {
                super();
                length = 1.0;
        }
        public Cylinder(double radius, double length) {
                super(radius);
                this.length = length;
        }

        public double getLength() {
                return length;
        }

        public double findVolume() {
                return findArea()*length;
        }

}//end class Cylinder
```

Inheritance Example



```java
public class TestCylinder {

    public static void main(String[] args) {
        // Create a Cylinder object and display its properties
        Cylinder myCylinder = new Cylinder(5.0, 2.0);
        System.out.println("The length is " + myCylinder.getLength());
        System.out.println("The radius is " + myCylinder.getRadius());
        System.out.println("The volume of the cylinder is " + myCylinder.findVolume());
        System.out.println("The area of the circle is " + myCylinder.findArea());
    }//end main

}//end TestCylinder
```



```
run:
The length is 2.0
The radius is 5.0
The volume of the cylinder is 157.0795
The area of the circle is 78.53975
```

The Super Keyword

- Refers to the superclass & used in two ways.
  1. To call a superclass constructor.
  2. To call a superclass method.

- You could rewrite the findVolume() method in the Cylinder class with the following:

```
double findVolume() {
  return super.findArea()*length;
 }
```

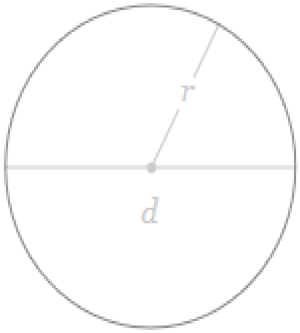
Overriding Methods Example

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

Overriding Methods Example

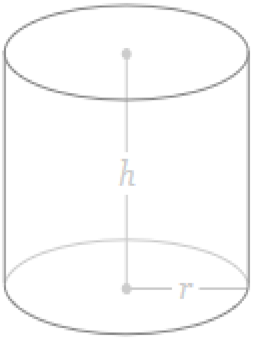
Area of
a Circle

$$A = \pi r^2$$



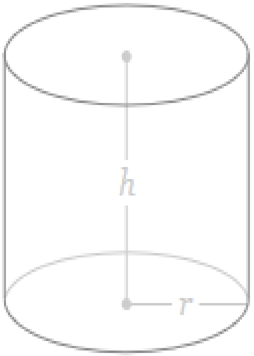
(Surface) Area
of a Cylinder

$$A = 2\pi r h + 2\pi r^2$$

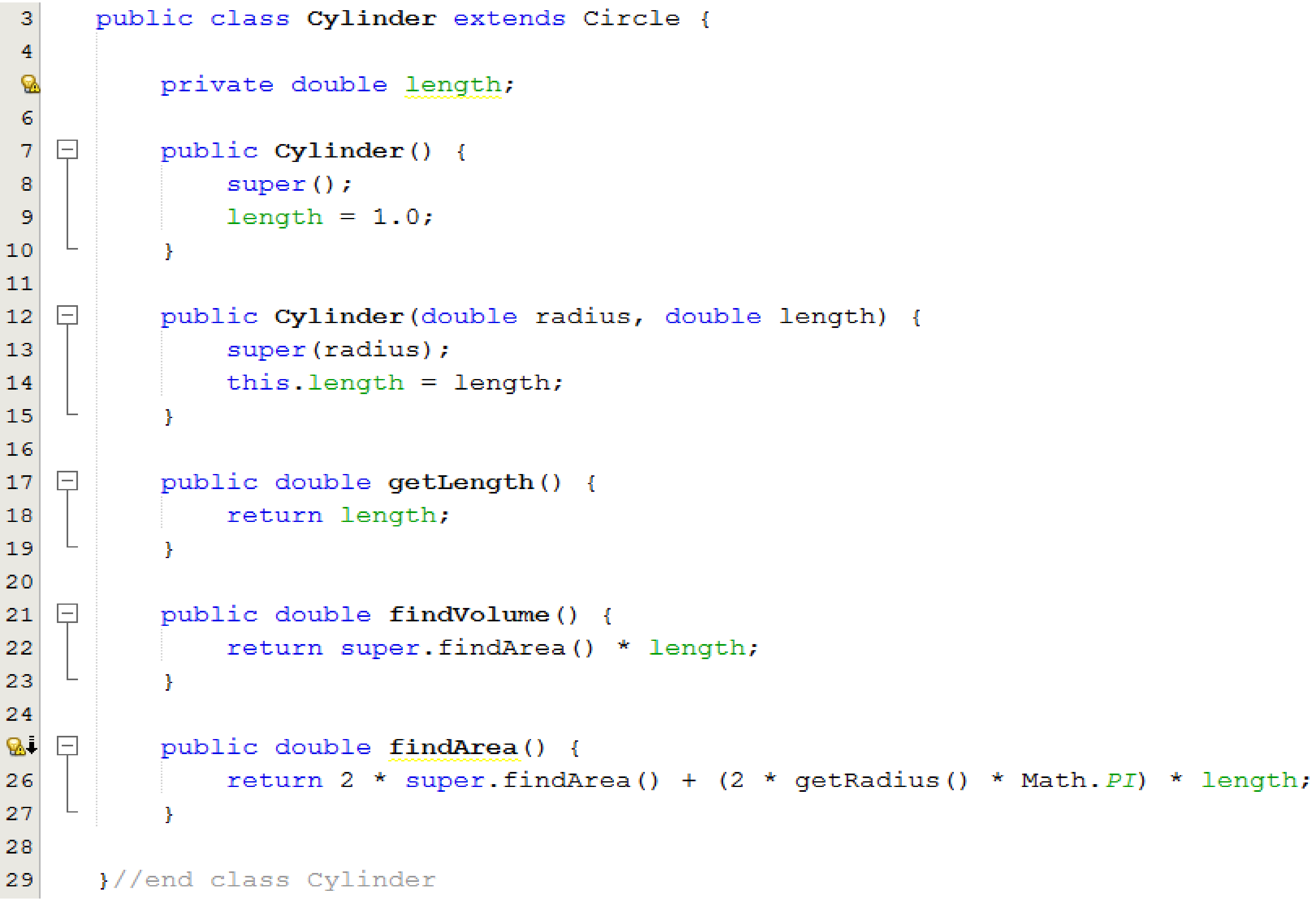


Volume of a
Cylinder

$$V = \pi r^2 h$$

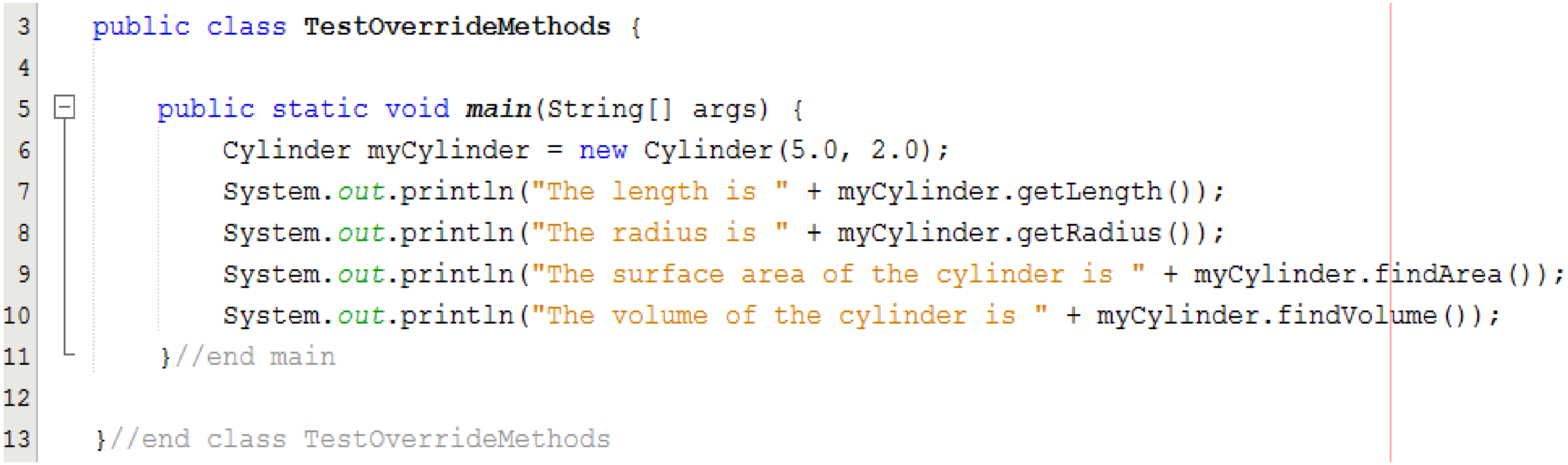


Overriding Methods Example

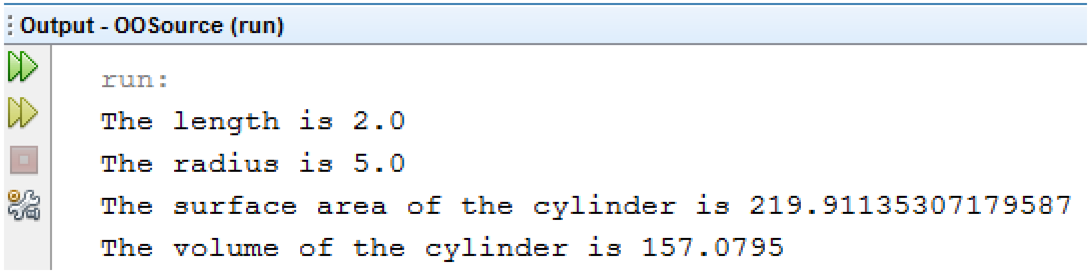


```java
public class Cylinder extends Circle {

    private double length;

    public Cylinder() {
        super();
        length = 1.0;
    }

    public Cylinder(double radius, double length) {
        super(radius);
        this.length = length;
    }

    public double getLength() {
        return length;
    }

    public double findVolume() {
        return super.findArea() * length;
    }

    public double findArea() {
        return 2 * super.findArea() + (2 * getRadius() * Math.PI) * length;
    }

}//end class Cylinder
```

Overriding Methods Example



```java
public class TestOverrideMethods {

    public static void main(String[] args) {
        Cylinder myCylinder = new Cylinder(5.0, 2.0);
        System.out.println("The length is " + myCylinder.getLength());
        System.out.println("The radius is " + myCylinder.getRadius());
        System.out.println("The surface area of the cylinder is " + myCylinder.findArea());
        System.out.println("The volume of the cylinder is " + myCylinder.findVolume());
    }//end main

}//end class TestOverrideMethods
```



```
run:
The length is 2.0
The radius is 5.0
The surface area of the cylinder is 219.91135307179587
The volume of the cylinder is 157.0795
```

Object Class

- Every class in Java is derived from the `java.lang.Object` class.
- If no inheritance is specified when a class is defined; the superclass of the class is `Object`.
- Important to be familiar with some of the more useful the methods with the `Object` class.

Object Class – equals Method

- Tests if two objects are equal.
- The syntax is:

```
object1.equals(object2)
```

- In the above case, object1 and object2 are objects of the same class.

- The default implementation of the equals method in the `Object` class is as follows:

```
public boolean equals(Object obj) {
        return(this ==obj)
}
```

Object Class – equals Method

- Using the equals method is equivalent to the == operator in the `Object` class.
- Its intended for the subclasses of the `Object` class to modify the equals method to test whether two distinct objects of the same class have the same contents.

Object Class – toString Method

- Invoking `object.toString()` returns a `String` representation of this object.
- By default, it returns a string consisting of a class name of which the object is an instance, the at sign (@), and a number representing this object.
- For example, consider the following:

```
Cylinder cyl = new Cylinder(5.0, 2.0);
System.out.println(cyl.toString());
```

Object Class – toString Method

- You should override the `toString` method so that it returns a useful string.
- For example, you can override the `toString` method in the `Cylinder` class.

```
public String toString() {
return “Cylinder length = “ + length;
}
```

- `System.out.print(myCylinder.toString())` would display something like this:

Cylinder length = 2;

Object Class – clone Method

- Sometimes you need to make a copy of an object.
- Mistakenly, you might use the following assignment statement like this one.

```
newObject = oldObject;
```

- This statement does not create a duplicate object.
- It simply assigns the reference of oldObject to newObject.
- To create a new object with separate memory space, you need to use the clone() method.

```
newObject = oldObject.clone();
```

Abstract Classes

- In an inheritance hierarchy, classes become more specific and concrete with each new subclass.
- If you move from a subclass back up to a superclass, the classes become more general and less specific.
- Proper class design should ensure that a subclass contains common features of its subclasses.
- Sometimes a superclass is so abstract that is cannot have any specific instances.
  - An Abstract class.

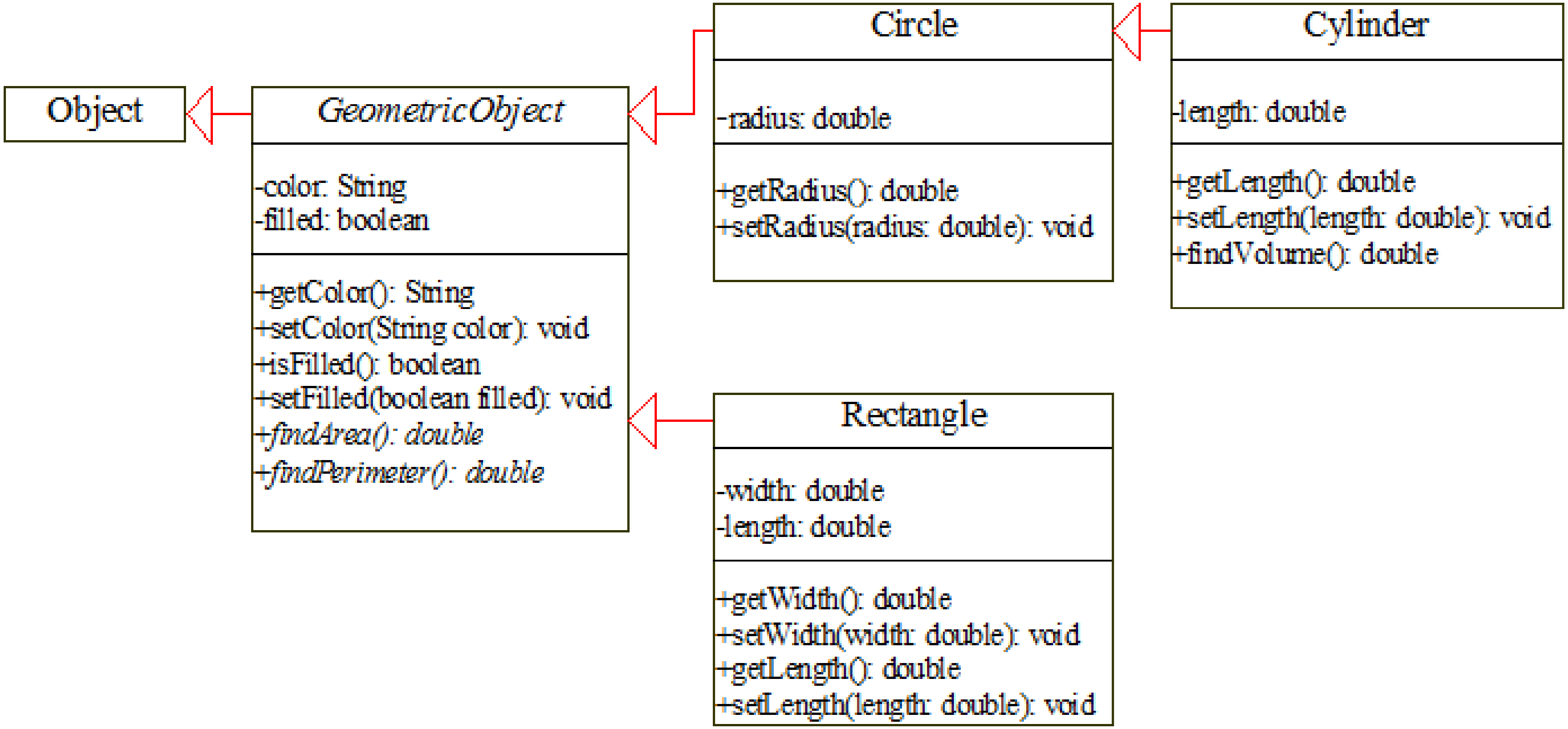
Abstract Classes

- Suppose you want to design the classes to model geometric objects like circles, cylinders and rectangles.
- Geometric objects have common properties and behaviours.
- They can be drawn in a certain colour, filled or unfilled.
    - Colour and filled are two common properties.
- Common behaviours include the fact that the areas and perimeters of geometric objects can be computed.
- A general class `GeometricObject` can be used to model all geometric objects.
    - This class contains the properties `colour` and `filled` along with the methods `findArea` and `findPerimeter`.

Abstract Classes

- A circle is a special type of geometric object, and thus it shares common properties and methods with a geometric object.
- A cylinder is a special type of circle, and thus it shares common properties and behaviours with a circle.
- It makes sense to define a `Circle` class that extends the `GeometricObject` class and a `Cylinder` class, which extends the `Circle` class.

Abstract Classes



Object

GeometricObject

-color: String
-filled: boolean

+getColor(): String
+setColor(String color): void
+isFilled(): boolean
+setFilled(boolean filled): void
+findArea(): double
+findPerimeter(): double

Circle

-radius: double

+getRadius(): double
+setRadius(radius: double): void

Cylinder

-length: double

+getLength(): double
+setLength(length: double): void
+findVolume(): double

Rectangle

-width: double
-length: double

+getWidth(): double
+setWidth(width: double): void
+getLength(): double
+setLength(length: double): void

Abstract classes are like regular classes with data and methods, but you cannot create instances of abstract classes using the new operator.

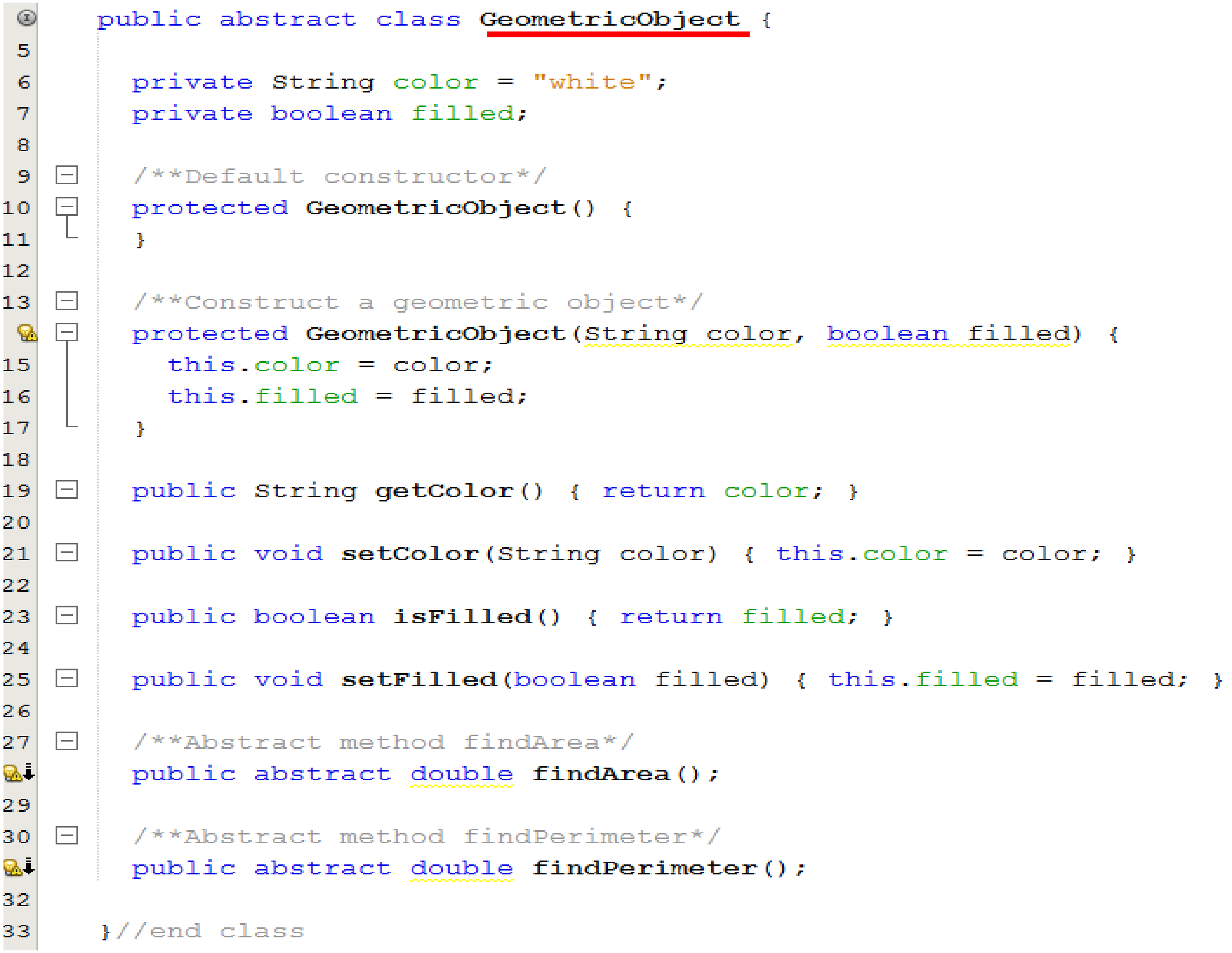
Abstract Classes

- The methods `findArea` and `findPerimeter` cannot be implemented in the `GeometricObject` class, because their implementation is dependent on the specific type of geometric object.
- Such methods are referred to as abstract methods.
  - An abstract method is a method signature without implementation.
  - Its implementation is provided by the subclasses.
  - A class that contains abstract methods must be declared abstract.

Abstract Classes

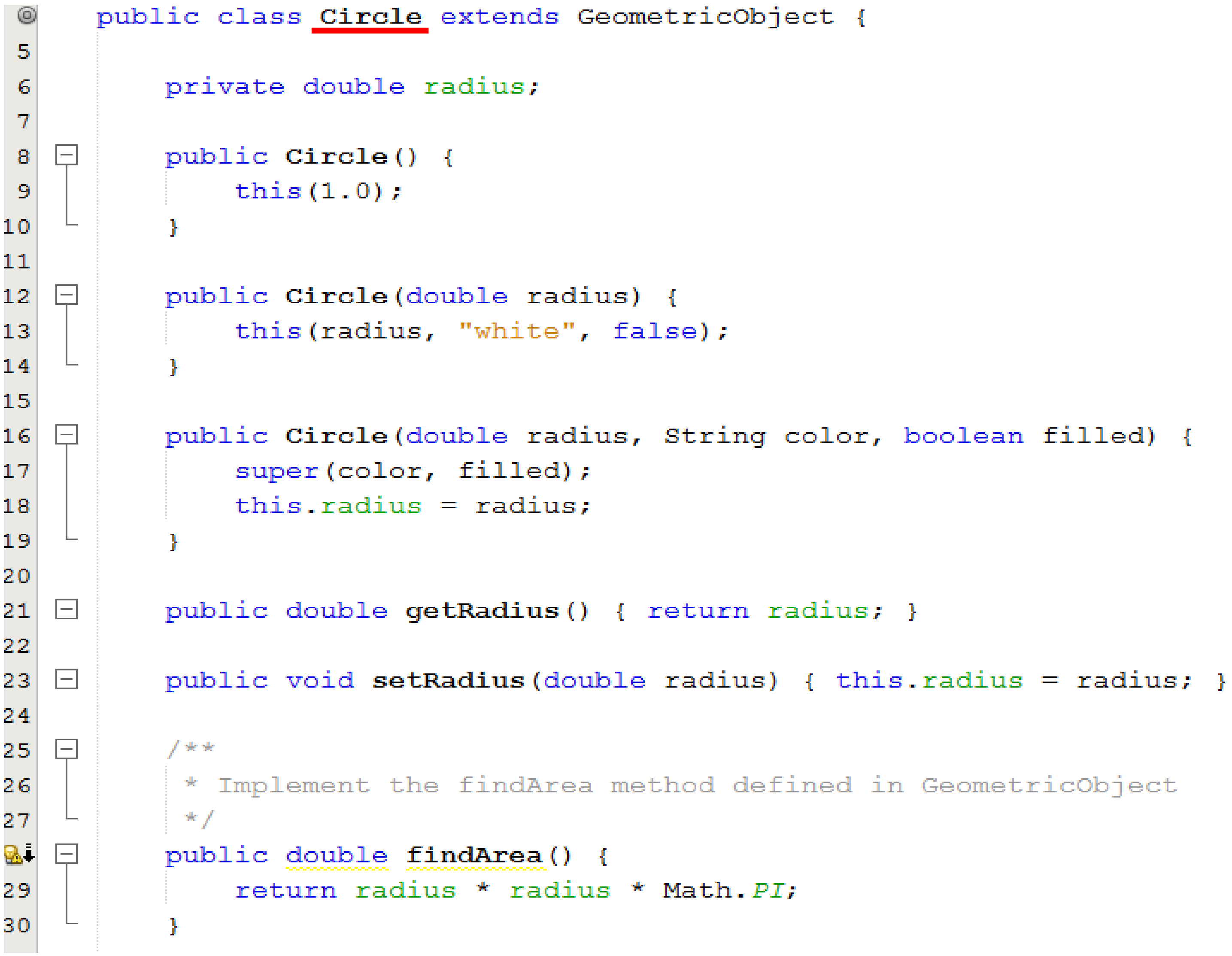
- The `GeometricObject` abstract class provides the common features (data and methods) for geometric objects.
- Because you don't know how to compute areas and perimeters of geometric objects, `findArea` and `findPerimeter` are defined as abstract methods.
- These methods are implemented in the subclasses.

Abstract Classes - Example



```java
public abstract class GeometricObject {

  private String color = "white";
  private boolean filled;

  /**Default constructor*/
  protected GeometricObject() {
  }

  /**Construct a geometric object*/
  protected GeometricObject(String color, boolean filled) {
    this.color = color;
    this.filled = filled;
  }

  public String getColor() { return color; }

  public void setColor(String color) { this.color = color; }

  public boolean isFilled() { return filled; }

  public void setFilled(boolean filled) { this.filled = filled; }

  /**Abstract method findArea*/
  public abstract double findArea();

  /**Abstract method findPerimeter*/
  public abstract double findPerimeter();

}//end class
```

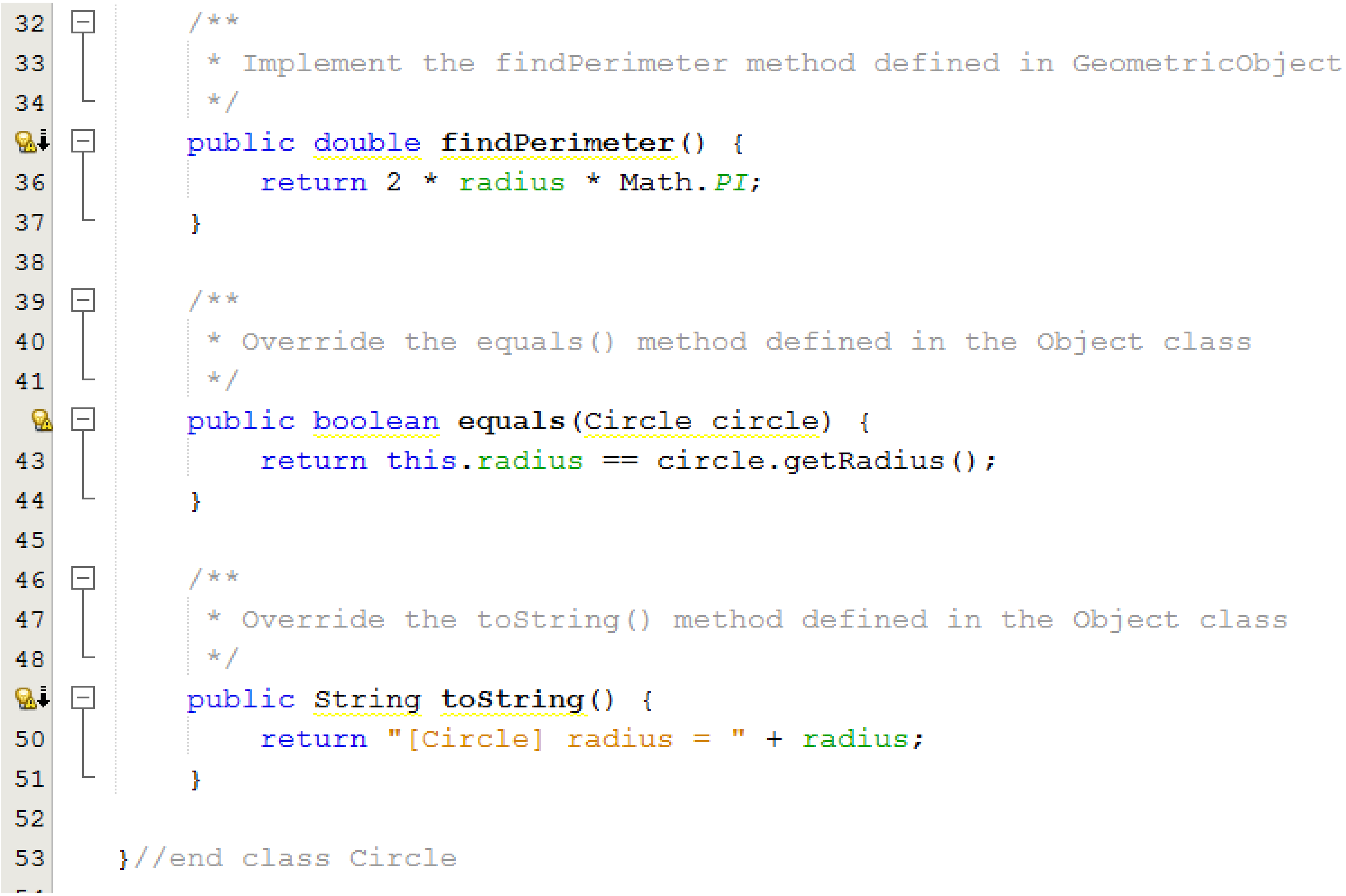
Abstract Classes - Example



```java
public class Circle extends GeometricObject {

    private double radius;

    public Circle() {
        this(1.0);
    }

    public Circle(double radius) {
        this(radius, "white", false);
    }

    public Circle(double radius, String color, boolean filled) {
        super(color, filled);
        this.radius = radius;
    }

    public double getRadius() { return radius; }

    public void setRadius(double radius) { this.radius = radius; }

    /**
     * Implement the findArea method defined in GeometricObject
     */
    public double findArea() {
        return radius * radius * Math.PI;
    }
```

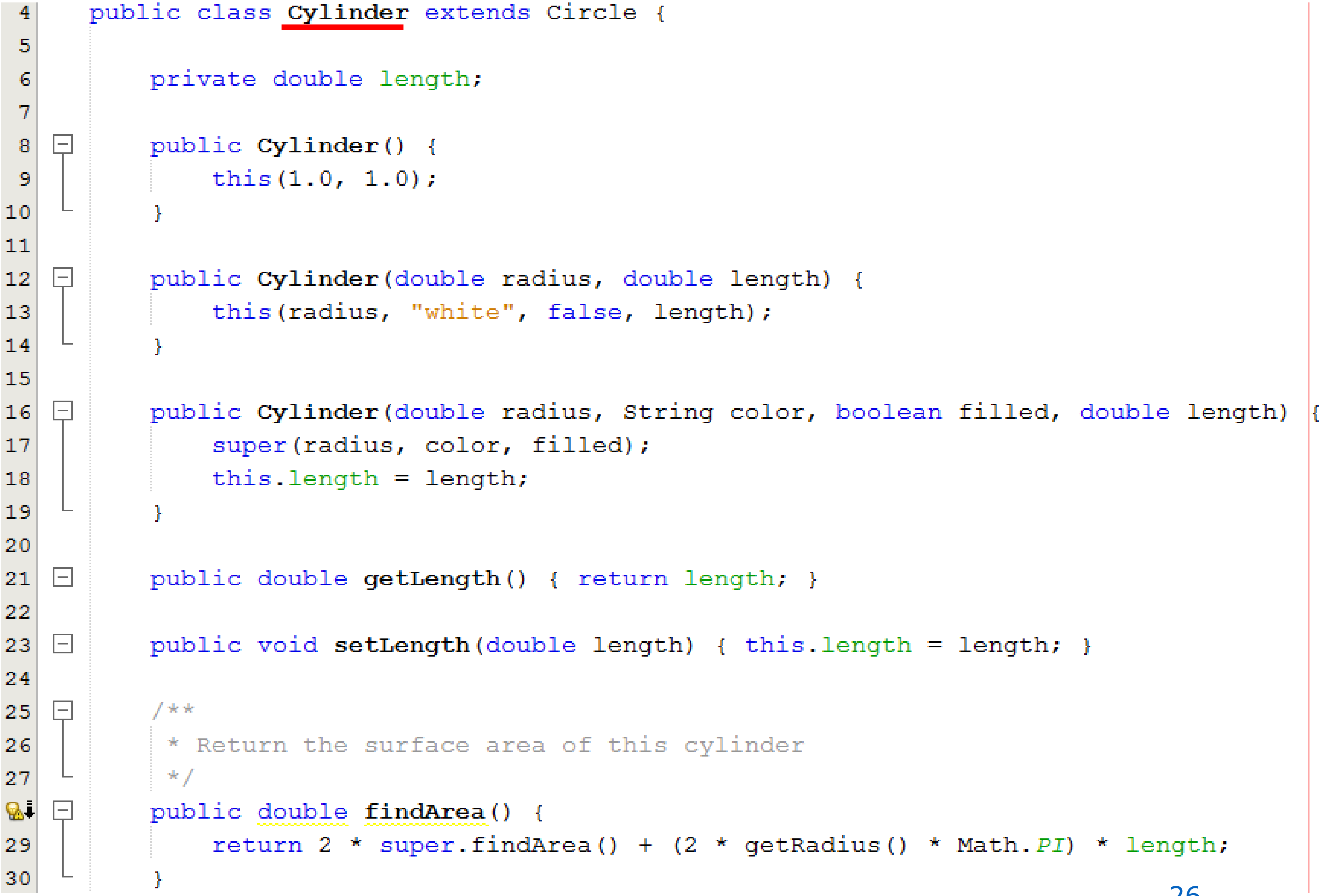
Abstract Classes - Example

Circle class contd…



```
    /**
     * Implement the findPerimeter method defined in GeometricObject
     */
    public double findPerimeter() {
        return 2 * radius * Math.PI;
    }

    /**
     * Override the equals() method defined in the Object class
     */
    public boolean equals(Circle circle) {
        return this.radius == circle.getRadius();
    }

    /**
     * Override the toString() method defined in the Object class
     */
    public String toString() {
        return "[Circle] radius = " + radius;
    }

}//end class Circle
```

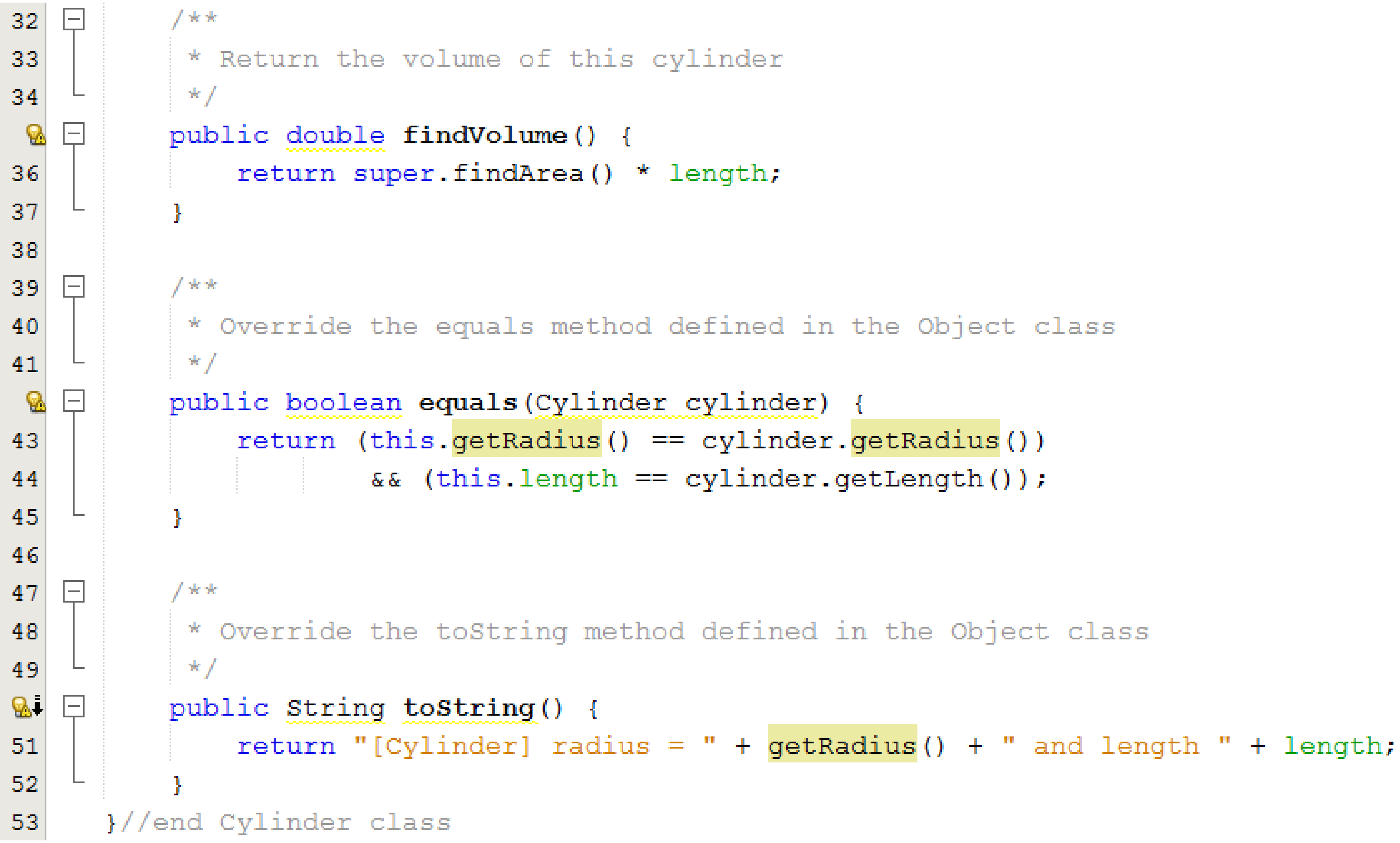
Abstract Classes - Example



```java
public class Cylinder extends Circle {

    private double length;

    public Cylinder() {
        this(1.0, 1.0);
    }

    public Cylinder(double radius, double length) {
        this(radius, "white", false, length);
    }

    public Cylinder(double radius, String color, boolean filled, double length) {
        super(radius, color, filled);
        this.length = length;
    }

    public double getLength() { return length; }

    public void setLength(double length) { this.length = length; }

    /**
     * Return the surface area of this cylinder
     */
    public double findArea() {
        return 2 * super.findArea() + (2 * getRadius() * Math.PI) * length;
    }
```

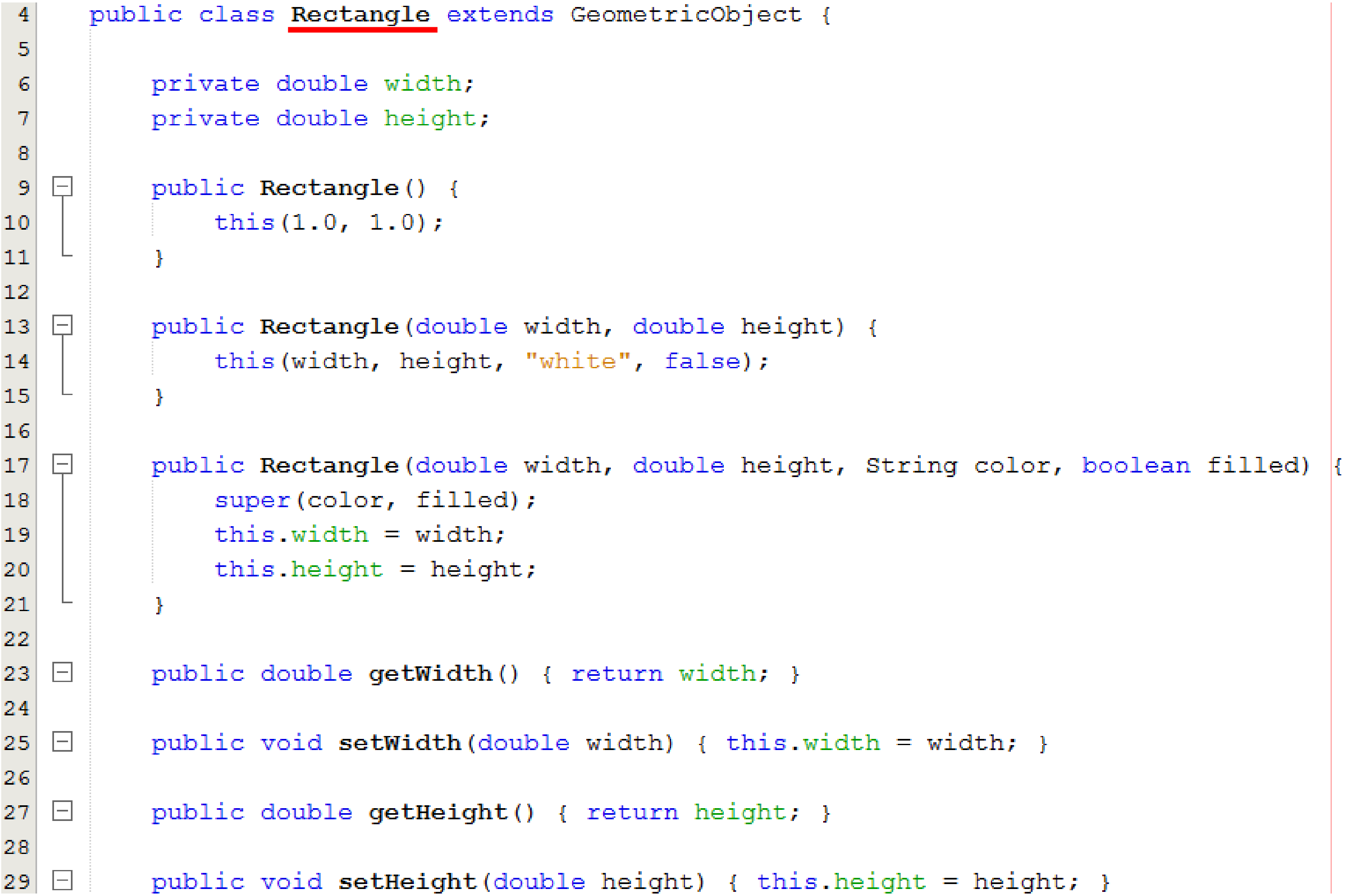
Abstract Classes - Example

Cylinder class contd…



```java
  /**
   * Return the volume of this cylinder
   */
  public double findVolume() {
    return super.findArea() * length;
  }

  /**
   * Override the equals method defined in the Object class
   */
  public boolean equals(Cylinder cylinder) {
    return (this.getRadius() == cylinder.getRadius())
        && (this.length == cylinder.getLength());
  }

  /**
   * Override the toString method defined in the Object class
   */
  public String toString() {
    return "[Cylinder] radius = " + getRadius() + " and length " + length;
  }
}//end Cylinder class
```

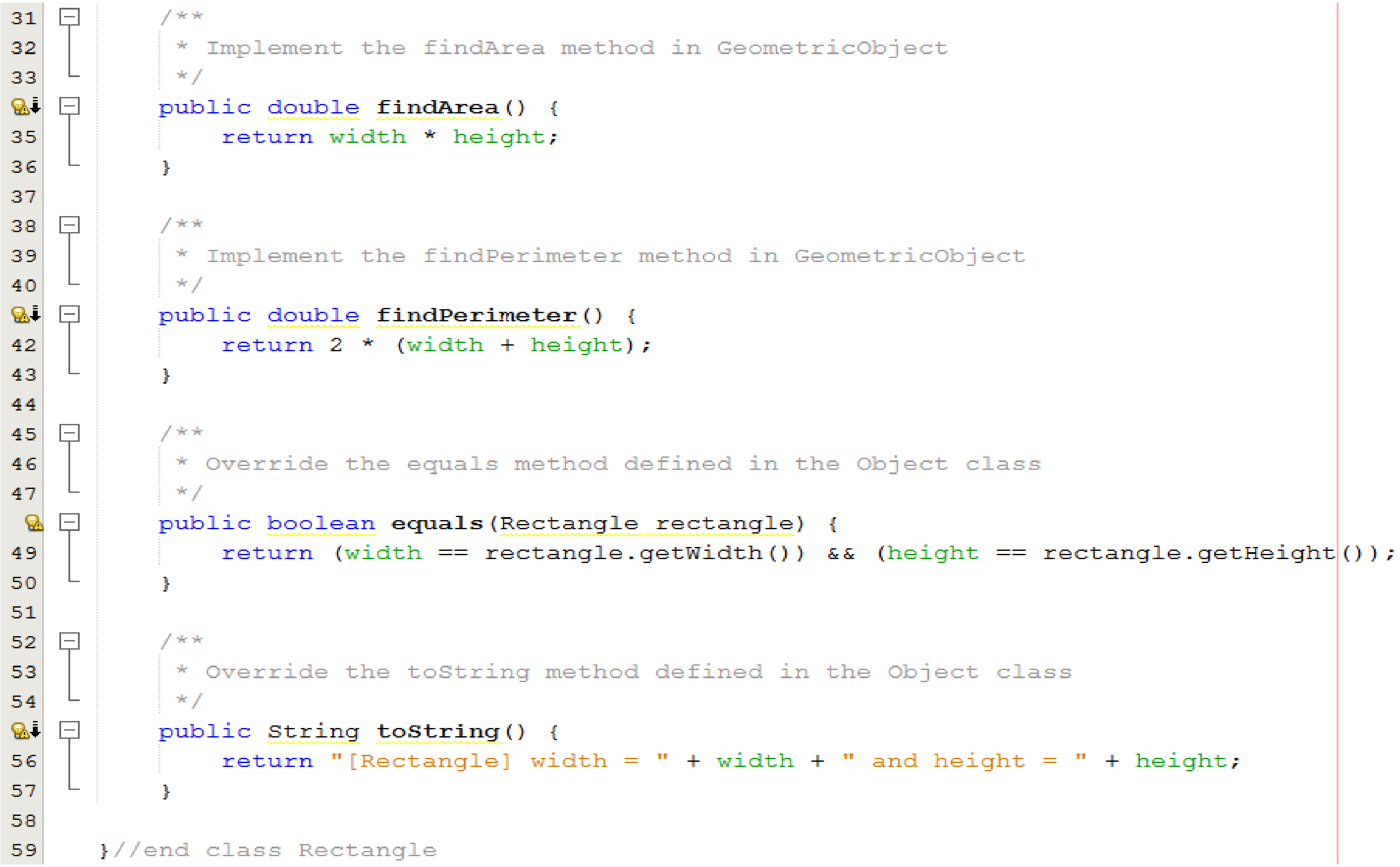
Abstract Classes - Example



```java
public class Rectangle extends GeometricObject {

    private double width;
    private double height;

    public Rectangle() {
        this(1.0, 1.0);
    }

    public Rectangle(double width, double height) {
        this(width, height, "white", false);
    }

    public Rectangle(double width, double height, String color, boolean filled) {
        super(color, filled);
        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; }
```

Abstract Classes - Example

Rectangle class contd…



```java
    /**
     * Implement the findArea method in GeometricObject
     */
    public double findArea() {
        return width * height;
    }

    /**
     * Implement the findPerimeter method in GeometricObject
     */
    public double findPerimeter() {
        return 2 * (width + height);
    }

    /**
     * Override the equals method defined in the Object class
     */
    public boolean equals(Rectangle rectangle) {
        return (width == rectangle.getWidth()) && (height == rectangle.getHeight());
    }

    /**
     * Override the toString method defined in the Object class
     */
    public String toString() {
        return "[Rectangle] width = " + width + " and height = " + height;
    }

}//end class Rectangle
```

Polymorphism & Dynamic Binding

- An object of a subclass can be used by any code designed to work with an object of its superclass.
  - If a method expects a parameter of the `GeometricObject` type, you can invoke it with a `Circle` object.
- A `Circle` object can be used as both a `Circle` object and a `GeometricObject` object.
  - This is Polymorphism.
- A method may be defined in a superclass but is overridden in a subclass.
- The JVM will determine at runtime which implementation of the method is used on a particular call dynamically.
  - This is Dynamic Binding.

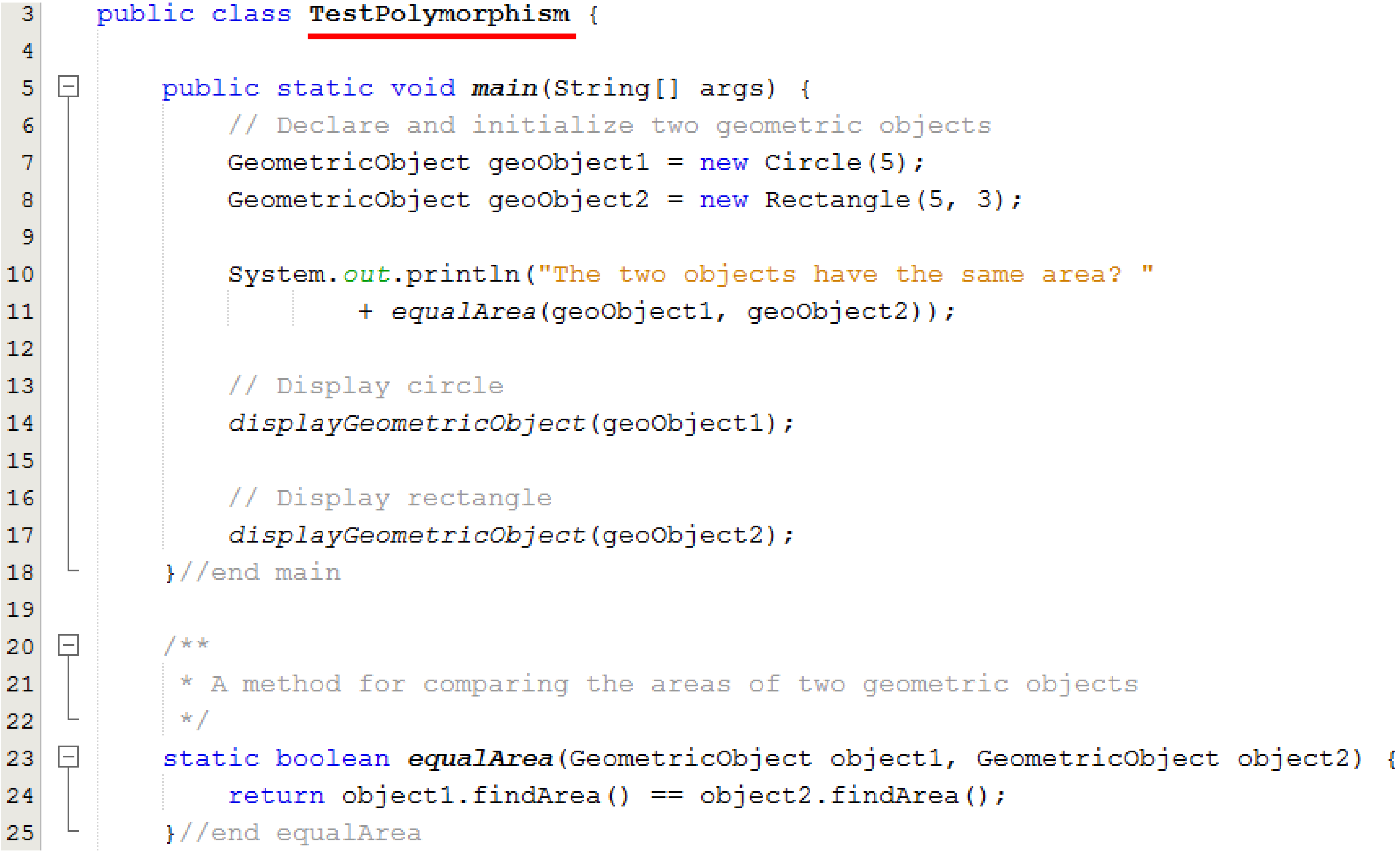
Polymorphism & Dynamic Binding

- Dynamic binding works as follows.
- Suppose an object o is an instance of classes c1, c2….cn-1 an cn; where c1 is a subclass of c2, c2 is a subclass of c3, and cn-1 is a subclass of cn. That is, cn is the most general class, and c1 must be the most specific class.
- cn is the Object class.
- If o invokes a method p, the JVM searches for the implementation of the method p in c1, c2…..cn-1, cn, in this order, until it is found.
- Once an implementation is found, the search stops and the first found implementation is invoked.

Polymorphism & Dynamic Binding -Example

- This example demonstrates polymorphism and dynamic binding.
- It creates two `GeometricObject`s, a `Circle` and a `Rectangle` and invokes the `equalArea` method to check whether the two objects have equal areas, and invokes the `dsiplayGeometricObject` method to display the objects.

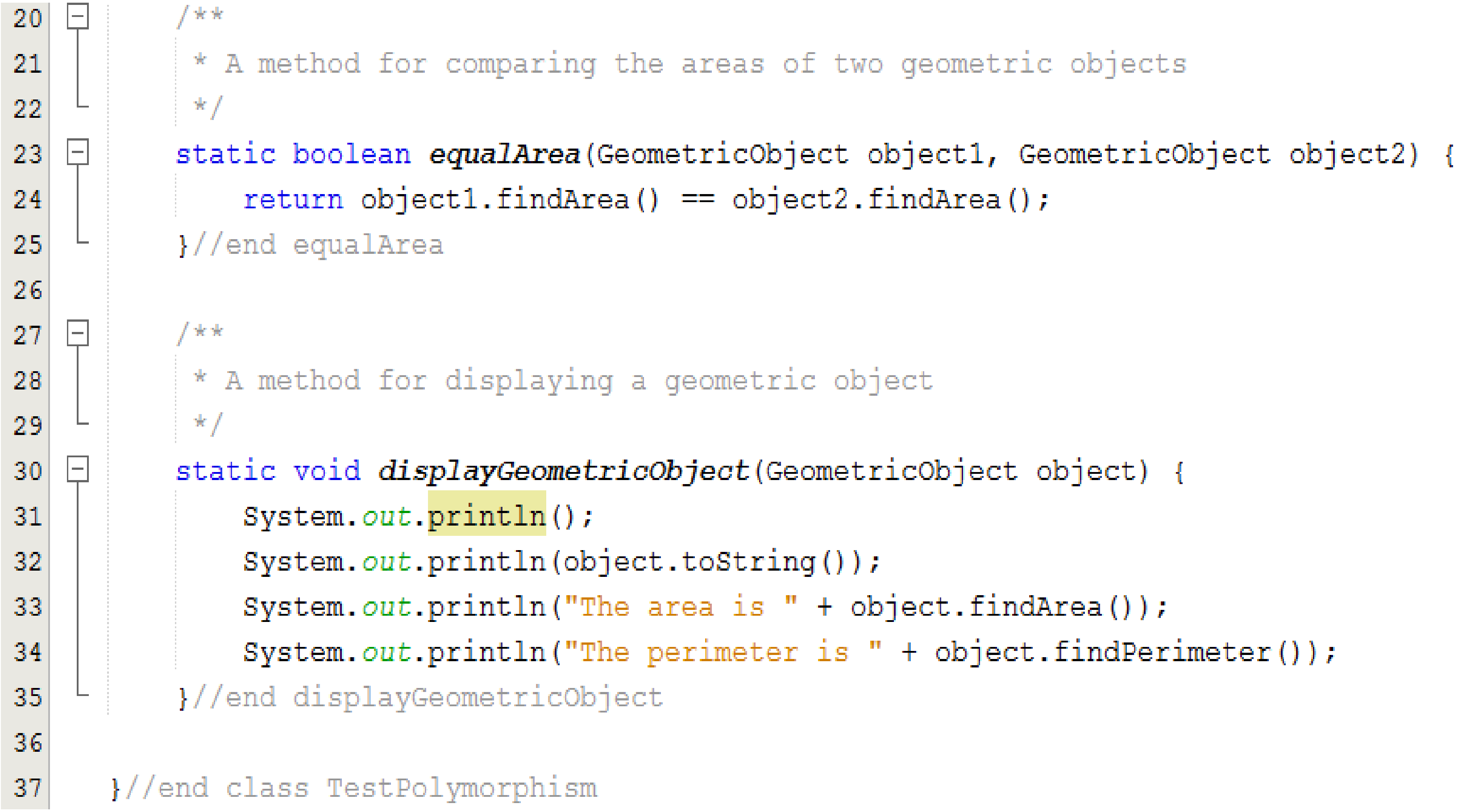
Polymorphism & Dynamic Binding -Example



```java
public class TestPolymorphism {

    public static void main(String[] args) {
        // Declare and initialize two geometric objects
        GeometricObject geoObject1 = new Circle(5);
        GeometricObject geoObject2 = new Rectangle(5, 3);

        System.out.println("The two objects have the same area? "
                + equalArea(geoObject1, geoObject2));

        // Display circle
        displayGeometricObject(geoObject1);

        // Display rectangle
        displayGeometricObject(geoObject2);
    }//end main

    /**
     * A method for comparing the areas of two geometric objects
     */
    static boolean equalArea(GeometricObject object1, GeometricObject object2) {
        return object1.findArea() == object2.findArea();
    }//end equalArea
```

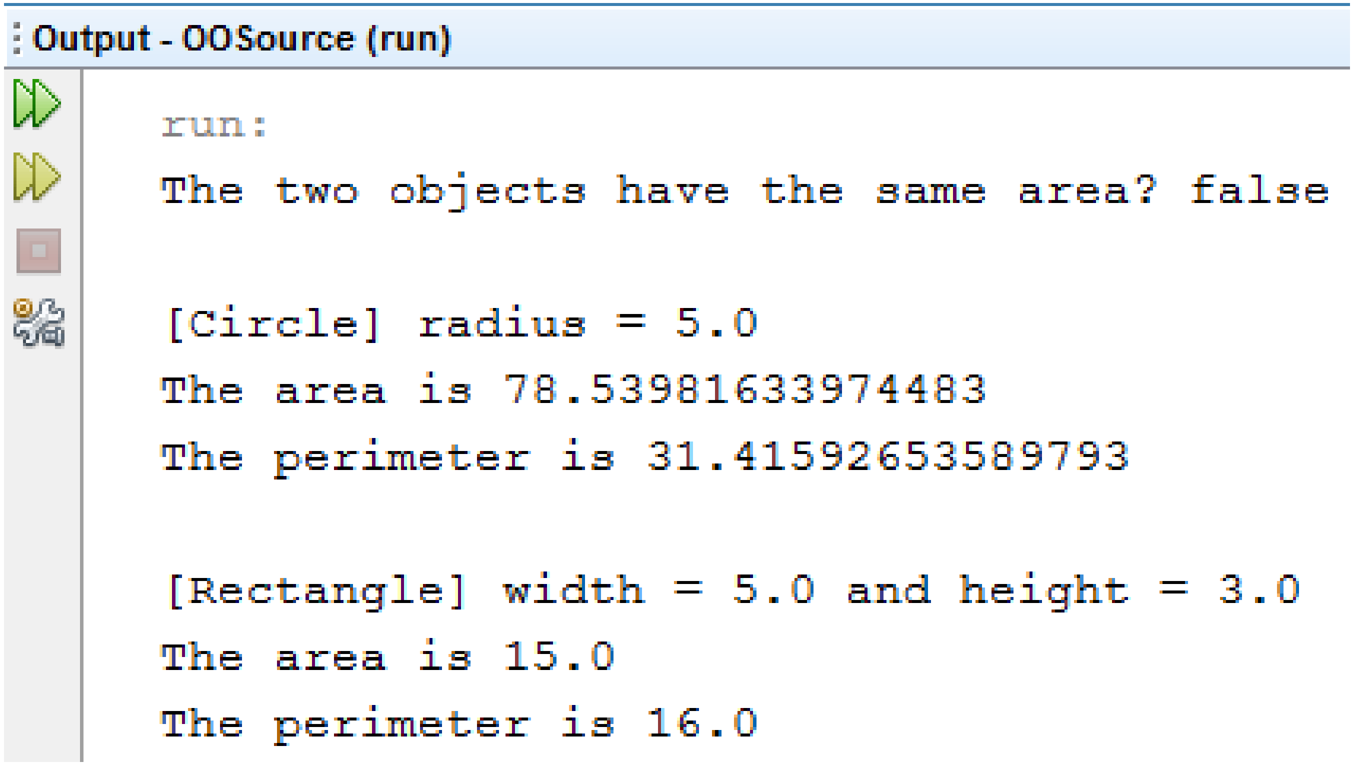
Polymorphism & Dynamic Binding -Example

TestPolymorphsim class contd…



```java
    /**
     * A method for comparing the areas of two geometric objects
     */
    static boolean equalArea(GeometricObject object1, GeometricObject object2) {
        return object1.findArea() == object2.findArea();
    }//end equalArea

    /**
     * A method for displaying a geometric object
     */
    static void displayGeometricObject(GeometricObject object) {
        System.out.println();
        System.out.println(object.toString());
        System.out.println("The area is " + object.findArea());
        System.out.println("The perimeter is " + object.findPerimeter());
    }//end displayGeometricObject

}//end class TestPolymorphism
```

Polymorphism & Dynamic Binding -Example



Output - OOSource (run)

```
run:
The two objects have the same area? false

[Circle] radius = 5.0
The area is 78.53981633974483
The perimeter is 31.41592653589793

[Rectangle] width = 5.0 and height = 3.0
The area is 15.0
The perimeter is 16.0
```

Polymorphism & Dynamic Binding -Example

- The statements

```
GeometricObject geoObject1 =
                new Circle(5);
GeometricObject geoObject2 =
            new Rectangle(5, 3);
```

- Create a new circle and rectangle, and assign them to variables geoObject1 and geoObject2. These two variables are of the `GeometricObject` type (class).
- These assignments are known as implicit casting, are legal since both circles and rectangles are geometric objects.

Polymorphism & Dynamic Binding -Example

- Similarly, when invoking `displayGeometricObject(geoObject1)`, the methods `findArea` and `findPerimeter`, and `toString` defined in the `Circle` class are used, and when invoking `displayGeometricObject(geoObject2)`, the methds `findArea`, `findPerimeter`, and `toString` defined in the `Rectangle` class are used.
- Which of these methods are invoked is dynamically determined at runtime, depending on the type of the object.

Casting Objects and the Instanceof Operator

- You have already used casting before, whereby you convert objects of one type to objects of another.
- The statement

```
GeometricObject geoObject1 =
                 new Circle(5);
```

is known as implicit casting, it assigns a circle to a variable of the `GeometricObject` type.

Casting Objects and the Instanceof Operator

- To perform explicit casting, use a syntax similar to the one used for casting among primitive data types.
- Enclose the target object type in parenthesis and place it before the object to be cast.

```
Circle myCircle =
       (Circle)myCylinder;
Cylinder myCylinder =
           (Cylinder)myCircle;
```

Casting Objects and the Instanceof Operator

- The first statement converts `myCylinder` to its superclass variable `myCircle`; the second converts `myCircle` to its subclass variable `myCylinder`.

Casting Objects and the Instanceof Operator

- It is always possible to convert an instance of a subclass to an instance of a superclass, because an instance of a subclass is also an instance of its superclass.
- For example, an apple is an instance of the Apple class, which is a subclass of the Fruit class. An apple is always a fruit. Therefore, you can always assign an apple to a variable of the Fruit class.
- For this reason, explicit casting can be omitted in this case.
- Thus,

```
Circle myCircle = myCylinder;
```

Casting Objects and the Instanceof Operator

- is equilivent to:

```
Cylinder myCylinder =
     (Cylinder)myCircle;
```

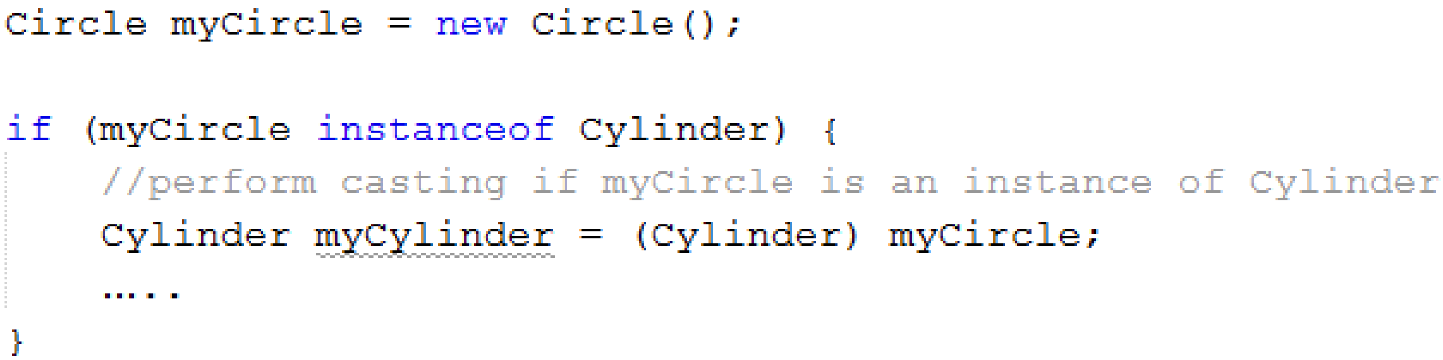
- When converting an instance of a superclass to an instance of its subclass, explicit casting must be used to confirm your intention to the compiler with the (Subclass Name) cast notation.
- For the casting to be successful, you must make sure that the object to be cast is an instance of the subclass.

Casting Objects and the Instanceof Operator

- If the superclass object is not an instance of the subclass, a runtime exception occurs.
- For example, if the fruit is an orange, an instance of the Fruit class cannot be cast into an instance of the Apple class.
- It is good practice, therefore, to ensure that the object is an instance of another object before attempting a casting.

Casting Objects and the Instanceof Operator

- This can be accomplished by using the `instanceof` operator. Consider the following code:



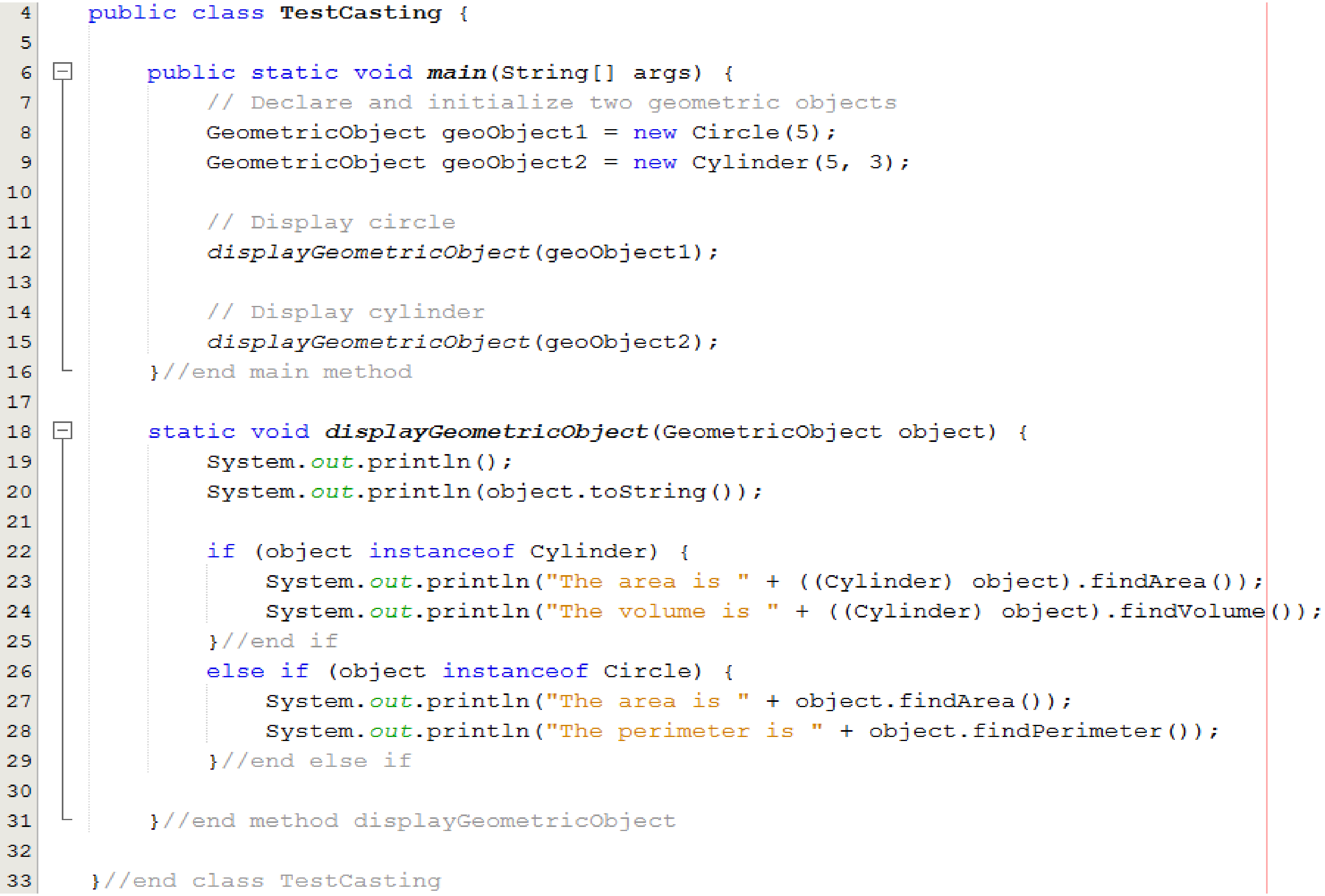
```
Circle myCircle = new Circle();

if (myCircle instanceof Cylinder) {
    //perform casting if myCircle is an instance of Cylinder
    Cylinder myCylinder = (Cylinder) myCircle;
    .....
}
```

- You may be wondering how `myCircle` could become an instance of the Cylinder class and why it is necessary to perform casting.

Casting Objects and the Instanceof Operator

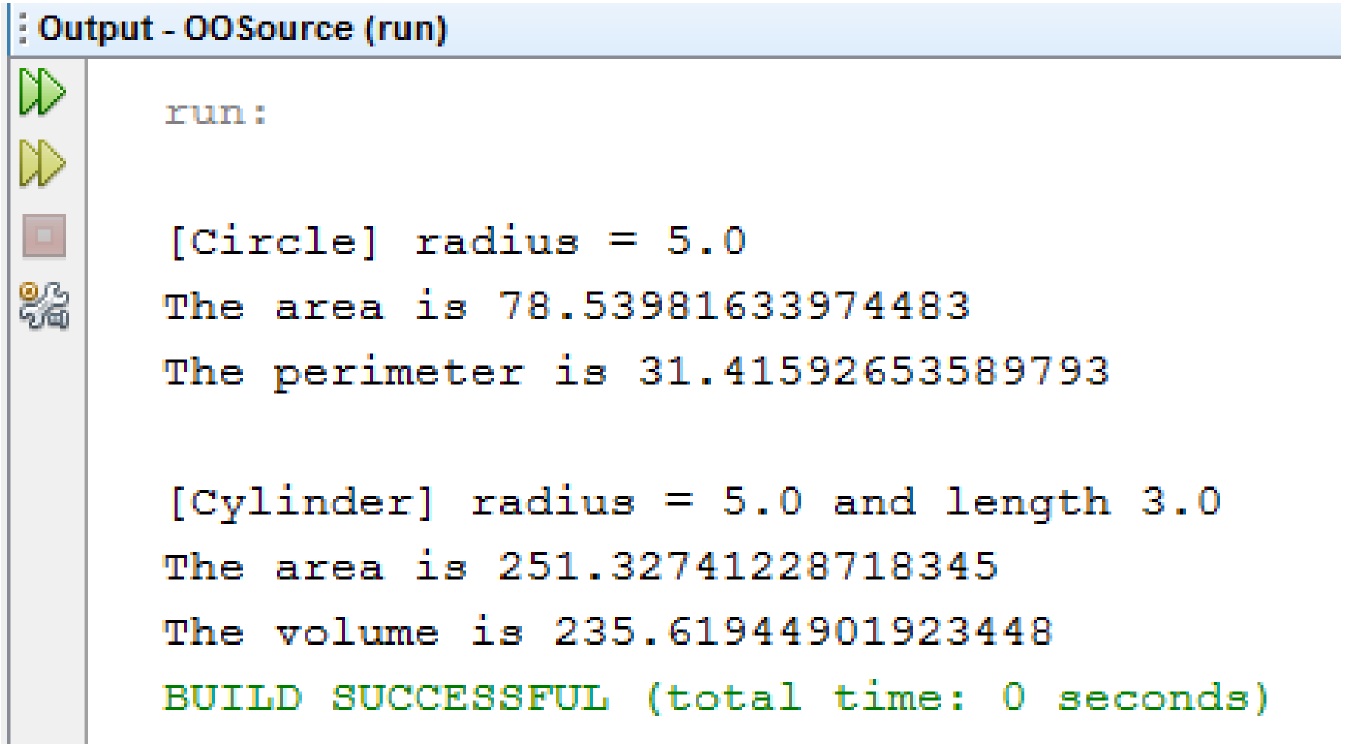
- There are some cases in which a variable of a superclass holds an instance of a subclass.
- To fully explore the properties and functions, you need to cast the object to its subclass.
- This is shown in the following example.
- This example creates two geometric objects, a circle and a cylinder, and invokes the `displayGeometricObject` method to display them. The `displayGeometricObject` method displays area and perimeter if the object is a circle, and area and volume if the object is a cylinder.

Casting Objects and the Instanceof Operator - Example



```java
public class TestCasting {

    public static void main(String[] args) {
        // Declare and initialize two geometric objects
        GeometricObject geoObject1 = new Circle(5);
        GeometricObject geoObject2 = new Cylinder(5, 3);

        // Display circle
        displayGeometricObject(geoObject1);

        // Display cylinder
        displayGeometricObject(geoObject2);
    }//end main method

    static void displayGeometricObject(GeometricObject object) {
        System.out.println();
        System.out.println(object.toString());

        if (object instanceof Cylinder) {
            System.out.println("The area is " + ((Cylinder) object).findArea());
            System.out.println("The volume is " + ((Cylinder) object).findVolume());
        }//end if
        else if (object instanceof Circle) {
            System.out.println("The area is " + object.findArea());
            System.out.println("The perimeter is " + object.findPerimeter());
        }//end else if

    }//end method displayGeometricObject

}//end class TestCasting
```

Casting Objects and the Instanceof Operator - Example



```
run:

[Circle] radius = 5.0
The area is 78.53981633974483
The perimeter is 31.41592653589793

[Cylinder] radius = 5.0 and length 3.0
The area is 251.32741228718345
The volume is 235.61944901923448
BUILD SUCCESSFUL (total time: 0 seconds)
```

Interfaces

- Classes group similar objects together, and inheritance groups similar classes together.
- You might like to group together objects that are related because they all play a particular role.
  - Group together classes that model humans, birds, ants & robots, because they all walk.
  - Classes may not have anything else in common.
  - Not appropriate to give them a single superclass, because there isn’t a useful category that contains them all, but they do share a particular ability.
  - A mechanism exists for modelling such a situation and its called an interface.

Interfaces

- Sometimes it is necessary to derive a subclass from several classes, thus inheriting their data and methods.
- Java, does not allow multiple inheritance.
- If you use the extends keyword to define a subclass, it allows only one parent class.
- With interfaces, you can obtain the effect of multiple inheritance.
- An interface is a “classlike” construct that contains only constants and abstract methods.
- Similar to an abstract class, but an abstract class can contain constants and abstract methods as well as variables and concrete methods.

Interfaces

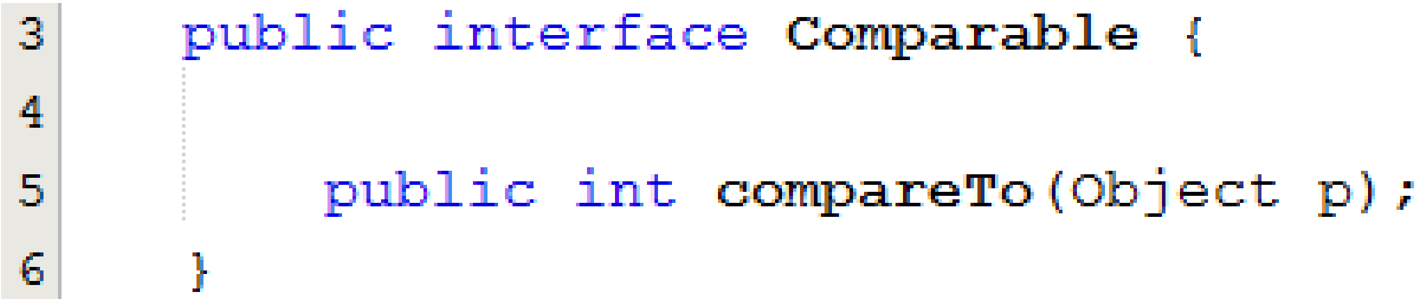
- Each interface is compiled into a separate bytecode file, just like a regular class.
- As with an abstract class, you cannot create an instance for the interface using the new operator.
- You can use an interface more or less the same way you use an abstract class.
- For example, you can use an interface as a data type for a variable, as the result of casting etc.

Interfaces - Example

- You want to design a generic method to find the larger of two objects.
- The objects can be students, circles, cylinders or Bank Accounts.
- Compare methods are different for different types of objects, you need a generic compare method to determine the order of the two objects.
- Then you can tailor the method to compare students, circles, or cylinders.
  - For example, you can use student ID as the key for comparing students, radius as the key for comparing circles, and volume as the key for comparing cylinders.

Interfaces - Example

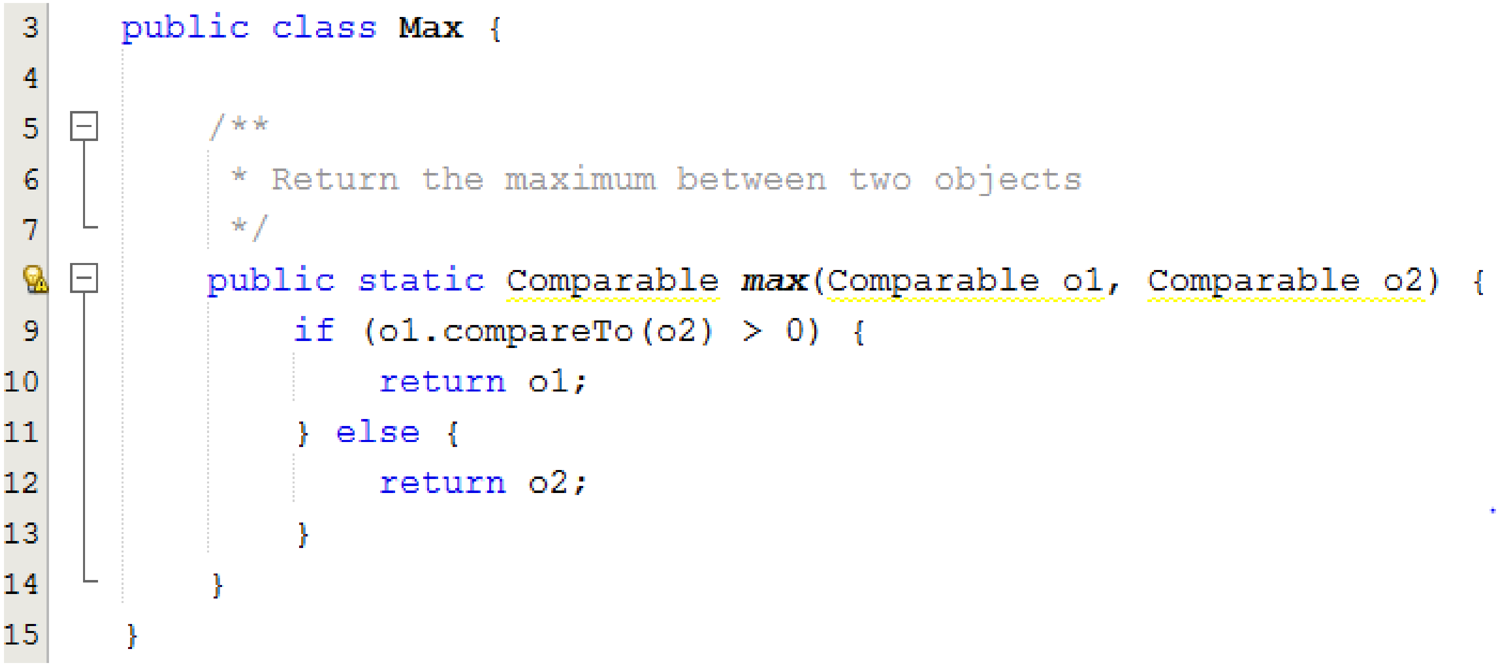
- You can use an interface to define a generic `compareTo` method, as follows:



```java
public interface Comparable {

    public int compareTo(Object p);
}
```

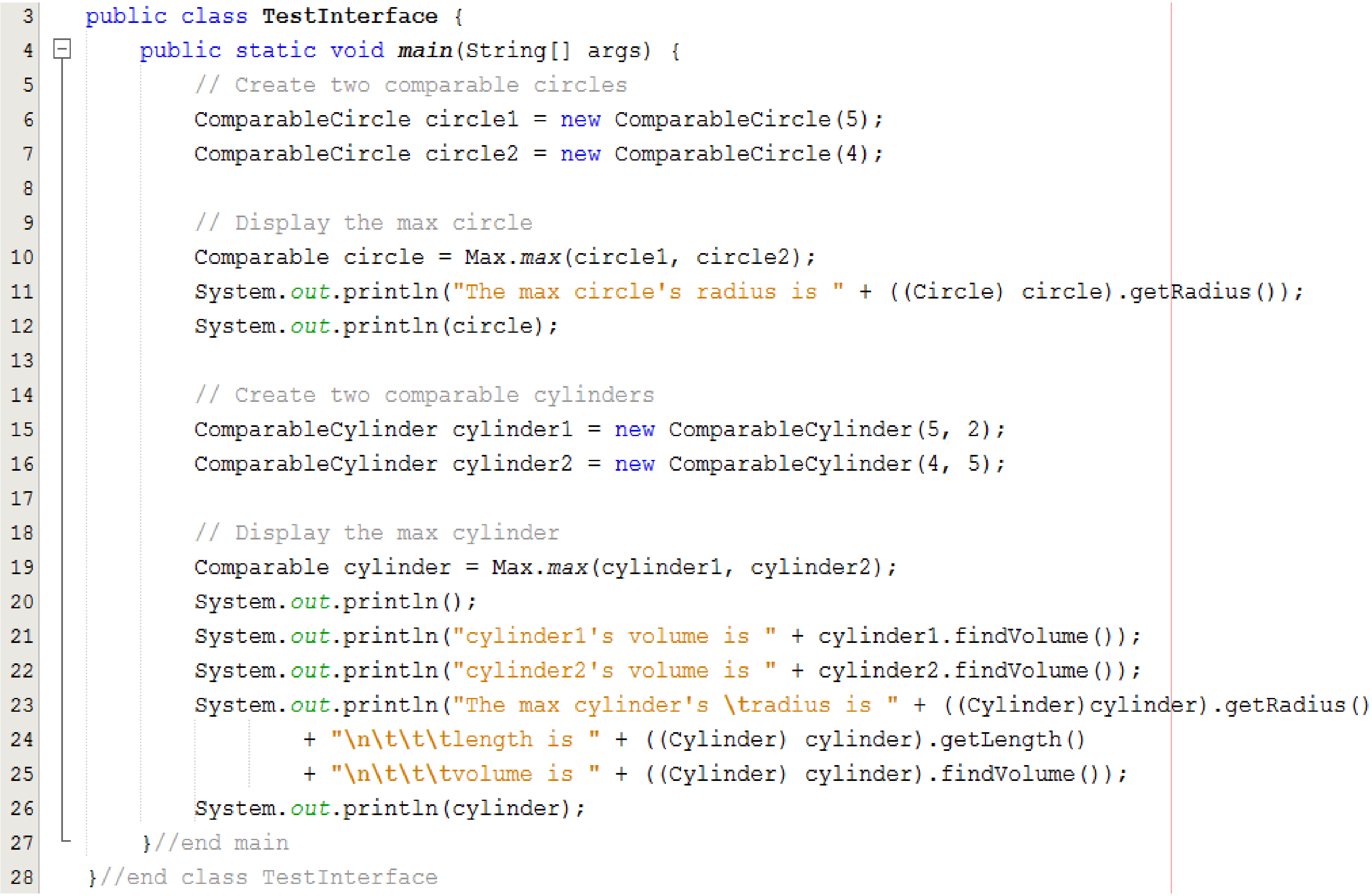
Interfaces - Example

- The `compareTo` method determines the order of this object with the specified object o, and returns a negative integer, zero, or a positive integer if this object is less than, equal to, or greater than the specified object o.



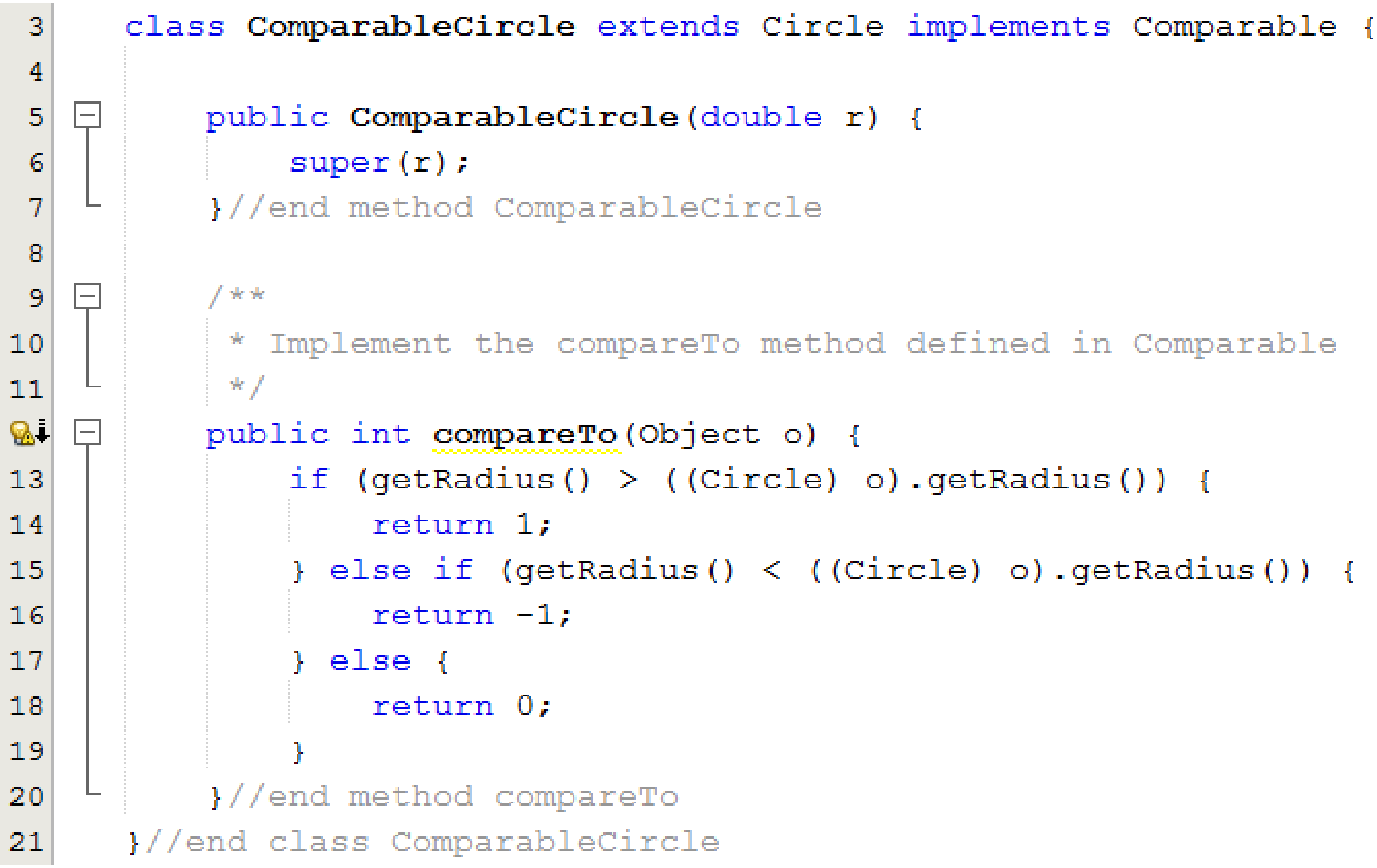
```java
public class Max {

    /**
     * Return the maximum between two objects
     */
    public static Comparable max(Comparable o1, Comparable o2) {
        if (o1.compareTo(o2) > 0) {
            return o1;
        } else {
            return o2;
        }
    }
}
```

Interfaces - Example



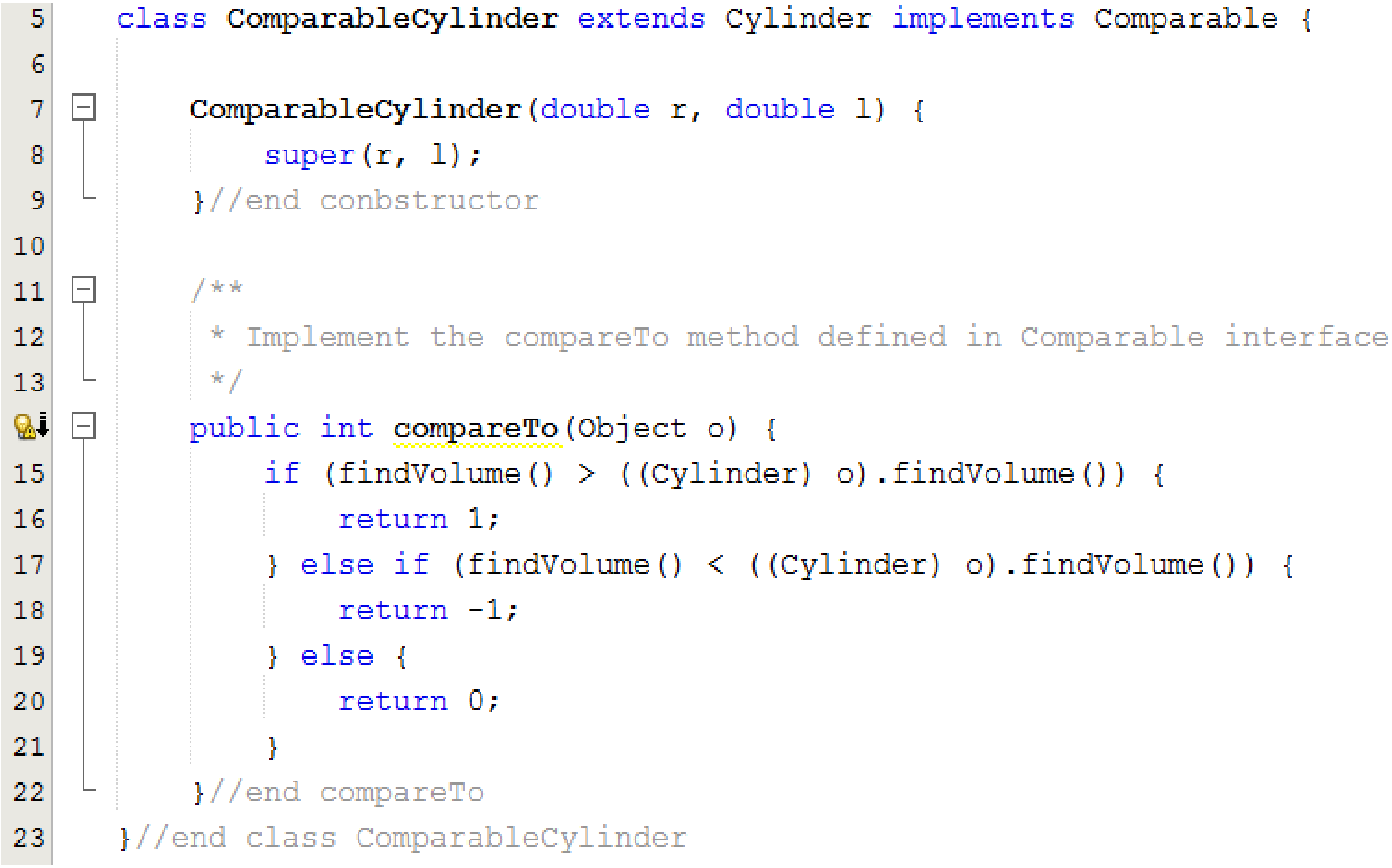
```
public class TestInterface {
    public static void main(String[] args) {
        // Create two comparable circles
        ComparableCircle circle1 = new ComparableCircle(5);
        ComparableCircle circle2 = new ComparableCircle(4);

        // Display the max circle
        Comparable circle = Max.max(circle1, circle2);
        System.out.println("The max circle's radius is " + ((Circle) circle).getRadius());
        System.out.println(circle);

        // Create two comparable cylinders
        ComparableCylinder cylinder1 = new ComparableCylinder(5, 2);
        ComparableCylinder cylinder2 = new ComparableCylinder(4, 5);

        // Display the max cylinder
        Comparable cylinder = Max.max(cylinder1, cylinder2);
        System.out.println();
        System.out.println("cylinder1's volume is " + cylinder1.findVolume());
        System.out.println("cylinder2's volume is " + cylinder2.findVolume());
        System.out.println("The max cylinder's \tradius is " + ((Cylinder)cylinder).getRadius()
                + "\n\t\t\tlength is " + ((Cylinder) cylinder).getLength()
                + "\n\t\t\tvolume is " + ((Cylinder) cylinder).findVolume());
        System.out.println(cylinder);
    }//end main
}//end class TestInterface
```

Interfaces - Example



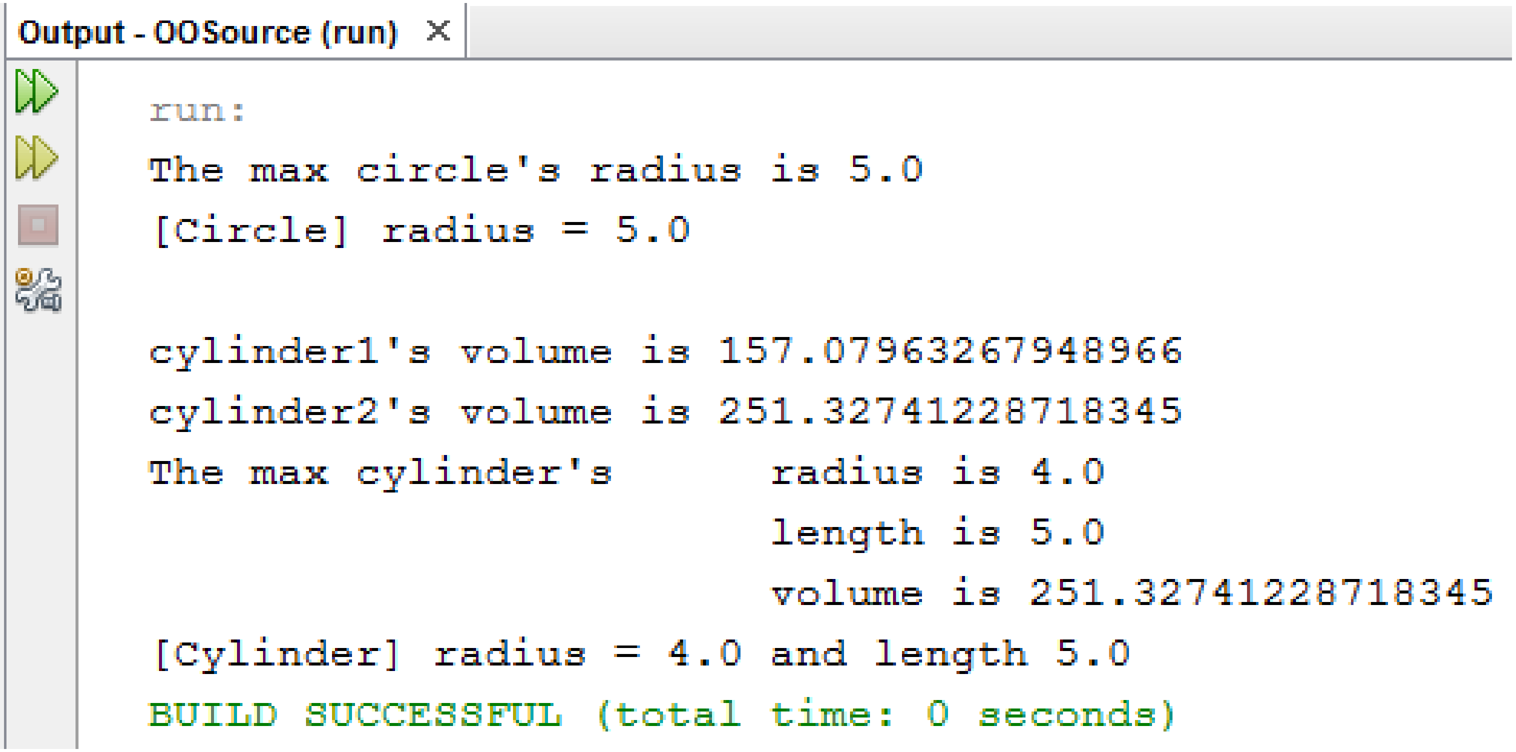
```java
class ComparableCircle extends Circle implements Comparable {

    public ComparableCircle(double r) {
        super(r);
    }//end method ComparableCircle

    /**
     * Implement the compareTo method defined in Comparable
     */
    public int compareTo(Object o) {
        if (getRadius() > ((Circle) o).getRadius()) {
            return 1;
        } else if (getRadius() < ((Circle) o).getRadius()) {
            return -1;
        } else {
            return 0;
        }
    }//end method compareTo
}//end class ComparableCircle
```

Interfaces - Example



```
class ComparableCylinder extends Cylinder implements Comparable {

    ComparableCylinder(double r, double l) {
        super(r, l);
    }//end conbstructor

    /**
     * Implement the compareTo method defined in Comparable interface
     */
    public int compareTo(Object o) {
        if (findVolume() > ((Cylinder) o).findVolume()) {
            return 1;
        } else if (findVolume() < ((Cylinder) o).findVolume()) {
            return -1;
        } else {
            return 0;
        }
    }//end compareTo
}//end class ComparableCylinder
```

Interfaces - Example



```
run:
The max circle's radius is 5.0
[Circle] radius = 5.0

cylinder1's volume is 157.07963267948966
cylinder2's volume is 251.32741228718345
The max cylinder's      radius is 4.0
                        length is 5.0
                        volume is 251.32741228718345
[Cylinder] radius = 4.0 and length 5.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

References

Paul J Deitel (2016) *Java How To Program.* 10/E. ISBN-13 9780134800271 (Link)

Y. Daniel Liang (2014) *Intro to Java Programming and Data Structures, Comprehensive Version.* 11/E. Pearson. ISBN-13 978-0134670942 (Link)