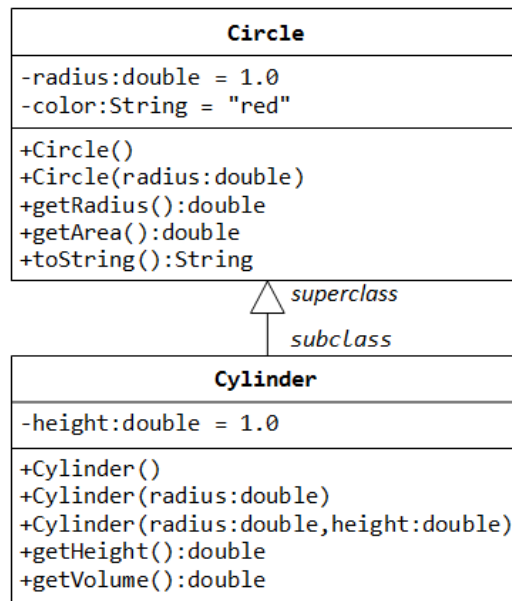


1. Exercise 1: The Circle and Cylinder Class

In this exercise, a subclass called Cylinder is derived from the superclass Circle as shown in the class diagram (where an arrow pointing up from the subclass to its superclass). Study how the subclass Cylinder invokes the superclass' constructors (via `super()` and `super(radius)`) and inherits the variables and methods from the superclass Circle.



You can reuse the Circle class that you have created in the previous exercise. Make sure that you keep "Circle.java" in the same directory.

```
public class Cylinder extends Circle { //save as "Cylinder.java"
    private double height; // private variable

    // Constructor with default color, radius and height
    public Cylinder() {
        super(); // call superclass no-arg constructor Circle()
        height = 1.0;
    }
    // Constructor with default radius, color but given height
    public Cylinder(double height) {
        super(); // call superclass no-arg constructor Circle()
        this.height = height;
    }
    // Constructor with default color, but given radius, height
    public Cylinder(double radius, double height) {
        super(radius); // call superclass constructor Circle(r)
        this.height = height;
    }
}
```

```

// A public method for retrieving the height
public double getHeight() {
    return height;
}

// A public method for computing the volume of cylinder
// use superclass method getArea() to get the base area
public double getVolume() {
    return getArea()*height;
}
}

```

Write a test program (says TestCylinder) to test the Cylinder class created, as follow:

```

public class TestCylinder { // save as "TestCylinder.java"
    public static void main (String[] args) {
        // Declare and allocate a new instance of cylinder
        // with default color, radius, and height
        Cylinder c1 = new Cylinder();
        System.out.println("Cylinder:"
            + " radius=" + c1.getRadius()
            + " height=" + c1.getHeight()
            + " base area=" + c1.getArea()
            + " volume=" + c1.getVolume());

        // Declare and allocate a new instance of cylinder
        // specifying height, with default color and radius
        Cylinder c2 = new Cylinder(10.0);
        System.out.println("Cylinder:"
            + " radius=" + c2.getRadius()
            + " height=" + c2.getHeight()
            + " base area=" + c2.getArea()
            + " volume=" + c2.getVolume());

        // Declare and allocate a new instance of cylinder
        // specifying radius and height, with default color
        Cylinder c3 = new Cylinder(2.0, 10.0);
        System.out.println("Cylinder:"
            + " radius=" + c3.getRadius()
            + " height=" + c3.getHeight()
            + " base area=" + c3.getArea()
            + " volume=" + c3.getVolume());
    }
}

```

Method Overriding and "Super": The subclass Cylinder inherits `getArea()` method from its superclass Circle. Try *overriding* the `getArea()` method in the subclass Cylinder to compute the surface area ($=2\pi \times \text{radius} \times \text{height} + 2 \times \text{base-area}$) of the cylinder instead of base area. That is, if `getArea()` is called by a Circle instance, it returns the area. If `getArea()` is called by a Cylinder instance, it returns the surface area of the cylinder.

If you override the `getArea()` in the subclass Cylinder, the `getVolume()` no longer works. This is because the `getVolume()` uses the *overridden* `getArea()` method found in the same class. (Java runtime will search the superclass only if it cannot locate the method in this class). Fix the `getVolume()`.

Hints: After overriding the `getArea()` in subclass Cylinder, you can choose to invoke the `getArea()` of the superclass Circle by calling `super.getArea()`.

TRY:

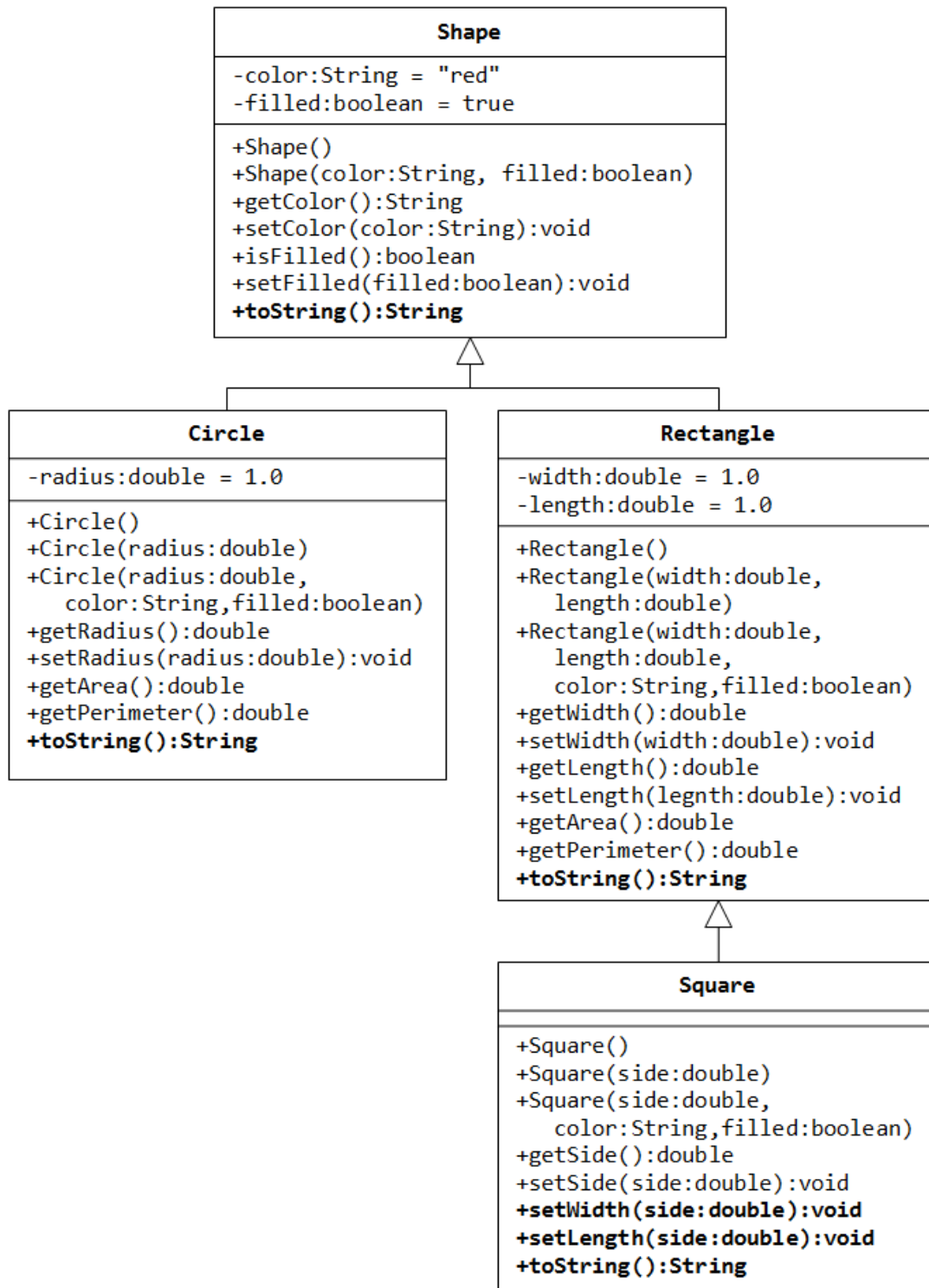
Provide a `toString()` method to the Cylinder class, which overrides the `toString()` inherited from the superclass Circle, e.g.,

```
@Override
public String toString() {    // in Cylinder class
    return "Cylinder: subclass of " + super.toString() // use Circle's toString()
        + " height=" + height;
}
```

Try out the `toString()` method in `TestCylinder`.

Note: `@Override` is known as *annotation* (introduced in JDK 1.5), which asks compiler to check whether there is such a method in the superclass to be overridden. This helps greatly if you misspell the name of the `toString()`. If `@Override` is not used and `toString()` is misspelled as `Tostring()`, it will be treated as a new method in the subclass, instead of overriding the superclass. If `@Override` is used, the compiler will signal an error. `@Override` annotation is optional, but certainly nice to have.

2. Exercise 2: Superclass Shape and its subclasses Circle, Rectangle and Square



Write a superclass called Shape (as shown in the class diagram), which contains:

- Two instance variables color (String) and filled (boolean).
- Two constructors: a no-arg (no-argument) constructor that initializes the color to "red" and filled to true, and a constructor that initializes the color and filled to the given values.
- Getter and setter for all the instance variables. By convention, the getter for a boolean variable xxx is called isXXX() (instead of getXxx() for all the other types).
- A toString() method that returns "A Shape with color of xxx and filled/Not filled".

Write a test program to test all the methods defined in Shape.

Write two subclasses of Shape called Circle and Rectangle, as shown in the class diagram.

The Circle class contains:

- An instance variable radius (double).
- Three constructors as shown. The no-arg constructor initializes the radius to 1.0.
- Getter and setter for the instance variable radius.
- Methods getArea() and getPerimeter().
- Override the toString() method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

The Rectangle class contains:

- Two instance variables width (double) and length (double).
- Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.
- Getter and setter for all the instance variables.
- Methods getArea() and getPerimeter().
- Override the toString() method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

Write a class called Square, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram). Hint:

```
public Square(double side) {
    super(side, side); // Call superclass Rectangle(double, double)
}
```

- Override the toString() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.
- Do you need to override the getArea() and getPerimeter()? Try them out.
- Override the setLength() and setWidth() to change both the width and length, so as to maintain the square geometry.