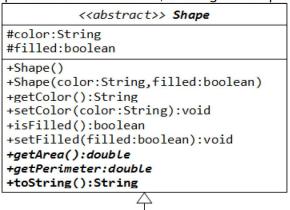
2.1 Exercise: Abstract Superclass Shape and Its Concrete Subclasses

Rewrite the superclass Shape and its subclasses Circle, Rectangle and Square, as shown in the class diagram.



+Square() +Square(side:double) +Square(side:double,color:String, filled:boolean) +getSide():double +setSide(side:double):void +setWidth(side:double):void +setLength(side:double):void +toString():String

In this exercise, Shape shall be defined as an abstract class, which contains:

- Two protected instance variables color (String) and filled (boolean). The protected variables can be accessed by its subclasses and classes in the same package. They are denoted with a "#" sign in the class diagram.
- Getter and setter for all the instance variables, and toString().
- Two abstract methods getArea() and getPerimeter() \{\shown in italics in the class diagram\}.

The subclasses Circle and Rectangle shall override the abstract methods getArea() and getPerimeter() and provide the proper implementation. They also override the toString().

Write a test class to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any.

```
Shape s1 = new Circle(5.5, "RED", false); // Upcast Circle to Shape
System.out.println(s1);
                                           // which version?
System.out.println(s1.getArea());
                                           // which version?
System.out.println(s1.getPerimeter());
                                           // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());
Circle c1 = (Circle)s1;
                                          // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());
Shape s2 = new Shape();
Shape s3 = new Rectangle(1.0, 2.0, "RED", false);
                                                    // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());
Rectangle r1 = (Rectangle)s3; // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());
Shape s4 = new Square(6.6);
                                // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());
// Take note that we downcast Shape s4 to Rectangle,
// which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());
// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

What is the usage of the abstract method and abstract class