Exercise Three: Inheritance, Polymorphism, Interface

1. (30 Points) Superclass Shape and its subclasses Circle, Rectangle and Square

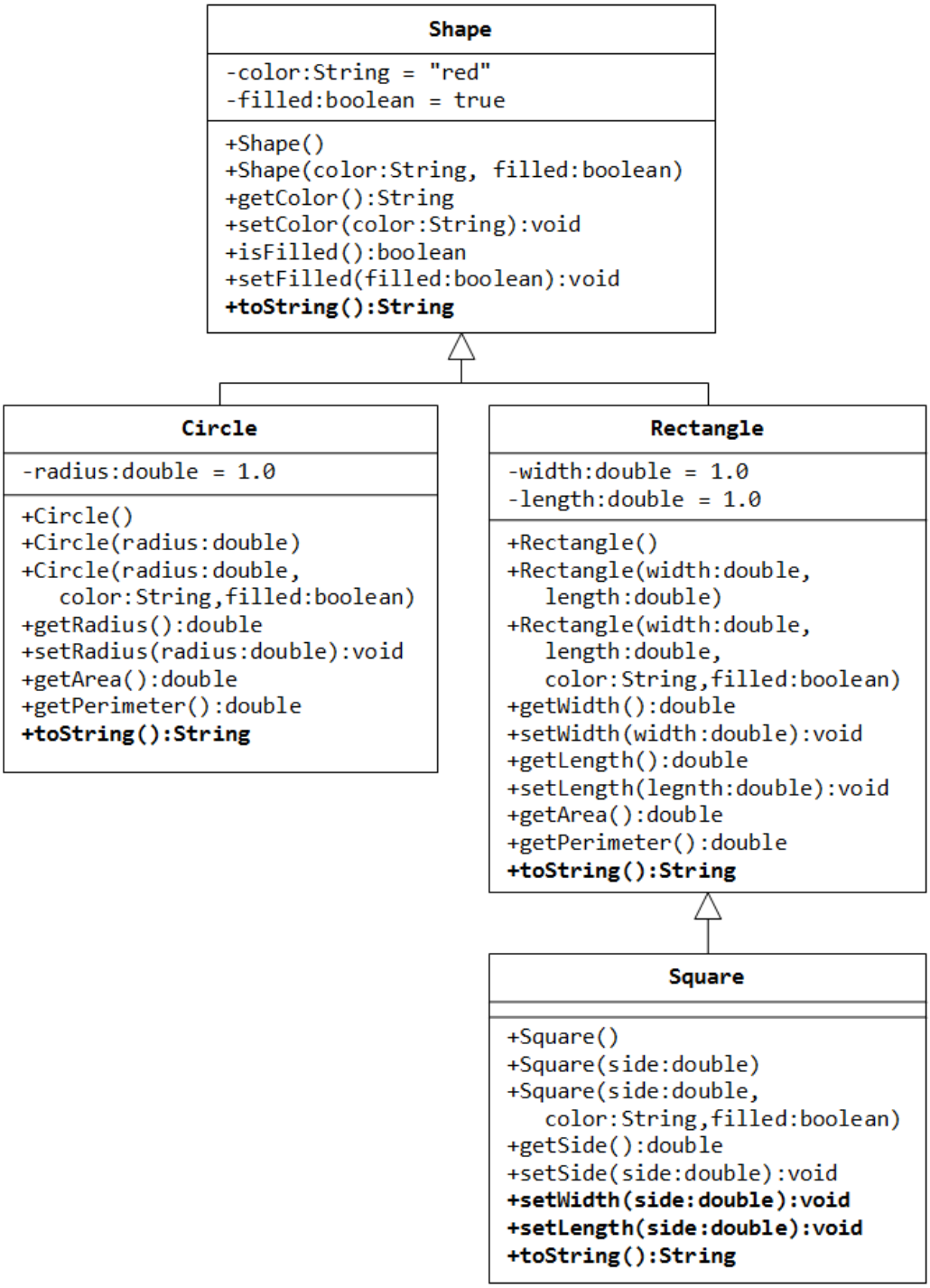

Fig. 1: Superclass Shape and its subclasses Circle, Rectangle and Square

(1) Write a superclass called Shape (as shown in Fig.1), which contains:

- Two instance variables color (String) and filled (boolean).
- Two constructors: a no-arg (no-argument) constructor that initializes the color to "green" 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".

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

(3) Write two subclasses of Shape called Circle and Rectangle, as shown in Fig.1.

①The 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.

②The 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.

(4) 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 Fig.1). 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.

2 (20 points) Abstract Superclass Shape and Its Concrete Subclasses

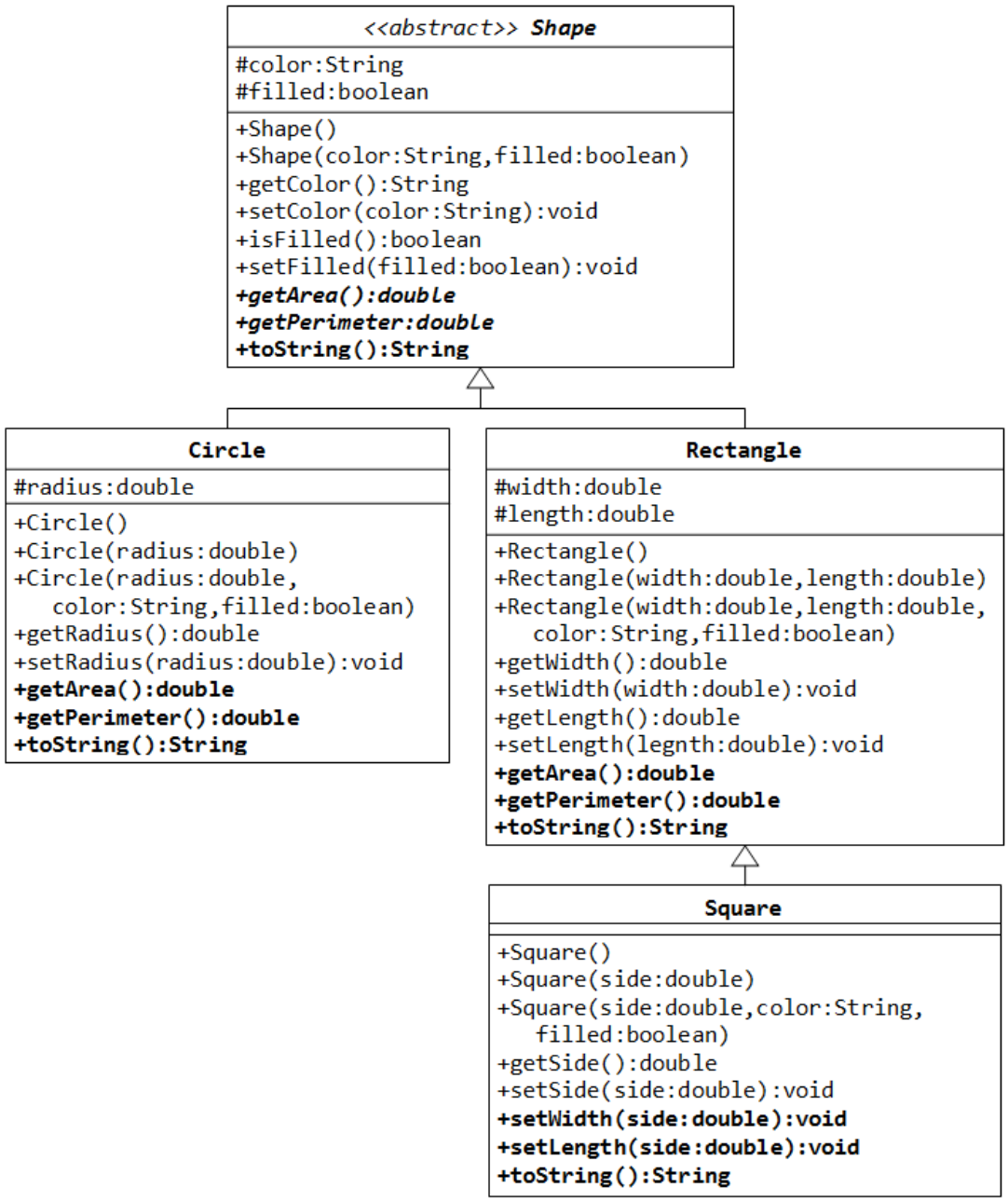

Fig. 2: Abstract Shape and its subclasses Circle, Rectangle and Square

(1) Rewrite the superclass Shape and its subclasses Circle, Rectangle and Square, as shown in Fig.2. 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 Fig.2.
- Getter and setter for all the instance variables, and toString().
- Two abstract methods getArea() and getPerimeter() (shown in italics in Fig.2).

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

(3) 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());
```

3. (25 Points) Polymorphism

(1) Examine the following codes.

```
abstract public class Animal {
    abstract public void greeting();
}
public class Cat extends Animal {
    @Override
    public void greeting() {
        System.out.println("Meow!");
    }
}
public class Dog extends Animal {
    @Override
    public void greeting() {
        System.out.println("Woof!");
    }

    public void greeting(Dog another) {
        System.out.println("Wooooooooof!");
    }
}
public class BigDog extends Dog {
    @Override
    public void greeting() {
        System.out.println("Woow!");
    }
```

```
    @Override
    public void greeting(Dog another) {
        System.out.println("Wooooooowwwww!");
    }
}
```

(2) Explain the outputs (or error) for the following test program.

```
public class TestAnimal {
    public static void main(String[] args) {
        // Using the subclasses
        Cat cat1 = new Cat();
        cat1.greeting();
        Dog dog1 = new Dog();
        dog1.greeting();
        BigDog bigDog1 = new BigDog();
        bigDog1.greeting();

        // Using Polymorphism
        Animal animal1 = new Cat();
        animal1.greeting();
        Animal animal2 = new Dog();
        animal2.greeting();
        Animal animal3 = new BigDog();
        animal3.greeting();
        Animal animal4 = new Animal();

        // Downcast
        Dog dog2 = (Dog)animal2;
        BigDog bigDog2 = (BigDog)animal3;
        Dog dog3 = (Dog)animal3;
        Cat cat2 = (Cat)animal2;
        dog2.greeting(dog3);
        dog3.greeting(dog2);
        dog2.greeting(bigDog2);
        bigDog2.greeting(dog2);
        bigDog2.greeting(bigDog1);
    }
}
```

4. (25 Points) Interface Movable and its implementations

Suppose that we have a set of objects with some common behaviors: they could move up, down, left or right. The exact behaviors (such as how to move and how far to move) depend on the objects themselves. One common way to model these common behaviors is to define an *interface* called Movable, with abstract methods moveUp(), moveDown(), moveLeft() and moveRight(). The classes that implement the Movable interface will provide actual implementation to these abstract methods.

(1) Let's write two concrete classes - MovablePoint and MovableCircle - that implement the Movable interface. The code for the interface Movable is straight forward.

```
public interface Movable {    // saved as "Movable.java"
    public void moveUp();
    ......
}
```

For the MovablePoint class, declare the instance variable x, y, xSpeed and ySpeed with package access as shown with '~' in the class diagram in Fig.3 (i.e., classes in the same package can access these variables directly). For the MovableCircle class, use a MovablePoint to represent its center (which contains four variable x, y, xSpeed and ySpeed). In other words, the MovableCircle composes a MovablePoint, and its radius.

```
public class MovablePoint implements Movable { // saved as "MovablePoint.java"
    // instance variables
    int x, y, xSpeed, ySpeed;         // package access

    // Constructor
    public MovablePoint(int x, int y, int xSpeed, int ySpeed) {
        this.x = x;
        ......
    }
    ......

    // Implement abstract methods declared in the interface Movable
    @Override
    public void moveUp() {
        y -= ySpeed;      // y-axis pointing down for 2D graphics
    }
    ......
}

public class MovableCircle implements Movable { // saved as "MovableCircle.java"
    // instance variables
    private MovablePoint center;      // can use center.x, center.y directly
                                      //    because they are package accessible
```

```
    private int radius;

    // Constructor
    public MovableCircle(int x, int y, int xSpeed, int ySpeed, int radius) {
        // Call the MovablePoint's constructor to allocate the center instance.
        center = new MovablePoint(x, y, xSpeed, ySpeed);
        ......
    }
    ......

    // Implement abstract methods declared in the interface Movable
    @Override
    public void moveUp() {
        center.y -= center.ySpeed;
    }
    ......
}
```

(2) Write a test program and try out these statements:

```
Movable m1 = new MovablePoint(5, 6, 10, 15);          // upcast
System.out.println(m1);
m1.moveLeft();
System.out.println(m1);

Movable m2 = new MovableCircle(1, 2, 3, 4, 20);    // upcast
System.out.println(m2);
m2.moveRight();
System.out.println(m2);
```

5. (Bonus Question: 20 Points) The Discount System

You are asked to write a discount system for a beauty saloon, which provides services and sells beauty products. It offers 3 types of memberships: Premium, Gold and Silver. Premium, gold and silver members receive a discount of 20%, 15%, and 10%, respectively, for all services provided. Customers without membership receive no discount. All members receives a flat 10% discount on products purchased (this might change in future). Your system shall consist of three classes: Customer, Discount and Visit, as shown in the class diagram in Fig.4. It shall compute the total bill if a customer purchases $x of products and $y of services, for a visit. Also write a test program to exercise all the classes.

The class DiscountRate contains only static variables and methods (underlined in Fig.4).

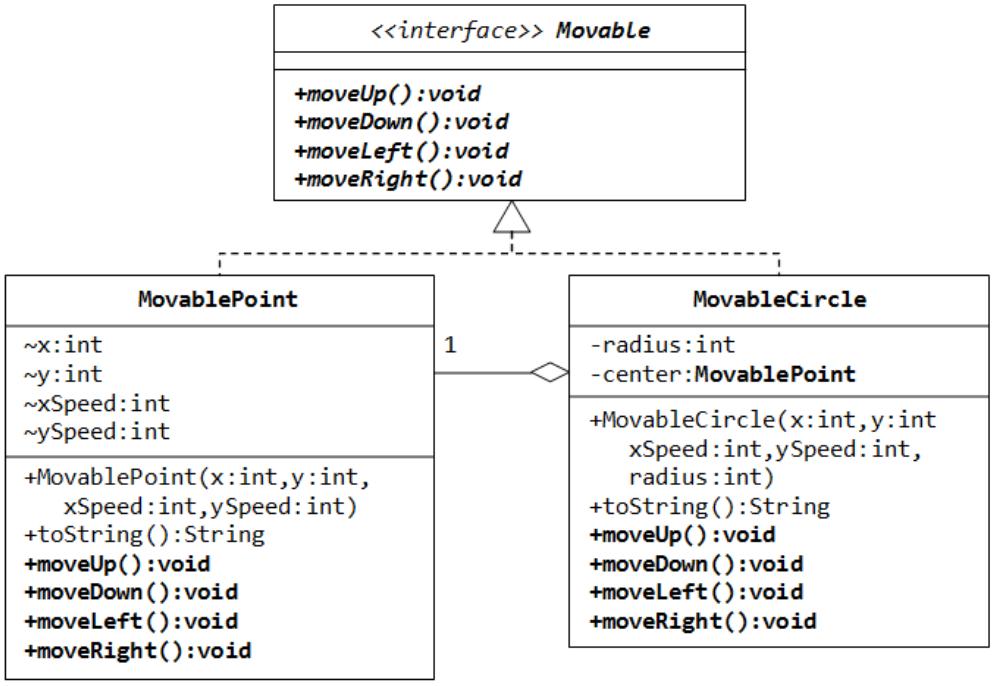

Fig. 3: Interface Movable and its implementations

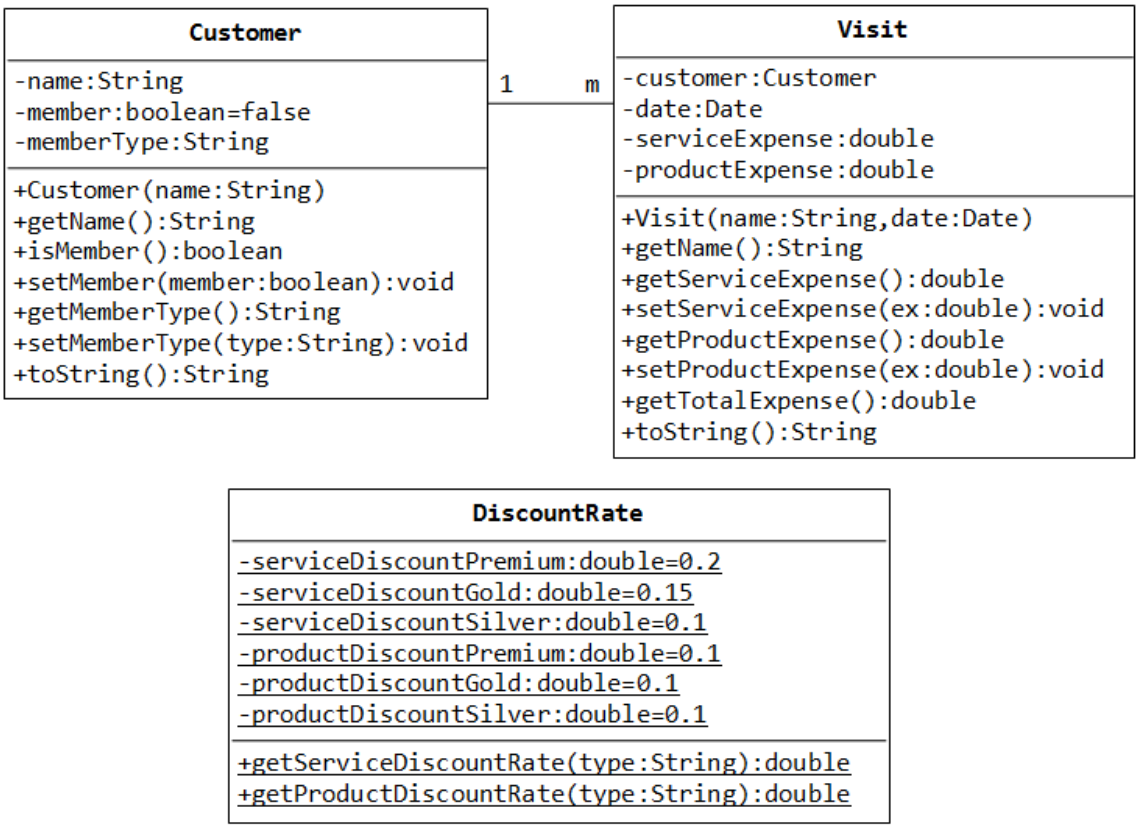

Fig. 4: Class diagram for the Discount System