Labsheet 7: Inheritance and Polymorphism

Question 1

(a) Create a super class called Car. The Car class has the following fields and methods.
- int speed;
- double regularPrice;
- String color;
- Car();
- double getSalePrice();

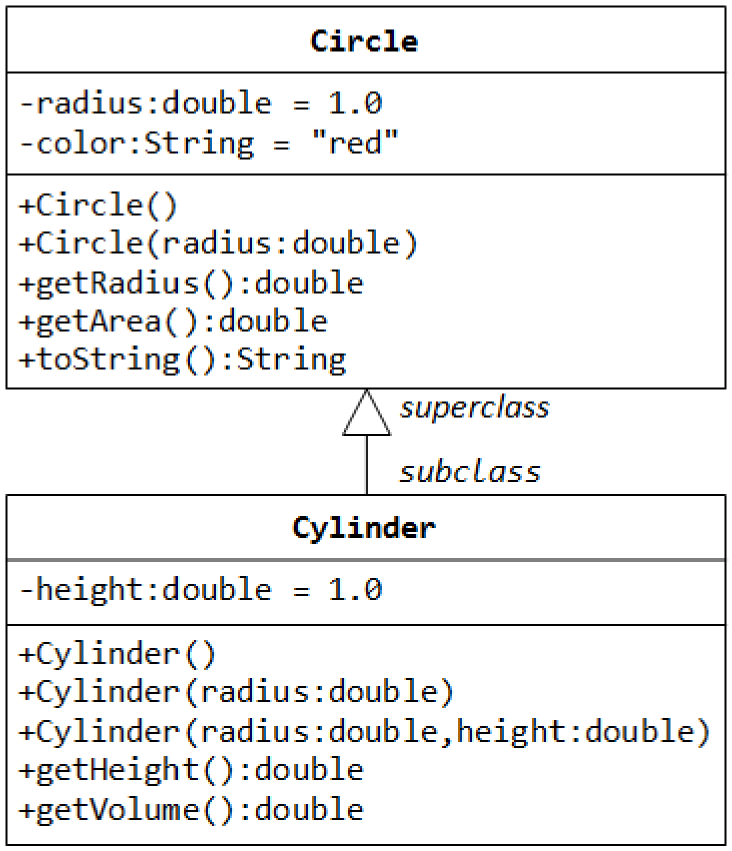
(b) Create a sub class of Car class and name it as Truck. The Truck class has the following fields and methods.
- int weight;
- Truck();
- double getSalePrice();//Ifweight>2000,10%discount.Otherwise,20%discount.

(c) Create a subclass of Car class and name it as Ford. The Ford class has the following fields and methods
- int year;
- int manufacturerDiscount;
- Ford();
- doubleget SalePrice(); //From the sale price computed from Car class, subtract the manufacturer Discount.

(d) Create a subclass of Car class and name it as Sedan. The Sedan class has the following fields and methods.
- int length;
- Sedan();
- double getSalePrice(); //Iflength>20feet,5%discount,Otherwise,10%discount.
-

(e) Create MyOwnAutoShop class which contains the main() method. Perform the following within the main() method.
- Create an instance of Sedan class and initialize all the fields with appropriate values. Use super(...) method in the constructor for initializing the fields of the superclass.
- Create two instances of the Ford class and initialize all the fields with appropriate values. Use super(...) method in the constructor for initializing the fields of the super class.
- Create an instance of Car class and initialize all the fields with appropriate values.
- Display the sale prices of all instance.

Question 2



In this exercise, a subclass called `Cylinder` is derived from the superclass `Circle` as shown in the class diagram (where an 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.class`" 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 (=2π×radius×height + 2×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 overridding the `getArea()` in subclass `Cylinder`, you can choose to invoke the `getArea()` of the superclass `Circle` by calling `super.getArea()`.

TRY:

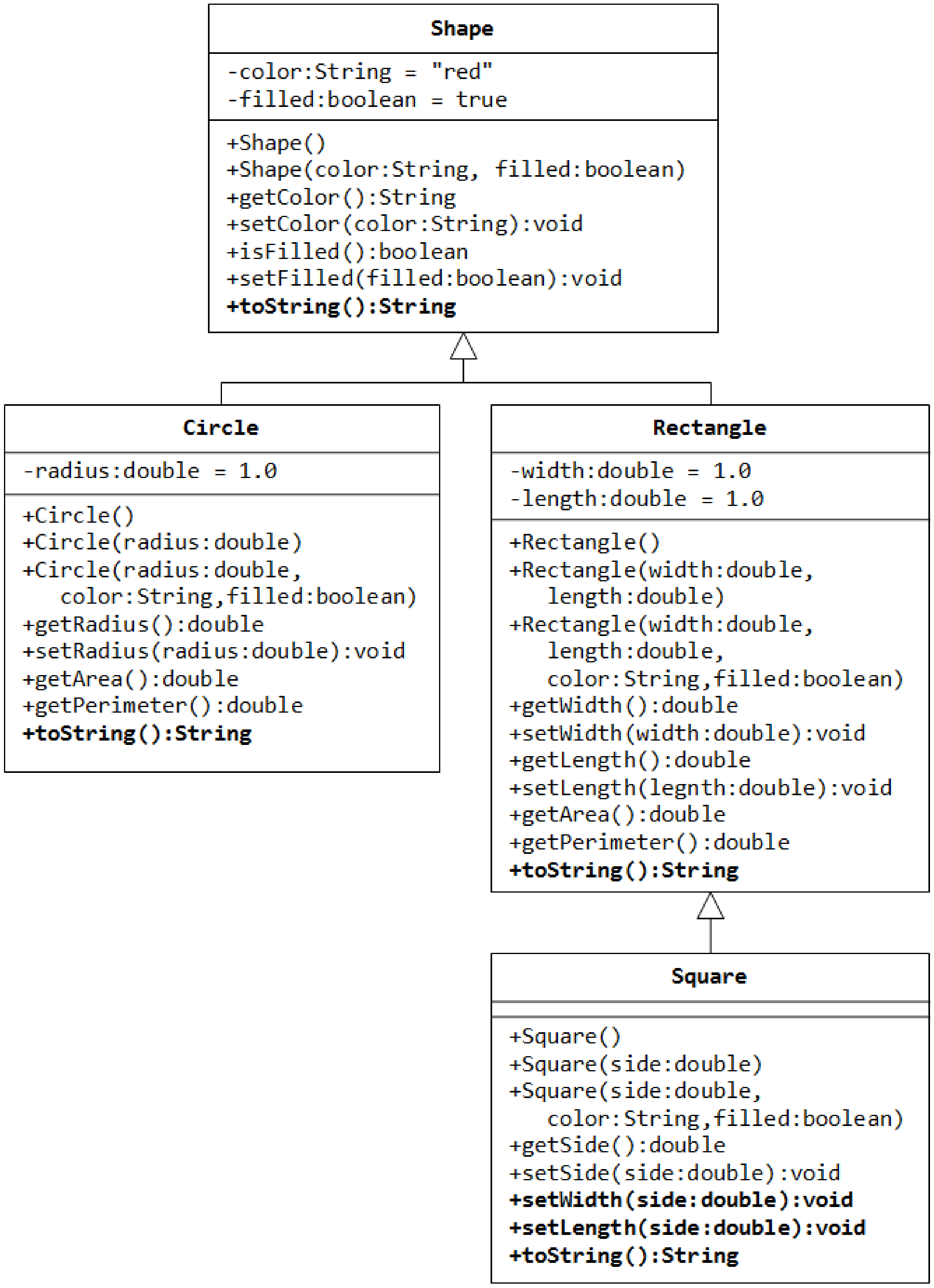
Provide a `toString()` method to the `Cylinder` class, which overrides the `toString()` inherited from the superclass `Circle`, e.g.,

```
@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.

Question 3


Shape
-color:String = "red"
-filled:boolean = true
+Shape()
+Shape(color:String, filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+toString():String
Circle
-radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double, color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
Rectangle
-width:double = 1.0
-length:double = 1.0
+Rectangle()
+Rectangle(width:double, length:double)
+Rectangle(width:double, length:double, color:String,filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
Square
+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

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 "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`".

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.

Question 4

A small company dealing with transportation has just purchased a computer for its new automated reservations system. You have been asked to program the new system. You are to write a program called *ReservationSystem* to assign seats on a vehicle. Your class also requires the following:

- a constructor method, which initialise the variables
- a method to assign the capacity of seating.
- a method for assigning seats. Use a 1-d array to represent the seating chart of the plane. Initialize all the elements of the array to 0 to indicate that all the seats are empty. As each seat is assigned, set the corresponding elements of the array to 1 to indicate that the seat is no longer available. Your program should, of course never assign a seat that has already been assigned.
- appropriate mutator and accessor methods

The company also needs a program dealing especially with its only plane with each flight having a capacity of 10 seats. Name this class *AirlineReservationSystem.* This class is a type of *ReservationSystem* but the way it reserves seats are different.

Your program should display the following menu of alternatives for reserving a seat on the flight:

Please type 1 for "smoking"

Please type 2 for "non-smoking"

If the person types 1,then your program should assign a seat in the smoking section(seats 1-5) If the person types 2,then your program should assign a seat in the non-smoking section(seats 6-10). Your program should then print a boarding pass indicating the person's seat number and whether it is in the smoking or non-smoking section of the plane.

When the smoking section is full, your program should ask the person if it is acceptable to be placed in the non-smoking section (and vice versa). If yes, then make the appropriate seat assignment. If no, then print the message "**Next flight leaves in 3 hours.**"

Create a main() method to test the next scheduled flight.