CS102A Introduction to Computer Programming

Fall 2020

Lab 11

Objectives

1. Learn about class inheritance.
2. Learn when and how to use the `protected` keyword.
3. Learn about object polymorphism.

1 Prework

Copy the following code to `Circle.java`:

```
public class Circle {
  private double radius;
  private double x;
  private double y;
  static final int DEFAULT_RADIUS = 5;
  private static int screenSize = 10;
  private ShapeColor color = ShapeColor.GRAY;

  public Circle(double radius, double x, double y) {
    this.radius = radius;
    this.x = x;
    this.y = y;
  }
```

```
public Circle(double radius) {
  this.radius = radius;
  this.x = 0;
  this.y = 0;
}

public Circle(double x, double y) {
  this.radius = DEFAULT_RADIUS;
  this.x = x;
  this.y = y;
}

public static int getScreenSize() {
  return screenSize;
}

public static void setScreenSize(int screenSize) {
  Circle.screenSize = screenSize;
}

public void checkColor() {
  if (isInBoundary()) {
    color = ShapeColor.GREEN;
  } else {
    color = ShapeColor.RED;
  }
}

public boolean isInBoundary() {
  if (-1 * Circle.screenSize > this.x - this.radius || Circle.
     screenSize < this.x + this.radius) {
    return false;
```

```
    }
    if (-1 * Circle.screenSize > this.y - this.radius || Circle.
        screenSize < this.y + this.radius) {
      return false;
    }
    return true;
  }

  @Override
  public String toString() {
    return "Circle{" + "radius=" + " x=" + x + ", y=" + y + ",
        color=" + color + "}\n";
  }

  public double getRadius() {
    return radius;
  }

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

  public double getX() {
    return x;
  }

  public void setX(double x) {
    this.x = x;
  }

  public double getY() {
    return y;
  }
```

```
  public void setY(double y) {
    this.y = y;
  }

  public void draw() {
    StdDraw.setPenColor(color.getColor());
    StdDraw.filledCircle(x, y, radius);
  }
}
```

Copy the following code to `Rectangle.java`:

```
public class Rectangle {
  private double x;
  private double y;
  private double width;
  private double height;
  private static int screenSize = 10;
  private ShapeColor color = ShapeColor.GRAY;

  public Rectangle(double x, double y) {
    this.x = x;
    this.y = y;

  }

  public Rectangle(double x, double y, double width, double
     height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
```

```
  }

  public static int getScreenSize() {
    return screenSize;
  }

  public static void setScreenSize(int screenSize) {
    Rectangle.screenSize = screenSize;
  }

  public void checkColor() {
    if (isInBoundary()) {
      color = ShapeColor.GREEN;
    } else {
      color = ShapeColor.RED;
    }
  }

  public boolean isInBoundary() {
    if (-1 * Rectangle.screenSize > this.x - this.width / 2 ||
        Rectangle.screenSize < this.x + this.width / 2) {
      return false;
    }
    if (-1 * Rectangle.screenSize > this.y - this.height / 2 ||
        Rectangle.screenSize < this.y + this.height / 2) {
      return false;
    }
    return true;
  }

  public double getX() {
    return x;
  }
```

```
public void setX(double x) {
  this.x = x;
}

public double getY() {
  return y;
}

public void setY(double y) {
  this.y = y;
}

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;
}

public String toString() {
  return "Rectangle{" + "width=" + width + ", height=" + height
      + " x=" + x + ", y=" + y + ", color=" + color + "}\n";
}
```

```
  public void draw() {
    StdDraw.setPenColor(color.getColor());
    StdDraw.filledRectangle(x, y, this.width / 2, this.height /
       2);
  }
}
```

Copy the following code to `ShapeColor.java`:

```
import java.awt.Color;

public enum ShapeColor {
  GREEN("The shape is in the Screen", Color.GREEN), RED("The
     shape is not in the Screen", Color.RED), GRAY("Haven't
     tested", Color.GRAY);

  private String desc;  // The description of instance
  private Color color;  // The color of instance

  ShapeColor (String desc, Color color) {
    this.desc = desc;
    this.color = color;
  }

  public String getDesc() {
    return this.desc;
  }

  public Color getColor() {
    return this.color;
  }
}
```

Download `StdDraw.java` from `https://introcs.cs.princeton.edu/java/stdlib/StdDraw.java`. Place it within the same directory as the above three `.java` files.

At this point, you should have four `.java` files, in preparation for the following exercises.

2 Exercises

2.1 Class Inheritance

From the source code, you may observe that the two classes `Circle` and `Rectangle` have a lot of common fields, e.g., `screenSize`, `x`, `y` and `ShapeColor`. They also have a lot of similar methods. It is a good time to practice class inheritance by refactoring the code.

The idea of class inheritance is simple but powerful: when you want to create a new class and there is an existing class which includes some of the code that you want, you can extend your new class from the existing class. In doing so, you can reuse the fields and methods of the existing class without having to write (and debug!) them yourself.

A *subclass* inherits all the members (fields, methods, and nested classes) of its *superclass*. Constructors are not class members, so they are not inherited by subclasses, but the subclass must invoke one of the constructors in its superclass.

> **Further Reading**
> See `https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html` for more details.

As mentioned above, the attributes `x`, `y`, `color` and `screenSize` are in both `Circle` and `Rectangle`. It is more elegant and efficient to place these common attributes into a superclass named `Shape`, whose attributes and methods can all be used by its subclasses. Let us take the following steps to create the `Shape` class.

1. Add the following attributes:

```
private double x;
private double y;
private ShapeColor color = ShapeColor.GRAY;
private static int screenSize = 10;
```

2. Add a constructor with parameters `x` and `y`:

```
public Shape(double x, double y) {
  this.x = x;
  this.y = y;
}
```

3. Add getter/setter methods for the `private` variables.

4. Add a `toString()` method (override the corresponding method of the `Object` class) to print the properties of the `Shape` object:

```
@Override
public String toString() {
  return " x=" + x + ", y=" + y + ", color=" + color;
}
```

Now, modify the `Circle` class. Let it inherit the `Shape` class by using the `extends` keyword. Remove all methods and attributes that can be inherited from `Shape`. This way, the `Circle` class only needs to define two specific attributes: `radius` and `DEFAULT_RADIUS`:

```
private double radius;
private static final int DEFAULT_RADIUS = 5;
```

Modify the constructor of the `Circle` class as follows and use `super()`:

```
public Circle(double radius, double x, double y) {
  this.radius = radius;
  this.x = x;
  this.y = y;
}

public Circle(double radius) {
  this.radius = radius;
  this.x = 0;
  this.y = 0;
}
```

```
12
13 public Circle(double x, double y) {
14   this.radius = DEFAULT_RADIUS;
15   this.x = x;
16   this.y = y;
17 }
```

Now, attributes `x` and `y` are inherited from `Shape`. It is recommended to use the constructor of the superclass to initialize any subclass. For example:

```
1 public Circle(double radius) {
2   super(0,0);
3   this.radius = radius;
4 }
```

Note that `this` points to the current object, while `super` indicates the only superclass of the current object. Rewrite the remaining constructors accordingly.

2.1.1 Accessing the `instance` and `static` fields of a superclass from a subclass.

You will find that some errors occur in other methods, for example:

```
1 public boolean isInBoundary() {
2   if (-1 * Circle.screenSize > this.x - this.radius || Circle.
       screenSize < this.x + this.radius) {
3     return false;
4   }
5   if (-1 * Circle.screenSize > this.y - this.radius || Circle.
       screenSize < this.y + this.radius) {
6     return false;
7   }
8   return true;
9 }
```

Let us change `Circle.screenSize` to `Shape.getScreenSize()`, since `screenSize` is a `private static` field. Next, change `this.x` to `super.getX()`, since `x` is a `private` field of the superclass. Change other methods accordingly.

2.1.2 When and how to use the `protected` keyword?

For security or other reasons, it may be undesirable for the `Circle` class to have direct, unrestricted access to the `private` attributes of its superclass. As such, we can consider making the frequently-used attributes of `Shape` accessible to its subclasses. The `protected` keyword can be used for this.

Change `x`, `y`, and `color` from `private` to `protected`. Then, change `isInBoundary()` back to the original one except `Shape.getScreenSize()`, which already works as intended.

```
public boolean isInBoundary() {
  if (-1 * Shape.getScreenSize() > this.x - this.radius || Shape.
     getScreenSize() < this.x + this.radius) {
    return false;
  }
  if (-1 * Shape.getScreenSize() > this.y - this.radius || Shape.
     getScreenSize() < this.y + this.radius) {
    return false;
  }
  return true;
}
```

Change the remaining methods accordingly. Next, modify the given `Rectangle` class to make it inherit from `Shape`:

1. Make `Rectangle` extend `Shape`.
2. Modify the constructors of `Rectangle`.
3. Modify other methods of `Rectangle`.
4. Modify the `toString()` method.

Run the following `ShapeTest` to test your modifications.

```
public class ShapeTest {
  public static void main(String[] args) {
    Circle c1=new Circle(0.1,1,1);
    Circle c2=new Circle(0.1,0.5,2);
```

```
    Circle.setScreenSize(2);
    System.out.print(c1);
    c1.checkColor();
    c2.checkColor();
    System.out.print(c1);
    System.out.print(c2);

    Rectangle r1=new Rectangle(0,0,0.5,0.5);
    Rectangle r2=new Rectangle(2,1,0.5,0.5);
    Rectangle.setScreenSize(2);
    System.out.print(r1);
    r1.checkColor();
    r2.checkColor();
    System.out.print(r1);
    System.out.print(r2);

    StdDraw.setXscale(-Circle.getScreenSize(), Circle.
       getScreenSize());
    StdDraw.setYscale(-Circle.getScreenSize(), Circle.
       getScreenSize());
    c1.draw();
    c2.draw();
    r1.draw();
    r2.draw();
    Circle c3=new Circle(0.1,0.5,-2);
    Rectangle r3=new Rectangle(-2,1,0.5,0.5);
    c3.draw();
    r3.draw();
  }
}
```

2.2 Polymorphism

Create a class named `PolymorphismTest`:

```
public class PolymorphismTest {
  public static void main(String[] args) {
    ArrayList<Shape> shapeList = new ArrayList<Shape>();

    Shape.setScreenSize(9);
    StdDraw.setXscale(-Shape.getScreenSize(), Shape.getScreenSize
       ());
    StdDraw.setYscale(-Shape.getScreenSize(), Shape.getScreenSize
       ());

    for (int i = 0; i < 3; i++) {
      shapeList.add(new Circle(1, 4 * i + 1, 1));
      shapeList.add(new Rectangle(4 * i + 1, -1, 1,1));
    }

    for (int i = 0; i < shapeList.size(); i++) {
      shapeList.get(i).checkColor();
      System.out.print(shapeList.get(i));
      shapeList.get(i).draw();
    }
  }
}
```

Two errors would arise in `checkColor()` and `draw()`. Although these two methods have been defined in both `Circle` and `Rectangle` class, we cannot invoke them directly if they have not been defined in their superclass `Shape`. Thus, let us define these two methods in `Shape`:

```
1 public void checkColor() {
2 }
3
4 public void draw() {
5 }
```

Run the above code and observe the result:

```
Circle{radius=1.0 x=1.0, y=1.0, color=GREEN}
Rectangle{width=1.0, height=1.0 x=1.0, y=-1.0, color=GREEN}
Circle{radius=1.0 x=5.0, y=1.0, color=GREEN}
Rectangle{width=1.0, height=1.0 x=5.0, y=-1.0, color=GREEN}
Circle{radius=1.0 x=9.0, y=1.0, color=RED}
Rectangle{width=1.0, height=1.0 x=9.0, y=-1.0, color=RED}
```