CS102A Introduction to Computer Programming
Fall 2020
Lab 12

Objectives

1. Learn about the `abstract` class.
2. Learn how to define and implement an `interface`.
3. Learn how to use the `java.long.Comparable<T>` interface.

1 Prework

1.1 Class Inheritance

1.1.1 Step 1

Start from the code you wrote in the previous lab. We can see that there are two `public` methods which have no code.

```
public void checkColor() {
}

public void draw() {
}
```

It is important to note that we do not need to create an instance of the `Shape` class. In this case, we should change the `Shape` class to an `abstract` class, as follows:

1. Add `abstract` before `class`:

```
public abstract class Shape
```

2. Define the `draw()` method as `abstract`:

```
abstract public void draw();
```

1.1.2 Step 2

In `ShapeTest`, let us write the following code in `main()`:

```
Shape shape=new Shape();
```

Executing the above will produce an error, stating that the type `Shape` cannot be instantiated.

1.1.3 Step 3

Suppose we have several circles, each with a different radius. We want to sort them by radius in descending order, i.e., from big to small. How to do so?

This is where the `Comparable` interface comes in handy. This `interface` imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the `compareTo()` method in it is referred to as its *natural comparison* method.

Lists (and arrays) of objects that implement this interface can be sorted automatically using `Collections.sort()` (or `Arrays.sort()`).

1. Let `Circle` implement the `Comparable` interface.

```
public class Circle extends Shape implements Comparable<
   Circle>
```

   It is important to note that, if a class implements an interface, it must override all `abstract` methods in it.

2. Override the method `compareTo()` defined in `Comparable`.

```
@Override
public int compareTo(Circle o) {
```

```
  if (this.radius < o.radius) {
    return 1;
  } else if (this.radius > o.radius) {
    return -1;
  }
  return 0;
}
```

Normally, `compareTo()` would determine a sort order by comparing the current object (referenced by `this`) with the object passed to the method via the `o` argument. `compareTo()` can return a negative, zero, or positive integer; this means that the current object would be ranked lower, equal, or higher than the parameter-provided object `o`, respectively. However, in this case, we want to sort the `Circle` objects in descending order according to their `radius`, so that when the `radius` of the current object is less than the `radius` of the parameter-referenced object, the returned value would be 1 (a positive integer).

3. Rewrite `ShapeTest` using the following code:

```
public static void main(String[] args) {
  List<Circle> circleList = new ArrayList<>();
  Circle.setScreenSize(14);
  StdDraw.setScale(-Shape.getScreenSize(), Shape.
     getScreenSize());
  for (int i = 0; i < Shape.getScreenSize(); i += 2) {
    circleList.add(new Circle(i, 0, -Shape.getScreenSize()));
  }
  Collections.sort(circleList);
  for (int i = 0; i < circleList.size(); i++) {
    circleList.get(i).setColor(ShapeColor.values()[i%3]);
    circleList.get(i).draw();
  }
}
```

Inspect the result.

1.1.4 Step 4

In the previous step, we can see that the way colors are set is not very elegant, while `ShapeColor` is mainly used to check if the shape is within the boundaries. We can define an interface named `ColorDraw`, in which we will declare an `abstract` method `customizedColor()`.

1. Define an `enum` class `ColorScheme`:

```
public enum ColorScheme {
  SKY(new Color[]{new Color(0, 102, 204),
      new Color(0, 128, 255),
      new Color(51, 153, 255),
      new Color(102, 178, 255),
      new Color(153, 204, 255),
      new Color(204, 229, 255)}),
  RAINBOW(new Color[]{
      Color.RED,
      Color.ORANGE,
      Color.YELLOW,
      Color.GREEN,
      Color.CYAN,
      new Color(0, 128, 255),
      new Color(204, 153, 255)}),
  GRAY(new Color[]{
      Color.DARK_GRAY,
      Color.GRAY,
      Color.LIGHT_GRAY});

  Color[] colorList;

  private ColorScheme(Color[] color) {
    colorList = color;
  }

  public Color[] getColorScheme() {
```

```
    return colorList;
  }
}
```

If you are using an IDE, it may remind you to choose the package from which the `Color` class is imported. Here, let us import from `java.awt`.

2. Define an `interface`:

```
public interface ColorDraw {
  public void customizedColor(ColorScheme colorScheme, int
     index);
}
```

3. Implement `ColorDraw` in `Circle`:

```
public class Circle extends Shape implements Comparable<
   Circle>, ColorDraw

@Override
public void customizedColor(ColorScheme colorScheme, int
   index) {
  Color[] colorList = colorScheme.getColorScheme();
  if (index < 0){
    index = 0;
  }
  if (index >= colorList.length){
    index = index % colorList.length;
  }
  StdDraw.setPenColor(colorList[index]);
  StdDraw.filledCircle(Shape.getX(), Shape.getY(), radius);
}
```

4. In `ShapeTest`, let us change the `main` method according to the following code and inspect the result:

```
List<Circle> circleList = new ArrayList<Circle>();
Shape.setScreenSize(14);
StdDraw.setScale(-Shape.getScreenSize(), Shape.getScreenSize
   ());

for (int i = 1; i < Shape.getScreenSize(); i += 2) {
  circleList.add(new Circle(i, 0, -Shape.getScreenSize()));
}

Collections.sort(circleList);

for (int i = 0; i < circleList.size(); i++) {
  circleList.get(i).customizedColor(ColorScheme.RAINBOW, i);
}
```

2 Exercises

1. Modify the `ShapeTest` class to draw some circles as shown in the following image:

2. Modify the `Rectangle` class from Lab 10.

   (a) Make `Rectangle` implement `Comparable`, then override the `compareTo()` method to sort the rectangles from largest to smallest according to their `area`. If two rectangles have the same `area`, sort the rectangles from smallest to largest according to `x`.

   (b) Make `Rectangle` implement `ColorDraw`, then override the `customizedColor()` method to draw the rectangle according to the specified `ColorScheme` and index.

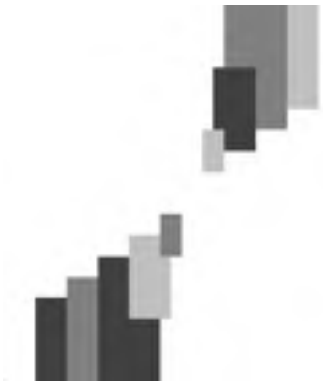
3. Create a `RectangleTest` class for testing.

```java
public class RectangleTest {
  public static void main(String[] args) {
    Shape.setScreenSize(9);
    StdDraw.setScale(-Shape.getScreenSize(), Shape.
       getScreenSize());

    List<Rectangle> rectanglList = new ArrayList<Rectangle>()
       ;
    for (int i = -5; i < 5; i ++) {
      rectanglList.add(new Rectangle(i,2*i,Math.abs(i), 2*
         Math.abs(i)));
    }
    Collections.sort(rectanglList);

    for (int i = 0; i < rectanglList.size(); i++) {
      rectanglList.get(i).customizedColor(ColorScheme.GRAY, i
         );
      System.out.println(rectanglList.get(i));
    }
  }
}
```

Here is a sample run:



4. You can design your own pattern that contains circles and rectangles, or other shapes of your choosing.