Tutorial of Inheritance

> Based on the tutorial of "2020S-Java-A" designed by teaching group in SUSTech
>
> Modified (only change to markdown file) by ZHU Yueming in 2021. May. 5th

Objective

- Learn how to define and implement an interface.
- Learn how to use the interface `java.lang.Comparable<T>`.

Part 1: Comparable interface.

1. Before Exercise

1.1 Comparable Interface

Comparable is very useful. 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 by `Collections.sort` (and `Arrays.sort`).

- Let **Circle** implements **Comparable**

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

  After implements the interface Comparable, it arise a mistake that if a class implements an interface, it must override all abstract methods in it.

- Override the method **compareTo** defined in **Comparable.**

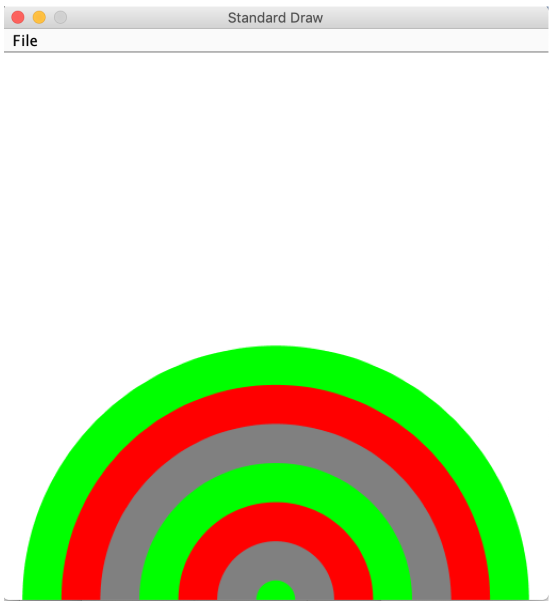
  ```
  @Override
  public int compareTo(Circle o) {
      if(this.radius < o.radius){
          return 1;
      }else if(this.radius > o.radius){
          return -1;
      }
      return 0;
  }
  ```

Normally, this method compares current object with the parameter object to determine a sort order. The return value of the method can be a negative integer, zero, or a positive integer, which means current object would in former place, all equal, or in latter place than parameter respectively. However, in this case, we want to sort the Circles in descending order of its radius, so that when the radius of current object is less than the parameter object, the return value would be 1(a positive integer)

- Rewrite the **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();
    }
}
```

Run result:



- In **Step3**, we can see that the color scheme is not beautiful and the **ShapeColor** is mainly used to check if the shape is in the boundary. We can define an interface named **ColorDraw**, in which declare an abstract method **customizedColor**.

1.2 User Defined Interface

- Define an `enum` class **ColorScheme**:

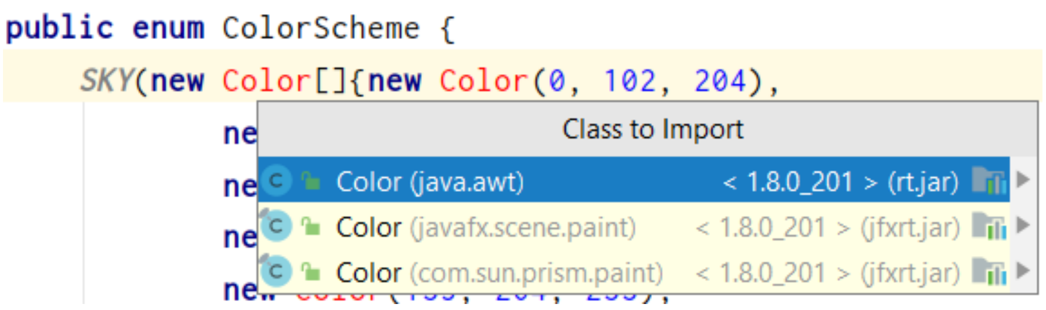
```java
public enum ColorScheme {
    SKY(new Color[]{new Color(0, 102, 204),
            new Color(0, 128, 255),
            new Color(51, 153, 255),
            new Color(102, 178, 255),
            new Color(153, 204, 255),
            new Color(204, 229, 255)}),
    RAINBOW(new Color[]{
            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 IDE, it may remind you to choose the package from which the Color class is imported. Here, please import from java.awt.



```java
public enum ColorScheme {
    SKY(new Color[]{new Color(0, 102, 204),
            ne
            ne
            ne
            new
```

Class to Import
Color (java.awt) < 1.8.0_201 > (rt.jar)
Color (javafx.scene.paint) < 1.8.0_201 > (jfxrt.jar)
Color (com.sun.prism.paint) < 1.8.0_201 > (jfxrt.jar)

- Define an interface:

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

- Implement the **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);
}
```

- In **ShapeTest**, we change main method to the following code:

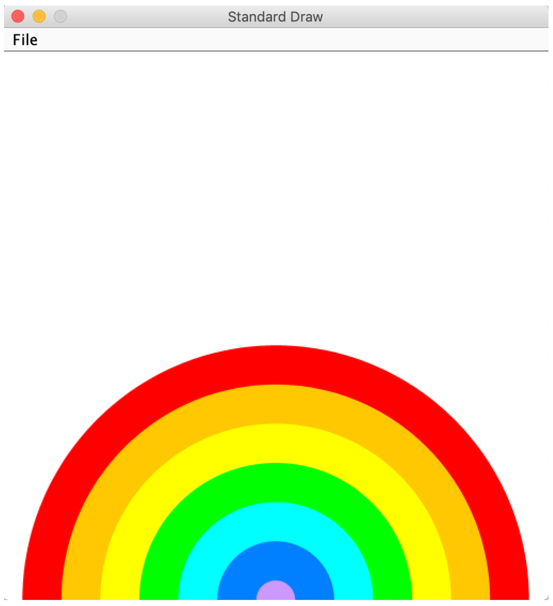
```
List<Circle> circleList = new ArrayList<Circle>();
      Shape.setScreenSize(14);
      StdDraw.setScale(-Shape.getScreenSize(), Shape.getScreenSize());

      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);
      }
```

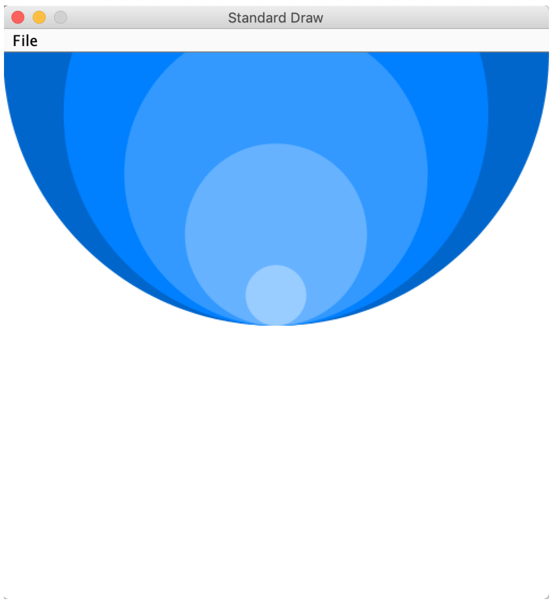
Run result:



2. Exercises

2.1 Exercise1

Modify the class **ShapeTest,** draw some circles like the following image:



Hint: ColorScheme.Sky

x	y	radius
0	1	1
0	3	3
0	5	5
0	7	7
0	9	9

screen size = 9

2.2 Exercise 2

Modify the class **Rectangle** given to you.

a. Make **Rectangle** implements **Comparable**, override the method **compareTo** to order the rectangles from the largest to smallest according their area. If two rectangles have the same area, order the rectangles from smallest to largest according **x**.

b. Make **Rectangle** implements **ColorDraw**, override the method **customizedColor** to draw the rectangle according to the specific **ColorScheme** and the index.

2.3 Exercise 3

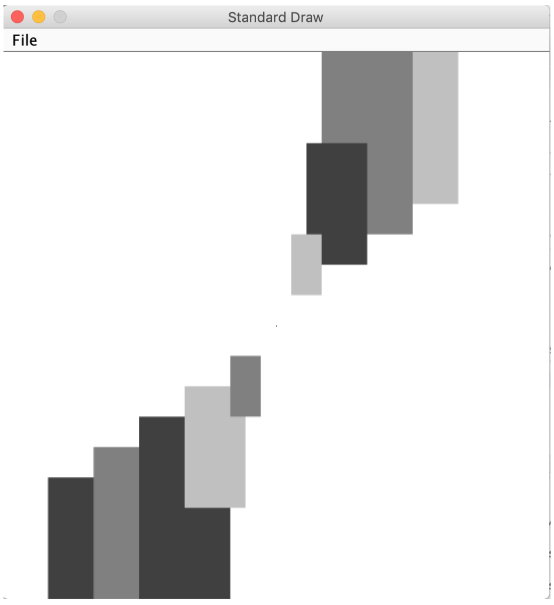
Create a class **RectangleTest** for test.

```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:



2.4 Exercise 4

You can design yourself pattern that contains circles and rectangles, or other yourself defined shapes.