Tutorial of Inheritance

> Based on the tutorial of "2020S-Java-A" designed by teaching group in SUSTech (Designed by ZHAO Yao)
>
> Modified (only change to markdown file) by ZHU Yueming in 2021. April. 25th
>
> Minor changes by Yida Tao, Nov. 17 2022

Objectives

- Learn inheritance.
- Learn the `protected` keyword.

Before Exercise

Please download all .java code from Sakai, and place all .java files at the same package.

1. Source code analysis

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

The idea of 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 create your new class by extending the existing class. By doing so, you can reuse the fields and methods of the existing class without having to (re)write (and debug!) them yourself.

A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the subclass must invoke one of the constructors in its superclass. (https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html)

We found that, the attributes `x`, `y`, `color` and `screenSize` are all in `Circle` and `Rectangle`. Hence, these common attributes can be extracted into a super class named `Shape`, from which the subclass can use all of its attributes and methods.

2. Create a super class

Create a class `Shape`, which contains following members:

- Adding attributes

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

- Adding constructors with two parameter `x` and `y`

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

- Adding getter/setter method of all attributes.
- Adding `toString()` method (override the method of the Object class) to output the property of the Shape object;

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

3. Modify subclass

- Now, modify class `Circle`. Let it inherit class `Shape` by using the keyword `extends`.

  ```
  public class Circle extends Shape
  ```

- Remove those methods and attributes which can be inherited from class `Shape`. Now, class `Circle` 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 class `Circle` as follows and use `super()` (`this` serves as the current object, while `super` serves as the only supper class for current object).

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

      public Circle(double radius) {
          super(0,0);
          this.radius = radius;
      }

      public Circle(double x, double y) {
  ```

```
        super(x,y);
        this.radius = DEFAULT_RADIUS;
    }
```

4. Access the instance fields and static fields of super class in a subclass.

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

- Change `Circle.screenSize` to `Shape.getScreenSize()` since `screenSize` is a private static field.
- Change `this.x` to `super.getX()` since `x` is a private field of supper class. Change other methods in the same way.

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

5. Using the `protected` keyword

We can find that `Circle` is inconvenient to access the private attributes of superclass; we can consider making these frequently used attributes accessible to subclass. The `protected` keyword can help us.

- Change `x`, `y` and `color` from `private` to `protected`.
- Then we change `isInBoundary()` back to the original one except `Shape.getScreenSize()`, it now works well. Change other methods in the same way.

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

The access right of each access modifier is shown as follows:

	private	default	protected	public
Same package same class	ok	ok	ok	ok
Same package other classes		ok	ok	ok
Other packages other classes. Inheritance			ok	ok
Other packages other classes. No inheritance				ok

Exercise

Modify the given class `Rectangle` to make it inherits from class `Shape`, and:

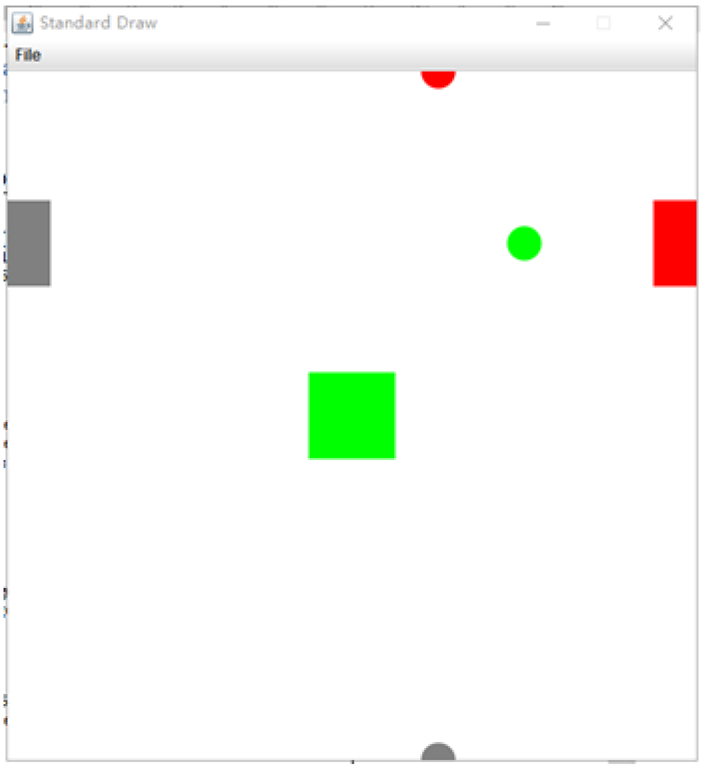
- Modify the constructors and other methods of `Rectangle`.
- Modify `toString()` method.

Run the following `ShapeTest` to test your modifications.

```java
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();
    }
}
```

Output

```
Circle{radius=0.1, x=1.0, y=1.0, color=GRAY}
Circle{radius=0.1, x=1.0, y=1.0, color=GREEN}
Circle{radius=0.1, x=0.5, y=2.0, color=RED}
Rectangle{width=0.5, height=0.5 x=0.0, y=0.0, color=GRAY}
Rectangle{width=0.5, height=0.5 x=0.0, y=0.0, color=GREEN}
Rectangle{width=0.5, height=0.5 x=2.0, y=1.0, color=RED}
```


Standard Draw
File