Basis of Computer Programming (java A)
Tutorial 10

[Experimental Objective]

- Learn inheritance.
- Learn protected keyword.
- Learn polymorphism.

[Before Exercises]

1. Inheritance exercise.

Step1: Download **lab10_base_src.zip** from sakai which is modified base on the last exercise. We should find that:

1. Both **Circle** and **Rectangle** have some common fields, for example, screenSize, x, y and CircleColor.
2. Most methods of **Circle** and **Rectangle** have the same function.

We can refactor the code.

The idea of inheritance is simple but powerful: When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, 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) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass. (https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html)

Sometimes if we found that two classes have many common attributes, we can also use inheritance to design a supper class. The key point is that relation between subclass and super class is "is a".

We can find that **Circle** is a Shape, **Rectangle** is a Shape too. So, we can design a supper class named **Shape,** which including some common attributes and methods.

Create a class Shape, which contains:

(1) the following attributes:

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

Notice that **CircleColor** should be changed to **ShapleColor.**

(2) getter/setter methods for the private variable;

(3) the toString() method (override the method of the Object class) to output the property of the **Shape** object;

Step2: Redefine **Circle**, make it **extends Shape**
Now the Circle only have two attributes: radius and DEFAULT_RADIUS.

```
public class Circle extends Shape{
    private double radius;
    static final int DEFAULT_RADIUS = 5;
```

Step3: Learn how to use super ().

In the constructor of Circle, we will find that some errors occur.

```
    this.x = x;
    this.y = y;
    this.radius = radius;
```

Now x and y are the attributes of **Shape.** A recommend way is use the constructor of super class to initial the supper class.
For example:

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

this means the object itself, **super** is a keyword, object can use it to access the members and constructors of its supper class.
Rewrite other constructors in the same way.

Step4: Learn how to 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:

```
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;
}
```

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, and so on.

```
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;
}
```

Change other methods in the same way.

2. Learn protected keyword

We can see Circle is inconvenient to access the private attributes of superclass, so we can consider that make these frequently-used attributes accessible to subclass.

Protected can help us.
Step1: Change **x, y** and **color** from **private** to **public**.

```
protected double x;
protected double y;
protected ShapeColor color = ShapeColor.GRAY;
```

Step2: Then we change the isInBoundary() back to the original one except Shape.getScreenSize(), it can work well .

```
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 other methods in the same way.

Step3: Learn access modifier:

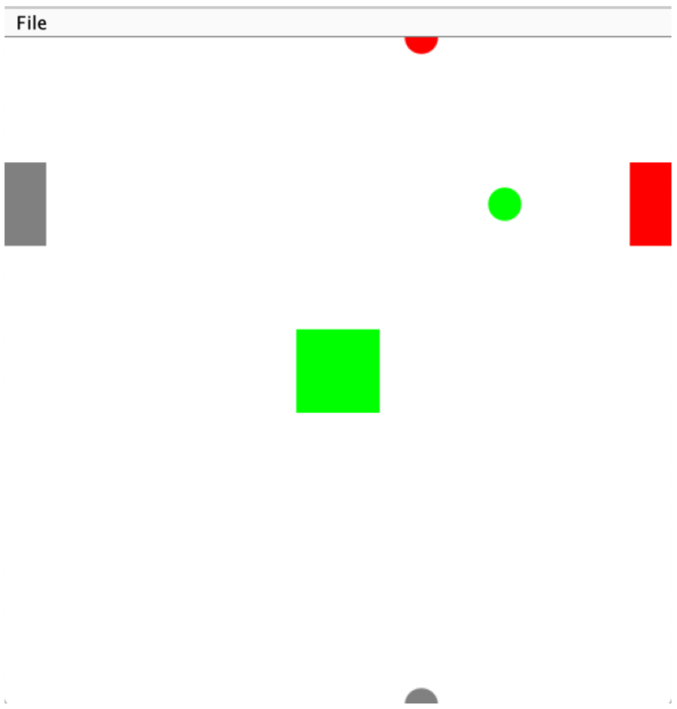
	private	default	protected	public
Same package same class	√	√	√	√
Same package other classes		√	√	√
Other packages Other classes Inheritance			Inherit but cannot access directly	√
Other packages Other classes No inheritance				√

[Exercises]

1. Modify the class **Rectangle** in **lab10_base_src.zip**.
   a. Make **Rectangle** extends **Shape**.
   b. Modify the constructors of **Rectangle**
   c. Modify other methods of **Rectangle**.
   d. Modify **toString()** method.

Run **ShapeTest** to test our modifications.
Output:

```
The postion of the circle is (1.00, 1.00), the radius is 0.10, GRAY, Haven't tested.
The postion of the circle is (1.00, 1.00), the radius is 0.10, GREEN, The shape is in the Screen.
The postion of the circle is (0.50, 2.00), the radius is 0.10, RED, The shape is not in the Screen.
The postion of the rectangle is (0.00, 0.00), the width is 0.50, the height is 0.50, GRAY, Haven't tested.
The postion of the rectangle is (0.00, 0.00), the width is 0.50, the height is 0.50, GREEN, The shape is in the Screen.
The postion of the rectangle is (2.00, 1.00), the width is 0.50, the height is 0.50, RED, The shape is not in the Screen.
```



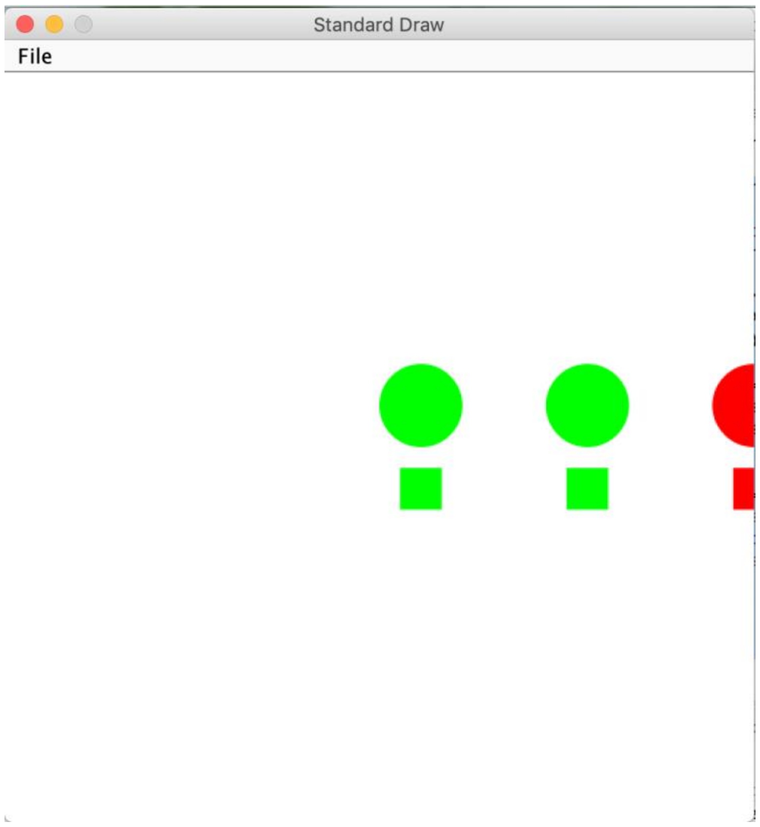
2. Polymorphism

1. Create a class **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.println(shapeList.get(i));
                        shapeList.get(i).draw();
                }

        }

}
```

2. Run above code, observe the result:

```
The postion of the circle is (1.00, 1.00), the radius is 1.00, GREEN, The shape is in the Screen.

The postion of the rectangle is (1.00, -1.00), the width is 1.00, the height is 1.00, GREEN, The shape is in the Screen.

The postion of the circle is (5.00, 1.00), the radius is 1.00, GREEN, The shape is in the Screen.

The postion of the rectangle is (5.00, -1.00), the width is 1.00, the height is 1.00, GREEN, The shape is in the Screen.

The postion of the circle is (9.00, 1.00), the radius is 1.00, RED, The shape is not in the Screen.

The postion of the rectangle is (9.00, -1.00), the width is 1.00, the height is 1.00, RED, The shape is not in the Screen.
```



3. From above sample, we can see that the Java virtual machine (JVM) calls the appropriate method for the object that is referred to in each variable. It does not call the method that is defined by the variable's type. This behavior is referred to as *virtual method invocation* and demonstrates an aspect of the important polymorphism features in the Java language.
   (Reference link: https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html)