# Basis of Computer Programming (java A)
## Tutorial 11

*(The original document was designed by ZHAO Yao in 2018/12, and given simple modified by ZHU Yueming in 2019/4)*

**[Experimental Objective]**

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

**[Before Exercises]**

**1. Inheritance exercise.**

**Step1:** Download **sourceCode_lab11.zip** from sakai, and read carefully about it. From which we can 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 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 both in **Circle** and **Rectangle**, then for those common attributes are more appropriated to be extracted into a super class named **Shape**, from which the subclass can use all attributes and methods.

Create a class Shape, which contains following steps:

(1) Refactor the name of enum class **CircleColor** to **ShapeColor**

```
public enum ShapeColor {
    GREEN("The circle is in the Screen", Color.GREEN),
    RED("The circle 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;
    }
```

(2) Adding attributes:

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

(3) Adding constructors with two parameters **x** and **y** in the Shape class.

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

(4) Adding getter/setter methods for the private variable;

(5) 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;
}
```

**Step2:** Redefine **Circle**, make it **extends Shape**

Now the Circle only has two attributes: radius and DEFAULT_RADIUS.

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

**Step3:** Learn how to use super ().

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

```
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;
}
```

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) {
    super(0,0);
```

```
        this.radius = radius;
    }
```

**this** serves as the current object, while **super** serves as the only supper class for current object.
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 find that Circle is inconvenient to access the private attributes of superclass, so we can consider that make these frequent-used attributes accessible to subclass.
**Protected** can help us.
**Step1:** Change **x, y** and **color** from **private** to **protected**.

```
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() > x - this.width / 2
            || Shape.getScreenSize() < x + this.width / 2) {
        return false;
    }
    if (-1 * Shape.getScreenSize() > y - this.height / 2
            || Shape.getScreenSize() < y + this.height / 2) {
        return false;
    }
    return true;
}
```

Change other methods in the same way.

**Step3:** Learn access modifier:

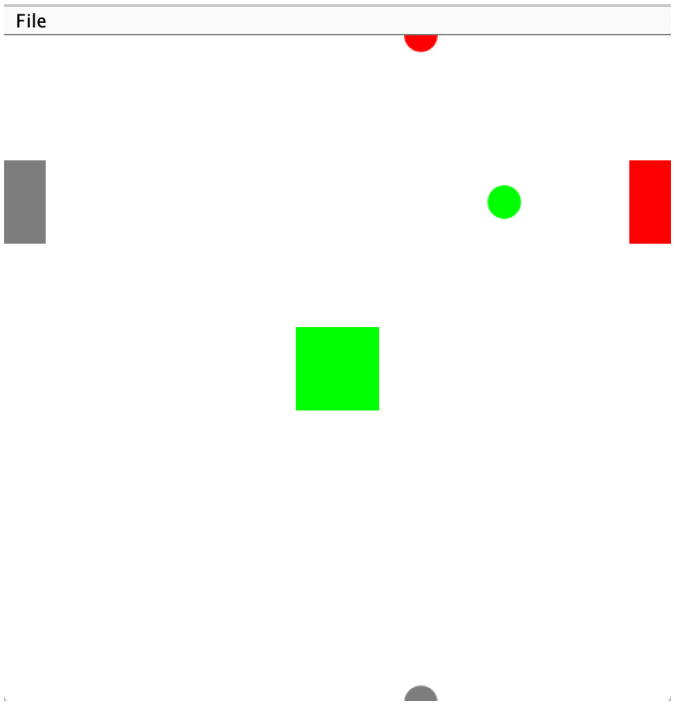
| | private | default | protected | public |
|---|---|---|---|---|
| Same package same class | √ | √ | √ | √ |
| Same package other classes | | √ | √ | √ |
| Other packages Other classes Inheritance | | | Inherit but cannot access directly | √ |
| Other packages Other classes No inheritance | | | | √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

**[Exercises]**

1. Modify the class **Rectangle** in **sourceCode_lab11.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:

```
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}
```




2. Polymorphism
   1. Create a class **PolymorphismTest:**

```
public class Polymorphism {
    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();
        }
    }
}
```

Obviously, two mistakes would arise in checkColor() and draw(). Although we understand those two methods have been defined in both Circle and Rectangle class, when we using subclass to instantiate their super class, we cannot invoke them directly if they haven't been defined in their super class Shape.
Define those two methods in Shape:

```
public void checkColor() {
}

public void draw() {
}
```

2. Run above code, 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}