Tutorial of Polymorphism and Abstract Class

> 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
>
> Update polymorphism introduce and add Shopping Mall exercise by ZHU Yueming in 2023. Nov. 24th.

Objective

- Learn polymorphism
- Learn Abstract class

Part 1: Polymorphism

1.1 Create a ShapeTest class

We want to add several circles and rectangles in to one screen, check the color of those and then draw them, in this case we can use following statements:

```java
public static void main(String[] args) {
        Shape.setScreenSize(9);
        StdDraw.setXscale(-Shape.getScreenSize(), Shape.getScreenSize());
        StdDraw.setYscale(-Shape.getScreenSize(), Shape.getScreenSize());
        ArrayList<Circle> circleList = new ArrayList<>();
        ArrayList<Rectangle> rectangleList = new ArrayList<>();

        for (int i = 0; i < 3; i++) {
            circleList.add(new Circle(1, 4 * i + 1, 1));
            rectangleList.add(new Rectangle(4 * i + 1, -1, 1,1));
        }
        for (Circle circle : circleList) {
            circle.checkColor();
            System.out.println(circle);
            circle.draw();
        }
        for (Rectangle rectangle: rectangleList) {
            rectangle.checkColor();
            System.out.println(rectangle);
            rectangle.draw();
        }
    }
```

However, if we want to add other shapes like Triangle, Five-Points-Star, Hexagon etc into screen. How to design the code?

It is not a better design to create one Colloction for each type of Shapes. Then we can use Polymorphism to accomplish the requirement.

1.2 Reference of Superclass is Instantiated by Subclass

Try the code:

```
Shape s1 = new Circle(1,2,3);
Shape s2 = new Rectangle(-1,0,1,2);
```

In this case, we can add all circles and rectangles into a Collections of `Shape` type.

1.3 Reference of superclass can invoke the method in Subclass

Two prerequisites are required:

- The method of the same signature should be defined both in superclass and subclass.
- The reference of the superclass is instantiated by subclass.

Try the code:

In `Shape` class, add two methods:

```
public void checkColor(){}
public void draw(){}
```

In `ShapeTest` class, do following code:

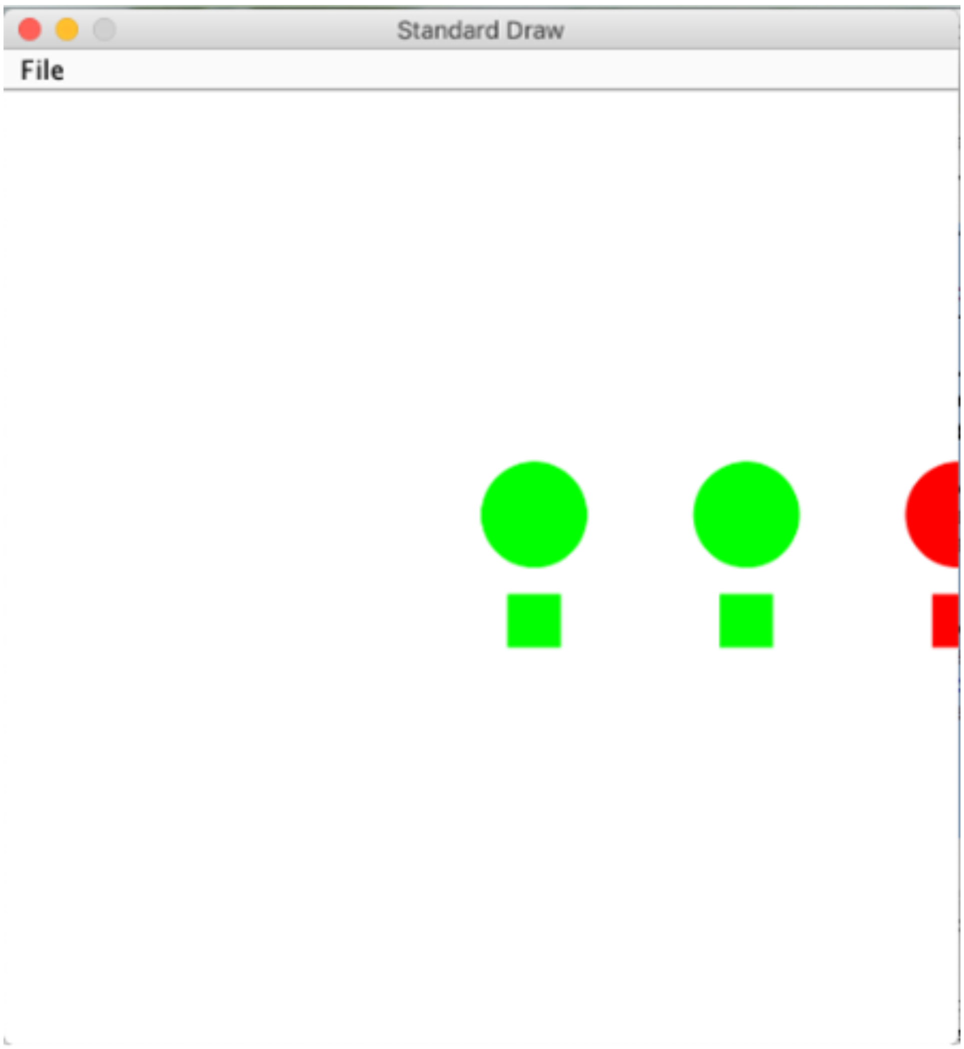
```
s1.checkColor();
s1.draw();
s2.checkColor();
s2.draw();
```

1.4 Update the original ShapeTest class

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

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}
```



Part 2: Abstract Class

Start from the codes you finished in the previous exercise. We can see that there are two public methods, which have no valid code.

```
public void checkColor(){}
public void draw(){}
```

In this case, it is no necessary to instantiate a Shape reference by a Shape type. Then, we can update the Shape class to an abstract class.

- Add `abstract` before class:

  ```
  public abstract class Shape
  ```

- Change `draw()` and `checkColor()` method to abstract method:

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

After that, in ShapeTest, try to write the following code:

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

There will be an error: Cannot instantiate the type Shape.

Exercise

Exercise 1: Animal

> Exercise of Polymorphism and Abstract Class
>
> Basic Exercise

Class Animal

Create an abstract class `Animal`, which contains a public abstract method `speak()` that has no return values.

Monkey and Human

Create a class `Monkey`, which extends from `Animal`. Please implement the abstract method `speak()`, and make it simply print "aaaa" to the console to simulate how monkeys make sound.

Create a class `Human`, which extends from `Animal`. Please implement the abstract method `speak()`, and make it print "Hello World!" to the console.

AnimalTest

Create a class `AnimalTest`, which contains a `main` method doing the following things:

- The main methods creates an `Animal` array of size 6, named `animals`
- For each array element, if the index is an even number (i.e., 0, 2, 4), make the element point to a new `Monkey` object; otherwise, make the element point to a new `Human` object.
- Iterate through the array using the following for loop: `for (Animal a : animals) { a.speak();}`

If your code is correct, the main method should print the following content:

```
aaaa
Hello World!
aaaa
Hello World!
aaaa
Hello World!
```

Exercise 2: Shopping Mall

> The fifth assignment in 2022 Fall semester. The question below only account for 20% of the whole assignment.
>
> Contributes: Zhu Yueming (designer), Liu Shengqi, Huang Huihui (junit), Qin Yao (document), Zhao zhao (tester)
>
> Exercise of Polymorphism and Abstract Class

In the shopping mall, there are two types of members, each type has a different rule of discount. The items sold in the shopping mall are divided into 7 categories. Members can place orders in the 7 categories. Can you help him to design the shopping mall?

In this shopping mall, there are two types of members, one is `GoldCardMember`, the other is `SilverCardMember`. In this question, you are going to build an abstract class `Member` and implement class `GoldCardMember` and class `SilverCardMember` based on the class `Member`.

Class Member:

Create Member as an abstract class, GoldCardMember and SilverCardMember are the subclasses of the Member class.

1. Fields:

```
private String memberId; //Every member have an id and the id is unique for
each member
private char gender; //Gender can only be 'F' or 'M'
private int age; //Age can only be a positive integer
```

2. Methods:

Constructor:

```
public Member(String info)
```

Type of info follows the format below:

```
[memberId] [gender] [age]
```

It is guaranteed in testcases:
Gender can **only** be 'F' or 'M'.
Age can **only** be a positive integer.
Example:
`M01 M 28`

consume:

```
public abstract double consume(int amount);
```

Computes the money actually spent when buying an item with an original price of `amount`.
Return:
return how many money the member actually spent.
Notice:
This is an abstract method. You need to implement it in the subclasses according to the different kinds of discounts for the two types of members.

- **For `GoldCardMember`:**
  The discount strength is tiered, decided by the original price.

The part of original price within the range	Discount strength	Original price	Actual payment	Explanation
<2000	No discount	1500	1500	No discount
2000-5000	5% off	3000	2950	2950 = 2000 + 1000 * 0.95
5000-10000	10% off	8000	7550	7550 = 2000 + 3000 * 0.95 + 3000 * 0.9
10000-20000	15% off	12000	11050	11050 = 2000 + 3000 * 0.95 + 5000 * 0.9 + 2000 * 0.85
>20000	20% off	40000	33850	33850 = 2000 + 3000 * 0.95 + 5000 * 0.9 + 10000 * 0.85 + 20000 * 0.8

- **For `SilverCardMember`:**
  They will get 1 reward point for every 30 yuan they spend. The points can be spent as money in next consumption, 1 point equals 1 yuan. When the total actual payment is more than 10000 yuan, starting from the next consumption, they will get 1.5 points for every 30 yuan they spend.

Original price	Actual payment	Points owned after consumption	Explanation
2000	2000	66	Have no points, get 66 points after consumption
3000	2934	100	Spend 66 points and get 100 points
6000	5900	200	Spend 100 points and get 200 points. Now, the total actual payment is more than 10000 yuan. Starting from the next consumption, 1.5 points will be got for every 30 yuan spent
3000	2800	150	Spend 200 points and get 150 points
4000	3850	199.5	Spend 150 points and get 199.5 points
50	0	151	Spend 50 points and get 1.5 points

getTotalCost:

```
public double getTotalCost();
```

Get the total money spent by the member.
Notice:
You can override the method in subclasses if necessary.

toString

```
public String toString();
```

Return:
A string that shows the information of the member.

- **For `GoldCardMember`:**
  Format:
  `GoldCardMember: [memberId] [gender] [age]`
  Separated by 1 space
  Example:
  `GoldCardMember: M01 M 28`
- **For `SilverCardMember`:**
  Format:
  `SilverCardMember: [memberId] [gender] [age] points=[points owned]`
  Separated by 1 space and the points owned should be **rounded to 1 decimal place**.
  Example:
  `SilverCardMember: H01 F 35 points=0.0`

Notice:
You can override the method in subclasses if necessary.

3. Hint:

The class names should be same as requirements. Do not modify or remove any fields or methods that have been already defined. You can add other classes, fields and methods if you need. You can also override superclass' methods if necessary.

4. TestCode:

```
public class TestMember {
    public static void main(String[] args) {
        test01CreateMember();
        test02SilverCardMemberConsume();
        test03GoldCardMemberConsume();
    }

    public static  void test01CreateMember(){
        Member m1 = new GoldCardMember("M01 M 28 G");
        Member m2 = new SilverCardMember("M02 M 29 S");
```

```java
        System.out.println(m1);
        System.out.println(m2);
    }

    public static void test02SilverCardMemberConsume(){
        Member m1 = new SilverCardMember("H01 F 35 S");
        System.out.println(m1);
        System.out.println(m1.consume(2000));
        System.out.println(m1.consume(3000));
        System.out.println(m1.consume(6000));
        System.out.println(m1.consume(3000));
        System.out.println(m1.consume(4000));
        System.out.println(m1.getTotalCost());
        System.out.println(m1.getTotalDiscountPrice());
    }

    public static void test03GoldCardMemberConsume(){
        Member m1 = new GoldCardMember("C001 M 54 G");
        System.out.println(m1);
        System.out.println(m1.consume(2000));
        System.out.println(m1.consume(3000));
        System.out.println(m1.consume(5000));
        System.out.println(m1.consume(8000));
        System.out.println(m1.consume(12000));
        System.out.println(m1.consume(18000));
        System.out.println(m1.consume(40000));
        System.out.println(m1.getTotalCost());
        System.out.println(m1.getTotalDiscountPrice());
    }

}
```

Result:

```
GoldCardMember: M01 M 28
SilverCardMember: M02 M 29 points=0.0
SilverCardMember: H01 F 35 points=0.0
2000.0
2934.0
5900.0
2800.0
3850.0
17484.0
516.0
GoldCardMember: C001 M 54
2000.0
2950.0
4850.0
```

```
7550.0
11050.0
16150.0
33850.0
78400.0
9600.0
```