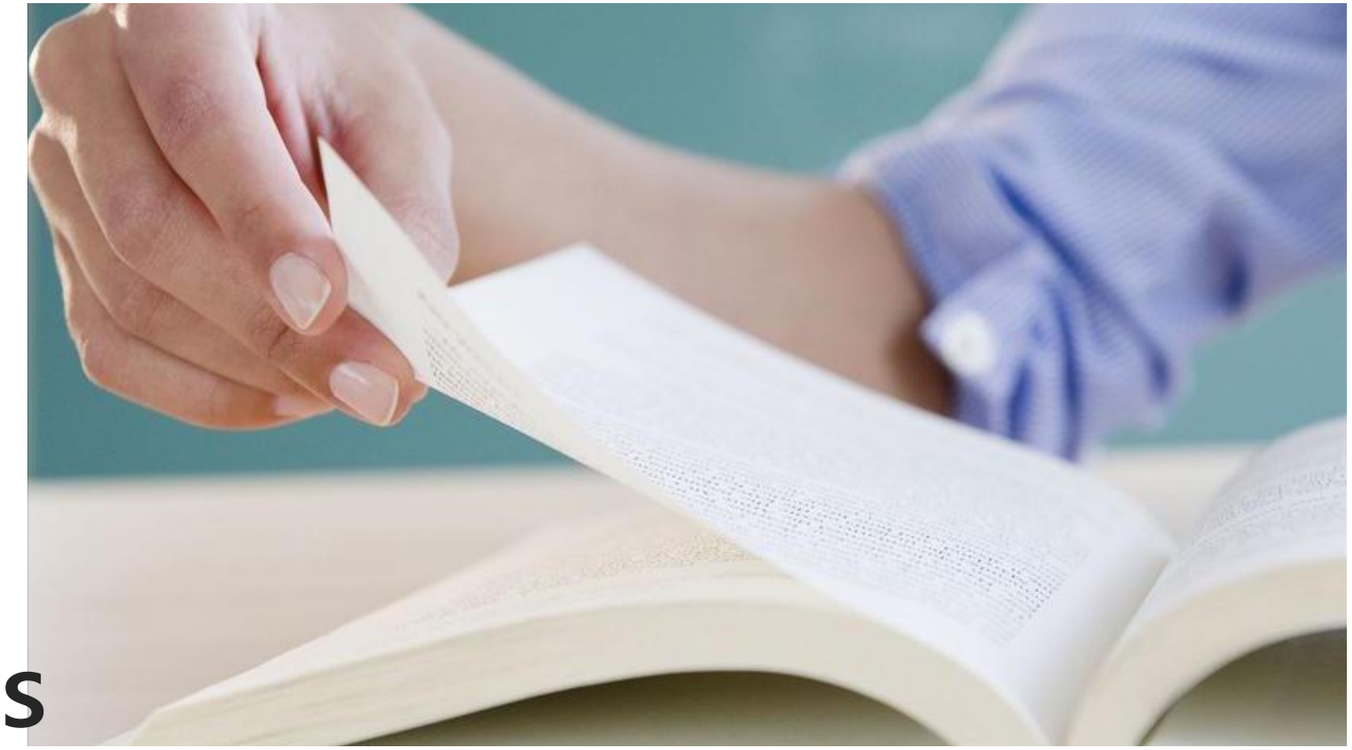
Introduction to computer programming A
LAB11

贾艳红 Jana
Email:jiayh@mail.sustech.edu.cn

LAB OBJECTIVES

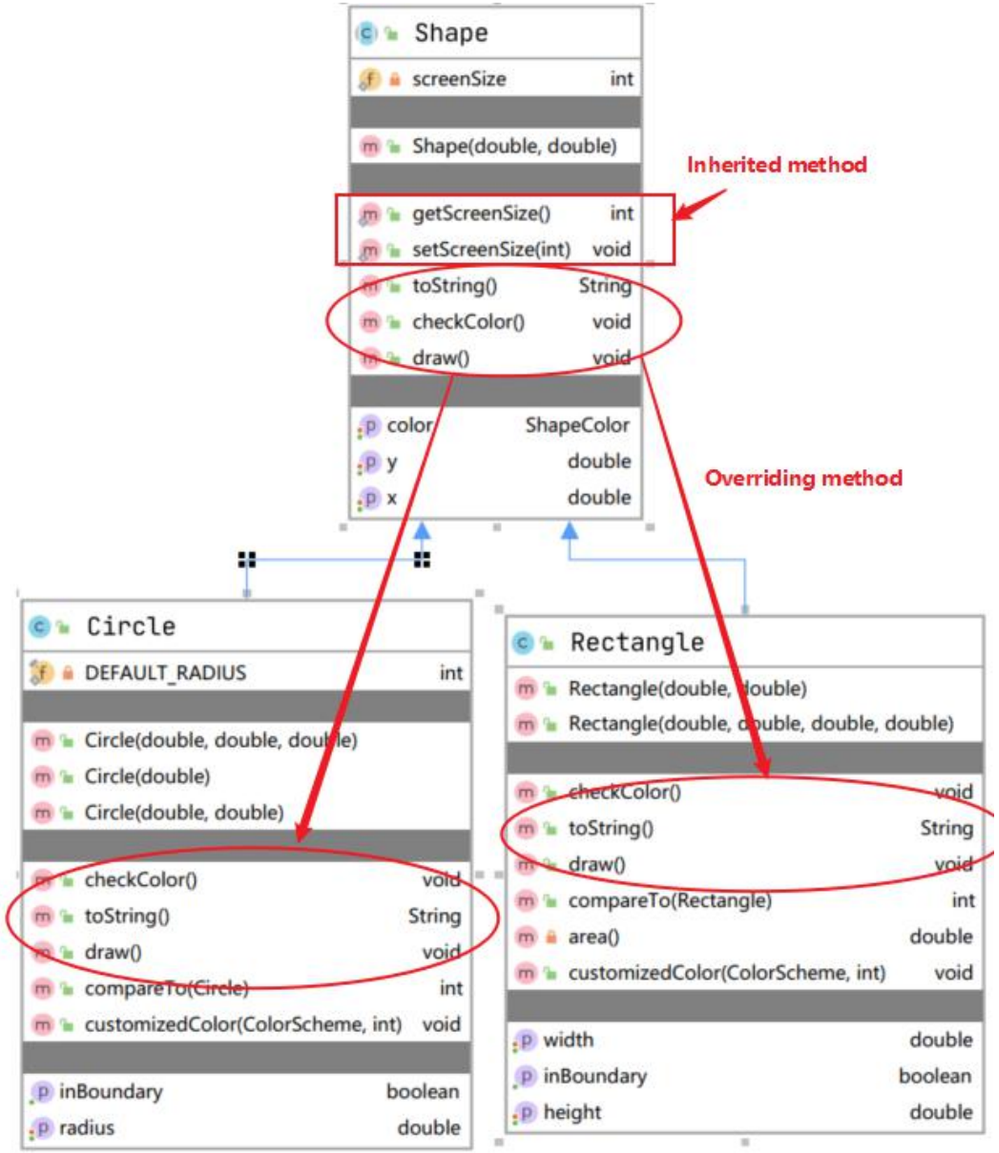
- Learn polymorphism.
- Learn abstract class.
- Learn how to define and implement an interface.
- Learn how to use the interface java.long.Comparable<T>.

knowledge points



Polymorphic

Polymorphism means that a variable of a supertype can refer to a subtype object. When calling a method using polymorphic , first check whether the method exists in the superclass.




```java
import java.util.ArrayList;

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();
        }
    }
}
```

output:

```
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}
```

The method inherited from **super class** can be override in its **sub class** while MUST keep the declaration same(**same method name, same parameter declaration and same return type**) but with different behaviors(statements)

Overriding vs Overloading

overriding

```
public class TestOverriding {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

invoke the p(double i)

the same signature

overloading

```
public class TestOverloading {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

invokes the p(int i)

invokes the p(double i)

the same name
but different parameter lists

Note the following:

■ **Overridden** methods are **in different classes** related by inheritance; **overloaded** methods can be either in the same class, or in different classes related by inheritance.

■ **Overridden methods have the same signature**; **overloaded methods have the same name but different parameter lists.**

Overriding vs Overloading

To avoid mistakes, you can use a special Java syntax, called override annotation, to place @Override before the overriding method in the subclass. For example,

```java
public class TestOverriding {

    public static void main(String[] args) {
        A a = new A();
        a.p( i: 10);
        a.p( i: 10.0);
        B b = new A();
        b.p( i: 10);
        b.p( i: 10.0);
    }
}

abstract  class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overrides the method in B
    @Override
    public void p(double i) { System.out.println(i); }
}
```

Dynamic Binding

```java
public class TestOverriding {

    public static void main(String[] args) {
        A a = new A();
        a.p( i: 10);
        a.p( i: 10.0);
        B b = new A();
        b.p( i: 10);
        b.p( i: 10.0);
    }
}

abstract  class B {
    public void p(double i) {
        System.out.println(i * 2);
    }
}

class A extends B {
    // This method overrides the method in B
    @Override
    public void p(double i) { System.out.println(i); }
}
```

declared type

actual type

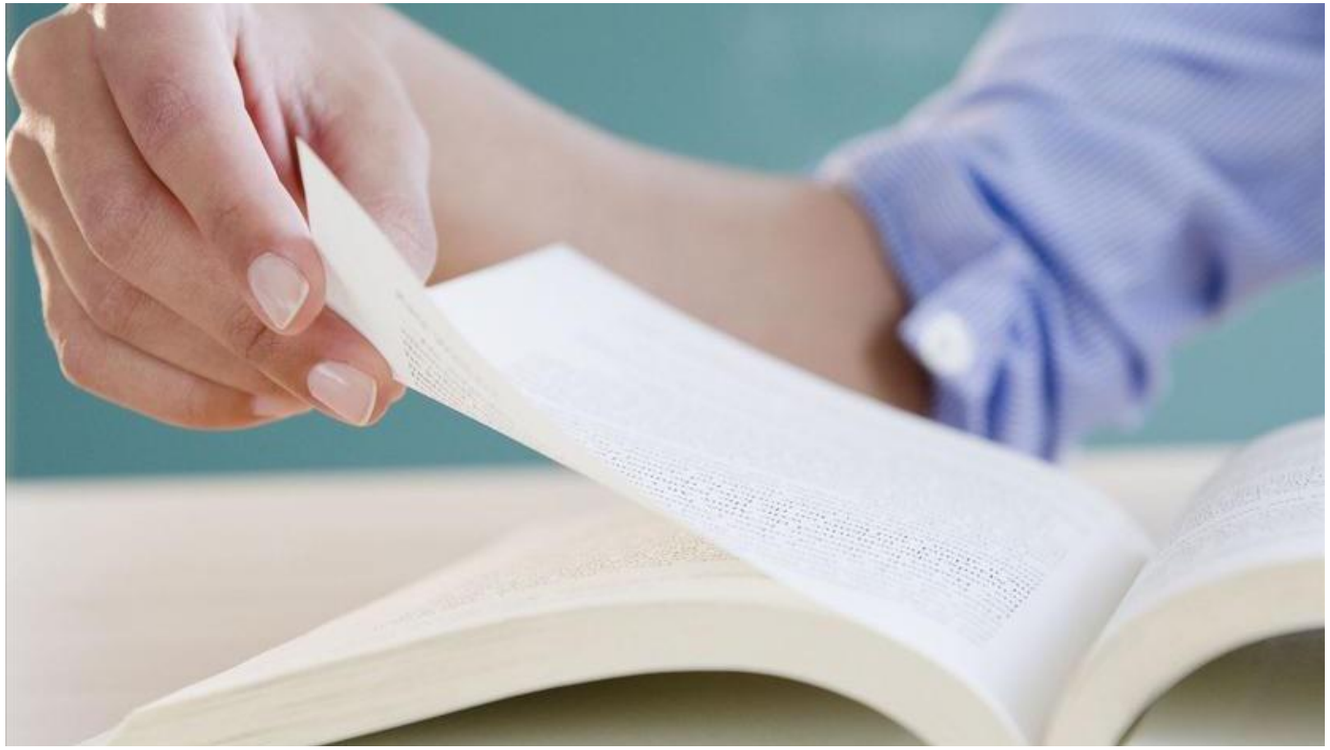
Which p(double i) method is invoked by object? why?

Which **p(double i)** method is invoked by object is determined by object's **actual type.** This is known as dynamic binding.

output:

```
10.0
10.0
10.0
10.0
```

Exercises



Complete the exercises in the **2020S-Java-A-Lab-11.pdf** and submit to the blackboard as required.

THANK YOU

贾艳红 Jana
Email:jiayh@mail.sustech.edu.cn