



Object Oriented Programming Lab

CSE 1206

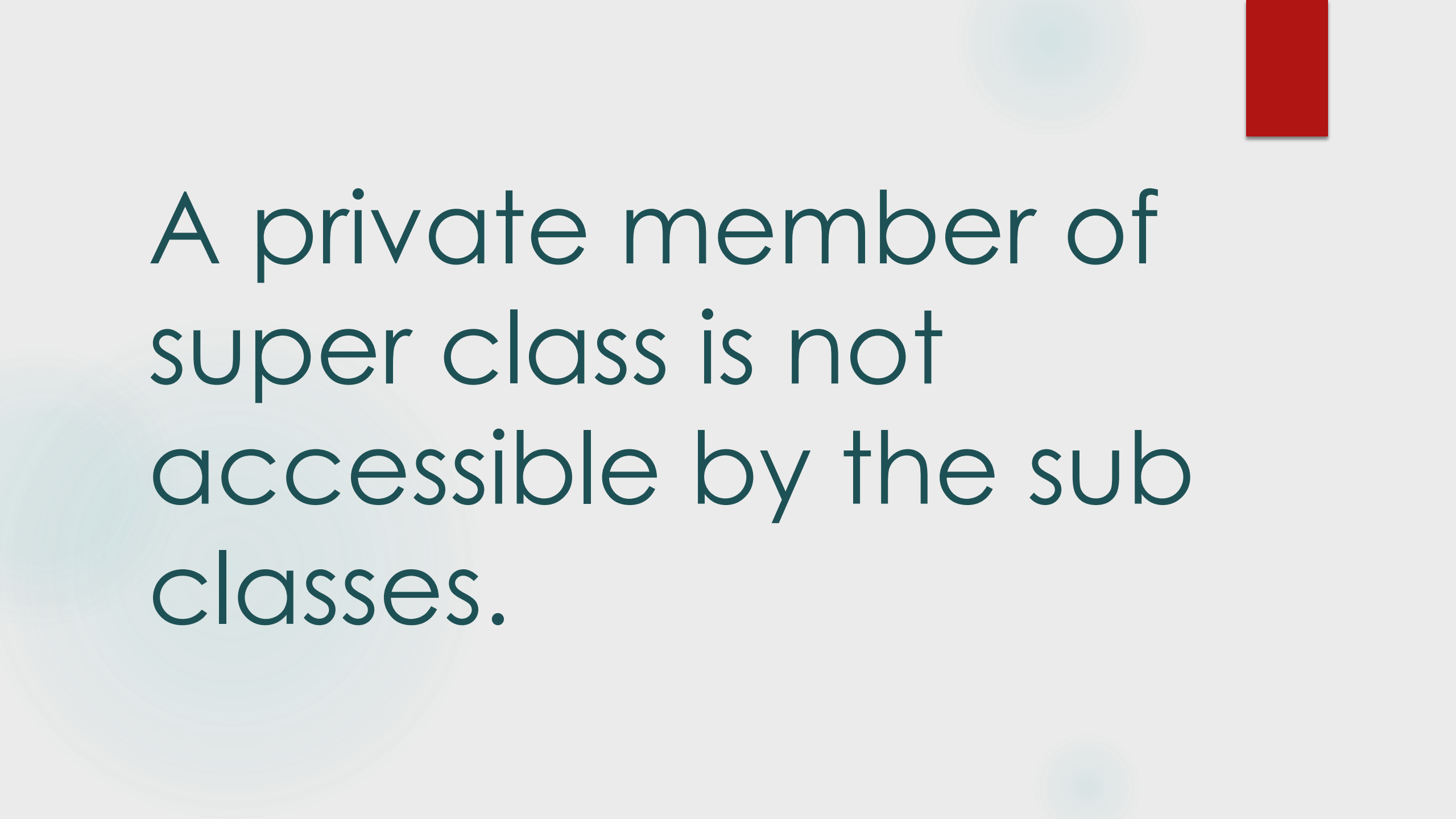


Course Teacher

Dr. Shahriar Mahbub, Professor
Nibir Chandra Mandal , Lecturer
Nowshin Nawar Arony, Lecturer

Contents of the Slide

- ▶ Super Keyword
- ▶ Method Overriding
- ▶ Final Keyword
- ▶ Dynamic Method Dispatch



A private member of
super class is not
accessible by the sub
classes.

```
public class Shape {  
  
    private String color;  
    private boolean filled;  
}
```

Create the getter setter methods.

```
public class Circle extends Shape{
```

```
    double radius;
```

```
    public Circle(double radius, String color, boolean filled) {
```

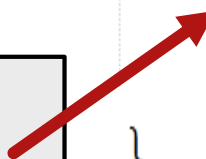
```
        this.radius = radius;
```

```
        this.color = color;
```

```
        this.filled = filled;
```

```
    }
```

Error:
color has
private
access in
Shape



Solution

```
public class Circle extends Shape{  
  
    double radius;  
  
    public Circle(double radius, String color, boolean filled) {  
        super(color, filled);  
        this.radius = radius;  
    }  
}
```

- 
- Now convert all variables in all classes to private
 - declare getter setter methods for each variable
 - Use super wherever needed.

If both super class and sub class have a variable with same name

```
public class Shape {  
  
    String color;  
    boolean filled;  
}
```

```
public class Circle extends Shape{  
  
    private double radius;  
    private String color;  
}
```

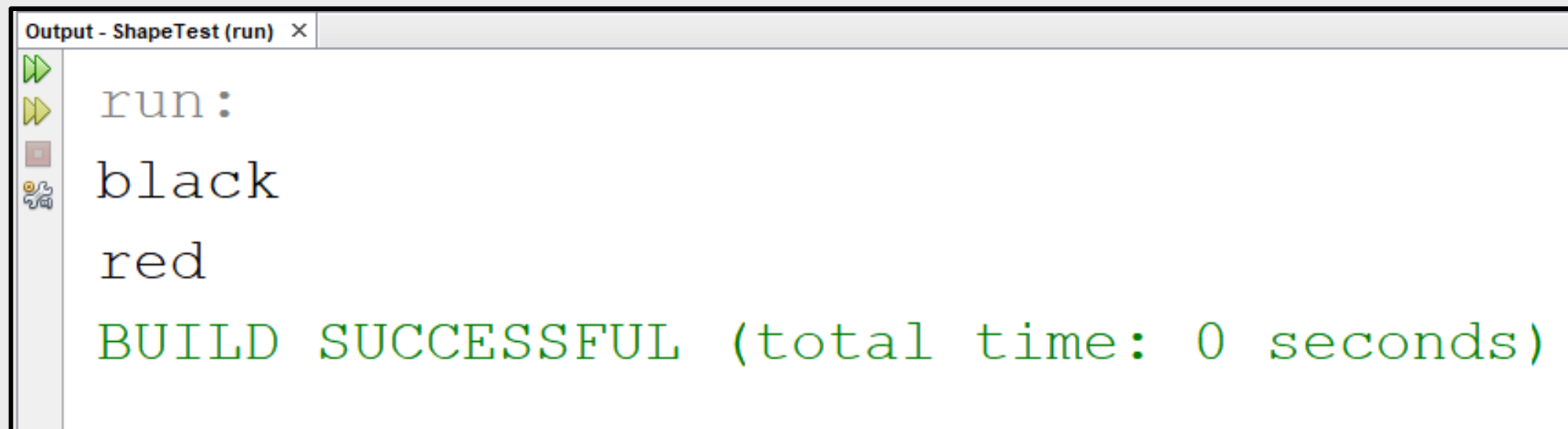
Super keyword is quite similar to the use of **this** keyword.

```
public class Circle extends Shape{

    private double radius;
    String color;

    public void seeColor()
    {
        color = "black";
        super.color = "red";
        System.out.println(color);
        System.out.println(super.color);
    }
}
```

```
public class ShapeTest {  
  
    public static void main(String[] args) {  
  
        Circle obj = new Circle();  
        obj.setColor();  
    }  
  
}
```




```
Output - ShapeTest (run) ×  
run:  
black  
red  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding.

Rules for Java Method Overriding

- 1. method must have same name as in the parent class**
- 2. method must have same parameter as in the parent class.**
- 3. must be in a relationship (inheritance).**

- 
- Now go to Shape
 - Create a method named **calculateArea()** whose return type is double and has no parameter.
 - return 1.0;

```
public class Shape {  
  
    String color;  
    private boolean filled;  
  
    public double calculateArea()  
    {  
        return 0.0;  
    }  
}
```

Now override this method in Circle, Rectangle and Square class



Now go to main class create objects for Shape, Circle, Rectangle and Square classes.

Then call the calculateArea() method for each object and print.


```
public class ShapeTest {  
  
    public static void main(String[] args) {  
  
        Shape obj1 = new Shape();  
        Circle obj2 = new Circle(10.8, "blue", true);  
        Rectangle obj3 = new Rectangle(4.5, 3.2, "ash", false);  
        Square obj4 = new Square(2.0, "brown", true);  
  
        System.out.println("Area of Shape: "+obj1.calculateArea());  
        System.out.println("Area of Circle: "+ obj2.calculateArea());  
        System.out.println("Area of Rectangle: "+ obj3.calculateArea());  
        System.out.println("Area of Square: "+ obj4.calculateArea());  
    }  
}
```

```
System.out.println(obj1.getColor());  
System.out.println(obj1.isFilled());  
  
System.out.println(obj2.getColor());  
System.out.println(obj2.isFilled());  
System.out.println(obj2.getRadius());  
  
System.out.println(obj3.getColor());  
System.out.println(obj3.isFilled());  
System.out.println(obj3.getLength());  
System.out.println(obj3.getWidth());  
  
System.out.println(obj4.getColor());  
System.out.println(obj4.isFilled());  
System.out.println(obj4.getLength());  
System.out.println(obj4.getWidth());
```

Printing all the
values

Real example of Java Method Overriding

Consider the following scenario:

Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example,

BRAC, DBBL and HSBC banks could provide 8%, 7% and 9% rate of interest respectively.

```
public class Bank {  
  
    int getRateOfInterest() {  
        return 0;  
    }  
}
```

```
public class DBBL extends Bank {  
  
    int getRateOfInterest() {  
        return 7;  
    }  
}
```

```
public class BRAC extends Bank {  
  
    int getRateOfInterest() {  
        return 8;  
    }  
}
```

```
public class HSBC extends Bank {  
  
    int getRateOfInterest() {  
        return 9;  
    }  
}
```

Test the method Overriding

```
public class TestOverriding {  
  
    public static void main(String[] args) {  
        BRAC b = new BRAC();  
        DBBL d = new DBBL();  
        HSBC h = new HSBC();  
        System.out.println("BRAC Rate of Interest: " + b.getRateOfInterest() + "%");  
        System.out.println("DBBL Rate of Interest: " + d.getRateOfInterest() + "%");  
        System.out.println("HSBC Rate of Interest: " + h.getRateOfInterest() + "%");  
    }  
}
```

Final Keyword

- Final class -> used to prevent inheritance
- Final variable -> used to create constant variables
- Final methods -> prevents method overriding

Preventing Inheritance Using Final keyword



► Final keyword is used to prevent a class from being inherited.

► Syntax

```
final class A{
```

```
}
```

```
class B extends A { // Error
```

```
}
```



```
public final class Box {  
  
    public double getVolume(double length, double width, double height) {  
        return length * width * height;  
    }  
  
}
```

```
public class GiftBox {  
  
    private double length;  
    private double width;  
    private double height;  
  
    GiftBox(double length, double width, double height) {  
  
        this.length = length;  
        this.width = width;  
        this.height = height;  
    }  
    // this method takes Box object as argument and using the Box  
    // getVolume method it prints the volume of the GiftBox  
    public void printVolume(Box bObj) {  
        double volume = bObj.getVolume(this.length, this.width, this.height);  
        System.out.println("Volume of the GiftBox: " + volume);  
    }  
}
```

```
public class TestFinal {  
  
    public static void main(String[] args) {  
  
        // instantiate object  
        Box bObj = new Box();  
        GiftBox obj = new GiftBox(5, 4, 3);  
  
        // output  
        obj.printVolume(bObj);  
    }  
}
```

Final Method

```
public class Box2 {  
  
    final public String getColor() {  
        return "Yellow";  
    }  
  
}
```

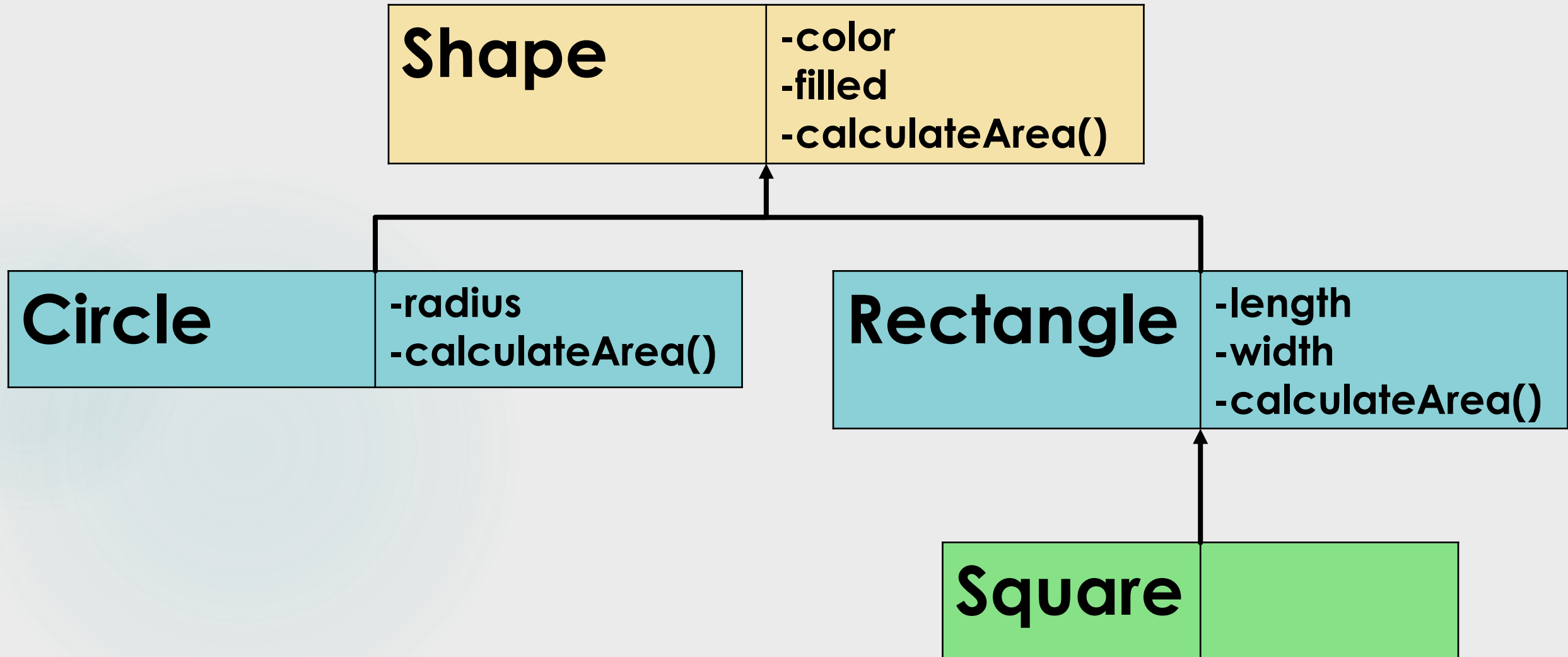
```
public class GiftBox extends Box2 {  
  
    public String getColor() {  
        return "Red";  
    }  
  
}
```

Final Variable

```
public class GiftBox {  
    final public String color="Yellow";  
}
```

```
public class TestFinal {  
    public static void main(String[] args) {  
  
        // instantiate object  
        Box bObj = new Box();  
        GiftBox obj = new GiftBox(5, 4, 3);  
  
        obj.color="Red";  
    }  
}
```

Remember the inheritance example?



A superclass reference variable can refer to a subclass object.

```
Shape obj1 = new Circle(5.5);  
Shape obj2 = new Rectangle(5.5, 6.7);  
  
Rectangle obj3 = new Square(5.5);  
Shape obj4 = new Square(6.5);
```

This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

```
public static void main(String[] args) {
```

```
    Shape obj1 = new Circle(5.5);
```

```
    Shape obj2 = new Rectangle(5.5, 6.7);
```

```
    Rectangle obj3 = new Square(5.5);
```

```
    Shape obj4 = new Square(6.5);
```

```
    System.out.println(obj1.calculateArea());
```

```
    System.out.println(obj2.calculateArea());
```

```
    System.out.println(obj3.calculateArea());
```

```
    System.out.println(obj4.calculateArea());
```


Dynamic Method Dispatch

- ▶ This upcasting is used to call overridden methods.
- ▶ The super class variable will only recognize the overridden methods.
- ▶ When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs.
- ▶ Thus, this determination is made at run time.

Example

```
public static void callArea(Shape shpObj, String type)
{
    if(type.equals("circle"))
    {
        shpObj= new Circle(3.5);
        System.out.println(shpObj.calculateArea());
    }
    else if (type.equals("rectangle")) {

        shpObj= new Rectangle(5,6);
        System.out.println(shpObj.calculateArea());
    }
    else if (type.equals("square")) {

        shpObj= new Square(6.7);
        System.out.println(shpObj.calculateArea());
    }
}
```

```
Shape callObj = null;
callArea(callObj, "rectangle");
```