Software Development and Best Practices

Part 1 : Object-oriented programming Revision

Mahmoud Khalaf Saeed

Section (3)

Abstraction And Interface

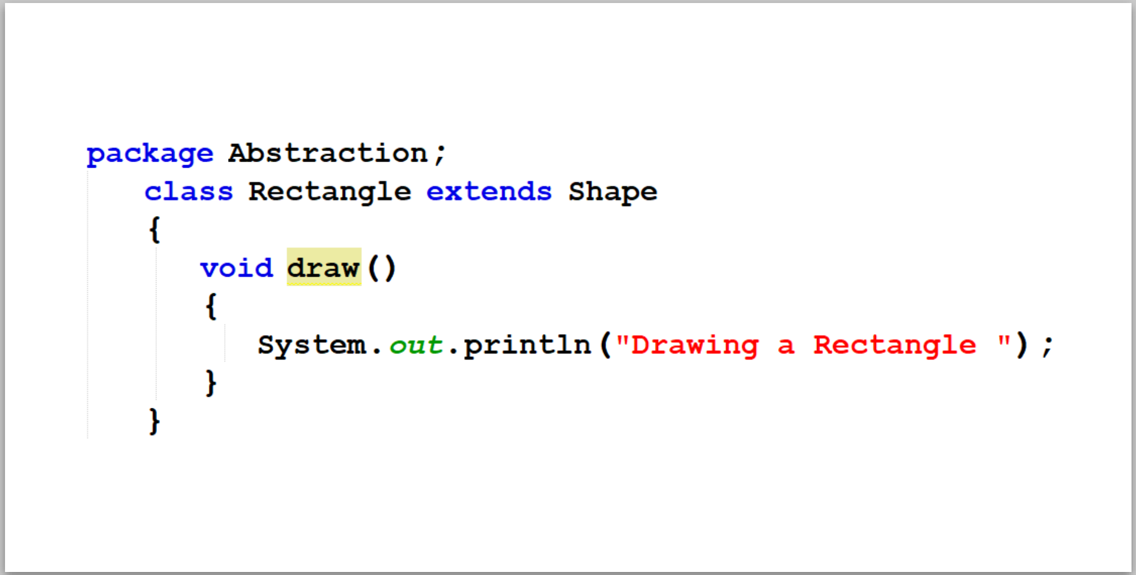
ABSTRACTION

Abstraction : is a process of hiding the implementation details and showing only functionality to the user.

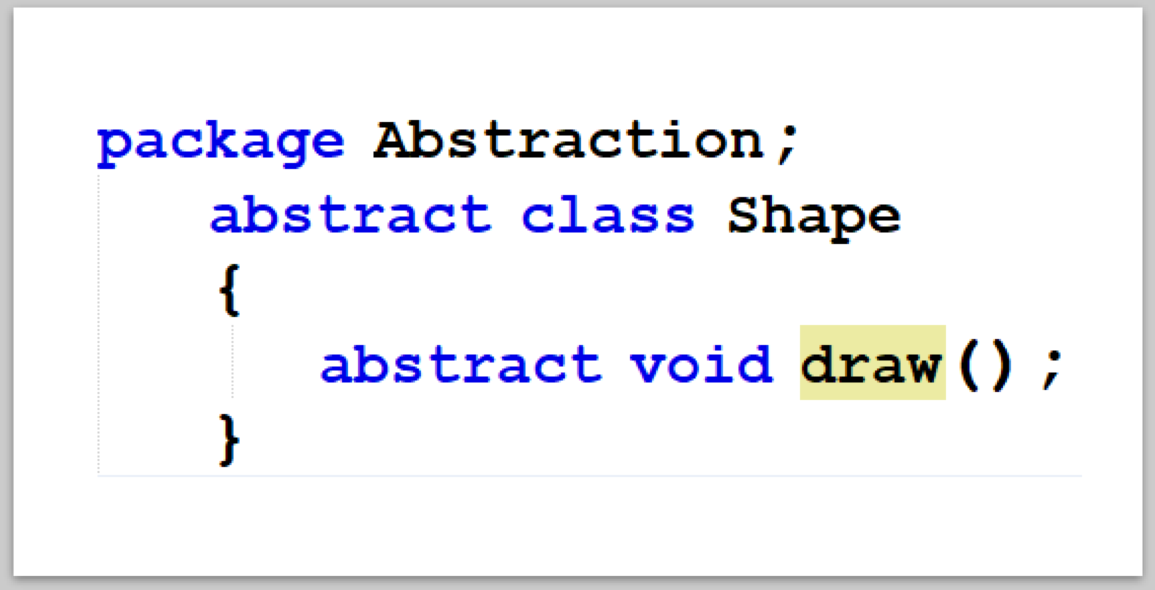
A class which is declared with the abstract keyword is known as an abstract class .

it shows only essential things to the user and hides the internal details,

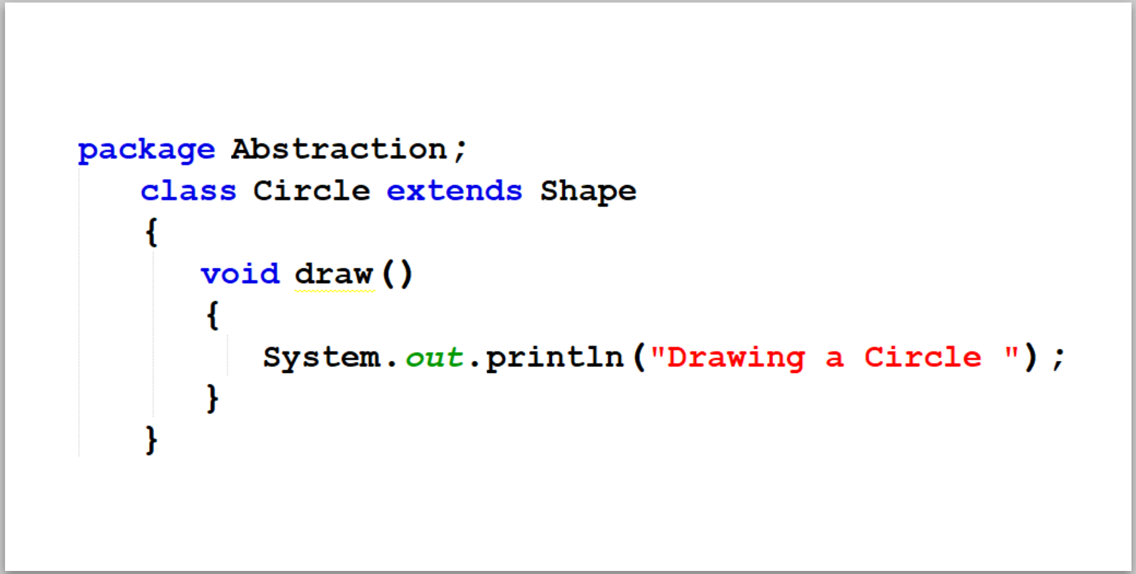
- It can have abstract and non-abstract methods.
- It cannot be instantiated {cuz it’s abstarcted }.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.



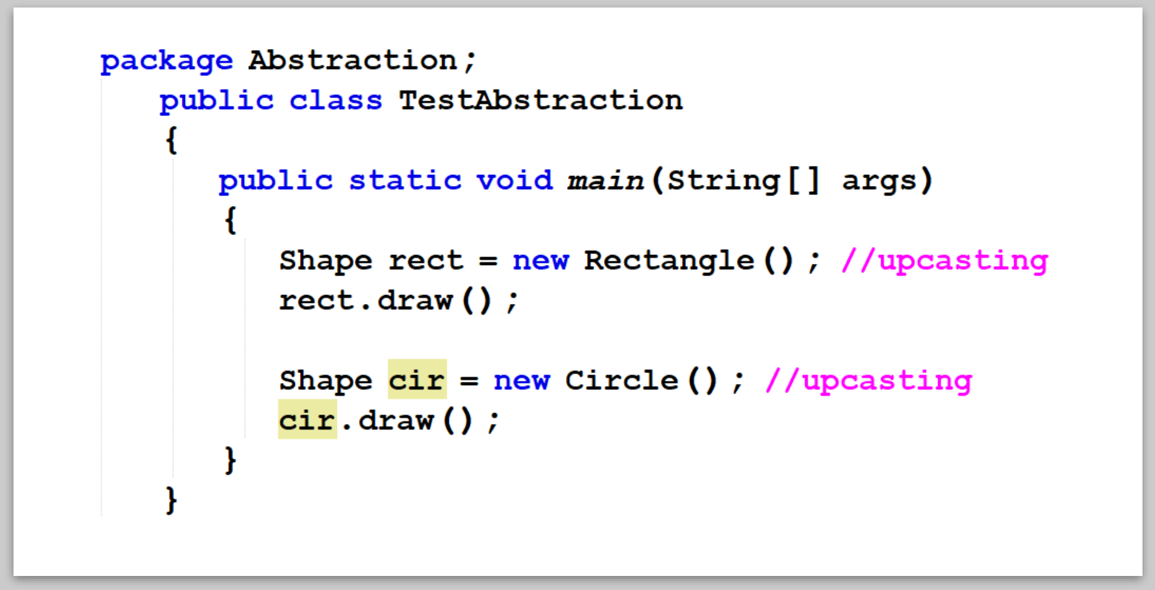
```
package Abstraction;
    class Rectangle extends Shape
    {
        void draw()
        {
            System.out.println("Drawing a Rectangle ");
        }
    }
```



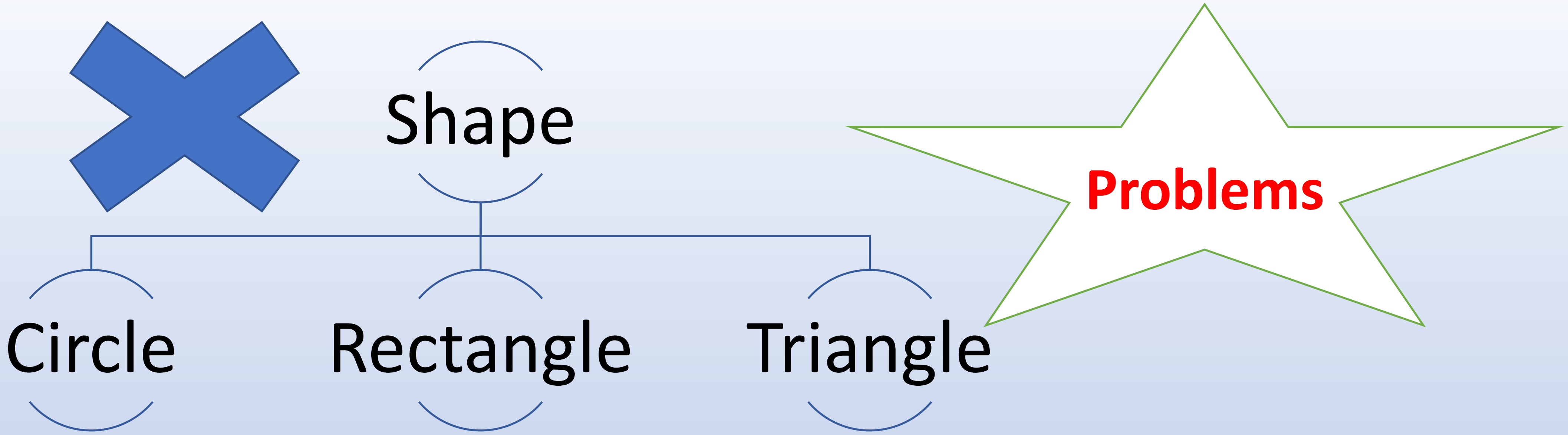
```
package Abstraction;
    abstract class Shape
    {
        abstract void draw();
    }
```



```
package Abstraction;
    class Circle extends Shape
    {
        void draw()
        {
            System.out.println("Drawing a Circle ");
        }
    }
```



```
package Abstraction;
    public class TestAbstraction
    {
        public static void main(String[] args)
        {
            Shape rect = new Rectangle(); //upcasting
            rect.draw();

            Shape cir = new Circle(); //upcasting
            cir.draw();
        }
    }
```



- logically there aren't a shape object
- Area() , Circum() methods implements differently in child classes
- for all Childs Area() , Circum() must be overridden

ject

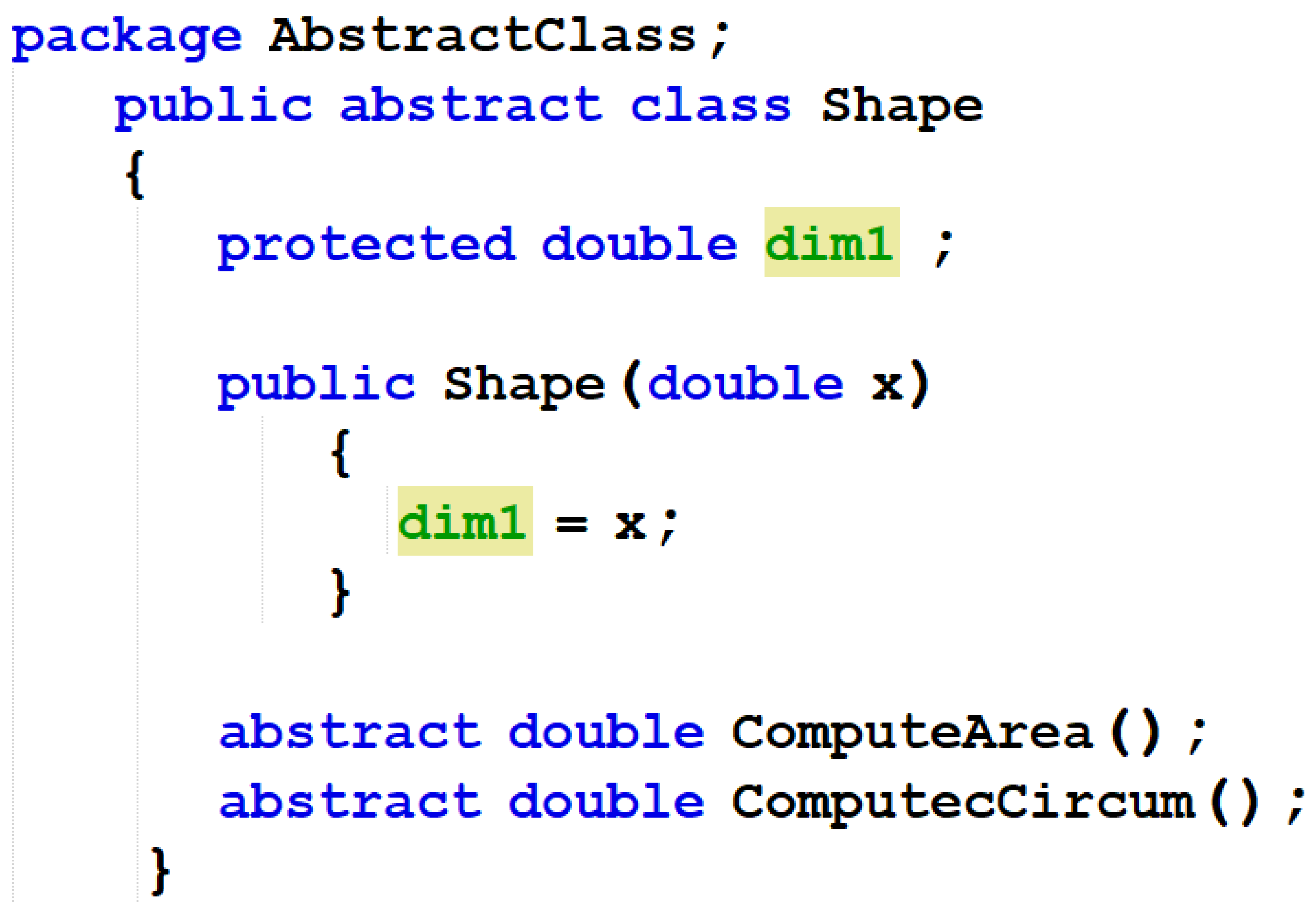
Area() , Circum() me

abstraction

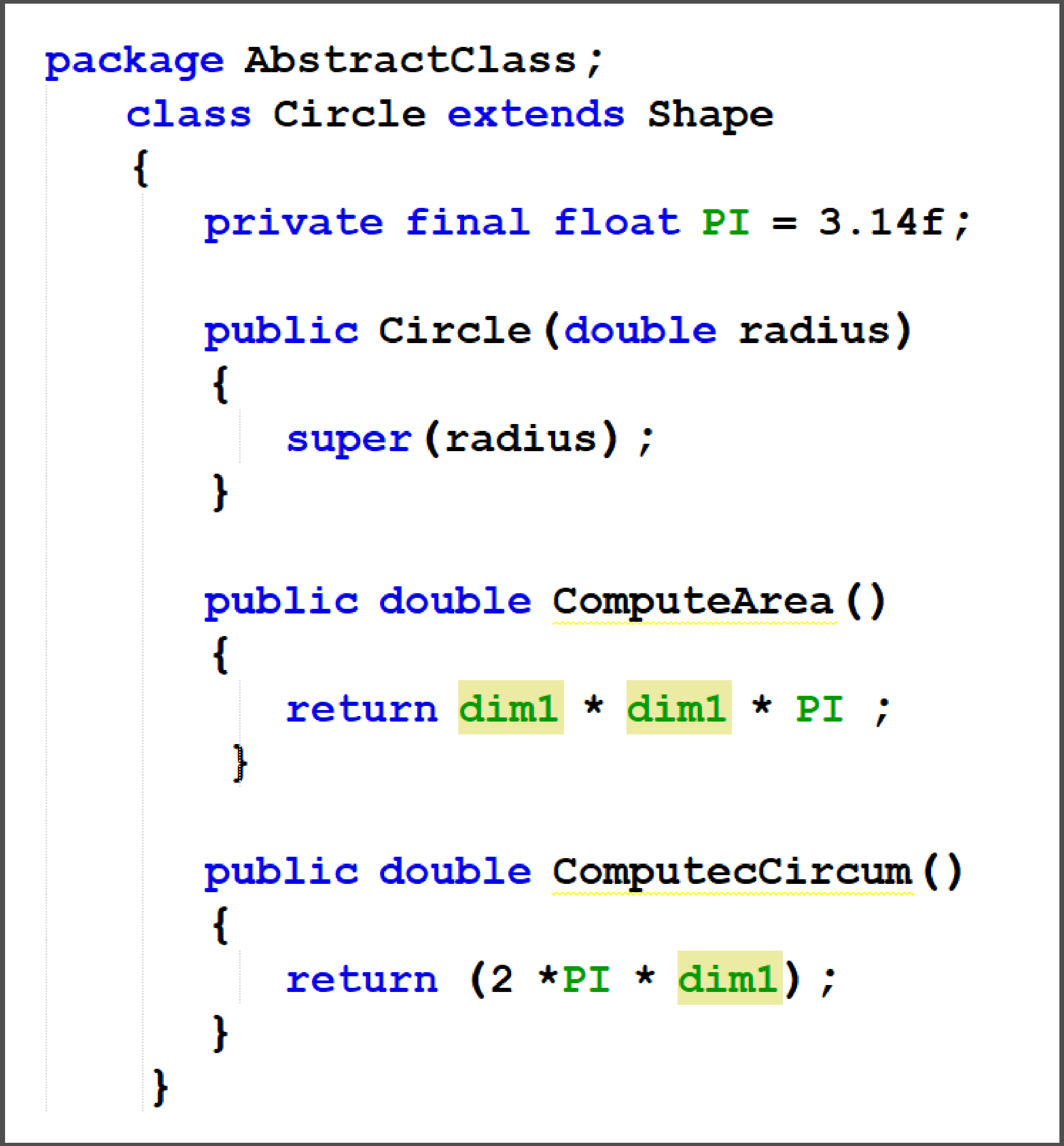
ifferently in child classs

for all

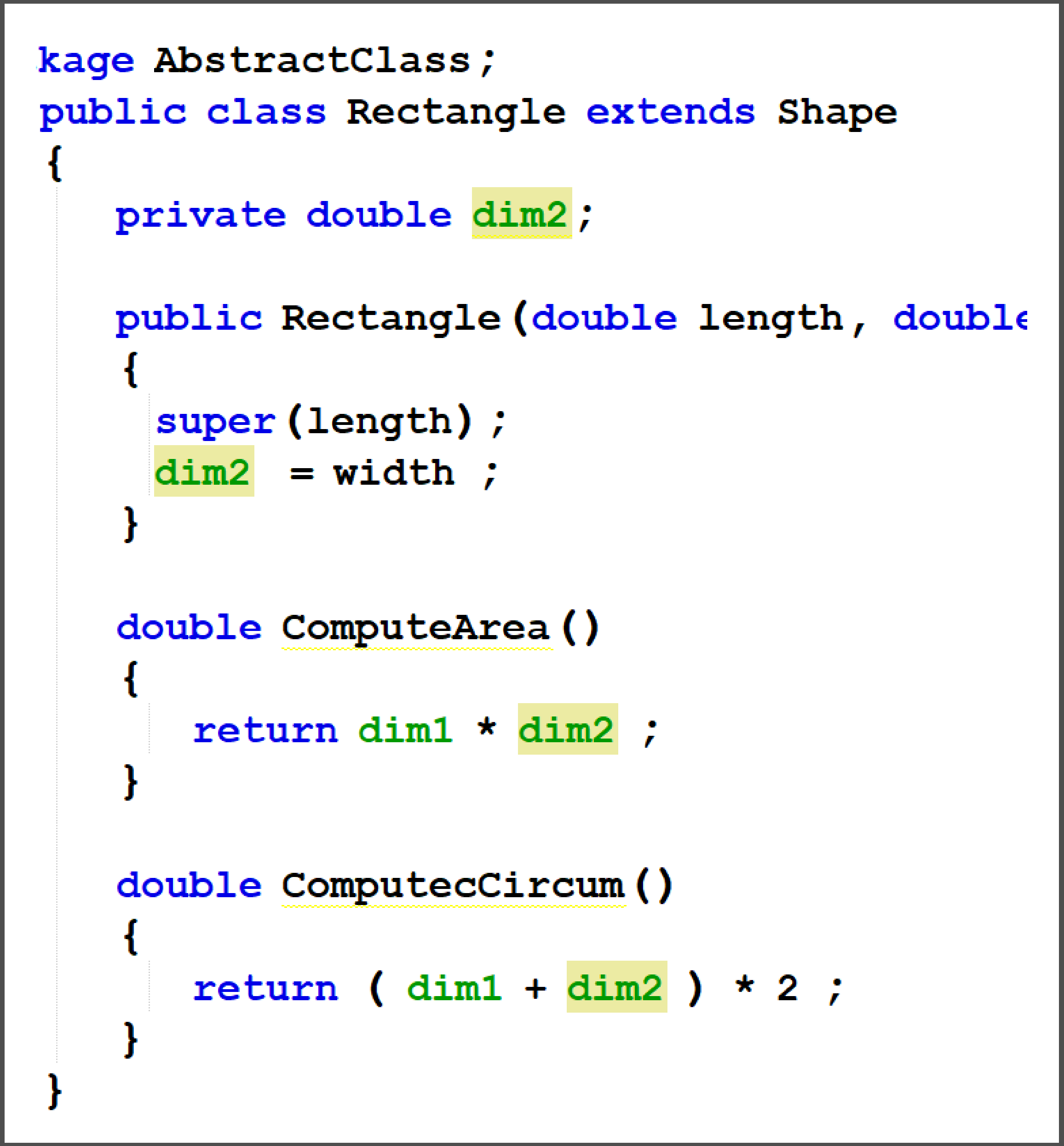
overrided



```java
package AbstractClass;
  public abstract class Shape
  {
     protected double dim1 ;

     public Shape(double x)
       {
          dim1 = x;
       }

     abstract double ComputeArea();
     abstract double ComputecCircum();
   }
```



```
package AbstractClass;
    class Circle extends Shape
    {
        private final float PI = 3.14f;

        public Circle(double radius)
        {
            super(radius);
        }

        public double ComputeArea()
        {
            return dim1 * dim1 * PI ;
         }

        public double ComputecCircum()
        {
            return (2 *PI * dim1);
        }
    }
```

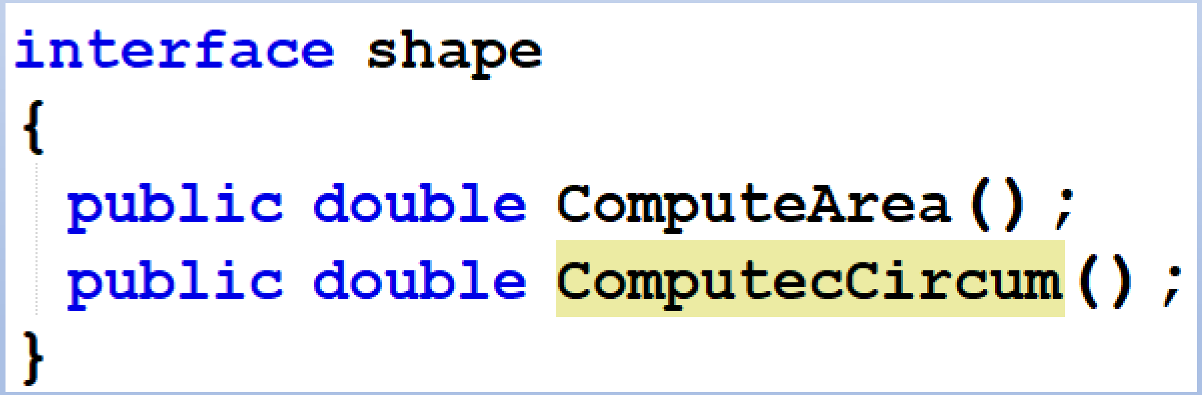


```
kage AbstractClass;
public class Rectangle extends Shape
{
    private double dim2;

    public Rectangle(double length, double
    {
      super(length);
      dim2  = width ;
    }

    double ComputeArea()
    {
        return dim1 * dim2 ;
    }

    double ComputecCircum()
    {
        return ( dim1 + dim2 ) * 2 ;
    }
}
```

```java
package AbstractClass;
    public class TestShape
    {
        public static void main(String[] args)
        {
            // Shape sh = new Shape(); // Shape is abstract; cannot be instantiated
            Circle cir = new Circle(10) ;
            System.out.println("Area of Circle is " + cir.ComputeArea()+ "m^2");
            System.out.println("circumestence of Circle is " + cir.ComputecCircum()+ "m");

            Rectangle rect = new Rectangle(10 , 20) ;
            System.out.println("Area of Rectangle is " + rect.ComputeArea() + "m^2");
            System.out.println("circumestence of Rectangle is " + rect.ComputecCircum()+"m");
        }
    }
```

INTERFACE IN JAVA

- ❑ It is a blueprint of a class. It has static constants and abstract methods.
- ❑ Another way to achieve abstraction in Java.
- ❑ An interface is a completely "abstract class" that is used to group related methods with empty bodies:

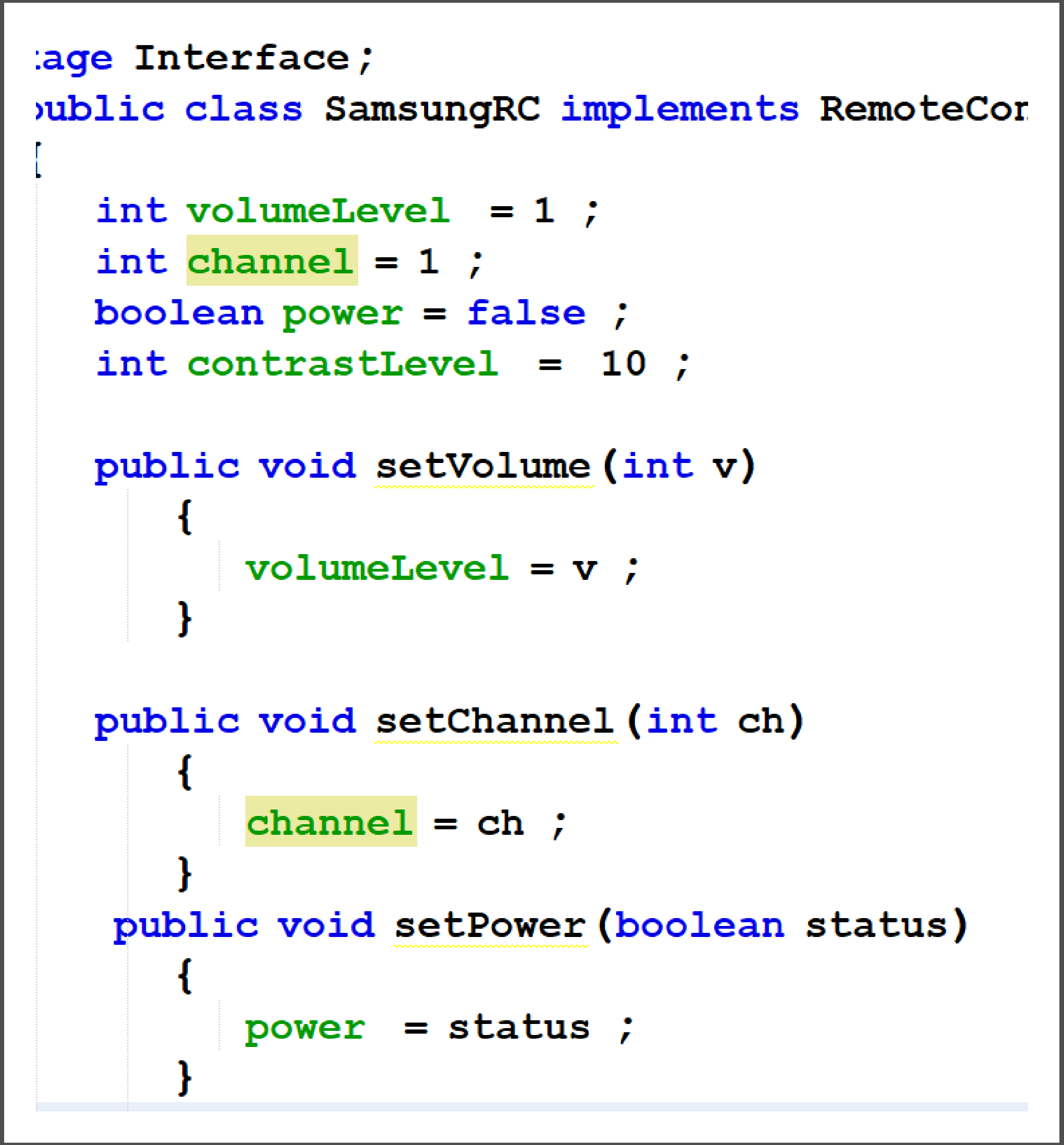


```
interface shape
{
  public double ComputeArea();
  public double ComputecCircum();
}
```

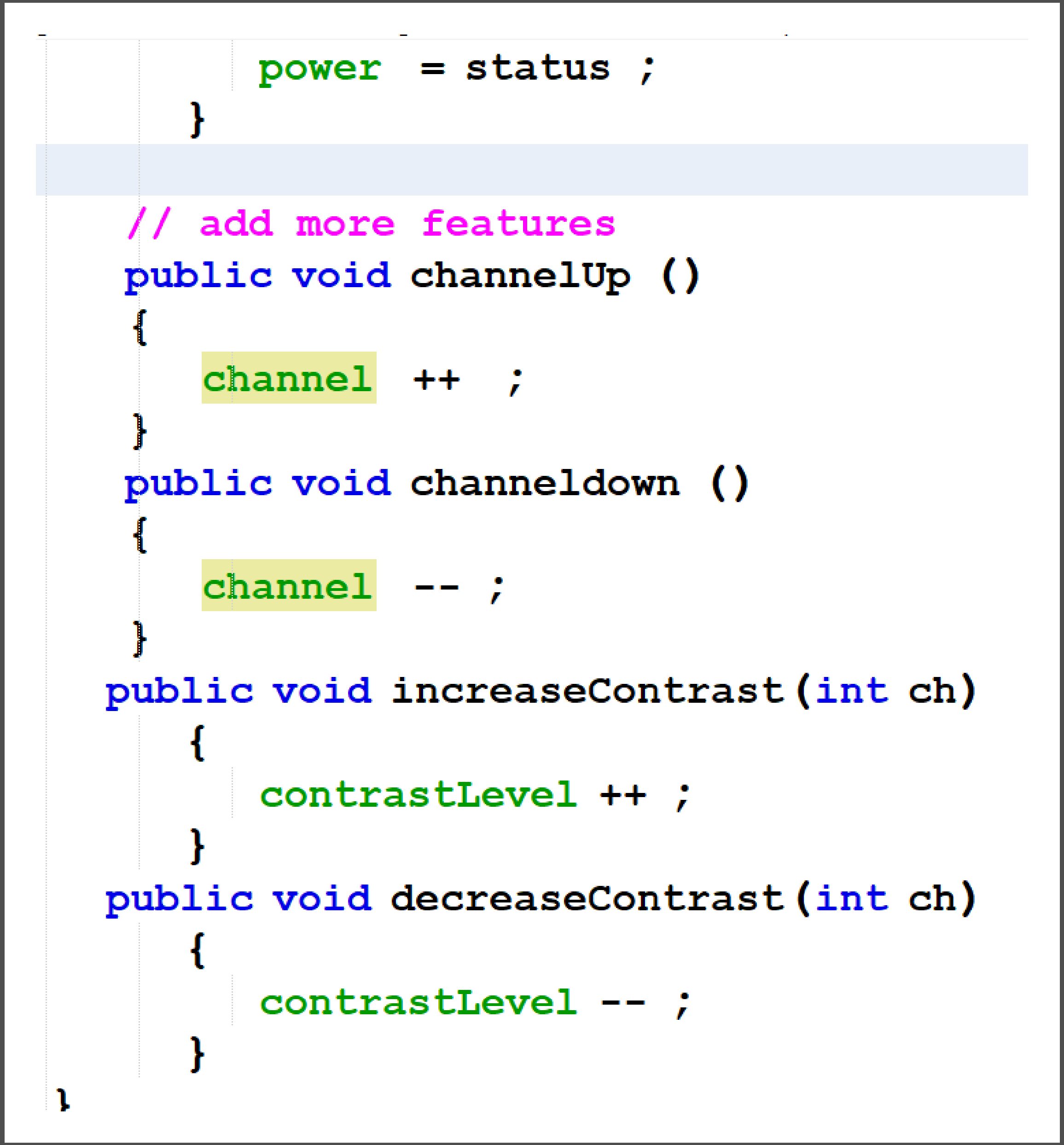
To access the interface methods, the interface must be "implemented" by another class with the implements keyword

The body of the interface method is provided by the "implement" class

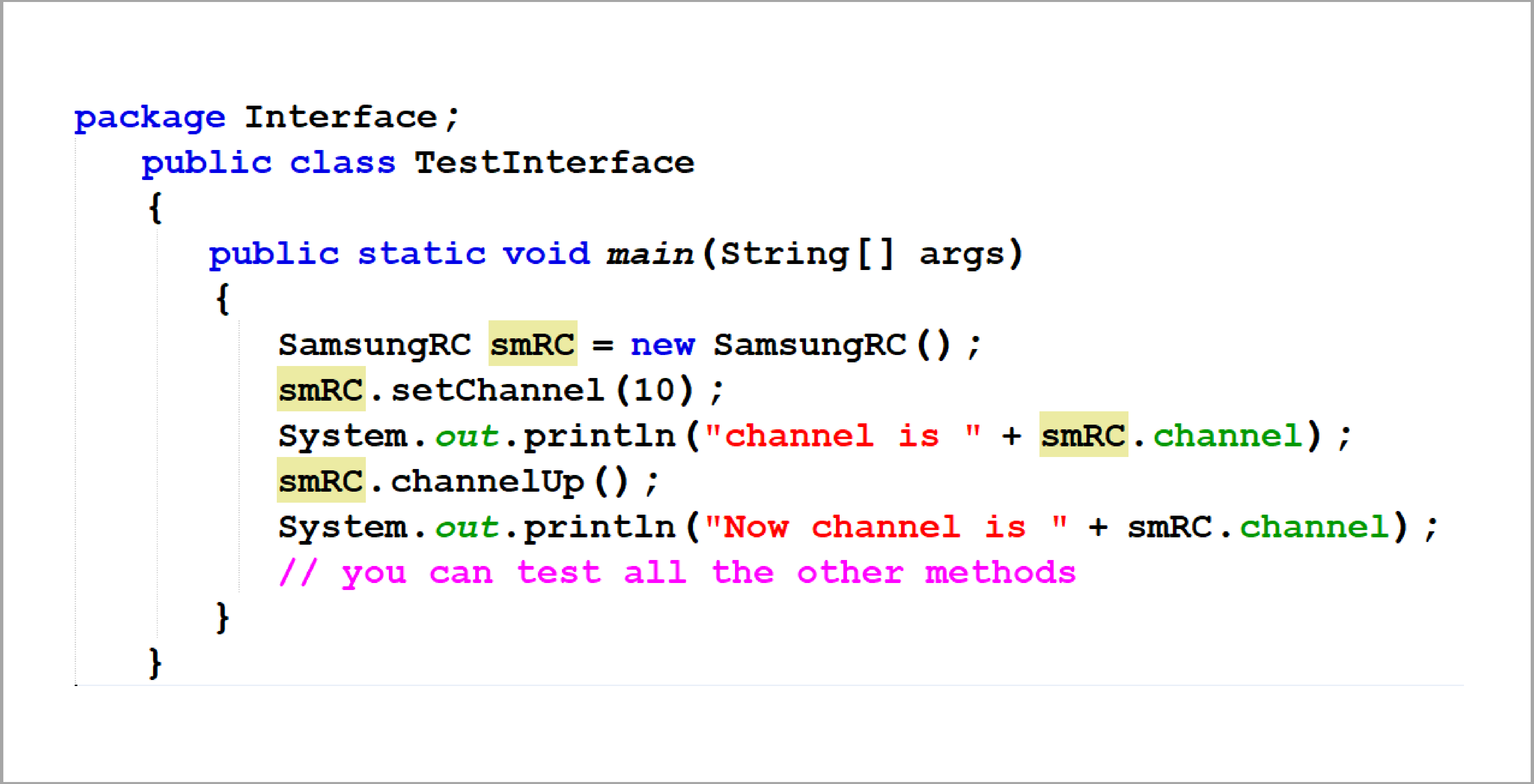
```java
package Interface;
    public interface RemoteControl
        {
            public void setVolume (int v);
            public void setChannel(int ch);
            public void setPower (boolean status);
        }
```



```
:age Interface;
ublic class SamsungRC implements RemoteCor
[
   int volumeLevel  = 1 ;
   int channel = 1 ;
   boolean power = false ;
   int contrastLevel  =  10 ;

   public void setVolume(int v)
       {
           volumeLevel = v ;
       }

   public void setChannel(int ch)
       {
           channel = ch ;
       }
    public void setPower(boolean status)
       {
           power  = status ;
       }
```



```
           power  = status ;
       }

     // add more features
     public void channelUp ()
     {
         channel  ++  ;
     }
     public void channeldown ()
     {
         channel  -- ;
     }
    public void increaseContrast(int ch)
       {
           contrastLevel ++ ;
       }
    public void decreaseContrast(int ch)
       {
           contrastLevel -- ;
       }
}
```

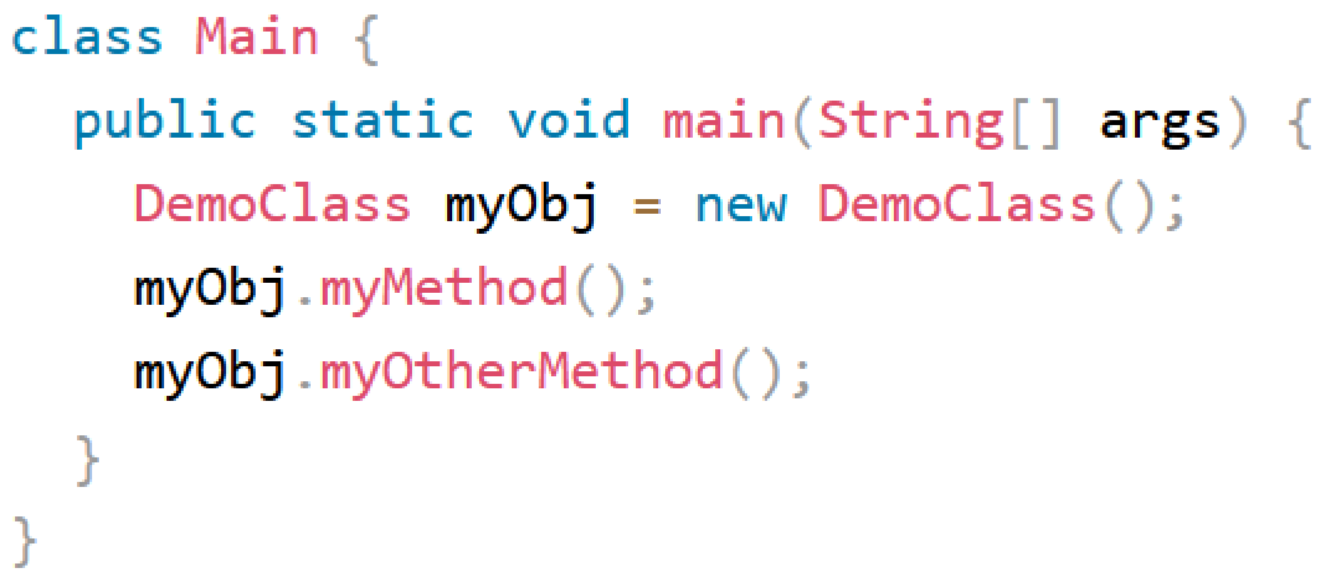


```java
package Interface;
    public class TestInterface
    {
        public static void main(String[] args)
        {
            SamsungRC smRC = new SamsungRC();
            smRC.setChannel(10);
            System.out.println("channel is " + smRC.channel);
            smRC.channelUp();
            System.out.println("Now channel is " + smRC.channel);
            // you can test all the other methods
        }
    }
```

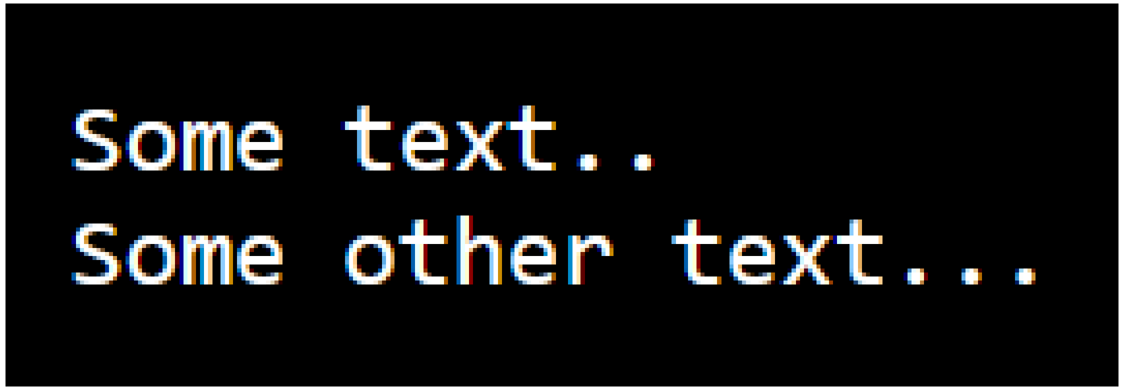
Multiple Interfaces

```java
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}
```

MULTIPLE INTERFACES



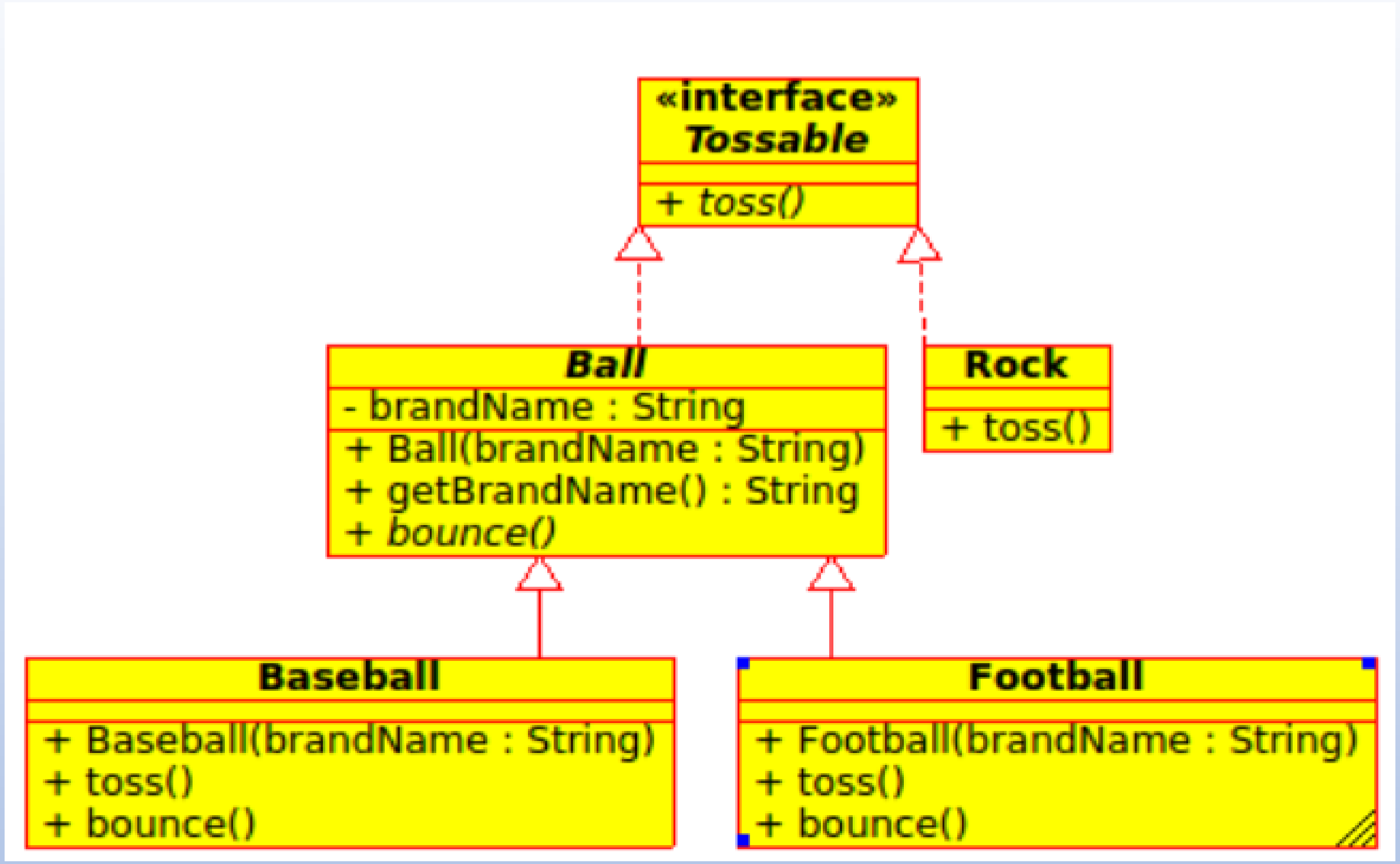
```java
class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```



```
Some text..
Some other text...
```

Assignment 3

Assignment



Implement the previous class hierarchy. You do not need to fill in the method bodies for the toss or bounce methods.

JUST TRY TO CODE 😄

- ❑ Create an interface called animal which provide two public methods eat() and travel()
- ❑ implement the previous interface by creating a Mammal class that provide complete implementation of the previous two methods in addition to add more methods such as NoOfLegs the return the number of legs of a mammal and FavFood that return the favorite food such as leaves, stems, roots and nuts

Have a good day 😍