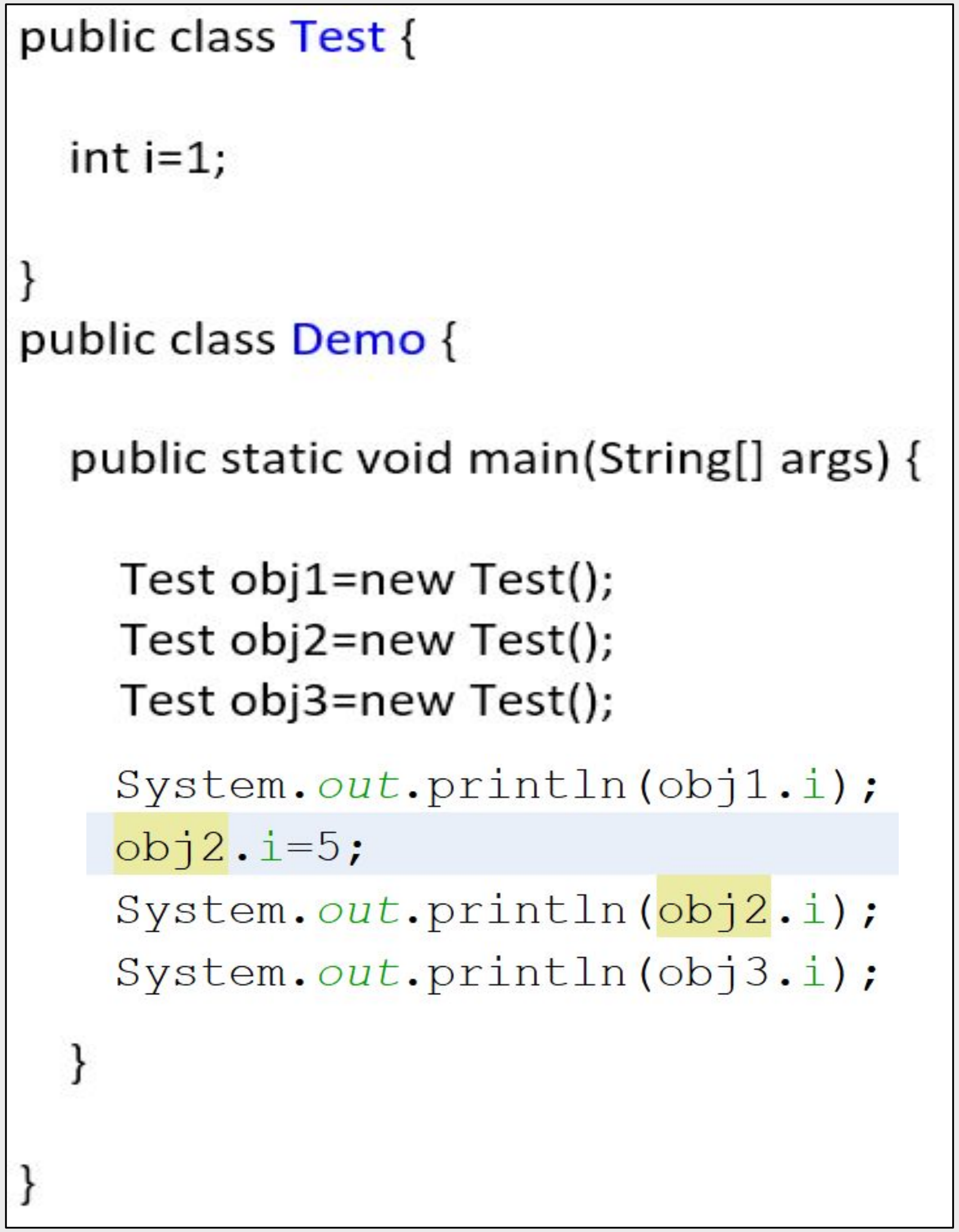
Object Oriented Programming Lab CSE 1206

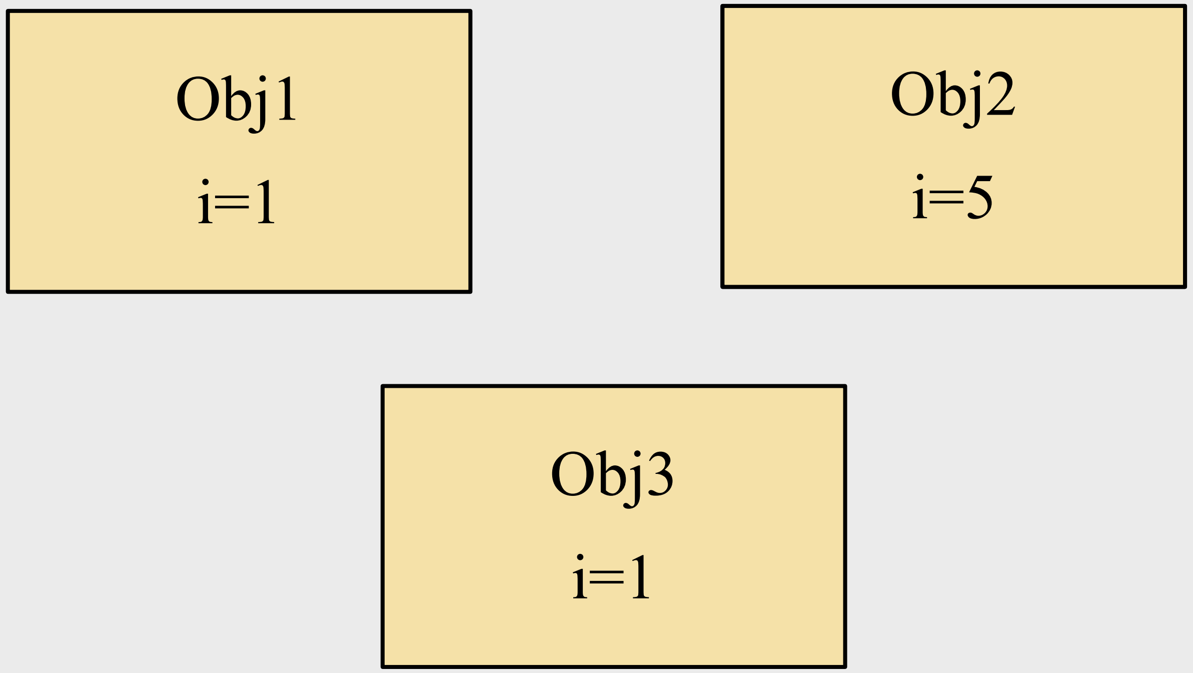
Course Teacher

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

Variables



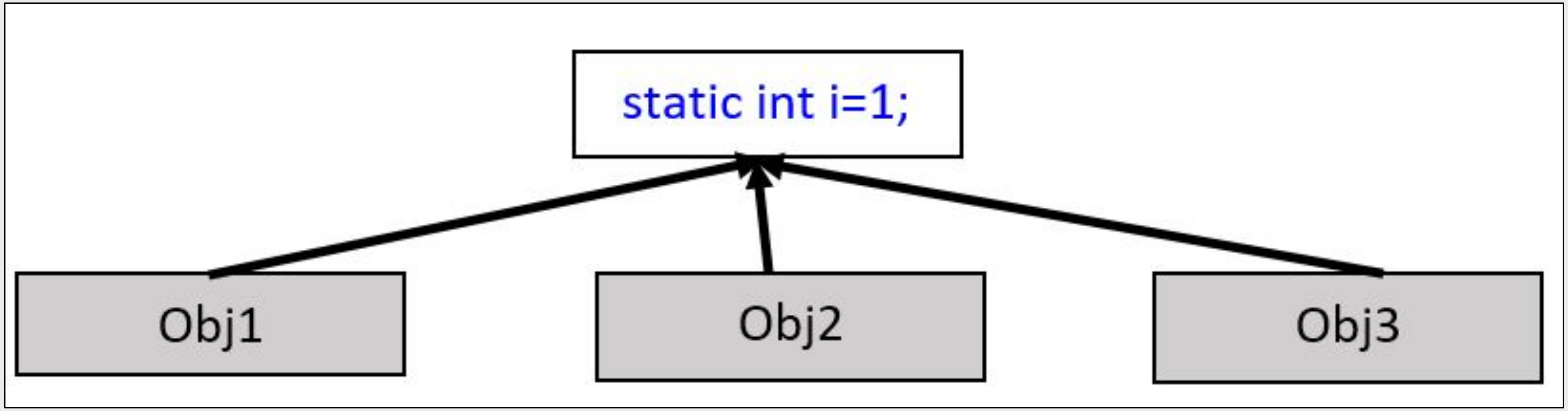
```
public class Test {

   int i=1;

}
public class Demo {

   public static void main(String[] args) {

      Test obj1=new Test();
      Test obj2=new Test();
      Test obj3=new Test();

      System.out.println(obj1.i);
      obj2.i=5;
      System.out.println(obj2.i);
      System.out.println(obj3.i);

   }

}
```



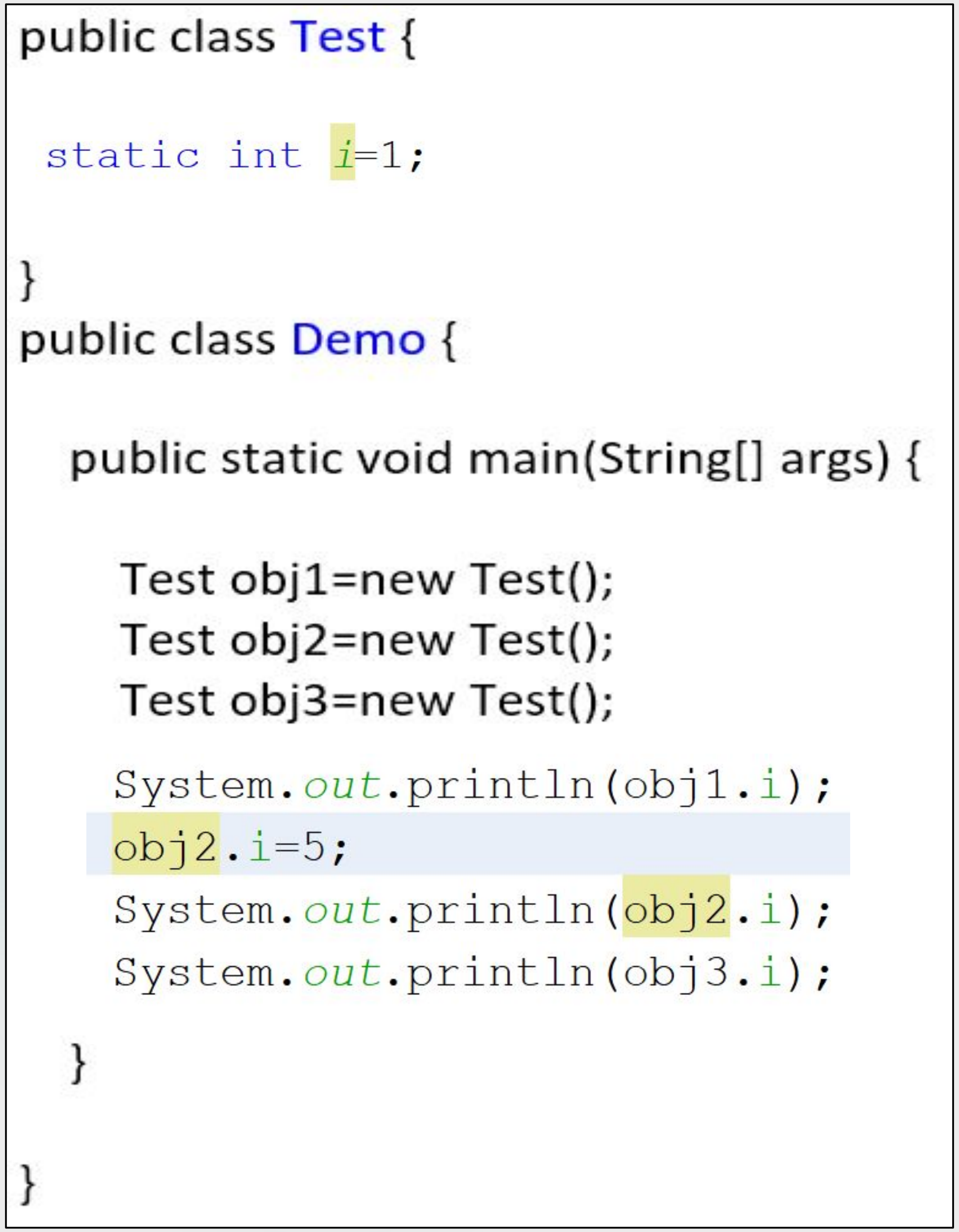
Each variable is exclusive to its own object. One object cannot access or modify another object’s variables normally.

Static Variables:

1. When a variable is declared static, it has a permanent place in the memory.
2. All objects have access to that common static variable.
3. Value of static variable can be changed. Once changed it is permanently changed for all objects.



Static Variables



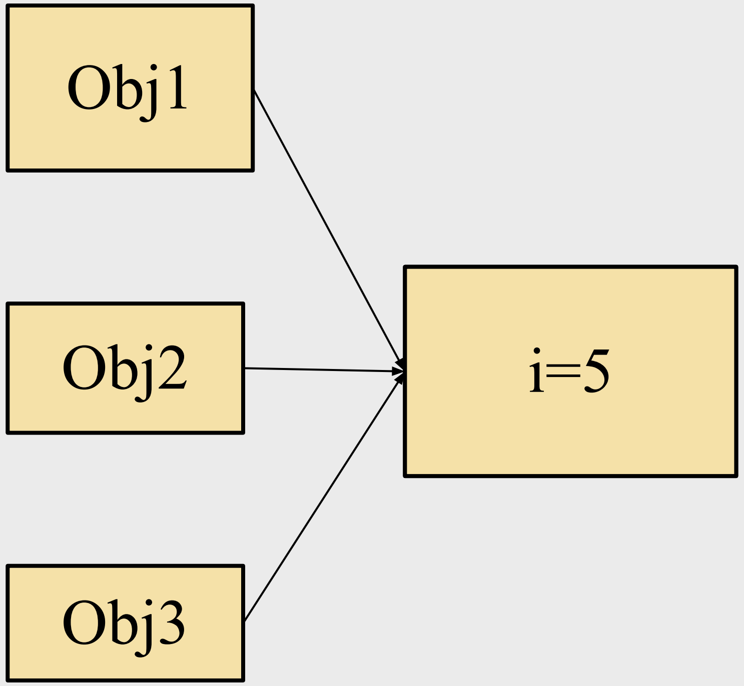
```
public class Test {

 static int i=1;

}
public class Demo {

   public static void main(String[] args) {

      Test obj1=new Test();
      Test obj2=new Test();
      Test obj3=new Test();
      System.out.println(obj1.i);
      obj2.i=5;
      System.out.println(obj2.i);
      System.out.println(obj3.i);

   }

}
```

Output:

1

5

5



Methods

```
public void printName(String name){
System.out.println(name);
}
```

Methods when created are generally stored in permanent memory

instance Methods (normal)	**static methods**
Every object has its own copy of method.	Method shared among all objects.

static methods

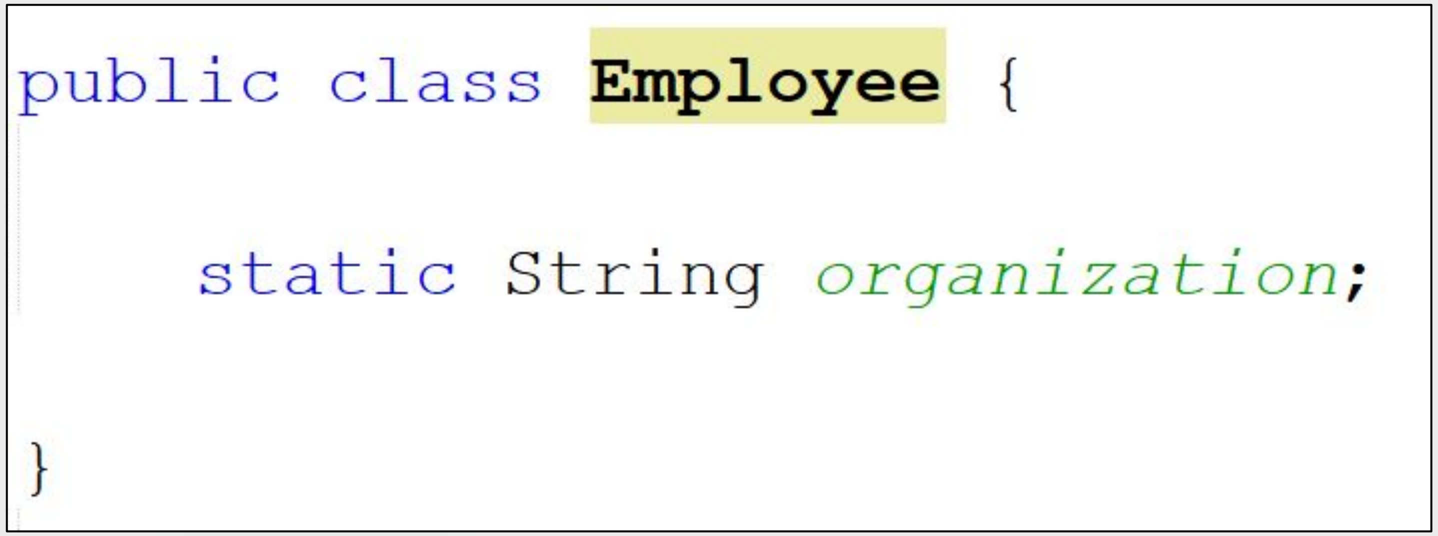
- A static method belongs to the class rather than the object of a class.
- A static method can access **static data member** and can change the value of it.
- A static method can be called without the need for creating an instance of a class.

Project Name: **TestEmployee** (main class)

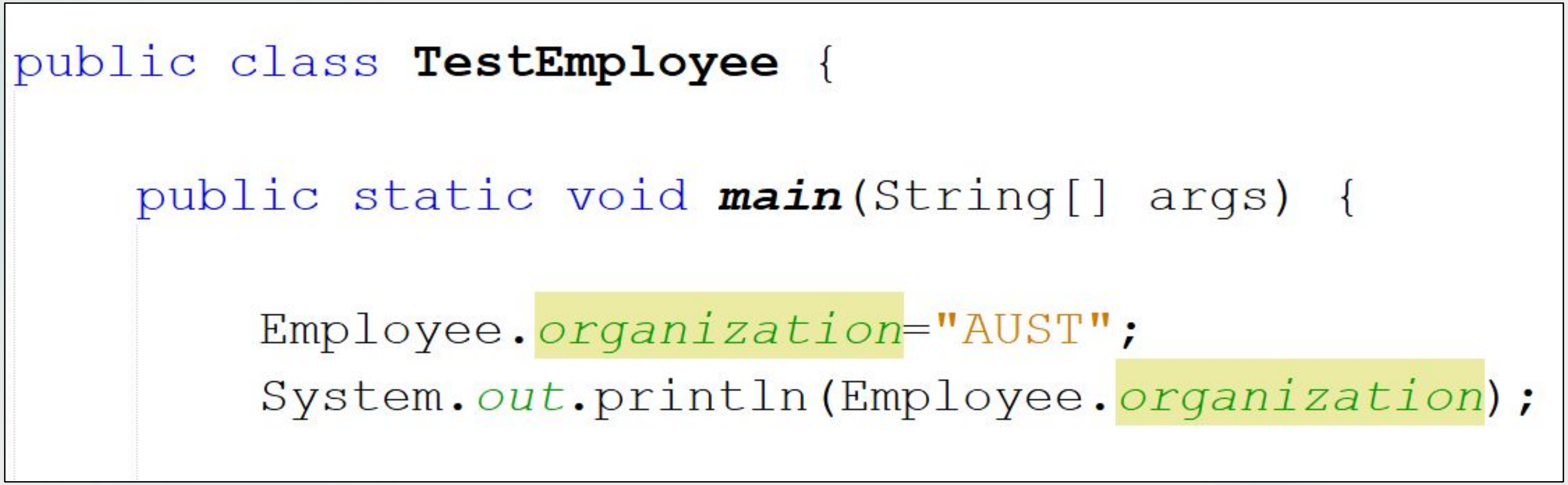
Class Name: **Employee**

Variables in Employee:

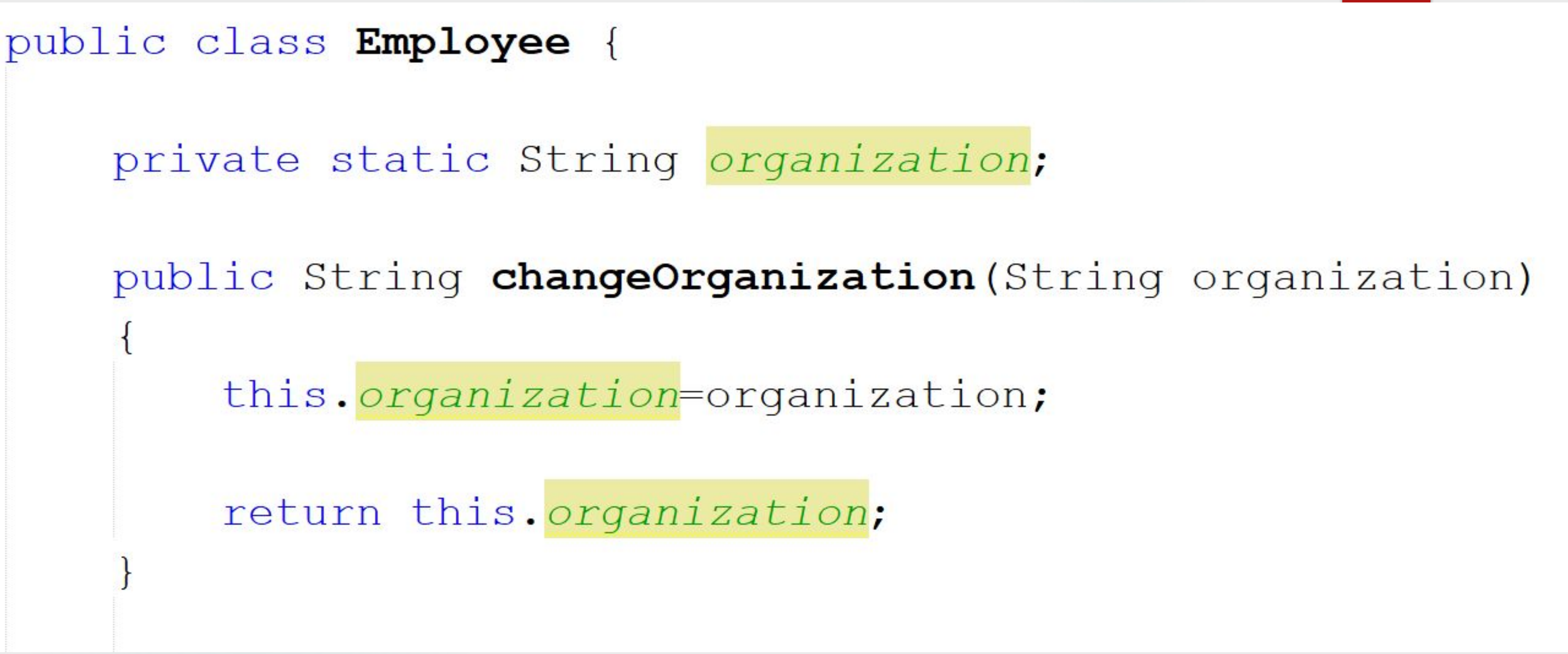
```
    static String organization;
    private int eid;
    private String name;
    private String designation;
    private int age;
```



```java
public class Employee {

    static String organization;

}
```

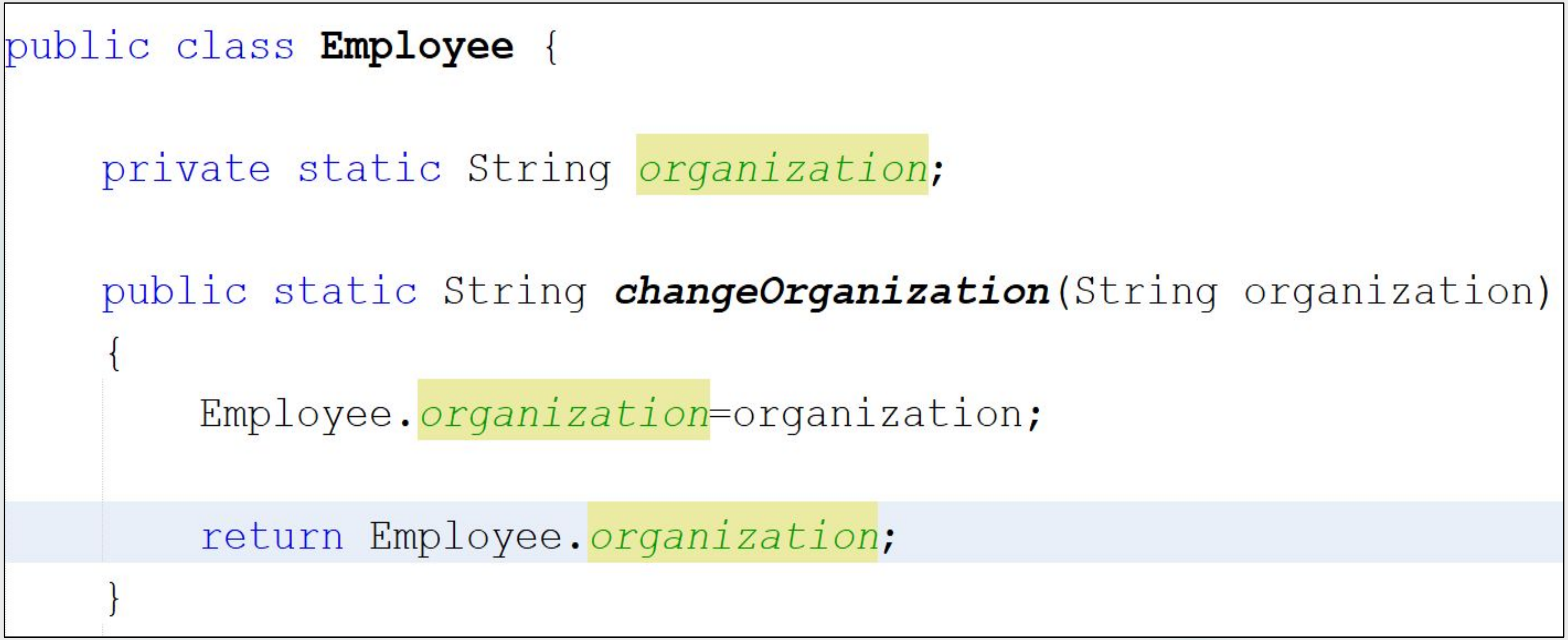


```java
public class TestEmployee {

    public static void main(String[] args) {

        Employee.organization="AUST";
        System.out.println(Employee.organization);
```



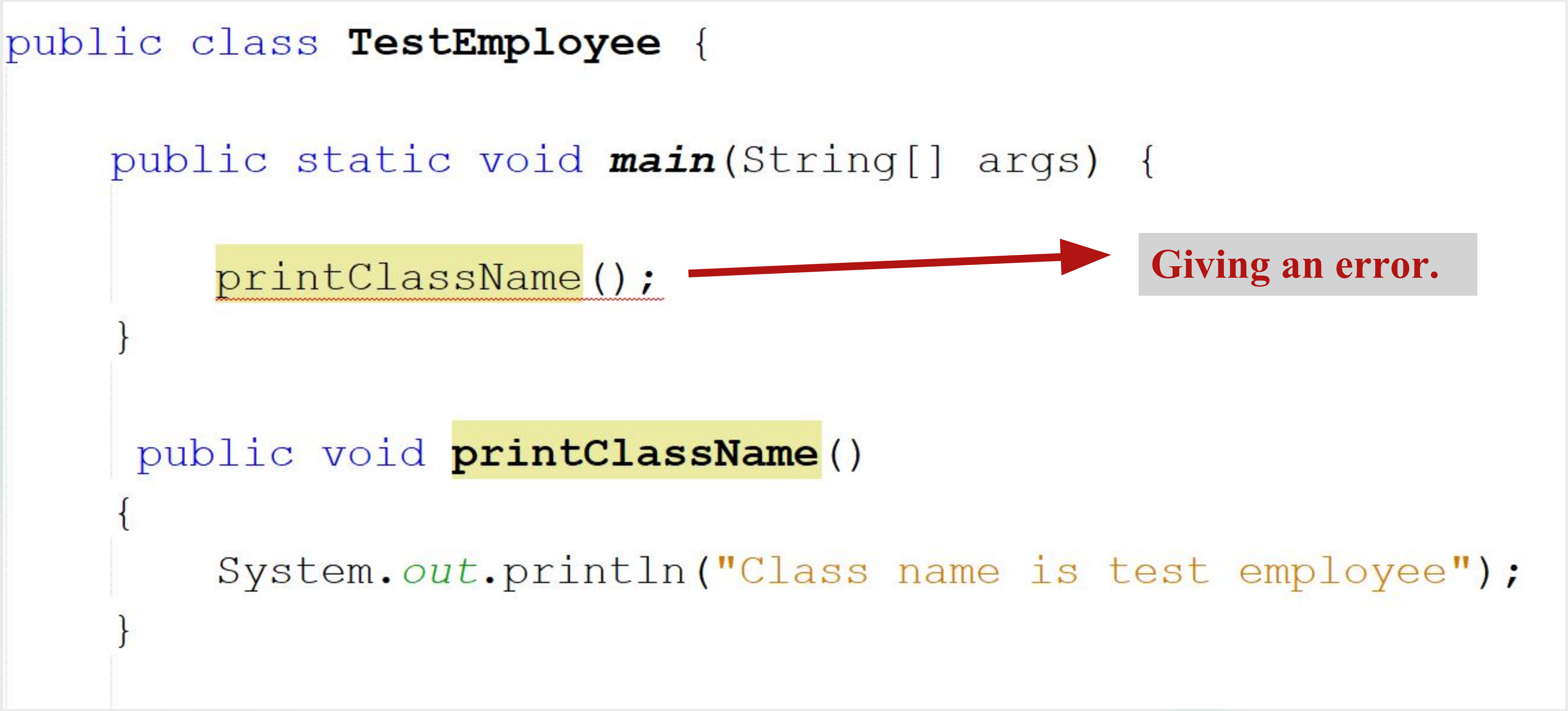
```java
public class Employee {

    private static String organization;

    public String changeOrganization(String organization)
    {
        this.organization=organization;

        return this.organization;
    }
```

Usually static methods are used to change static variables



```
public class Employee {

    private static String organization;

    public static String changeOrganization(String organization)
    {
        Employee.organization=organization;

        return Employee.organization;
    }
```

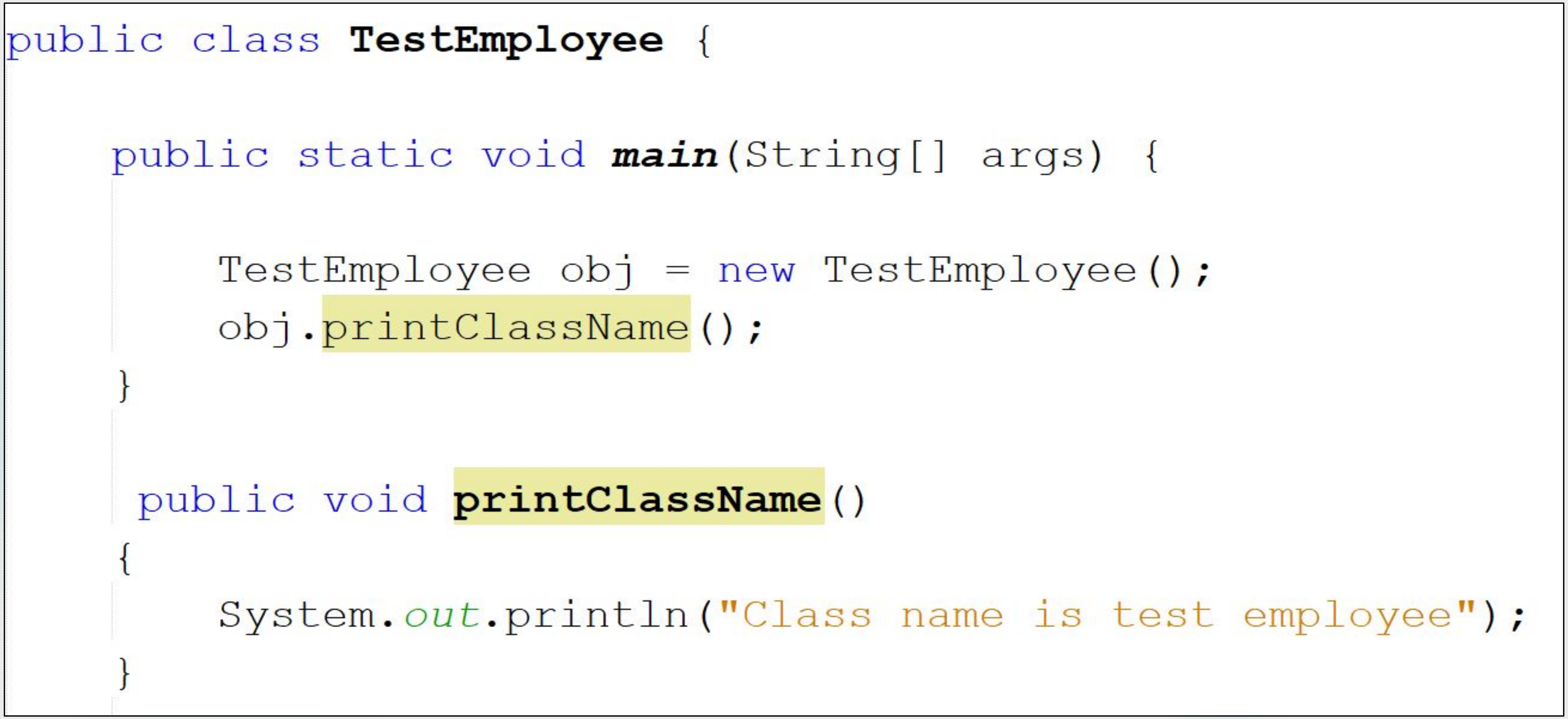
Calling methods in Main method



```
public class TestEmployee {

    public static void main(String[] args) {

        printClassName();
    }

     public void printClassName()
    {
        System.out.println("Class name is test employee");
    }
```

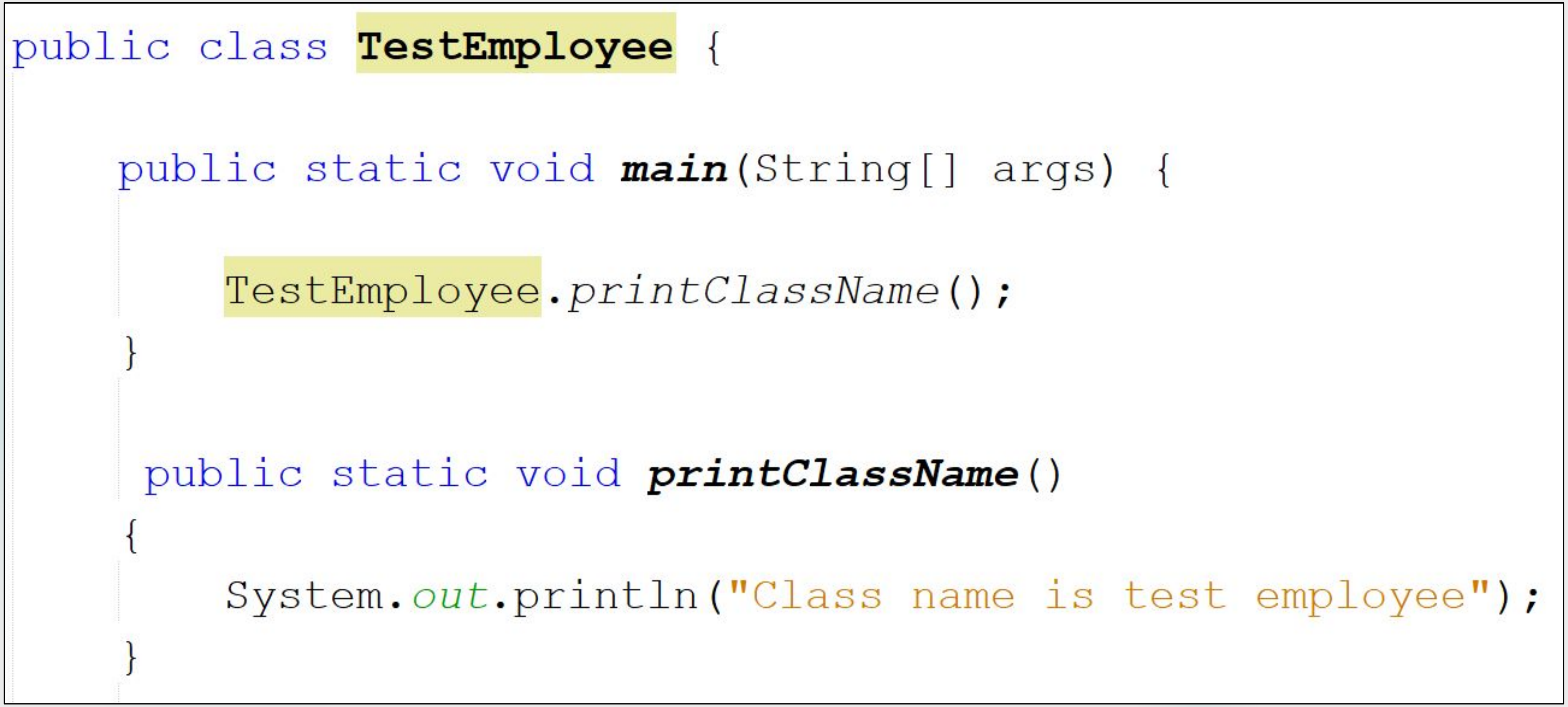
Giving an an error.

You will need to create object



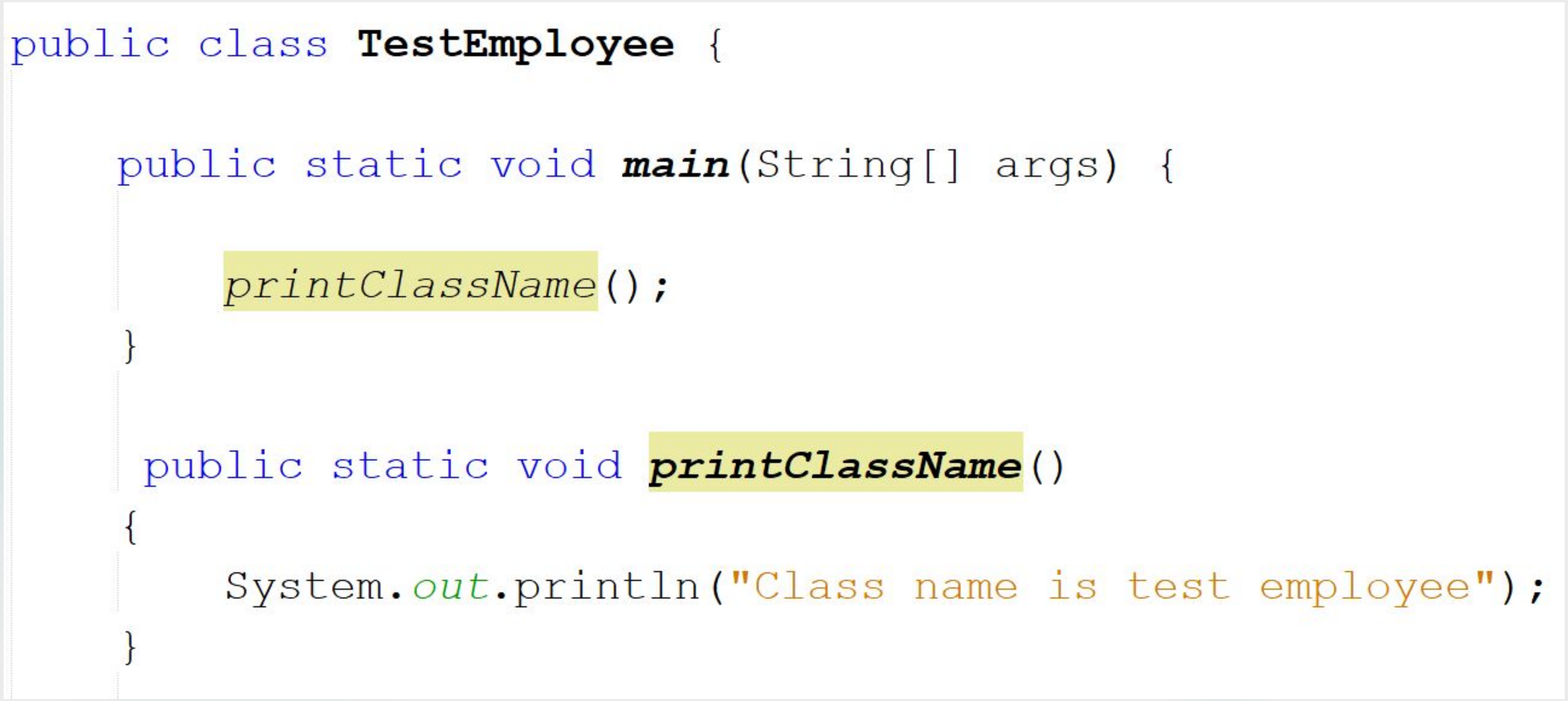
```
public class TestEmployee {

    public static void main(String[] args) {

        TestEmployee obj = new TestEmployee();
        obj.printClassName();
    }

     public void printClassName()
    {
        System.out.println("Class name is test employee");
    }
```

Or you can make the method static



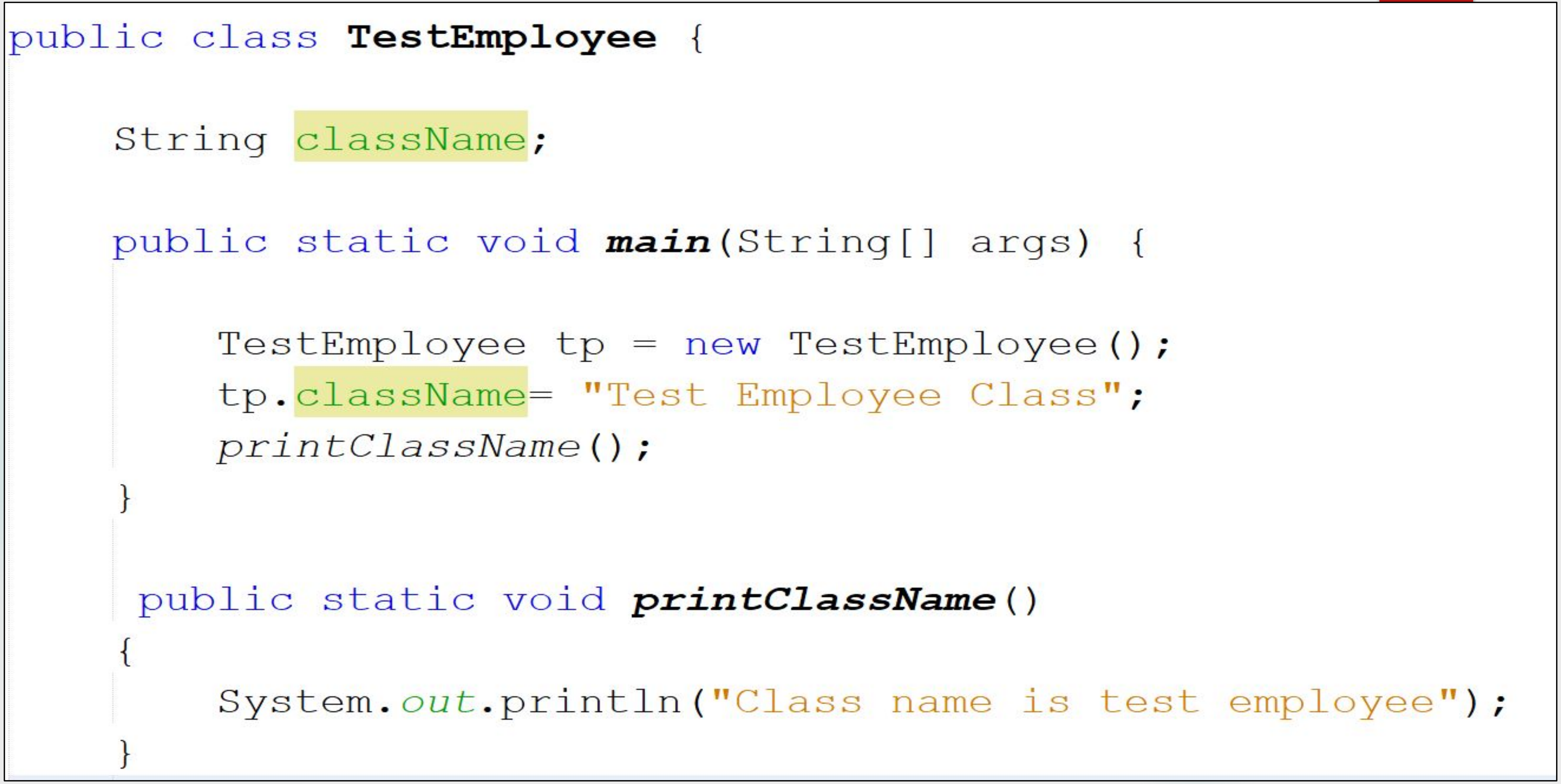
```
public class TestEmployee {

    public static void main(String[] args) {

        TestEmployee.printClassName();
    }


     public static void printClassName()
    {
        System.out.println("Class name is test employee");
    }
```

Or call it directly without the Class name



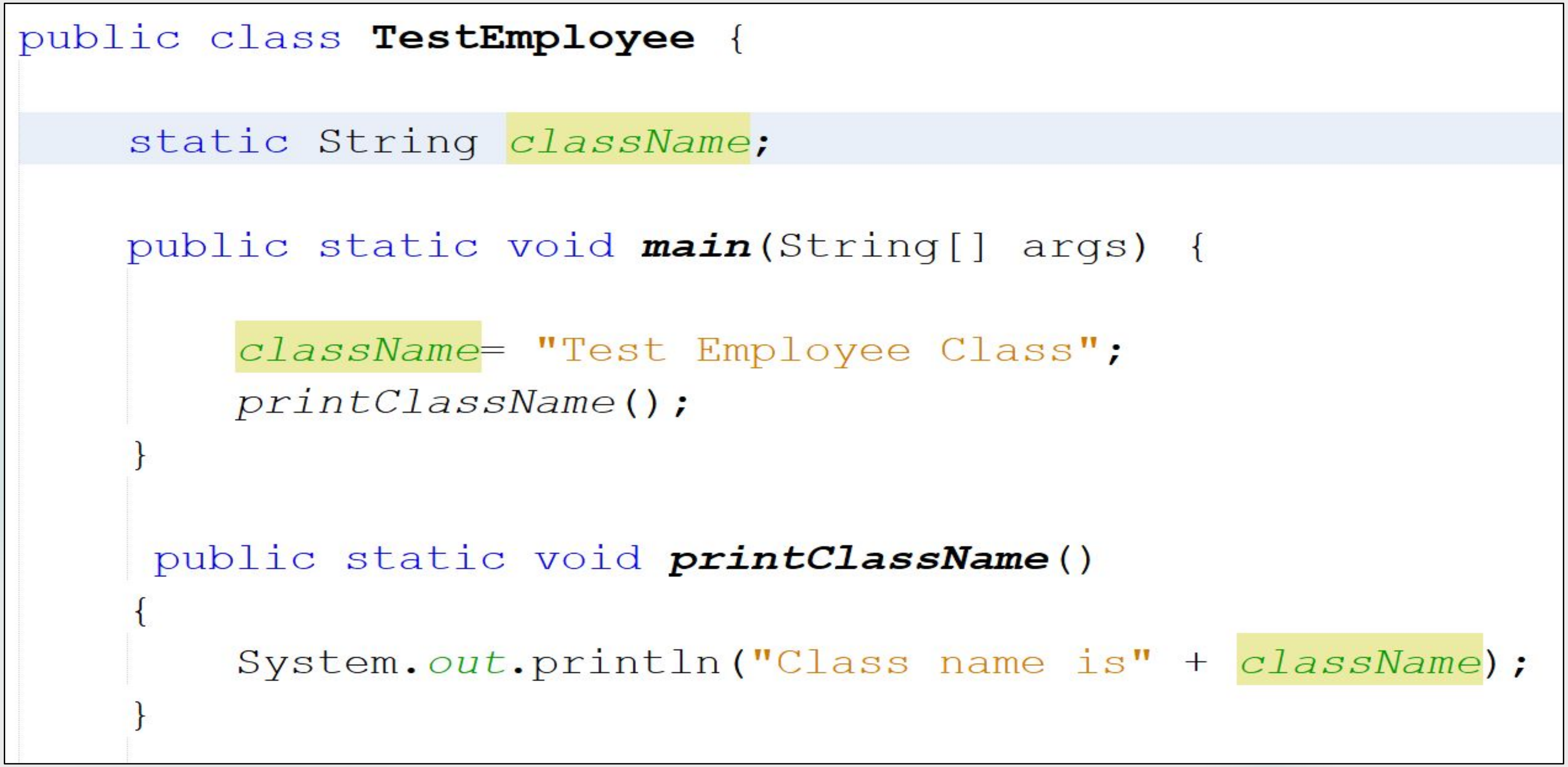
```
public class TestEmployee {

    public static void main(String[] args) {

        printClassName();
    }

     public static void printClassName()
    {
        System.out.println("Class name is test employee");
    }
```

Similar case for variables



```
public class TestEmployee {

    String className;

    public static void main(String[] args) {

        TestEmployee tp = new TestEmployee();
        tp.className= "Test Employee Class";
        printClassName();
    }

     public static void printClassName()
    {
        System.out.println("Class name is test employee");
    }
```

Or declare the variable static



```
public class TestEmployee {

    static String className;

    public static void main(String[] args) {

        className= "Test Employee Class";
        printClassName();
    }

     public static void printClassName()
    {
        System.out.println("Class name is" + className);
    }
```

Introduction to Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

Inheritance

Class that is inherited is called SUPERCLASS

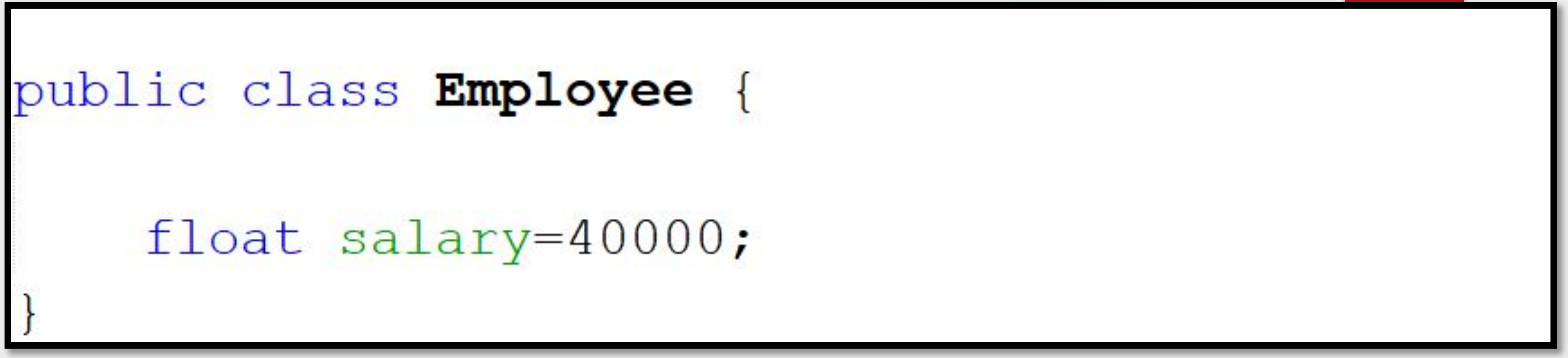
Class that does the inheriting is called SUBCLASS

Syntax: using extends keyword

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

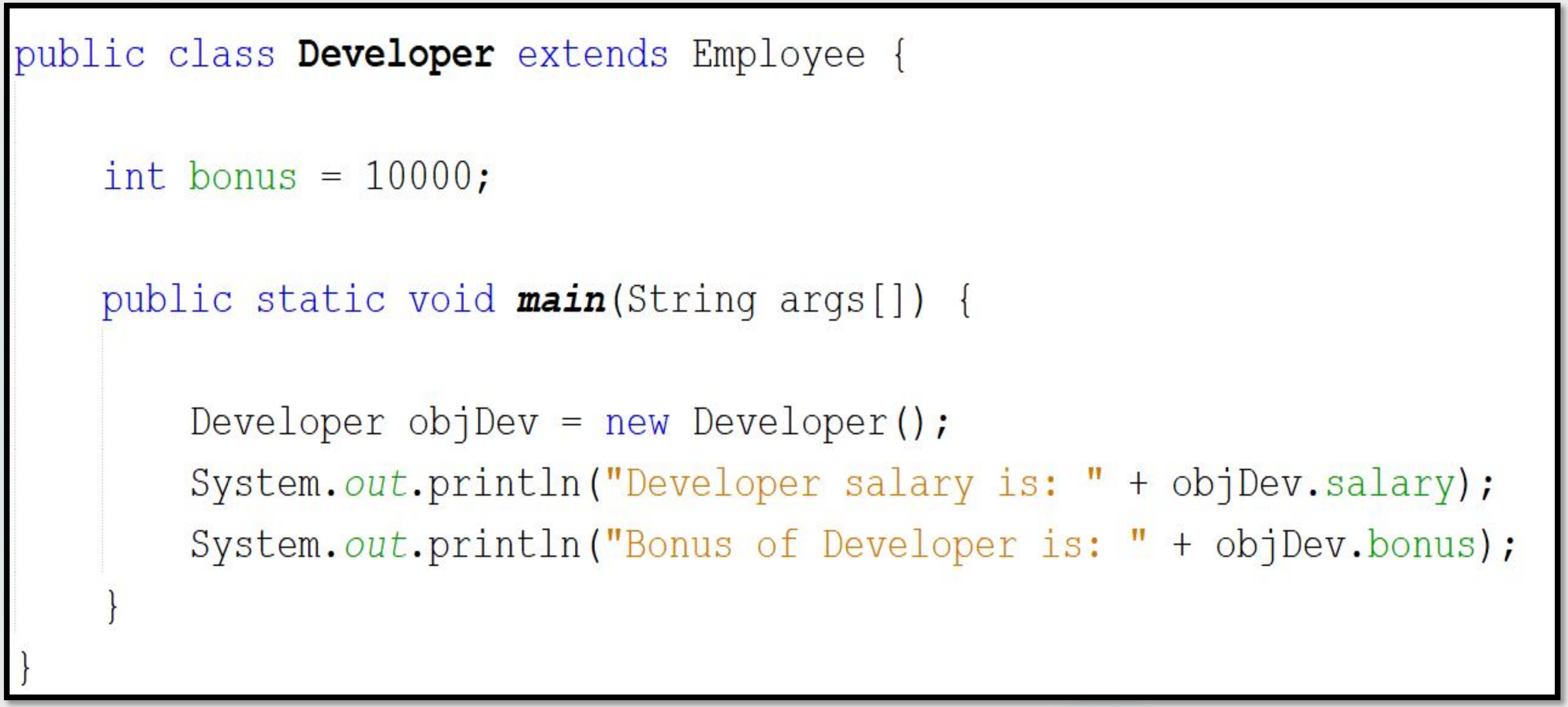
Single Level Inheritance

Super Class



```java
public class Employee {

    float salary=40000;
}
```

Sub Class



```java
public class Developer extends Employee {

    int bonus = 10000;

    public static void main(String args[]) {

        Developer objDev = new Developer();
        System.out.println("Developer salary is: " + objDev.salary);
        System.out.println("Bonus of Developer is: " + objDev.bonus);
    }
}
```

Create a project with whatever name you want to. It will be the main class. Then create the following inheritance.

Super Class Name: Shape

Variables: color (String), filled (Boolean)

Two constructors:

- a no-argument constructor that initializes color =“white” and filled = true - parameterized constructor.

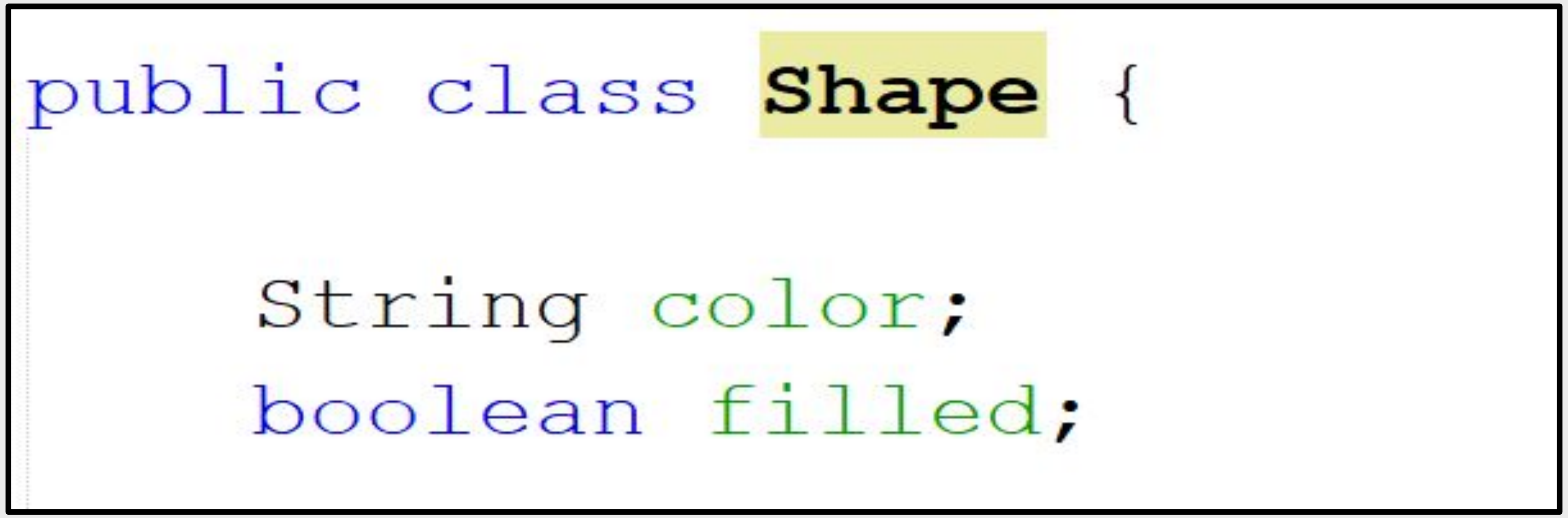
Sub Class Name: Circle

Variables: radius (double)

Three constructors:

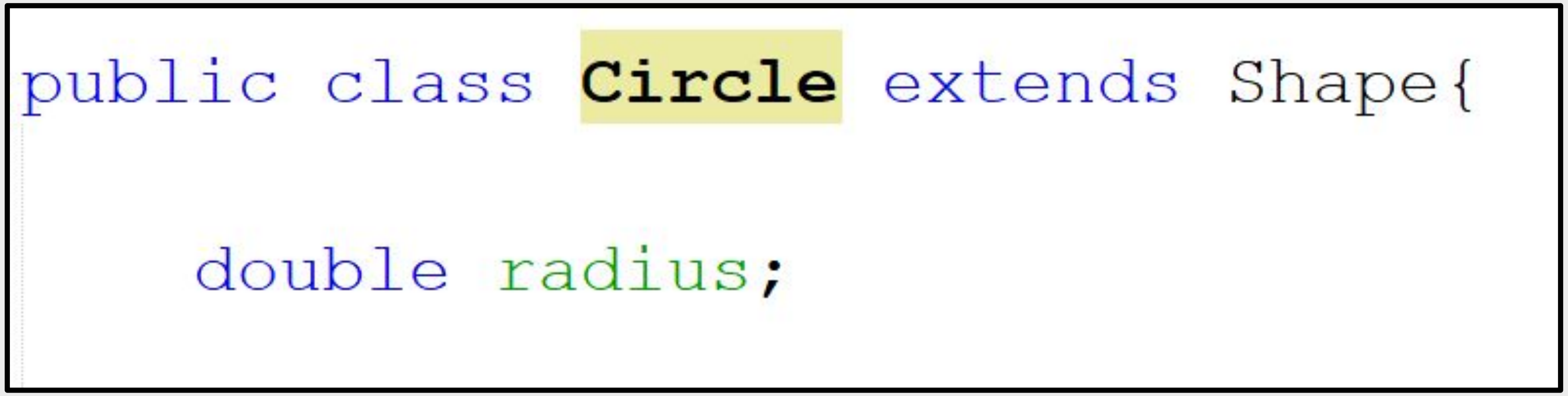
- a no-argument constructor that initializes the radius = 1.0
- A parameterized constructor with only radius as parameter.
- Another parameterized constructor with radius, color and filled

Super Class



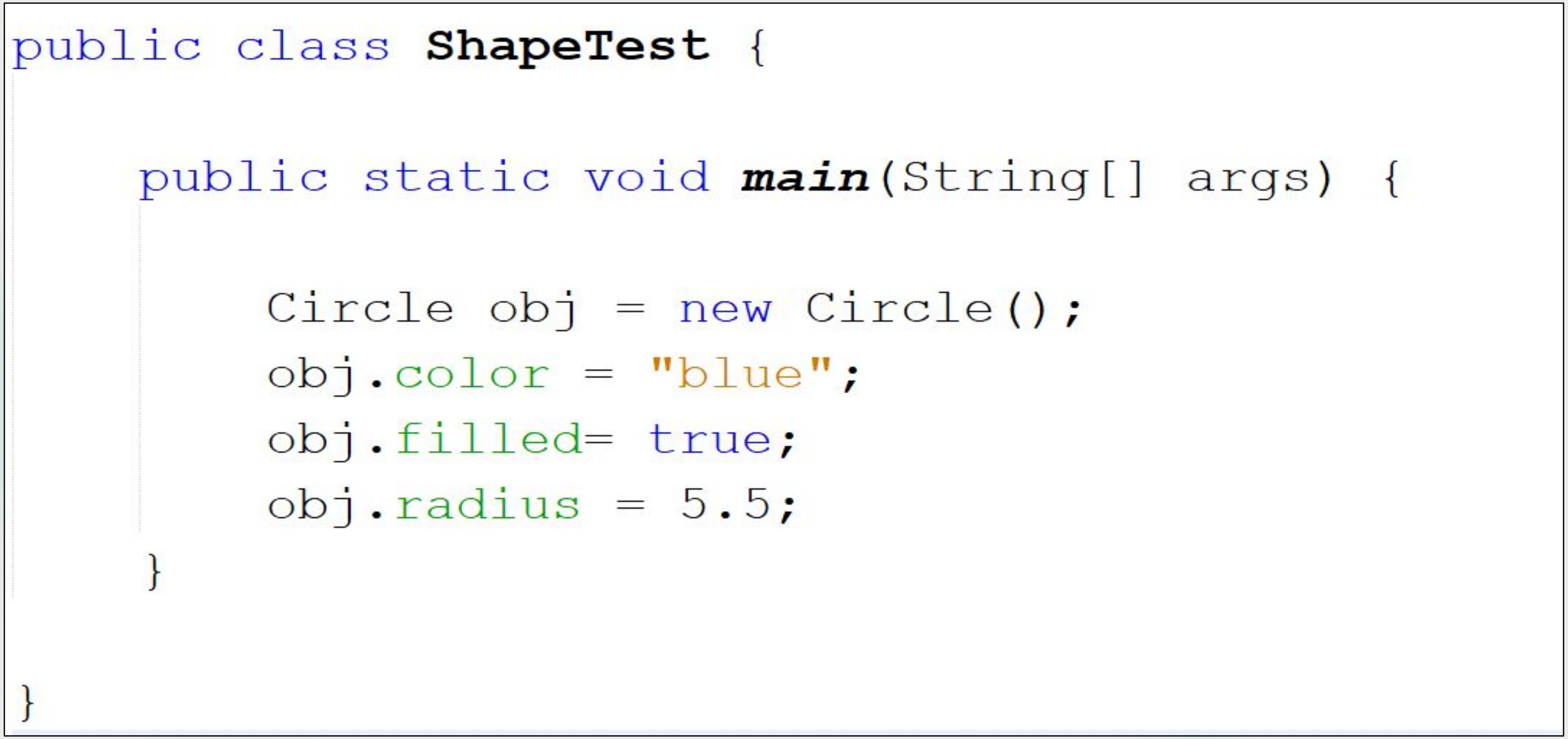
```
public class Shape {

    String color;
    boolean filled;
```

Sub Class



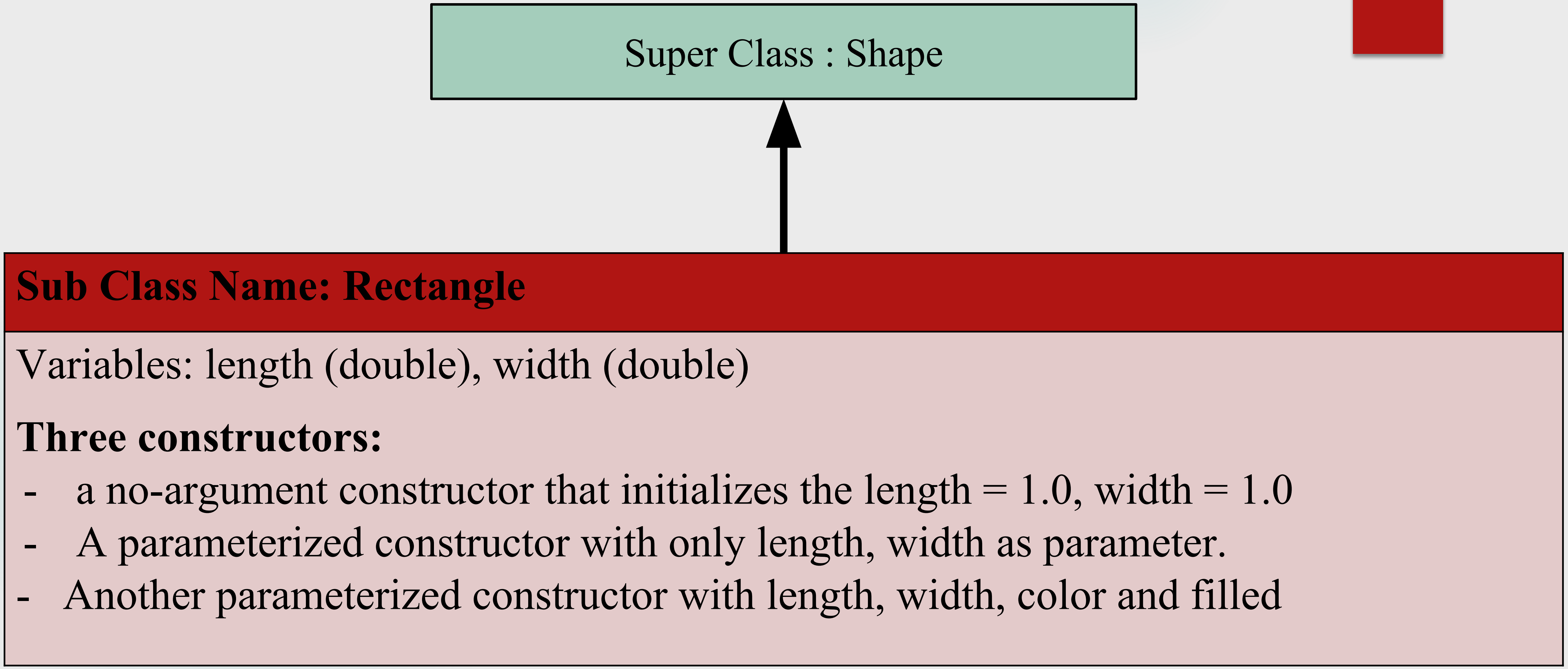
```
public class Circle extends Shape{

    double radius;
```

Testing the Inheritance in Main Class



```
public class ShapeTest {

    public static void main(String[] args) {

        Circle obj = new Circle();
        obj.color = "blue";
        obj.filled= true;
        obj.radius = 5.5;
    }

}
```

Hierarchical Inheritance



Super Class : Shape

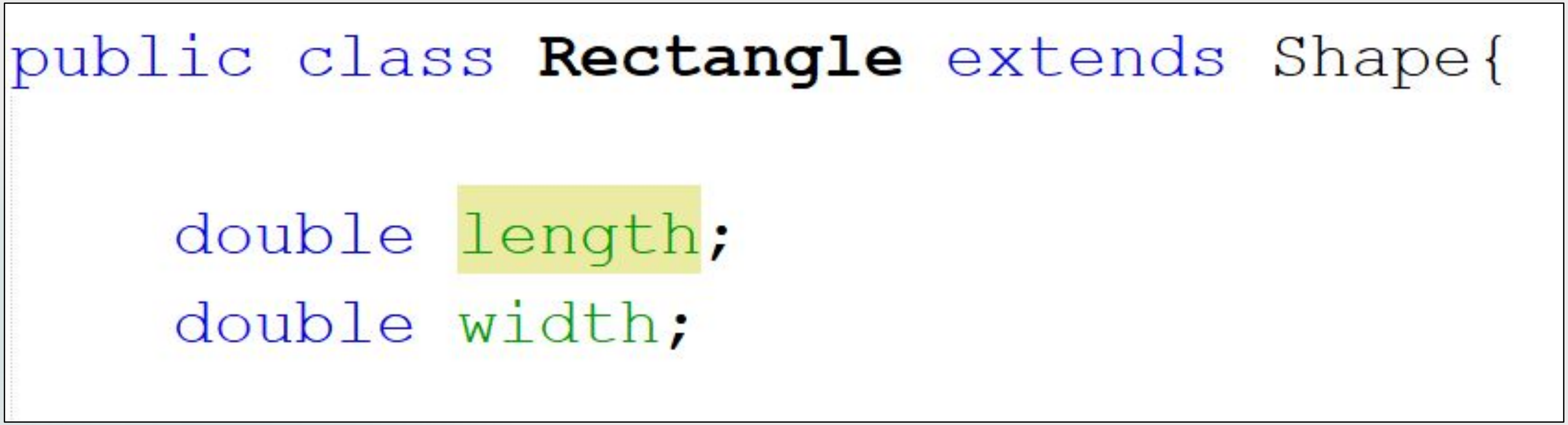
Sub Class Name: Rectangle

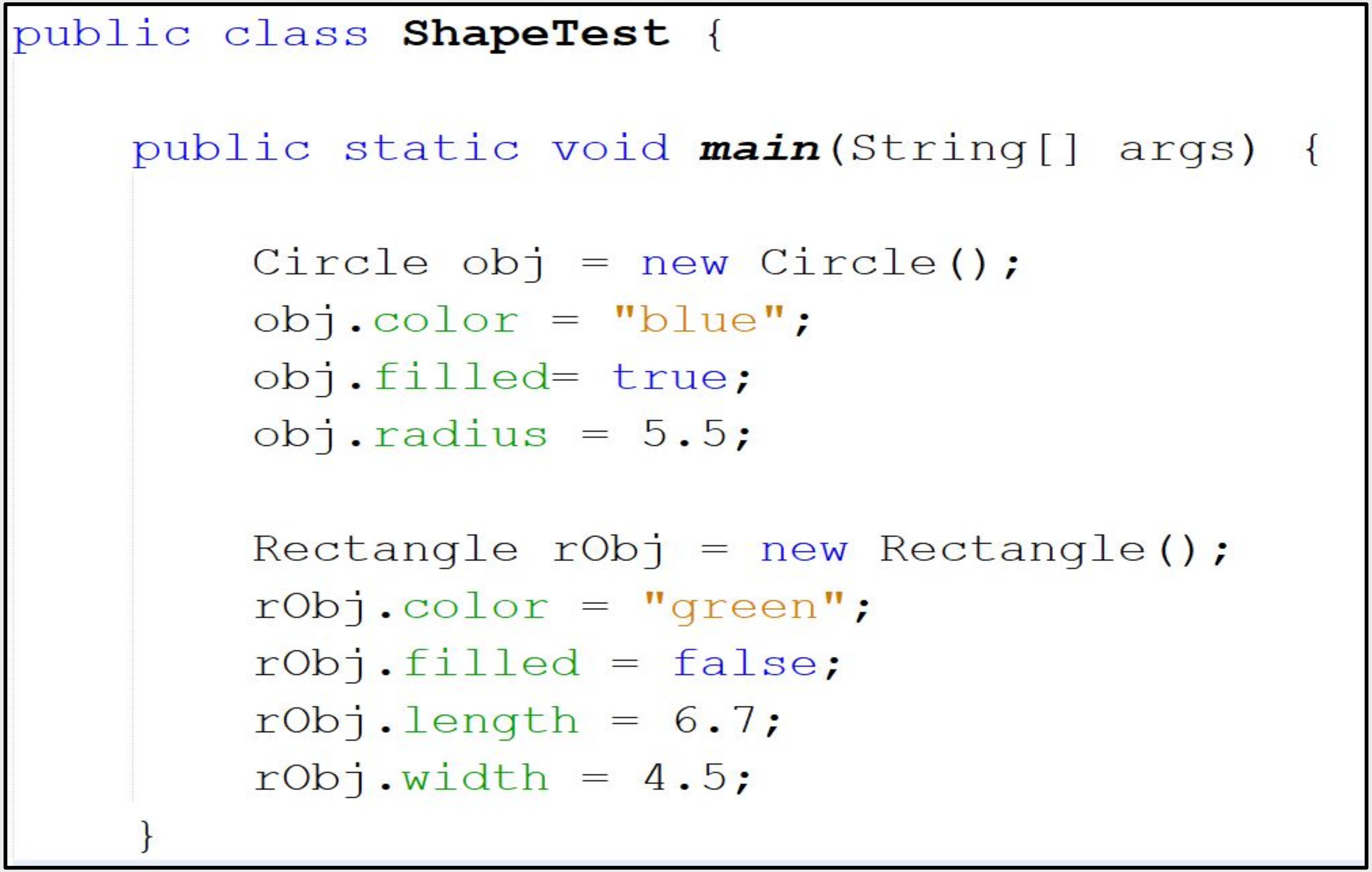
Variables: length (double), width (double)

Three constructors:

- a no-argument constructor that initializes the length = 1.0, width = 1.0
- A parameterized constructor with only length, width as parameter.
- Another parameterized constructor with length, width, color and filled

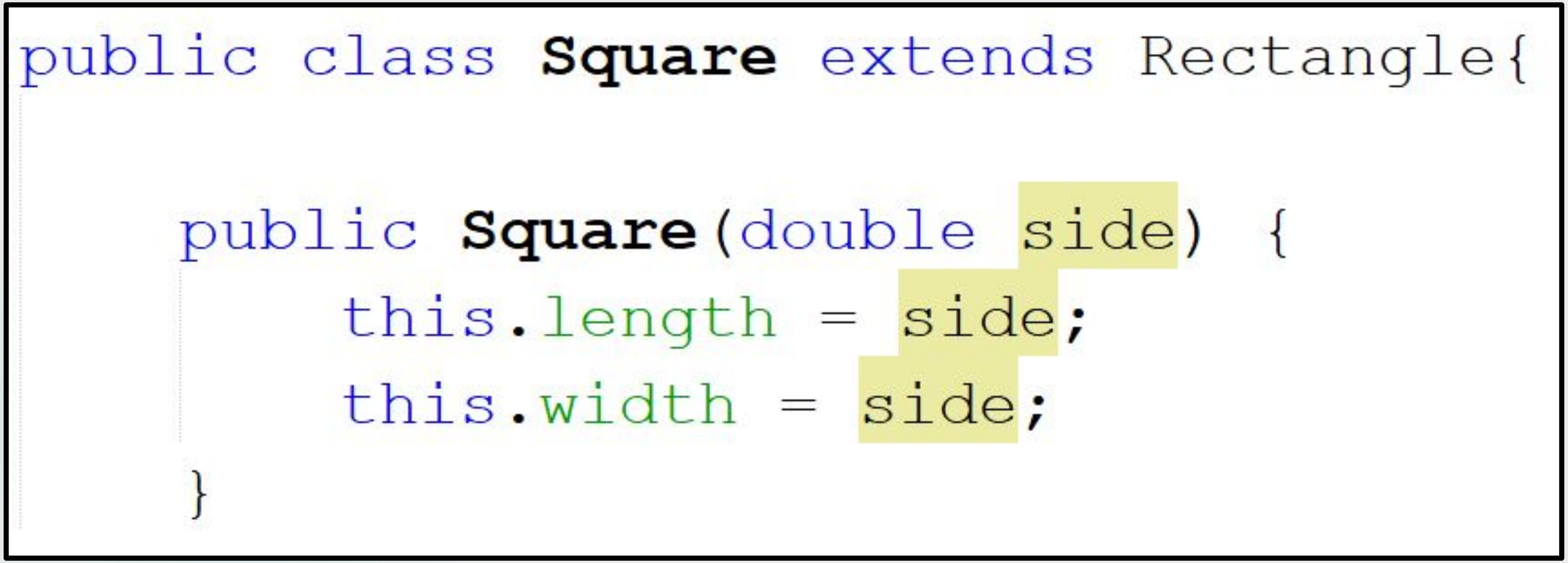
Sub Class 2 of Super Class Shape


```
public class Rectangle extends Shape{

    double length;
    double width;
```

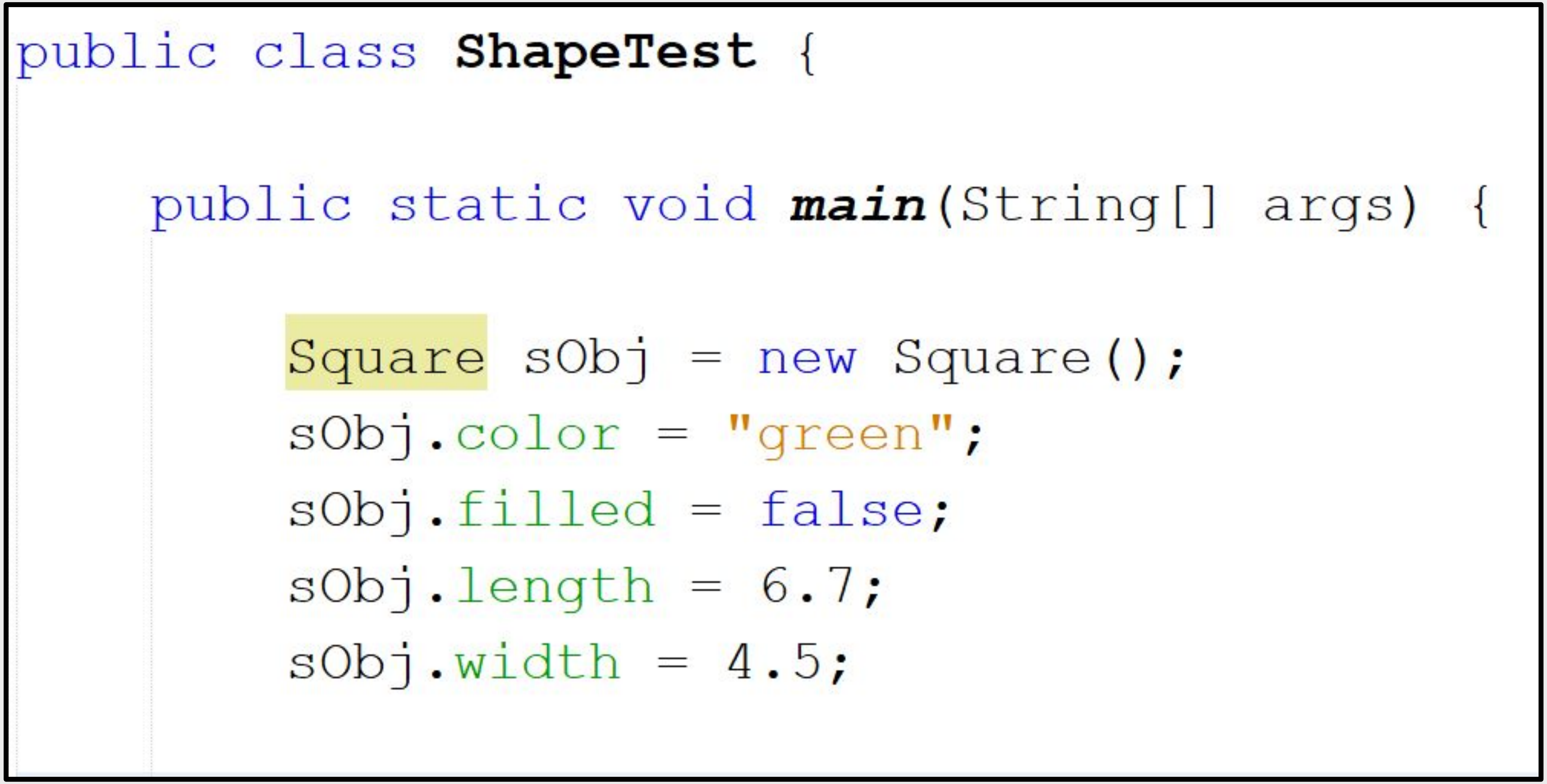


```java
public class ShapeTest {

    public static void main(String[] args) {

        Circle obj = new Circle();
        obj.color = "blue";
        obj.filled= true;
        obj.radius = 5.5;

        Rectangle rObj = new Rectangle();
        rObj.color = "green";
        rObj.filled = false;
        rObj.length = 6.7;
        rObj.width = 4.5;
    }
```

Multilevel Inheritance



```java
public class Square extends Rectangle{

    public Square(double side) {
        this.length = side;
        this.width = side;
    }
```



```
public class ShapeTest {

    public static void main(String[] args) {

        Square sObj = new Square();
        sObj.color = "green";
        sObj.filled = false;
        sObj.length = 6.7;
        sObj.width = 4.5;

```