CS 162FZ: Introduction to Computer Science II

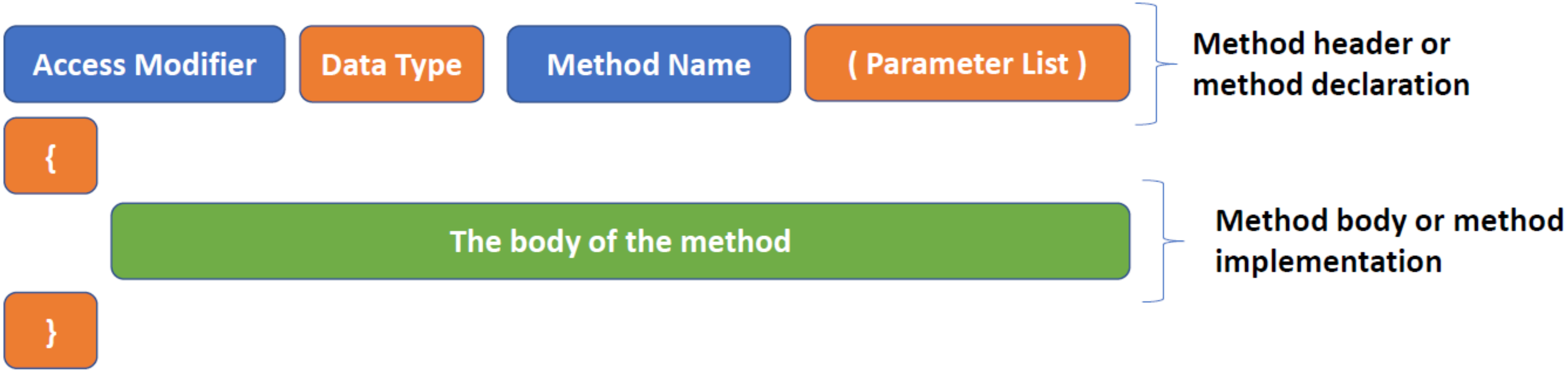
Lecture 03

Methods II

Dr. Chun-Yang Zhang

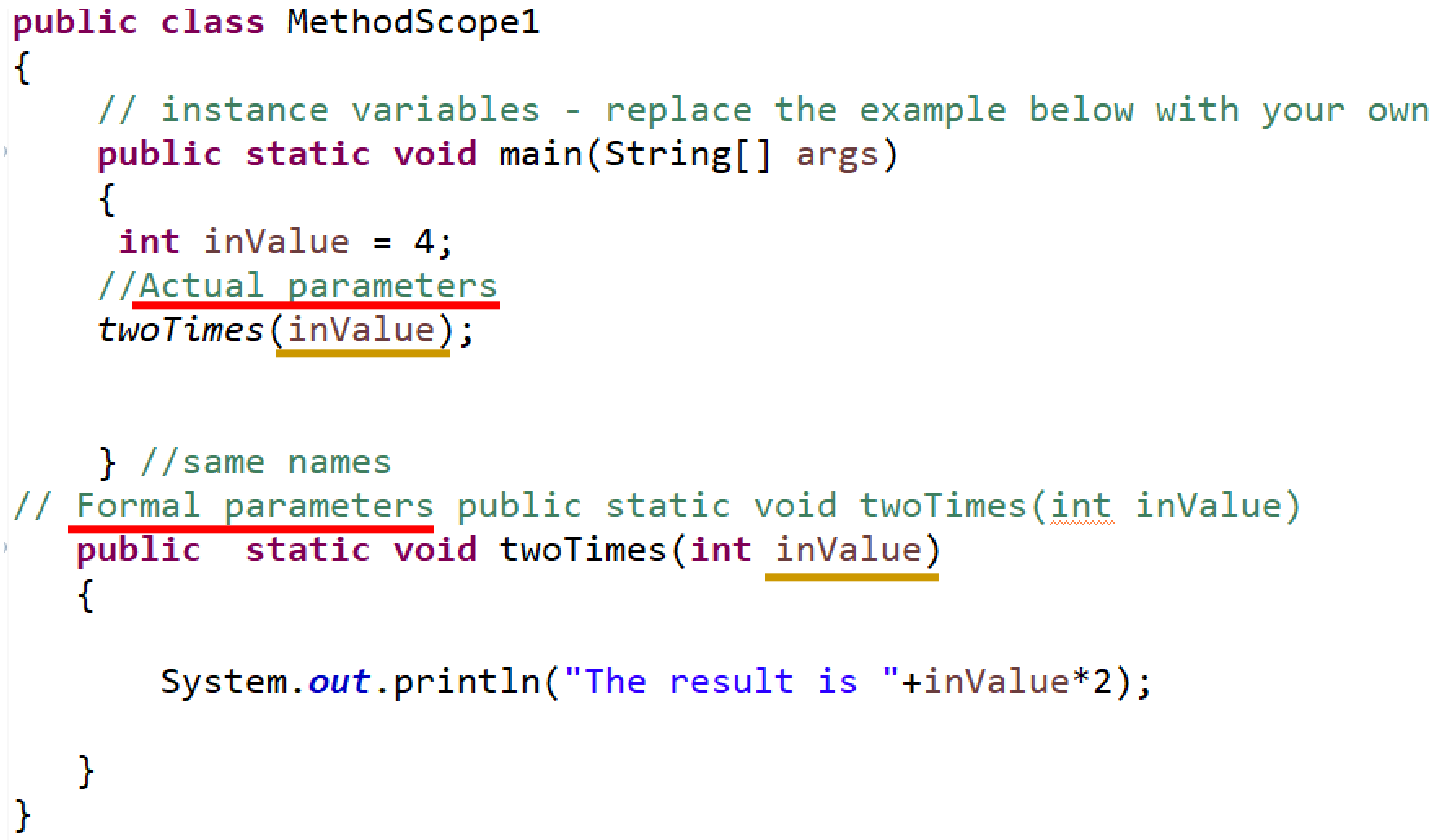
Quick Recap

- A method definition consists of an access modifier (optional), data type of returning value, a method name, a list of parameters and a body.



Actual and Formal Parameters

You can use a same name for the parameters when you call a method (actual parameters) to the manes used in the method signature (formal parameters).



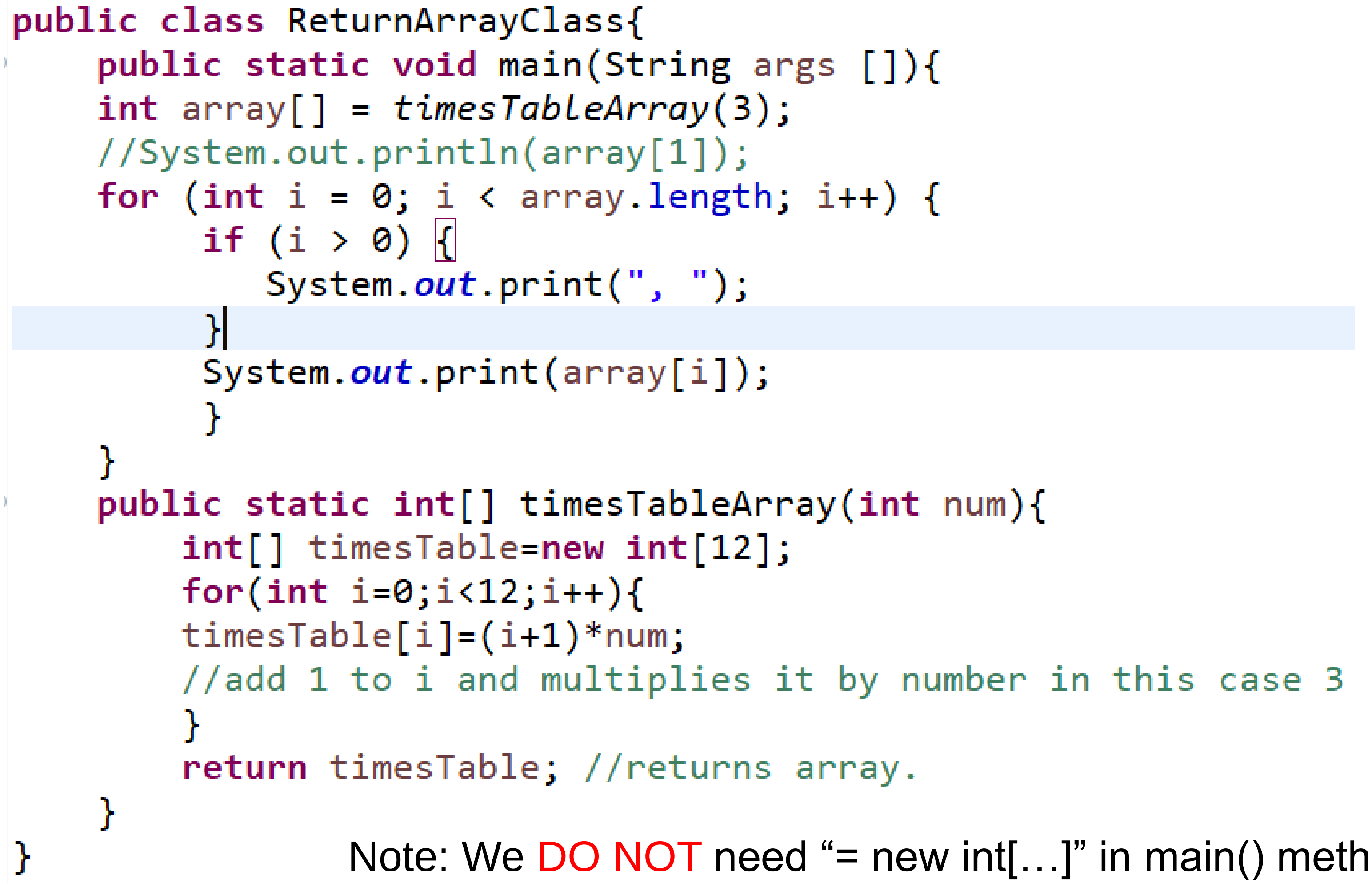
```
public class MethodScope1
{
    // instance variables - replace the example below with your own
    public static void main(String[] args)
    {
     int inValue = 4;
    //Actual parameters
    twoTimes(inValue);


    } //same names
// Formal parameters public static void twoTimes(int inValue)
   public  static void twoTimes(int inValue)
   {

       System.out.println("The result is "+inValue*2);

   }
}
```

Outline

- Returning object from method
- Passing value
- Passing reference
- Non-static method
- Calling chain of method

Returning an Array

Methods could return an array back to main() method to where it is called

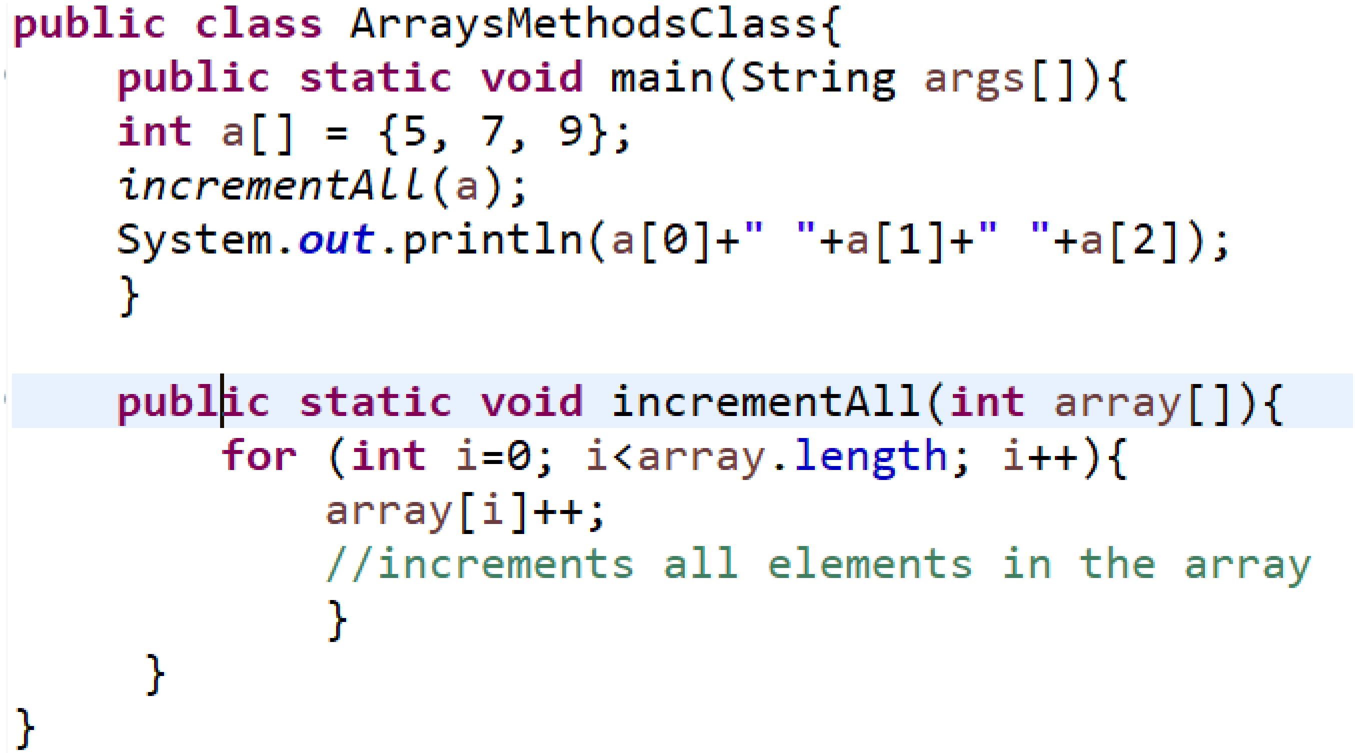


```
public class ReturnArrayClass{
    public static void main(String args []){
    int array[] = timesTableArray(3);
    //System.out.println(array[1]);
    for (int i = 0; i < array.length; i++) {
         if (i > 0) {
            System.out.print(", ");
         }
         System.out.print(array[i]);
         }
    }
    public static int[] timesTableArray(int num){
        int[] timesTable=new int[12];
        for(int i=0;i<12;i++){
        timesTable[i]=(i+1)*num;
        //add 1 to i and multiplies it by number in this case 3
        }
        return timesTable; //returns array.
    }
}
```

Note: We DO NOT need “= new int[…]” in main() method.

Arrays in Methods

Arrays may be used as parameters of methods. For example:

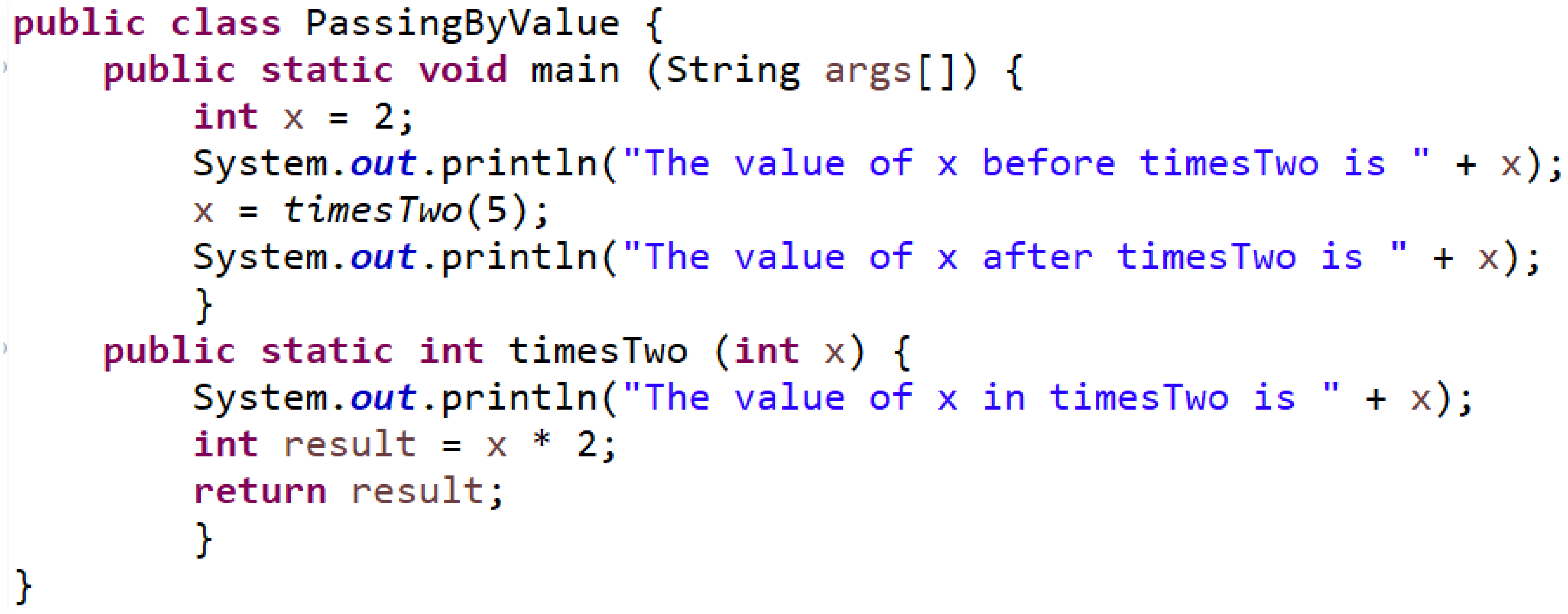


```
public class ArraysMethodsClass{
    public static void main(String args[]){
    int a[] = {5, 7, 9};
    incrementAll(a);
    System.out.println(a[0]+" "+a[1]+" "+a[2]);
    }

    public static void incrementAll(int array[]){
        for (int i=0; i<array.length; i++){
            array[i]++;
            //increments all elements in the array
            }
     }
}
```

When the above program is executed it displays the values 6 8 10 on the screen. Notice how we do not need to return the array back to the main() method. This is because we are not actually sending the array to the incremenlAll() method, we are sending a reference to the array.

Passing Variables

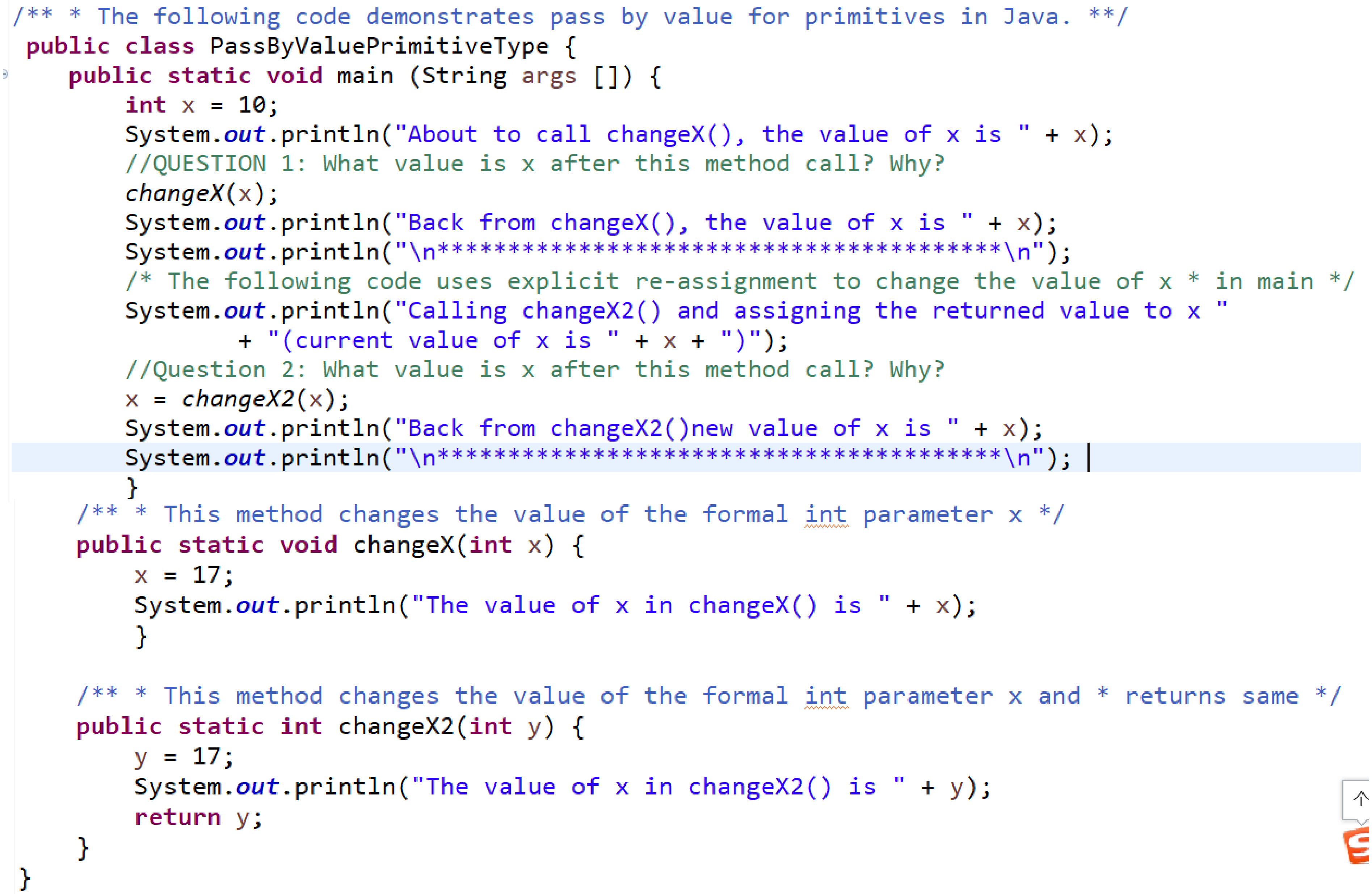
When we are passing variables between methods, the variables are passed by value. For example, when a primitive data type is passed into a method a copy of the value is sent to the method.



```
public class PassingByValue {
    public static void main (String args[]) {
        int x = 2;
        System.out.println("The value of x before timesTwo is " + x);
        x = timesTwo(5);
        System.out.println("The value of x after timesTwo is " + x);
        }
    public static int timesTwo (int x) {
        System.out.println("The value of x in timesTwo is " + x);
        int result = x * 2;
        return result;
        }
}
```

Note: 5 is passed to timesTwo() method and stored in a variable (also called x).

Passing Primitive data type- example



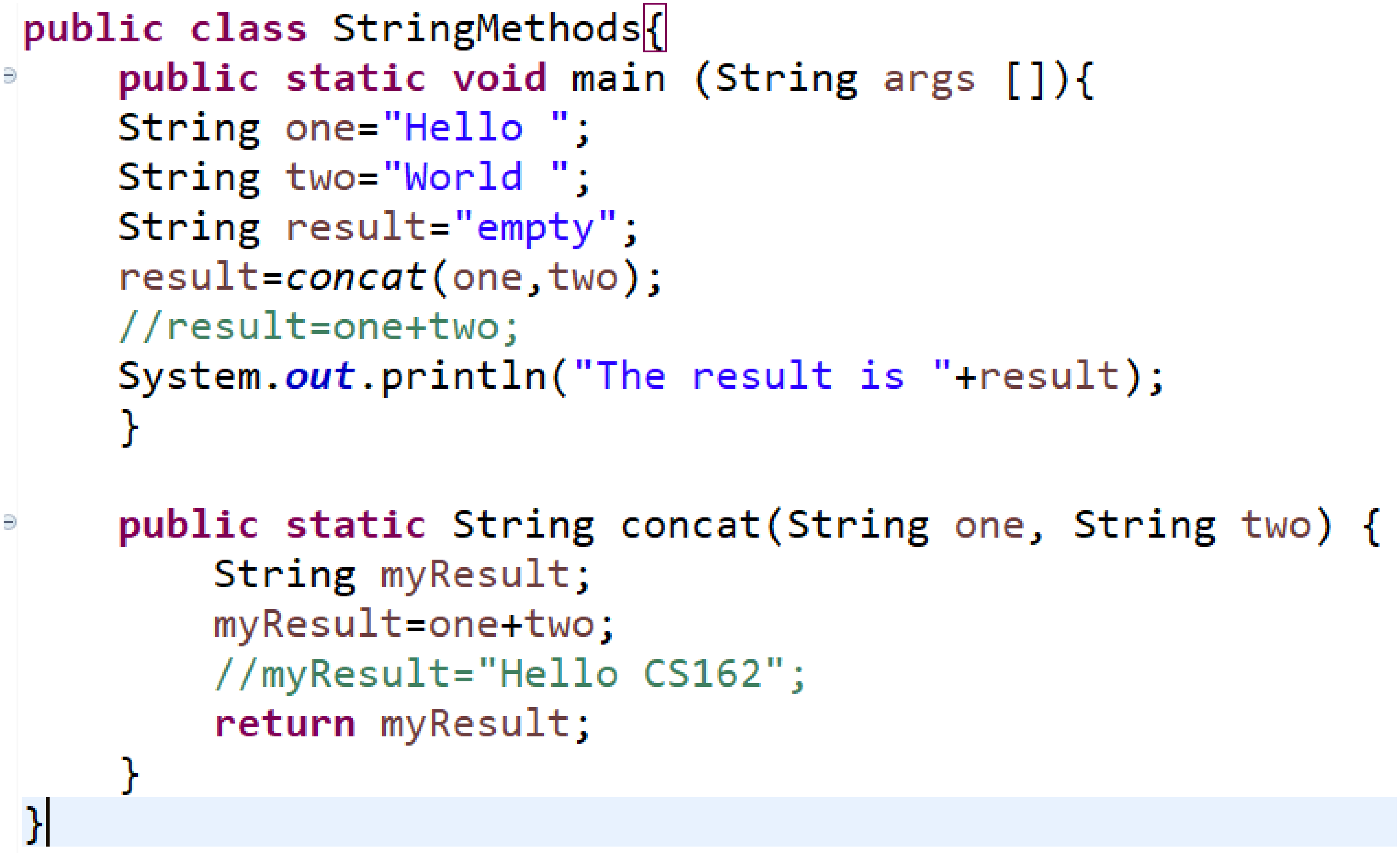
```
/** * The following code demonstrates pass by value for primitives in Java. **/
 public class PassByValuePrimitiveType {
    public static void main (String args []) {
        int x = 10;
        System.out.println("About to call changeX(), the value of x is " + x);
        //QUESTION 1: What value is x after this method call? Why?
        changeX(x);
        System.out.println("Back from changeX(), the value of x is " + x);
        System.out.println("\n****************************************\n");
        /* The following code uses explicit re-assignment to change the value of x * in main */
        System.out.println("Calling changeX2() and assigning the returned value to x "
                + "(current value of x is " + x + ")");
        //Question 2: What value is x after this method call? Why?
        x = changeX2(x);
        System.out.println("Back from changeX2()new value of x is " + x);
        System.out.println("\n****************************************\n");
        }
     /** * This method changes the value of the formal int parameter x */
     public static void changeX(int x) {
         x = 17;
         System.out.println("The value of x in changeX() is " + x);
         }

     /** * This method changes the value of the formal int parameter x and * returns same */
     public static int changeX2(int y) {
         y = 17;
         System.out.println("The value of x in changeX2() is " + y);
         return y;
     }
}
```

Passing Primitive data types

- This also works to other primitive data types, including short, long, float, double and Boolean.
- Try by yourself.

Arrays and Strings in Methods

Arrays and strings may be used as parameters of methods. For example:



```java
public class StringMethods{
    public static void main (String args []){
    String one="Hello ";
    String two="World ";
    String result="empty";
    result=concat(one,two);
    //result=one+two;
    System.out.println("The result is "+result);
    }

    public static String concat(String one, String two) {
        String myResult;
        myResult=one+two;
        //myResult="Hello CS162";
        return myResult;
    }
}
```

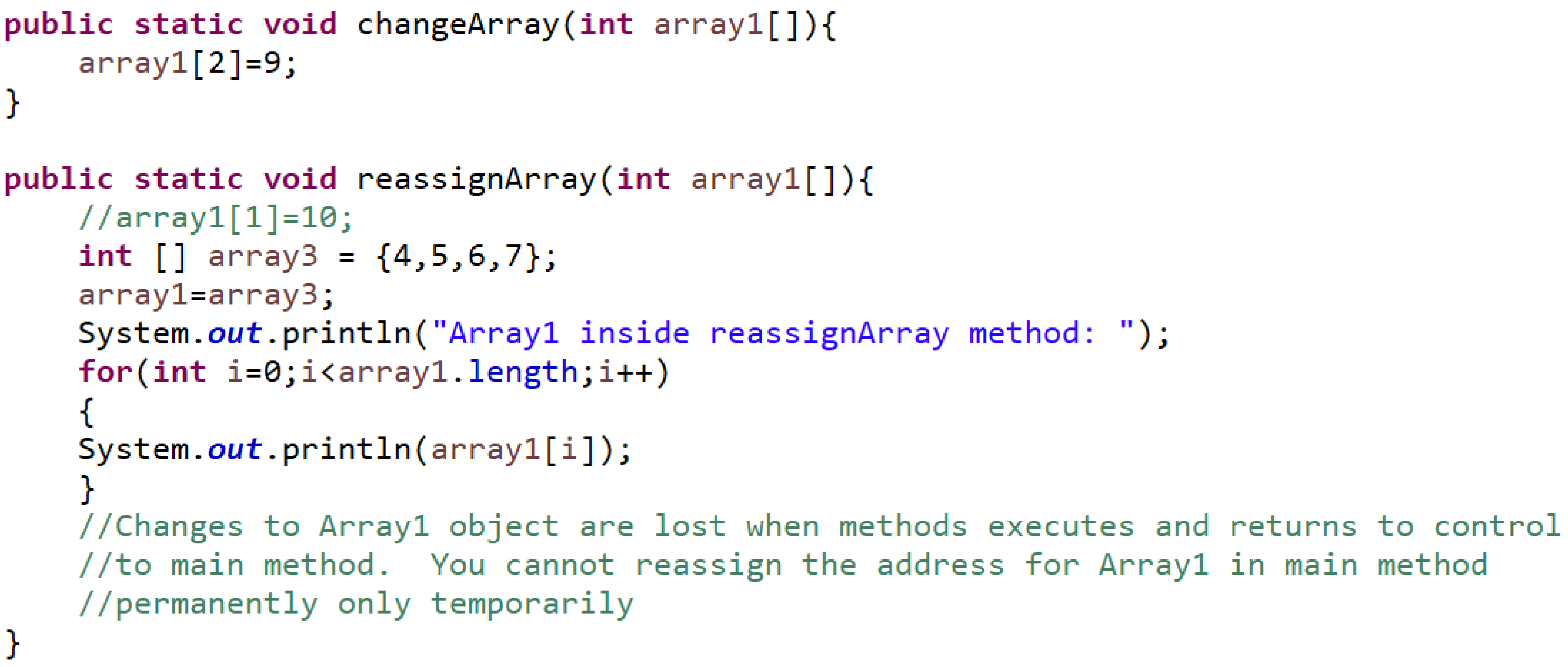
Passing Reference Types

When passing reference types of type object (e.g. arrays, Strings, etc.), java passes the memory address of that object to the method rather than passing the actual value.

When we create an array as follows:
int a[] = {1, 2, 3, 4}
a piece of memory is set aside that is referenced by the variable a. Here, a will store the memory address of the first value in the array.

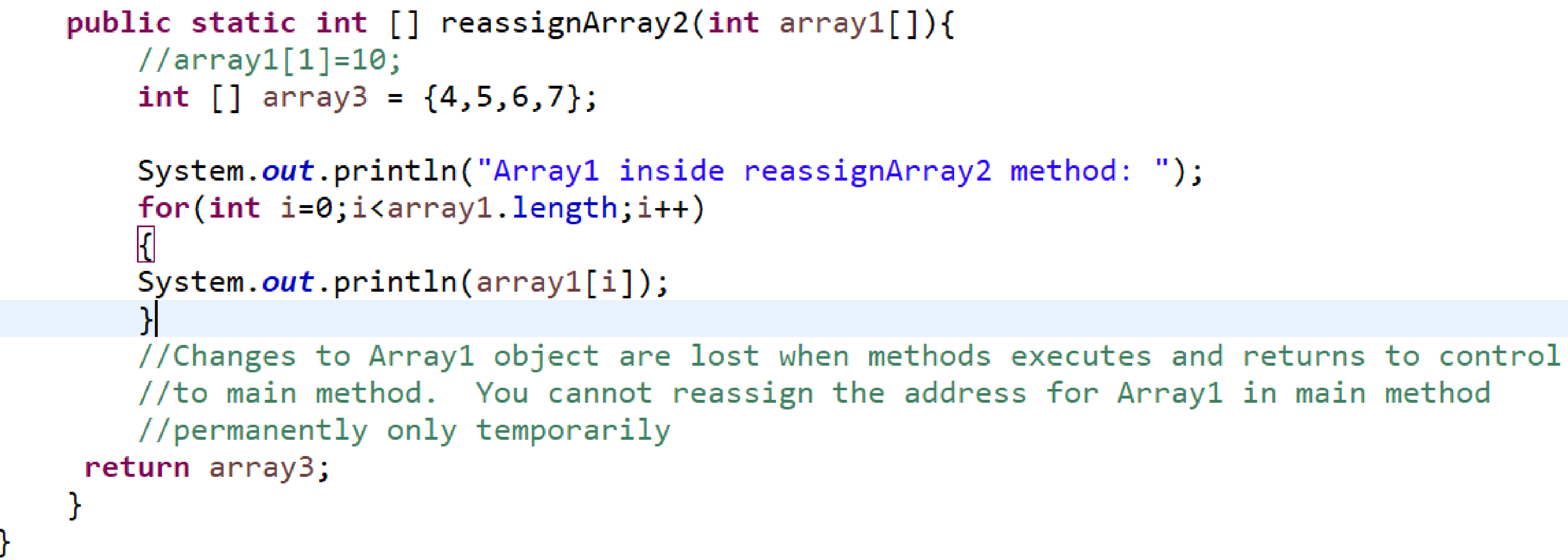
It is important to understand that what a forma parameter object is used, it modifies the actual object state unless you reassign the object. That is, change the address the variable is pointing to.

Passing Reference- Array Example



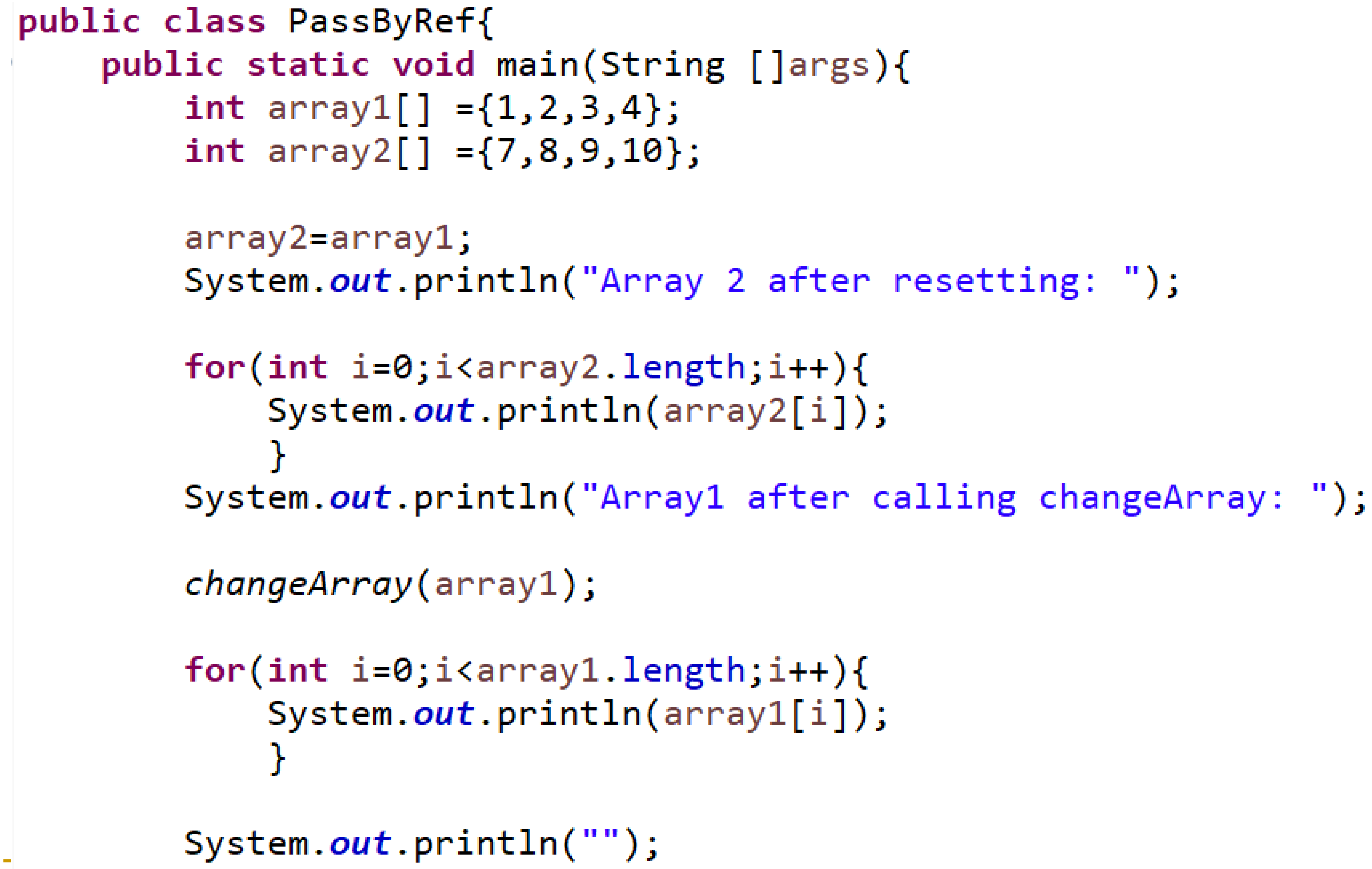
```
public static void changeArray(int array1[]){
    array1[2]=9;
}

public static void reassignArray(int array1[]){
    //array1[1]=10;
    int [] array3 = {4,5,6,7};
    array1=array3;
    System.out.println("Array1 inside reassignArray method: ");
    for(int i=0;i<array1.length;i++)
    {
    System.out.println(array1[i]);
    }
    //Changes to Array1 object are lost when methods executes and returns to control
    //to main method.  You cannot reassign the address for Array1 in main method
    //permanently only temporarily
}
```

Passing Reference- Array Example



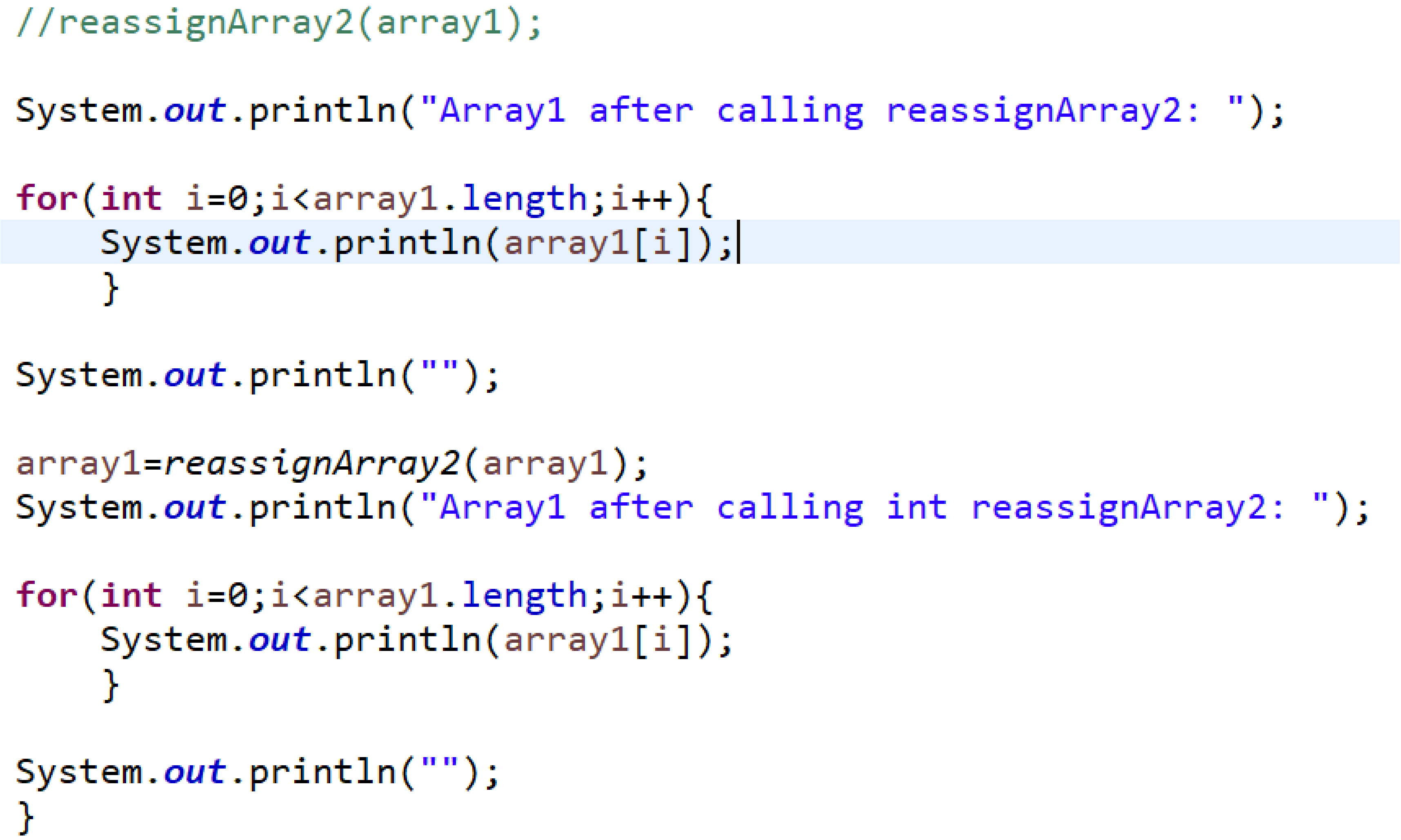
```
    public static int [] reassignArray2(int array1[]){
        //array1[1]=10;
        int [] array3 = {4,5,6,7};

        System.out.println("Array1 inside reassignArray2 method: ");
        for(int i=0;i<array1.length;i++)
        {
        System.out.println(array1[i]);
        }
        //Changes to Array1 object are lost when methods executes and returns to control
        //to main method.  You cannot reassign the address for Array1 in main method
        //permanently only temporarily
     return array3;
    }
}
```

Passing Reference- Array Example



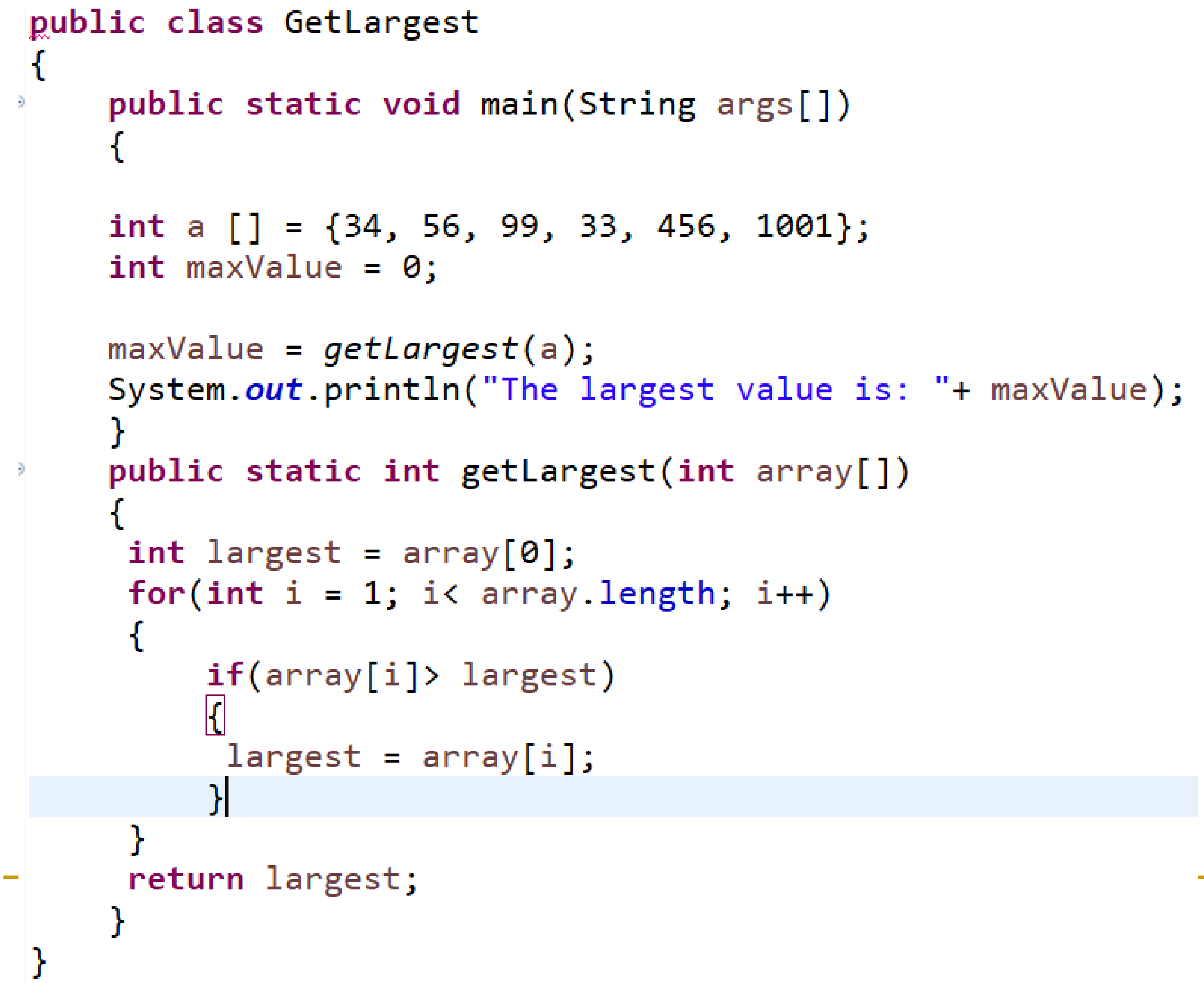
```
public class PassByRef{
    public static void main(String []args){
        int array1[] ={1,2,3,4};
        int array2[] ={7,8,9,10};

        array2=array1;
        System.out.println("Array 2 after resetting: ");

        for(int i=0;i<array2.length;i++){
            System.out.println(array2[i]);
            }
        System.out.println("Array1 after calling changeArray: ");

        changeArray(array1);

        for(int i=0;i<array1.length;i++){
            System.out.println(array1[i]);
            }

        System.out.println("");
```

Passing Reference- Array Example



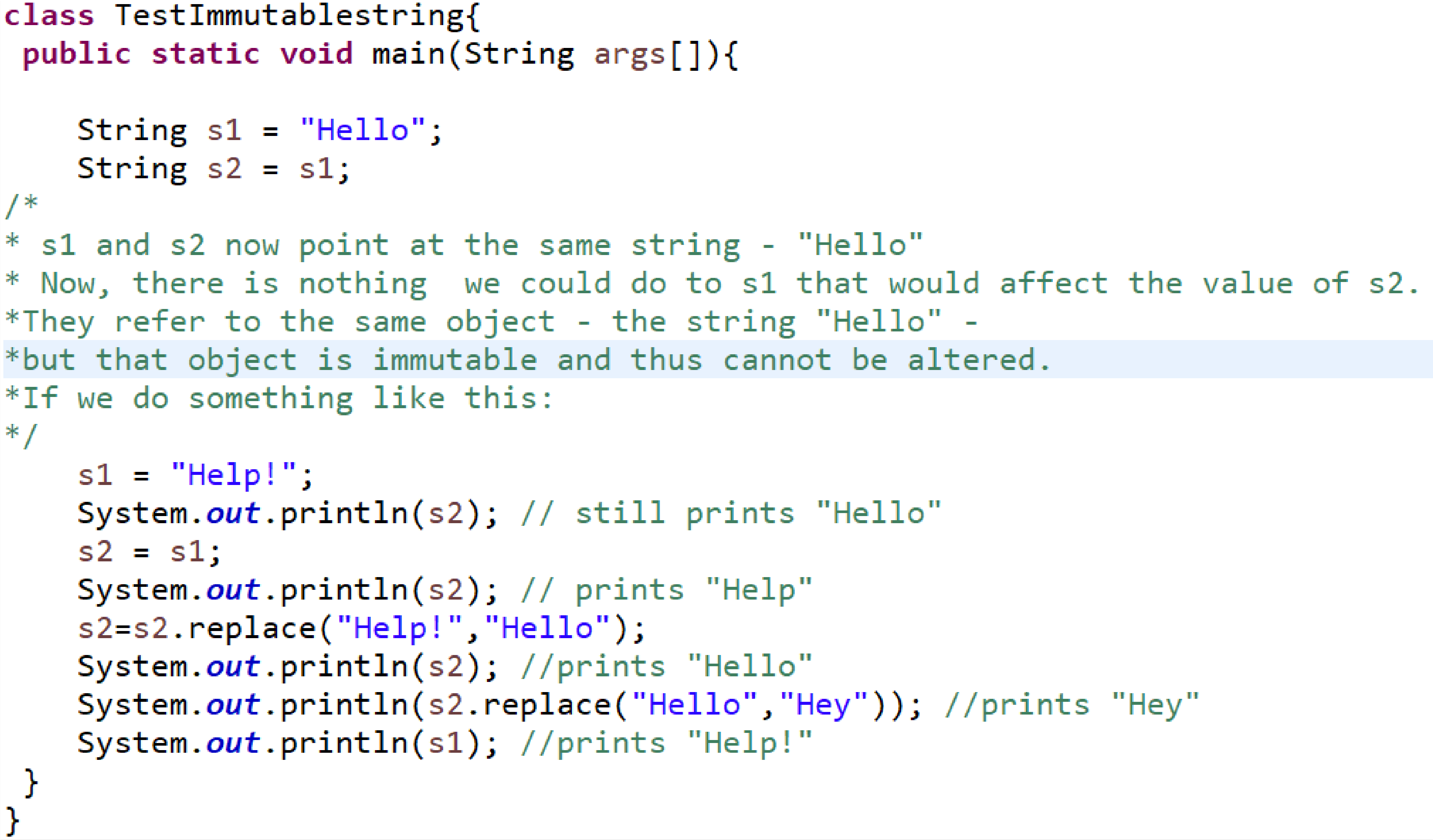
```
//reassignArray2(array1);

System.out.println("Array1 after calling reassignArray2: ");

for(int i=0;i<array1.length;i++){
    System.out.println(array1[i]);
    }

System.out.println("");

array1=reassignArray2(array1);
System.out.println("Array1 after calling int reassignArray2: ");

for(int i=0;i<array1.length;i++){
    System.out.println(array1[i]);
    }

System.out.println("");
}
```

Passing Reference- Array Example 2



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

    int a [] = {34, 56, 99, 33, 456, 1001};
    int maxValue = 0;

    maxValue = getLargest(a);
    System.out.println("The largest value is: "+ maxValue);
    }
    public static int getLargest(int array[])
    {
     int largest = array[0];
     for(int i = 1; i< array.length; i++)
     {
         if(array[i]> largest)
         {
          largest = array[i];
         }
     }
     return largest;
    }
}
```

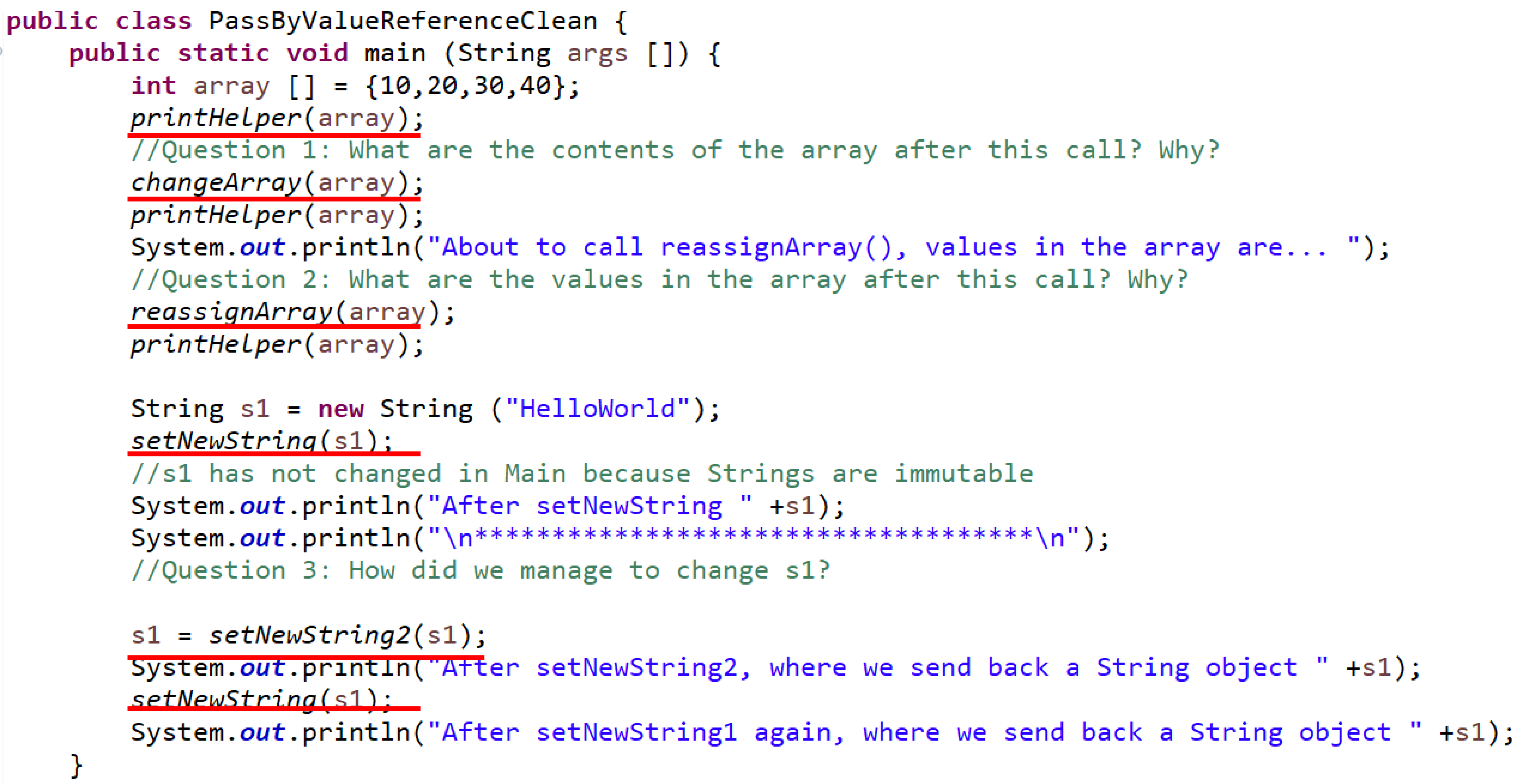
String is Immutable



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

    String s1 = "Hello";
    String s2 = s1;
/*
* s1 and s2 now point at the same string - "Hello"
* Now, there is nothing  we could do to s1 that would affect the value of s2.
*They refer to the same object - the string "Hello" -
*but that object is immutable and thus cannot be altered.
*If we do something like this:
*/
    s1 = "Help!";
    System.out.println(s2); // still prints "Hello"
    s2 = s1;
    System.out.println(s2); // prints "Help"
    s2=s2.replace("Help!","Hello");
    System.out.println(s2); //prints "Hello"
    System.out.println(s2.replace("Hello","Hey")); //prints "Hey"
    System.out.println(s1); //prints "Help!"
 }
}
```

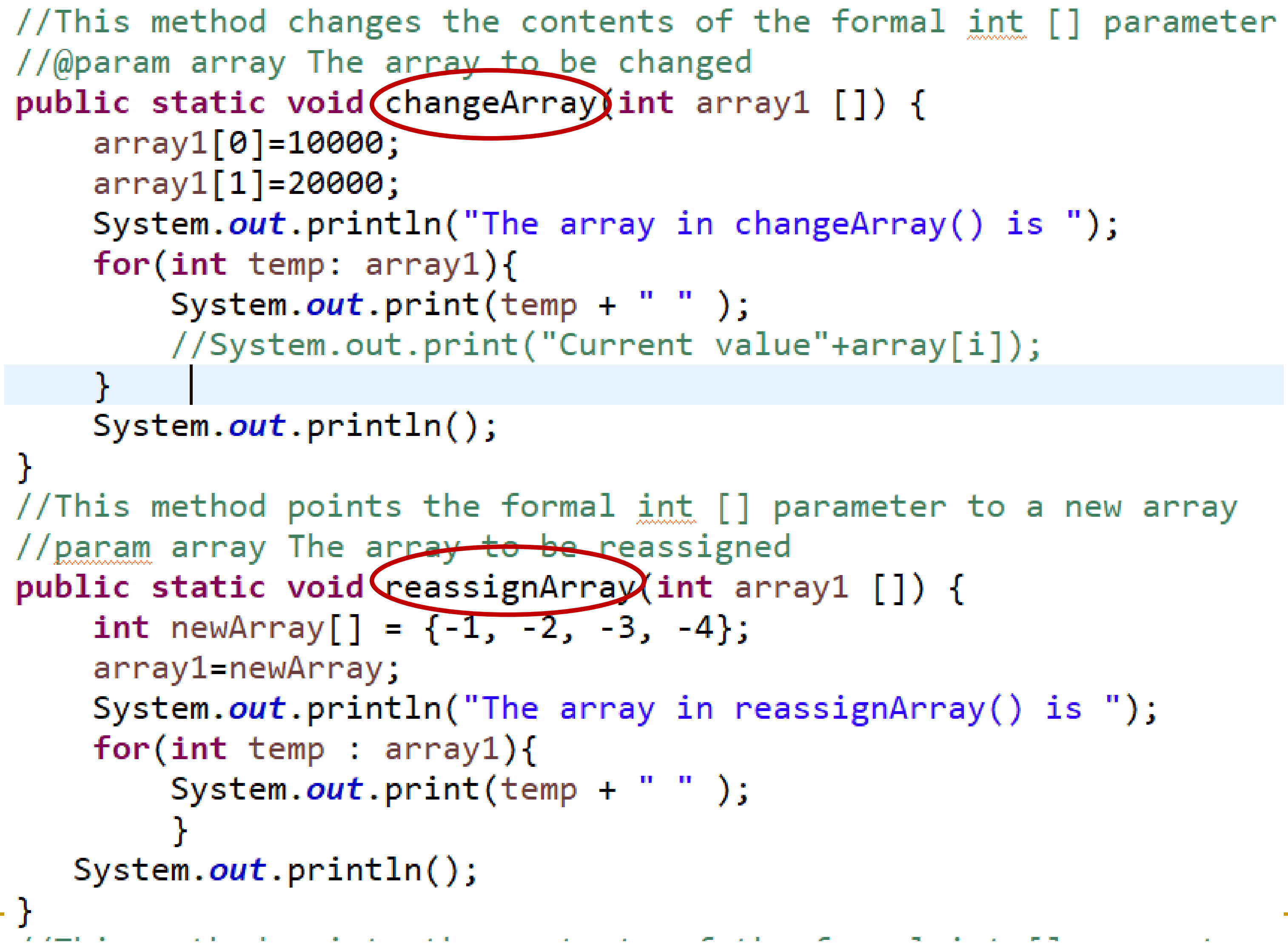
Note: String are immutable which means that they cannot change. All we can do is to create a new String to replace an older String.

Passing Reference- String Example



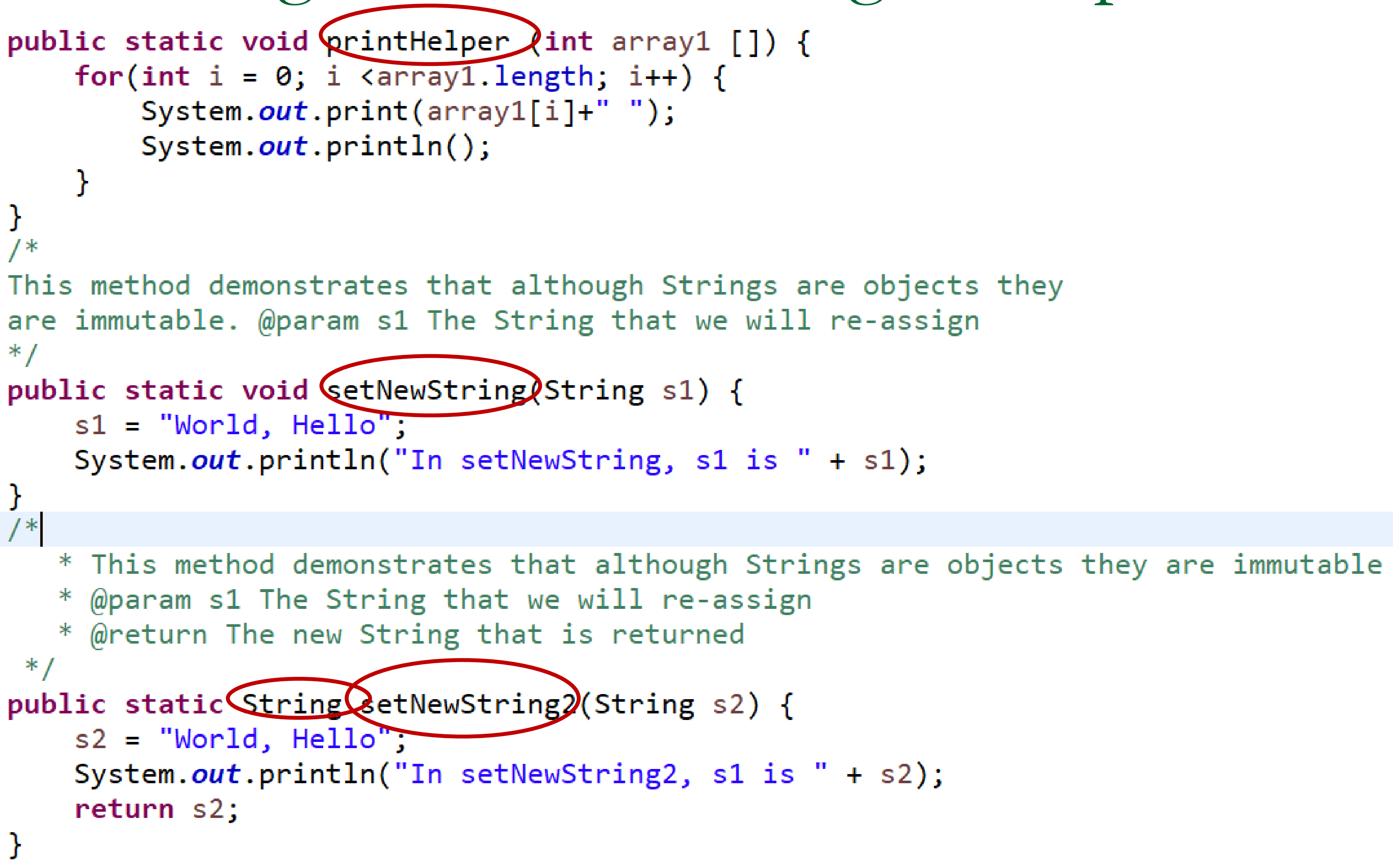
```java
public class PassByValueReferenceClean {
    public static void main (String args []) {
        int array [] = {10,20,30,40};
        printHelper(array);
        //Question 1: What are the contents of the array after this call? Why?
        changeArray(array);
        printHelper(array);
        System.out.println("About to call reassignArray(), values in the array are... ");
        //Question 2: What are the values in the array after this call? Why?
        reassignArray(array);
        printHelper(array);

        String s1 = new String ("HelloWorld");
        setNewString(s1);
        //s1 has not changed in Main because Strings are immutable
        System.out.println("After setNewString " +s1);
        System.out.println("\n************************************\n");
        //Question 3: How did we manage to change s1?

        s1 = setNewString2(s1);
        System.out.println("After setNewString2, where we send back a String object " +s1);
        setNewString(s1);
        System.out.println("After setNewString1 again, where we send back a String object " +s1);
    }
```

Passing Reference- String Example



```
//This method changes the contents of the formal int [] parameter
//@param array The array to be changed
public static void changeArray(int array1 []) {
    array1[0]=10000;
    array1[1]=20000;
    System.out.println("The array in changeArray() is ");
    for(int temp: array1){
        System.out.print(temp + " " );
        //System.out.print("Current value"+array[i]);
    }
    System.out.println();
}
//This method points the formal int [] parameter to a new array
//param array The array to be reassigned
public static void reassignArray(int array1 []) {
    int newArray[] = {-1, -2, -3, -4};
    array1=newArray;
    System.out.println("The array in reassignArray() is ");
    for(int temp : array1){
        System.out.print(temp + " " );
        }
   System.out.println();
}
```

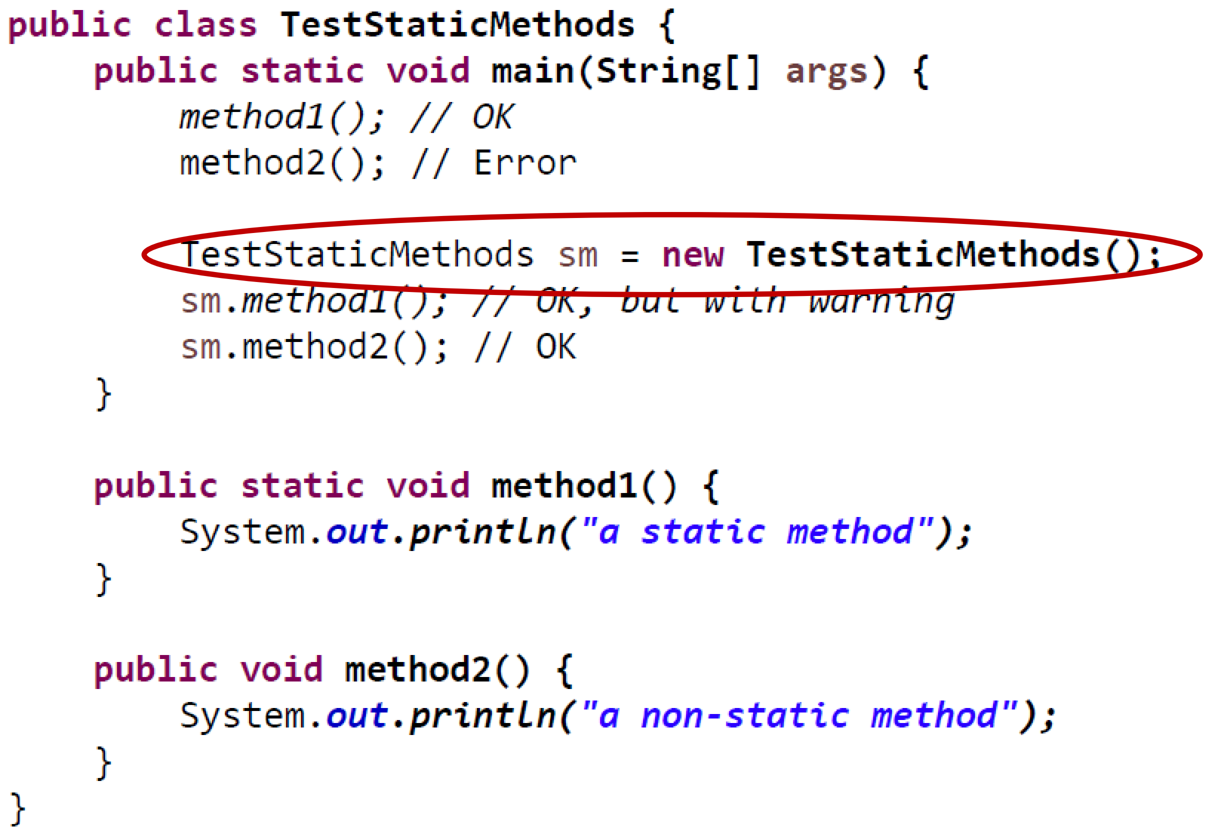
Passing Reference- String Example



```
public static void printHelper (int array1 []) {
    for(int i = 0; i <array1.length; i++) {
        System.out.print(array1[i]+" ");
        System.out.println();
    }
}
/*
This method demonstrates that although Strings are objects they
are immutable. @param s1 The String that we will re-assign
*/
public static void setNewString(String s1) {
    s1 = "World, Hello";
    System.out.println("In setNewString, s1 is " + s1);
}
/*
   * This method demonstrates that although Strings are objects they are immutable
   * @param s1 The String that we will re-assign
   * @return The new String that is returned
 */
public static String setNewString2(String s2) {
    s2 = "World, Hello";
    System.out.println("In setNewString2, s1 is " + s2);
    return s2;
}
```

Static Methods vs Non-static Method

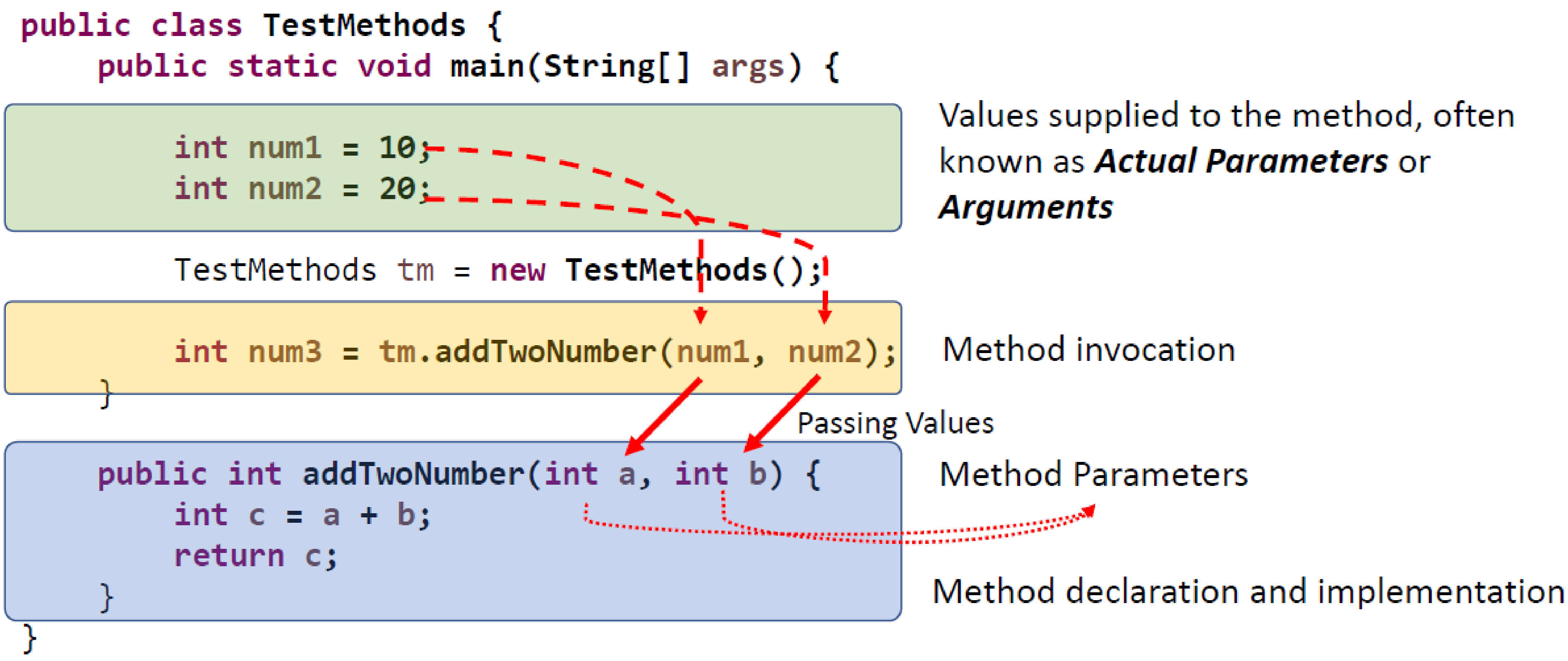
- A static method can only invoke other static method(s).
- Static methods are often called class methods, in contrast, non static methods are often called instance methods



```
public class TestStaticMethods {
    public static void main(String[] args) {
        method1(); // OK
        method2(); // Error

        TestStaticMethods sm = new TestStaticMethods();
        sm.method1(); // OK, but with warning
        sm.method2(); // OK
    }

    public static void method1() {
        System.out.println("a static method");
    }

    public void method2() {
        System.out.println("a non-static method");
    }
}
```

Instance Method Invocation

- Before invoke a non-static method, you should create an object first.
- By invoking a method, the caller must provide all values specified by the method’s parameter list.
- A method can be invoked (or called) by other methods.



```
public class TestMethods {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;

        TestMethods tm = new TestMethods();

        int num3 = tm.addTwoNumber(num1, num2);
    }

    public int addTwoNumber(int a, int b) {
        int c = a + b;
        return c;
    }
}
```

Values supplied to the method, often known as ***Actual Parameters*** or ***Arguments***

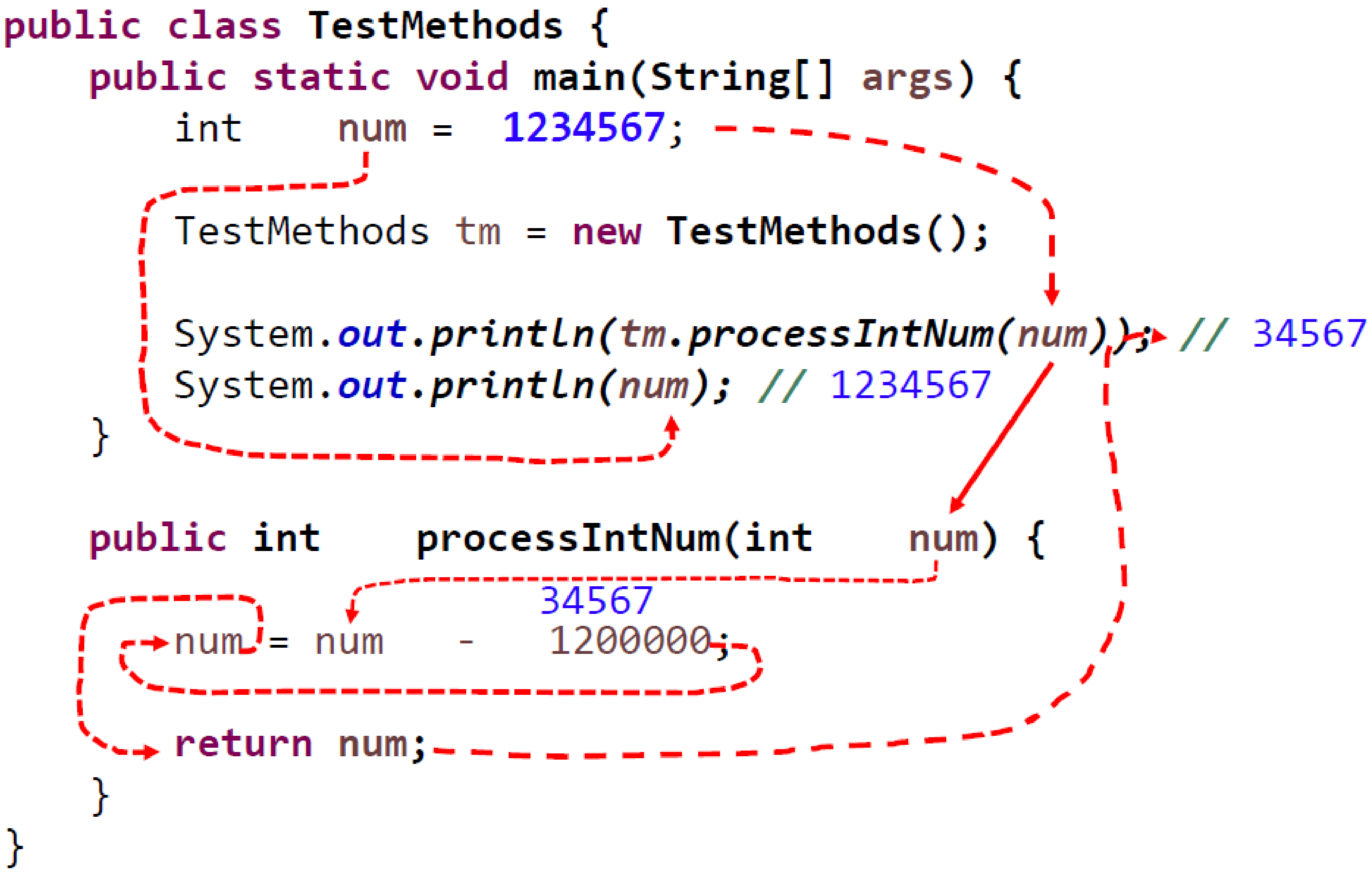
Method invocation

Passing Values

Method Parameters

Method declaration and implementation

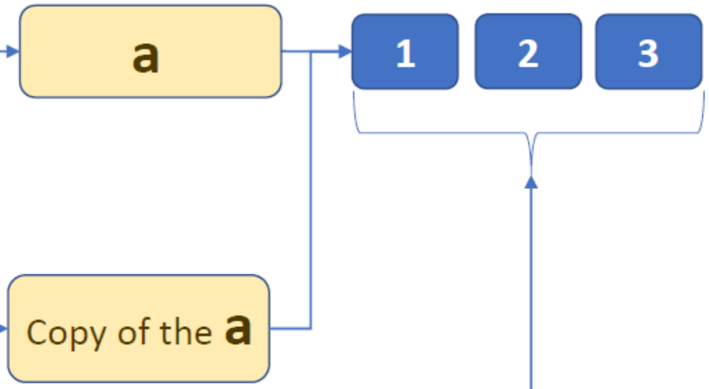
Passing by Values (Primitive Data Types)



```
public class TestMethods {
    public static void main(String[] args) {
        int     num =  1234567;

        TestMethods tm = new TestMethods();

        System.out.println(tm.processIntNum(num)); // 34567
        System.out.println(num); // 1234567
    }

    public int    processIntNum(int    num) {
        num = num    -    1200000;

        return num;
    }
}
```

Passing References by Values (Objects)

```java
import java.util.Arrays;
public class TestPassingRefByValue {
    public static void main(String[] args) {
        TestPassingRefByValue prv = new TestPassingRefByValue();

        int[] a = { 1, 2, 3 };

        prv.manipulateArray(a);
    }

    private void manipulateArray(int[] anArray) {
        anArray[0] = 4;
        anArray[1] = 5;
        anArray[2] = 6;
        System.out.println(Arrays.toString(anArray));

    }
}
```

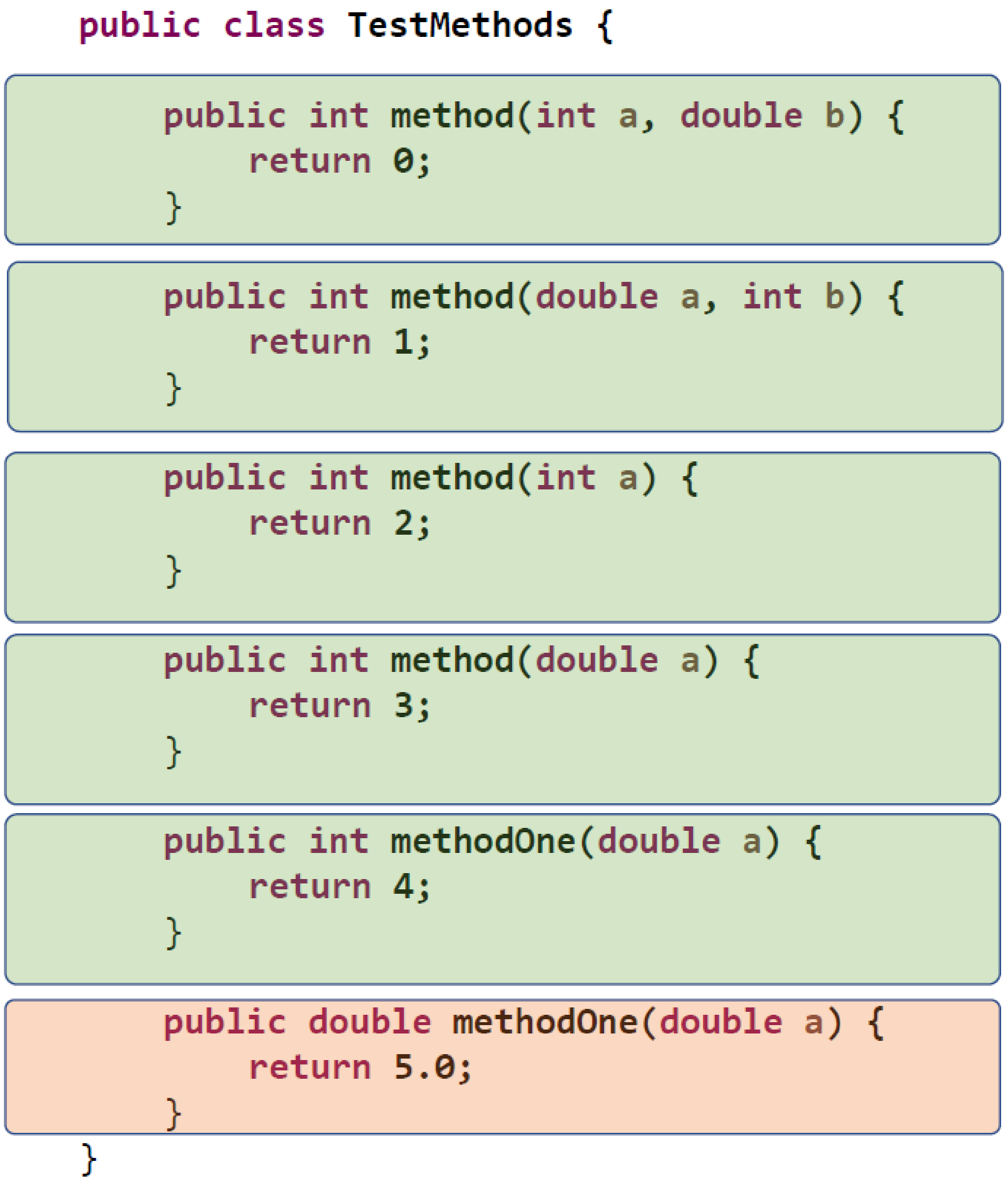


Exercise: Passing Reference by Values

```java
import java.util.Arrays;
public class TestPassingRefByValueSwapArray {
    public static void main(String[] args) {
        TestPassingRefByValueSwapArray prv = new TestPassingRefByValueSwapArray();

        int[] a = { 1, 2, 3 };
        int[] b = { 4, 5, 6 };

        prv.swapArray(a, b);

        System.out.println(Arrays.toString(a));
        System.out.println(Arrays.toString(b));
    }

    private void swapArray(int[] a, int[] b) {
        a[0] = 7;
        a[1] = 8;
        a[2] = 9;
        int[] temp = b;
        b = a;
        a = temp;
    }
}
```

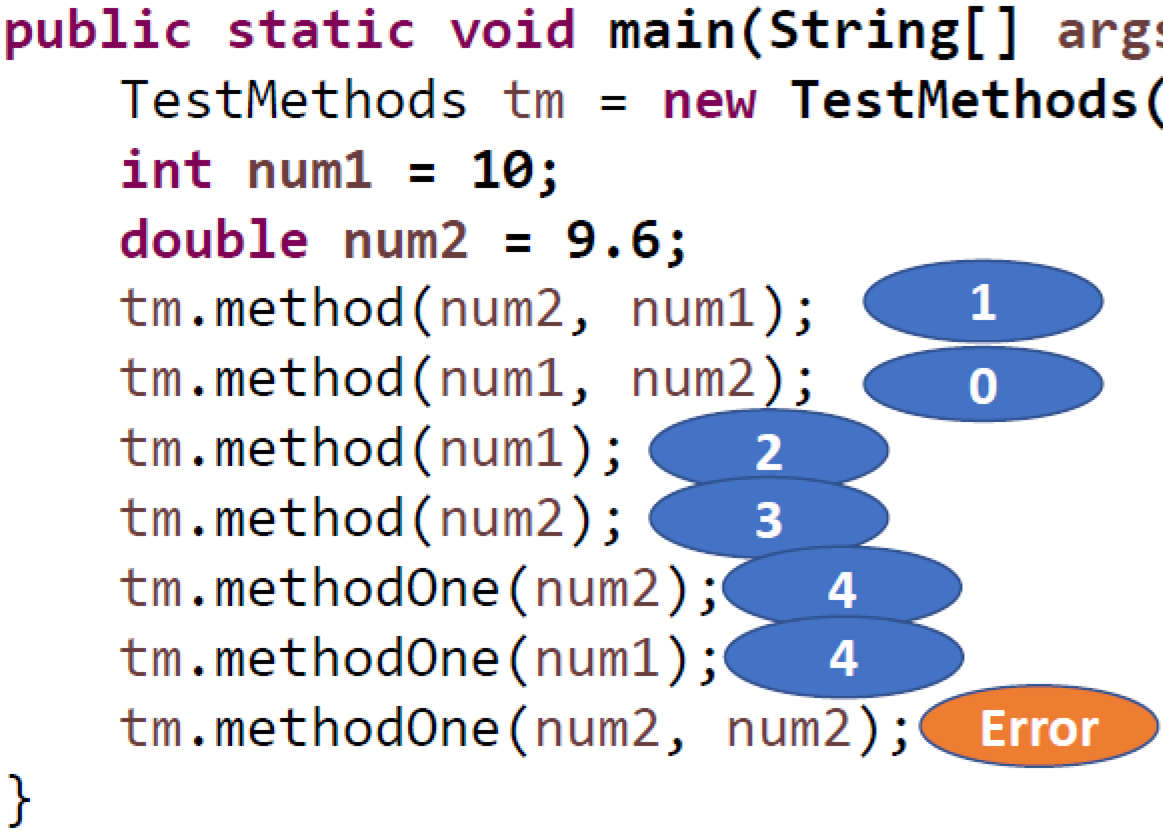
```
[7, 8, 9]
[4, 5, 6]
```

Example



```
public class TestMethods {
        public int method(int a, double b) {
            return 0;
        }

        public int method(double a, int b) {
            return 1;
        }

        public int method(int a) {
            return 2;
        }

        public int method(double a) {
            return 3;
        }

        public int methodOne(double a) {
            return 4;
        }

        public double methodOne(double a) {
            return 5.0;
        }
}
```

Examples

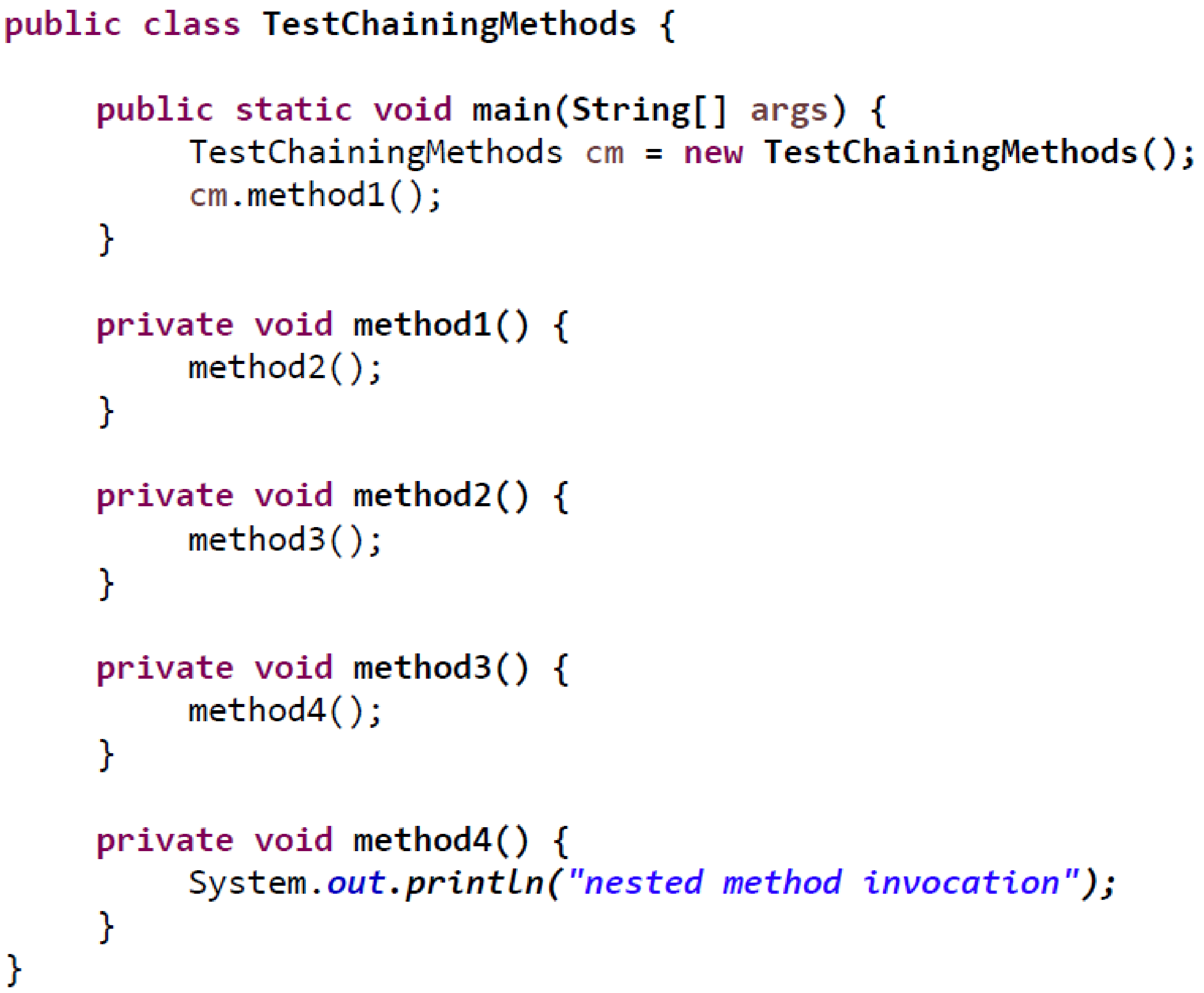


```
public static void main(String[] args) {
    TestMethods tm = new TestMethods();
    int num1 = 10;
    double num2 = 9.6;
    tm.method(num2, num1);          1
    tm.method(num1, num2);          0
    tm.method(num1);         2
    tm.method(num2);         3
    tm.methodOne(num2);       4
    tm.methodOne(num1);       4
    tm.methodOne(num2, num2);    Error
}
```

Chaining Method Calls

- A method can invoke as many methods as it needed. Methods can have nested calls of other methods. The depth of the method calls is unlimited.

- There is no limit on the number of methods can be defined in a class

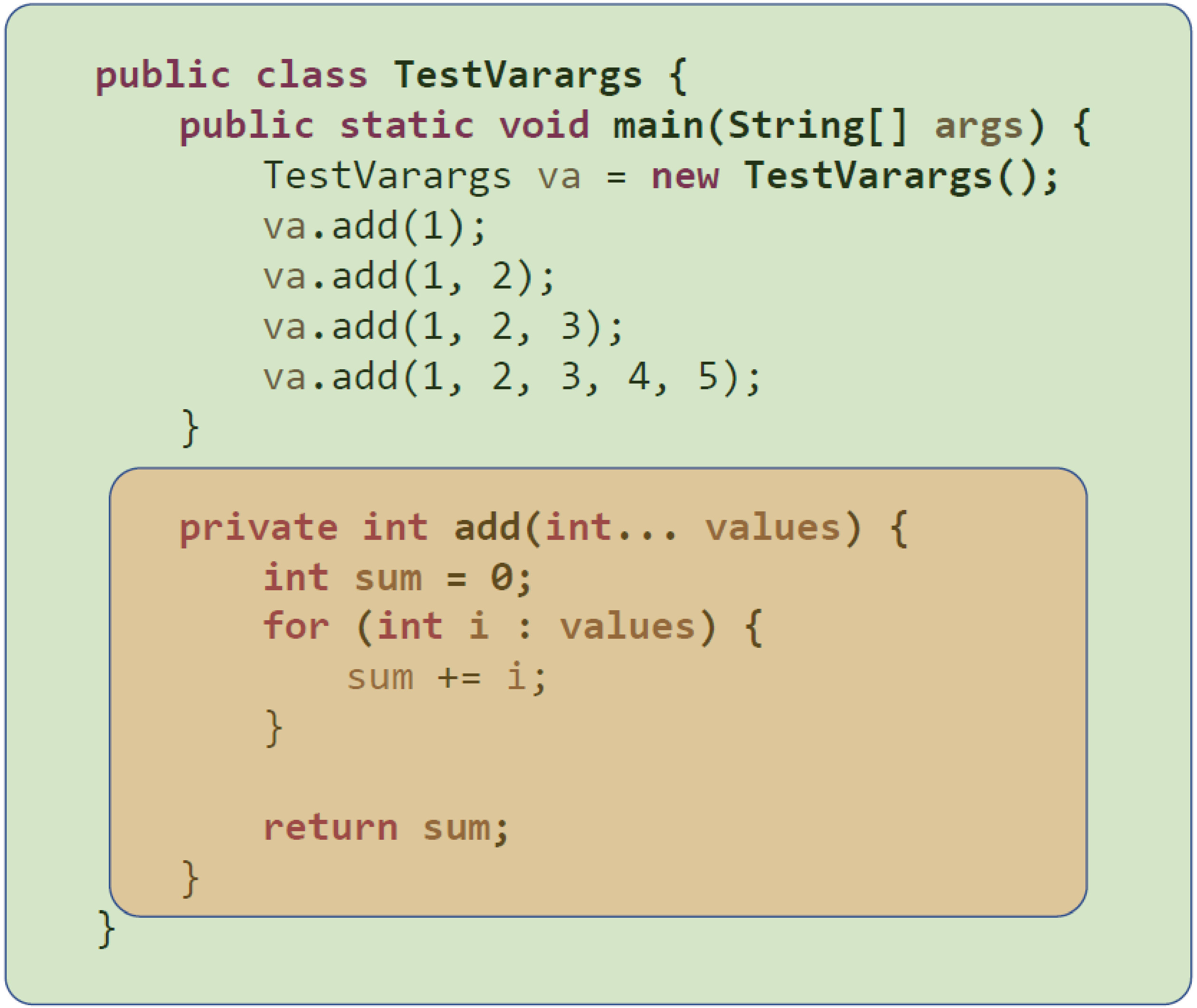


```
public class TestChainingMethods {

    public static void main(String[] args) {
        TestChainingMethods cm = new TestChainingMethods();
        cm.method1();
    }

    private void method1() {
        method2();
    }

    private void method2() {
        method3();
    }

    private void method3() {
        method4();
    }

    private void method4() {
        System.out.println("nested method invocation");
    }
}
```

Methods with Unknown Number of Parameters

- The technique that allows a method to take an arbitrary number of parameters is known as varargs (variable arguments).

Example



```java
public class TestVarargs {
    public static void main(String[] args) {
        TestVarargs va = new TestVarargs();
        va.add(1);
        va.add(1, 2);
        va.add(1, 2, 3);
        va.add(1, 2, 3, 4, 5);
    }

    private int add(int... values) {
        int sum = 0;
        for (int i : values) {
            sum += i;
        }

        return sum;
    }
}
```

Scope of Variables

"The scope of a variable is the part of the program where the variable can be referenced"

- A variable defined inside a method is called a local variable .
- A local variable must be declared and initialized before it can be used.
- Variables defined at the class level is called class variables.