Java Methods

User-Defined Parameter Methods

```
// ExampleOne.java
// This program introduces user-defined methods with parameters.
// This examples demonstrates
// the the mechanics and the manner in which information
// is passed from one program module to another program module.

public class ExampleOne
{
    public static void main(String args[])  //main method
    {
        displayParameter(100);              //method call
    } // end of main

    public static void displayParameter(int number)  //method header
    {  //body of method
        System.out.println();
        System.out.println("The parameter value is " + number);
        System.out.println();
    }    // end of displayParameter
}
```

Parameters Terminology

Argument: variable assignment

The **values** used in the method call.

This is the information that you are sending to the method.

Method Call:

```
showSum(10,15);
```

Parameter: variable declaration

The parameter is a local variable declared in the method header. The values of these variables are assigned during method call.

```
public static void showSum(int n1, int n2)
```

```
// Example2.java
/*This program demonstrates that the calling parameter can be:
a constant, a variable, a String,
an expression with only constants,
an expression with a variable and a constant,
or a call to a method that returns a value */

public class Example2
{
	public static void main(String args[])
	{
		System.out.println("\nExample2.JAVA\n");
		double value = 100;
		displayParameter(100);			// constant
		displayParameter(value);		// variable
		displayParameter(100 + 5);		// expression
		displayParameter(value + 5);		//expression containing variable and constant
		displayParameter(Math.sqrt(100));	// call to another method
		System.out.println();
	}
	public static void displayParameter(double number)
	{
		System.out.println();
		System.out.println("The parameter value is " + number);
	}
}
```

```
// Example3.java
// This program demonstrates passing two arguments to a method.
// The <showArea> method is called twice.  In this case reversing
// the sequence of the arguments is not a problem.

public class Example3{
      public static void main(String args[])
      {
            System.out.println("\nExample3.JAVA\n");
            int width = 100;
            int height = 50;
            showArea(width, height);
            showArea(height, width);
            System.out.println();
      }

      public static void showArea(int w, int h )
      {
            System.out.println();
            int area = w * h;
            System.out.println("The rectangle area is " + area);
            System.out.println();
      }
}
```

Argument Sequence Matters

The first argument passes the copied value to the first parameter.

The second argument passes the copied value to the second parameter.

Parameters placed out of sequence may result in *compile errors* or *logic errors*.

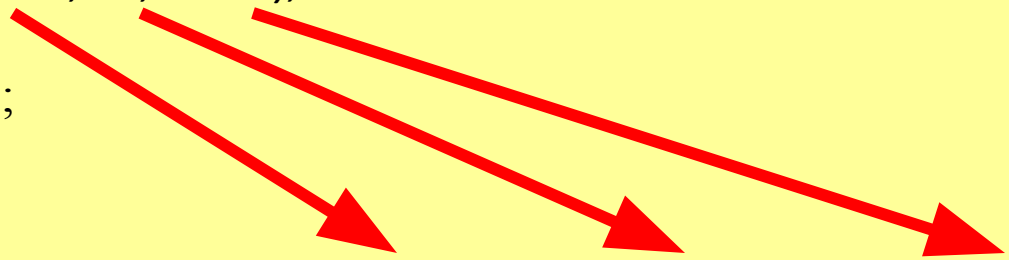
```
// Example6.java
// This program demonstrates that multiple parameters may be different data types.

public class Example6
{
    public static void main(String args[])
    {
        System.out.println("\nExample6.JAVA\n");

        // 3 different type parameters method call
        multiTypeDemo("Hans", 30, 3.575);

        System.out.println();
    }

    public static void multiTypeDemo(String studentName, int studentAge, double studentGPA)
    {
        System.out.println("\nThis method has 3 parameters with three different types");
        System.out.println("Name: " + studentName);
        System.out.println("Age:  " + studentAge);
        System.out.println("GPA:  " + studentGPA);
    }
}
```



Argument sequence is important!

```
// Example4.java
// This program demonstrates that argument sequence matters.
// In this example method <showDifference> will display different
// results when the calling arguments are reversed.

public class Example4 {

    public static void main(String args[])
    {
        System.out.println("\nExample4.JAVA\n");
        int num1 = 100;
        int num2 = 50;
        showDifference(num1, num2);      //output: The difference is 50
        showDifference(num2, num1);      //output: The difference is -50
        System.out.println();
    }

    public static void showDifference(int a, int b)
    {
        System.out.println();
        int difference = a - b;
        System.out.println("The difference is " + difference);
        System.out.println();
    }
}
```

Parameter Rules

The arguments in the method call and the parameters in the method heading must match in three ways:

1. They must be the same quantity
2. They must be the same type
3. They must be the same sequence

Important Rules About Using Parameters with Methods

- The number of arguments in the method call must match the number of parameters in the method header.
- The arguments in the method call must be the same type as the corresponding parameters in the header.
- The sequence of the arguments in the method call must match the sequence of the parameters in the heading.
- The argument identifiers in the method call may be the same identifier or a different identifier as the parameters in the heading.

Common Mistakes

Method Header vs. Call

Wrong	Correct

//method call
qwerty(int num1, int num2);
	int num1 = 100;
	int num2 = 200;
	qwerty(num1,num2);

//method header public static void qwerty(int a, b);	public static void qwerty(int a, int b)

```
// Example5.java
// This program demonstrates a COMMON MISTAKES made by students.
// Parameters are declared in the method heading, but may not be
// declared in the method call.  This program will not compile.

public class Example5
{
    public static void main(String args[])
    {
        System.out.println("\n Example5.JAVA\n");
        showDifference(int num1, int num2);          // common mistake 1
        System.out.println();
    }

    public static void showDifference(int a, b)          // common mistake 2
    {
        System.out.println();
        int difference = a - b;
        System.out.println("The difference is " + difference);
        System.out.println();
    }
}
```

Common Confusion

```
public static void main(String[ ] args) {
    int num = 1;
    prestoChango(num);
    System.out.println(num);    //what prints?
}

public static void prestoChango(int num) {
    num = 2;
}
```

Is the variable `num` in the main method the same variable as `num` in the `prestoChango` method?

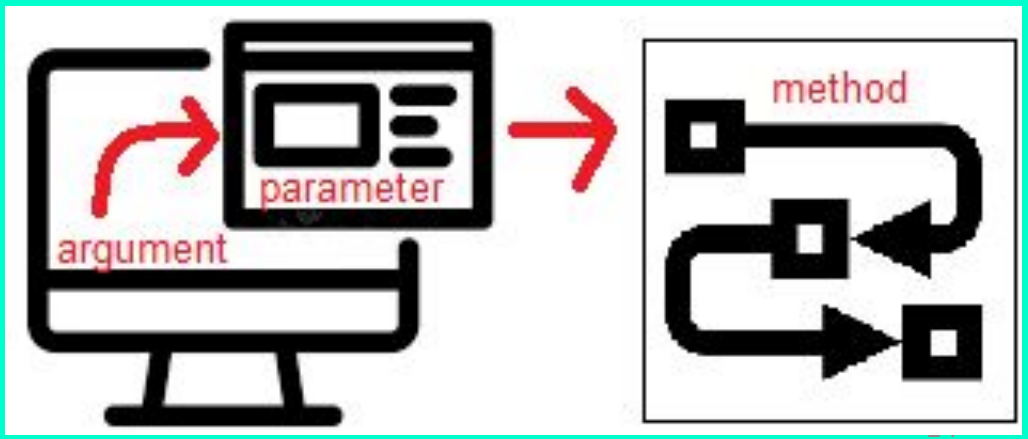
What prints out after the main method runs?

Did the method change the value of the main method variable `num`?

Remember, a **copy** of the argument's value is passed and assigned to the parameter.

The Screenshot Analogy

1) Think of the *argument* like information on a screen.
2) When a method is called, the method takes a screenshot of the argument. The image is a **copy** of what was on the screen.
3) This screenshot is passed to the *parameter*.
4) The method can modify the image or use it to execute other tasks (based on the statements in the block).
5) When the method has finished running, the screenshot is thrown in the garbage.

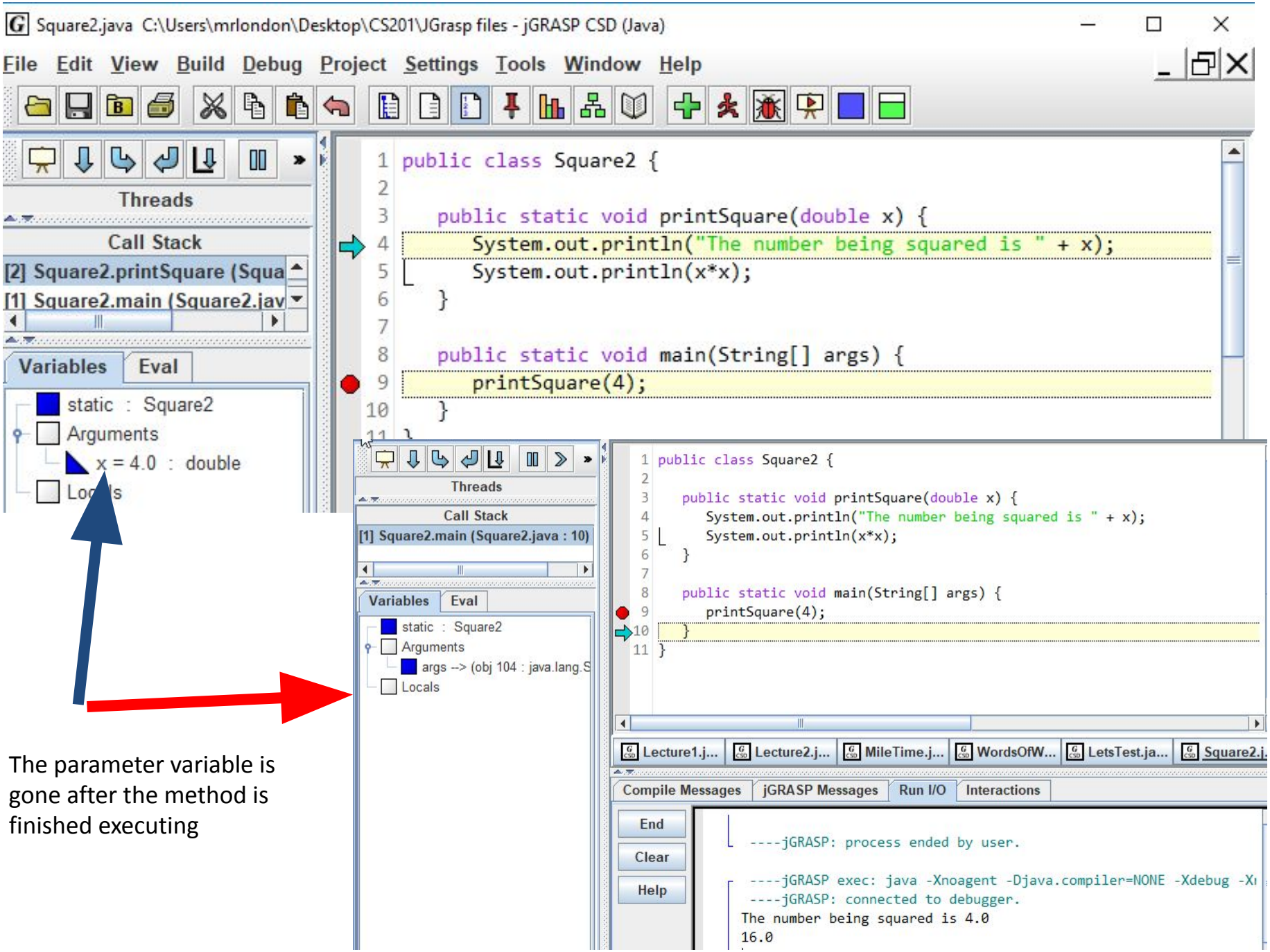


***Did the method change anything on the original screen?*

Disappearing Parameters!

What happens to the local variable (parameter) after the method is finished executing?

It is automatically destroyed by the Java garbage collector (really!).



The parameter variable is gone after the method is finished executing

Scanners as parameters

- If many methods need to read input, declare a `Scanner` in `main` and pass it to the other methods as a parameter.

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int sum = readSum3(console);
    System.out.println("The sum is " + sum);
}

// Prompts for 3 numbers and returns their sum.
public static int readSum3(Scanner console) {
    System.out.print("Type 3 numbers: ");
    int num1 = console.nextInt();
    int num2 = console.nextInt();
    int num3 = console.nextInt();
    return num1 + num2 + num3;
}
```

so you don't have to create many Scanner objects, one in each method!