Java Chapter 5: Methods

Contents

Time required: 90 minutes

Video Walkthrough

https://somup.com/c36QqTvhVI

DRY

Don't **R**epeat **Y**ourself (**DRY**) is a principle of software engineering aimed at reducing repetition of software patterns. It you are repeating any code, there is probably a better solution.

Read: Think Java 2

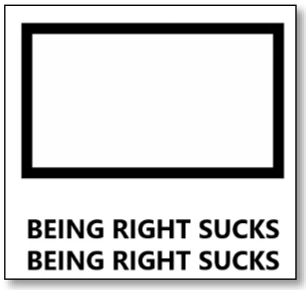
- Chapter 4 Methods and Testing

Online Tutorials

- Java Methods
- Java Method Parameters
- Java Method Overloading
- Java Recursion
- Java Scope
- Java Math Methods

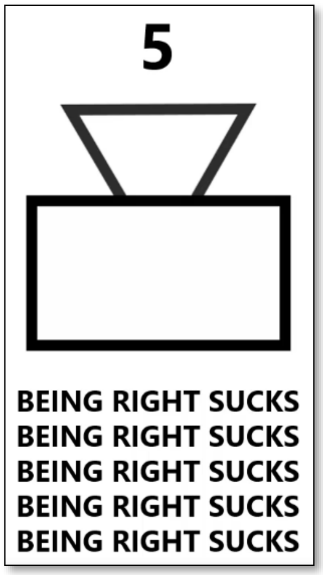
Java Methods

Void method

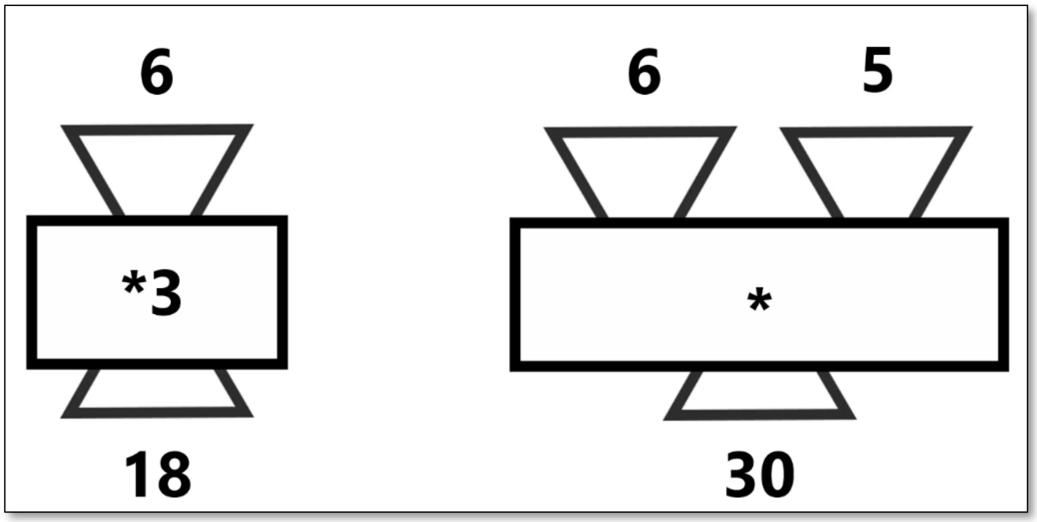


```
BEING RIGHT SUCKS
BEING RIGHT SUCKS
```

Void method with parameter



A method can take an input, does something, then can return an output.



Tutorial 5.1: Void Method

Time to create our first method. This program has two methods, **message()** and **main()**. The **main()** method is where the program starts.

Put in your own favorite saying in this tutorial.

```java
// Name: VoidMessageMethod.java
// Written by:
// Written on:
// Purpose: Void method

public class VoidMessageMethod {
   public static void main(String[] args) {
      System.out.println("First method call");
      // Method call
      message();
      System.out.println("\nSecond method call");
      // Method call
      message();
      System.out.println("\nThird method call");
      // Method call
      message();
   }
   // Create void method
   static void message() {
      System.out.println("It is not necessary to change. Survival is not mandatory.");
      System.out.println("W. Edwards Deming");
   }
}



```

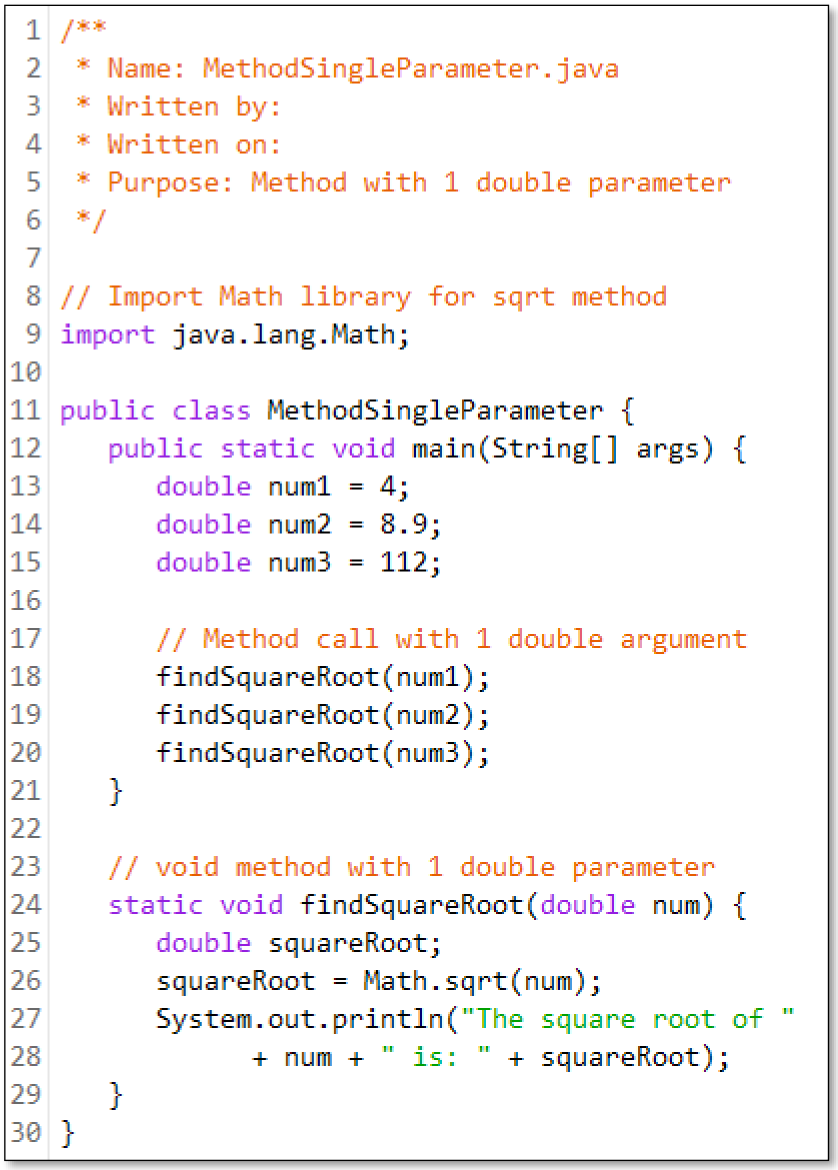
Example run:

```
First method call
It is not necessary to change. Survival is not mandatory.
W. Edwards Deming

Second method call
It is not necessary to change. Survival is not mandatory.
W. Edwards Deming

Third method call
It is not necessary to change. Survival is not mandatory.
W. Edwards Deming
```

Tutorial 5.2: Method with a Single Parameter

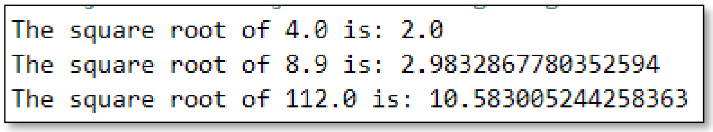
The following program has a method with a single parameter.

Create a Java program named: **MethodSingleParameter.java**



```java
/**
 * Name: MethodSingleParameter.java
 * Written by:
 * Written on:
 * Purpose: Method with 1 double parameter
 */

// Import Math library for sqrt method
import java.lang.Math;

public class MethodSingleParameter {
   public static void main(String[] args) {
      double num1 = 4;
      double num2 = 8.9;
      double num3 = 112;

      // Method call with 1 double argument
      findSquareRoot(num1);
      findSquareRoot(num2);
      findSquareRoot(num3);
   }

   // void method with 1 double parameter
   static void findSquareRoot(double num) {
      double squareRoot;
      squareRoot = Math.sqrt(num);
      System.out.println("The square root of "
            + num + " is: " + squareRoot);
   }
}
```

Example run:

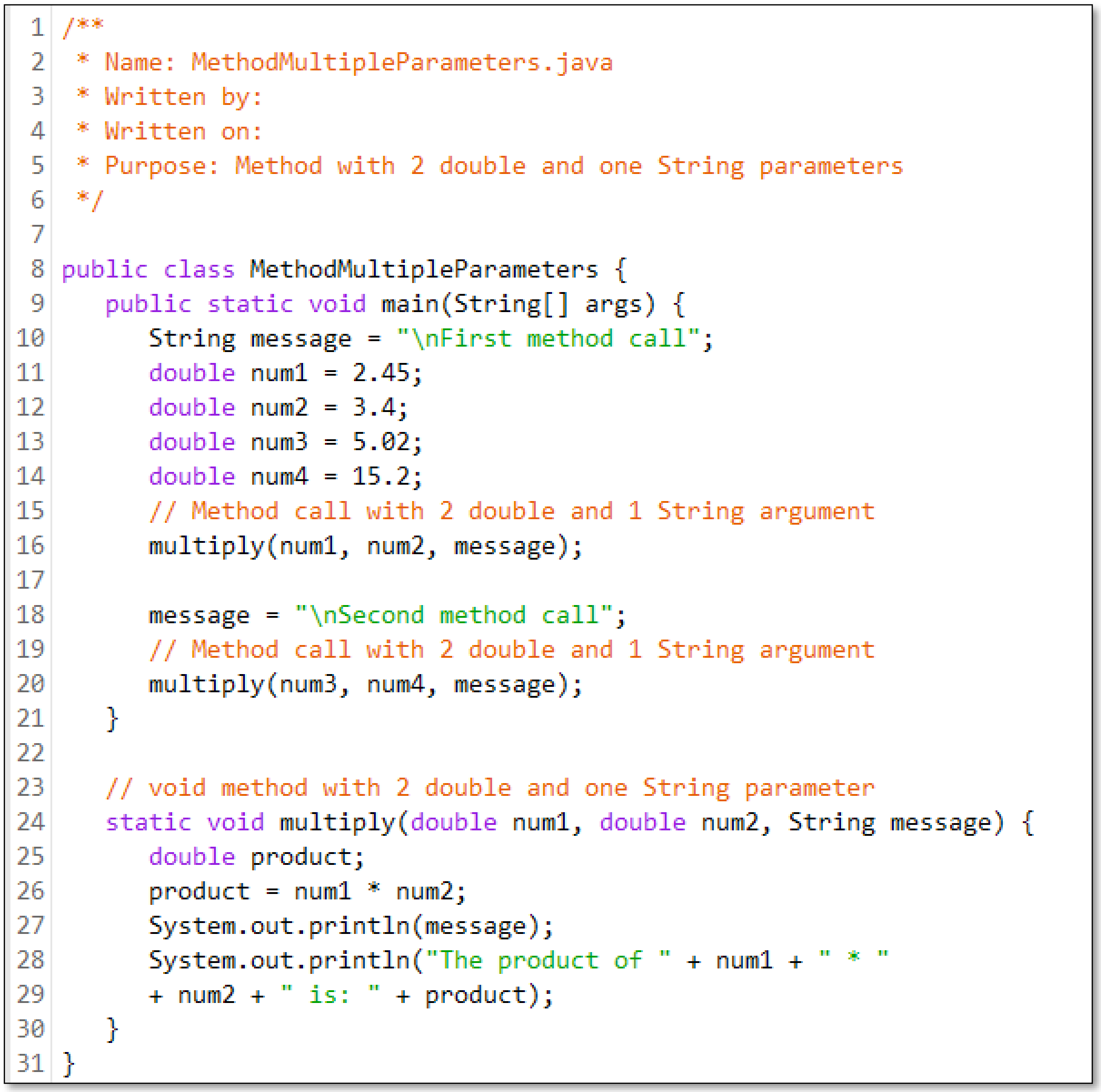


```
The square root of 4.0 is: 2.0
The square root of 8.9 is: 2.9832867780352594
The square root of 112.0 is: 10.583005244258363
```

Tutorial 5.3: Method with Multiple Parameters

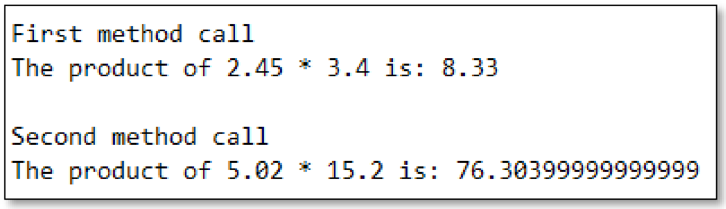
The following program demonstrates a function with 2 datatypes and 3 parameters.

Create a Java program named: **MethodMultipleParameters.java**



```java
/**
 * Name: MethodMultipleParameters.java
 * Written by:
 * Written on:
 * Purpose: Method with 2 double and one String parameters
 */

public class MethodMultipleParameters {
   public static void main(String[] args) {
      String message = "\nFirst method call";
      double num1 = 2.45;
      double num2 = 3.4;
      double num3 = 5.02;
      double num4 = 15.2;
      // Method call with 2 double and 1 String argument
      multiply(num1, num2, message);

      message = "\nSecond method call";
      // Method call with 2 double and 1 String argument
      multiply(num3, num4, message);
   }

   // void method with 2 double and one String parameter
   static void multiply(double num1, double num2, String message) {
      double product;
      product = num1 * num2;
      System.out.println(message);
      System.out.println("The product of " + num1 + " * "
      + num2 + " is: " + product);
   }
}
```

Example run:



```
First method call
The product of 2.45 * 3.4 is: 8.33

Second method call
The product of 5.02 * 15.2 is: 76.30399999999999
```

Tutorial 5.4: Method with Return Value

Methods can do some work and return a value.

Create a Java program named: **MethodReturnValue.java**

```java
/**
 * Name: MethodReturnValue.java
 * Written by:
 * Written on:
 * Purpose: Method with a return value
 */

// Import Math library for ceil method
import java.lang.Math;

public class MethodReturnValue {
   public static void main(String[] args) {
      double num1 = 4.54;
      double num2 = 8.2;
      double ceil2;

      // Method call with 1 argument and return value
      System.out.println("\nPrint method return value directly");
      System.out.println(getCeil(num1));

      System.out.println("Assign return value to variable");
      ceil2 = getCeil(num2);
      System.out.println(ceil2);
   }

   // method with 1 double parameter
   // and 1 double return value
   static double getCeil(double num) {
      double numCeil;
      // ceil rounds a floating point
      // up to the nearest integer value
      numCeil = Math.ceil(num);
      return numCeil;
   }
}
```

Example run:

```
Print method return value directly
5.0
Assign return value to variable
9.0
```

Tutorial 5.5: Input Method

This method takes a prompt string, does error checking and/or input validation, then returns a value.

```java
/**
 * Name: InputMethod.java
 * Written by:
 * Written on:
 * Purpose: Recursive method for valid user input
 */

import java.util.Scanner;

public class InputMethod {
   public static void main(String[] args) {
      double inches;
      // Call method with String prompt parameter
      inches = getDouble("Enter inches: ");
      System.out.println("Inches: " + inches);
   }

   /*
    * Prompt the user for input, return value
    *
    * @param String prompt the user for the value to be entered
    */
   static double getDouble(String prompt) {
      Scanner input = new Scanner(System.in);
      String lineInput;
      double doubleValue = 0;
      // Print the prompt parameter
      System.out.print(prompt);
      // Get the input line into a string
      lineInput = input.nextLine();
      // Catch any input mismatch exceptions
      try {
         // Cast the string to a double
         doubleValue = Double.parseDouble(lineInput);
         // Return value
         return (doubleValue);
      } catch (Exception e) {
         // return method with recursion until the user enters a number
         return (getDouble("Enter a number: "));
      }
   }
}
```

Example run:

```
Enter inches: w
Enter a number: e
Enter a number: 6
Inches: 6.0
```

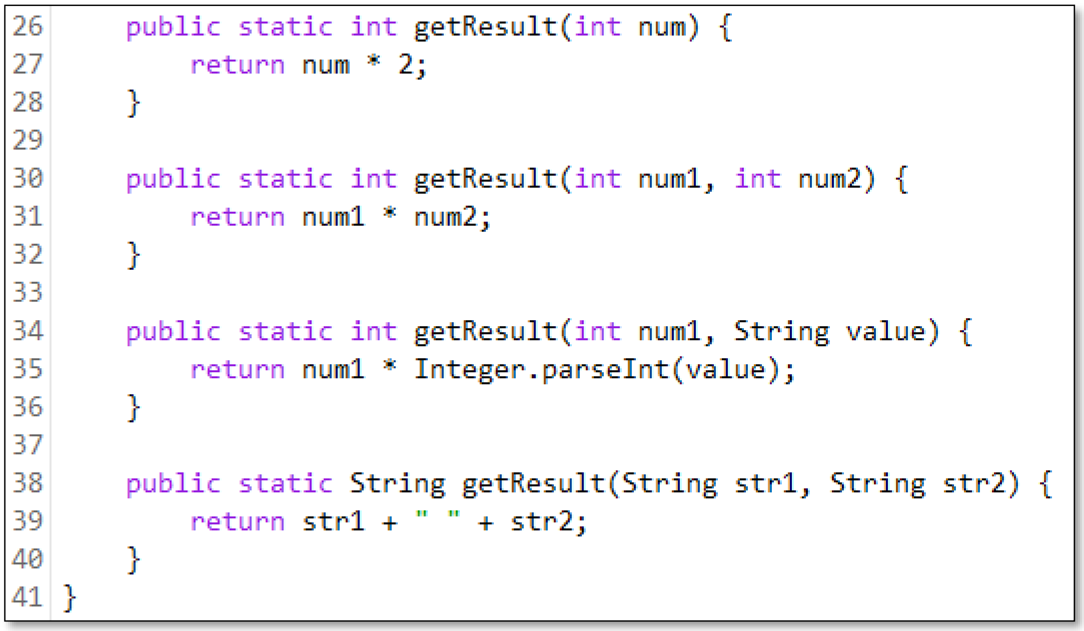
Method Overloading

Method overloading occurs when a method has the same name but is defined multiple times with different signatures. A method signature defines a method's name, its return type, and its parameters.

Tutorial 5.6: Method Overloading

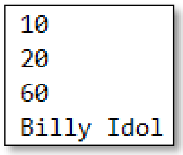
Create a Java file with the name **MethodOverloading.java**

```java
/**
 * Name: MethodOverloading.java
 * Written by:
 * Written on:
 * Purpose: Overload method with different signatures
 */
public class MethodOverloading {
    public static void main(String[] args) {
        int result;
        // Call method with a single int argument
        result = getResult(5);
        System.out.println(result);

        // Call method with two int arguments
        result = getResult(5, 4);
        System.out.println(result);

        // Call method with an int and a string argument
        result = getResult(5, "12");
        System.out.println(result);

        // Call method with two string arguments
        System.out.println(getResult("Billy", "Idol"));
    }
```



```java
    public static int getResult(int num) {
        return num * 2;
    }

    public static int getResult(int num1, int num2) {
        return num1 * num2;
    }

    public static int getResult(int num1, String value) {
        return num1 * Integer.parseInt(value);
    }

    public static String getResult(String str1, String str2) {
        return str1 + " " + str2;
    }
}
```

Example run:



```
10
20
60
Billy Idol
```

Tutorial 5.7: Recursion

A method can call other methods or itself. Recursion is when a method calls itself to accomplish a task.

```java
/**
 * Name: Countdown.java
 * Written by:
 * Written on:
 * Purpose: Recursion demo
 */
public class CountDown {
    public static void main(String[] args) {
        System.out.println("Countdown");
        countDownFrom(10);
        System.out.println();

        System.out.println("Countup");
        countUpTo(0, 10);
        System.out.println();
    }

    static void countDownFrom(int num) {
        if (num >= 0) {
            System.out.print(num + " ");
            // Resursive call
            countDownFrom(num - 1);
        }
    }

    static void countUpTo(int from, int to) {
        if (from <= to) {
            System.out.print(from + " ");
            // Resursive call
            countUpTo(from + 1, to);
        }
    }
}
```

Example run:

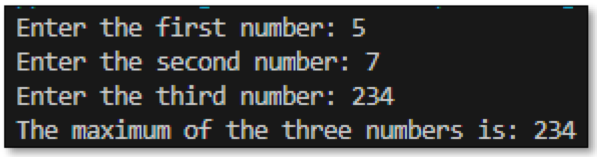
```
Countdown
10 9 8 7 6 5 4 3 2 1 0
Countup
0 1 2 3 4 5 6 7 8 9 10
```

Assignment 1: Find Maximum of Three Numbers

Requirement: Create a method to find the maximum of three numbers.

1. Write a Java program with a method **findMax** that takes three integer parameters and returns the maximum of the three.
2. In the **main** method, ask the user to input three integers.
3. Call the **findMax** method to get the maximum value.
4. Print the result.

Example run:



```
Enter the first number: 5
Enter the second number: 7
Enter the third number: 234
The maximum of the three numbers is: 234
```

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the programs.
3. Submit in Blackboard.