Using Methods

Pieter Joubert

September 27, 2023

Using Methods

Table of Contents

Section 1

Method Construction

Methods

- A method is a *sub-module* of code within our program.
- It's a technique we can use to split up our code into smaller and more manageable chunks.
- Each method generally performs a single *task*.
- We *name* each method and call each method in code using this name.

```java
        public static void main(String args[]) {
            message();
        }

        private static void message() {
            System.out.println("This is a message");
        }
```

Advantages of Methods

- Methods are easily reusable.
- Methods help make it clear what we are trying to do.

```
calculateVAT();
// vs
value = 100.0 * 0.15;
```

- Methods help us to organise our programs, e.g. keeping our main() method short and easy to follow.

Method Parts

- All methods contain a *method header* and a *method body*.
- The *method header* provides information about how other methods can interact with it. It is sometimes also know as the *method signature*.
- The *method body* contains statements that carry out the work of the method.
- This is all the code between the opening and closes curly braces.
- This is also known as the method's *implementation*.

```java

    private static void message()   // header
    {                               // body
        System.out.println("Hi!");  // body
    }                               // body
```

Note that we moved the opening curly brace to a new line to show the parts of the method.

Method Header

```
<optional access specifier> <optional static modifier
    > <return type> <identifer>(<parameters>)

// Examples

public static void message()
private double calculateTip(double price)
protected int addTwoNumbers(int number1, int number2)
```

Section 2

Parameters

Parameters

- Methods allow us to reuse a small chunk of code.
- Very often we want to change some aspect of this code.
- We can still use the same method, but we can pass through a *parameter* that will change the how the method executes.
- We define these *parameters* in the method signature, between the parentheses.

Parameters - Example

```
// no parameters
private static void message() {
        System.out.println("Hello");
}
// one parameter
private static void message(String name) {
        System.out.println("Hello: " + name);
}
// two parameters of different types
private static void message(String name, int age) {
        System.out.println("Hello: " + name);
        System.out.println("You are " + age + " years
    old");
}
```

Arguments

- While parameters are the data items received by the method, *arguments* are data items used to call the method.
- We can also call a Java program with arguments on the commandline, which we can access through the *Stringargs*[] parameter in the main method.
- Parameters and Arguments must match in terms of number and type.

```
      public static void main (String args[]) {
       message(/* ARGUMENTS */);
      }
      private static void message(/* PARAMETERS */) {
  }
```

Section 3

Returning Values

Ending Methods

- Methods can end when one of the following occur:
  - Method completes all of its statements
  - Method throws an exception
  - Method reaches a return statement

Returing Values

- We can return values from a method using the *return* statement.
- We usually return a result based on the code that was executed in the method, e.g. the results of a calculation.
- The return type can be any valid type used in Java.
- Remember the method header must now also include the return *type*.

```java
public static int addTwoNumbers(int num1, int num2) {
    int answer = num1 + num2;
    return answer;
}
```

Section 4

Blocks and Scope

Blocks of code

- Any code between a pir of curly braces.
- For example: all the code inside a class, or all the code inside a method.

```
public class Block { //Block 1 start
    public static void main (String args[]) { //Block 2 start
     message();
    } //Block 2 end

    private static void message() {  //Block 3 start
        System.out.println(”Hello”);
    } //Blok 3 end
} //Block 1 end
```

Scope

- Scope defines whether a variable is valid to be used within the current block.
- A variable comes into scope after it is declared.
- A variable goes out of scope at the end of the block in which it was declared.

```java
public class Scope {

    public static void main(String args[]) {
        String name = "Pieter Joubert";
        changeName();
        System.out.println(name);
    }

    public static void changeName() {
        String name = "John Wick";
    }
}
```

Section 5

Method Overloading

Overloading

- Allows you to use one identifier to execute diverse tasks
- Writing multiple methods in the same scope with the same name but different parameter lists
- Lists must have different numbers of parameters
- Lists must have parameter data types in different orders
- Multiple methods share a name
- Compiler understands which to use based on arguments in the method call (i.e. the method signature is different)

Overloading - Example

```
public class Overloading {
    public static void  main (String args[]) {
          System.out.println(addTwoNumbers(0.5, 1.0));
          System.out.println(addTwoNumbers(5,5));
    }

    public static int addTwoNumbers(int num1, int num2)
      {
     return num1 + num2;
    }

    public static double addTwoNumbers(double num1,
     double num2) {
     return num1 + num2;
    }
}
```

Section 6

Lecture summary

Lecture summary

- Method Construction
- Parameters
- Returning Values
- Blocks and Scope
- Method Overloading

Questions

Thank you! Questions?