Java Language Introduction to Methods

CONTENTS

- **Objectives**
  - Understand the authoring and subsequent calling of methods
  - Appreciate how supplied arguments match defined parameters

- **Contents**
  - Syntax of a method
  - Passing arguments – ‘by value’ using ‘positional’ notation
  - Returning a value from a method
  - Introducing class library methods for console interaction
  - Method overloading
  - Formatting strings in Java

- **Hands on Labs**
  - Method authoring and calling

Defining methods

- **You may have heard of Function, Subroutine, Procedure**
- **In OO languages only the word 'method' is used**

Zero or more Parameters

Signature line

Code

```
public void doStuff( int num )
{
   ..;// statements that 'do' stuff
}  // end of method
```

Methods are only defined inside the body { } of a class

```
class Program {
   public static void main(...)  { statements... }
   public static int  other(...) { statements... }
}
// but not here !!
```

Example of a method with n**o parameter and no returned value**

```
public class Program {
    public static void main(String[] args) {
        updateAllSalaries();
    }

    private static void updateAllSalaries() {
        // Code to update all salaries
        System.out.println("Salaries updated.");

    }
}
```

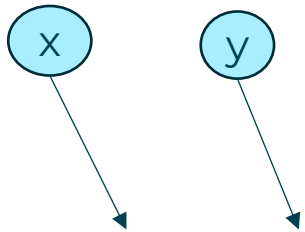
The caller just calls the method and lets it do its work!

Example of a void method with two parameters

```
public class Program {
    public static void main(String[] args) {
        int x = 1, y = 2;
        add(x, y);

        add(3, 7);
    }

    private static void add(int a, int b) {
        print(a + b);
    }
}
```



Example of method returning a value

```
public class Program {
    public static void main(String[] args) {
        int x = 1, y = 2;

        int result = getSum(x, y);

        print("Sum is = ", result);
    }

    private static int getSum(int a, int b) {
        return a + b;
    }
}
```

A method can only return one thing
The caller decides what to do with the returned value

The order of parameters matter

```
public class Program {

    public static void main(String[] args) {
        double result1 = getTax(2000, 21);
        print(result1);

        double result2 = getTax(21, 2000);
        print(result2);
    }


    private static double getTax(double salary, int age) {
        return salary * 0.25;
    }

}
```

Prints **500**

Prints **5.25**

Method overloading

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

        double result1 = getTax(2000);
        print(result1);

        double result2 = getTax(2000, 0.4);
        print(result2);
    }

    private static double getTax(double salary) {
         return salary * 0.25;
    }

    private static double getTax(double salary, double rate) {
        return salary * rate;
    }
}
```

Prints **500**

Prints **800**

Same method name different parameters

Calling a method in another class

- **You don't have to place all your code where the main() method is!**

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

        Tax tax = new Tax();

        double result1 = tax.getTax(2000);
        print(result1);
    }
}
```

Create an object of type Tax

Use the object (tax) to call its method

```
public class Tax{
    public double getTax(double salary) {
        return salary * 0.25;
    }
}
```

Define the class

Define the methods as public so it's visible to other classes

POWERING POTENTIAL

Introducing a few Java library Methods

Introducing some Java library methods

- **Use the `java.util.Scanner` class to input a value**

```
Scanner s = new Scanner(System.in);

System.out.println("What is your name?");
String name = s.nextLine();

System.out.println("What is your age?");
int age = s.nextInt();

System.out.println( "Hi " + name + "\n next year you'll be " + (age + 1));
```

Printing formatted output

`System.out.println(..)` **is hugely overloaded, already seen..**

- Can build a 'String' via concatenation (using +). This is onerous and error prone (spacing)

```
int age = 21;
String name = "Bob";
System.out.println("Hi " + name + ", next year you will be " + age + 1);
```

Hi Bob, next year you will be 211

- Best use `printf()` and String.`format()` for formatting

```
System.out.printf("Hi %s, next year you will be %d", name, age + 1);
```

Format string

More on printf()

- **It does not 'throw a linefeed'. Put a "\n" inside the format string**

```
System.out.printf("%s is %d\n", name, age);
```

- A format string contains plain text as well as format specifiers,.
  - Dictate how the other parameters get formatted
  - They begin with % and end with a converter character.
    - *converter* indicates the type of argument to be formatted.

- In between % and the *converter* you can have optional flags and specifiers.

- See examples on the next few pages

Examples - printf()

- **%s** used to represent a String

```
String word = "wibble";
System.out.printf("My favourite word is %s\n", word);
```

%d	for an integer value
%8d	len 8, right justified
%08d	with leading zeros
%+8d	include sign +/-
%,8d	thousand separator

```
int num = 123456;
System.out.printf("%d\n",     num);  // --> "123456"
System.out.printf("%d8\n",    num);  // --> "  123456"
System.out.printf("%08d\n",   num);  // --> "00123456"
System.out.printf("%+8d\n",   num);  // --> " +123456"
System.out.printf("%,8d\n",   num);  // --> " 123,456"
System.out.printf("%+,8d\n",  num);  // --> "+123,456"
```

Examples - printf()

%f	for a floating point value
%.3f	3 decimal places (rounded)
%10.3f	right justified to length 8, 3 decimals
%-10.3f	left justified to length 10, 3 decimals

```
double pi = Math.PI;
System.out.printf("%f\n", pi);     // "3.141593"

System.out.printf("%.3f\n", pi);   // "3.142"

System.out.printf("%8.3f\n", pi); // "   3.142"


System.out.printf("%-10.3f\n", pi);  // "3.142"
```

```
String name = "Fred";
int age = 23;
double aveExamGrade = 3.57;

System.out.printf("%s who is %3d achieved %.1f\n", name, age, aveExamGrade);
```

```
Fred who is  23 achieved 3.6
```

String.format(String, .., .., ..)

Returns a formatted string

- “Format string” is 1st parameter with built-in format specifiers
- Useful when a method has to return a formatted string or when a String needs to be built for more than one time use

```
public static void main(String[] args) {
  String data = getData();
  System.out.println(data);
}
```

```
public static String getData() {
  String name = “Bob";
  int age = 28;
  return String.format("%s is %d", name, age )
}
```

Review

- Methods are the ‘function’ members of a class
- Signature line plus code block
  - Any parameters defined are mandatory parameters
- Return type of ‘void’ means ‘produces no value’
  - Method call cannot be used in an assignment
- ‘Arguments’ are passed to methods with ‘parameters’
  - Passed by value & using positional notation
- Parameters of a method are effectively local variables
- Signatures can be overloaded
  - Occurs widely in Java
- Methods for formatting strings

Hands On Labs

- Part 1 - ‘Authoring a helper method’
- Part 2 – ‘Performing data conversions’
- Part 3 – ‘Weight Conversions’