

# Methods, Arguments and Parameters

Structuring your Programs for Code Reuse

Alwin Tareen

# Method Types

## What is a Method?

- ▶ A **method** is simply a block of code that is assigned a name.
- ▶ In Java, there are two types of methods that you can write. These methods are identified by the type of task that they perform.

## void methods

These are methods that do not give you any data back. They have no return statement.

## Non-void methods

These are methods that give you some kind of data back. They must have a return statement.

# void Methods

- ▶ Consider the following void method called printName()
- ▶ This method's **return type** is declared void.
- ▶ This method performs a task, but it does not give any data back.

```
public static void printName()  
{  
    System.out.println("Bob Smith");  
}
```

## Calling void methods

- ▶ To execute or call a void method, you write the method name along with open and closed parentheses.
- ▶ Note that a void method is called as a single statement on a line by itself.

## Methods Called by Other Methods

```
public static void printFirstName()  
{  
    System.out.println("Bob");  
}  
  
public static void printLastName()  
{  
    System.out.println("Smith");  
}  
  
public static void printName()  
{  
    printFirstName();  
    printLastName();  
}
```

# Non-void Methods

- ▶ Non-void methods differ from void methods in that they give data back. What they give back depends upon their return type.

```
public static int calcSum()  
{  
    int first = 5;  
    int second = 8;  
    return first + second;  
}
```

- ▶ The return type for the method `calcSum()` is `int`.
- ▶ The keyword `return` is **required** in all non-void methods.
- ▶ Since the return type is `int`, the method must return an integer.

# Non-void Methods

## Type Matching

- ▶ A non-void method's return type can be any valid data type, as long as the code after the return keyword matches the return data type.

## Examples using return

- ▶ A method can return a literal value:

```
return 500;  
return 98.6;  
return "Bob Smith";  
return true;
```

# Non-void Methods

## Examples using return

- ▶ A method can return the contents of a variable:

```
return average;  
return name;
```

- ▶ A method can return the result of a mathematical expression:

```
return 15 * 7 + 20;  
return (first + second) / 2;
```

- ▶ A method can return the result of a boolean expression:

```
return num != 0 && num < 100;  
return name.equals("Harry Potter")
```

# Calling Non-void Methods

## Handling the returned value

- ▶ Non-void methods must be called from within other Java statements.
- ▶ Since a non-void method always returns a value, then this value has to be stored in a variable, printed, or passed to a Java control structure.

## Examples of method calls

- ▶ A method call can be in an assignment statement:

```
int num = sum();  
double average = calcAverage();  
String name = getName();
```



# Calling Non-void Methods

## Examples of method calls

- ▶ A method call can be within a println statement:

```
System.out.println("Sum = " + sum());  
System.out.println("Name = " + getName());
```

- ▶ A method call can be inside a Java control structure, such as an if statement, or a looping structure:

```
if (sum() > 500)  
while (!getName().equals("Stop"))
```

- ▶ A method call can even be on the return line of another method:

```
return sum() / items;
```

# Arguments

## What is an argument?

- ▶ An argument is a value that is passed to a method, so that the method can use that value in its code block.
- ▶ An argument is found inside the parentheses that follow a method name.

```
int num = squareRoot(16);
```

- ▶ The value 16 is the argument for the method `squareRoot`.
- ▶ The method calculates the square root of this value, and assigns it to the variable `num`.

# Arguments

## Examples of arguments

- ▶ An argument can be a literal value:

```
calculateArea(81);  
place.equals("exit");
```

- ▶ An argument can be a variable:

```
calculateArea(num);  
place.equals(location);
```

- ▶ An argument can be a mathematical expression:

```
calculateArea(20 + 50 * 3.14);
```

# Arguments

## Examples of arguments

- ▶ An argument can be a call to another method:

```
calculateArea(findRadius());
```

- ▶ A method can have 0, 1, or more arguments. Multiple arguments are separated by commas:

```
calculateVolume(10, 2, 8);
```

# Parameters

## Writing a Method with Parameters

- ▶ To add parameters to your own method definitions, you simply declare one or more variables in the parentheses that follow the method name.
- ▶ These variable declarations need to include a data type and an identifier name.

```
public double volume(int r, int h, double pi)
{
    return pi * r * r * h;
}
```

# Parameters

## Parameter declaration

- ▶ The following method `average` declares three parameters all of type `int`, named `one`, `two`, and `three`.

```
public double average(int one, int two, int three)
{
    return (one + two + three) / 3.0;
}
```

- ▶ These parameters are used to store the values of the three integers that are passed to the method.
- ▶ The method uses these parameters to calculate the average of the three integers, and returns the answer.

# Parameters

## Parameter scope

- ▶ The scope of a parameter is limited to the method that it is declared in.
- ▶ The parameters `one` and `two` have method scope. This means that they are local variables for the method `minimum`. They do not exist outside of this method.

```
public int minimum(int one, int two)
{
    if (one < two)
        return one;
    else
        return two;
}
```