**Object Oriented Development using Java**

OOD Week 1 – Module 5

Tutorial

Methods

© FDM Group Ltd 2020. All Rights Reserved.

Any unauthorised reproduction or distribution in part  
or in whole will constitute an infringement of copyright.

# What does this walkthrough cover?

This walkthrough will discuss methods

# How long will the walkthrough take to complete?

2 hours

# What do you need?

In order to complete this tutorial exercise you will need:

* Java Development Kit 1.8 or above
* Apache Maven
* Eclipse IDE Kepler or above
* Any extra software should go here (Git, SQL, etc.)

# What does this walkthrough cover?

* Syntax
* Example methods
* Calling methods
* Using Varargs
* Return Values

## Syntax

Methods are useful when you want to execute a piece of code and reuse it in several places.

The general syntax for a method is:

<return type> method\_name(arguments){

}

The arguments you pass the method are optional.

The return value is also optional. if you do not return a value from the method then the return value must be void.

# Example Methods

An example of a method is shown below:

**public** **void** display() {

}

This method returns void (there is no return value) and there are no arguments.

During this section we will also mark methods as static. For the time being we must do this but later in the course you will see how to use a better technique.

Here are some other methods:

// returns void and takes an argument

**public** **void** display(String value)

{

}

// returns a String but has no arguments

**public** String display()

{

}

// returns a String but has one argument

**public** String display(**int** value)

{

}

You can see there are many variations. You can only have one return type but you can have ANY number of arguments.

# Calling Methods

Let's call some of these methods and make them work in a program. We will add the static modifier (for the time being)

**public** **class** Runner {

**public** **static** **void** display()

{

System.***out***.println("display called");

}

**public** **static** **void** main(String args[]){

*display*(); // call the display method

}

}

Notice how we call the method, we have the display method and we call it with no arguments. When you call a method you must make sure the argument list matches. In this case it is, the method expects no arguments to be passed to it and we are calling it with no arguments.

Let's change the method so that it has an argument:

**public** **class** Runner {

**public** **static** **void** display(String stringValue)

{

System.***out***.println("display called");

}

**public** **static** **void** main(String args[]){

*display*("some string"); // call the display method

}

}

We have called the display method with a String. We must pass it a String otherwise it will not compile. Try removing the argument and see what happens.

A method can take ANY number of arguments. When you call the method you must make sure you pass the right types of values:

**public** **class** Runner {

//the display method expects these three values in the correct order

**public** **static** **void** display(String stringValue, **int** intValue, **double** doubleValue)

{

System.***out***.println("display called");

}

**public** **static** **void** main(String args[]){

*display*("some string", 0, 0.0);

// call the display method with three arguments, "some string", 0 and 0.0

}

}

When calling the display method we must pass it three values, a String an int and a double. If not, then you will get a compiler error. You must also call them in the order that the method expects. The method expects a String as the first argument, an int as the second and a double as the third argument.

**public** **class** Runner {

//the display method expects these three values in the correct order

**public** **static** **void** display(String stringValue, **int** intValue, **double** doubleValue)

{

System.***out***.println("display called");

}

**public** **static** **void** main(String args[]){

*display*("some string", 0, 0.0);

// call the display method with three arguments, "some string", 0 and 0.0

}

}

# Using vargargs

You can also use varargs to represent any number of arguments of the same type:

**public** **class** Runner {

**public** **static** **void** display (**int**... intValues)

{

System.***out***.println("display called with int arguments: ");

**for**(**int** eachInt: intValues){

System.***out***.println(eachInt);

}

}

**public** **static** **void** main(String args[]){

*display*(); // call the display method

}

}

# Return Values

Let's revert to our method which has no arguments. Let's give it a return value:

**public** **class** Runner {

**public** **static** String display()

{

System.***out***.println("display called");

**return** "a value";

}

**public** **static** **void** main(String args[]){

*display*(); // call the display method

}

}

We have called the method, which returns a value. We have not used the value, but the code works. It is optional, but best, to use the value returned from the method as shown in the example below:

**public** **class** Runner {

**public** **static** String display()

{

System.***out***.println("display called");

**return** "a value";

}

**public** **static** **void** main(String args[]){

String returnValue = *display*(); // call the display method

System.***out***.println("value returned was " + returnValue);

}

}

In this example we will use the value which is displayed.

Here are some tips when writing a method:

1. Decide what your method needs to do then break down the steps to achieve it. If you don't know what you want your method to do then you won’t be able to write a method to achieve it.

2. Decide if it needs to be passed a value (or values)

3. Decide if the method need to return a value

4. Keep your methods small. This way you might be able to reuse the method in several places and save rewriting the code several times. If the method is too large you may find that it is hard to reuse. Using methods can also prevent errors because instead of writing the same code several times in different places and using something incorrectly, you only need to write it once in the method and then call the method whenever you want to execute that code.