**Object Oriented Development using Java**

OOD Week 1 – Module 3

Conditionals

Tutorial

© 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 tutorial cover?

This tutorial will introduce you to ways of allowing your code to make decisions. You’ll cover if statements, conditions and switch/case statements.

# How long will the tutorial take to complete?

30 minutes

# What should you have already completed?

Module 1 (Data types)

# 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

# What does this tutorial cover?

* If statements
* Else & else if statements
* Conditions
* Switch/case statements

# What are conditionals

Conditionals are ways to allow our code to make decisions. In the previous two modules, every line of code in our program has run in sequence from start to finish. To put it another way, each time our programs have run, they’ve done exactly the same thing.

In reality, we want programs to be dynamic, to run differently each time depending on the situations they encounter. We can do this using two kinds of statement: if statements and switch/case statements.

# If statements

Let’s start with a very simple example:

**double** priceOfPhone = 500;

**if** (priceOfPhone > 450) {

System.***out***.println("phone is expensive");

}

If you run this code, you’ll see that it prints the words “phone is expensive”. Let’s now change the value of priceOfPhone to something lower:

**double** priceOfPhone = 200;

**if** (priceOfPhone > 450) {

System.***out***.println("phone is expensive");

}

Run the code again and you’ll see that it doesn’t print anything.

Our print statement runs or doesn’t run based on the condition. In this case it will only run where the condition is true, i.e. the price is above 450.

# Else statements

In our first example, the print statement either runs or it doesn’t depending on whether the condition is true or false. There are many times when this is exactly what we need. But often we might want to choose from two different paths depending on whether the condition is true or false. To do this we can add and ‘else’ statement to our if statement:

**if** (priceOfPhone > 450) {

System.***out***.println("phone is expensive");

} **else** {

System.***out***.println("phone is cheap");

}

If we run this with the value 200 we’ll get the message “phone is cheap”, with the value 500 we’ll get “phone is expensive”.

No matter what the value of priceOfPhone, one of the two statements will run.

# Else if statements

These allow us to add extra conditions into our if statement:

**if** (priceOfPhone < 150) {

System.***out***.println("phone is cheap");

}

**else** **if** (priceOfPhone < 450) {

System.***out***.println("phone is medium priced");

}

**else** {

System.***out***.println("phone is expensive");

}

Try setting the value of priceOfPhone to 100. Which of the three messages do you think will be displayed?

The first and second conditions are both true. The phone costing 100 has a price less than 150 and less than 450. However we’ll only see the “phone is cheap” message.

The if statement terminates after reaching the first true condition. Even if later conditions are also true, they won’t run.

In this case we need to be careful to sequence the conditions correctly. What would be wrong with the code below?

**if** (priceOfPhone < 450) {

System.***out***.println("phone is medium priced");

}

**else** **if** (priceOfPhone < 150) {

System.***out***.println("phone is cheap");

}

**else** {

System.***out***.println("phone is expensive");

}

In this case, we would never see the “phone is cheap” message as the first condition would catch all of the prices which were below 150.

# Different types of condition

## Conditions for primitives

In our previous example, we compared two values using the greater than sign. We could have used any of the following operators to compare two primitive values:

== equals

!= not equals

> greater than

< less than

>= greater than or equals

<= less than or equals

Where a condition tests a boolean variable, there’s no need to use ==.

**boolean** isTrue = **true**;

**if** (isTrue) {

System.***out***.println("value is true");

}

To check if the boolean variable is false, you can simply put an exclamation mark before it:

(! isTrue)

## Conditions for objects

We should avoid using == and != when comparing two object values.

(string1 == string2)

If we try to compare two Strings with identical values using == the condition may return true, but sometimes it may return false. We’ll go into the detail of why this is in Module 13 on Mutable & Immutable Objects.

The same is true for !=

We should compare two Strings using the equals() method:

(string1.equals(string2))

This condition will always return true when the two values are the same.

If we want to check whether two Strings are not equal, we can do this:

(! string1.equals(string2))

The exclamation mark reverses the condition.

What we’ve seen is true for **all** object variables. So for the same reason, if you’re comparing a pair of Integers or a pair of Doubles, you should use the equals() method.

## Combining multiple conditions with && and ||

The ‘and’ operator (&&) allows us to combine two conditions into one. Overall the condition is true both of the two conditions it’s made up of are also true.

Let’s say that we want a user to enter a number between 1 and 10. We could use the code below to check whether the number they entered was valid or invalid:

**int** number = 9;

**if** (number >= 1 && number <= 10) {

System.***out***.println("valid number");

} **else** {

System.***out***.println("invalid number");

}

Another way of combining two conditions into one is by using the ‘or’ operator (||). In this case the overall condition is true if either one or both of the conditions it’s made up of are true. Let’s try the same example, but using ||

**int** number = 9;

**if** (number < 1 || number > 10) {

System.***out***.println("invalid number");

} **else** {

System.***out***.println("valid number");

}

You could even combine 3 or more conditions using both && and ||

In this case you need to be careful and use brackets to make sure that you end up with the right condition overall.

The conditions below contain the same 3 sub conditions, but their results will be different:

((value1 == value2 && value3 == value4) || value5 == value6)

(value1 == value2 && (value3 == value4 || value5 == value6))

If the values of the variables were as follows:

**int** value1 = 1;

**int** value2 = 2;

**int** value3 = 3;

**int** value4 = 4;

**int** value5 = 5;

**int** value6 = 5;

The first condition would be true and the second condition would be false. Look at them carefully and make sure you understand why this is.

# Switch/case statements

A switch/case statement is an alternative to an if statement. It can do exactly the same thing as an if statement, but you’ll see that it can also do more.

**int** code = 2;

**switch** (code) {

**case** 1:

System.***out***.print('a');

**break**;

**case** 2:

System.***out***.print('b');

**break**;

**case** 3:

System.***out***.print('c');

**break**;

**case** 4:

System.***out***.print('d');

**break**;

**default** :

System.***out***.println("Invalid code");

}

Try running this code, you’ll see that it prints out the letter ‘b’. In fact it does exactly the same as the following if statement:

**if** (code == 1) {

System.***out***.print('a');

} **else** **if** (code == 2) {

System.***out***.print('b');

} **else** **if** (code == 3) {

System.***out***.print('c');

} **else** **if** (code == 4) {

System.***out***.print('d');

} **else** {

System.***out***.println("Invalid code");

}

You’ll notice that each of the case conditions in the switch/case statement is followed by a break statement.

Where this is the case the switch/case statement does exactly the same thing as an if statement. However we could remove the break statements. Try running the code below. What do you think it will produce:

**int** code = 2;

**switch** (code) {

**case** 1:

System.***out***.print('a');

**case** 2:

System.***out***.print('b');

**case** 3:

System.***out***.print('c');

**case** 4:

System.***out***.print('d');

**break**;

**default** :

System.***out***.println("Invalid code");

}

In this case the code will print out “bcd”. So as well as getting the print statement for option 2, we’re also getting the print statements for the options below it.

Notice that in this case the final break statement has been left in place to avoid getting the default option (the equivalent of an else).

# A reminder about clean code

You’ll remember that clean code is about making code as easy to read as possible. This is particularly important with conditionals. You’ll have noticed that in all of our examples we’ve used a lot of indentation to make the code readable. You should **always** do this.

In Eclipse you can use the auto-indent feature by pressing the Ctrl+Alt+f keys together. This is so easy to do, that there’s no excuse for not doing it. Note that this will only work if your code doesn’t have any compile errors.