Control Flow Statements

Below statements are used to break up the flow of the execution.

## Decision Making Statements

### If - then

This will inform your program to execute a certain piece of code, only if a particular condition evaluates to true.

Syntax:

If (boolean){ // the "if" clause

…. // the "then" clause:

….

} // curly braces are optional.

### If - then - else condition

By using this, we are able to specify the alternate solution when an **“if”** clause evaluates to false.

Syntax:

if (boolean) {

} else {

}

### Nested if - else conditions

We are going to use this type of statements to perform multiple condition checks based on requirement.

if (PrimaryCondition) {

if (SecondaryCondition) {

} else {

}

} else {

}

IfElseSample.java

Above program will demonstrate the simple if-else statements.

### Switch

It provides an easy way to select the suitable choice from the list of available option based on the switch condition.

Syntax:

Switch(expression){

Case value1:

Statement1;

Break;

Case value2:

Statement2;

Break;

default:

StatementN;

Break;

}

Allowed values for expression are:

Version 1.4: byte, short, int , char

Version 1.5: Byte, Short, Integer, Character, Enum and previous version types

Version 1.7: String and previous version types

A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression,then executes all statements that follow the matching case label.

#### Break Statement

Each **break** statement terminates the enclosing **switch** statement.

If we didn’t specify the break statement inside the switch, all statements after the matching case label are executed in sequence, regardless of the expression of subsequent **case** labels, until a **break** statement is encountered. We call it as **fall through** inside the switch block.

SwitchCaseSample.java

The above program demonstrates the normal switch case and the fall through scenarios.

**Note**: It is legal to have multiple case labels for a statement.

Switch(expression){

Case value1: Case value2:

Case value4: Case value9:

Statement1;

Break;

}

## Looping statements

### While

It continuously executes the block of statements while a particular condition is true.

Syntax:

while (expression) { // expression must be boolean type

statement(s)

}

If the expression evaluates to true, the while statement executes the statement(s) in the while block.

We prefer the while loop, if we don’t know the number of iterations in advance.

Note: curly braces are optional. Without this only one statement is allowed to take and it should not be declarative statement.

WhileLoopSample.java

In the above program, we are using the while statement to print the values from 1 to 10.

**Infinite while loop**

while (true){

// your code goes here

}

### Do – While

The difference between **do-while** and **while** is that **do-while** evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

DoWhileSample.java

In the above program, we are using the do- while statement to print the values from 1 to 10.

### For

For loop gives us the flexibility to iterate over a range of values.

Syntax:

for (initialization; termination; increment or decrement) {

statement(s)

}

#### Key Points:

The initialization expression initializes the loop. It is executed once, as the loop begins.

When the termination expression evaluates to false, the loop terminates.

The increment/decrement expression is invoked **after** each iteration of the loop.

**Infinite for loop**

for ( ; ; ) {

// your code goes here

}

ForLoopSample.java

In the above program, check for statement of scenario 1 and 2.

### For – Each

It is introduced in the 1.5version. This loop is specially designed to iterate the values of an array/collection.

ForLoopSample.java

In the above program, check for statement of scenario 3.

**Disadvantage**: It hides the iterator object. So we can’t call the remove() method.