**2.7 Flow Control – Decision :-**

The statements inside your source files are generally executed from top to bottom, in the order that they appear. *Control flow statements*, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to *conditionally* execute particular blocks of code. This section describes the decision-making statements (if-then, if-then-else, switch), the looping statements (for, while, do-while), and the branching statements (break, continue, return) supported by the Java programming language.

* while
* for
* do while
* break
* continue
* return
* if
* if….else
* Nesting if……else
* Ladder if……else
* switch case
* The conditional (? :) Operator

**Flow Control**

Programs are much more than the simple sequential execution of statements.  We often need to excute a section of code repeatedly, perhaps with a change of some of the program's state each time.  We also commonly need to execute different sections of code depending upon whether some condition is true of false.  This lesson will discuss these two types or flow control that are commonly seen in programs: Looping , Branching and decision making .

**2.7.1 Java Looping Statements :-**

**a) The while Loop:**

A while loop is a control structure that allows you to repeat a task a certain number of times.

**Syntax:**

The syntax of a while loop is:

while(Boolean\_expression)

{

//Statements

}

When executing, if the *boolean\_expression* result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

**Example:**

public class Test {

public static void main(String args[]) {

int x = 10;

while( x < 20 ) {

System.out.print("value of x : " + x );

x++;

System.out.print("\n");

}

}

}

This would produce following result:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

**b) The do...while Loop:**

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

**Syntax:**

The syntax of a do...while loop is:

do

{

//Statements

}while(Boolean\_expression);

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

**Example:**

public class Test {

public static void main(String args[]){

int x = 10;

do{

System.out.print("value of x : " + x );

x++;

System.out.print("\n");

}while( x < 20 );

}

}

This would produce following result:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

**3) The for Loop:**

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

**Syntax:**

The syntax of a for loop is:

for(initialization; Boolean\_expression; update)

{

//Statements

}

Here is the flow of control in a for loop:

* The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
* Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
* After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.
* The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step,then Boolean expression). After the Boolean expression is false, the for loop terminates.

**Example:**

public class Test {

public static void main(String args[]) {

for(int x = 10; x < 20; x = x+1) {

System.out.print("value of x : " + x );

System.out.print("\n");

}

}

}

This would produce following result:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

## 2.7. 2 Java Branching Statements :-

## a) The break Keyword:

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

## Syntax:

The syntax of a break is a single statement inside any loop:

break;

## Example:

public class Test {

public static void main(String args[]) {

int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) {

if( x == 30 ) {

break;

}

System.out.print( x );

System.out.print("\n");

}

}

}

This would produce following result:

10

20

## b) The continue Keyword:

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

* In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
* In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

## Syntax:

The syntax of a continue is a single statement inside any loop:

continue;

## Example:

public class Test {

public static void main(String args[]) {

int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) {

if( x == 30 ) {

continue;

}

System.out.print( x );

System.out.print("\n");

}

}

}

This would produce following result:

10

20

40

50

**C) The return Statement**

The last of the branching statements is the return statement. The return statement exits from the current method, and control flow returns to where the method was invoked. The return statement has two forms: one that returns a value, and one that doesn't. To return a value, simply put the value (or an expression that calculates the value) after the return keyword.

return ++count;

The data type of the returned value must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value.

return;

The [Classes and Objects](http://docs.oracle.com/javase/tutorial/java/javaOO/methods.html) lesson will cover everything you need to know about writing methods.

**2.7.3 Java Decision Making Statements :-**

In Java theor are three type of decision statement are involoves:

* **if** statement
* **switch** statement
* **Conditonal** operator statement

First we learn the IF Statement

|  |
| --- |
| **DECISION MAKING WITH IF STATEMENT** |

* This statement is used to control the flow of execution of statements.
* It is basically two way decision statement and used in combination with an expression.
* It allows the computer to evalute the expression first , if the first condition or expression is true then first part is executed otherwise the second part will be executed.

The if statement may be implemented in different forms depending on the complexities of condition to be tested.

**a) Simple if statement**

Syntax of simple if statement

|  |
| --- |
| **if**(test expression) { statement-block; } statement-x; |

In that 'statement-block' may be single statement or group of statementsIf the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the sttaement-x.

NOTE :

When the condition is true both the statement-block and the statement-x are executed in sequence.

* 1. **IF.......ELSE STATEMENT**

It is an extension of simple if statement .  
The syntax of if.....else statement

|  |
| --- |
| **if**(test expression) { True-block statements(s) } **else** { false-block statement(s) } statement-x |

In that if the test expression is true,then true-block statements executed otherwise the false-block statement executed.

**C) NESTING OF IF.....ELSE STATEMENTS**

When we want to test multiple decision , in that we use more than one if..else statement in nested form as follows:

|  |
| --- |
| **if**(test condition1) { if(test condition2) {statement-1; } else { statement-2;} } else { statement-3; }stement-x; } |

1. **THE ELSE IF LADDER**

There is another way of putting ifs together when multipath decision are involved.  
The syntax of else if ladder as follows:

|  |
| --- |
| if (condition1) statement-1 else if(condition 2) statement-3; else if(condition n) statement-n; else default-statement; } |

|  |
| --- |
| 1. THE SWITCH STATEMENT |

* The switch statement test the value of given variable (or expression) againest a list of case values
* and when match is found , a block of statement associated with that case is executed.  
  The syntax of switch statement are as followes:

|  |
| --- |
| switch(expression) { case value-1:       block-1       break; { case value-2:       block-2       break; ......... ......... default: default-block break; } statement-x; |

In that **expression** is an integer expression or character. **value-1, value-2** ...are constants or constants expressionand in that case labels each of these values should be unique within switch statement.block-1,block-2...are statement list .There is no need to put braces around these blocks but the important that case labels end with colon(:).The **break** statement causes an exit from the switch statement.The **default** is an optional case,If the condition is not match with any case label then default case will be executed.

|  |
| --- |
| 1. THE CONDITIONAL (?:) OPERATOR |

* These operator are used for taking two-way decision.
* This opertaor is a combination of ? and : and takes three operands.

The syntax of conditional operator as follows:

|  |
| --- |
| **conditional expression? expression1: expression2** |

The conditional expression is evaluated first.If the resultc is true , expression1 is evaluted and is return as the value of the conditional expression, Otherwise expression2 is evaluted and its value is returned.  
For Example: if(x<0)  
flag=0;  
else  
flag=1;  
can be written as   
flag=(x<0)?0:1;