Java **flow control** Statements

There are three types of flow control statements in java

1) Selection Statements

2) Iteration statements

3) Transfer statements.

When we need to execute a set of statements **based on a condition** then we need to use **control flow statements**. For example, if a number is greater than zero then we want to print “Positive Number” but if it is less than zero then we want to print “Negative Number”. In this case we have two print statements in the program, but only one print statement executes at a time based on the input value.

a) if statement
b) nested if statement
c) if-else statement
d) if-else-if statement.

If statement

If statement consists a condition, followed by statement or a set of statements as shown below:

```
if(condition){
  Statement(s);
}
```

The statements get executed only when the **given condition is true**. If the condition is false then the statements inside if statement body are completely ignored.

Nested if statement in Java

When there is an if statement inside another if statement then it is called the **nested if statement**.
The structure of nested if looks like this:

```
if(condition_1) {
   Statement1(s);
```

```
   if(condition_2) {
      Statement2(s);
   }
}
```

Statement1 would execute if the condition_1 is true. Statement2 would only execute if both the conditions (condition_1 and condition_2) are true.

Example of Nested if statement

```
public class NestedIfExample {

   public static void main(String args[]){
        int num=70;
        if( num < 100 ){
           System.out.println("number is less than 100");
           if(num > 50){
              System.out.println("number is greater than 50");
           }
        }
   }
}
```

If else statement in Java

This is how an if-else statement looks:

```
if(condition) {
   Statement(s);
}
else {
   Statement(s);
}
```

The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false.

if-else-if Statement

if-else-if statement is used when we need to check multiple conditions. In this statement we have only one “if” and one “else”, however we can have multiple “else if”. It is also known as **if else if ladder**. This is how it looks:

```
if(condition_1) {
   /*if condition_1 is true execute this*/
   statement(s);
}
else if(condition_2) {
   /* execute this if condition_1 is not met and
    * condition_2 is met
    */
```

```
   statement(s);
}
else if(condition_3) {
   /* execute this if condition_1 & condition_2 are
    * not met and condition_3 is met
    */
   statement(s);
}
.
.
.
else {
   /* if none of the condition is true
    * then these statements gets executed
    */
   statement(s);
}
```

Note: The most important point to note here is that in if-else-if statement, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside “else” gets executed.

Example of if-else-if

```
public class IfElseIfExample {

   public static void main(String args[]){
         int num=1234;
         if(num <100 && num>=1) {
           System.out.println("Its a two digit number");
         }
         else if(num <1000 && num>=100) {
           System.out.println("Its a three digit number");
         }
         else if(num <10000 && num>=1000) {
           System.out.println("Its a four digit number");
         }
         else if(num <100000 && num>=10000) {
           System.out.println("Its a five digit number");
         }
         else {
           System.out.println("number is not between 1 & 99999");

         }
   }
}
```

Switch Case statement in Java with example

Switch case statement is used when we have number of options (or choices) and we may need to perform a different task for each choice.

The syntax of Switch case statement looks like this –

```
switch (variable or an integer expression)
{
     case constant:
     //Java code
     ;
     case constant:
     //Java code
     ;
     default:
     //Java code
     ;
}
```

Switch Case statement is mostly used with break statement even though it is optional. We will first see an example without break statement and then we will discuss switch case with break

A Simple Switch Case Example

```
public class SwitchCaseExample1 {

   public static void main(String args[]){
     int num=2;
     switch(num+2)
     {
        case 1:
          System.out.println("Case1: Value is: "+num);
        case 2:
          System.out.println("Case2: Value is: "+num);
        case 3:
          System.out.println("Case3: Value is: "+num);
        default:
          System.out.println("Default: Value is: "+num);
      }
   }
}
```

Break statement in Switch Case

Break statement is optional in switch case but you would use it almost every time you deal with switch case. Before we discuss about break statement, Let's have a look at the example below where I am not using the break statement:

```
public class SwitchCaseExample2 {

   public static void main(String args[]){
      int i=2;
      switch(i)
```

```
      {
         case 1:
           System.out.println("Case1 ");
         case 2:
           System.out.println("Case2 ");
         case 3:
           System.out.println("Case3 ");
         case 4:
          System.out.println("Case4 ");
         default:
           System.out.println("Default ");
      }
   }
}
```

Few points about Switch Case

1) Case doesn't always need to have order 1, 2, 3 and so on. It can have any integer value after case keyword. Also, case doesn't need to be in an ascending order always, you can specify them in any order based on the requirement.

2) You can also use characters in switch case. for example –

```
public class SwitchCaseExample2 {

   public static void main(String args[]){
      char ch='b';
      switch(ch)
      {
         case 'd':
           System.out.println("Case1 ");
           break;
         case 'b':
           System.out.println("Case2 ");
           break;
         case 'x':
           System.out.println("Case3 ");
           break;
         case 'y':
          System.out.println("Case4 ");
          break;
         default:
           System.out.println("Default ");
      }
   }
}
```

For loop in Java with example

Loops are used to execute a set of statements repeatedly until a particular condition is satisfied. In Java we have three types of basic loops: for, while and do-while. In this tutorial we will learn how to use "**for loop**" in Java.

Syntax of for loop:

```
for(initialization; condition ; increment/decrement)
{
   statement(s);
}
```

Flow of Execution of the for Loop

As a program executes, the interpreter always keeps track of which statement is about to be executed. We call this the control flow, or the flow of execution of the program.

First step: In for loop, initialization happens first and only one time, which means that the initialization part of for loop only executes once.

Second step: Condition in for loop is evaluated on each iteration, if the condition is true then the statements inside for loop body gets executed. Once the condition returns false, the statements in for loop does not execute and the control gets transferred to the next statement in the program after for loop.

Third step: After every execution of for loop's body, the increment/decrement part of for loop executes that updates the **loop counter**.

Fourth step: After third step, the control jumps to second step and condition is re-evaluated.

Example of Simple For loop

```
class ForLoopExample {
    public static void main(String args[]){
         for(int i=10; i>1; i--){
              System.out.println("The value of i is: "+i);
         }
    }
}
```

For loop example to iterate an array:

Here we are iterating and displaying array elements using the for loop.

```
class ForLoopExample3 {
    public static void main(String args[]){
         int arr[]={2,11,45,9};
         //i starts with 0 as array index starts with 0 too
         for(int i=0; i<arr.length; i++){
              System.out.println(arr[i]);
         }
    }
}
```

Enhanced For loop

Enhanced for loop is useful when you want to iterate Array/Collections, it is easy to write and understand.

Let's take the same example that we have written above and rewrite it using **enhanced for loop**.

```
class ForLoopExample3 {
   public static void main(String args[]){
      int arr[]={2,11,45,9};
      for (int num : arr) {
         System.out.println(num);
      }
   }
}
```

While loop in Java with examples

Syntax of while loop

```
while(condition)
{
   statement(s);
}
```

How while Loop works?

In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute. When condition returns false, the control comes out of loop and jumps to the next statement after while loop.

Note: The important point to note when using while loop is that we need to use increment or decrement statement inside while loop so that the loop variable gets changed on each iteration, and at some point condition returns false. This

way we can end the execution of while loop otherwise the loop would execute indefinitely.

Simple while loop example

```
class WhileLoopExample {
    public static void main(String args[]){
         int i=10;
         while(i>1){
              System.out.println(i);
              i--;
         }
    }
}
```

do-while loop in Java with example

First, the statements inside loop execute and then the condition gets evaluated, if the condition returns true then the control gets transferred to the “do” else it jumps to the next statement after do-while.

do-while loop example

```
class DoWhileLoopExample {
    public static void main(String args[]){
         int i=10;
         do{
              System.out.println(i);
              i--;
         }while(i>1);
    }
}
```

Continue statement is mostly used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop’s body for the current iteration. This is particularly useful when you want to continue the loop but do not want the rest of the statements(after continue statement) in loop body to execute for that particular iteration.

Syntax:
continue word followed by semi colon.

```
continue;
```

Example: continue statement inside for loop

```
public class ContinueExample {
```

```
   public static void main(String args[]){
        for (int j=0; j<=6; j++)
        {
           if (j==4)
           {
              continue;
           }

           System.out.print(j+" ");
        }
   }
}
```

The **break statement** is usually used in following two scenarios:

a) Use break statement to come out of the loop instantly. Whenever a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated for rest of the iterations. It is used along with if statement, whenever used inside loop so that the loop gets terminated for a particular condition.

The important point to note here is that when a break statement is used inside a nested loop, then only the inner loop gets terminated.

b) It is also used in switch case control. Generally all cases in switch case are followed by a break statement so that whenever the program control jumps to a case, it doesn't execute subsequent cases (see the example below). As soon as a break is encountered in switch-case block, the control comes out of the switch-case body.

Syntax of break statement:
"break" word followed by semi colon

Example - Use of break in a while loop

In the example below, we have a while loop running from o to 100 but since we have a break statement that only occurs when the loop value reaches 2, the loop gets terminated and the control gets passed to the next statement in program after the loop body.

```
public class BreakExample1 {
   public static void main(String args[]){
      int num =0;
      while(num<=100)
      {
          System.out.println("Value of variable is: "+num);
          if (num==2)
```

```
            {
               break;
            }
            num++;
        }
        System.out.println("Out of while-loop");
    }
}
```

Example - Use of break in a for loop

The same thing you can see here. As soon as the `var` value hits 99, the for loop gets terminated.

```
public class BreakExample2 {

   public static void main(String args[]){
        int var;
        for (var =100; var>=10; var --)
        {
            System.out.println("var: "+var);
            if (var==99)
            {
                 break;
            }
         }
         System.out.println("Out of for-loop");
   }
}
```