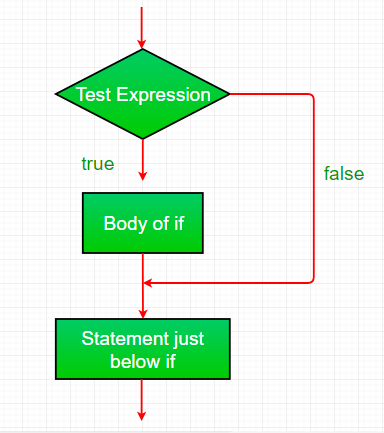
**Flow control in Java**

**Decision Making in Java (if, if-else, switch, break, continue)**

Decision Making in programming is similar to decision-making in real life. In programming also face some situations where we want a certain block of code to be executed when some condition is fulfilled.

A programming language uses control statements to control the flow of execution of a program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

**Types :**

1. if
2. if-else
3. nested-if
4. if-else-if
5. switch-case
6. jump – (break, continue, return)
7. **if:**

if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statements is executed otherwise not.

**if(condition)**

**{**

// Statements to execute if

// condition is true

**}**

1. **if-else:**

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false? Here comes the else statement. We can use the else statement with the if statement to execute a block of code when the condition is false.

**if (condition)**

**{**

// Executes this block if

// condition is true

**}**

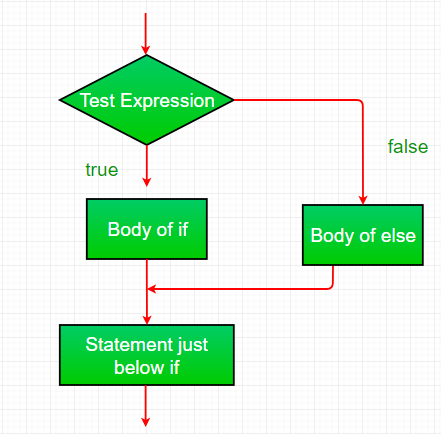
**else**

**{**

// Executes this block if

// condition is false

**}**

****

1. **nested-if:**

A nested if is an if statement that is the target of another if or else. Nested if statements mean an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e., we can place an if statement inside another if statement.

**if (condition1)**

**{**

// Executes when condition1 is true

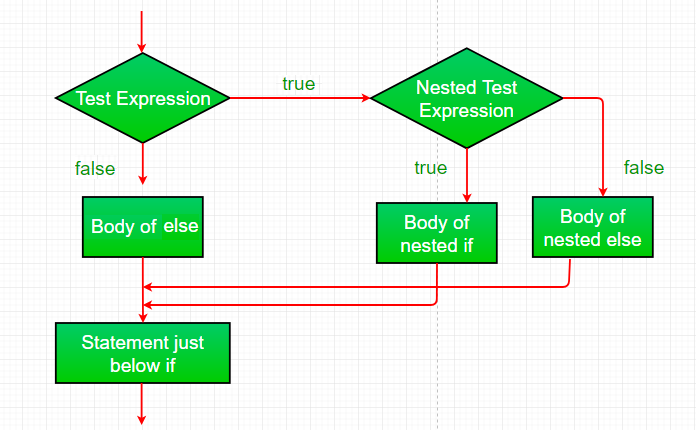
if (condition2)

**{**

// Executes when condition2 is true

**}**

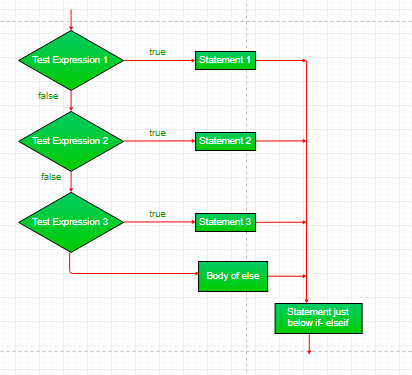
**}**



1. **if-else-if ladder:**

Here, a user can decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that ‘if’ is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. There can be as many as ‘else if’ blocks associated with one ‘if’ block but only one ‘else’ block is allowed with one ‘if’ block.

|  |
| --- |
| **if (condition)**  statement;  **else if (condition)**  statement;  .  .  **else**  statement; |



1. **Switch Statement :**

It is like an if-else-if ladder statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression. The expression can be a byte, short, char, or int primitive data type. It tests the equality of variables against multiple values.

* **Some Important Rules for Switch Statements**
* There can be any number of cases just imposing condition check but remember duplicate case/s values are not allowed.
* The value for a case must be of the same data type as the variable in the switch.
* The value for a case must be constant or literal. Variables are not allowed.
* The break statement is used inside the switch to terminate a statement sequence.
* The break statement is optional. If omitted, execution will continue into the next case.

// switch statement

**switch(expression)**

**{**

// case statements

// values must be of the same type of expression

**case value1 :**

// Statements

break; // break is optional

**case value2 :**

// Statements

break; // break is optional

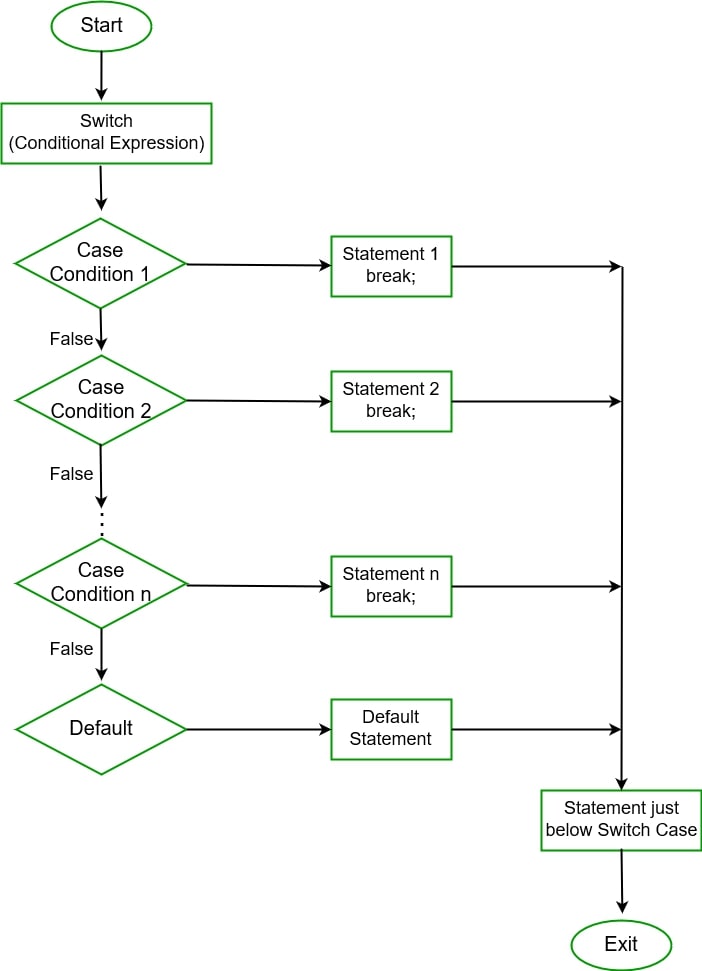
// We can have any number of case statements

// below is the default statement, used when none of the cases is true.

// No break is needed in the default case.

**default :**

// Statements **}**



**6.Jump Statements:**

Java supports three jump statements**: break, continue,** and **return**. These three statements transfer control to another part of the program.

1. **Break:** In Java, a break is majorly used for

* Terminate a sequence in a switch statement (discussed above).
* To exit a loop.

1. **Continue:**

Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration. This is, in effect, a go-to just past the body of the loop, to the loop’s end. The continue statement performs such an action.

1. **Return:**

The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method.

* **Loops in Java :**

Looping in programming languages is a feature that facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. Java provides three ways for executing the loops.

* **While loop :**

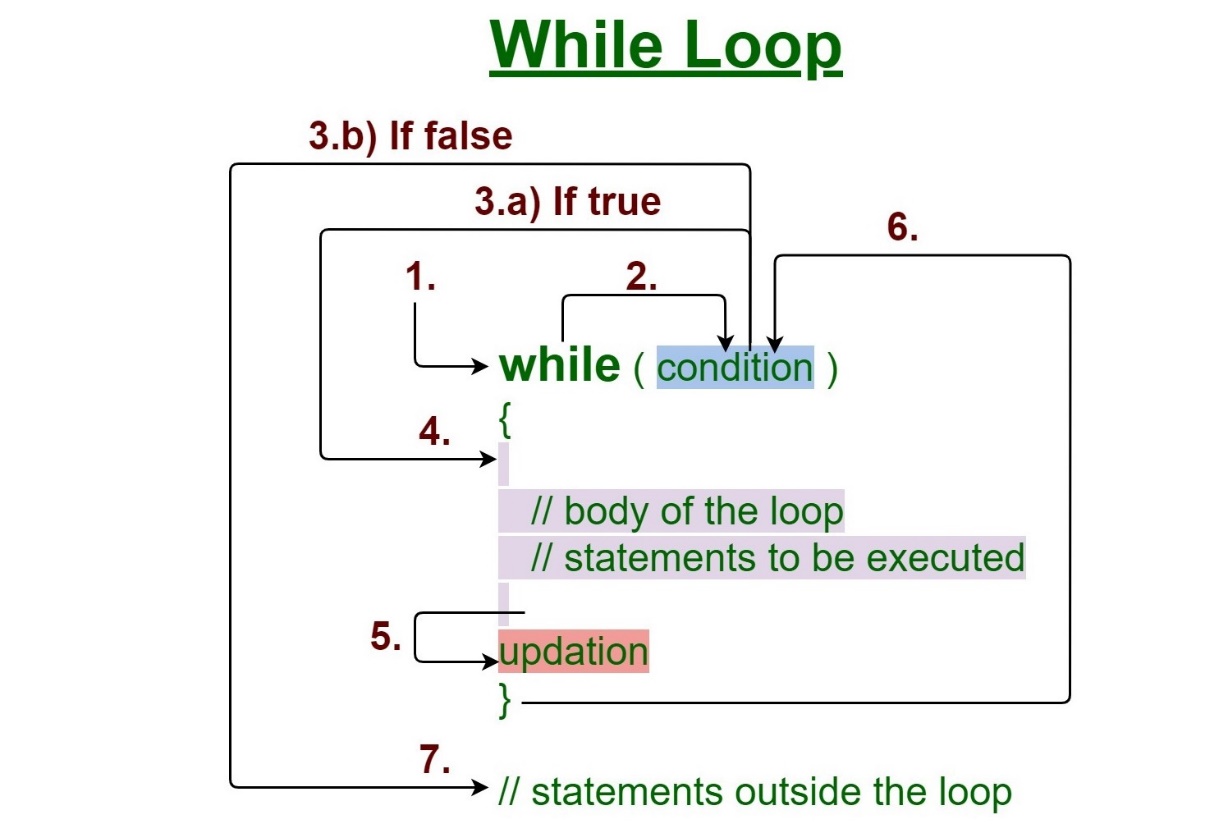
A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

**while (test\_expression)**

**{**

// statements

update\_expression;

**}**

**How Does a While loop execute?**

1. Control falls into the while loop.
2. The flow jumps to Condition
3. Condition is tested.
4. If the Condition yields true, the flow goes into the Body.
5. If the Condition yields false, the flow goes outside the loop
6. The statements inside the body of the loop get executed.
7. Updation takes place.
8. Control flows back to Step 2.
9. The while loop has ended and the flow has gone outside.

* **for loop:**

for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

**for (initialization expr; test expr; update exp)**

**{**

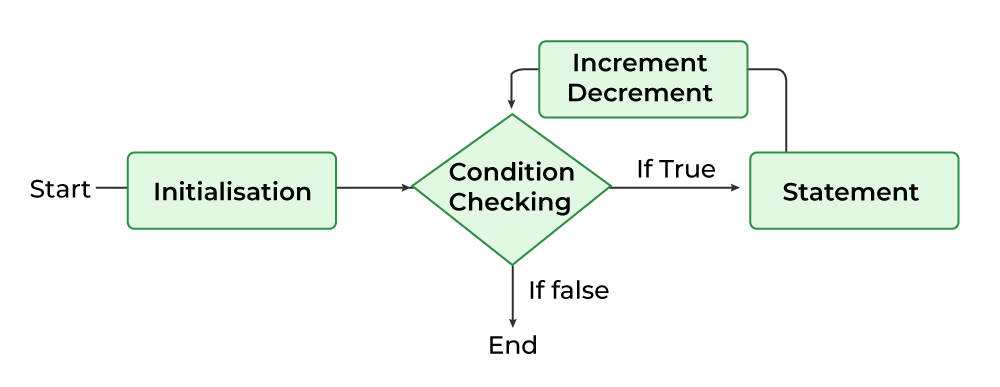
// body of the loop

// statements we want to execute

**}**

**How does a For loop work?**

1. Control falls into the for loop. Initialization is done
2. The flow jumps to Condition
3. Condition is tested.
4. If the Condition yields true, the flow goes into the Body
5. If the Condition yields false, the flow goes outside the loop
6. The statements inside the body of the loop get executed.
7. The flow goes to the Updation
8. Updation takes place and the flow goes to Step 3 again
9. The for loop has ended and the flow has gone outside.



**For-Each Loop**

Enhanced For Loop or Java For-Each loop in Java is another version of for loop introduced in Java 5. Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. It is inflexible and should be used only when there is a need to iterate through the elements in a sequential manner without knowing the index of the currently processed element.

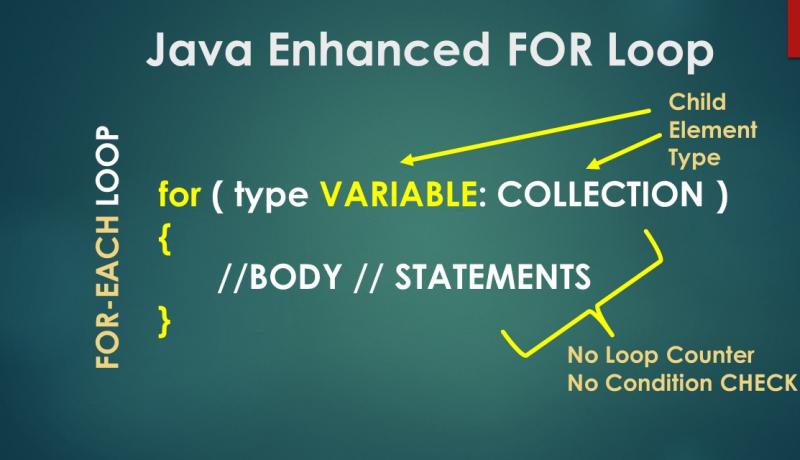
**for (T element: Collection obj/array)**

**{**

// loop body

// statement(s)

**}**



**Do-While loop :**

Java do-while loop is an Exit control loop. Therefore, unlike for or while loop, a do-while check for the condition after executing the statements of the loop body**.**

**do**

**{**

// Loop Body

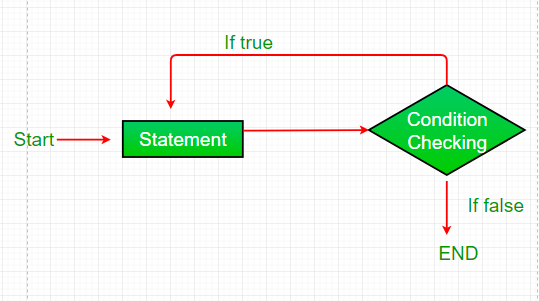
Update\_expression

**}**

// Condition check

**while (test\_expression);**

**Execution of do-While loop**

1. Control falls into the do-while loop.
2. The statements inside the body of the loop get executed.
3. Updation takes place.
4. The flow jumps to Condition
5. Condition is tested.
6. If Condition yields true, go to Step 6.
7. If Condition yields false, the flow goes outside the loop
8. The flow goes back to Step 2.