
Pranaya
JAVA

Java Control Statements

The statements that control the execution flow of the program are known as control statements.

When we try to execute a program, we modify and repeat the data several times. We need some tools for these modifications that will control the flow of the program, and perform this type of task Java Provides control statements.

There are three types of control statement

1- Decision Making statements

* ***if statements***

  Simple if statement
  if-else statement
  if-else-if ladder
  Nested if-statement

* ***switch statement***

2- Loop statements

do while loop
while loop
for loop
for-each loop

3- Jump statements (we will learn it on the next PDF)

break statement
continue statement

if Statement

In Java, the "if" statement is used to evaluate a condition.

Simple if-statement

Use if to specify a block of code to be executed, if a specified condition is true. Use else to specify a block of code to be executed if the same condition is false. Use else if to specify a new condition to test, if the first condition is false.

SYNTAX:

if(condition) {
statement 1; //executes when condition is true
}

Example:

```
public class one{

public static void main(String[] args) {
int markes=100;
if(markes>33)
{
System.out.println("pass");
}
}
}
```

Output:
pass

The if-else statement

is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
if(condition) {
statement 1; //executes when condition is true
}
else{
statement 2; //executes when condition is false
}
```

Example


```
public class ifelseex {
public static void main(String[] args) {
int markes=100;
if(markes<33)
{
System.out.println("pass");
}
else
{
System.out.println("failed");
}

}

}
```

if-else-if :

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that creates a decision tree where the program may enter the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax:

```
if(condition 1) {
statement 1; //executes when condition 1 is true
}
else if(condition 2) {
statement 2; //executes when condition 2 is true
}
else {
statement 2; //executes when all the conditions are false
}
```

Example

```
public class Student {
public static void main(String[] args) {
String city = "Delhi";
if(city == "Meerut") {
System.out.println("city is meerut");
}
else if (city == "Noida") {
System.out.println("city is noida");
}
else if(city == "Agra") {
System.out.println("city is agra");
}
else {
System.out.println(city);
}
}
```

Output: Delhi

Nested if-statement

In nested if-statements, the if statement can contain an if or if-else statement inside another if or else-if statement.

Syntax.

```
if(condition 1) {
statement 1; //executes when condition 1 is true
if(condition 2) {
statement 2; //executes when condition 2 is true
}
else{
statement 2; //executes when condition 2 is false
}
```

Example

```
public class Student {
public static void main(String[] args) {
String address = "Delhi, India";

if(address.endsWith("India")) {
if(address.contains("Meerut")) {
System.out.println("Your city is Meerut");
}
else if(address.contains("Noida")) {
System.out.println("Your city is Noida");
}
else {
System.out.println(address.split(",")[0]);
}
}
else {
System.out.println("You are not living in India");
}
}
}
```

Output:- Delhi

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of the if-else-if statements. It also enhances the readability of the program.

Points to be noted about the switch statement:

1. The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
2. Cases cannot be duplicated
3. A default statement is executed when any of the cases doesn't match the value of the expression. It is optional.
4. The break statement terminates the switch block when the condition is satisfied.
5. It is optional, if not used the next case is executed.
6. While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

Syntax

```
switch (expression){
    case value1:
     statement1;
     break;
    .
    .
    .
    case valueN:
     statementN;
     break;
    default:
     default statement;
}
```

example

```
public class switchassn {
public static void main(String[] args) {
String country="nepal";
switch(country)
{
case "india":
System.out.println("Capital of india is New Delhi");
break;
case "nepal":
System.out.println("capital of nepal is kathmandu");
break;
case "uk":
System.out.println("capital of uk is london");
break;
case "usa":
System.out.println("capital of usa is Washingtone-Dc");
break;
case "uae":
System.out.println("capital of uae is abu dhabi");
break;
case "australia":
System.out.println("capital of australia is canberra");
break;
default:
System.out.println("There is no information about this name");
}
}
}
```

Output:

capital of nepal is Kathmandu

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly

1. while loop
2. for loop
3. do-while loop

for loop

When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

Syntax

```
for (statement 1; statement 2; statement 3) {
 // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.

Example

```
public class Main {
public static void main(String[] args) {
for (int i = 0; i < 5; i++) {
System.out.println(i);
}
}
}
```

Output : 0 1 2 3 4

for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collections. In the for-each loop.

Syntax:

```
for(data_type var : array_name/collection_name)
{
//statements
}
```

Example

```
1.public class Calculation {
2.public static void main(String[] args) {
3.// TODO Auto-generated method stub
4.String[] names = {"Java","C","C++","Python","JavaScript"};
5.System.out.println("Printing the content of the array names:\n");
6.for(String name:names) {
7.System.out.println(name);
8.}
9.}
10. }
```

Output:

Printing the content of the array names:

Java
C
C++
Python
JavaScript

While Loop

The while loop loops through a block of code as long as a specified condition is true:

Syntax

```
while (condition)
{
  // code block to be executed
}
```

Example

```
public class Main {
public static void main(String[] args) {
int i = 0;
while (i < 5) {
System.out.println(i);
i++;
}
}
}
```

Output: 0 1 2 3 4

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
  // code block to be executed
}
while (condition);
```

Example

```
public class whileex {

public static void main(String[] args) {
int a=0;
do
{
System.out.println("nice");
a++;
}
while(a<=3);

}

}
```

Output: nice
nice
nice
nice

Thank you!

Priya