Java Conditions and if Statements

Java supports the usual logical conditions from mathematics.

1) less than: ⟶ a<b
2) less than or equal to: ⟶ a<=b
3) Greater than: ⟶ a>b
4) Greater than or equal to ⟶ a>=b
5) Equal to: ⟶ a==b
6) Not Equal to: ⟶ a!=b

Java has the following conditional statements.

1) if
2) else
3) else if
4) switch.

The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

Syntax:-

```
if (condition) {
  // block of code to be executed if the
     condition is true.
}
```

Note:- if if is in lower case letters, it is condition. (if or If) are different. If will generate error.

eg:-
```
if (20>18) {
    System.out.println("20 is greater");
}
```

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:-
```
if (condition) {
// block of code to be executed if the
   condition is true
}
else {
// block of code to be executed if the
   condition is false
}
```

Eg:-
```
int time = 20;
if (time < 18) {
  System.out.println("Good day");
} else {
  System.out.println("Good evening.");
}
```

output :- "Good evening."

The else..if Statement

use the else if statement to specify a new condition if the first condition is false.

Syntax:-
```
if (condition1) {
  // block of code to be executed if condition 1
     is true.
} else if (condition 2) {
  // block of code to be executed if the condition
  is false and condition 2 is true.
} else {
  // block of code to be executed if the
     condition1 is false and condition 2 is
     false.
```

Eg:

```
int time = 22;
if (time < 10) {
    System.out.println("Good morning");
}
else if (time < 20)
{
  System.out.println("Good day");
}
else {
  System.out.println("Good evening");
}
```

output :- "Good evening".

Short Hand If ... else

There is also a short-hand if else, which is known as the ternary operator because it consists of three operands.

It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements.

Syntax :-

```
variable = (condition) ? expression true :
           expression false ;
```

Instead of writing :-

```
int time = 20;
if (time < 18) {
    System.out.println ("Good day");
}
else {
    System.out.println ("Good evening");
}
```

You can simply write

```
int time = 20;
String result = (time < 18) ? "Good day" :
                "Good evening";

System.out.println (result);
```

output :- Good evening.

JAVA Switch

Instead of writing many if-else statements, you can use the switch statement.

The switch statement selects one of many code blocks to be executed.

Syntax:

```
switch (expression) {
    case x:
        //code block
        break;

    case y:
        //code block
        break;

    default:
        //code block.

}
```

Working:-

1) The switch expression is evaluated once.
2) The value of the expression is compared with the values of each case.

3) If case is a match, the associated block of code is executed.

4) The break and default keywords are optional, and will be described later in this chapter.

The example below uses the weekday number to calculate the weekday name:

eg:

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
}
//outputs "Thursday" (day 4)
```

break Keyword

When Java reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match found, and the job is done, its time for a break. There is no need for more testing.

NOTE:-
A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

NOTE! The default keyword:-

The default keyword specified some code to run if there is no case match.

eg:-
```
default:
    System.out.println("Happy programming");
}
```

Java while Loop

Loops:-

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

Java 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
}
```

eg :-

```
int i = 0;
while (i<5)
{
  System.out.println(i);
  i++;
}
```

NOTE:- Do not forget to increse the variable used in the condition, otherwise the loop will never end!

The 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.

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

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Eg:-

```
int i = 0;
do {
    System.out.println(i);
    i++;
} while (i<5);
```

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

Java 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.

Statement3 is executed (every time) after the code block has been executed.

eg:

```
for (int i=0; i<5; i++)
{
    System.out.println(i);
}
```

eg:

```
for (int i=0; i<=10; i=i+2)
{
    System.out.println(i);
}
```

For each Loop

There is also a "for-each" loop, which is used exclusively to loop through elements in an array.

Syntax

```
for (type variableName : arrayName)
{
  // code block to be executed
}
```

The following example outputs all elements in the cars array, using a "for-each" loop:

Eg:

```
String[] cars = {"Volvo", "BMW", "Ford"};
for (String i : cars)
{
  System.out.println(i);
}
```

NOTE :- we will learn more about Arrays further.