Conditional Statements

Java supports the usual logical conditions from mathematics:

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to a == b
- Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.
Java has the following conditional statements:

- 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
- Use switch to specify many alternative blocks of code to be executed

If-else statements

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

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

- Note that if is in lowercase letters. Uppercase letters (If or IF) will generate an error.
- In the example below, we test two values to find out if 20 is greater than 18. If the condition is true, print some text:

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

- We can also test variables:

```
int x = 20;
        int y = 18;
        if (x > y) {
            System.out.println("x is greater than y");
        }
```

Example explained

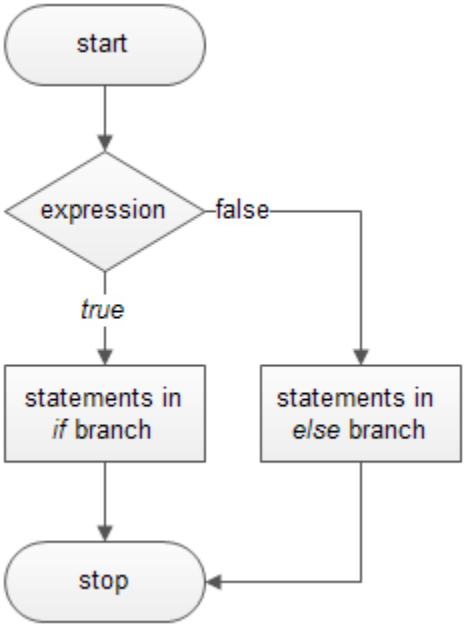
• In the example above we use two variables, **x** and **y**, to test whether x is greater than y (using the > operator). As x is 20, and y is 18, and we know that 20 is greater than 18, we print to the screen that "x is greater than y".

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
        }
```




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

Example explained

In the example above, time (20) is greater than 18, so the condition is false. Because of this, we move on to the else condition and print to the screen "Good evening". If the time was less than 18, the program would print "Good day".

The same task could be done using the Ternary Operator. Please refer the Java Operators note.

```
int num =3;
String output = (num % 2 == 0 ) ? "Even number" :"Odd number";
System.out.println(output);
```

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 condition1 is true
} else if (condition2) {
    // block of code to be executed if the condition1 is false and
condition2 is true
} else {
   // block of code to be executed if the condition1 is false and
condition2 is false
}
```

```
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.");
        }
        // Outputs "Good evening."
```

Example explained

• In the example above, time (22) is greater than 10, so the first condition is false. The next condition, in the else if statement, is also false, so we move on to the else condition since condition1 and condition2 is both false - and print to the screen "Good evening".

• However, if the time was 14, our program would print "Good day."

EXCERSISE:

Write a program to check POSITIVE, NEGATIVE or ZERO:

```
int number = -13;
if (number <0) {
    System.out.println("This is a negative number");
}else if(number > 0){
    System.out.println("This is a positive number");
}else{
    System.out.println("This is zero");
}
```

Java Nested if statement

- The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

```
if(condition){
    //code to be executed
    if(condition){
        //code to be executed
         }
}
```

Example: A program to know whether you are eligible to donate blood.

```
int age = 20;
double weight = 65.5;

if (age >17) {
    if (weight>49.9) {
        System.out.println("You can donate blood.");
    }else {
        System.out.println("Sorry,you cannot donate blood
because you are underweight.");
    }
}else {
    System.out.println("Sorry, you cannot donate blood because
you are underage.");
}
```

Java Switch

Use the switch statement to select one of many code blocks to be executed. The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums* and string.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

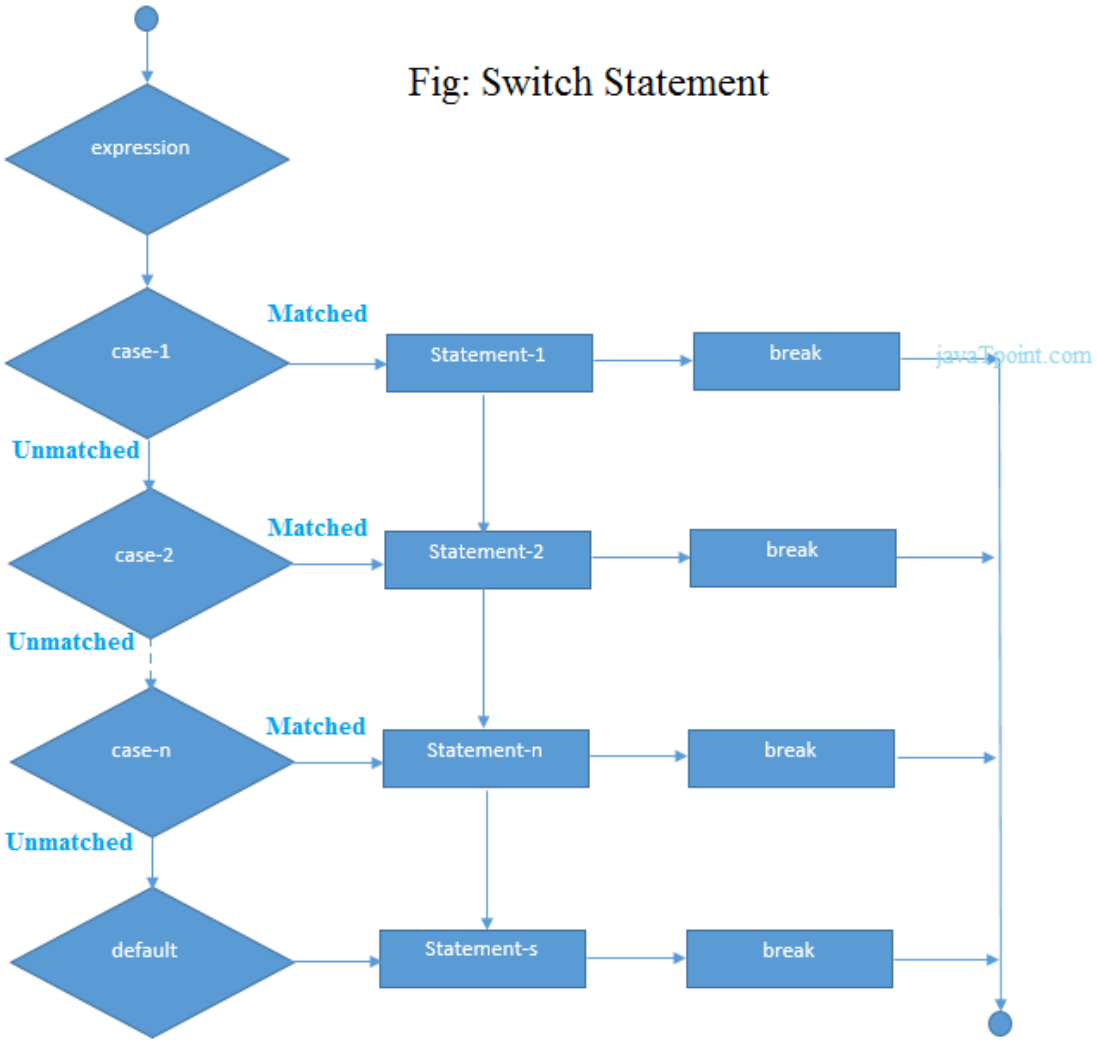
Syntax

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

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and default keywords are optional, and will be described later in this chapter

Fig: Switch Statement



Example:

```
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;
```

```
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

Java Switch Statement is fall-through

The Java switch statement is fall-through. It means it executes all statements after the first match if a break statement is not present.

```
        int num = 20;

        switch (num){
            case 10:System.out.println("This is number 10");
            case 20:System.out.println("This is number 20");
            case 30:System.out.println("This is number 30");
            case 40:System.out.println("This is number 40");
            default:System.out.println("This is not 10,20,30 or 40");
        }

/**
Output

This is number 20
This is number 30
This is number 40
This is not 10,20,30 or 40

*/
```

Java Switch Statement with String

Java allows us to use strings in switch expression since Java SE 7. The case statement should be string literal.

Example:

```
String level ="beginner";
int levelInt;

switch (level){
    case "beginner":
        levelInt =1;
        break;
    case "intermediate":
        levelInt = 2;
        break;
    case "expert":
        levelInt =3 ;
        break;
    default:
        levelInt =0;
}
```

The 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 is found, and the job is done, it's time for a break. There is no need for more testing.
- 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.

The default Keyword

- The default keyword specifies some code to run if there is no case match: