

## If else statements in Java

In Java, **if-else statements** are used to perform **decision-making operations**. It lets your program take different actions based on **conditions**.

### Syntax of if, if-else, and if-else if in Java

#### 1. Simple if statement

```
if (condition) {  
    // Code to execute if condition is true  
}
```

#### Example:

```
int age = 18;  
if (age >= 18) {  
    System.out.println("You are eligible to vote.");  
}
```

#### 2. if-else statement

#### Example:

```
int age = 16;  
if (age >= 18) {  
    System.out.println("You are eligible to vote.");  
} else {  
    System.out.println("You are not eligible to vote.");  
}
```

#### 3. if-else if-else ladder

#### Example:

```
int marks = 75;  
  
if (marks >= 90) {  
    System.out.println("Grade: A");  
} else if (marks >= 75) {  
    System.out.println("Grade: B");  
} else if (marks >= 60) {  
    System.out.println("Grade: C");  
} else {  
    System.out.println("Grade: D");  
}
```

#### Notes:

- The **condition** must return a boolean value (`true` or `false`).
- You can nest `if-else` blocks inside each other.
- Java uses **curly braces** `{ }` to group multiple statements, though for one-line statements, they are optional (not recommended for beginners).

## Switch case Statement in Java

The `switch` statement is used to execute **one block of code among many options**. It is a better alternative to using many `if-else-if` statements when you're checking a **single variable** against multiple constant values.

### Important Points:

- The expression should be **byte, short, int, char**, `String` (since Java 7), or enum type.
- case values must be **unique constants**.
- `break` is used to **exit** the switch block after a match is found. If not used, execution will "fall through" to the next case.
- default is **optional**, and it runs if no case matches.

### Example 1: Using `int`

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

### Output:

Wednesday

### Example 2: Using `String` (Java 7+)

```
String fruit = "Apple";
switch (fruit) {
    case "Apple":
        System.out.println("Red fruit");
        break;
    case "Banana":
        System.out.println("Yellow fruit");
        break;
    default:
        System.out.println("Unknown fruit");
}
```

### Example 3: Without `break` (fall-through behavior)

```
int num = 2;
switch (num) {
```

```

    case 1:
        System.out.println("One");
    case 2:
        System.out.println("Two");
    case 3:
        System.out.println("Three");
}

```

### Output:

```

Two
Three

```

Because there's no `break`, it continues executing all the following cases.

### When to Use:

Use `switch` when you have:

- One variable to check
- Multiple possible constant values to match

## String Methods in Java

In Java, the `String` class provides many **useful methods** to manipulate and process text. Below is a list of commonly used methods with examples.

### 1. `length()`

Returns the number of characters in the string.

```

String str = "Hello";
System.out.println(str.length()); // Output: 5

```

### 2. `charAt(int index)`

Returns the character at the specified index.

```

String str = "Java";
System.out.println(str.charAt(2)); // Output: v

```

### 3. `substring(int beginIndex)`

Returns a substring from the given index to the end.

```

String str = "Programming";
System.out.println(str.substring(3)); // Output: gramming

```

### 4. `substring(int beginIndex, int endIndex)`

Returns a substring from `beginIndex` to `endIndex - 1`.

```
System.out.println(str.substring(0, 4)); // Output: Prog
```

### 5. equals(String anotherString)

Checks if two strings are **exactly equal** (case-sensitive).

```
String a = "hello";  
String b = "hello";  
System.out.println(a.equals(b)); // Output: true
```

### 6. equalsIgnoreCase(String anotherString)

Checks equality **ignoring case**.

```
String a = "Hello";  
String b = "hello";  
System.out.println(a.equalsIgnoreCase(b)); // Output: true
```

### 7. toUpperCase(), toLowerCase()

Converts the string to upper/lower case.

```
String str = "Java";  
System.out.println(str.toUpperCase()); // Output: JAVA  
System.out.println(str.toLowerCase()); // Output: java
```

### 8. contains(CharSequence seq)

Returns true if the sequence is present in the string.

```
String str = "Welcome";  
System.out.println(str.contains("come")); // Output: true
```

### 9. replace(char oldChar, char newChar)

Replaces characters in the string.

```
String str = "banana";  
System.out.println(str.replace('a', 'o')); // Output: bonono
```

### 10. trim()

Removes whitespace from both ends of the string.

```
String str = " Hello World ";  
System.out.println(str.trim()); // Output: Hello World
```

### 11. startsWith() / endsWith()

Checks if string starts/ends with a given prefix/suffix.

```
String str = "hello.java";
System.out.println(str.startsWith("hello")); // true
System.out.println(str.endsWith(".java"));   // true
```

## 12. indexOf() / lastIndexOf()

Returns index of first/last occurrence of a character or substring.

```
String str = "programming";
System.out.println(str.indexOf('g'));          // Output: 3
System.out.println(str.lastIndexOf('g'));      // Output: 10
```

## 13. isEmpty() / isBlank() (Java 11+)

Checks if the string is empty or blank (only spaces).

```
String str1 = "";
String str2 = " ";
System.out.println(str1.isEmpty()); // true
System.out.println(str2.isBlank()); // true (Java 11+)
```

# Nested Loops in Java

A **nested loop** means a loop **inside another loop**. It's useful for working with patterns, matrices, or multi-level iterations.

### Syntax:

```
for (initialization; condition; update) {
    for (initialization; condition; update) {
        // inner loop body
    }
    // outer loop body
}
```

You can nest any kind of loop:

- for **inside** for
- while **inside** for
- do-while **inside** while, etc.

### Example 1: Nested for loop (Print a rectangle of stars)

```
for (int i = 1; i <= 3; i++) {           // outer loop (rows)
    for (int j = 1; j <= 5; j++) {       // inner loop
        (columns)
        System.out.print("* ");
    }
}
```

```
        System.out.println();           // move to the next
line
}
```

**Output:**

```
* * * * *
* * * * *
* * * * *
```