



# CORE JAVA

By : Kalpesh Chauhan



# VARIABLES

# JAVA VARIABLES

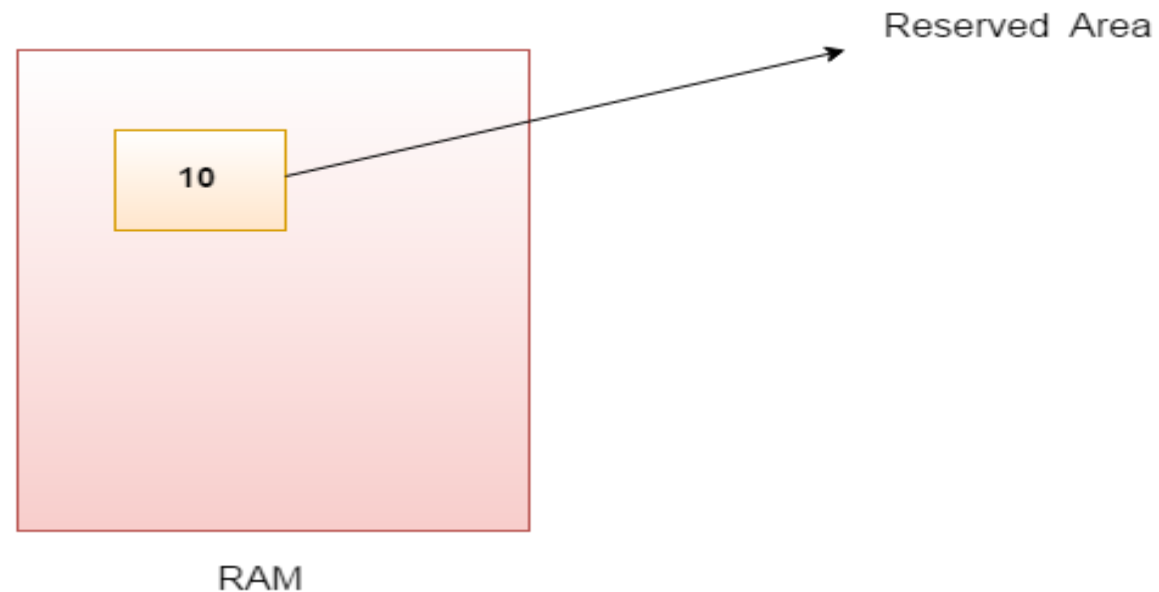
A variable is a container which holds the value while the java program is executed. A variable is assigned with a datatype.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

# VARIABLE

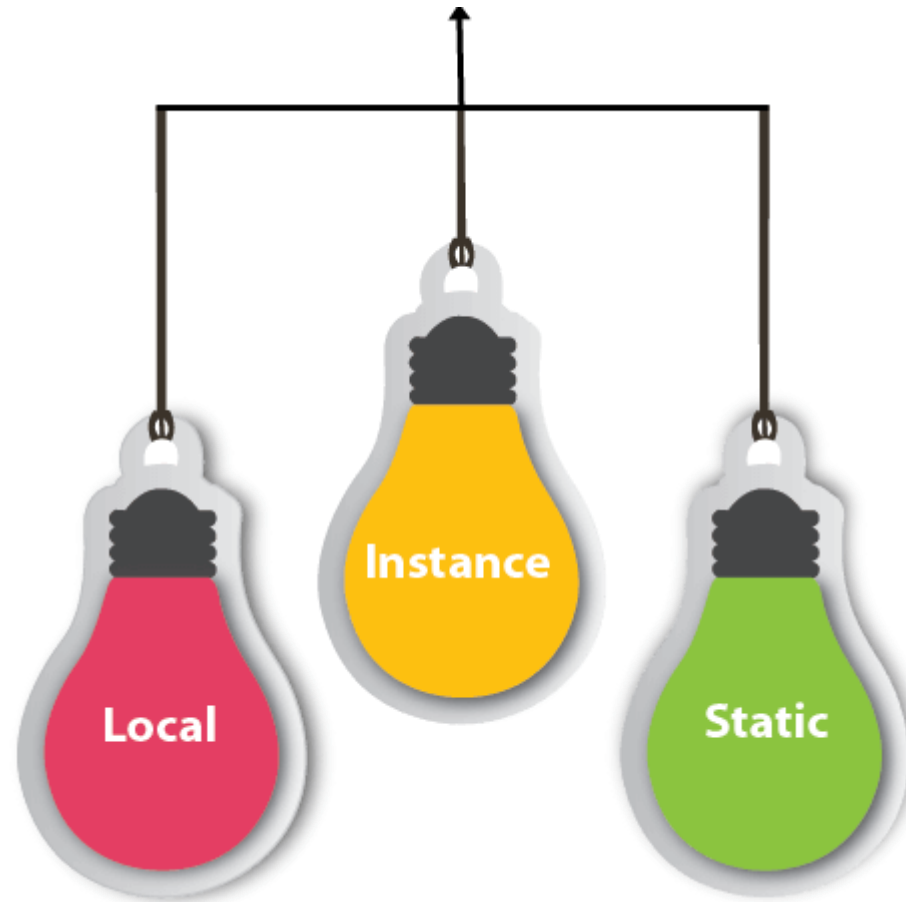
**Variable** is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.



# TYPES OF VARIABLES

There are three types of variables in java:

1. local variable
2. instance variable
3. static variable



## 1) *Local Variable*

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists. A local variable cannot be defined with "static" keyword.

## 2) *Instance Variable*

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static. It is called instance variable because its value is instance specific and is not shared among instances.

### 3) *Static variable*

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

# EXAMPLE TO UNDERSTAND THE TYPES OF VARIABLES IN JAVA

[Click here to view Program](#)



# DATA TYPES IN JAVA

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

**Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.

**Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

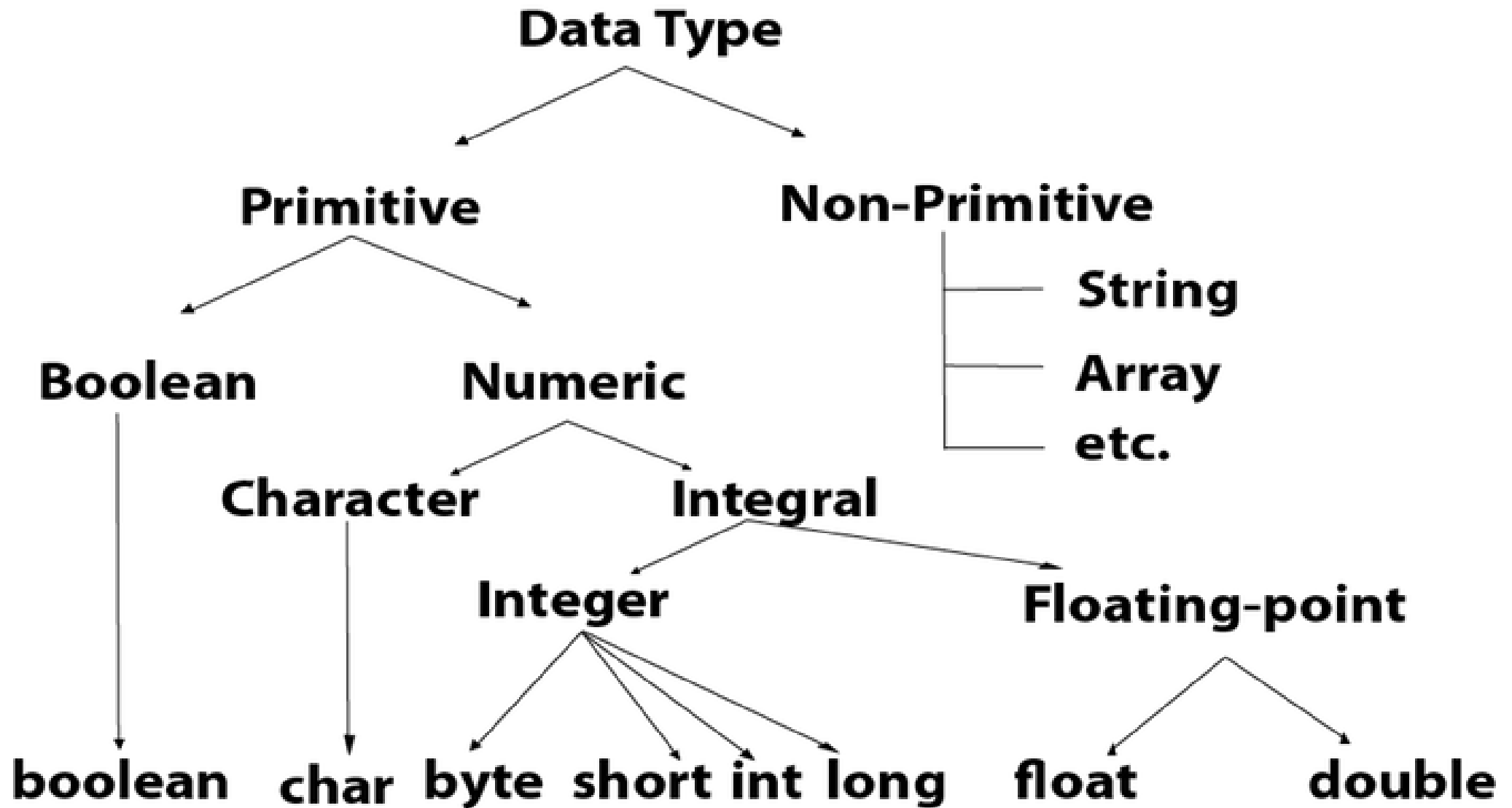
# JAVA PRIMITIVE DATA TYPES

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

“Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.”

# THERE ARE 8 TYPES OF PRIMITIVE DATA TYPES:

1. boolean data type
2. byte data type
3. char data type
4. short data type
5. int data type
6. long data type
7. float data type
8. double data type



Data Type	Default Value	Default size
Boolean	false	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

# BOOLEAN DATA TYPE

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

**Example:** Boolean one = false

# BYTE DATA TYPE

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

**Example:** byte a = 10, byte b = -20

# SHORT DATA TYPE

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

**Example:** short s = 10000, short r = -5000



# INT DATA TYPE

The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 ( $-2^{31}$ ) to 2,147,483,647 ( $2^{31} - 1$ ) (inclusive). Its minimum value is - 2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

**Example:** int a = 100000, int b = -200000

# LONG DATA TYPE

The long data type is a 64-bit two's complement integer. Its value-range lies between  $-9,223,372,036,854,775,808$  ( $-2^{63}$ ) to  $9,223,372,036,854,775,807$  ( $2^{63} - 1$ )(inclusive). Its minimum value is  $-9,223,372,036,854,775,808$  and maximum value is  $9,223,372,036,854,775,807$ . Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

**Example:** long a = 100000L, long b = -200000L

# FLOAT DATA TYPE

The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**Example:** float f1 = 234.5f

# DOUBLE DATA TYPE

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

**Example:** double d1 = 12.3

# CHAR DATA TYPE

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). The char data type is used to store characters.

**Example:** `char letterA = 'A'`

# WHY CHAR USES 2 BYTE IN JAVA AND WHAT IS \U0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system.

Why java uses Unicode System?

Before Unicode, there were many language standards:

**ASCII** (American Standard Code for Information Interchange) for the United States.

**ISO 8859-1** for Western European Language.

**KOI-8** for Russian.

**GB18030 and BIG-5** for chinese, and so on.

# PROBLEM

## **This caused two problems:**

A particular code value corresponds to different letters in the various language standards.

The encodings for languages with large character sets have variable length. Some common characters are encoded as single bytes, other require two or more byte.

# SOLUTION

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

**lowest value:** \u0000

**highest value:** \uFFFF





# OPERATORS

# OPERATORS IN JAVA

**Operator** in java is a symbol that is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in java which are given below:

# TYPES OF OPERATOR

1. Unary Operator,
2. Arithmetic Operator,
3. Shift Operator,
4. Relational Operator,
5. Bitwise Operator,
6. Logical Operator,
7. Ternary Operator and
8. Assignment Operator.

Operator Type	Category	Precedence
Unary	postfix	<code>expr++ expr--</code>
	prefix	<code>++expr -- expr +expr -expr ~ !</code>
Arithmetic	multiplicative	<code>* / %</code>
	additive	<code>+ -</code>
Shift	shift	<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>
Relational	comparison	<code>&lt; &gt; &lt;= &gt;= instanceof</code>
	equality	<code>== !=</code>

Operator Type	Category	Precedence
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical	logical AND	&&
	logical OR	
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^=  = <<= >>= >>>=

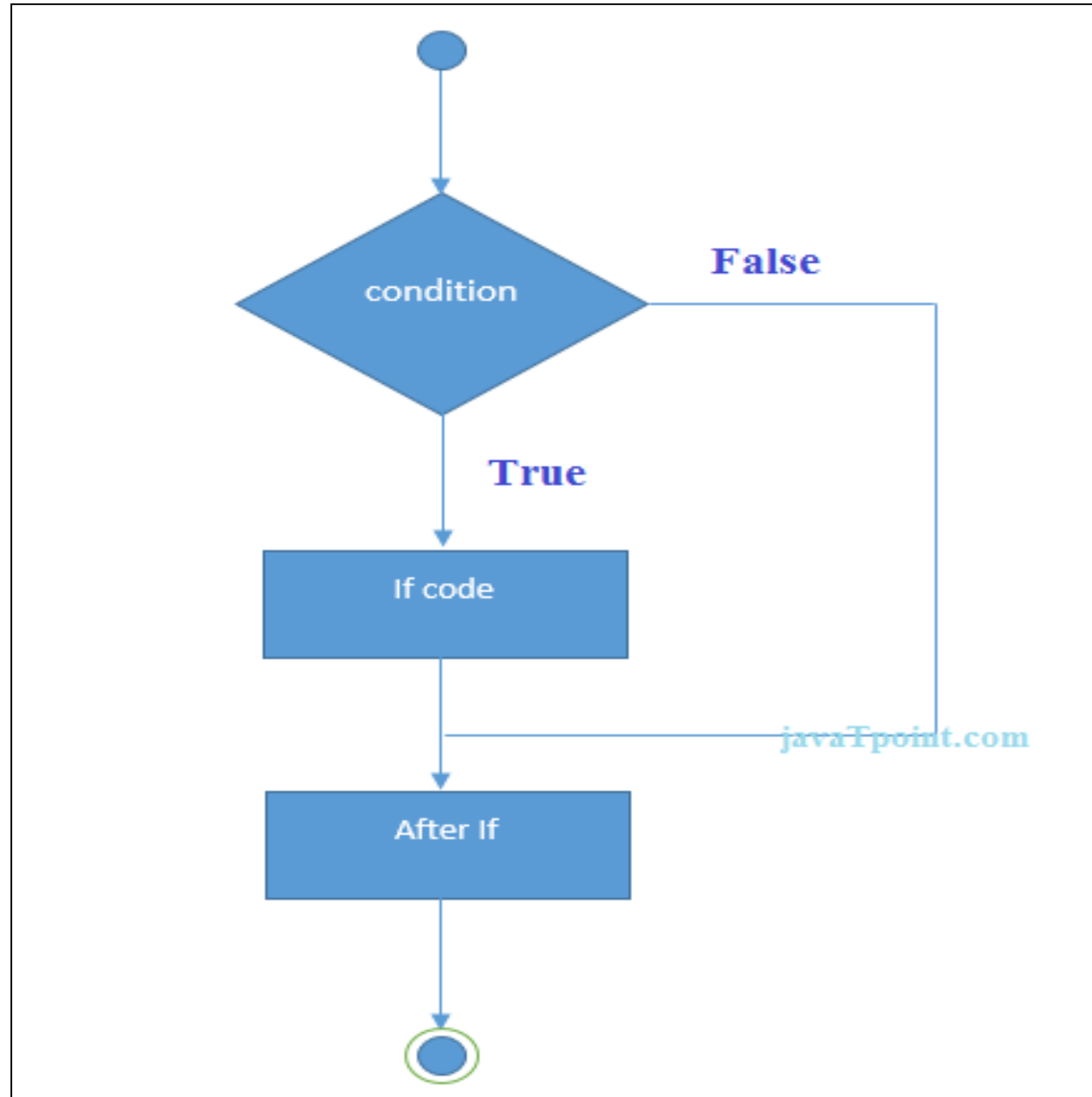


# CONDITIONAL STATEMENTS

# JAVA IF-ELSE STATEMENT

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

1. If Statement
2. If – else Statement
3. If – else ladder Statement
4. Nested if Statement





# JAVA IF STATEMENT

The Java if statement tests the condition. It executes the *if block* if condition is true.

## **Syntax:**

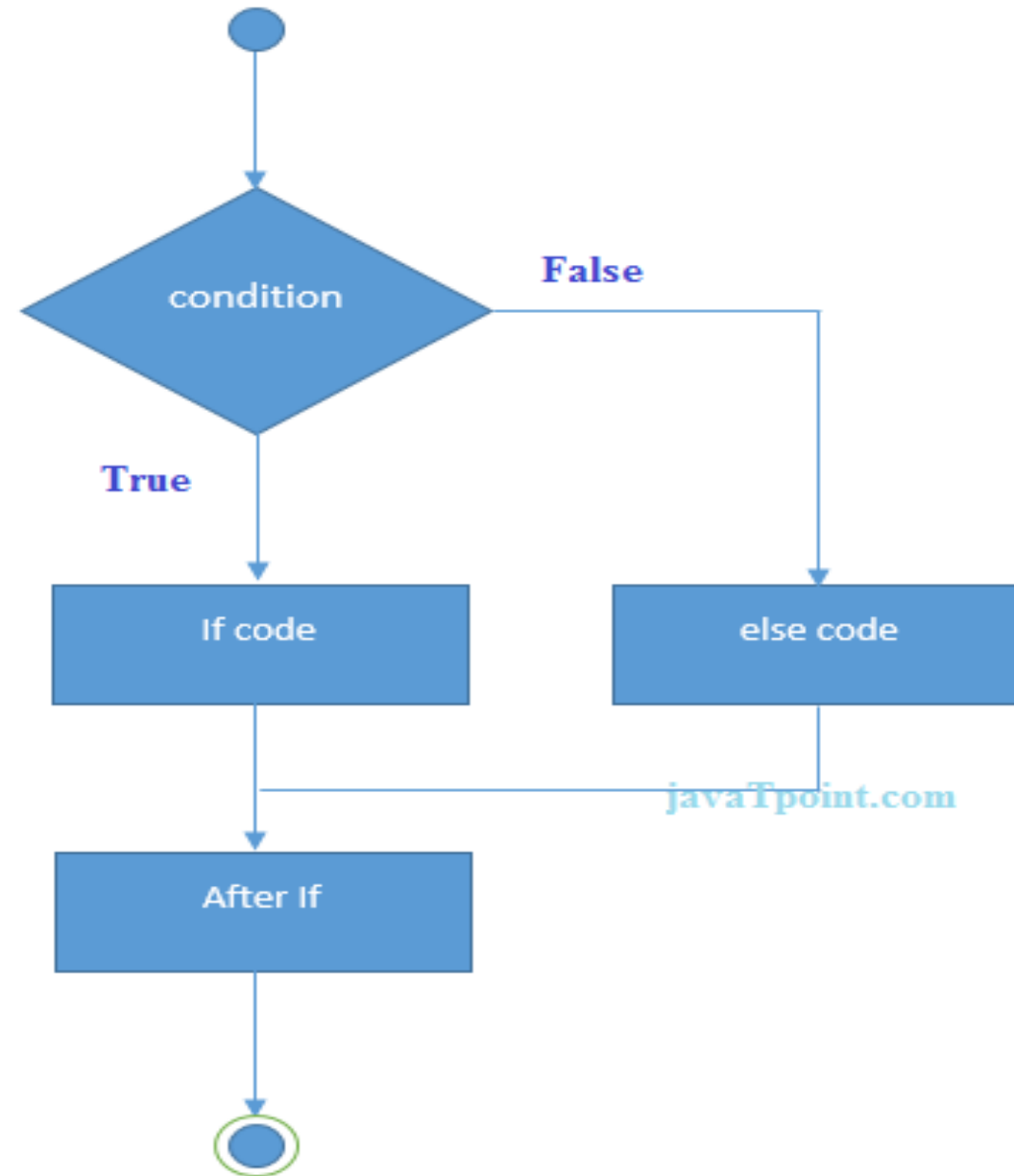
```
if(condition)
{
//code to be executed
}
```

# JAVA IF-ELSE STATEMENT

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

## **Syntax:**

```
if(condition) {  
    //code if condition is true  
} else {  
    //code if condition is false  
}
```



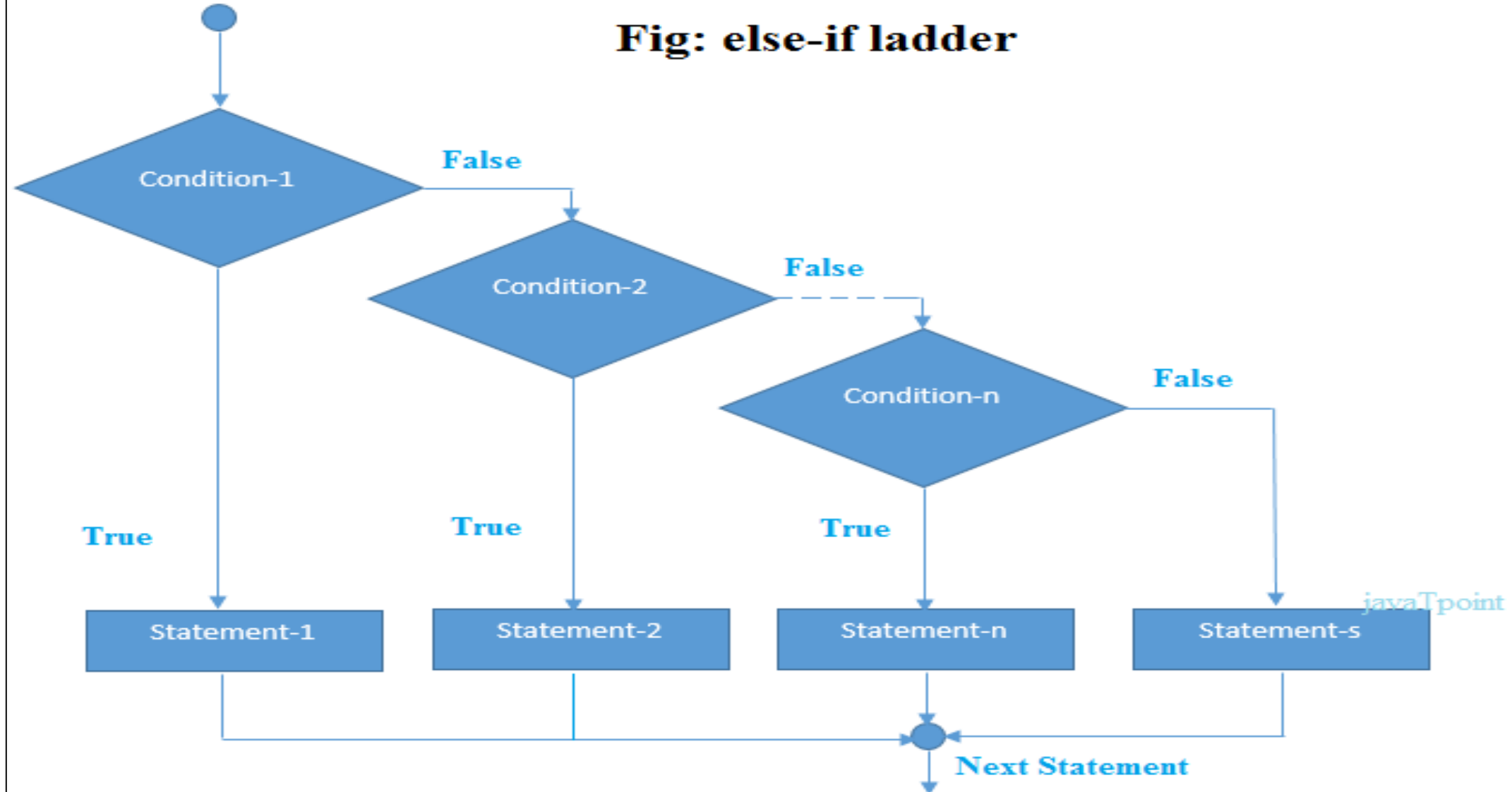
# JAVA IF-ELSE-IF LADDER STATEMENT

The if-else-if ladder statement executes one condition from multiple statements.

Syntax

```
If(condition 1) {  
}  
else if(condition 2){  
}  
else{  
}
```

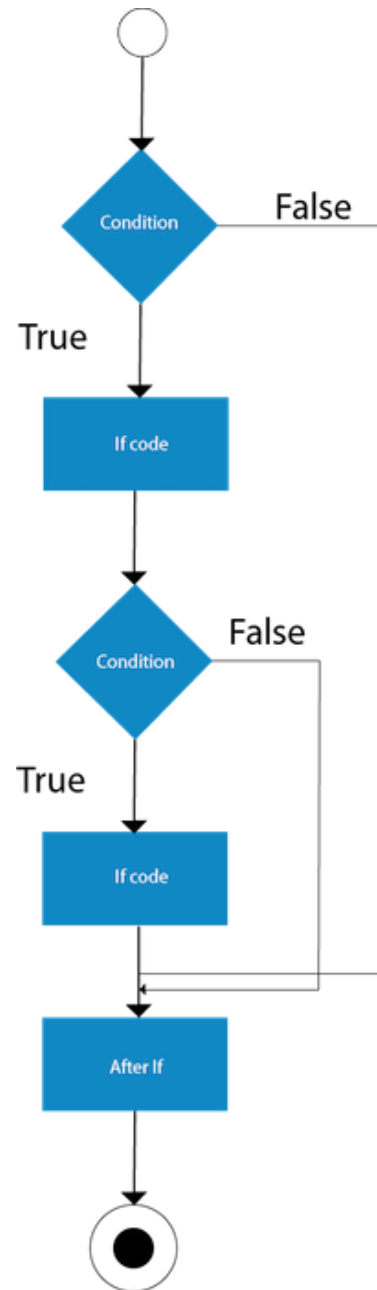
**Fig: else-if ladder**



# 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  
    }  
}
```



# JAVA SWITCH STATEMENT

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 case values must be *unique*. In case of duplicate value, it renders compile-time error.

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.



# LOOPING STRUCTURE IN JAVA

# LOOPS IN JAVA

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in java.

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

## for loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

## while loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

## do-while loop

The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Comparison	for loop	while loop	do while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Comparison	for loop	while loop	do while loop
Syntax	<pre>for(init;condition;incr/decr){ // code to be executed }</pre>	<pre>while(condition){ //code to be executed }</pre>	<pre>do{ //code to be executed }while(condition);</pre>
Example	<pre>//for loop for(int i=1;i&lt;=10;i++){ System.out.println(i); }</pre>	<pre>//while loop int i=1; while(i&lt;=10){ System.out.println(i); i++; }</pre>	<pre>//do-while loop int i=1; do{ System.out.println(i); i++; }while(i&lt;=10);</pre>
Syntax for infinitive loop	<pre>for( ; ; ){ //code to be executed }</pre>	<pre>while(true){ //code to be executed }</pre>	<pre>do{ //code to be executed }while(true);</pre>

# JAVA FOR LOOP

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loops in java.

1. Simple For Loop
2. For-each or Enhanced For Loop
3. Labeled For Loop



# JAVA SIMPLE FOR LOOP

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

**Initialization:** It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.

**Condition:** It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.

**Statement:** The statement of the loop is executed each time until the second condition is false.

**Increment/Decrement:** It increments or decrements the variable value. It is an optional condition.



## Syntax:

```
for(initialization;condition;incr/decr)
```

```
{
```

```
    //statement or code to be executed
```

```
}
```

# JAVA FOR-EACH LOOP

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.



## Syntax:

```
for(Type var:array)
```

```
{
```

```
    //code to be executed
```

```
}
```

# JAVA LABELED FOR LOOP

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Usually, break and continue keywords breaks/continues the innermost for loop only.



## Syntax:

labelname:

**for**(initialization;condition;incr/decr)

{

    //code to be executed

}

# JAVA INFINITIVE FOR LOOP

If you use two semicolons `;;` in the for loop, it will be infinitive for loop.

**Syntax:**

```
for( ; ; )
```

```
{
```

```
    //code to be executed
```

```
}
```

# JAVA WHILE LOOP

The Java *while* loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

## **Syntax:**

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



# JAVA INFINITIVE WHILE LOOP

If you pass **true** in the while loop, it will be infinitive while loop.

**Syntax:**

```
while(true){
```

```
//code to be executed
```

```
}
```

# JAVA DO-WHILE LOOP

The Java *do-while* loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while* loop is executed at least once because condition is checked after loop body.

## **Syntax:**

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



# LOOP CONTROL KEYWORDS

# JAVA BREAK STATEMENT

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

# JAVA CONTINUE STATEMENT


The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.

The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.



# JAVA COMMENTS



The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

# TYPES OF JAVA COMMENTS

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment



# Types of Java Comments

**01**

**Single Line**

The single line comment is used to comment only one line.

**02**

**Multi Line**

The multi line comment is used to comment multiple lines of code.

**03**

**Documentation**

The documentation comment is used to create documentation API. To create documentation API, you need to use javadoc tool.

# JAVA SINGLE LINE COMMENT

The single line comment is used to comment only one line.

## **Syntax:**

```
//This is single line comment
```

# JAVA MULTI LINE COMMENT

The multiline comment is used to comment multiple lines of code.

## **Syntax:**

```
/*
```

```
This is
```

```
multi line
```

```
comment
```

```
*/
```

# JAVA DOCUMENTATION COMMENT

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

## **Syntax:**

```
/**
```

```
This is
```

```
documentation
```

```
comment
```

```
*/
```



# JAVA DOCUMENTATION COMMENT EXAMPLE

/\*\* The Calculator class provides methods to get addition and subtraction of given 2 numbers.\*/

**public class** Calculator {

/\*\* The add() method returns addition of given numbers.\*/

**public static int** add(int a, int b){

**return** a+b;

}

/\*\* The sub() method returns subtraction of given numbers.\*/

**public static int** sub(int a, int b){

**return** a-b;

}

}



Compile it by javac tool:

```
javac Calculator.java
```

Create Documentation API by javadoc tool:

```
javadoc Calculator.java
```

Now, there will be HTML files created for your Calculator class in the current directory. Open the HTML files and see the explanation of Calculator class provided through documentation comment.



**CONTINUE IN NEXT UNIT.....**