Flavours of Java:

- J2SE (Standard Edition)
- J2EE (Enterprise Edition)
- J2ME (Micro Edition)

Java Program Syntax

```
class FirstJavaProgram {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

class -> Reserved keyword(we can't change like Class, cLaSS…etc)

FirstJavaProgram -> Class Name (we can named anything but meaningful like FirstProgram, MyFirstProgram, HelloWorldProgram … etc)

{ ….. } -> Represents block (It contains n number of statements)

public static void main(String[] args){..} -> Main method of java program. It is the starting point of your program.

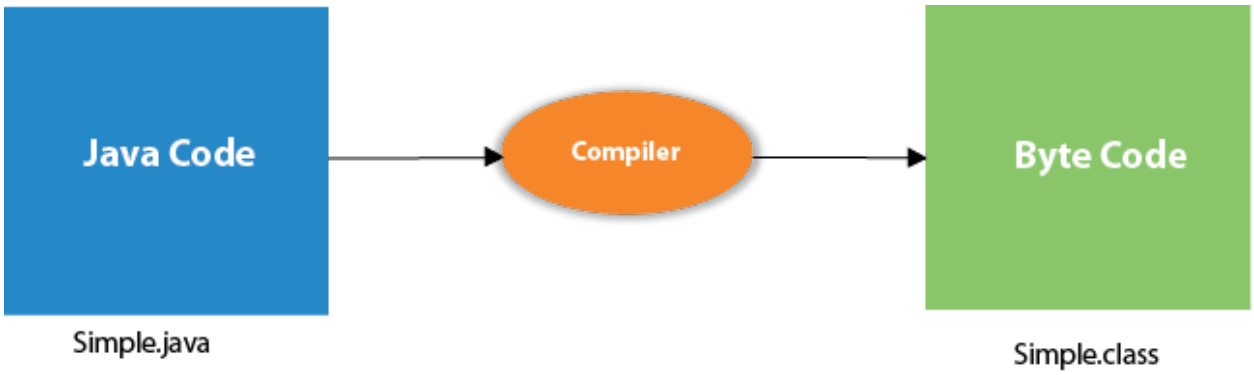
System.out.println(“hello”) -> It is a method used to print messages.

"Hello" -> This text will display on command prompt. Text must be within double quotes (**" "**)

Compilation & Execution

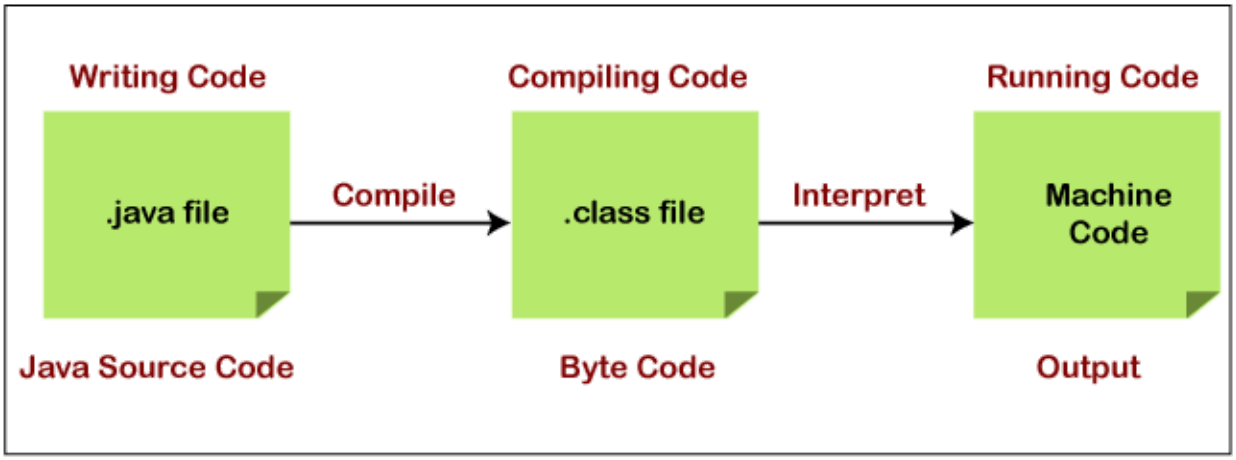
Compilation

In Java, programs are not compiled into executable files; they are compiled into bytecode. Java source code is compiled into bytecode when we use the javac compiler.



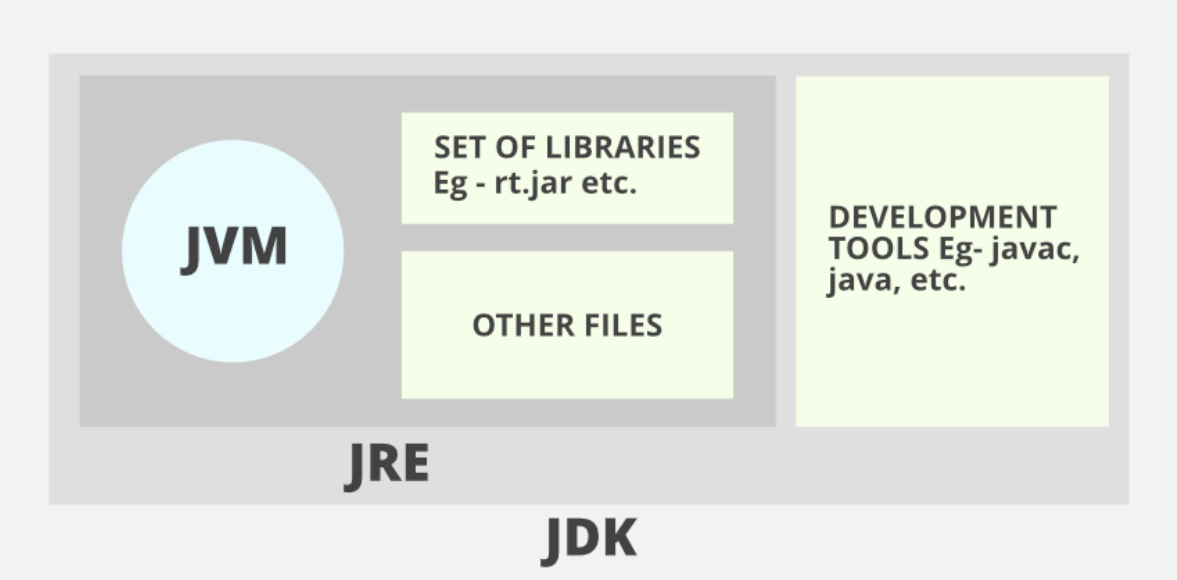
Execution

Interpreter in Java is a computer program that converts high-level program statements into Assembly Level Language.



JDK

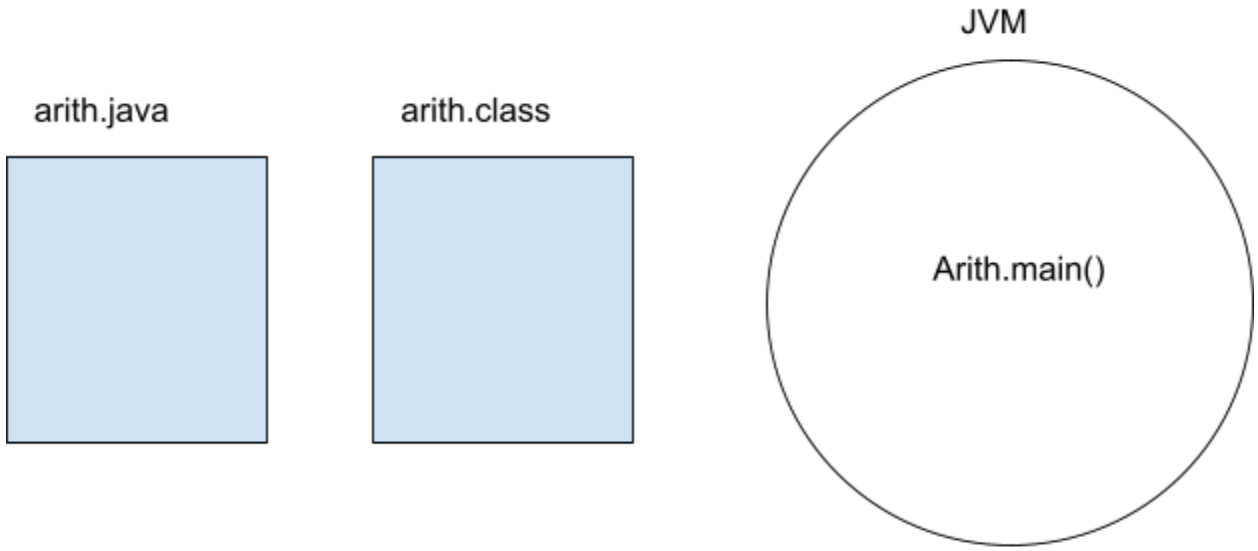
Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.



JRE (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

JVM (**Java Virtual Machine)** is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an **i*nterpreter*.**

JVM



High level language→ Java,

Low level language → Machine code(microprocessor 8085)

Features of Java

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Secure

Java is said to be more secure programming language because it does not have pointers concept, java provides a feature "applet" which can be embedded into a web application. The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

Portable

Portability is one of the core features of java which enables the java programs to run on any computer or operating system. For example, an applet developed using java runs on a wide variety of CPUs, operating systems, and browsers connected to the Internet.

Object-oriented

Java is said to be a pure object-oriented programming language. In java, everything is an object. It supports all the features of the object-oriented programming paradigm. The primitive data types java also implemented as objects using wrapper classes, but still, it allows primitive data types to archive high-performance.

Robust

Java is more robust because the java code can be executed on a variety of environments, java has a strong memory management mechanism (garbage collector), java is a strictly typed language, it has a strong set of exception handling mechanism, and many more.

Architecture-neutral (or) Platform Independent

Java has invented to archive "write once; run anywhere, any time, forever". The java provides JVM (Java Virtual Machine) to to archive architectural-neutral or platform-independent. The JVM allows the java program created using one

operating system can be executed on any other operating system.

Multi-threaded

Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

Interpreted

Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. The byte code is interpreted to any machine code so that it runs on the native machine.

High performance

Java provides high performance with the help of features like JVM, interpretation, and its simplicity.

Distributed

Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet. Java also supports Remote Method Invocation (RMI), this feature enables a program to invoke methods across a network.

Dynamic

Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

Lexical tokens

A token is the smallest element of a program that is meaningful to the compiler.

Keywords:

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

```
Abstract continue for new switch assert default
goto package synchronized boolean do if private
this break double implements protected
throw byte else import public Throws
case enum instanceof return transient
catch extends int short try char final
interface static void class finally long
strictfp volatile const float Native super while
```

Identifiers

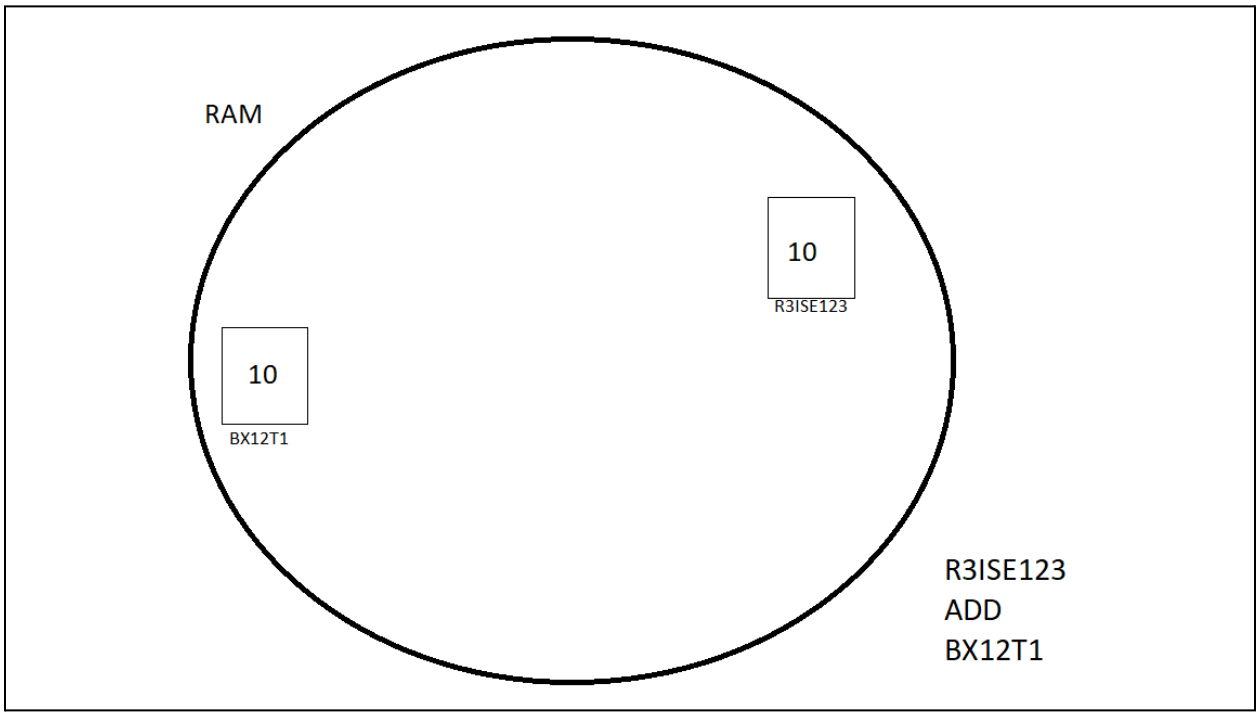
Identifiers are used as the general terminology for naming of variables, functions and arrays…etc

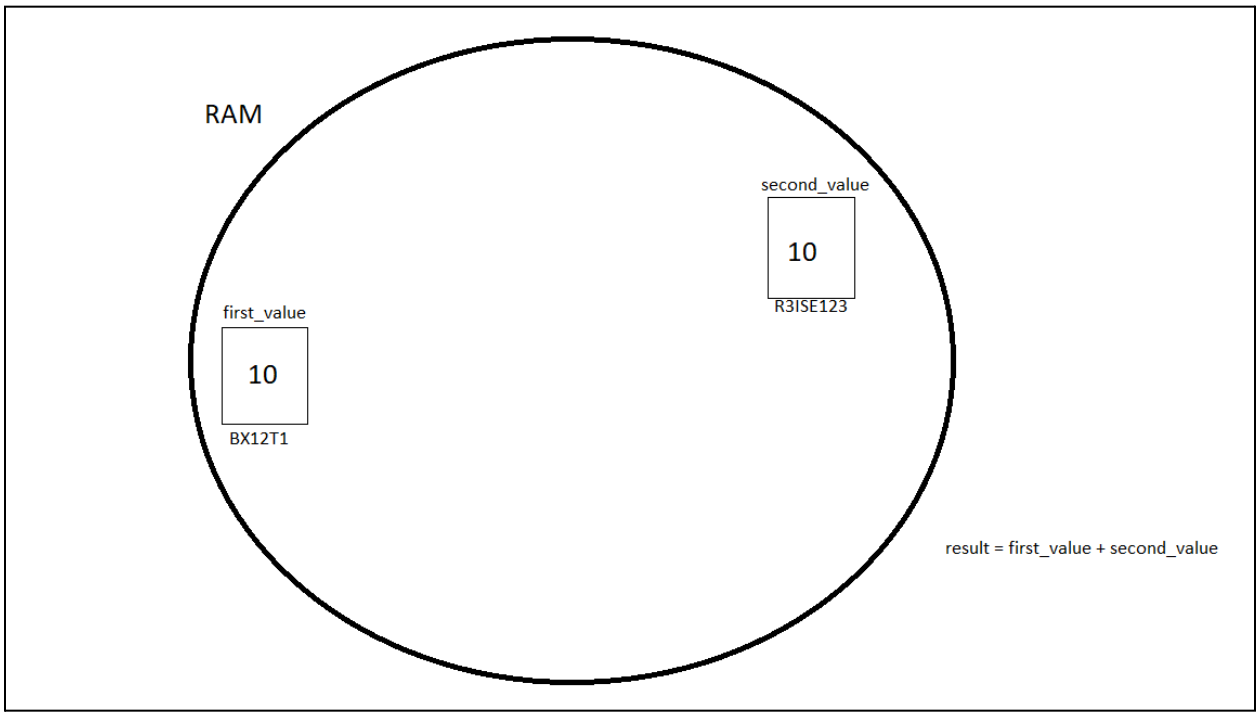
Rules for defining Java Identifiers:

- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), '**$**'(dollar sign) and '**_**' (underscore).For example "value@" is not a valid java identifier as it contain '@' special character.
- Identifiers should **not** start with digits(**[0-9]**). For example "123value" is a not a valid java identifier.
- **Reserved Words** can't be used as an identifier. For example "int class= 20;" is an invalid statement as while is a reserved word. There are **53** reserved words in Java.

Variable:

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location.





Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

Data types

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

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Primitive data types:

boolean - True or False

char - ('\u0000' (or 0) to '\uffff')

byte - (-128 to 127)

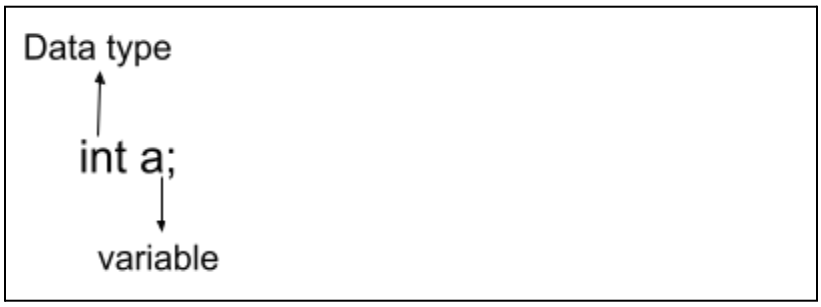
short - (-32,768 to 32,767)

int - (- 2,147,483,648 to 2,147,483,647)

long - (- 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)

float - 32-bit IEEE 754 floating point

double - 64-bit IEEE 754 floating point



Variable Declaration & Initialization

```
int a; //varible declaration

int a1 = 10; //variable declaration with initialization

float f; //varible declaration

float f1 = 3.5f; //variable declaration with initialization
```

Literal: Any constant value which can be assigned to the variable is called literal/constant. In simple words, Literals in Java is a synthetic representation of boolean, numeric, character, or string data.

```
// Here 100 is a constant/literal.
int x = 100;
```

Comments: The Java comments are the statements in a program that are not executed by the compiler and interpreter.

Single Line Comment

The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements. Single line comments starts with two forward slashes **(//)**. Any text in front of // is not executed by Java.

Multi Line Comment

The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time (as it will be difficult to use single-line comments there). Multi-line comments are placed between /* and */. Any text between /* and */ is not executed by Java.

Type casting:

In Java, type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.

int x=10;
 	
float y=x; // implicit conversion
 	
System.out.println(y);
	int x=10;
	
	float y=(float)x; // explicit conversion
	
	System.out.println(y);

Operators

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

1. **Arithmetic Operators**

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

2. **Assignment Operators**

Operator	Example	Equivalent to
`=`	`a = b;`	`a = b;`
`+=`	`a += b;`	`a = a + b;`
`-=`	`a -= b;`	`a = a - b;`
`*=`	`a *= b;`	`a = a * b;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`a %= b;`	`a = a % b;`

3. **Relational Operators**

Operator	Description	Example
`==`	Is Equal To	`3 == 5` returns **false**
`!=`	Not Equal To	`3 != 5` returns **true**
`>`	Greater Than	`3 > 5` returns **false**
`<`	Less Than	`3 < 5` returns **true**
`>=`	Greater Than or Equal To	`3 >= 5` returns **false**
`<=`	Less Than or Equal To	`3 <= 5` returns **true**

4. Logical Operators

Operator	Example	Meaning
`&&` (Logical AND)	expression1 **&&** expression2	`true` only if both `expression1` and `expression2` are `true`
`||` (Logical OR)	expression1 **||** expression2	`true` if either `expression1` or `expression2` is `true`
`!` (Logical NOT)	**!**expression	`true` if `expression` is `false` and vice versa

5. Unary Operators

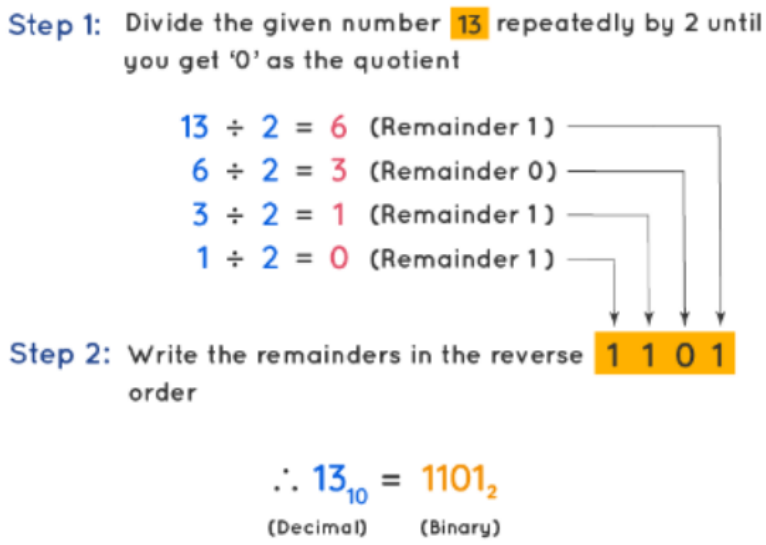
Operator	Meaning
`+`	**Unary plus**: not necessary to use since numbers are positive without using it
`-`	**Unary minus**: inverts the sign of an expression
`++`	**Increment operator**: increments value by 1
`--`	**Decrement operator**: decrements value by 1
`!`	**Logical complement operator**: inverts the value of a boolean

```
int i = 3;
int a = i++; // a = 3, i = 4
int b = ++a; // b = 4, a = 4
```

6. Bitwise Operators

Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

Decimal to Binary



Signed Left Shift Operator

int a = 8;

int b = a << 2;

a = 1 0 0 0

b = 1 0 0 0 0 0   (decimal -> 32)

<< Left Shift Operator appends two more zero. Increasing element counts.

Signed Right Shift Operator

int a = 8;

int b = a >> 2;

a = 1 0 0 0

b = 1 0 (decimal -> 2)

>> Right Shift Operator skip two bits. Increasing element counts.

Unsigned Right Shift Operator

int a = 240;

int b = a >>> 2;

a = 1 1 1 1 0 0 0 0

b = 0 0 1 1 1 1 0 0 (decimal -> 60)

>>> Right Shift Operator moved the right side number of bits. Shifted values are filled with 0. Excess bits are discarded

Looping and Control Statements (if - else)

if(condition)
{
//code if condition is true
}
else
{
//code if condition is false
}
	if(condition1){
	//code to be executed if condition1 is true
	}**else if**(condition2){
	//code to be executed if condition2 is true
	}
	else if(condition3){
	//code to be executed if condition3 is true
	}
	...
	else{
	//code to be executed if all the conditions are false
	}

for loop

```
for(initialization; condition; increment/decrement){
//statement or code to be executed
}
```

while loop

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

do while loop

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

break, continue

```
jump-statement;
break;
```

```
jump-statement;
continue;
```

switch case

```
switch(expression){
case value1:
 //code to be executed;
 break;
case value2:
 //code to be executed;
 break;
.....
default:
 code to be executed if all cases are not matched;
}
```

Ternary Operator(? :)

variable = (condition) ? expression1 : expression2

Methods

A method is a block of code or collection of statements to perform a certain task or operation.

It is used to achieve the **reusability of code**. We write a method once and use it many times.

It also provides the easy modification and readability of code.

The method is executed only when we **call** or invoke it.

```
<access_specifier>  <return_type>  method_name(parameters)

{

// method definition

}
```

---

1. Java Program to Swap Two Numbers
2. Java Program to Check Whether a Number is Even or Odd
3. Java Program to Check Whether a Number is Perfect or Not Perfect
4. Java Program to Calculate the Sum of Natural Numbers
5. Java Program to Generate Multiplication Table
6. Java Program to Reverse a Number
7. Java Program to Display Prime Numbers Between Two Intervals
8. Java Program to Display ODD numbers Between Two Intervals

9. Java Program to Display EVEN numbers Between Two Intervals

10. Java Program to Find Factorial of a Number Using Recursion