JAVA PROGRAMMING

JAVA INTRODUCTION:

- ➔ Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.
- ➔ Java was developed by Sun Microsystems in the year 1995.

Hello World Program:

```
Class Sample
{
Public static void main(String args[])
{
System.out.println("Hello World");
}
}
```

Features of java:

1. Simple
2. Portable
3. Object oriented
4. Platform independent
5. Secured
6. Dynamic
7. High performance
8. Multi threaded

Basic Concepts of OOP:

1. Class
2. Object
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Basic Programming:

```
Class Sample
{
Public static void main(String args[])
{
System.out.println("Hello World");
}
}
```

To Compile:javac filename.java

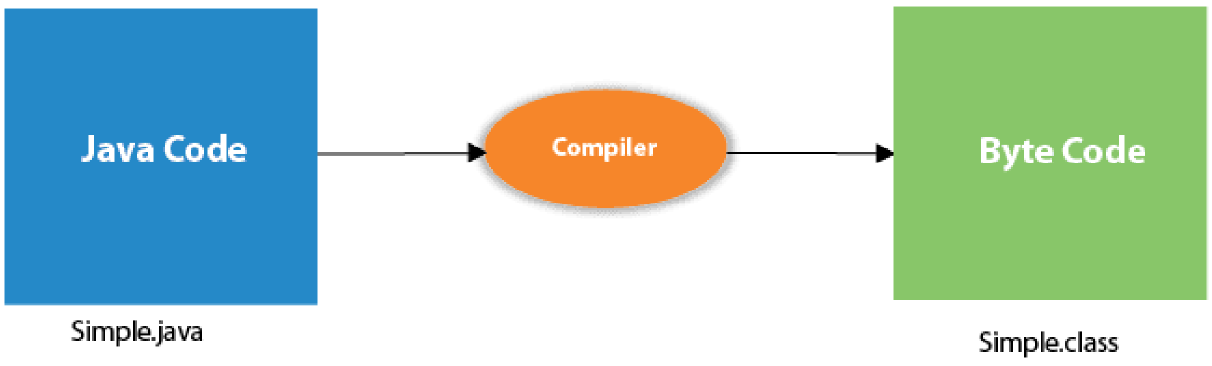
To Run:java filename

Terms in program:

- **Class** keyword is used to define a class in java.
- **Public** is an access modifier which represents visibility.It is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used to print statements. Here, System is a class, out is the object of PrintStream class, println() is the method of PrintStream class.

Internal Work:

At compile time, a java file is compiled by Java Compiler (It does not interact with OS) and converts the java code into bytecode.



Variables in Java:

- ➔ Variable is a name of a memory location.
- ➔ Variables are of three types.
    1. Local variables
    2. Static variables
    3. Instance Variables

1.Local variables:

- ➔ A variable declared inside the body of the method is called a local variable.
- ➔ A local variable cannot be defined with a "static" keyword.

2.Instance variables:

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

Program:

```
 Class Variables
{
int data=10;  //Instance variable
static int m=100; //Static variable
public static void main(String args[])
{
```

```
int n=200; /Local Variable
}
}
```

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:

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.

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

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.

Byte Data Type:

- ➔ The byte data type is an example of primitive data type.
- ➔ Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

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.

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.

Long Data Type:

➔ The long data type is a 64-bit two's complement integer.

➔ The default value is 0.

➔ Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807.

Float data type:

➔ Float is used to store numbers with decimal parts.

➔ These are two floating point data types in Java namely, the float (32 bits, single precision) and the double(64 bits, double precision).

Character Data Type:

➔ It is used for storing individual characters.

➔ The char type has 16 bits of precision and it is unsigned.

Operators in Java:

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

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

➔ Unary Operator,

- ➔ Arithmetic Operator,
- ➔ Shift Operator,
- ➔ Relational Operator,
- ➔ Bitwise Operator,
- ➔ Logical Operator,
- ➔ Ternary Operator and
- ➔ Assignment Operator.

Java Operator Precedence:

Operator Type	**Category**	**Precedence**
Unary	postfix	*expr*++, *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator:

- ➔ The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:
    1. incrementing/decrementing a value by one
    2. negating an expression
    3. inverting the value of a boolean

Java Arithmetic Operators:

- ➔ Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.
- ➔ They act as basic mathematical operations.

Java Left Shift Operator:

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Right Shift Operator:

The Java right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Java AND Operator Example: Logical && and Bitwise &:

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether the first condition is true or false.

Java OR Operator Example: Logical || and Bitwise | :

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

Java Assignment Operator:

Java assignment operators are one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Ternary Operator:

- ➔ Java Ternary operator is used as one liner replacement for if-then-else statements and used a lot in Java programming.
- ➔ It is the only conditional operator which takes three operands.

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 statements in Java.

- if statement
- if-else statement
- if-else-if ladder
- 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(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2)
{
//code to be executed if condition 2 is true
}
else if(condition3)
{
//code to be executed if condition 3 is true
}
...
else
{
//code to be executed if all the conditions are false
}
```

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.

Syntax:

```
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.

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 , enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches 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 value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

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.

Java Nested Switch Statement

We can use switch statements inside other switch statements in Java. It is known as a nested switch statement.

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.

- for loop
- while loop
- do-while loop

Java For Loop:

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

There are three types of for loops in java.

- Simple For Loop
- For-each or Enhanced For Loop
- 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:

1. **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.
2. **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.

3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **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 Nested For Loop:

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever the outer loop executes.

Java for-each Loop:

The for-each loop is used to traverse arrays or collections 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 an element basis not index. It returns elements 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 for each Java **for loop**. To do so, we use a 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/continue 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 iterations is not fixed, it is recommended to use a while loop.

Syntax:

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

Java Infinite 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 iterations is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.

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

Syntax:

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

Java Infinitive do-while Loop:

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

Syntax:

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

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.

Syntax:

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

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.

Syntax:

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