JAVA

1. What is Java

Java is a **programming language**.

Java is a high level, robust, secured and object-oriented programming language.

Where it is used?

According to Sun, 3 billion devices run java. There are many devices where java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Types of Java Applications

There are mainly 4 type of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

2. Simple Program of Java

Example:

```
public class MyFirstJavaProgram {

  /* This is my first java program.

   This will print 'Hello World' as the output

   */

   public static void main(String []args) {

      System.out.println("Hello World"); // prints Hello World

   }

}
```

Understanding first java program

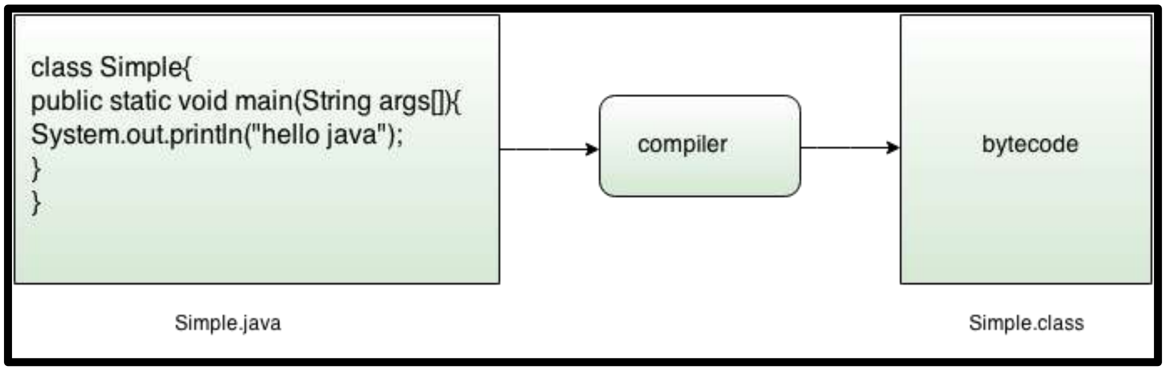
Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

Internal Details of Hello Java Program

What happens at compile time?

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


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

Basic Syntax:

- **Case Sensitivity -** Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names -** For all class names the first letter should be in Upper Case.

  If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

Example *class MyFirstJavaClass*

- **Method Names -** All method names should start with a Lower Case letter.

  If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example *public void myMethodName()*

- **Program File Name -** Name of the program file should exactly match the class name.

  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match your program will not compile).

  Example : Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

- **public static void main(String args[]) -** Java program processing starts from the main() method which is a mandatory part of every Java program..

Comments in Java:

```
public class MyFirstJavaProgram{


  /* This is my first java program.

   This will print 'Hello World' as the output
```

```
     This is an example of multi-line comments.
    */

    public static void main(String []args){
      // This is an example of single line comment
      /* This is also an example of single line comment. */
      System.out.println("Hello World");
    }
}
```

Java Literals:

"Hello World" String

"two \n lines"

Notation	Character represented
\n	Newline (0x0a)
\b	Backspace (0x08)
\t	Tab

\"	Double quote
\'	Single quote
\\	Backslash

Variable and Datatype in Java

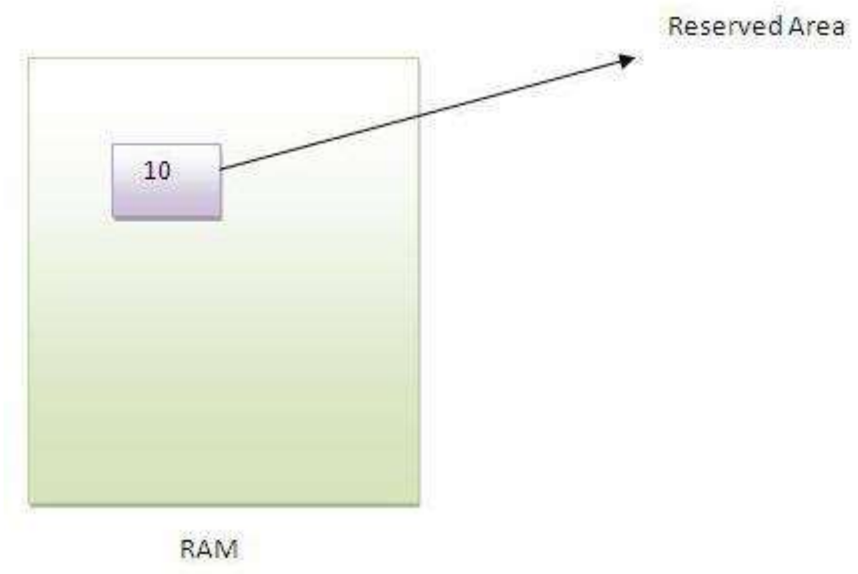
Syntax:

data type variable [= value][, variable [= value] ...] ;

Variable is a name of memory location.

Variable

Variable is name of reserved area allocated in memory.

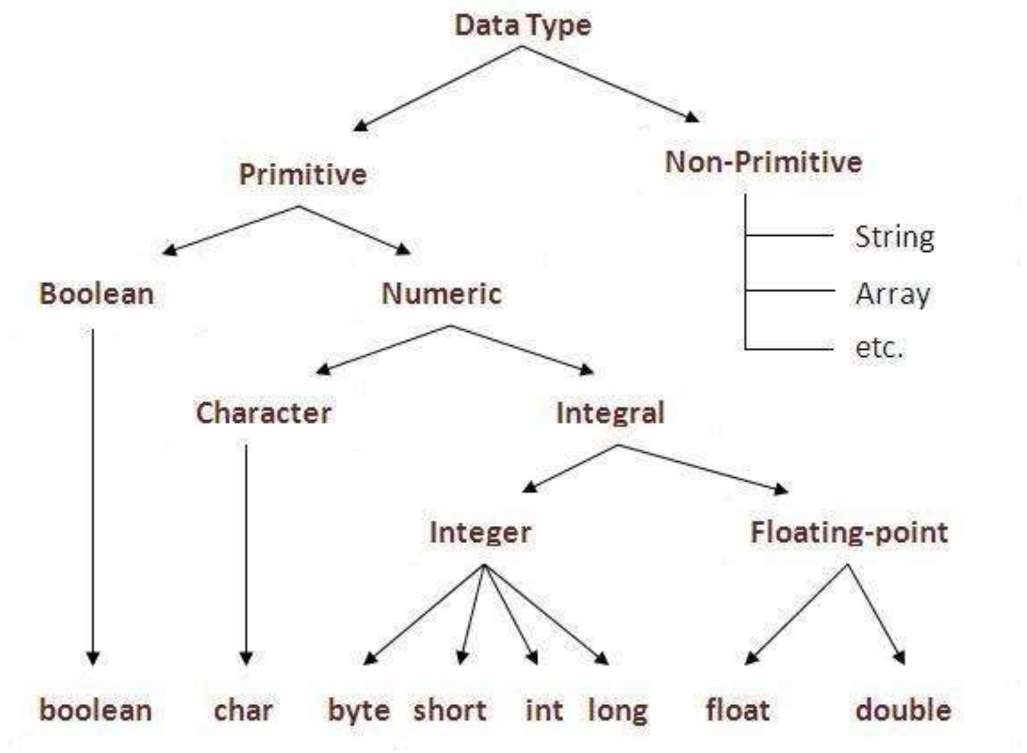


```
int data=10;//Here data is variable
int a, b, c;         // Declares three ints, a, b, and c.

a=100;  // Assigning value to a variable

b=200;

int a = 10, b = 10;  // Example of initialization

byte B = 22;         // initializes a byte type variable B.

double pi = 3.14159; // declares and assigns a value of PI.

char a = 'a';        // the char variable a iis initialized with value 'a'
```

Data Types in Java

In java, there are two types of data types

- primitive data types
- non-primitive data types



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

JAVA Operators:

The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

Assume integer variable A holds 10 and variable B holds 20, then:

A=10,B=20

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0

++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

```java
public class SumOfTwoNumbers {
     public static void main(String[] args) {
          int a=2;
          int b=3;
          int sum;
          sum=a+b;
          System.out.println("sum");
          System.out.println(sum);
          System.out.println("sum="+sum);
     }
}
```

B=20; Value 20 is assigned to variable B

B++ B=B+1

B-- B=B-1

C=B++; Post increment c=20

Print c; 20

C=++B; Pre increment

C=++B; C=21

b=20
b++; ==> b=b+1; (Post inc) b=20

b=21;
b=20;

```
++b ==> b=b+1; (Pre Incr Op) b=21

b=20;
b--; ==> b=b-1;  Post Dec Op, b=20
            b=19;
b=20;
--b; ==> b=b-1; Pre decr Op b=19
```

The Relational Operators:

There are following relational operators supported by Java language

Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is false.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is false.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is false.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true

The Logical Operators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

Operator	**Description**	**Example**
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

The Assignment Operators:

There are following assignment operators supported by Java language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Java - Decision Making (Conditional Statements)

There are two types of decision making statements in Java. They are:

- if statements
- switch statements

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements.

Syntax:

The syntax of an if statement is:

```
if(Boolean_expression/Condition)

{

  //Statements will execute if the Boolean expression is true

}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Example:

```
public class SimpleIf {


  public static void main(String args[]){

    int x = 10;


    if( x < 20 ){

      System.out.print("This is if statement");
      }
   }
}
```

This would produce the following result:

```
This is if statement
```

The if...else Statement:

An if statement can be followed by an optional *else* statement, which executes when the Boolean expression is false.

Syntax:

The syntax of an if...else is:

```
if(Boolean_expression){
   //Executes when the Boolean expression is true
}else{
   //Executes when the Boolean expression is false
}
```

Example:

```
public class IfElse {
  public static void main(String args[]){
    int x = 30;

    if( x < 20 ){
      System.out.print("This is if statement");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

This would produce the following result:

```
This is else statement
```

The if...else if...else Statement:

An if statement can be followed by an optional *else if...else* statement, which is very useful to test various conditions using single if...else if statement.

When using if , else if , else statements there are few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax:

The syntax of an if...else is:

```
if(Boolean_expression 1){
  //Executes when the Boolean expression 1 is true
}else if(Boolean_expression 2){
  //Executes when the Boolean expression 2 is true
}else if(Boolean_expression 3){
  //Executes when the Boolean expression 3 is true
}else {
  //Executes when the none of the above condition is true.
}
```

Example:

```
public class IfElseIfElse {

  public static void main(String args[]){
    int x = 30;

    if( x == 10 ){
      System.out.print("Value of X is 10");
    }else if( x == 20 ){
      System.out.print("Value of X is 20");
    }else if( x == 30 ){
      System.out.print("Value of X is 30");
    }else{
      System.out.print("This is else statement");
    }
  }
}
```

This would produce the following result:

```
Value of X is 30
```

Nested if...else Statement:

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

Syntax:

The syntax for a nested if...else is as follows:

```
if(Boolean_expression 1){
  //Executes when the Boolean expression 1 is true
  if(Boolean_expression 2){
    //Executes when the Boolean expression 2 is true
  }
}
```

You can nest *else if...else* in the similar way as we have nested *if* statement.

Example:

```
public class NestedIf {


  public static void main(String args[]){
    int x = 30;
    int y = 10;


    if( x == 30 ){
      if( y == 10 ){
         System.out.print("X = 30 and Y = 10");
       }
     }
  }
}
```

This would produce the following result:

```
X = 30 and Y = 10
```

The switch Statement:

A *switch* statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

The syntax of enhanced for loop is:

```
switch(expression){
   case value :
     //Statements
     break; //optional
   case value :
     //Statements
     break; //optional
   //You can have any number of case statements.
   default : //Optional
     //Statements
}
```

The following rules apply to a switch statement:

- The variable used in a switch statement can only be a byte, short, int, or char.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
- When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will *fall through* to subsequent cases until a break is reached.

- A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

Example:

```
public class SwitchEx1 {
  public static void main(String args[]){
     char grade = 'C';
    switch(grade)
    {
      case 'A' :
        System.out.println("Excellent!");
        break;
      case 'B' :
      case 'C' :
        System.out.println("Well done");
        break;
      case 'D' :
        System.out.println("You passed");
      case 'F' :
        System.out.println("Better try again");
        break;
      default :
        System.out.println("Invalid grade");
    }
    System.out.println("Your grade is " + grade);
  }
}
```

Java - Loop Control

When we need to execute a block of code several number of times, and is often referred to as a loop.

Java has very flexible three looping mechanisms. You can use one of the following three loops:

- while Loop
- do...while Loop
- for Loop

The while Loop:

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax:

The syntax of a while loop is:

```
while(Boolean_expression)
```

```
{
  //Statements
}
```

When executing, if the *boolean_expression* result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

Here, key point of the *while* loop is that the loop might not ever run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

```
public class WhileLoop {


  public static void main(String args[]) {
    int x = 10;


    while( x < 20 ) {
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
    }
  }
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19
```

The do...while Loop:

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

The syntax of a do...while loop is:

```
do

{

  //Statements

}while(Boolean_expression);
```

Notice that the Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested.

If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

Example:

```
public class DoWhileLoop {


  public static void main(String args[]){
    int x = 10;


    do{
      System.out.print("value of x : " + x );
      x++;
      System.out.print("\n");
    }while( x < 20 );
  }
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

The for Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

A for loop is useful when you know how many times a task is to be repeated.

Syntax:

The syntax of a for loop is:

```
for(initialization; Boolean_expression; update)
{
  //Statements
}
```

Here is the flow of control in a for loop:

- The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.
- Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement past the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the Boolean expression.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

Example:

```
public class ForLoop {


  public static void main(String args[]) {


    for(int x = 10; x < 20; x = x+1) {
      System.out.print("value of x : " + x );
      System.out.print("\n");
    }
  }
}
```

This would produce the following result:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

The break Keyword:

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop:

```
break;
```

The continue Keyword:

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:

The syntax of a continue is a single statement inside any loop:

```
continue;
```