28-01-2025

Tokens in java :

Token is the smallest unit of the program which is identified by the compiler.

Without token we can't complete statement or expression in java.

Token is divided into 5 types in java

1) Keyword
2) Identifier
3) Literal
4) Punctuators (Seperators)
5) Operator

Keyword

A keyword is a predefined word whose meaning is already defined by the compiler.

In java all the keywords must be in lowercase only.

A keyword we can't use as a name of the variable, name of the class or name of the method.

true, false and null look like keywords but actually they are literals.

As of now, we have 67+ keywords in java.

Identifiers :

A name in java program by default considered as identifiers.

Assigned to variable, method, classes to uniquely identify them.

We can't use keyword as an identifier.

Ex:-

```
class Fan
{
   int coil  ;

   void switchOn()
```

```
  {
  }
}
```

Here Fan(Name of the class), coil (Name of the variable) and switchOn(Name of the Method) are identifiers.

Rules for defining an identifier :
---
1) Can consist of uppercase(A-Z), lowercase(a-z), digits(0-9), $ sign, and underscore (_)
2) Begins with letter, $, and _
3) It is case sensitive
4) Cannot be a keyword
5) No limitation of length

Literals :

Any constant which we are assiging to variable is called Literal.

In java we have 5 types of Literals :

1) Integral Literal
2) Floating Point Literal
3) Boolean Literal
4) Character Literal
5) String Literal

Note : null is also a literal

Integral Literal :

If a numeric literal does not contain any decimal or fraction then it is called Integral Literal.

Example : 90, 23, 67, 1234

In Integral literal we have total 4 data types :

a) byte (8 bits)
b) short (16 bits)
c) int (32 bits)
d) long (64 bits)

An integral literal we can represent in four different forms :

a) Decimal Literal (Base is 10)
b) Octal Literal (Base is 8)
c) Hexadecimal Literal (Base is 16 [0-9 and a - f])
d) Binary Literal (Base 2) [Available from JDK 1.7v]

Decimal Literal :

By default our numeric literals are decimal literal. Here base is 10 so, It accepts 10 digits i.e. from 0-9.

Example :

```
int x = 20;
int y = 123;
int z = 234;
```

Octal Literal :

If any Integer literal starts with 0 (Zero) then it will become octal literal. Here base is 8 so it will accept 8 digits i.e 0 to 7.

Example :

```
int a = 018; //Invalid becuase it contains digit 8 which is out of range
int b = 0777; //Valid
int c = 0123; //Valid
```

Hexadecimal Literal :

If any integric literal starts with 0X or 0x (Zero capital X Or 0 small x) then it is hexadecimal literal. Here base is 16 so it will accept 16 digits i.e 0 to 9 and A to F OR [a to f]

Example :

```
int x = 0X12; //Valid
int y = 0xadd; //Valid
int z = 0Xface; //valid
int a = 0Xage; //Inavlid because character 'g' out of range
```

Binary Literal :

If a numeric literal starts with 0B (Zero capital B) or 0b (Zero small b) then it will become Binary literal. Binary literal is available from JDK 1.7v.

Here base is 2 so it will accept 2 digits i.e 0 and 1.

Example :
---
int x = 0B101; //valid
int y = 0b111; //Valid
int z = 0B112; //Invalid [2 is out of range]

The deafult type is decimal literal so to generate the output for any different literal JVM converts into decimal literal.

29-01-2025

//Program on Octal Literal

```
public class Test
{
	public static void main(String[] args)
	{
		int x = 015;
		System.out.println(x); //13
	}
}
```

```
public class Test2
{
	public static void main(String[] args)
	{
		int x = 0Xadd;
		System.out.println(x);
	}
}
```

```
public class Test3
{
	public static void main(String[] args)
	{
		int x = 0b101;
		System.out.println(x);
	}
}
```

By default every integral literal is of type int only. byte and short are below than int so we can

assign integral literal(Which is by default int type) to byte and short but the values must be within the range. [for Byte -128 to 127 and for short -32768 to 32767]

Actually whenever we are assigning integral literal to byte and short data type then compiler internally converts into corresponding type.

byte b = (byte) 12; [Compiler is converting int to byte]
short s = (short) 12; [Compiler is converting int to short]

In order to represent long value explicitly we should use either L OR l (Capital L OR Small l) as a suffix to integral literal.

According to IT industry, we should use L bacause l looks like digit 1.

```
/* By default every integral literal is of type int only*/
public class Test4
{
public static void main(String[] args)
        {
            byte b = 128; //error
                System.out.println(b);

                short s = 32768; //error
                System.out.println(s);
   }
}
```

```
//Assigning smaller data type value to bigger data type
public class Test5
{
public static void main(String[] args)
        {
            byte b = 125;
                short s = b;  //Automatic type casting OR Widening
                System.out.println(s);

                short x = 345;
                int y = x;  //Automatic type casting OR Widening
                System.out.println(y);
        }
}
```

```
//Converting bigger type to smaller type
public class Test6
{
public static void main(String[] args)
        {
            short s = 127;
                byte b = (byte) s;
                System.out.println(b); //127
        }
}
```

```
public class Test7
 {
 public static void main(String[] args)
 {
                byte x = (byte) 1L;
                System.out.println("x value  = "+x);

                long l = 29L;
                System.out.println("l value  = "+l);

       int y = (int) 18L;
                System.out.println("y value  = "+y);

  }
 }
```

30-01-2025

Is java pure Object Oriented language or not ?

No, Java is not a pure object oriented langauge because it is accepting primary data type, Actually any language which accepts primary data type is not a pure object oriented language.

Only Objects are moving in the network but not the primary data type so java has introduced Wrapper class concept to convert the primary data types into corresponding wrapper object.

Primary Data type		Wrapper Object
byte	:	Byte
short	:	Short
int	:	Integer
long	:	Long
float	:	Float

double : Double
char : Character
boolean : Boolean

Note : Apart from these 8 data types, Everything is an object in java so, if we remove all these 8 primitive data types then java will become pure OOP language.

```
//Wrapper claases
public class Test8
{
	public static void main(String[] args)
	{
		Integer x = 24;
		Integer y = 24;
		Integer z = x + y;
		System.out.println("The sum is :"+z);

		Boolean b = true;
		System.out.println(b);

		Double d = 90.90;
		System.out.println(d);

		Character c = 'A';
		System.out.println(c);
	}
}
```

Note : From JAVA 1.5 version we have two conecpt :

AutoBoxing : Converting Primitive to Wrapper Object (int to Integer)
UnBoxing : Converting Wrapper object back to primitive (Integer to int)

How to find out the minimum value, maximum value as well as size of different data types :

The Warpper classes like Byte, Short, Integer and Long has provided predefined static and final variables to represent minimum value, maximum value as well as size of the respective data type.

Example :

If we want to get the minimum value, maximum value as well as size of byte data type then Byte

class (Wrapper class) has provided the following final and static variables

Byte.MIN_VALUE : -128

Byte.MAX_VALUE : 127

Byte.SIZE : 8 (bits format)

```
//Program to find out the range and size of Integeral Data type
public class Test9
{
	public static void main(String[] args)
	{
		System.out.println("\n Byte range:");
		System.out.println(" min: " + Byte.MIN_VALUE);
		System.out.println(" max: " + Byte.MAX_VALUE);
		System.out.println(" size :"+Byte.SIZE);

		System.out.println("\n Short range:");
		System.out.println(" min: " + Short.MIN_VALUE);
		System.out.println(" max: " + Short.MAX_VALUE);
		System.out.println(" size :"+Short.SIZE);

		System.out.println("\n Integer range:");
		System.out.println(" min: " + Integer.MIN_VALUE);
		System.out.println(" max: " + Integer.MAX_VALUE);
		System.out.println(" size :"+Integer.SIZE);

		System.out.println("\n Long range:");
		System.out.println(" min: " + Long.MIN_VALUE);
		System.out.println(" max: " + Long.MAX_VALUE);
		System.out.println(" size :"+Long.SIZE);

	}
}
```

Writing _ symbol in Integral Literal :

Java software people has provided to write _ symbol facility while writing the big integral literal numbers so the readability of the number will enhance.

Example : 98_7854_3986

We can't start or end an integral literal with _ we will get compilation error.

```
//We can provide _ in integral literal [JDK 1.7]
public class Test10
{
	public static void main(String[] args)
	{
	   long mobile = 98_1234_5678L;
		System.out.println("Mobile Number is :"+mobile);
	}
}
```

```
public class Test11
{
	public static void main(String[] args)
	{
		final int x = 127;
		byte b = x;
		System.out.println(b);
	}
}
```

Note : Program will compile and run.

Integer class has provided the following static methods to convert decimal to octal, hexadecimal and binary.

1) public static String toBinaryString(int x) : Will convert the decimal into binary in String format.

2) public static String toOctalString(int x) : Will convert the decimal into octal in String format.

3) public static String toHexString(int x) : Will convert the decimal into hexadecimal in String format.

Note : All the above static methods are available in Integer class.

```
class Test
{
	public static void main(String[] args)
      {
	   int x = 15;
	   //Decimal to octal
```

```
        System.out.println(Integer.toOctalString(x));

        int y = 2781;
        //decimal to hexadecimal
        System.out.println(Integer.toHexString(y));

        int z = 5;
        //decimal to Binary
        System.out.println(Integer.toBinaryString(z));

    }
}
```

---

var keyword in java :

Java has provided var keyword from java 10V. By using var keyword we can directly assign any type of value to that particular variable.

Using var keyword, Initialization is compulsory in the same line (Compiler will get the information of variable type based on the value)

var keyword is allowed only for local variable.

```
//var keyword [Introduced from java 10]
public class Test13
{
    public static void main(String[] args)
    {
        var x = 100; //Initialization is compulsory here only

        //x = "NIT";  //Invalid

        x = 900;

        System.out.println(x);

    }
}
```

31-01-2025

Floating Point Literal :

If any numeric literal contains decimal or fraction then it is called Floating Point Literal.
Example : 45.9, 7.8, 4.5

In floating point literal we have 2 data types :
1) float (32 bits)
2) double (64 bits)

By default every floating point literal is of type double only so, the following statement will generate compilation error

```
float f = 1.2; //Error
```

In order to represent double value as a float value we can write the following statements :

```
float f1 = (float) 12.90;
float f2 = 12.66F;
float f3 = 56.89f;
```

As we know by default every floating point literal is of type double only but still compiler has provided two flavors to represent the double value explicitly.

```
double d1 = 10d;
double d2 = 18D;
```

Floating point literal we can represent in exponent form also.

```
double d1 = 15e2; [15 X 10 to the power of 2]
double d2 = 15e-2;
```

* Any integral literal (byte, short, int and long) we can assign to floating point literal (float and double) but a floating point literal we can't assign to integral literal.

* While working with integral literal, we can represent an integral literal in four different forms i.e decimal, octal,
binary and hexadecimal but floating point literal we can represent in only one form i.e decimal.

```
double d1 = 0.129; //Valid
```

```
//Programs :
------------
public class Test
{
	public static void main(String[] args)
	{
		float f = 0.0; //error
		System.out.println(f);
	}
}
-----------------------------------------------------
public class Test1
{
	public static void main(String[] args)
	{
		float b = 15.29F;
		float c = 15.25f;
		float d = (float) 15.30;

		System.out.println(b + " : "+c+ " : "+d);

	}
}
-------------------------------------------------------------
public class Test2
{
	public static void main(String[] args)
	{
		double d = 15.15;
		double e = 15d;
		double f = 90D;

		System.out.println(d+" , "+e+" , "+f);
	}
}
-------------------------------------------------------------
public class Test3
{
	public static void main(String[] args)
	{
		 double x = 0129.89;
```

```
                double y = 0167;

                double z = 0178; //error

                System.out.println(x+","+y+","+z);
        }
}
-----------------------------------------------------------
class Test4
{
        public static void main(String[] args)
        {
                double x = 0X29;

                double y = 0X91.5; //error

                System.out.println(x+","+y);
        }
}
-----------------------------------------------------------
public class Test5
{
        public static void main(String[] args)
        {
                double d1 = 15e-3;
                System.out.println("d1 value is :"+d1);

                double d2 = 15e3;
                System.out.println("d2 value is :"+d2);
        }
}
-----------------------------------------------------------
public class Test6
{
        public static void main(String[] args)
        {
                double a = 0791; //Error

                double b = 0791.0;

                double c = 0777;

                double d = 0Xdead;
```

```
                double e = 0Xdead.0; //Error
        }
}
```

```
public class Test7
{
        public static void main(String[] args)
        {
          double a = 1.5e3;
          float b = 1.5e3;  //E
          float c = 1.5e3F;
          double d = 10;
          int e = 10.0;  //E
          long f = 10D; //E
          int g = 10F; //E
          long l = 12.78F; //E
        }
}
```

```
//Range and size of floating point literal
public class Test8
{
        public static void main(String[] args)
        {
                System.out.println("\n Float range:");
                System.out.println(" min: " + Float.MIN_VALUE);
                System.out.println(" max: " + Float.MAX_VALUE);
                System.out.println(" size :"+Float.SIZE);

                System.out.println("\n Double range:");
                System.out.println(" min: " + Double.MIN_VALUE);
                System.out.println(" max: " + Double.MAX_VALUE);
                System.out.println(" size :"+Double.SIZE);
        }
}
```
---
boolean literal :

It is used to represent two states i.e. either true or false.

In boolean literal, we have only one data type i.e boolean data type which accepts 1 bit of memory OR depends upon JVM

implementation.

```
boolean isValid = true;
   boolean isEmpty = false;
```

Unlike C and C++, We can't assign 0 and 1 to boolean data type because in java 0 and 1 both are representing numeric type (int type)

```
boolean x = 0; //[Valid in C and C++ but Invalid in java]
boolean y = 1; //[Valid in C and C++ but Invalid in java]
```

We can't assign String type to boolean where String contains true or false internally.

```
boolean x = "true";  //Invalid
```

Programs :

```
public class Test1
{
   public static void main(String[] args)
   {
      boolean isValid = true;
      boolean isEmpty = false;

      System.out.println(isValid);
      System.out.println(isEmpty);
    }
}


public class Test2
{
   public static void main(String[] args)
   {
         boolean c = 0; //error
      boolean d = 1; //error
      System.out.println(c);
      System.out.println(d);
   }
}

public class Test3
{
```

```
        public static void main(String[] args)
        {
                boolean x = "true";  //error
                boolean y = "false";  //error
                System.out.println(x);
            System.out.println(y);
        }
}
```

01-02-2025

Character Literal :

It is also known as char literal.

In Character literal we have only one data type i.e char data type
which accepts 16 bits of memory.

We can represent character literal in the following ways :

a) Single character enclosed with single quotes.
   Example : char ch = 'A';

b) We can assign char literal to integral literal data types to know the ASCII value of that particular character literal.
   Example : int x = 'A';

c) In older langugaes like c and c++, which supports ASCII value format and the range is 0 - 255, on the other hand java
   supports UNICODE (ASCII + NON ASCII) value format where the
   range is 0 - 65535.

```
        char ch = 65535; //Valid
```

d) We can represent char literal in 4 digit hexadecimal format
   to support UNICODE value format.

   The format is '\udddd'

   Here u represents it is UNICODE and d represents the digit
   which is in hexadecimal format.

   Here the range is : '\u0000' to '\uffff'

e) All the escape sequences we can represent as Character literal.

```
char ch = '\n';
```

Programs :

```
public class Test1
{
	public static void main(String[] args)
	{
		char ch1 = 'a';
		System.out.println("ch1 value is :"+ch1);

	}
}
```

```
public class Test2
{
	public static void main(String[] args)
	{
		int ch = 'A';
		System.out.println("ch value is :"+ch);

		char ch1 = 'A';
		System.out.println("ch1 value is :"+ch1);
	}
}
```

```
//The UNICODE value for ? character is 63
public class Test3
{
	public static void main(String[] args)
	{
		char ch1 = 63;
		System.out.println("ch1 value is :"+ch1); //?

		char ch2 = 64;
		System.out.println("ch2 value is :"+ch2); //@

		char ch3 = 65;
		System.out.println("ch3 value is :"+ch3); //A
	}
}
```

```
public class Test4
{
	public static void main(String[] args)
	{
		char ch1 = 65000;
       System.out.println("ch1 value is :"+ch1);

		char ch2 = 0Xadd;
		System.out.println("ch2 value is :"+ch2);


	}
}
```

Note : Here we will get the output as ? symbol which represents unknown type character because the language translator is not available in the current system.

```
//Addition of two character in the form of Integer
public class Test5
{
public static void main(String txt[])
  {
	int x = 'A';
    int y = 'B';

	System.out.println(x + y); //131
	System.out.println('A'+'B');//131
	System.out.println("A"+"B");//AB
   }
}
```

```
//Range of UNICODE Value (65535) OR '\uffff'
class Test6
{
	public static void main(String[] args)
	{
		char ch1 = 65535;
		System.out.println("ch value is :"+ch1);

		char ch2 = 65536; //error
		System.out.println("ch value is :"+ch2);
	}
```

```
}
-------------------------------------------------------------
//WAP in java to describe unicode representation of char in hexadecimal format
public class Test7
{
	public static void main(String[] args)
	{
		int ch1 = '\u0000';
		System.out.println(ch1);

		int ch2 = '\uffff';
		System.out.println(ch2);

		char ch3 = '\u0041';
	    System.out.println(ch3); //A

		char ch4 = '\u0061';
		System.out.println(ch4); //a
	}
}
-----------------------------------------------------------------
class Test8
{
	public static void main(String[] args)
	{
		char c1 = 'A';
		char c2 = 65;
		char c3 = '\u0041';

		System.out.println("c1 = "+c1+", c2 ="+c2+", c3 ="+c3);
	}
}
-----------------------------------------------------------------
class Test9
{
	public static void main(String[] args)
	{
		int x = 'A';
		int y = '\u0041';
		System.out.println("x = "+x+" y ="+y);
	}
}
-----------------------------------------------------------------
```

```
//Every escape sequence is char literal
class Test10
{
	public static void main(String [] args)
	{
		char ch ='\n';
		System.out.println("Hello");
		System.out.println(ch);

	}
}
-----------------------------------------------------------------
public class Test11
{
	public static void main(String[] args)
	{
		System.out.println("Min Value :"+(int)Character.MIN_VALUE);
		System.out.println("Max Value :"+(int)Character.MAX_VALUE);

		int min = '\u0000';
		int max = '\uffff';

		System.out.println("Min Value :"+min);
		System.out.println("Max Value :"+max);


	}
}
-----------------------------------------------------------------
```

03-02-2025

String Literal :

String is a predefined class available in java.lang Package.

String is a collection of alpha-nemeric character which is enclosed by double quotes. These characters can be alphabets, numbers, symbol or any special character.

In java we can create String object by using following 3 ways :

1) By using String Literal

```
String str1 = "india";
```

2) By using new keyword
  String str2 = new String("Hyderabad");

3) By using Character array [Old Technique]
  char ch[] = {'R', 'A', 'J'};

Programs :

```
//Three Ways to create the String Object
public class StringTest1
{
	public static void main(String[] args)
	{
		String s1 = "Hello World";	//Literal
		System.out.println(s1);

		String s2 = new String("Ravi"); //Using new Keyword
		System.out.println(s2);

		char s3[] = {'H','E','L','L','O'}; //Character Array
		System.out.println(s3);

	}
}
```

```
//String is collection of alpha-numeric character
public class StringTest2
{
	public static void main(String[] args)
	{
		String x="B-61 Hyderabad";
		System.out.println(x);

		String y = "123";
		System.out.println(y);

		String z = "67.90";
		System.out.println(z);

		String p = "A";
		System.out.println(p);
	}
}
```

```
//IQ
public class StringTest3
{
	public static void main(String []args)
	{
		String s = 15+29+"Ravi"+40+40;
		System.out.println(s);

	}
}
```
---

4) Punctuators :

It is also called separators.

It is used to inform the compiler how things are grouped in the code.

() {} [] ; , . @ ... (var args)

What is a local variable ?

If a variable is declared inside the body of method OR block OR Constructor then it is called Local /Automatic/ Stack/
temporary variable.

Example :

```
public void accept()
{
  int x = 100;  //x is a local variable
}
```

A local variable must be initialized by the developer before
use because local variable does not have any default value.

We can't apply any kind of modifier on local variable except final.

```
public void accept()
{
  final int x = 100;  //final is a valid modifier
}
```

As far as it's scope is concerned, It must be used within the same method body only that menas we can't use local variable outside of the method body/block/constructor.

All local variables are the part of the method body, all the methods are executed in a special memory in java called Stack Memory so local variables are the part of Stack memory.

A local variable must be pre-declared and initialized before use.

```
public void m1()
{
  System.out.println(x); //error
  int x = 100;
}
```

Why we can't use local variables outside of the method OR body OR Constructor ?

In java, Every methods are executed in a special memory called
Stack Memory.

Stack Memory works on LIFO (Last In First Out) basis.

In java, Whenever we call a method then a separate Stack Frame will be created for each and every method.[03-FEB]

Once the method execution is over then the corresponding method Stack frame will also be deleted from Stack Area, that is the reason we can't use local variable outside of the method.

Each stack frame contains 3 parts :

1) Local Variable Array
2) Frame Data
3) Operand Stack

Progarm :

```
public class LocalVariableScope
{
	public static void main(String[] args)
	{
		System.out.println("Main method Started!!!");
		m1();
		System.out.println("Main method ended!!!");
	}
```

```
	public static void m1()
	{
		System.out.println("M1 method Started!!!");
		m2();
		System.out.println("M1 method ended!!!");
	}

	public static void m2()
	{
		System.out.println("Inside the m2 method body!!!");
		int x = 100;
		System.out.println(x);
	}

}
```

Limitation of Commnd line Argument ?

As we know by using Command Line Argument, we can pass some value at runtime, These values are stroed in String array variable and then only the exceution of the program will be started.
In Command line Argumenet we can't ask to enter the value from our end user as shown in the Program.

```
package com.ravi.command_line_argument;

//Program to read gender [M/F] from the user

public class ReadCharacter
{
	public static void main(String[] args)
	{
		System.out.println("Please Enter your Geneder [M/F]");
		char gender = args[0].charAt(0);
		System.out.println("Your Gender is :"+gender);

	}

}
```

Note : In the above program, after providing the gender value, It is asking for Gender which is not a recommended way.

Note : charAt(int indexPosition) is a predefined non static method of String class, which is used to retrieve a character
from the given String.

//Program on charAt(int indexPosition) method :
---
public char charAt(int indexPosition)

```
public class ReadCharacter
{
	public static void main(String[] args)
	{
		String str = "India";
		 char ch = str.charAt(0);
		System.out.println("0th index character is :"+ch); //I
	}
}
```

How to read the data from the End user with user friendly mesasage :

There are so many ways we can read the data from client which are as follows :

1) DataInputStream class (java.io)
2) BufferedReader class (java.io)
3) System.in.read() (java.lang)
4) Console (java.io)
5) Scanner class (java.util)

04-02-2025

Scanner class :

In order to read the data from client with user-friendly input
message, Java software people has provided a predefined class called Scanner available in java.util package from JDK 1.5V.

static variables of System class :

System class has provided 3 static variables which as as follows :

1) System.out : Used to print normal message on the Screen.

2) System.err : Used to print error message on the screen.

3) System.in : Used to take the input from the source.

How to create an Object for Scanner class :

Scanner sc = new Scanner(System.in);

Non static Methods of Scanner class :

The following are the non static method of Scanner class :

1) public String next() : Used to read a single word.

2) public String nextLine() : Used to read a multiple words or
complete line.

3) public byte nextByte() : Used to read byte value.

4) public short nextShort() : Used to read short value.

5) public int nextInt() : Used to read int value.

6) public long nextLong() : Used to read long value.

7) public float nextFloat() : Used to read float value.

8) public double nexDouble() : Used to read double value.

9) public boolean nextBoolean() : Used to read boolean value.

10) public char next().charAt(0) : Used to read a single character

//How to read name from the user using Scanner class :

```
import java.util.*;

public class ReadName
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter your Name :");
```

```
		String name =  sc.nextLine();
		System.out.println("Your Name is :"+name);
	}
}
```

//WAP to read your age from the Scanner class :

```
package com.ravi.scanner;

import java.util.Scanner;

public class ReadAge {

	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter your Age :");

		int age =  sc.nextInt();
	   System.out.println("Your Age is :"+age);

	   sc.close();
	}

}
```

//WAP to read a character from the user using Scanner class :

```
package com.ravi.scanner;

public class ReadCharacter
{
	public static void main(String[] args)
	{
		java.util.Scanner sc = new java.util.Scanner(System.in);
		System.out.print("Enter your Gender [M/F] :");

		char gender = sc.next().charAt(0);
		System.out.println("Your Gender is :"+gender);
		sc.close();

	}
```

```
}
```

WAP to read student data from the Scanner class :

```
package com.ravi.scanner;

import java.util.Scanner;

public class StudentData {

	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);

		System.out.print("Enter Student Id :");
		int id = sc.nextInt();

		System.out.print("Enter Student Name :");
		String name = sc.nextLine(); //Buffer Problem
		name = sc.nextLine();

		System.out.println("Student Id is :"+id);
		System.out.println("Student name is :"+name);

		sc.close();

	}

}
```

Note : Here we have a buffer problem so, we need to write nextLine() method one more time to accept the user input.

Note : java.lang is a default package so we need not to import but if we use the classes from another package then we should use
import statement after package statement in java.
==========

5) Operators

It is a symbol which describes that how a calculation will be performed on operands.

Types Of Operators :
---
1) Arithmetic Operator (Binary Operator)

2) Unary Operators

3) Assignment Operator

4) Relational Operator (>, >=, <, <=, !=, ==)

5) Logical Operators (&& || !)

6) Boolean Operators (& |)

7) Bitwise Operators (^ ~)

8) Ternary Operator

*9) Member Operator(Dot . Operator)

*10) new Operator

*11) instanceof Operator [It is also relational operator]

Basic Concepts of Operators :

```
public class Test
{
	public static void main(String[] args)
	{
	int x = 15;
		int y = x++;
		System.out.println(x + " : "+ y);
	}
}
```

```
class Test
{
	public static void main(String[] args)
	{
	int x = 15;
		int y = --x;
		System.out.println(x + " : "+ y);
```

```
        }
}
-----------------------------------------------------------
class Test
{
        public static void main(String[] args)
        {
            int x = 20;
                int y = ++20; //error
                System.out.println(x + " : "+ y);
        }
}
-----------------------------------------------------------
class Test
{
        public static void main(String[] args)
        {
      int x = 20;
                int y = ++(++x);
                System.out.println(x + " : "+ y);
        }
}
-----------------------------------------------------------
class Test
{
        public static void main(String[] args)
        {
        char ch = 'A';
           ch++;
           System.out.println(ch);
        }
}
-----------------------------------------------------------
class Test
{
        public static void main(String[] args)
        {
        double d1 = 12.89;
           d1++;
           System.out.println(d1);
        }
}
```

Note : Increment and decrement operator we can apply on any data type execpt boolean.

05-02-2025

```
public class Test
{
	public static void main(String[] args)
	{
	while(false)
	  {
		System.out.println("Inside the loop body");
	  }

	  System.out.println("Outside the loop body");
	}
}
```

Note : In the above program, We will get compilation error because the body of while loop is un-rechable.

```
public class Test
{
	public static void main(String[] args)
	{
		boolean b = false;

      while(b)
	  {
		System.out.println("Inside the loop body");
	  }

	  System.out.println("Outside the loop body");
	}
}
```

Note : Here code will compile and execute.

```
public class Test
{
	public static void main(String[] args)
	{
		final int x = 10;
		final int y = 20;
```

```
        while(x > y)
                {
                        System.out.println("x is Greater than y");
                }
                System.out.println("Hello World");
        }
}
```

Note : Unrechable statement.

```
public class Test
{
        public static void main(String[] args)
        {
                for(int i=1; i<=10; i++)
                {
                  if(i=5)
                        break;
                  System.out.println(i);
                }
        }
}
```

Note : error, if condition requires booelan expression.

```
public class Test
{
        public static void main(String[] args)
        {

                do
                {
                        int i=1; //Block level variable
                        System.out.println("i value is :"+i);
                        i++;
                }
                while (i<=10);
        }
}
```

Expression Conversion :

While working with Arithmetic Operator (+, -, *, /) and Unary minus operator (-), the minimum

data type is required to hold the expression result would be int data type that means we need 32 bits of data to hold the result otherwise we will get
compilation error.

```
public class Test
{
	public static void main(String[] args)
	{
		short s = 1;
		short t = 1;
		short u =  s + t;  //error
		System.out.println(u);

	}
}
```

```
public class Test
{
	public static void main(String[] args)
	{
		byte b1 = 1;
		byte b2 = 2;
		byte res = b1*b2; //error
		System.out.println(res);

	}
}
```

Unary Minus Operator :

```
public class Test
{
	public static void main(String[] args)
	{
		int b = 1;
		System.out.println(-b); //-1

	}
}
```

```
public class Test
{
	public static void main(String[] args)
```

```
	{
		byte b1 = 1;
		byte b2 = -b1; //error
		System.out.println(b2);

	}
}
```

```
public class Test
{
	public static void main(String[] args)
	{
		byte b1 = 1;
		byte b2 = --b1;
		System.out.println(b2);

	}
}
```

```
public class Test
{
	public static void main(String[] args)
	{
		byte b = 5;
		b += 6;   //b = b + 6;
		System.out.println(b);

	}
}
```

Here we are using short hand operator so it will generate the output.

Program On Boolean Operator :

```
//Program on Boolean AND operator(&)

public class Test
{
	public static void main(String[] args)
	{
		int z = 5;
		if(++z > 6 & ++z> 6)
		{
```

```
                        System.out.println("Inside If");
                        z++;
                }
                System.out.println(z);

        }
}
```

```
public class Test
{
        public static void main(String[] args)
        {
                int z = 5;
                if(++z > 5 || ++z > 6)   //Logical OR
                {
                        z++;
                }
                System.out.println(z); //7

       System.out.println(".....");

                z = 5;
                if(++z > 5 | ++z > 6)   //Boolean OR
                {
                        z++;
                }
                System.out.println(z); //8




        }
}
```

Working with Bitwise AND(&), Bitwise OR(|) and Bitwise X-OR (^) :

```
public class Test
{
        public static void main(String[] args)
        {
                System.out.println(false ^ true); //true
        }
}
```

Here we will get the output based on the input i.e Same input output will be zero.

```
public class Test
{
	public static void main(String[] args)
	{
		System.out.println(5 & 6);
		System.out.println(5 | 6);
		System.out.println(5 ^ 6);
	}
}
```

Bitwise Complement Operator :

It will not work with boolean.

```
public class Test
{
	public static void main(String[] args)
	{
		System.out.println(~ true);
	}

}
```

```
public class Test
{
	public static void main(String[] args)
	{
		System.out.println(~4); //-5
		System.out.println(~-5);//4
	}

}
```

Member Operator :

Dot operator is known as Member access operator.
It is used to access the member of the class.

```
class Welcome
{
	static int x = 100; //static field
```

```
	public static void access() //static method
	{
		System.out.println("Access static Method");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Welcome.access();
		System.out.println(Welcome.x);
	}

}
```

Note : Welcome class contains static variable and static method so, we can directly call static veraible and static method with the help of class name using dot operator.

---

new Keyword :

It is a keyword as well as operator in java.
It is used to provide default value for non static data member.
It is also useful to create the object with the help of Constructor.

```
class Welcome
{
	int x; //non static field

	public void access() //non static method
	{
		System.out.println("Non static method");
	}
}

public class Test
{
	public static void main(String [] args)
	{
		Welcome w = new Welcome();
		System.out.println(w.x);
		w.access();

	}
}
```

Note : Welcome class contains non static variable and non static metohd so, Object is required.

Limitation of if else :

The major drawback with if condition is, it checks the condition again and again so It increases the burdon over CPU so we introduced switch-case statement to reduce the overhead of the CPU.

Switch case statement in java :

It is a selective statement so, we can select one statement among the available statements.

break is optional but if we use break then the control will move from out of the switch body.

We can write default so if any statement is not matching then default will be executed.

In switch case we can't pass long, float, double and boolean value.

[long we can pass in switch case from java 14v]

We can pass String from JDK 1.7v and we can also pass enum from JDK 1.5v.

06-02-2025

//Programs on Switch case

```
import java.util.*;
public class SwitchDemo
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
	System.out.print("Please choose any color name :");
		char colour = sc.next().toLowerCase().charAt(0);

		switch(colour)
		{
		case 'r' : System.out.println("Red") ; break;
		case 'g' : System.out.println("Green");break;
		case 'b' : System.out.println("Blue"); break;
		case 'w' : System.out.println("White"); break;
		default : System.out.println("No colour");
```

```
                }
                System.out.println("Completed") ;
        }
}
------------------------------------------------------------
import java.util.Scanner;
public class SwitchDemo1
{
public static void main(String args[])
 {
                System.out.println("\t\t**Main Menu**\n");
                System.out.println("\t\t**100 Police**\n");
                System.out.println("\t\t**101 Fire**\n");
                System.out.println("\t\t**102 Ambulance**\n");
                System.out.println("\t\t**139 Railway**\n");
                System.out.println("\t\t**181 Women's Helpline**\n");

                System.out.print("Enter your choice :");
                Scanner sc = new Scanner(System.in);
                int choice = sc.nextInt();

                switch(choice)
                {
                case 100:
                System.out.println("Police Services");
                break;
                case 101:
                System.out.println("Fire Services");
                break;
                case 102:
                System.out.println("Ambulance Services");
                break;
                case 139:
                System.out.println("Railway Enquiry");
                break;
                case 181:
                System.out.println("Women's Helpline ");
                break;
                default:
                System.out.println("Your choice is wrong");
                }
   }
}
```

```
-----------------------------------------------------------------
import java.util.*;
public class SwitchDemo2
{
	public static void main(String[] args)
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter the name of the season :");
		String season = sc.next().toUpperCase();

		switch(season)  //String allowed from 1.7 [It is constant]
		{
			case "SUMMER" :
				System.out.println("It is summer Season!!");
			 break;

			 case "RAINY" :
				System.out.println("It is Rainy Season!!");
			 break;
		}
	}
}
-----------------------------------------------------------------
public class Test
{
	public static void main(String[] args)
	{
       long l = 12;

		switch(l) //We can't pass long, float,double and
		{				boolean
			case 12 :
				System.out.println("It is case 12");
			break;
		}


	}

}
-----------------------------------------------------------------
public class Test
{
```

```
	public static void main(String[] args)
	{
       int x = 12;
		int y = 12;

		switch(x)
		{
			case y : //error it is not a constatnt
				System.out.println("It is case 12");
			break;
		}


	}

}
-----------------------------------------------------------------
public class Test
{
	public static void main(String[] args)
	{
       int x = 12;
		final int y = 12;

		switch(x)
		{
			case y :
				System.out.println("It is case 12");
			break;
		}


	}

}
-----------------------------------------------------------------
public class Test
{
	public static void main(String[] args)
	{
      byte x = 12;

		switch(x)
```

```
		{
			case 127 :
				System.out.println("It is case 127");
			break;

			case 128 :
				System.out.println("It is case 128");
			break;
		}


	}

}
```

```
public class Test
{
	public static void main(String[] args)
	{
      short x = 12;

		switch(x)
		{
			case 32767 :
				System.out.println("It is case 32767");
			break;

			case 32768 : //error
				System.out.println("It is case 32768");
			break;
		}


	}

}
```

Loops in java :

A loop is nothing but repeatation of statements based on the specified condition.

In java we have 4 types of loops :

---
1) do-while loop
2) while loop
3) for loop
4) for each loop

Program on do while loop :

```
public class DoWhileDemo
{
	public static void main(String[] args)
	{
	  int x = 1;
      do
      {
                  System.out.println(x);
                  x++;
      }
      while (x<=10);

	}

}
```

Program on while loop :

```
public class WhileDemo
{
	public static void main(String[] args)
	{
		int x = 1;

	  while(x>=-10)
		{
		  System.out.println(x);
		  x--;
		}

	}

}
```
---

for loop :

```
public class Test
{
	public static void main(String[] args)
	{
	  for(int i=0; i<=10; i++)
	  {
		System.out.println(i);
	  }
	}

}
```

==========

for-each loop in java :

It is also known as enhanced for loop available from JDK 1.5V.

Internally compiler will generate ordinary for loop for this
for each loop.

It is mainly used to retrieve the values from the collection one by one.

Actually It will iterate the elements from the Collection but element type must be Iterable type (java.lang.Iterable)

```
package com.ravi.for_each_loop;

public class ForEachDemo1 {

	public static void main(String[] args)
	{
		int []numbers = {10,20,30,40,50,60};

		for(int number : numbers)
		{
			System.out.println(number);
		}

	}

}
```

How to sort Array data :

In java.util pacakge, there is a predefined class called Arrays which has various static methods to sort the array in ascending or alphabetical order.

Example :

```
Arrays.sort(int []arr);  //For sorting int array
   Arrays.sort(Object []arr) //For sorting String array
```

```
package com.ravi.for_each_loop;

import java.util.Arrays;

public class ForEachDemo2 //
{
        public static void main(String[] args)
        {
    int []numbers = {89, 67, 45, 12, 9};

    Arrays.sort(numbers);

    for(int number : numbers)
    {
        System.out.println(number);
    }

        }

}
```

```
package com.ravi.for_each_loop;

import java.util.Arrays;

public class FoEachDemo3 {

        public static void main(String[] args)
        {
                String []fruits = {"Orange", "Grapes", "Apple", "Mango"};

                Arrays.sort(fruits);
```

```
		for(String fruit : fruits)
		{
			System.out.println(fruit);
		}

	}

}
```

//How to take the value from the Command line Argument and sort the String data using Arrays.sort() method

```
package com.ravi.for_each_loop;

import java.util.Arrays;

public class ForEachDemo4 {

	public static void main(String[] args)
	{
		Arrays.sort(args);

		for(String arg : args)
		{
			System.out.println(arg);
		}

	}

}
```

Creating arrays in different way and retrieving using forEach loop.

```
package com.ravi.for_each_loop;

public class ForEachDemo5 {

	public static void main(String[] args)
	{
		//Example 1 :
		int []arr1 = new int[3];
		arr1[0] = 10;
```

```
		arr1[1] = 20;
		arr1[2] = 30;

		for(int x : arr1)
		{
			System.out.println(x);
		}

		System.out.println(".............");

		//Example 2
		int []arr2 = new int[] {100,200,300};

		for(int y : arr2)
		{
			System.out.println(y);
		}
		System.out.println(".............");

		//Example 3

		int arr3  [] = {1000,2000, 3000};

		for(int z : arr3)
		{
			System.out.println(z);
		}

	}

}
```

In java, Can we hold hetrogeneous types of data using array ?

Yes, by using Object array we can hold hetrogeneous type of data but we can't peform sorting operation using Arrays.sort(), It will generate
java.lang.ClassCastException

```
package com.ravi.for_each_loop;

public class ForEachDemo6
{
	public static void main(String[] args)
```

```
	{
		Object []arr = {12, 23.90, 'A', true, "NIT", new String("A")};

		// Arrays.sort(arr);  java.lang.ClassCastException

		for(Object obj : arr)
		{
			System.out.println(obj);
		}

	}

}
```

07-02-2025

What is BLC and ELC class in java ?

BLC :

It stands for Business Logic class. A developer should write the logic inside the BLC class. BLC class will never contain
main method.

Example :

```
public class Calculate
{
}
```

ELC :

It stands for Executable logic class. It contains main method.
The execution of the program will always start from ELC class
only.

Example :

```
public class ELC
{
  public static void main(String [] args)
  {
```

```
  }
}
```

In java, Is it possible to write multiple classes in single .java file ?

Yes, We can write multiple classes but in a single .java file,
we should have only one public class and that class must be file
name (FileName.java).

Example :

ELC.java

```
class A  //A class must be non public
{
}

class B //B class must be non public
{
}

public class ELC
{
	public static void main(String[] args)
	{
	}
}
```

Note : Here file name is ELC.java so, We can declare only ELC class with public access modifier.

//Writing the java program with static and non static methods
using BLC and ELC approach

```
package com.ravi.blc_elc;

//BLC class
class Calculate
{
	//static method
    public static double getSquare(double num)
```

```
    {
        return num*num;
    }

    //non static method
    public double getCube(double num)
    {
        return num*num*num;
    }

}


//ELC class
public class ELC
{
        public static void main(String[] args)
        {
                //Calling the static method
                double square = Calculate.getSquare(5);
                System.out.println("Squre of 5 is :"+square);

                //Calling non static method
                Calculate calc = new Calculate();
                double cube = calc.getCube(3);
                System.out.println("Cube of 3 is :"+cube);


        }
}
```

Why we should use BLC and ELC approach to write java program ?

If we write logic and execution, everything in a single class then it is not a recommended way because in this approach,
THE REUSABILITY OF BLC CLASS IS NOT POSSIBLE.

In order to reuse our BLC classes from same package or even from different packages we should use BLC and ELC approach as shown in the Program.

com.ravi.m1

In this package we have 3 public classes :

Square.java

```
package com.ravi.m1;

//BLC (public)
public class Square
{
	public static void getSquare(int x)
	{
		System.out.println("Square of "+x+" is :"+(x*x));
	}
}
```

Rectangle.java

```
package com.ravi.m1;

//BLC (public BLC class)
public class Rectangle
{
  public static void getRectangle(int length, int breadth)
  {
	System.out.println("Area of rectangle is :"+(length* breadth));
  }
}
```

ELC.java

```
package com.ravi.m1;

public class ELC {

	public static void main(String[] args)
	{
		Square.getSquare(10);
		Rectangle.getRectangle(5, 4);

	}

}
```

We have another package com.ravi.m2 which contains a public class Main.java

Main.java

```
package com.ravi.m2;

import com.ravi.m1.Rectangle;
import com.ravi.m1.Square;

public class Main {

	public static void main(String[] args)
	{
		Rectangle.getRectangle(9, 9);
		Square.getSquare(5);

	}

}
```

Note : In the above program, We are importing the Rectangle and Square class for execution.

How to reuse a class in java ?

The slogan of java is "WORA" write once run anywhere.

A public class created in one package can be reuse from different packages also by using import statement.

In a single .java file, we can declare only one public class that must be our .java file and that class can be reusable to all the packages.

*In a single java file, we can write only one public class and multiple non-public classes but it is not a recommended approach because the non public class we can use within the same package only.

So the conclusion is, we should declare every java class in a separate file to enhance the reusability of the BLC classes.
[Note we have 10 classes -> 10 java files]

How many .class file will be created in the above approach :

For a public class in a single file, Only 1 .class file will be created.

For a public class in a single file which contains n number of non public classes then compiler will generate n number of .class file as shown in the program

```
class Alpha
{
}

class Beta
{
}

class Gamma
{
}

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

Here total 4 .class files will be created
1) Alpha.class
2) Beta.class
3) Gamma.class
4) Test.class

==========

Working with Static Method with different return type :
