Datatype In Java:

Data Type of JAVA											
Prefixed Datatype											Enumerated User-defined
Primitive							**Derived**		**User Defined**		
Numeric					bool	void	Array	Graph	Structure		enum
integral				char			String	Pointer	Union		
integer		Floating Point					Stack		class		
byte 1B		Float					Queue		interface		
short 2B		double					Linked List				
int 4B							Tree				
long 8B											

It indicates type of value to be stored and memory requirement of the variable.

Integer data type:

byte = -128 -to 127 = 1byte

short = -32,768 to 32,767 = 2byte

int = -2,147,483,648 to 2,147,483,647= 4byte

long = -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 = 8 byte

all the integer data type is sign 2's complement integer.

floating data type:

float = it is 32 bit floating point. Float can represent up to 7digit after the decimal point.

Ranges from 1.4012984643248170e-45f to 3.40282347663852886 0e+38f

Double = it is 64 bit floating point. Double can represent up to 15 digit after the decimal point.

Ranges from 4.94065645841246544e-324 to 1.79769313486231570e+308

Character data type:

The unsigned char data type is a single 16-bit Unicode character. It ranges from \u0000 to \uffff

Boolean data type:

It has only two possible value either true or false. This data type is not precisely defined data type.

Void data Type:

it is a reserved keyword. Void is used in function and pointer concept. Void is used if do not want any type of return value from the function or a specific method. Void is used in pointer which has no associated data type with it. Void pointer point to any data of any data type and can be type casted to any type.

```
/*DIFFERENT TYPES OF DATA TYPES IN JAVA*/
public class JVP02 {
    public static void main(String args[]) {
        // integer data type

        byte num0 = -128;
        System.out.println("byte num0 = " + num0); // byte num0 = -128
        byte num1 = 127;
        System.out.println("byte num1 = " + num1); // byte num1 = 127

        short num2 = -32768;
        System.out.println("short num2 = " + num2); // short num2 = -32768
        short num3 = 32767;
        System.out.println("short num3 = " + num3); // short num3 = 32767

        int num4 = -2147483648;
        System.out.println("int num4 = " + num4); // int num4 = -2147483648
        int num5 = 2147483647;
        System.out.println("int num5 = " + num5); // int num5 = 2147483647

        long num6 = -9223372036854775808L;
        System.out.println("long num6 = " + num6); // long num6 = -9223372036854775808
        long num7 = 9223372036854775807L;
        System.out.println("long num7 = " + num7); // long num7 = 9223372036854775807

        // floating data type

        float fr1 = (float) 0.123456789;
        System.out.println("float fr1 = " + fr1); // float num8 = 0.12345679

        double dou1 = (double) 0.12345678901234567890;
        System.out.println("double dou1 = " + dou1); // double num9 = 0.12345678901234568

        // character data type

        char ch1 = '\u0000';
        System.out.println("char ch1 = " + ch1); //char ch1 =

        char ch2 = '\uffff';
        System.out.println("char ch2 = " + ch2); //char ch2 = ?

        //boolean data type
        boolean bool1 = true ;
        System.out.println("boolean bool1 = "+bool1); //boolean bool1 = true

        boolean bool2 = false ;
        System.out.println("boolean bool2 = "+bool2); //boolean bool2 = false
    }
}
```

```java
class test {
    /* return type method/function */
    int getUserAge() {
        int Age = 15;

        return Age;
    }

    String getUserName(String Name) {
        // Name = "Mritunjay Kumar Sahu" ;

        return Name;
    }

    /* void or no return type method/function */
    void getName() {
        String name = "ABCDEFGH";
        System.out.println("name : " + name);// name : ABCDEFGH
    }

    void getAGe() {
        int AGe = 20;
        System.out.println("age : " + AGe); // age : 20
    }
}

public class JVP03 {
    public static void main(String args[]) {
        test t = new test();
        System.out.println("USER NAME : " +t.getUserName("Mritunjay Kumar Sahu")); // USER NAME : Mritunjay Kumar Sahu
        System.out.println("USER AGE : " +t.getUserAge()); // USER AGE : 15

        t.getName(); // name : ABCDEFGH
        t.getAGe(); // age : 20
    }
}
```

Literals And Type Casting In JAVA:

LITERALS:

A literal is a value that can be passed to a variable or constant in a program. literals can be numeric, boolean, character, string or null literals. Literals are represented in binary, decimal, octal, hexadecimal notation. These literals can be assigned to all numeric type in java including char data type.

Binary Literals:

Binary literals are combination of 0's and 1's. binary literals can be assigned to variables and must be prefixed with "0b"/"0B".

```java
//binary literals

char num3 = 0b01001010 ;
System.out.println(num3); //J

byte num7 = 0b01010010 ;
System.out.println(num7); //82

short num2 = 0B00001010 ;
System.out.println(num2); //10

int num1 = 0b00000001;
System.out.println(num1); //1

long num6 = 0b11110000;
System.out.println(num6); //240

float num4 = 0b11100011 ;
System.out.println(num4); //227.0

double num5 = 0b11001100 ;
System.out.println(num5); //204.0
```

Octal Literals:

In case octal literal we have to specified the value must be prefixed with zero(0) and the digits from 0 to 7.

```
//octal literals

char oc1 = 0127 ;
System.out.println(oc1); //W

byte oc2 = 0146;
System.out.println(oc2); //102

short oc3 = 0766;
System.out.println(oc3); //502

int oc4 = 0523;
System.out.println(oc4); //339

long oc5 = 0117;
System.out.println(oc5); //79

float oc6 = 0256;
System.out.println(oc6); //174.0

double oc7 = 0371;
System.out.println(oc7); //249.0
```

Hexadecimal Literals:

In hexadecimal literals are prefixed with “0x” or “0X”. the digits 0 to 9 and A to F.

```
//hexadecimal literals

char hd1 = 0xA11B;
System.out.println(hd1); //?

byte hd2 = 0X0022;
System.out.println(hd2); //34

short hd3 = 0x0AF9;
System.out.println(hd3); //2809

int hd4 = 0xADF1;
System.out.println(hd4); //44529

long hd5 = 0xFFDD;
System.out.println(hd5); //65501

float hd6 = 0xA1B4;
System.out.println(hd6); //41396.0

double hd7 = 0xAAFF;
System.out.println(hd7); //43775.0
```

Integral Literals:

these are by default int. to define them as long data type we can place a suffix of “L” or “l” after the number for instance. At java version 7 onwards the readability of literals can be enhanced by using underscore with numeric literals.

```
//integer literals

byte il2 = 1_1_2;
System.out.println(il2); //112

int il1 = 100_100_100;
System.out.println(il1); //100100100

short il3 = 25_1_55;
System.out.println(il3); //25155

long il4 = 148_236_579_0L;
System.out.println(il4); //1482365790

float il5 = 12_24.56_74F;
System.out.println(il5); //1224.5674

double il6 = 145_236.789_111;
System.out.println(il6); //145236.789111
```

Char Literals:

a single character is enclosed in single quotes. You can also use the prefix “\u” followed by four hexadecimal digits representing the 16-bit Unicode character.

```
//character literals

char ch1 = '\u0048';
System.out.println(ch1); //H
```

Boolean Literals:

this specified as either true or false. By default it takes the value true or false.

```
//boolean literal
boolean b1 = true ;
boolean b2 = false;
System.out.println(b1 +" "+b2); //true false
```

Null Literals:

the null literal are assigned to object reference variables.

```
//null literal
Object nu = null ;
System.out.println(nu); //null
```

String Literals:

it consist of zero or more characters within double quotes.

```
//string literal
String str = "ATISH KUMAR SAHU";
System.out.println(str); //ATISH KUMAR SAHU
```

Type Casting Java:

It defines as when one type of value is assigned into another type of variable known as type casting. It is two types implicit type casting explicit type casting

Implicit type casting: when the type conversion between two data type is done automatically known as implicit type casting. Automatic type conversion will take place when the destination data type is larger than the source data type.

byte -> char -> int -> long -> float -> double

Explicit type casting: when two data types are not compatible or the source type is larger than the destination data type then we must use a cast.

double -> float -> long -> int -> short -> byte

```
// implicit type casting

byte num = 127;
System.out.println(num); // 127

short num1 = num;
System.out.println(num1); // 127

int num2 = num1;
System.out.println(num2); // 127

long num3 = num2;
System.out.println(num3); // 127

float num4 = num3;
System.out.println(num4); // 127.0

double num5 = num4;
System.out.println(num5); // 127.0
```

```
// explicit type casting

double NUM1 = 123.0;
System.out.println(NUM1); // 123.0

float NUM2 = (float) NUM1;
System.out.println(NUM2); // 123.0

long NUM3 = (long) NUM2;
System.out.println(NUM3); // 123

int NUM4 = (int) NUM3;
System.out.println(NUM4); // 123

short NUM5 = (short) NUM4;
System.out.println(NUM5); // 123

byte NUM6 = (byte) NUM5;
System.out.println(NUM6); // 123
```

Operators In Java:

An operator performs an action on one or more operands. There are three types of operators are used in java language.

UNARY OPERATOR # TERNARY OPERATOR # BINARY OPERATOR

Unary Operator:

an operator that performs an action on one operand is called a unary operator.

Increment & Decrement Operator:

increment decrement operator can be applied to all integers and floating point data types. This can be used prefix (--x, ++x) or postfix(x--, x++).

```
/*prefix increment and decrement operation*/
//first operation then store the value.
int num1 = 15 ;
System.out.println("num1 = "+num1); //num1 = 15
int res1 = ++num1 ;
System.out.println("res1 = "+res1); //res1 = 16
System.out.println("new num1 = "+num1); //new num1 = 16

int num2 = 28 ;
System.out.println("num2 = "+num2); //num2 = 28
int res2 = --num2 ;
System.out.println("res2 = "+res2); //res2 = 27
System.out.println("new num2 = "+num2); //new num2 = 27


/*Postfix increment and decrement operation*/
//first store the value then operation

int num3 = 35 ;
System.out.println("num3 = "+num3); //num3 = 35
int res3 = num3++;
System.out.println("res3 = "+res3); // res3 = 35
System.out.println("new num3 = "+num3); // new num3 = 36

int num4 = 28 ;
System.out.println("num4 = "+num4);//num4 = 28
int res4 = num4--;
System.out.println("res4 = "+res4);//res4 = 28
System.out.println("new num4 = "+num4);//new num4 = 27
```

Ternary Operator:

Ternary operators are applied to three operands. This conditional operator (? :) decides on the basis of the first expression, which of the two expressions to be evaluated.

```
/*TERNARY OPERATOR*/
import java.util.Scanner;

public class JVP14 {
    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);

        System.out.println("value of num1 = ");
        int num1 = in.nextInt();

        System.out.println("value of num2 = ");
        int num2 = in.nextInt();

        String result = num1 > num2 ? "num1 is greater" : "num2 is greater";

        System.out.println("RESULT = " + result);
    }
}
/* OUTPUT
value of num1 =
28
value of num2 =
36
RESULT = num2 is greater
*/
```

Binary Operator:

An operator that performs an action on two operands is called a binary operator. Java provides arithmetic, assignment, relational, shift, conditional, logical, bitwise, and member access operators.

Arithmetic Operator:

```
/*ARITHMETIC OPERATOR*/
public class JVP15 {
    public static void main(String args[]) {
        int num1 = 23;
        int num2 = 52;

        System.out.println("ADDITION = " + (num1 + num2)); //ADDITION = 75
        System.out.println("SUBTRACTION = " + (num1 - num2)); //SUBTRACTION = -29
        System.out.println("MULTIPICATION = " + (num1 * num2)); //MULTIPICATION = 1196
        System.out.println("DIVISION = " + (num1 / num2)); //DIVISION = 0
        System.out.println("MODULO = " + (num1 % num2)); //MODULO = 23
    }
}
```

Relational Operator:

```java
/*RELATIONAL OPERATOR*/
public class JVP16 {
    public static void main(String args[]) {
        int num1 = 234;
        int num2 = 243;
        System.out.println("num1 == num2 = " + (num1 == num2)); //num1 == num2 = false
        System.out.println("num1 != num2 = " + (num1 != num2)); //num1 != num2 = true
        System.out.println("num1 > num2 = " + (num1 > num2)); //num1 > num2 = false
        System.out.println("num1 >= num2 = " + (num1 >= num2)); //num1 >= num2 = false
        System.out.println("num1 < num2 = " + (num1 < num2)); //num1 < num2 = true
        System.out.println("num1 <= num2 = " + (num1 <= num2)); //num1 <= num2 = true
    }
}
```

Boolean Logical Operator:

```java
/*OR OPERATION*/
System.out.println("true | true = "+(true | true)); //true | true = true
System.out.println("true | false = "+(true | false)); //true | false = true
System.out.println("false | true = "+(false | true)); //false | true = true
System.out.println("false | false = "+(false | false)); //false | false = false

/*AND OPERATION*/
System.out.println("true & true = "+(true & true)); //true & true = true
System.out.println("true & false = "+(true & false)); //true & false = false
System.out.println("false & true = "+(false & true)); //false & true = false
System.out.println("false & false = "+(false & false));//false & false = false

/*XOR OPERATION*/
System.out.println("true ^ true = "+(true ^ true)); //true ^ true = false
System.out.println("true ^ false = "+(true ^ false)); //true ^ false = true
System.out.println("false ^ true = "+(false ^ true)); //false ^ true = true
System.out.println("false ^ false = "+(false ^ false));//false ^ false = false

/*NOR OPERATION*/
System.out.println("!(true | true) = "+!(true | true)); //!(true | true) = false
System.out.println("!(true | false) = "+!(true | false)); //!(true | false) = false
System.out.println("!(false | true) = "+!(false | true)); //!(false | true) = false
System.out.println("!(false | false) = "+!(false | false)); //!(false | false) = true

/*NAND OPERATION*/
System.out.println("!(true & true) = "+!(true & true)); //!(true & true) = false
System.out.println("!(true & false) = "+!(true & false)); //!(true & false) = true
System.out.println("!(false & true) = "+!(false & true)); //!(false & true) = true
System.out.println("!(false & false) = "+!(false & false));//!(false & false) = true

/*XNOR OPERATION*/
System.out.println("!(true ^ true) = "+!(true ^ true)); //!(true ^ true) = true
System.out.println("!(true ^ false) = "+!(true ^ false)); //!(true ^ false) = false
System.out.println("!(false ^ true) = "+!(false ^ true)); //!(false ^ true) = false
System.out.println("!(false ^ false) = "+!(false ^ false));//!(false ^ false) = true

/*NOT OPERATION*/
System.out.println("!true = "+(!true)); //!true = false
System.out.println("!false = "+(!false)); //!false = true
```

Conditional Operator:

```
/*CONDITIONAL OR*/
System.out.println("true || true = "+(true || true)); //true || true = true
System.out.println("true || false = "+(true || false)); //true || false = true
System.out.println("false || true = "+(false || true)); //false || true = true
System.out.println("false || false = "+(false || false)); //false || false = false

 /*CONDITIONAL AND*/
System.out.println("true && true = "+(true && true)); //true && true = true
System.out.println("true && false = "+(true && false)); //true && false = false
System.out.println("false && true = "+(false && true)); //false && true = false
System.out.println("false && false = "+(false && false)); //false && false = false


/*CONDITIONAL NOR*/
System.out.println("!(true || true) = "+!(true || true)); //!(true || true) = false
System.out.println("!(true || false) = "+!(true || false)); //!(true || false) = false
System.out.println("!(false || true) = "+!(false || true)); //!(false || true) = false
System.out.println("!(false || false) = "+!(false || false)); //!(false || false) = true

 /*CONDITIONAL AND*/
System.out.println("!(true && true) = "+!(true && true)); //!(true && true) = false
System.out.println("!(true && false) = "+!(true && false)); //!(true && false) = true
System.out.println("!(false && true) = "+!(false && true)); //!(false && true) = true
System.out.println("!(false && false) = "+!(false && false)); //!(false && false) = true
```

Bitwise Operator:

```
byte num1 = 15;
byte num2 = 10;
System.out.println("BIT WISE OR 15 | 10 = " + (15 | 10)); // BIT WISE OR 15 | 10 = 15
System.out.println("BIT WISE AND 15 & 10 = " + (15 & 10)); // BIT WISE AND 15 & 10 = 10
System.out.println("BIT WISE XOR (15 ^ 10) = "+(15 ^ 10)); //BIT WISE XOR (15 ^ 10) = 5

System.out.println("BIT WISE NOT ~(15) = " + ~(15)); // BIT WISE NOT ~(15) = -16


System.out.println("BIT WISE NOR ~(15 | 10) = " + ~(15 | 10)); // BIT WISE NOR ~(15 | 10) = -16
System.out.println("BIT WISE NAND ~(15 & 10) = " + ~(15 & 10)); // BIT WISE NAND ~(15 & 10) = -11
System.out.println("BITWISE XNOR ~(15 ^ 10) = "+~(15 ^ 10)); //BITWISE XNOR ~(15 ^ 10) = -6
```

Shift Operator:

```
byte num1 = 10;

byte res1 = (byte) (num1 >> 1);
System.out.println("RIGHT SHIFT OPERATOR = "+(res1)); //RIGHT SHIFT OPERATOR = 5

byte res2 = (byte)(num1 << 1);
System.out.println("LEFT SHIFT OPERATOR = "+(res2)); //LEFT SHIFT OPERATOR = 20

byte res3 = (byte)(num1 >>> 1);
System.out.println("UNSIGNED RIGHT SHIFT OPERATOR = "+(res3)); //UNSIGNED RIGHT SHIFT OPERATOR = 5
```

```java
byte num2 = -10;
byte res4 = (byte) (num2 >> 1);
System.out.println("RIGHT SHIFT OPERATOR = " + (res4)); // RIGHT SHIFT OPERATOR = -5

byte res5 = (byte) (num2 << 1);
System.out.println("LEFT SHIFT OPERATOR = " + (res5)); // LEFT SHIFT OPERATOR = -20

byte res6 = (byte) (num2 >>> 1);
System.out.println("UNSIGNED RIGHT SHIFT OPERATOR = " + (res6)); // UNSIGNED RIGHT SHIFT OPERATOR = -5
```

Assignment Operator:

GENERAL STATEMENT	ASSIGNMENT OPERATOR STATEMENT
A + B	A += B
A - B	A -= B
A * B	A *= B
A / B	A /= B
A % B	A %= B
A |B	A |= B
!(A | B)	!(A |= B)
A & B	A &= B
!(A & B)	!(A &= B)
A ^ B	A ^= B
!(A ^ B)	!(A ^= B)
A || B	A ||= B
!(A|| B)	!(A ||= B)
A && B	A &&= B
A >> B	A >>= B
A << B	A <<= B
A >>> B	A >>>= B

Instance Operator:

We can use instanceof operator to check whether the given object is of particular type or not.

r instanceof x

r = object reference x = class/interface name

example-1:

Thread t = new Thread();

S.O.P(t instanceof Thread); S.O.P(t instanceof Object);

S.O.P(t instanceof Runnable);

Thread is child of object. Thread implements runnable interface.

```
class test
{
public static void main(String args[])
{
Thread t = new Thread();
System.out.println(t instanceof Thread);
System.out.println(t instanceof Object);
System.out.println(t instanceof Runnable);
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
true
true
true
```

```
class test
{
public static void main(String args[])
{
Thread t = new Thread();
System.out.println(t instanceof String);
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java
test.java:6: error: incompatible types: Thread cannot be converted to String
System.out.println(t instanceof String);
                   ^
1 error
```

To use instanceof operator there should be some relationship argument type either child-to-parent, parent-to-child or same type otherwise we will get compile time error saying. “incompatible types”. For any class or interface “x” null instanceof x is always false.

```
class test
{
public static void main(String args[])
{
Thread t = new Thread();
System.out.println(null instanceof Object);
System.out.println(null instanceof Thread);
System.out.println(null instanceof Runnable);
}
}
```

```
C:\Users\Atish kumar sahu\desktop>javac test.java

C:\Users\Atish kumar sahu\desktop>java test
false
false
false
```

Variable:

It is a symbolic name refer to a memory location used to store values that can change during the execution of program.

int A = 10 ; -> the whole is called as variable declaration

int = data type

A = variable name/identifier

10 = value of the variable

Identifier:

Identifier are names assigned to variables, constants, methods, classes, packages, and interfaces.

The first character of an identifier must be a letter and underscore(_), or a dollar sign$. The subsequent characters can be a letter, an underscore, dollar sign or a digit. The white space is not allowed with in identifiers. Identifiers are case-sensitive. “Total” and “total” are different identifiers.

Example: Myjava, $amount, _cosplay, total_amont

Types Of Variable:

Based on type of value represented by a variable all variables are divided into two types

PRIMITIVE VARIABLE: it is a type of variable can be used to represent primitive values.

EXAMPLE: int a = 10 ;

REFERENCE VARIABLE: it is a type of variable can be used to refer objects.

EXAMPLE: student s = new student(); s --------> object

Based on position of declaration and behavior all variables are divided into 3 types.

1. INSTANCE VARIABLE 2. STATIC VARIABLE 3. LOCAL VARIABLE

Instance Variable:

If the value of a variable is varied from object to object such type of variables are called instance variables. For every object a separate copy of instance variable will be created. Instance variable should be declared with in the class directly but outside of any method, constructor and block.

Instance variable will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variable is exactly same as scope of object.

Instance variable will be stored in heap memory as a part of object. We can't access instance variable directly from static area but we can access by using object reference but we can access instance variable directly form instance area.

For instance variable JVM will provide default values and we are not required to initialization explicitly. Instance variable also known as object level variable or attributes.

```java
/*INSTANCE VARIABLE*/
public class JVP21 {

    int a = 10;
    double b = 20.33;
    // INSTANCE VARIABLES
    String s;
    boolean o;

    public static void main(String args[]) {
        JVP21 jvp = new JVP21();

        System.out.println(jvp.a); //10

        System.out.println(jvp.s); //null

        jvp.m1();
    }

    void m1() {
        System.out.println(b); //20.33

        System.out.println(o); //false
    }
}

```

```java
/*INSTANCE VARIABLE*/
public class JVP21 {

    int a = 10;
    double b = 20.33;
    // INSTANCE VARIABLES
    String s;
    boolean o;

    public static void main(String args[]) {

        System.out.println(a);

    }


}
```

```
<terminated> JVP21 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe (Mar 5, 2022, 7:37:57 PM – 7:38:00 PM)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Cannot make a static reference to the non-static field a

        at JVP21.main(JVP21.java:12)
```

Static Variable:

If the value of a variable is not varied from object to object then it is not recommended to declare variable as instance variable we have to declare such type of variable at class level by using static modifier.

In case of instance variables for every object a separate copy will be created but in case of static variable a single copy will be created at class level and shared by every object of class.

Static variable should be declared with in the class directly but outside of any block, method, or constructor. Static variable will be created at the time of class loading and destroyed at the time of class unloading hence scope of static variable is exactly same as a scope of (.class) file.

Static variable will be stored in method area. We can access static variables either by object reference or by class name but recommended to use class name with in the same class it is not required to use class name and we can access directly.

We can access static variable directly from both instance and static areas. For static variables JVM will provide default values and we are not required to perform initialization explicitly static variable also known as class level variable or fields.

```java
/*STATIC VARIABLE*/
public class JVP22 {

    static int a = 10;
    static double b = 25.314;
    static String c;

    static int e = 1425;

    public static void main(String args[]) {
        JVP22 jvp = new JVP22();
        System.out.println(JVP22.a); // recommended = 10
        System.out.println(jvp.b); // not recommended = 25.314
        System.out.println(c); // null

        jvp.m1(); // 1425
    }

    void m1() {
        System.out.println(e);
    }
}
```

```java
/*STATIC VARIABLE VS INSTANCE VARIABLE TEST*/
public class JVP23 {
    static int a = 1452;
    int b = 523;

    public static void main(String args[]) {
        JVP23 jvp1 = new JVP23();
        jvp1.a = 600;
        jvp1.b = 1200;
        System.out.println(jvp1.a); // 600
        System.out.println(jvp1.b); // 1200

        JVP23 jvp2 = new JVP23();
        System.out.println(jvp2.a); // 600
        System.out.println(jvp2.b); // 523
    }
}
```

Local Variable:

To meet the temporary requirements of programmer we can declare variables inside a method or block or constructor. Such type of variables are called as local variable. There are numerous other names are there like temporary variable/automatic variable. Local variable is stored inside a stack memory area.

Local variable will be created while executing the block in which we declared it, once block execution is completed the local variable is also destroyed. Hence the scope of the local variable is the block in which we declared it.

For local variable JVM won't provide default values, compulsory we should perform initialization explicitly. Before using that variable that is if we not using then it is not required to perform initialization.

```java
/*LOCAL VARIABLE*/
public class JVP24 {
    public static void main(String args[])
    {
        int i = 10;
        for (int j = 0 ; j < 3 ; j++)
        {
            i = i + j ;
        }
        System.out.println(i); //13
        //System.out.pritnln(j);
    }
}

```

*JVP24.java

```java
/*LOCAL VARIABLE*/
public class JVP24 {
    public static void main(String args[]) {
        int i = 10;
        for (int j = 0; j < 3; j++) {
            i = i + j;
        }
        System.out.println(i);
        System.out.pritnln(j);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe  (Mar 5, 2022, 1
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        j cannot be resolved to a variable

        at JVP24.main(JVP24.java:11)
```

```java
/*LOCAL VARIABLE*/
public class JVP24 {
    public static void main(String args[]) {
        int i;
        System.out.println(i);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe  (Mar 5, 2022, 10
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The local variable i may not have been initialized

        at JVP24.main(JVP24.java:5)
```

```java
/*LOCAL VARIABLE*/
public class JVP24 {
    public static void main(String args[]) {

        try
        {
            int a = Integer.parseInt("ten");
        }
        catch(NumberFormatException e)
        {
            a = 10 ;
        }
        System.out.println(a);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe  (Mar 5, 2022, 10
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        a cannot be resolved to a variable
        a cannot be resolved to a variable

        at JVP24.main(JVP24.java:11)
```

```java
/*LOCAL VARIABLE*/
public class JVP24 {
    public static void main(String args[]) {

    int a ;

    if(args.length > 0)
    {
        a = 10;
    }
    System.out.println(a);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe  (Mar 5, 2022, 10
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The local variable a may not have been initialized

        at JVP24.main(JVP24.java:11)
```

```java
/*LOCAL VARIABLE*/
public class JVP24 {
    public static void main(String args[]) {

    int a ;

    if(args.length > 0)
    {
        a = 10;
    }
    else
    {
        a = 20 ;

    }
    System.out.println(a); //20
    }
}
```

it is not recommended to perform initialization for local variables inside logical blocks. Because there is no guarantee for the execution of these blocks always at runtime. It is highly recommended to perform initialization for local variables at the time of declaration at least with default values.

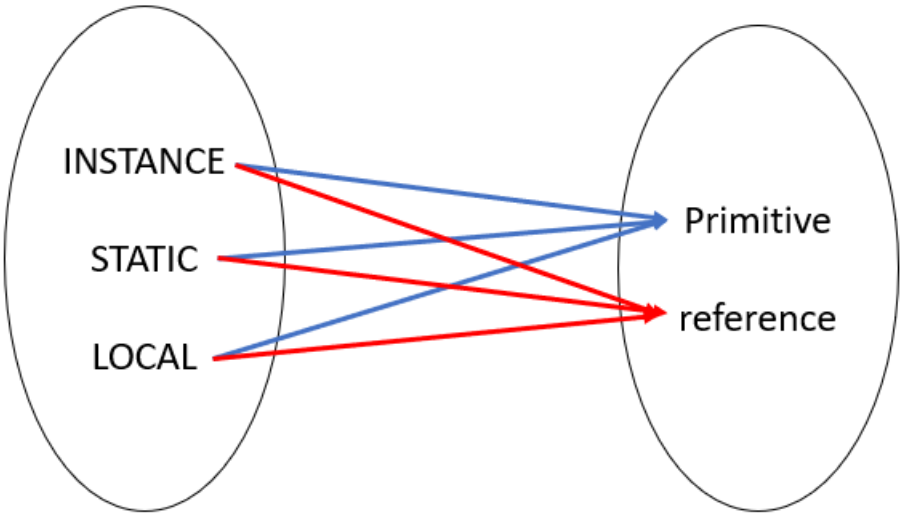
The only applicable modifier for local variable is final. If you trying to applying any other modifier then we get compile time error.

If you are not declaring with any modifier then by default it is default modifier but this is applicable for only instance variable and static variable not for local variable.

For instance variable and static variable JVM will provide default values and we are not required to perform initialization explicitly. For local variable JVM could not provide default values compulsory we perform initialization explicitly before using the variable.

Instance and static variable can be accessed by multiple threads simultaneously and hence these are not thread safe. In case of local variables for every thread a separate copy will created and hence local variable is thread safe.

Every variable in java should be either instance, static or local every variable in java should be either primitive or reference hence various possible combination in java are



Once we create an array every array element by default initialized with default values irrespective of whether it is instance static or local variable.

```
2  public class JVP24 {
3
4      /***** INSATNCE *****/
5      int[] a;
6
7      public static void main(String args[]) {
8
9          JVP24 t = new JVP24();
10         System.out.println(t.a);
11         System.out.println(t.a[0]);
12     }
13 }
```

Problems @ Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\jav
null
Exception in thread "main" java.lang.NullPointerException
        at JVP24.main(JVP24.java:13)
```

```java
public class JVP24 {

    /***** INSATNCE *****/
    int[] b = new int[3];

    public static void main(String args[]) {

        JVP24 t = new JVP24();
        System.out.println(t.b);
        System.out.println(t.b[0]);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8
[I@15db9742
0
```

```java
public class JVP24 {

    /***** STATIC *****/
    static int[] b ;

    public static void main(String args[]) {

        System.out.println(b);
        System.out.println(b[0]);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\ja
null
Exception in thread "main" java.lang.NullPointerException
        at JVP24.main(JVP24.java:10)
```

```java
public class JVP24 {

    /***** STATIC *****/
    static int[] b = new int[3];

    public static void main(String args[]) {


        System.out.println(b);
        System.out.println(b[0]);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.
[I@15db9742
0
```

```java
public class JVP24 {

    public static void main(String args[]) {

        /***** LOCAL *****/
        int [] a ;
        System.out.println(a);
        System.out.println(a[0]);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_291\bin\javaw.exe  (Mar 5, 2022, 11
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
	The local variable a may not have been initialized
	The local variable a may not have been initialized

	at JVP24.main(JVP24.java:7)
```

```java
public class JVP24 {

    public static void main(String args[]) {

        /***** LOCAL *****/
        int [] a = new int[3];
        System.out.println(a);
        System.out.println(a[0]);
    }
}
```

Problems Javadoc Declaration Console

```
<terminated> JVP24 [Java Application] C:\Program Files\Java\jre1.8.0_
[I@15db9742
0
```

Concept Of Reference Variable:

If you create reference variable within the method then you should have to use only within that created method. Any reference variable created within method those are called as local reference variable. Every reference variable which we will create holds memory address of a object and will be having data type as class name. if you create any reference within the method the scope of that reference only with in the method.

If you try to access local reference variable outside the area on where you created then you will get compilation error.

```java
/*REFERENCE VARIABLE*/
public class JVP25 {

    void get() {
        JVP25 obj1 = new JVP25();
        System.out.println("v2 : " + obj1); // JVP25@6d06d69c
    }

    public static void main(String args[]) {
        JVP25 obj = new JVP25();

        System.out.println("v1 : " + obj); // JVP25@15db9742

        obj.get(); // JVP25@6d06d69c
    }
}
```

```java
/*REFERENCE VARIABLE*/
public class JVP25 {
    static JVP25 obj1;

    void get() {
        System.out.println("v2 : " + obj1); // v2 : JVP25@15db9742
    }

    public static void main(String args[]) {

        System.out.println("v3 : " + obj1); // v3 : null

        obj1 = new JVP25();
        System.out.println("v1 : " + JVP25.obj1); // v1 : JVP25@15db9742

        obj1.get();

    }
}
```

```java
/*REFERENCE VARIABLE*/
class JVP25 {
    static JVP25 obj1;
    int num1 = 145;

    void get() {
        System.out.println("v2 : " + obj1); // v2 : JVP25@15db9742
        System.out.println("num1 : " + obj1.num1); // num1 : 145
    }

    void got() {
        System.out.println("v4 : " + JVP25.obj1); // v4 : JVP25@15db9742
    }

    public static void tea() {
        System.out.println("v5 : " + JVP25.obj1); // v5 : JVP25@15db9742
        System.out.println("num1 1 : " + obj1.num1); // num1 1 : 145
    }
```

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

        System.out.println("v3 : " + obj1); // v3 : null
        System.out.println("v5 : " + JVP25.obj1); //v5 : null
        System.out.println("v4 : " + JVP25.obj1); //v4 : null
        System.out.println("v1 : " + JVP25.obj1); //v1 : null
        System.out.println("v2 : " + obj1); //v2 : null

        obj1 = new JVP25();
        System.out.println("v1 : " + JVP25.obj1); // v1 : JVP25@15db9742

        obj1.get();
        obj1.got();
        obj1.tea();

    }
}
```

Once you create static reference variable at class level you can use that reference variable through out the class in both static and instance area. We no need to create an object multiple time to access instance member of the class within static area.

```
/*REFERENCE VARIABLE*/
class JVP25 {
    JVP25 obj;
    int num = 526;

    void get() {

        System.out.println("v1_obj : " + obj); // v1_obj : null
        System.out.println("num1.1 : " + num); // num1.1 : 526

        JVP25 obj2 = new JVP25();
        System.out.println("obj2 : " + obj2); // obj2 : JVP25@6d06d69c
        System.out.println("obj2_num1 : " + obj2.num); // obj2_num1 : 526
        System.out.println("----------------getmethod end----------------------");
    }

    void got() {
        System.out.println("v3_obj : " + obj); // v3_obj : null
        System.out.println("num1.3 : " + num); // num1.3 : 526

        JVP25 obj3 = new JVP25();
        System.out.println("obj3 : " + obj3); // obj3 : JVP25@6d06d69c obj3 : JVP25@7852e922
        System.out.println("obj3_num3 : " + obj3.num); // obj_num3 : 526
        System.out.println("-----------------got method end----------------------");
    }

    public static void tea() {
        // System.out.println("v4 : "+obj); //ERROR
        // System.out.println("num1.4 : "+obj.num);//ERROR

        JVP25 obj1 = new JVP25();
        System.out.println("obj1_v4 : " + obj1);// v4 : JVP25@6d06d69c obj1_v4 : JVP25@4e25154f
        System.out.println("obj1_num1.4 : " + obj1.num); // num1.4 : 526
        System.out.println("------------------static void tea method end--------------------");

    }
```

```
38      public static void main(String args[]) {
39
40          // System.out.println(obj); //error
41          // obj = new JVP25(); //ERROR
42
43          JVP25 obj = new JVP25();
44          System.out.println("obj_v2 : " + obj); // obj_v2 : JVP25@15db9742
45          System.out.println("obj_num1.2 : " + obj.num); // obj_num1.2 : 526
46          System.out.println("----------------main method end------------------");
47
48          obj.get(); // v1_obj : null num1.1 : 526
49
50          System.out.println("obj_v1 : " + obj); // v1 : JVP25@15db9742
51          System.out.println();
52
53          obj.got(); // v3_obj : null num1.3 : 526
54
55          System.out.println("obj_v3 : " + obj); // v3 : JVP25@15db9742
56          System.out.println();
57
58          obj.tea(); // v4 : JVP25@6d06d69c obj1_v4 : JVP25@4e25154f num1.4 : 526
59          System.out.println("obj_v4.4 : " + obj); // v4.4 : JVP25@15db9742
60
61      }
62  }
```