🎯

JAVA BASICS

INTRODUCTION TO JAVA

- Object-Oriented Programming Language .
- Developed in 1991 & released in 1995 .
- Founder : James Gosling .
- Developed by Sun Microsystems of USA .
- Currently owned by Oracle .

FEATURES OF JAVA

- Platform - Independent .
- Object-Oriented .
- Simple .
- Secured .
- Robust .
- Portable .

- Supports Multithreading .
- Compiled & Interpreted both .
- High Performance .

WORKING OF JAVA

```
               javac                          jvm
Source Code  -------------->  Byte Code  -------------->  Machi
( .java )     compiler        ( .class )   interpreter     ( .ex
```

JDK vs JRE vs JVM

▼ JDK

- Java Development Kit
- contains JRE + various development tools .
- contains javac compiler , debugger , jar archiver , etc .
- needed to develop and run java programs .

▼ JRE

- Java Runtime Enviornment .
- contains JVM + various libraries , deployment tools , gui toolkit , etc.
- needed to run & execute java programs .
- provides enviornment to run & execute java programs .

▼ JVM

- Java Virtual Machine
- contains JIT + Interpreter + Class Loader + Bytecode Verifier + Garbage Collector , etc.
- executes java code line by line .
- abstract machine or a java interpreter that converts bytecode to machine code .

- different for each machine .

> We need to download JDK & install it on our OS . Only then we can run java programs .

💡 JDB - Java Debugger

IDE

- It stands for Integrated Development Enviornment .
- It is a software that helps to write code , edit code , compile and run code , all these stuffs at one place .
- Various IDEs are there : VS Code , Intellij IDEA , Eclipse , Netbeans , etc.

BASIC STRUCTURE OF JAVA PROGRAM

```
class Hello {                           // defining or declaring
        public static void main(String[] args){   // starting po
                    System.out.println("Hello World");  // print
        }
}
```

1. **public :** access modifier , it tells that this class or method is accessible from anywhere .
2. **class :** keyword , it defines or declare the class .
3. **Hello :** class name .
4. **static :** keyword , it tells that method can be accessed without creating instance of the class .
5. **void :** keyword , a return type . It tells that the method does not return anything .

6. **main() :** method name , it is the starting point of the program execution . It is the entry point to the application .
7. **String[] args :** command line argument . It is an array of String type values .
8. **System :** a class of Printstream type .
9. **out :** a variable of Printsream class .
10. **println() :** method of Printstream , to print the passed arguments on the console .
11. **“Hello World” :** It is the String value that is passed as argument in println() method .

NAMING CONVENTIONS

▼ **Pascal Convention**

- First letter of each word should be capital .
- For classes , interfaces , etc. we use this .
- ex. Car , MyCar , FirstEmployeeClass , etc.

▼ **camel Case Convention**

- First letter of first word should be small , afterwards first letter of each word should be capital .
- For variables , methods , objects , etc. we use this .
- ex. car , myCar , firstEmployeeVar , etc.

> File name and Public Class name must be same .

> Only one public class can be there in the file .

> Not mandatory to have a public class in the file .

JAVA EDITIONS

- J2SE (Java Standard Edition) = used in desktop apps , standalone applications , etc.
- J2EE (Java Enterprise Edition) = used in websites , web apps , etc.
- J2ME (Java Micro Edition) = used in mobile apps , embedded systems , etc.
- Java FX = used in rich GUI .

COMMENTS

Some statements that are not executed by compiler , those statements are called comments . They are used to improve readibility of code . Programmer write comments for their understanding .

TYPES OF COMMENTS

1. Single Line Comment : we use //
2. Multi Line Comment : we use /*.....*/
3. Documentation Comment : we use /** ..... */

While writing a Java program , we have to follow some set of rules , that is called syntax in a programming language .

VARIABLE

- It is a container that is storing some data value in it .
- It is a name given to a memory location that is storing some data value .
- Its value can be changed during execution of Java program .

```
dataType varName = value;          // Syntax

int age = 20;                      // Examples
char ch = 'd';
```

- As Java is a statically typed language , so it is must to declare variables before using them .

RULES FOR NAMING VARIABLES

1. It can’t start with digits .
2. It can start with alphabets , $, _ character .
3. It cannot be a keyword .
4. It must not contain any white space .
5. They are case - sensitive .
6. It can contain alphabets , digits , $, _ character .

```
4age , my age , int     // INVALID NAMES
age , myAge , value4     // VALID NAMES
```

KEYWORDS

- There are some words that have been reserved by the complier , called keywords .
- They have some pre-defined meaning to compiler .
- They cannot be used as an variable name , class name , method name , etc.

abstract	double	instanceOf	private	try
assert	extends	int	return	transient
byte	enum	interface	static	void
boolean	final	long	short	volatile
break	finally	native	switch	while
char	float	new	synchronized	do
class	for	null	super	
continue	if	package	this	
catch	implements	public	throw	

case	import	protected	throws	

DATA TYPES

- They tell us about the type of the data that we are going to use in our program . They also tell how much space they take in memory .
- Two types of data types :
  1. Primitive (or Intrinsic or Primary)
  2. Non-Primitive (or Extrinsic or Secondary or Refrenced)

PRIMITIVE DATA TYPES :

- byte : takes 1 byte
- short : takes 2 byte
- int : takes 4 byte
- long : takes 8 byte (l or L)
- float : takes 4 byte (f or F)
- double : takes 8 byte (default for floating numbers)
- boolean : takes 1 bit
- char : takes 2 bytes

> Range : `-2^(n-1)` to `2^(n-1) - 1` where n = no. of bits

NON-PRIMITIVE DATA TYPES :

- Class
- Object
- Array
- String
- Interface , etc.

DEFAULT VALUES :

💡 These are considered for Instance variables and Class variables .

- Integers : 0
- Decimals : 0.0
- Object : null
- Boolean : false
- Character : '\u0000'

LITERALS

- They are fixed values or constant values that can be directly used in the program .

TYPES OF LITERALS

1. Integer literal - 8
2. Float literal - 9.7f
3. Double literal - 7.8
4. Character literal - 'c'
5. Boolean literal - true
6. String literal - "Name"

> For binary numbers , we use 0b as prefix . For hexadecimal numbers , we use 0x as prefix . For octal numbers , we use 0 as prefix .

```
int num = 0b1010;      // binary number
int num2 = 0x5E;       // hexadecimal number

int value = 10_000_00_000;    // this can also be done
```

```
int num = 23e7;      // 23 * 10(pow)7
```

TYPES OF VARIABLES

LOCAL VARIABLES :

- They are declared and initialized inside a method , block , constructor , etc.
- They can only be used in that method , block , constructor , etc.
- In Java , they don’t have any default values .
- They are stored in stack memory .

INSTANCE VARIABLES :

- They are declared inside class but outside the method , constructor .
- They can be accessed in other classes and other methods by creating instance of this class .
- They are created when object is created , and they are destroyed when objects get destroyed .
- They have as many copies as many times objects are created .
- They stored in heap memory .

CLASS VARIABLES :

- They are also known as static variables . They are also known as shared variables .
- They are declared similarly instance variables but using static keyword .
- They are created when program starts , and they are destroyed when program ends .
- They have only one copy throught the class .
- They are stored in method area .

SCOPE OF VARIABLES

1. **When variable is defined inside method :**

   ⇒ Then variable cannot be accessed outside the method .

2. **When variable is defined outside method :**

   ⇒ Then variable can be accessed using instances .

```
public class Scope{
    int var = 7;         // defined outside method
    public static void main(String[] args){
            Scope obj = new Scope();   // through instance
            obj.var;

            int num = 9;
            {
                int z = 70;      // defined inside block
                num += 5;        // can be done
            }
            num=9;            // can be done
            z = 5;            // cannot be done
    }
}
```

OUTPUT IN JAVA

To display output on console in java , we have different methods for it :

1. System.out.println( ); ⇒ it prints the value and moves to next line
2. System.out.print( ); ⇒ it prints the value and keep on same line
3. System.out.printf( ); ⇒ it is used to print formatted data
4. System.out.format( ); ⇒ similiar to printf , used mostly in C,C++

> + operator is used to concatenate strings . It is used with println( ) and print( ) .

> , is used to concatenate , when used with printf( ) and format( ) .

FORMAT SPECIFIERS

1. %d = for integers
2. %f = float values
3. %c = char values
4. %b = boolean values
5. %s = string values , etc.

TAKING INPUT FROM USER

- To take input from the user , we import Scanner class of java.util package .
- Then create object of that Scanner class .
- We use System.in , as it is used to read input from keyboard .`
- Then we call methods according to the needs through that object .

for ex. to take integer input , we use .nextInt( ) and similairly other methods .

```
import java.util.Scanner;

Scanner sc = new Scanner(System.in);

sc.nextInt( );            // for integer input ( within range of
sc.nextFloat( );          // for float input and large integer i
sc.nextDouble( );         // for double input
sc.next( );               // reads first word only
sc.nextLine( );           // reads whole String
...etc.
```

TYPE CONVERSION

- A process of converting one data type into another data type , is called Type conversion .
- It can be done explicitly (typecasting) and implicitly .

AUTOMATIC OR WIDENING TYPE CONVERSION :

- When we assign a smaller data type value into a bigger data type value , then it can be done automatically by javac compiler .
- Also called Implicit Type Conversion .

```
int i = 7;
float f = i;          // automatically done
```

EXPLICIT OR NARROWING TYPE CONVERSION :

- When we assign a larger data type value into a smaller data type value , then it can be done explicitly defining the data type .

```
float f = 7.8f;
int i = f;                // throws error

int i = ( int ) f;        // we can do this
```

> boolean and char not compatible , int and boolean also not compatible .

OPERATORS

- They are some special characters or symbols that are used to perform various operations on values & variables .

ex. 5 + 3 = 8 where , 5 and 3 are operands , and + is an operator , and 8 is the result value .

TYPES OF OPERATORS :

1. Arithmetic Operators ⇒ used to perform arithmetic operations . They can't work with booleans .

```
+ , - , * , / , %

+ ( adds two numbers )
- ( subtracts two numbers )
* ( multiplies two numbers )
/ ( divides two numbers , gives quotient )
% ( gives remainder ) ( can work on floats & doubles ) ( modulo
```

2. Assignment Operators ⇒ used to assign values & variables to other variables .

```
= , += , -= , *= , /= , %=
= ( assign values )
+= ( assign values after adding )
-= ( assign values after subtracting )
*= ( assign values after multiplicating )
/= ( assign values after getting )
%= ( assign values after getting remainder )

int a;
a+=2;         // a=a+2;
```

3. Relational / Comparison Operators ⇒ used to compare values & variables .

```
== , != , > , >= , < , <=
== ( equality check )
!= ( inequality check )
>  ( checks a number is greater than other or not )
>= ( checks a number is greater than or equals to other or not
<  ( checks a number is smaller than other or not )
<= ( checks a number is smaller than or equals to other or not
```

4. Logical Operators ⇒ used to evaluate expressions . They work with boolean expressions .

```
&& , || , !
&& ( returns True only if both expressions True )   Logical And
|| ( returns True if either expression gets True )  Logical Or
!  ( inverts the boolean result )`                  Logical Not
```

5. Bitwise Operators ⇒ used to perform operations on bits .

```
& , | , ^ , ~ , << , >> , >>>
& ( returns 1 if both bits are 1 )
| ( returns 1 if either bit is 1 ) ( inclusive OR )
^ ( returns 0 if both bits same )  ( exclusive OR )
~ ( inverse the bits )
<< ( shitfing the bits to left )
>> ( shifting the bits to right , leftmost vacant bit filled dep
>>> ( similiar to right shift , but leftmost vacant bit filled w
```

6. Unary Operators ⇒ used to perform operation on single operand .

```
+ , - , ! , ~ , ++ , --
+   ( indicates the positive value )
-   ( changes the positive value to negative and vice versa )
!   ( inverse the result of boolean )
~   ( complement of number )
++  ( increment )
--  ( decrement )

varName++ : post increment  ( firstly value is used and then inc
++varName : pre increment   ( firstly value increment and then u
```

PRECEDENCE & ASSOCIATIVITY

- When there are multiple operators are in an expression , then they are evaluated on the basis of precedence .
- Higher the precdence of operator , that operator will be evaluated firstly .
- But , if some operators having same precedence , then they are evaluated using associativity .
- Associativity gives the direction in which the operators will be executed either Right to Left or Left to Right .
- Mostly all operators have associativity left to right except assignment operators and pre - increment operators .

OPERATORS	PRECEDENCE	ASSOCIATIVITY
( )	Parenthesis	Left to Right
a++ , a- -	Post increment & decrement	||
++a , ++a , unary	Pre increment& decrement	Right to Left
* , / , %	Arithmetic	Left to Right
+ , -		||
>> , << , >>>	Shift	||
== , ! = , > < , > = , < =	Relational	||
&	Bitwise	||
^		||
|		||
&&	Logical	||
||		||
= , += , -= , *= , /= , %=	Assignment	Right to Left

RESULTING DATA TYPE AFTER ARITHMETIC OPERATIONS

```
int           operator     int           =>    int
int           operator     float/double  =>    float/double
int           operator     char          =>    int
```

```
float          operator      double        =>    double
float/double   operator      char          =>    float/double
```

OVERFLOW AND UNDERFLOW

- Overflow occurs when we try to assign a larger value to a variable than its range .
- Underflow occurs when we try to assign a smaller value to a variable than its range .
- Compiler doesn't throw any error , rather it changes the result .

```
byte b = (byte) 256;        // overflow
byte b1 = (byte) -300;      // underflow
```

CONDITIONAL STATEMENTS

- When we have to take some decisions that depend on certain conditions , then we can take those decisions with the help of conditional statements .
- Two Types of Conditional Statements :
  1. If - else statement
  2. switch statement

IF - ELSE STATEMENT :

```
if( Condition ){
        // statements          ---- gets executed when Condition
}
else {
        // statements          ---- gets executed when Condition
}
```

- else block is optional .

- Condition can either be True or False.

> = is used for assigning values , and == is used for equality check .

IF-ELSE ELSE-IF LADDER :

```
if( Condition 1 ){
        // statements       ---- gets executed when Condition 1
}
else if( Condition 2 ){
        // statements       ---- gets executed when Condition 2
}
else if( Condition 3 ){
        // statements       ---- gets executed when Condition 3
}
else{
        // statements       ---- gets executed when all above Co
}
```

- Instead of using multiple if statements , we can use if-else else-if ladder .
- It reduces indentation .
- We can use as many as else if statements .

> Ternary Operator is used as shortcut for if-else statement

```
int num = 7;
int var = 0;

// Using if-else statement

if ( num%2==0 ){
        var = 1;
}
```

```
else {
        var = 10;
}

// Using ternary operator

int var = ( num%2==0 ) ? 1 : 10;
```

SWITCH STATEMENT :

- Switch case is used when we have to choose or select from multiple alternatives or choices .

```
switch( varName ){

        case value1 : // statements     ---- gets executed when
                                    break;
        case value2 : // statements     ---- gets executed when
                                    break;
        case value3 : // statements     ---- gets executed when
                                    break;
        default : // statements         ---- gets executed when

}
```

- A switch variable can only be int , char , String .
- There can be nested switch statements but it is rare .

ENHANCED SWITCH STATEMENT :

```
switch( varName ){

        case value1 -> // statements     ---- gets executed when
        case value2 -> // statements     ---- gets executed when
        case value3 -> // statements     ---- gets executed when
```

```
        default : // statements          ---- gets executed when

}
```

GENERATE RANDOM NUMBER

```
import java.util.Random;              // import Random class fro

Random obj = new Random();            // create instance of Rando
int var = obj.nextInt(m);             // use nextInt() method

// it can generate any random number from 0 to m-1 .
```

LOOPS

- They are used when we have to repeatedly execute some set of statements .
- Ex. Printing Hello 500 times , Printing First 1000 Natural Numbers , etc.
- Types of loops :
    1. while loop
    2. do-while loop
    3. for loop

while LOOP : (it is used when we don't know about no. of iterations)

```
while( Condition ){
        // statements                   ---- gets executed as lo
}
```

do - while LOOP : (it is used when we have to execute loop atleast once)

```
do{
        // statements            ---- gets executed atleast once
}while( Condition );
```

for LOOP : (it is used when we know about no. of iterations)

```
for( initialization ; condition ; updation ){
        // statements               ---- gets executed as long a
}
```

INFINITE LOOP :

- When the Condition never gets False , always keep True , then loop executes infinite times .
- Called as Infinite loop .

while vs do - while LOOP :

- while loop ⇒ firstly condition is checked , then statements get executed on the basis of it .
- do-while loop ⇒ firstly statements get executed atleast once without checking condition , after that statements execute on the basis of condition .

BREAK STATEMENT

- In Java , break is a keyword . But it is also used as a statement .
- It is used to terminate the sequence or flow or loop that is going on .
- It is used to get exit from that sequence or loop , we use break keyword .

CONTINUE STATEMENT

- In Java , continue is a keyword . It is also used as a statement .
- It is used to skip the current iteration or current step or current stage and moves to next iteration .

- The code below it does not get executed .

```
```

STRINGS

- In Java , String is an object that represents sequence of characters .
- A character array work same as a string .
- java.lang.String that is used to create a string object .

WAYS TO CREATE STRINGS :

1. Using String literal

```
String str = "Welcome";

// firstly JVM checks current String object is present in String
// If it is present , then reference of that object is returned
// But if it is not present , then new object is created in pool


// it makes java more memory efficient
```

2. Using new keyword

```
String str = new String ("Welcome");

// it will create new String object in the heap memory .
// and the reference str will point to this object in heap memor
// even if same object is present in String pool , then also new
```

- Strings are immutuable .
- Although it is a class but due to its high usage , it is used as a priimitive .
- To compare Strings in java , we have some ways :
    1. using equals( ) method : (it compares values)

2. using == operator : (it compares references not values)
3. using compareTo( ) method : (it compares values lexicographically)

   Suppose s1 and s2 are two String objects. If:

   - **s1 == s2** : returns 0.
   - **s1 > s2** : returns a positive value.
   - **s1 < s2** : returns a negative value.

STRING METHODS :

- In Java , various methods are supported by Strings .
- In Java Strings , indexes begins from 0 and goes till length-1 .

```
String str = "Welcome"; // length = 7 , indexes = 0 to 6
             0123456  // indexes
```

METHODS	USES
.length( )	returns the length of String
.toLowerCase( )	returns new String of lowercase letter
.toUpperCase( )	returns new String of uppercase letter
.trim( )	returns new String after removing leading & trailing spaces .
.substring(starting index)	returns substring from passed index to end .
.substring(start index , end index)	returns substring from passed starting index to passed end index-1.
.replace (prev , new)	returns new String after replacing prev char with new char .
.startsWith( )	returns True if String starts with the passed value .
.endsWith( )	returns True if String ends with the passed value .
.charAt( )	returns character present at passed index .
.indexOf( )	returns first occuring index of passed character .
.indexOf(char ,	returns first occurence of passed character starting from passed

index)	index .
.lastIndexOf( )	returns last occuring index of passed character .
.lastIndexOf(char , index)	returns last occurence of passed chracter starting from passed index .
.equals( )	returns True if values of Strings are equal .
.equalsIgnoreCase( )	returns True if values of Strings are equal , ignoring the cases .

...etc.

Strings are immutuable but StringBuffer and StringBuilder are mutuable .

ESCAPE SEQUENCE CHARACTERS

- They are combination of more than one characters .
- But they work as a single character when used witthin a String .
- They are characters written after backslash .

```
/n    : insert a new line
/t    : insert a tab space
/'    : insert a single quote
/"    : insert a double quote
//    : insert a backslash
/b    : insert a backspace
...etc.
```

ARRAYS

- Array is an object that contains elements of similiar data types .
- Array elements are stored in contiguous memory locations .
- It is a data structure or collection that contains similiar elements .
- We can store fixed number of elements in an array .

- Array indices begins from 0 and goes till size - 1 .

ADVANTAGES :

1. It makes acessing of elements easy .
2. It makes code optimized .
3. It makes sorting of data efficient .

DISADVANTAGES :

1. As it has fixed size , once its size is defined . It can’t be change .

TYPES OF ARRAYS :

1. Single Dimensional Array
2. Multi Dimensional Array

SINGLE DIMENSIONAL ARRAY

- Also called 1-D Array .

```
// declaration of 1-D array
dataType [] arrayName ;

// memory allocation or object creation of 1-D array
arrayName = new dataType [size];         // size = no. of elemer

// initialization
arrayName[index] = value;

// All above things at once
dataType[] arrayName = { value1 , value2 , value3 };
```

FOR-EACH LOOP :

- In Java , arrays have a length property that tells the size of the array or no. of elements in the array .

```
// Display Array without for - each loop

for(int i=0; i<size; i++){
    System.out.print(arrayName[i]+" ");
}


// Display Array using for - each loop

for(dataType elem : arrayName){
    System.out.print(elem+" ");
}
```

- Java supports anonymous arrays . Anonymous array are those arrays who don’t have any name and don’t have any reference . They are created just for instant use .

```
new dataType[]{value1 , value2 , value3 };
```

- JVM throws ArrayIndexOutOfBoundsException , if we try to access the index which is out of range of array indices . If we declare the size of array as negative , then also this Exception occurs .

MULTI-DIMENSIONAL ARRAYS

- Mostly , we are going to see 2-D array only .
- They are an array which contains arrays .Their elements are an array itself .
- It can be treated as a matrix with rowsa and columns .

```
// declaration
dataType[][] arrayName ;

// memory allocation or object creation
arrayName = new dataType[rows][columns];           // rows : siz
```

```
                                                    // rows * columns =

// initialization
arrayName[row][column] = value;

// All above things at once
dataType[][] arrayName = { {value1 , value2 , value3} , {value1


// Display 2-D Array Using for loop

for(int i=0; i<rows; i++){
    for(int j=0; j<columns; j++){
            System.out.print(arrayName[i][j]+" ");
    }
    System.out.println();
}


// Display 2-D Array using for-each loop

for(dataType[] elem : arrayName){
    for(dataType value : elem ){
        System.out.print(value+" ");
    }
    System.out.println();
}
```

JAGGED ARRAY

- In 2-D Array , when we don’t specify no. of columns of a matrix , only specifies no. of rows in matrix . Then such type of array are called Jagged array .
- Then there you have to explicitly specify size of coulmn for each row .

```
int[][] arr = new int[3][];

arr[0] = new int[4];
arr[1] = new int[2];
arr[2] = new int[3];

for(int i=0; i<arr.length; i++){
    for(int j=0; j<arr[i].length; j++){
            // Body of loop
    }
}

OR

int[][] arr = {{8,7} , {6,7,9} , {5,3,5} };
```

ARRAY OF OBJECTS

```
ClassName Object1 = new ClassName();
ClassName Object2 = new ClassName();
ClassName Object3 = new ClassName();

ClassName [] arrayName = new ClassName [size];
arrayName[0] = Object1;
arrayName[1] = Object2;
arrayName[2] = Object3;
```

MEMORY MANAGEMENT

- Java handles memory management automatically . JVM divides memory in two parts - Stack Memory and Heap Memory .

STACK MEMORY :

- It stores local variables , methods , and reference variables of objects .
- It follows LIFO order .
- It is smaller in size .
- It is not flexible , once memory allocated , can’t alter it .
- It has faster access , allocation , deallocation .
- Compiler automatically do it .
- Memory allocation is continuous .
- java.lang.StackOverFlowError occurs if memory gets finished .

HEAP MEMORY :

- It stores actual objects and JRE classes .
- It doesn’t follow any order as Dynamic Memory Allocation occurs .
- It is larger in size .
- It is flexible .
- It has slower access , allocation , deallocation .
- Programmer manually do it .
- Memory allocation is random .
- java.lang.OutOfMemoryError occurs if memory gets finished .

FUNCTIONS

- Functions are some blocks of code that are used to perform any specific tasks .
- They organizes the code , makes the code reusable .
- It implements DRY principle .
- In Java , there is only pass by value or call by value .
- Only copy of reference variable is passed .

DEFINING FUNCTION :

```
accessModifier  returnType  functionName( ){
        // function ka code
}
```

CALLING FUNCTION :

```
functionName( );
```

- A function only executes when it is called .

PASSING PARAMETERS :

- A function can take any number of parameters .
- Sometimes , functions don’t take parameters .

```
accessModifier  returnType  functionName( parameters ){
        // function ka code
}
```

PASSING ARGUMENTS :

- Arguments are the actual values that are passed to the function .

```
functionName(arguments);
```

- When functions are defined inside classes , then they are called as methods . In Java , we call methods , not functions . These both terms are used interchangebly .
- In Java , the most important method is main( ) method . It is the starting point of execution of program .

```
 class Employee{
       public void addNumbers(int n1 , int n2){       // non st
```

```
                System.out.println(n1+n2);
        }
        public static void main(String[] args){       // static
                addNumbers(5,6);   // will give error

                Employee e1 = new Employee();
                e1.addNumbers(5,6);      // give desired output
        }
    }
```

- We can't access a non-static field from a static field directly . To access that field , create object of that class and then through object , call the method .

RETURN TYPE :

- It tells that what the values method will return after get executed .
- It can return any value or it can't return anything .
- There are various return types :
    1. int : return an integer value .
    2. String : return a String .
    3. void : does not return anything .

    ....etc.

METHOD OVERLOADING

- When two or more methods have same name but having different parameters , then they are overloaded methods .
- It can be implemented by keeping same method names but different parameters .
- Different parameters means :
    1. different number of parameters
    2. different dataType of parameters

3. different sequence of parameters

- By changing return type and keeping same method name , method overloading can’t implemented .
- It minimizes or reduces complexity of code .
- It increases readibility of code .

```
public void add(){                          --------
    int n1=7 ;                                      |
    int n2=9;                                       |
    System.out.println(n1+n2);                      |
}                                                   |  Overloaded Me
                                                    |
public void add(int n1 , int n2){                   |
    System.out.println(n1+n2);                       ---------
}

public int add(){                          ----- Not Overloaded
        // code
}
```

VARIABLE ARGUMENTS (VarArgs)

- When we want that the method can take variable arguments if they are passed in the same method . Then concept of VarArgs is used .

```
accessModifier  returnType  methodName( dataType ... parameterNa
        // function ka code
        // parameter will be treated as an array
}


// calling method
methodName();                              -----
```

```
methodName(value1);                          | all are valid
methodName(value1 , value2);                 |
methodName(value1 , value2 , value3);  -----|


// if we want atleast one paramter in our method
accessModifier  returnType  methodName( dataType parameter , dat
        // function ka code
        // parameter will be treated as an array
}

// calling method
methodName();            ---- invalid
methodName(value1);                     ------
methodName(value1 , value2);                 |   all are valid
methodName(value1 , value2 , value3);  -----|
```

RECURSION

- When a method calls itself , then it is a recursive method .
- This process is Recursion .

```
Example :             factorial(n)  =     n  *  factorial(n-1)
                      sumNatural(n) =     n  +  sumNatural(n-1
                      fibonacci(n)  =     fibonacci(n-1) + fibo
                      ...etc.
```

COMPILER vs INTERPRETER

Compiler

- translate entire source code into machine code .
- if any error occurs , then no execution .

Interpreter

- translate code line by line while program is running .
- if any error occurs ,then partial execution .

- compiled languages are faster .
- ex. C , C++ , C# , etc.

- interpreted languages are slower .
- ex. Python , Javascript , Ruby , etc.

Java is a hybrid language - both compiled & interpreted .