

## **JSpider**

## **JAVA**

### **JAVAC (JAVA COMPILER)**

- It checks the given program to find syntax mistakes.
- If there is any syntax mistakes then java compiler throws an error.
- The errors which is throw by javac are called as compile time errors.
- The java compiler generates .class file if there are no syntax errors in the program.

### **JVM (JAVA VIRTUAL MACHINE)**

- JVM is an interpreter which is going to
  - (i) Read one line of code.
  - (ii) Understand it.
  - (iii) Executes it.
- If JVM is not able to understand a line of code then JVM throws an error at runtime.
- The errors thrown by JVM are called as runtime errors or exceptions.

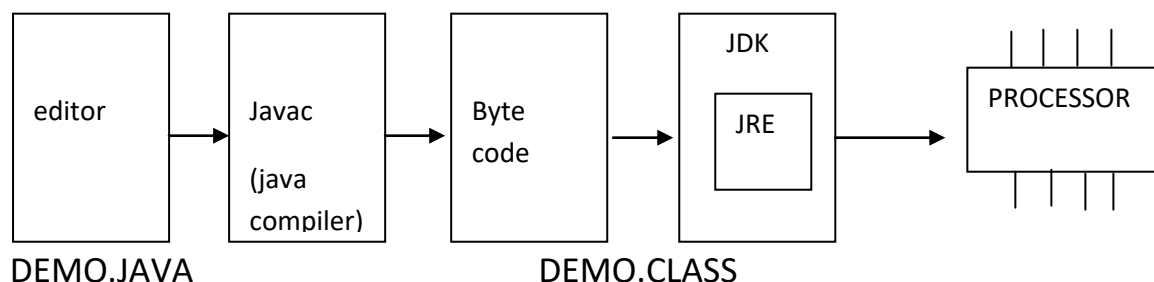
### **JRE (JAVA RUNTIME ENVIRONMENT)**

- It is used to setup the environment in with the help of operating system to execute the program of JVM.

### **JDK (JAVA DEVELOPMENT KIT)**

- It is a software package which contains java compiler, jvm and other necessary files which are required to compile and executes java programs.

### **JAVA WORK FLOW**



# CHAPTER 1 : KEYWORDS, IDENTIFIERS & VARIABLES

## KEYWORDS

- They are reserved words which have a predefined meaning.
- It is not possible to change the meaning of keywords in any programming language.
- Keywords are used to define a class, declare a variables or define a method etc. in java.

## Keywords in Java

Abstract	Continue	For	New	Switch
Assert ***	Default	Goto*	Package	Synchorinised
Boolean	Do	If	Private	This
Break	Double	Implements	Protected	Throws
Byte	Else	Imports	Public	Throws
Case	Enum****	Instanceof	Return	Transient
Catch	Extends	Int	Short	Try
Char	Final	Interface	Static	Void
Class	Finally	Long	Strictfp**	Volatile
Const*	Float	Native	Super	while

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0

## IDENTIFIERS :

- It is used to identify class or interface, methods, variables etc.

## Rules of Identifiers :

- Identifiers can be alphanumeric.
- Identifiers should not start with numbers.
- \$ and \_ (underscore) are the only two special character allow.
- An identifier can start with \$ and \_.
- Keywords can not be used as identifiers.

## VARIABLES :

- A variables is a named memory location which holds the value for the program.
- To use a variables we have to follow three steps :
  1. Declaration
  2. Initialization
  3. Utilization (Usage)

### Declaration of a variables :

#### Syntax :

**Datatype variablename;**

**Eg : int age;**

- It is a statement which defines what type of data will be stored in the given variables.

### Initialization of a variable :

#### Syntax :

**Variable=value;**

**Eg: age=25;**

- It is a statement which is written to store the value into the variable using assignment operator.

### Utilization of a variables:

- Statements which are written to use the values of the variables are called as utilization.

## Primitive Data types in java

				Default value
<b>Integer</b>	Byte	8 bits	1 byte	0
	Short	16 bits	2 byte	0
	Int	32 bits	4 bytes	0
	Long	64 bits	8 bytes	0l
<b>Decimal</b>	Float	32 bits	4 bytes	0.0f
	Double	64 bits	8 bytes	0.0

	Char	16 bits	2 bytes	Blank space
	Boolean	8 bits	1 bytes	false

## CHAPTER 2 : OPERATORS

- It performs operations on the given operands and produce results.

### Arithmetic Operator

- To perform any arithmetic operator
- $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$

Note- Variables of the program should not be printed with “double quotes”.

Note – System.out.print – prints & keep cursor in same line.

System.out.println – Prints & makes the cursor to next line.

### Formula to get how much a datatype maximum and min value range

$$-2^{n-1} \text{ to } 2^{n-1}-1$$

Eg : for bytes

$$-2^7 \text{ to } 2^7-1$$

$$-128 \text{ to } 127 \text{ (bytes)}$$

If we perform any arithmetic operation with the help of arithmetic operators with the help of arithmetic operators the result variables has to be decided by the following method.

**Max(int, type of 1<sup>st</sup> operand, type of 2<sup>nd</sup> operand)**

### Program :

(a)

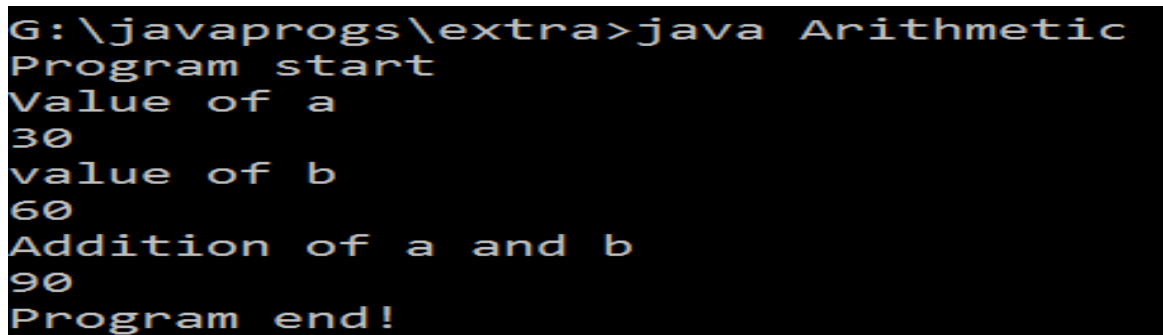
```
class Arithmetic
{
    public static void main(String[] args)
    {
        System.out.println("Program start");
        //declaration
        int a;
        int b;
```

```

int res;
//initialization
a=30;
b=60;
res=a+b;

        System.out.println("Value of a");
            System.out.println(a);
        System.out.println("value of b");
            System.out.println(b);
        System.out.println("Addition of a and b");
            System.out.println(res);
        System.out.println("Program end!");
    }
}

```



```

G:\javaprogs\extra>java Arithmetic
Program start
Value of a
30
value of b
60
Addition of a and b
90
Program end!

```

(b)

```

class Arithmetic2
{
    public static void main(String[] args)
    {

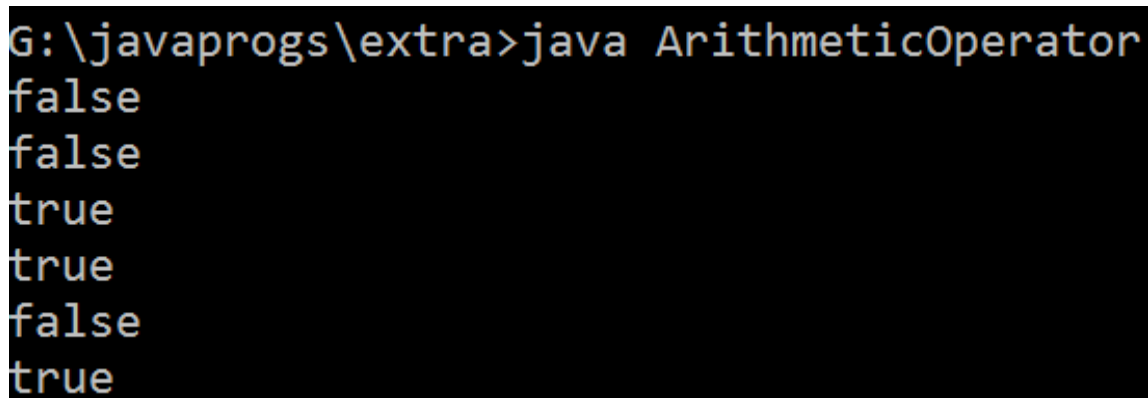
        int a=10; //declaration and initialization at same place
        int b=20;
        int c=a+b;

        System.out.println("value of a"+a);
        System.out.println("Value of b"+b);
        System.out.println("Addition of a and b"+c);
    }
}

```

(c)

```
class ArithmeticOperator
{
    public static void main(String[] args)
    {
        int a=5;
        int b=10;
        System.out.println(a>b);
        System.out.println(a>=b);
        System.out.println(a<b);
        System.out.println(a<=b);
        System.out.println(a==b);
        System.out.println(a!=b);
    }
}
```



```
G:\javaprogs\extra>java ArithmeticOperator
false
false
true
true
false
true
```

### Concatination Operator

- It helps in concatenating (joining) a string value with any other value.

### Combination of Concatination

- 2+ "hello" – 2hello
- "hello"+123 – hello123
- "hello" + "World" – helloworld
- 2+3+ "hello" – 5hello
- "hello"+2+3 – hello2+3 – hello23
- 2+ "hello"+3 – 2hello3

### Increment & Decrement Operator :

- Those operators are used to increase or decrease the value of a variable by 1 units.

## Increment Operator (++) :

- There are two types of increment operator.

### 1. Pre-increment.

### 2. Post-increment.

- If you write post or pre increment operators with a variables independently without any mathematical of expressions then, both operators will have same results.

Pre Increment	Post increment
First increment	Substitute
Substitute	Perform operation
Perform operation	Increment value

- Increment operator can not be used with Boolean datatypes.
- Any value can not be used directly with increment operators.
- Note :

```
Char a1= 'A';  
a1 = a1+1; //can not write like this  
a1=66;  
a1= 'B'
```

- **V.V.I notes**

<b>Byte b1=10; B1++; S.o.p(b1);</b>	<b>Byte b1=10; B1=b1+1; s.o.p(b1);</b>
<b>Does not show error</b>	<b>Show error just because of max function</b>

## Program

(a)

```
class Increment  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Program start");  
        int a1=10;  
        int b1=10;  
        int res;  
        System.out.println("a1 = "+a1);  
        System.out.println("b1 = "+b1);  
    }  
}
```

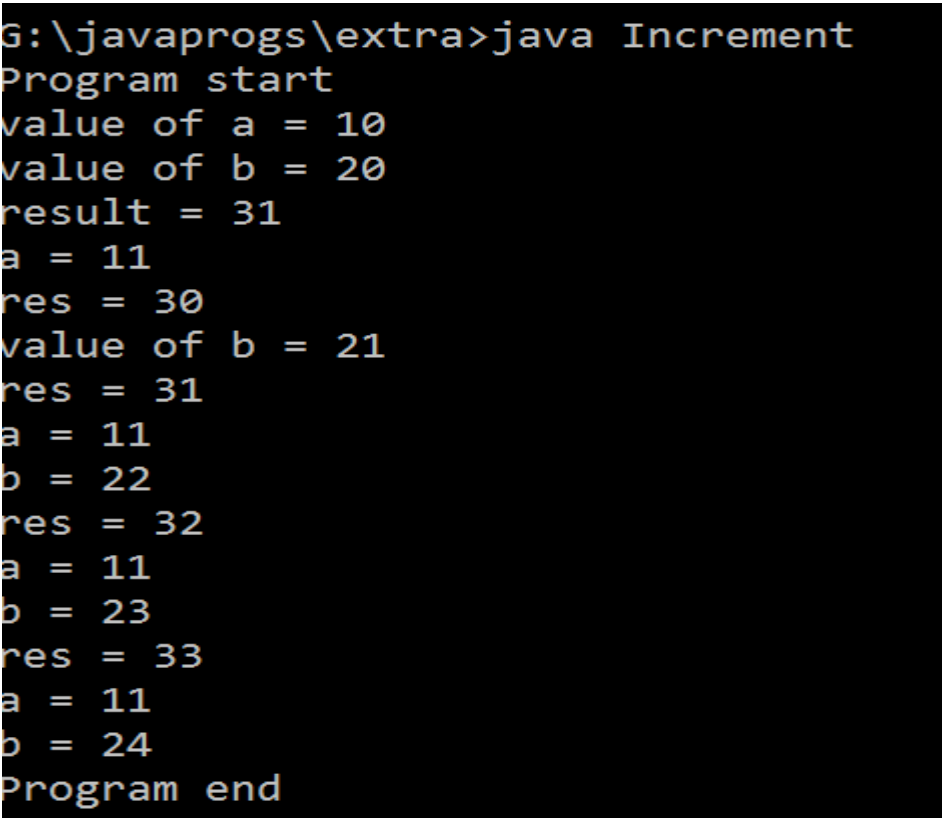
```

        res=++a1 + 10;
        System.out.println("res = "+res);

        res=b1++ + 10;
        System.out.println("res = "+res);
        System.out.println("a1 = "+a1);
        System.out.println("b1 = "+b1);

        res=b1++ + 10;
        System.out.println("res = "+res);
        System.out.println("a1 = "+a1);
        System.out.println("b1 = "+b1);
        res=b1++ + 10;
        System.out.println("res = "+res);
        System.out.println("a1 = "+a1);
        System.out.println("b1 = "+b1);
        System.out.println("Program end");
    }}

```



```

G:\javaprogs\extra>java Increment
Program start
value of a = 10
value of b = 20
result = 31
a = 11
res = 30
value of b = 21
res = 31
a = 11
b = 22
res = 32
a = 11
b = 23
res = 33
a = 11
b = 24
Program end

```



(b)

```
class Increment1
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
        int a1=10;
        System.out.println("a1 = "+a1);
        a1++;
        System.out.println( "a1 = "+a1);
    }
}
```

```
G:\javaprogs\extra>java Increment1
Hello World!
a1 = 10
a1 = 11
```

(C)

```
class Increment2
{
    public static void main(String[] args)
    {
        int a1=10;
        System.out.println(a1++);
        System.out.println(++a1);
    }
}
```

```
G:\javaprogs\extra>java Increment2
10
12
```

(d)

```
class Increment3
{
    public static void main(String[] args)
    {
        byte b1=10;
        b1++;
        System.out.println(b1);} }
```

```
G:\javaprogs\extra>java Increment3
11
```

## Decrement Operator: (--)

- Those operators are used to decrease the value of a variable by 1 units.
- There are two types of decrement operator.
  3. Pre-decrement
  4. Post-decrement
- If you write post or pre decrement operators with a variables independently without any mathematical of expressions then, both operators will have same results.

Pre decrement	Post decrement
First increment	Substitute
Substitute	Perform operation
Perform operation	Increment value

- decrement operator can not be used with Boolean datatypes.
- Any value can not be used directly with decrement operators.
- Note :

```
Char a1= 'B';  
a1 = a1-1;  
a1=65;  
a1= 'A'
```

## CHAPTER 3 : FLOW CONTROL STATEMENT

- It is a statement which is used to control the execution flow of a program.
- There are two types of flow control statements.
  1. Branching Statement (Decision making Statement)
  2. Looping Statement

### Branching Statement (Decision Making)

- These statements are used to execute a group of statements based on a Boolean condition.
- The type of decision making are :-
  - (a) If statement
  - (b) If else statement
  - (c) If else if statement
  - (d) Switch case statement

## IF STATEMENT

- If statement executes gives groups of statements within its body only if the Boolean condition result is true.
- Syntax :

```
If(condition) – true
{
    Statement ;
}
```

## PROGRAM

```
class IfStatement
{
    public static void main(String[] args)
    {
        System.out.println("Program Start");
        int v1=50;
        if(v1>10)
        {
            System.out.println("v1 is greater than 10");
        }
        System.out.println("Program end");
    }
}
```

## IF-ELSE STATEMENT

- Once the statements written in if block will be executed only if the Boolean condition is true and if Boolean is false then statements of else block will be executed.
- **Syntax:**

```
If(condition) true
{
    Statement;
}
Else
{
    Statement;
}
```

### **Program**

```
class IfElseStatement
{
    public static void main(String... args)
    {
        int a=97;

        if (a>b)
        {
            System.out.println("print a");
        }
        else
        {
            System.out.println("print b");
        }
    }
}
```

### **IF-ELSE-IF STATEMENT**

#### **SYNTAX :**

```
    If(Boolean condition 1)
    {
        Statement;
    }
    Else if(condition 1 )
    {
        Statement;
    }
    Else {}
```

### **Program**

```
class Ifelseif
{
    public static void main(String[] args)
    {
        System.out.println("Program Start");
        int marks=25;
        if(marks>79 && marks<=100)
        {
            System.out.println("first class distinction");
        }
        else if (marks>=60 && marks<=79)
        {
            System.out.println("first class ");
        }
    }
}
```

```

    }
    else if (marks>=50 && marks<=59)
    {
        System.out.println("second class ");
    }
    else if (marks>=35 && marks <=49)
    {
        System.out.println("third class ");
    }
    else if (marks>100)
    {
        System.out.println("Invalid number");
    }
    else
    {
        System.out.println("You are fail");
    }
}
}

```

\*\* note : logical operator are used to combine multiple conditions.

And operator (&&)				Or operator (  )		
C1	C2	Result		C1	C2	Result
T	T	T		T	T	T
T	F	F		T	F	T
F	T	F		F	T	T
F	F	F		F	F	F

## SWITCH CASE

- It is used whenever we have to compare the given value only equals (==) conditions.
- Switch case statements provides more readability for the program.
- It is not possible to compare the conditions other than equals conditions.
- Syntax :
 

```

Switch(choice)
{
    Case 1: Statement;
        Break;
    Case 2: Statement;
        Break;
    Default : Statement;
}

```
- Note : Break statement stop the execution of the given block at a given line of code. Writing break statement after default case is not mandatory in switch case statement.

## Program

```
class Switch
{
    public static void main(String[] args)
    {
        System.out.println("Program Start");
        char a='C';
        switch(a)
        {
            case 'A': System.out.println("alphabet A ");
            break;
            case 'C' : System.out.println("aphabet C ");
            break;
            default : System.out.println("Invalid ");
        }
    }
}
```

## LOOPING STATEMENTS

- They are used to perform repetitive task in the given program with lesser lines of code.
- Different types of loops are :-
  - (a) For loop
  - (b) While loop
  - (c) Do-while loop
  - (d) For each loop /advanced/enhanced loop

### FOR LOOP (ITERATION) :

- One complete execution cycle of a loop is called as iteration.
- The number of iteration of a loop depends on the condition of the loop.
- Syntax :

```
For(initialization; condition; counter)
{
    Statement;
}
```
- For loop is a type of loop which is used whenever the logical start and logical end is well defined.

**Basic Program :**

```

Class Basic
{
    Public static void main (String ar[])
    {
        (1)      (2)      (4)
        For(int count=1; count<=3; count++)
        {
            (3)
            System.out.println("Hello World");
        }
    }
}

```

**Op – Hello World**  
**Hello World**  
**Hello World**

Iteration 1<sup>st</sup> – 1, 2, 3, 4

Iteration 2<sup>nd</sup> – 2,3,4,

Iteration 3<sup>rd</sup> – 2,3,4

**Tracing :**

Count=1	
Count<=3, 1<=3, true print hello world	
Count=2	
Count<=3, 2<=3, true print hello world	
Count=3	
Count<=3, 3<=3, true print hello world	
Count=4	
Count<=3, 4<=3, false terminate	

**Prgm ; wap to print 1 to 5**

```

Class Pgm2
{
    Public static void main (String ar[])
    {
        For(int i=1; i<=5;i++)
        {
            System.out.println(i);
        }
    }
}

```

**Tracing**

I=1
1<=5 true print i=1

I=2 2<=5 return true print i=2
I=3 3<=5 return true print i=3
I=4 4<=5 return true print i=4
I=5 5<=5 return true print i=5
I=6 6<=5 return false for loop terminate

Pgm : wap to print the numbers between 33 to 74 in descending order.

Class A

```
{
    Public static void main (String ar[])
    {
        For (int i=74; i>=33;i--)
        {
            System.out.println(i);
        }}}
```

### Tracing

I=74 74>=33 true print i=74
I=73 73>=33 true print i=73
I=72 72>=33 return true print i=72
I=32 32>=33 return false and terminate

**Prgm : wap to display only even no between 1 to 100?**

```
For(int i=1; i<=10;i++)
{
    If(i%2==0)
    {
        S.O.P. (i);
    }
}
```

### Tracing

I=1 I<=10, 1<=10 return true then if block execute and check the condition If(1%2==0) return false so it terminate and return back to for block
I=2



I<=10, 2<=10 return true then if block execute and check the condition If(2%2==0) return true and print 2 and condition goes to again for loop block
I=3 I<=10, 3<=10 return true then if block execute and check the condition If(3%2==0) return false so it terminate and return back to for block
I=4 I<=10, 4<=10 return true then if block execute and check the condition If(4%2==0) return true and print 2 and condition goes to again for loop block -----
I=11 I<=10, 11<=10 return false so the for block is terminate and program is end.

Pgm : wap to display number between 1 to 100 which are divisible by both 2 and 5.

```

For( int i=1; i<=100;i++)
{
    If(i%2==0 && i%5==0)
    {
        System.out.println(i);
    }
}

```

### **NESTED FOR LOOPS**

- A for loop written within the body of another for loop is called as nested for loop.
- Syntax :

```

For(initialization; condition; counter)
{
    For(initialization; condition; counter)
    {
        Statement ;
    }
    Statement
}
}

```

**Pgrm :**

```

Class A
{
    Public static void main (String ar[])
    {
        For(int i=1;i<=3;i++)
        {
            System.out.println("outer loop");
            For(int j=1; j<=3;j++)
            {
                System.out.println("inner loop");} } }

```

## Tracing

I=1  
I<=3, 1<=3 return true print outer loop and goes to inner loop  
J=1  
J<=3, 1<=3 return true print inner loop  
J=2  
J<=3, 2<=3 return true print inner loop  
J=3  
J<=3, 3<=3 return true print inner loop  
J=4  
J<=3; 4<=3 return false inner loop terminate and outer loop execute again

I=2  
I<=3, 2<=3 return true print outer loop and goes to inner loop  
J=1  
J<=3, 1<=3 return true print inner loop  
J=2  
J<=3, 2<=3 return true print inner loop  
J=3  
J<=3, 3<=3 return true print inner loop  
J=4  
J<=3; 4<=3 return false inner loop terminate and outer loop execute again

I=3  
I<=3, 3<=3 return true print outer loop and goes to inner loop  
J=1  
J<=3, 1<=3 return true print inner loop  
J=2  
J<=3, 2<=3 return true print inner loop  
J=3  
J<=3, 3<=3 return true print inner loop  
J=4  
J<=3; 4<=3 return false inner loop terminate and outer loop execute again

I=4  
I<=3, 4<=3 return false and terminate outer loop

### Prgm : wap to print pattern

\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*

```
For(int i=1;i<=3;i++)  
{  
    For(int j=1;j<=3;j++)  
    {  
        System.out.print(" * ");  
    }  
    System.out.println(); // moves to next line  
}
```

#### Tracing

I=1  
I<=3; 1<=3, return true , inner loop is executed  
J=1  
J<=3;1<=3 return true print \*  
J=2  
J<=3;2<=3 return true print \*  
J=3  
J<=3;3<=3 return true print \*  
J=4  
J<=3;4<=3 return false and inner loop is terminated, condition again goes to outer for loop.

I=2  
I<=3; 2<=3, return true , inner loop is executed  
J=1  
J<=3;1<=3 return true print \*  
J=2  
J<=3;2<=3 return true print \*  
J=3  
J<=3;3<=3 return true print \*  
J=4  
J<=3;4<=3 return false and inner loop is terminated, condition again goes to outer for loop.

I=3  
I<=3; 3<=3, return true , inner loop is executed  
J=1  
J<=3;1<=3 return true print \*  
J=2

J<=3;2<=3 return true print * J=3 J<=3;3<=3 return true print * J=4 J<=3;4<=3 return false and inner loop is terminated, condition again goes to outer for loop.
I=4 I<=3; 4<=3, return false, outer loop is terminated.

## WHILE LOOP

Syntax :

```
While(Boolean_condition)
{
    Statement1;
}
```

- It is a type of looping statement. It is used whenever the logical start and logical end is not well defined.
- Pgm

```
Int count=1;
While(count<=5)
{
    System.out.println(count);
    Count;
}
```

## DO-WHILE LOOP

Syntax :

```
Do
{
    Statement;
}
While(Boolean condition);
```

Prgm :

```
Int count=1;
Do
{
    System.out.println(count);
    Count++;
}
While(count<=5);
```

## CHAPTER 4 : METHODS IN JAVA

- A method is named block of codes which perform a specific task and it may or may not return a value.
- **Syntax**  
Access\_specifier access\_modifier return\_type name(arg list)  
{  
    Statement;  
    Return;  
}
- The return type of the method can be any primitive datatype or class type or void.
- The return statement of the method is used to return the control from method back to calling method
- Calling method : A method which is calling another method is known as calling method.
- A method which is called by another method is known as called method.
- Every method should be written within only the scope or body of a class.
- No methods will be executed without calling the method.
- If a method is expecting the arguments then you should call the same method by passing the required values matching the datatype.

**Program : Wap to print circumference of circle, perimeter of square, volume of a cube, total surface area of cylinder, and simple interest using method.**

class Assignment

```
{  
    public static void Circum(double rad)  
    {  
        Double cir=2*3.142*rad;  
        System.out.println("circumeferance of a circle is "+cir);  
        return;  
    }  
    public static void Square(double side)  
    {  
        double perimeter=4*side;  
        System.out.println("perimeter of a square " + perimeter);  
        return;  
    }  
    public static void Cylinder(double rad, double h)  
    {
```

```

        double area=2*3.142*rad *(rad+h);
        System.out.println("Total surface area of a cylinder is "+area);
        return;
    }
    public static void Cube(double side)
    {
        double volume=side*side*side;
        System.out.println("Volume of a cube is = " +volume);
        return;
    }
    public static void SimpleInterest(double p, double r, double t)
    {
        double si=(p*r*t)/100;
        System.out.println("Simple Interest is " + si);
        return;
    }
    public static void main(String[] args)
    {
        Circum(5);
        Square(5);
        Cylinder(5,7);
        Cube(5);
        SimpleInterest(1000, 2, 2);
    }
}

```

- If you declare the return type of the methods as **void** then the **method returns only the control** from called method to back o calling method.
- If the return type of the method is void then the written statement will be written by the compiler even if the programmer miss it.

## **METHOD WITH RETURN VALUE**

- If a method is returning a value then the return type of the method should be a primitive datatype or a class type matching the returned value.
- If a method is returning a value then you should store the returned value within a variables matching the return type of the method.
- From a method we can return only one single value.

### **Pgm : WAP TO PRINT AVG OF A NUMBER USING METHOD**

```
class Methods
{
    public static double calcArea(int a, int b, int c)
    {
        double avg;
        avg=(a+b+c)/3;
        return avg;
    }
    public static void main(String[] args)
    {
        double res=calcArea(10,20,30);
        System.out.println("res = " + res*2);
        double res1=calcArea(20,10,30);
        System.out.println("res1 = " +res1*0.3);
    }
}
```

**Op – res = 60.0**

**Res1= 9.0**

### **METHODS WITH NO ARGUMENTS:**

```
Class pgm3
{
    Public static void displayMsg()
    {
        S.O.P. ("Hello");
        S.O.P. ("good morning");
    }
    Public static void main (String ar[])
    {
        displayMsg();
        displayMsg();
    }
}
```

## CHAPTER 5 : ARRAYS (BASICS)

- An array is homogeneous group of elements which has fixed size and index.
- Using array we can manage the data easily to perform different operations.

### (a) Array Declaration

#### Syntax :

Datatype [] array\_name;

Eg : int [] marks;

Or Datatype array\_name[];

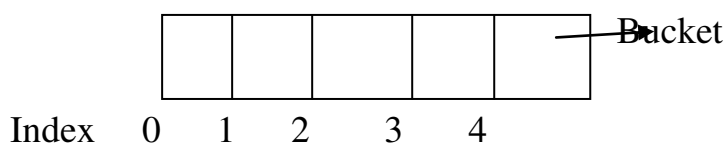
Int marks[];

### (b) Array Creation

#### Syntax :

Array\_name = new datatype [size];

Eg marks = new int[5];



### To Declare and Create Array in Single line

#### Syntax :

Datatype arrayname [] = new datatype [size];

Eg int marks[]=new int[5];

- After array creation every bucket of the array will be initialize with default values according to datatype of the array. For eg. If the datatype of an array is int then all the bucket will be filled with 0. Or if the datatype of an array is Boolean then all the bucket will be filled with false.

### (c) Initialize Array

#### Syntax :

Arrayname[index]=value;

For eg: marks[0]=20;



### **Pgm**

```
class Array
{
    public static void main(String[] args)
    {
        int marks[]=new int[5];
        marks[0]=67;
        marks[1]=20;
        marks[2]=62;
        marks[3]=85;
        marks[4]=55;
        for(int index=0; index<5; index++)
        {
            System.out.println(marks[index]);
        }
    }
}
```

**Op – 67    20    62    85    55**

### **Length (variables of array):**

- It contains count of number of bucket present in the given array.

### **Notes :**

- If you know the size of the array and the data to be stored in the array in advance then you can declare and initialize the array in same line.

### **- Syntax :**

**Datatype[] arrayname = {val1, val2....};**

### **pgm**

```
class Array1
{
    public static void main(String[] args)
    {
        String [] days={"mon", "tue", "wed", "thurs", "fri",
        "sat", "sun"};
        for (int index=0;index<days.length;index++ )
        {
            System.out.println(days[index]);
        }
    }
}
```

## ArrayIndexOutOfBoundsException

- Whenever we try to add the elements to array exceeding the size of array then JVM throws ArrayIndexOutOfBoundsException.
- If you try to retrieve the elements from the array exceeding the size of the last index of the array then JVM throws ArrayIndexOutOfBoundsException.

## Pgm

**//wap of array to find first middle and last element of an array**

```
class Array2
{
    public static void printNum(int[] a1)
    {
        int first=0;
        int last=a1.length-1;
        int mid=last/2;

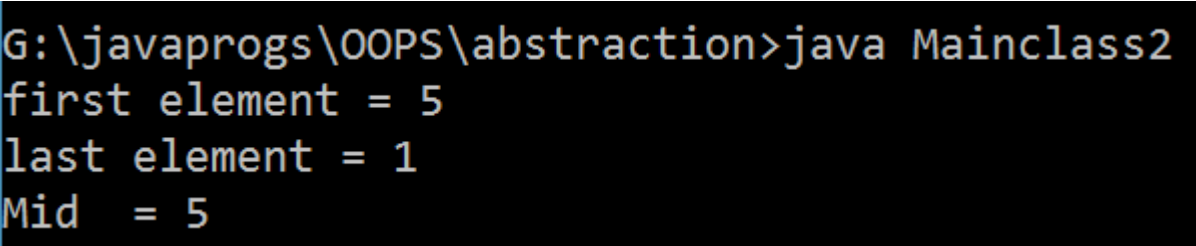
        System.out.println( "first = " +a1[first]);
        System.out.println( "last = " +a1[last]);
        System.out.println(      "mid = " +a1[mid]);
    }
    public static void main (String ar[])
    {
        int num[]={5,1,2,3,5};
        printNum(num);
    }
}
```

```
G:\javaprogs\OOPS\abstraction>java Mainclass2
first = 5
last = 5
mid = 2
```

(ii)

**//wap of array to find first middle and last element of an array**

```
class Array3
{
    public static void printNum(int[] a1)
    {
        int first=0;
        int last=a1.length-1;
        int mid=last/2;
        System.out.println( "first element = " + a1[first]);
        System.out.println( "last element = " + a1[last]);
        System.out.println("Mid = " a1[mid]);
    }
    public static void main (String ar[])
    {
        int num[]={5,1};
        printNum(num);
    }
}
```



```
G:\javaprogs\OOPS\abstraction>java Mainclass2
first element = 5
last element = 1
Mid = 5
```

**Notes :**

**First Index = 0;**

**Last Index = length -1;**

**Middle Index = last index /2;**

## CHAPTER 6 : STRING IN JAVA (BASICS)

- A string is group of characters which is written within the double quotes.
- Internally the string will be created with the help of character array.

### Methods of String

- Length()** – it returns count of number of character present in the given string.  
Eg. S1.length();
- Equals()** – this method compares every character present in the given two strings and returns true. If they are same else it returns false.  
String s1= “jspider”;  
String s2= “jspider”;  
String s3 = “JSPIDER”;  
S1.equals(s2)- return true;  
S1.equals(s3)- return false ;
- equalsIgnoreCase()** – this method compares every character in the given two strings by **ignoring their case**. And return true if they are same, else it returns false.  
String s1= “jspider”;  
String s2 = “JSPIDER”;  
S1.equalsIgnoreCase(s2)- return true;
- charAt () - // charAt(index)** – This method returns the character present at the given index.  
S1.charAt(4) – return d ;
- toCharArray()** – This function or this method returns the array representation of the given string.  
Char arr[] = s1.toCharArray();

### Program

```
class StringBasic
{
    public static void main(String[] args)
    {
        String s1="jspider";
        String s2="jspider";
        String s3="JSPIDER";

        int len = s1.length();
        System.out.println("Length of String = " +len);
    }
}
```

```
boolean res1=s1.equals(s2);
System.out.println("Equals or not = " +res1);

boolean res2=s1.equals(s3);
System.out.println("Equals or not = " +res2);

boolean res3=s1.equalsIgnoreCase(s3);
System.out.println("Equals or not = " +res1);

char c1=s1.charAt(4);
System.out.println("Character at index = " +c1);

char [] ar1=s1.toCharArray();
for(int index=0; index<ar1.length;index++)
{
    System.out.print(ar1[index]);
}
}}
```

```
G:\javaprogs\OOPS\abstraction>java Mainclass2
Length of String = 7
Equals or not = true
Equals or not = false
Equals or not = true
Character at index = d
jspider
```

# OBJECT ORIENTED PROGRAMMING SYSTEM

## CHAPTER 1 : CLASS & OBJECTS

**OBJECTS :** Any entity which has states and behaviour is called as objects. For eg. Pen, account, building.

<b>Pen</b>		<b>Account</b>		<b>Student</b>	
<b>State</b>	<b>Behaviour</b>	<b>State</b>	<b>Behaviour</b>	<b>State</b>	<b>Behaviour</b>
Color	Writing	a/c no.	Debit	Name	Studying
Brand	Drawing	Type of ac	Credit	Id	Exam
Shape	Pointing	Bank name	Payment	Gender	Reading
Price	.....	Balance .... etc	.....	Age	writing

### **CLASS : -**

- A class is blueprint of an objects.
- A class contains or defines states and behaviours of an objects.
- The states of the class are called as data members and the behaviours of the objects are called as function members.
- The data members of the class are represented by variables and the functions members of the class are represented by methods.

### **JAVA NAMING CONVENTION :**

#### **Class :**

- Any entity in java which starts with upper case declared with the keyword class is called as java class.
- Class names should be nouns, in mixed case with first letter of each internal word capitalize.  
Eg. Account, AccountName;

#### **Interface :**

- Any entity in java which starts with upper case and declared with the keyword interface is called java interface.
- Interface name should be capitalize like class name.
- Eg. : interface Storing, interface RunnableInterface

#### **Methods :**

- It defines the action which is performed on data members of the class.
- Methods should be in mixed case with the first letter lower case, with the first letter of each internal word capitalize.
- Eg : run(), getData()

**Variables :**

- Variables should be declared in mixed case with a lowercase first letter. Internal words starts with capital letters.
- Eg: i, c, myWidth;

**Constants :**

- The constants should be all uppercase with words separated by underscore (“\_”).
- Eg. MIN\_WIDTH = 5;

The members of class be classified into two types :

- (a) Static Members      (b) Non-Static members

**Static members**

- Any members of the class which is declared using static keyword is called as static members.

**Static Members present in Same Class:**

- If a static method is trying to access the static members present in the same class then it can refer to them directly with the name of static member.

**Program**

```
class Demo
{
    static int v1=100;
    public static void test()
    {
        System.out.println("this is test() of demo class");
    }
    public static void main (String ar[])
    {
        System.out.println("V1 is = " + v1);
        test();
    }
}
```

```
G:\javaprogs\OOPS\Static>javac Demo.java
```

```
G:\javaprogs\OOPS\Static>java Demo
```

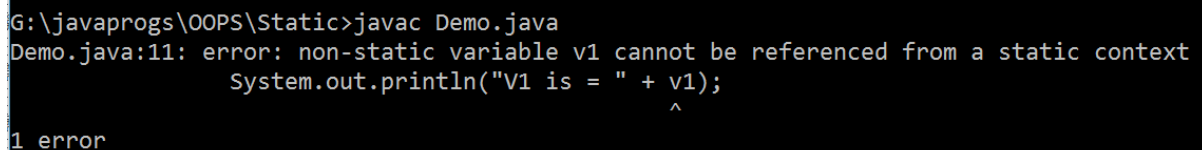
```
V1 is = 100
```

```
this is test() of demo class
```

## Program 2

class Demo

```
{
    int v1=100;
    public static void test()
    {
        System.out.println("this is test() of demo class");
    }
    public static void main (String ar[])
    {
        System.out.println("V1 is = " + v1);
        test();
    }}
}
```



```
G:\javaprogs\OOPS\Static>javac Demo.java
Demo.java:11: error: non-static variable v1 cannot be referenced from a static context
        System.out.println("V1 is = " + v1);
                                   ^
1 error
```

**Note : - A non static variable can not be referenced by a static context.**

Note : \*\*

- Within one java program we can write any number of class.
- If a program contains multiple classes then the class which contains main method should be used as filename.

### Static Members Present in Different Class :

- We can access static members of different class using the classname with dot(.) operator followed by member name.
- Syntax :

```
className.memberName;
className.memberFunction();
```

Note : we can not use static member in different class directly. If we use it throw error.

## Program

**// static member used by static method use in different class directly.**

class Demo

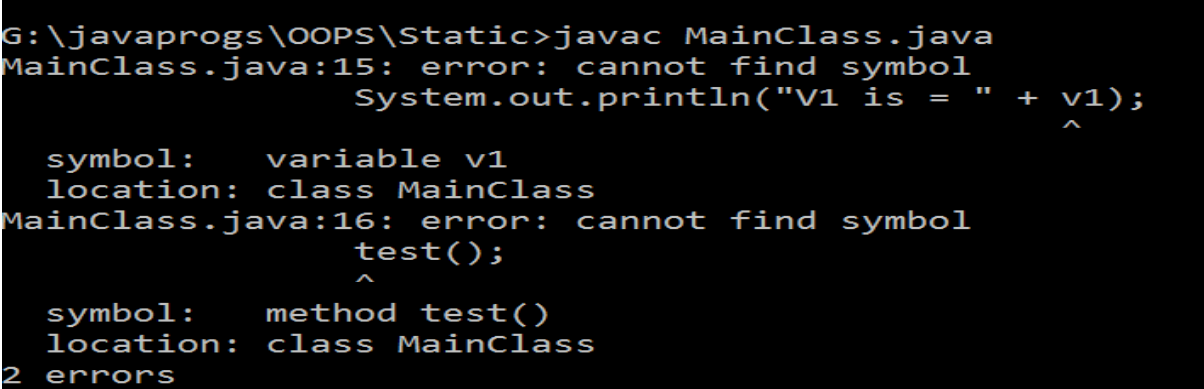
```
{
    static int v1=100;
    public static void test()
```



```

        {
            System.out.println("this is test() of demo class");
        }
    }
}
class MainClass
{
    public static void main (String ar[])
    {
        System.out.println("V1 is = " + v1);
        test();
    }
}

```



```

G:\javaprogs\OOPS\Static>javac MainClass.java
MainClass.java:15: error: cannot find symbol
    System.out.println("V1 is = " + v1);
                                   ^
symbol:   variable v1
location: class MainClass
MainClass.java:16: error: cannot find symbol
    test();
    ^
symbol:   method test()
location: class MainClass
2 errors

```

### Program :

**// static member used by static method use in different class by classname.member**

```

class Demo
{
    static int v1=100;
    public static void test()
    {
        System.out.println("this is test() of demo class");
    }
}
class MainClass2
{
    public static void main (String ar[])
    {
        System.out.println("V1 is = " + Demo.v1);
        Demo.test();
    }
}

```

## NON-STATIC MEMBERS :

- Any member of the class which is declared without using static keyword is called as non-static members.
- We can access non-static members of a class only by creating the object for the class.

### Object creation

#### Syntax :

**New className()**

Eg.

New Sample()

Here New is a keyword which creates a new object and Sample() is a **constructor call** which copy all the non-static member to object.

- A non static method can access non-static data members or non static function members present in the same class without creating any object.

### Program : calling non-static member and method from main method in a single class by not creating any object

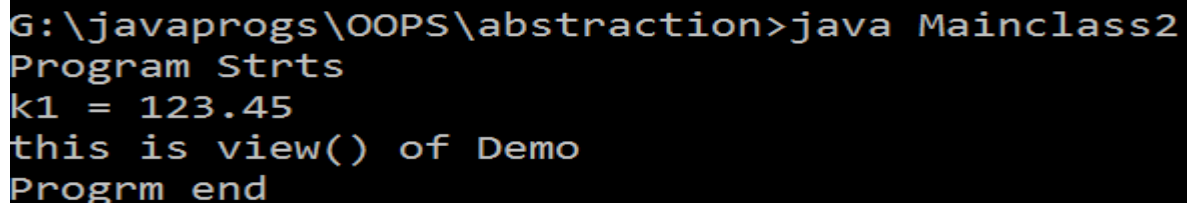
```
class Abc
{
    int z1=123;
    public void view()
    {
        System.out.println("this is view() of Abc");
        System.out.println("value of a = " + z1);
    }
    public static void main(String[] args)
    {
        System.out.println("Z1 = " + new Abc().z1);
        new Abc().view();
    }
}
```

```
G:\javaprogs\OOPS\abstraction>java Mainclass2
Z1 = 123
this is view() of Abc
value of a = 123
```

**Program :**

**// use of non static members and function in other class**

```
class Mainclass
{
    double k1=123.45;
    public void count()
    {
        System.out.println("this is view() of Demo");
    }
}
class Mainclass2
{
    public static void main(String[] args)
    {
        System.out.println("Program Strts ");
        System.out.println("k1 = " + new Mainclass().k1);
        new Mainclass().count();
        System.out.println("Progrm end " );
    }
}
```



A screenshot of a terminal window showing the execution of a Java program. The command 'G:\javaprogs\OOPS\abstraction>java Mainclass2' is entered. The output consists of five lines: 'Program Strts', 'k1 = 123.45', 'this is view() of Demo', and 'Progrm end'.

**Program // static member used by static method use in same class**

```
class Demo
{
    static int v1=100;
    public static void test()
    {
        System.out.println("this is test() of demo class");
    }
    public static void main (String ar[])
    {
        System.out.println("V1 is = " + v1);
        test();
    }
}
```

```
G:\javaprogs\OOPS\abstraction>java Mainclass2
V1 is = 100
this is test() of demo class
```

**Program // static member used by static method use in different class.**

```
class Demo
{
    static int v1=100;
    public static void test()
    {
        System.out.println("this is test() of demo class");
    }
}
class MainClass
{
    public static void main (String ar[])
    {
        System.out.println("V1 is = " + Demo.v1);
        Demo.test();
    }
}
```

```
G:\javaprogs\OOPS\abstraction>java Mainclass2
V1 is = 100
this is test() of demo class
```

## CHAPTER 2 : REFERENCE VARIABLES

Demo d1 = new Demo ();

- It is a type of variables which is used to store the address of the objects.

```
class RefVariable
```

```
{
```

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

```
    {
```

```
        RefVariable r = new RefVariable();
```

```
        System.out.println( " Address of Variable r = " +r);
```

```
    }}
```

**// output Address of variable r = RefVariable@6073f712**

- Within a reference variables we can not store any primitive data values.

```
class RefVariable
```

```
{
```

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

```
    {
```

```
        int r = new RefVariable(); // it throw error here
```

```
        System.out.println( " Address of Variable r = " +r);
```

```
    }
```

```
}
```

- Multiple reference variable can point to same objects.

```
Demo r1 = new Demo();
```

```
Demo r2 = r1;
```

```
class RefVariable
```

```
{
```

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

```
    {
```

```
        RefVariable r = new RefVariable();
```

```
        RefVariable r2=r;
```

```
        System.out.println( " Address of Variable r = " +r);
```

```
        System.out.println( " Address of Variable r1 = " +r2);
```

```
    }
```

```
}
```

**// output Address of variable r = RefVariable@6073f712**

**// output Address of variable r2 = RefVariable@6073f712**

- If multiple reference variables are pointing to same object then changes done through one reference variables will impact other reference variables.

```
class RefVariable
{
    int r = 20;
    public static void main(String[] args)
    {
        RefVariable r1 = new RefVariable();

        System.out.println( " value of r1 = " +r1.r);
        r1.r=30; //reinitialize the value of r by using the object

        System.out.println( " Address of Variable r = " +r1.r);

        RefVariable r2 = r1;
        System.out.println("Value of r2 = " +r2.r);
    }
}
```

- Static members are called as class members because they can be accessed using the classname.
- Non static members are called as instance members because they can be accessed only by creating the object or instance.
- Static members of the class will have only one copy in the memory.

```
Static int v1 = 50;
/S.O.P (Sample.v1);
Sample.V1=50;
```

- Non-static members will have multiple copies in the memory depending on number of objects created.

```
Int v1= 100;
S.O.P ( new Simple().V1);
S.O.P (new Simple().V2);
```

## Program

```
class Account
{
    int actno=12345;
    String name = "Dinga";
```

```

String branch = "basvangudi";
double balance = 5000;
String type = "Savings";

static String bankName = "ICICI";

    public void deposit(int amt)
    {
        balance = balance+amt;
    }
public void withdraw (int amt)
{
    balance = balance-amt;
}
public void checkBalance()
{
    System.out.println(" Avaialable balance "+balance);
}
public void showAccount()
{
    System.out.println("act no = "+actno);
    System.out.println("name = "+name);
    System.out.println("brnch = "+branch);
    System.out.println("balance = "+balance);
    System.out.println("acount type = "+type);
    System.out.println("acount type = "+bankName);
}
}
class MainAccount
{
    public static void main (String ar[])
    {
        Account a1=new Account();
        a1.checkBalance();
        System.out.println();
        a1.deposit(5000);
        a1.checkBalance();
    }
}

```

```

        System.out.println();
        a1.withdraw (2000);
        System.out.println();
        a1.checkBalance();

        a1.showAccount();
    }
}

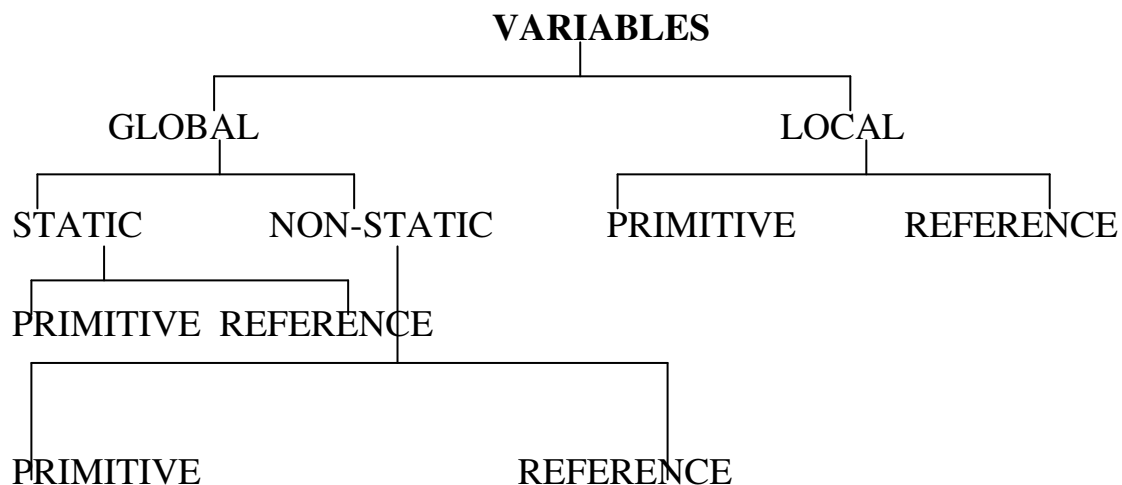
```

```

G:\javaprogs\extra>java StaticKeyword
Remaining Balance = 5000.0
Remaining Balance = 5500.0
Remaining Balance = 4500.0
Name is = Dinga
actno is = 123456
account type is = Saving
total balance = 4500.0
Bank name = ICICI

```

## GLOBAL AND LOCAL VARIABLES



## GLOBAL VARIABLES

- A variables which is declared within the scope of the class is called as global variables.



- Global variables can be accessed by all the methods present in the present in the same class.

## **LOCAL VARIABLES**

- Any variables which is declare within the method declaration or method definition are called as local variables.
- Local variables can be accessed only within the methods in which they are declared.
- Local variables can not be declared as static or non static.

### **Program : Declaration of local and global variables**

```
class Abc
{
    int z1=123; // global variables
    public static void view()
    {
        int y = 20; // local variables
        System.out.println("Value of y = " +y)
    }
    public static void main(String[] args)
    {
        System.out.println("Z1 = " + new Abc().z1);
        new Abc().view();
    }
}
```

### **Program : local variable can not used outside the method.**

```
class Abc
{
    int z1=123;
    public void view()
    {
        int a = 20;
        System.out.println("this is view() of Abc");
        System.out.println("value of a = " + z1);
        System.out.println("value of a = " + a);
    }

    public static void main(String[] args)
    {
        System.out.println("Z1 = " + new Abc().z1);
    }
}
```

```

        new Abc().view();
        System.out.println("value of a = " + a); // throws an error
    }
}

```

- If variables and local variables have same names then the compiler always give the preference to local variables.

```

class Abc
{
    static int z1=123;

    public static void main(String[] args)
    {
        int z1=400;
        System.out.println("Z1 = " + z1);
    }
}

```

**// output z1 = 400**

- If we want to use both variables together then use classname.variablesname

```

class Abc
{
    static int z1=123;

    public static void main(String[] args)
    {
        int z1=400;
        System.out.println("Z1 = " + z1);
        System.out.println("Z1 = " + Abc.z1);
    }
}

```

**// output z1 = 400**

**// output z1 = 123;**

### **Important notes :**

- Using the object of class we can access both static and non static members of the class.

- It is strictly not recommended to access static members of the class using objects.
- Global variables (both static and non-static) will be initialized by the compiler with the default values depending on the datatype.
- Local variables should be initialized by the programmer explicitly.

### **Program**

```
class Global2
{
    static double x1 ;

    public static void main(String[] args) {
        int z1;
        System.out.println("Value of x1 = "+x1);
        System.out.println("Value of z1 = " + z1);
        /* here z1 is not initialized so it show an error.
        but for x1 it does not show any error.
        it take default value of x1 according to theri data type.*/
    }
}
```

### **DECLARING CONSTANTS :**

- Final keyword is used to declare constants in java.
- If you declare any variables with final keyword then it can not be re-initialized.
- If you declare any class with final keyword then the class can not be inherited.
- If you declare any method with final keyword then it can not be overridden.

### **Program**

```
class Constants
{
    final double PI=3.142;

    public void area1 (int a)
    {
        double ar= PI*a*a;
        System.out.println("Area of 1st Circle" + ar);
    }
}
```

```

    public void area2 (int a)
    {
        double ar= PI*a*a;
        System.out.println("Area of 1st Circle" + ar);
    }
    public static void main(String[] args)
    {
        Constants c=new Constants();
        c.area1(5);
        c.area2(10);
    }
}

```

- Note : we can not re-initialize any constant value. If we do then it throw an error

Program

class Constants

```

{
    final double PI=3.142;

    public void area1 (int a)
    {
        PI=4525; here it throw error as cannot assign a value to final variables.
        double ar= PI*a*a;
        System.out.println("Area of 1st Circle" + ar);
    }
    public static void main(String[] args)
    {

        Constants c=new Constants();
        c.area1(5);

    }
}

```