

# CORE JAVA

## MODULE:-1

- + INTRODUCTION TO JAVA
- + BASIC PROGRAM STRUCTURE
- + COMMENTS
- + ARCHITECTURE OF JAVA
- + DATA TYPES
- + VARIABLES
- + OPERATORS AND TYPES
- + KEYWORDS & LIST OF KEYWORDS
- + IDENTIFIERS & RULES OF IDENTIFIERS
- + CONTROL FLOW STATEMENTS
- + LOOPING STATEMENTS
- + PATTERN PROGRAMMING
- + MODULE-1 HOME WORK

### INTRODUCTION TO JAVA

Organization Started → SUN MICRO SYSTEM

Started By → JAMES GOSWALIN

Team → GREEN TEAM

Project → GREEN PROJECT

Started Year → 1990

Reason to start → A Project to Develop an Electronic S/W

Completed → Early 1996

Current Vendor → ORACLE (2010-11)

First Version → JDK 1.0 (MARCH-1996)

Last Version → JDK 18 (MARCH-2022)

Initial Name → OAK

Final Name → JAVA

Software To Develop → JDK (1.8 to 18v)

**BASIS** → C++

**PRINCPILE** → WRITE ONCE RUN ANY WHERE

**OPERATING SYSETEM** → INDEPENDENT

- ⊕ JAVA is one of the most widely used programming language, due to its various characteristics.

### **SOME OF FEATURES / CHARACTERISTICS OF JAVA:**

- ⊕ Simple to learn
  - ⊕ Platform independent:- java program does not depend on any operating system. i.e program written under one operating system can be run on any other operating system.
  - ⊕ Object Oriented Programming Langauge:- It is one of the programming strategy which is required to fulfill every project requirement.
- It has 4 concept:-*
- ✓ IN-HERITENCE
  - ✓ ABSTRACTION
  - ✓ ENCAPSULATION
  - ✓ POLYMORPHISM
- ⊕ Portable:- Along with JAVA, we use any other programming language as a supporting language
  - ⊕ Multi-Threaded:- We can run 2 different part of our same program parallelly.
  - ⊕ Robust:- Most of the important things will happen automatically (**Robotic Nature**).
  - ⊕ Over -Loaded:- One thing doing multi task.

### **USES OF JAVA:**

- JAVA is widely used in many applications.

1. Stand Alone Application:-

No dependency on Internet Server.

*Ex: Calender, Calculator, Notepad.*

2. Client – Server Application:-

Which requires server access.

*Ex: Banking Application, Gmail, What's up.....etc*

3. Web Based Application:-

- without installing the app
- direct URL type and use

*Ex: YouTube, Face book*

**4. Mobile Application:-**

*Ex: What's up, Face book, Instagram.....etc.*

**5. SAP:- (SOFTWARE APPLICATION PRODUCT)**

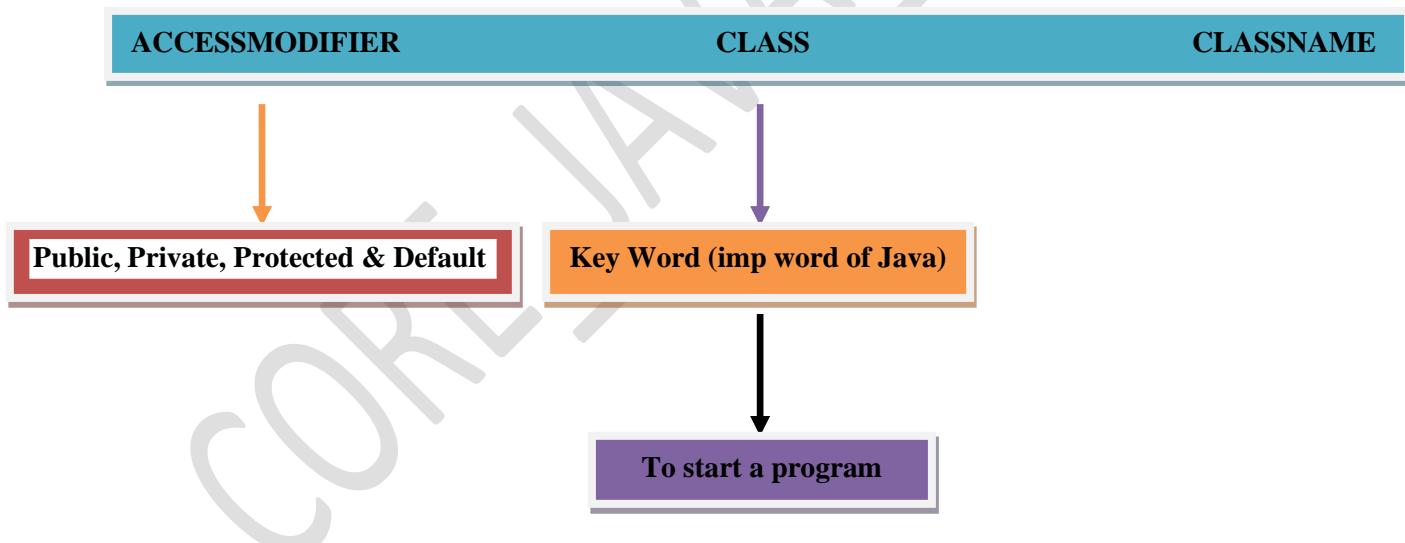
*GERMAN BASED APPLICATION*

**BASIC STRUCTURE OF THE PROGRAM:**

- ✚ A set of instruction given to a machinery following **Syntax** of any programming language is called as **program**.
- ✚ A group of program will make one application.
- ✚ An application is basically a communication b/w human & machine.
- ✚ Software is a type of applications.
- ✚ Every Java program consists of following 5 steps:-
  1. PACKAGE DECLARATION
  2. IMPORT STATEMENT
  3. CLASS DECLARATION
  4. MAIN METHOD
  5. PRINTING DECLARATION

**CLASS DECLARATION:-** Class declaration is the starting point of a Java program.

**Syntax:-**



**Ex:** Public class sample

Public class Demo

**Access Modifier:-** It provides accessing permission to different files in our program. We have 4 access modifiers

1. PUBLIC
2. PRIVATE
3. PROTECTED
4. DEFAULT

- + **Public:-** Public is an access modifier which indicate openly accessible.
- + **Class:-** Class is defined as key word (important words of Java) which used start program.

- + **ClassName:-** Class Name is a identifier, it can only be combination of (A-Z i.e a-z)

**Ex:** Public class sample

    Public class Demo

- **NOTE POINTS:-**

{ → Start of Logic  
} → End of Logic  
; → End of Line

**Public Class Sample**

{

**Logic**

}

- + In Java we will never write a logic directly inside a class in order develop any logic we will use main method.

### **MAIN METHOD:**

#### **Syntax:**

**Public     Static     Void Main (String arngs [ ] )**

- + Main method is defined as heart of the java program without that program will not be executed.
- + **Public:** Public is access modifier which indicate main method is freely accessible.
- + **Static:** Static is key word which indicate no need of object creation.
- + **Void:** Void is the return type which indicate main is not returning any specific value.
- + **Main:** Main is a identifier (Method Name)
- + **String arngs[ ]:** It is said as string arngs of arrays which is called command line arguments.(This related to array programming)

**Ex:**

### **Public Class Sample**

```
{  
    Public Static Void Main (String arngs [ ] )  
    {  
        System.Out.Println("HELLO");  
  
    }  
}
```

### **PRINTING STATEMENT:**

#### **Syntax:**

```
System.Out.println("Hello world")
```

- ⊕ If we want to print any message on the output screen then will use this statement.
- ⊕ **System:** System is a pre-define class.
- ⊕ **Out:** Out is a pre-define Object.
- ⊕ **println:** println is a pre-define Method.
- ⊕ **Dot:** DOT is an operator which is used for connecting multiple things at one place.

// Pre define class

    Class System

    {

        Object. Out

            -----> **println( )**

}

### **NOTE:**

### **Public Class Sample**

```

{
    Public Static Void Main (String arngs [ ] )
    {
        System.out.println("Hello world");
    }
}

```

- In above program all the word starting with smaller case expect  
*String – S      System – S*

### **DIFFERENT SOFTWARE IN JAVA:-**

*JAVA Software is available in 3 different parts*

1. JME → Java Micro Edition → Embedded S/W.
2. J2SE/JSE → Java Standard Edition → Core Java.
3. J2EE → Java to Enterprise Edition → Advance Java.

### **ARCHITECTURE OF JAVA:**

- Every Java app should follow following 5 steps
  1. Select an Editor:- A place to develop a program  
**Ex:** Notepad, Notepad++, Edit plus (or) Eclipse.
  2. Develop a code.
  3. Save a code:- Always save program with classname. Java  
**Ex:** Sample Java
  4. Compilation:-  
**Why to compile?**

*To convert our program to system understandable format.*

**Who will compile?**

*Javac compliers will compile.*

**How to compile?**

*Go to command prompt and type command as javac classname.java*

**What happens during compilation?**

*Compiler will checks Syntax error if no Syntax errors are there program complies successfully.*

**What happen after compilation?**

*A class file (or) byte code file will be created.*

5. Execution:-

**Why to execute?**

**For getting Output.**

**How to execute?**

**Go to command prompt and type command as java classname.**

**Who will execute?**

**JVM-java virtual machine.**

**What happen during execution?**

**JVM will identified any logical mistake is there (or) not.**

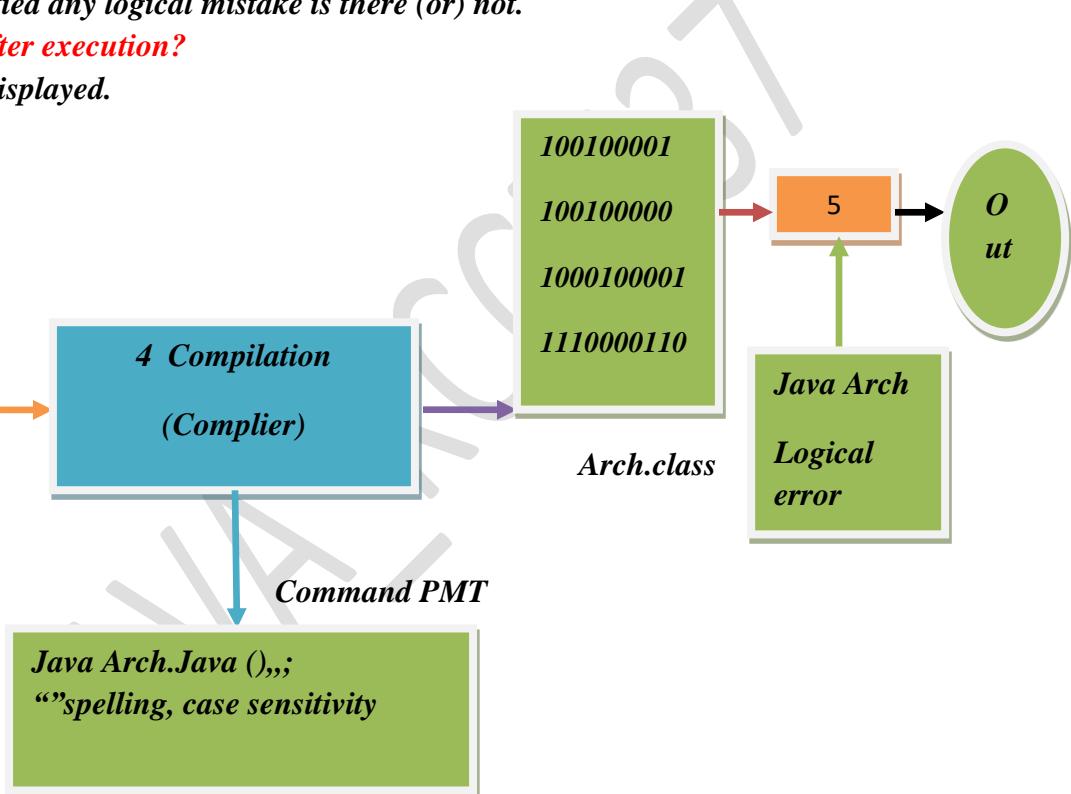
**What happen after execution?**

**Output will be displayed.**

### 1. Select the editor

Notepad

```
Public class Arch
{
    2
    Public Static Void Main
    (String arngs[ ])
    {
        System.Out.println("Arch of
        Java)
    }
}
```



**Arch.java**

### COMMENTS:

**Comments are used for two different purposes**

**1. For excluding for any line or line from execution.**

**2. To Make a program more readable.**

#### **1. Single line**

Syntax:

`// statement`

## **2. Multiple lines:**

*Syntax:*

```
/*
    statement1
    statement2
    statement3
    statement4
    statementN
*/

```

## **3. Documentation comment:-**

*We will discuss in eclipse.*

### **VARIATIONS IN PRINTING:**

- Print ("Message"): first print message then be in same line.
- Print ("Message"): first print message then go to new line.
- SOP ("\n"): In indicate break the line immediately.

*Ex:--1*

```
public class Info
{
    public static void main(String arngs[])
    {
        System.out.println("Mr John is data Scientist");
        System.out.print("in AT&T Labs");
        System.out.print("Since 2010");
        System.out.println("and he worked for research\n as well as development department");
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Info.java
```

```
C:\Users\user\Desktop\KCCM87>java Info
Mr John is data Scientist
in AT&T LabsSince 2010and he worked for research
as well as development department
```

## **DATA TYPES:-** Data is defined as same useful information

Ex: Name, Age, Email-id, Date of birth, Contact No, Aadhar no, Pan no...

- In above example data is of different types two represent different type of data in our program we use data types.
- Every data type is pre-defined word which represents of particular thing in our program.
- Data types are divided into two categories
  1. PREMETIVE DATA TYPE
  2. NON- PREMETIVE DATA TYPE

### **PREMETIVE DATA TYPE:**

- Also called as system define
- They are 8 in number
- All 8 of them have specific name, size and usage.

S.NO	NAME	MEMORY SIZE	USAGE
NUMERIC			
1	BYTE	1-BYTE	-128 TO +127
2	SHORT	2-BYTE	-32768 TO +32767
3	INT	4-BYTE	-2147483648 TO +2147483647
4	LONG	8-BYTE	>2147483647

### **FORMULA:**

$$\text{MIN: } -2^{(\text{NO:OF BITS}-1)} \longrightarrow -2^{(8-1)} \longrightarrow -128$$

$$\text{MAX: } +(2^{(\text{NO:OF BITS}-1)}) \longrightarrow +(2^{(8-1)})-1 \longrightarrow +127$$

### **(DECIMALS)**

5      FLOAT            4-BYTE → If we want upto 6 digits after decimal.

6      DOUBLE           8-BYTE → If we want more than 6 digits after decimal (15 dec)

### **(SINGLE-CHARACTER)**

7      CHAR            2-BYTE → a-z, A-Z, Single digits, Single Special Symbol.

### **(CONSTANTS)**

## **8 BOOLEAN 1-BYTE → TURE / FALSE.**

**NOTE:-** If primitive data type does not satisfy our requirement then we will go for non-primitive data type.

### **NON-PRIMITIVE DATA TYPE:-**

- When primitive does not fulfill our requirement then we have non primitive data type.
- These are also called as user define data type.
- They do not have specific memory size.

*Ex:- String, Arrays, Classes, Collection etc.*

*String → Group of characters, Alphanumeric, Decimals, Numeric etc...*

### **VARIABLES:-**

- Variables are used to store the data.
- Every Variable have 4 things.  
Name, Size, Type and Value
- For using Variables, we should follow two steps
- Variable name can be a-z

#### **1. VARIABLE DECLARATION:**

- Declaration means reserving a memory block.

##### **Syntax:-**

**DATA TYPE, VARIABLE NAME**

*Ex:- Byte a;*

*Short b;*

#### **2. VARIABLE DECLARATION-RULE WHEN WE INITIALISE:**

- Initialization means storing value into memory block.

##### **Syntax:-**

**VARIABLE NAME = VALUE;**

*Byte a; a = 120 → No rule for byte.*

*Short b; b = 3000 → No rule for short.*

*Int c; c = 5000 → No rule for int.*

*Long d; d = 67890004589L → Keep “L / l” after value.*

**Float e; e = 455.6789F** → Keep “F / f” after value.

**Double f; f = 89898.67676767** → No rule.

**Char g; g = “@”** → Keep value in single quotes.

**Boolean h; h = Ture** → No rule for Boolean.

**String k; k = “JAVA”** → Keep value in double quotes.

### 3. VARIABLE UTILISATION:-

Syntax:-

**SYSTEM.OUT.PRINTLN(VARIABLE NAME WITHOUT DOUBLE QUOTES)**

**Ex:** int age ----- → VAR DECLARATION

Age = 24 ----- → VAR INITILISATION

System.out.println(Age) ----- → VAR UTILISATION

**/\* WAP TO PRINT BOOK\_INFO ----- → NAME,AUTHOR,PAGES,COLOR,PRICE\*/**

```
public class Bookinfo
{
    public static void main(String arngs[])
    {
        //var dec-> datatype varname;
        String bname;
        String author;
        short pages;
        String color;
        short price;
        //var initi-> varname=value;
        bname = "Java";
        author="james goswlin";
```

```

    pages = 5000;
    color ="Black";
    price = 5000;
    //var utilisation
    System.out.println(bname);
    System.out.println(author);
    System.out.println(color);
    System.out.println(price);
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Bookinfo.java
C:\Users\user\Desktop\KCCM87>java Bookinfo
Java
james goswin
5000
Black
5000

```

/\* WAP TO PRINT STUDENTINFO USING DATATPYES & VARIABLES \*/

```

public class Studentinfo
{
    public static void main(String arngs[])
    {
        //var dec-> datatype varname;
        String name;
        int age;
        int sscyop;
        float sscper;
        String sklname;
    }
}

```

```
int interyop;
float interper;
String interclg;
int gradyop;
float gardper;
String gardclg;

//var initi-> varname=value;
name = "K.Sri Sai Vignesh";
age = 23;
sscyop = 2015;
sscper = 63.65f;
sklname = "Vigana High School";
interyop = 2017;
interper = 70;
interclg ="VJG";
gradyop =2022;
gardper =60;
gardclg ="VLITS";
//var utilisation
System.out.println("STUDENT DATA ACCORDING TO RECORDS : ");
System.out.println(name );
System.out.println(age);
System.out.println(sscyop);
System.out.println(sscper);
System.out.println(sklname);
```

```

        System.out.println(interyop );
        System.out.println(interper);
        System.out.println(interclg);
        System.out.println(gradyop);
        System.out.println(gardper);
        System.out.println(gardclg);

    }

}

```

```

C:\Users\user\Desktop\KCCM87>java Studentinfo
STUDENT DATA ACCORDING TO RECORDS :
K.Sri Sai Vignesh
23
2015
63.65
Vigana High School
2017
70.0
VJG
2022
60.0
VLIITS

```

```

/* WAP TO PRINT PRODUCT DETAILS WHICH YOU PURCHASED RECENTLY */

public class Purchasedinfo
{
    public static void main(String arngs[])
    {
        //var dec-> datatype varname;
        String productname;
        String productbrand;
        String color;
        int price;
        float discount;
    }
}

```

```
String modeofpurchase;  
String dateofpurchase;  
//var init-> varname=value;  
productname = "IPHONE";  
productbrand = "APPLE";  
color = "RED";  
price = 50000;  
discount = 50.0F;  
modeofpurchase = "APPTONIX storenmae";  
dateofpurchase = "06-NOV-2020";  
//var utilisation  
System.out.println(productname);  
System.out.println(productbrand);  
System.out.println(color);  
System.out.println(price);  
System.out.println(discount);  
System.out.println(modeofpurchase);  
System.out.println(dateofpurchase);  
}  
}  
  
C:\Users\user\Desktop\KCCM87>javac Purchasedinfo.java  
C:\Users\user\Desktop\KCCM87>java Purchasedinfo  
IPHONE  
APPLE  
RED  
50000  
50.0  
APPTONIX storename  
06-NOV-2020
```

## **REINITIATION OF A VARIABLE:-**

- *Whatever value initialization we can change it this process re-initialization of variable.*
- *We reinitialize the old value will be replaced with new one.*

```
public class Reinitiase
{
    public static void main(String arngs[])
    {
        float per =76.56F;//var initialisation
        System.out.println(per);//76.56

        per =87.65F;//reinitialisation
        System.out.println(per);//87.65

        per =97.67F;//reinitialisation
        System.out.println(per);//97.67
    }
}
```

## **OPERATORS:-**

- *Operators are used performance some operations in our program.*
- *In java we have different types of operators*

### **1. ASSIGNMENT OPERATOR:**

- *It is used assign (or) initialize the value into variable*
- *It is denoted as “=”*

**Ex:-** a = 100;

*Assignment operators works right side to left side. It always stores right side value in left variable.*

### **2. COMPARISON OPERATOR:**

- *It is used to compare two value & return result in Boolean format.*
- *If value matches if return true otherwise if return false.*
- *It is represented as ==*

**NOTE:** Every character in java represented in **UTF (UNICODE TRANSFER FORMAT)** where every character have some Unicode associated to it.

**Ex: A-65,B-66,C-67** -----> **Z-90**

**a-97,b-98,c-99** -----> **z-122**

```
public class Comp
{
    public static void main(String arngs[])
    {
        int a = 100,b=100;
        boolean c;
        c = a==b;
        System.out.println(c);
        char ch = 'A',sh = 'a';
        boolean d;
        d = ch==sh;
        System.out.println(d);
    }
}
```

### **3. ARITHMETIC OPEATOR:**

- *Arithmetic it is used to perform some arithmetic operator.*
- *+,-,\*,/,%.*

**10+2** -----> **12**

**10-2** -----> **8**

**10\*2** -----> **20**

**10/2** -----> **5 (quotient)**

**10%2** -----> **0 (remainder)**

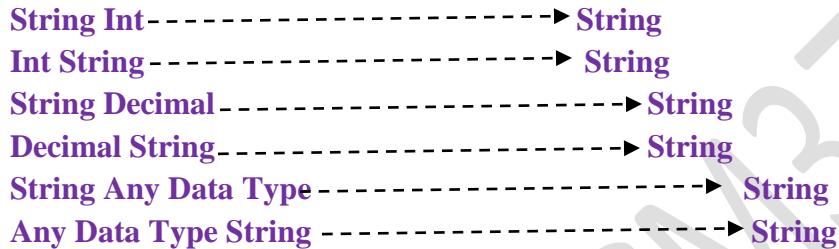
### **4. CONCATENATION OPEATOR:**

*Among all arithmetic operator + operator is working as addition and concatenation.*

## CONCATENATION:

- Will work as concatenations if any of the one operand is of String type & provide result also in String format.

Ex: a + b



Ex:

“JAVA” + 18 → JAVA18

“JAVA” + C → JAVAC

“JAVA”+500+300 → “JAVA500”+300 → JAVA500300

100 + 200 + “JAVA” → 300JAVA

Ex:-

```
String Name = “Pooja”;
```

```
System.out.println(“Name :”+Name);
```

O/P : Name : Pooja

Ex:

```
String Name = “Pooja”;
```

```
Int age = 22;
```

```
System.out.println(“Name :”+Name+”\nAge: “ +age);
```

O/P : Name : Pooja

Age : 22

Ex:-1

```
public class ProductDetails
```

```

{
    public static void main(String arngs[])
    {
        String pname="Shoes",brand="Adidas",color="White",
        mode="Adidas store",date="07-Sept-2022";
        int price=5999,dis=499;

        System.out.println("Product Details are : ");
        System.out.println("Product Name : "+pname);
        System.out.println("Product Brand : "+brand);
        System.out.println("Product Color : "+color);
        System.out.println("Purchase from : "+mode);
        System.out.println("Purchase On : "+date);
        System.out.println("Product Price : "+price+" rs/-");
        System.out.println("Discount avail: "+dis+" rs/-");
    }
}

```

```

C:\Users\user\Desktop\KCCM87>javac ProductDetails.java
C:\Users\user\Desktop\KCCM87>java ProductDetails
Product Details are :
Product Name : Shoes
Product Brand : Adidas
Product Color : White
Purchase from : Adidas store
Purchase On : 07-Sept-2022
Product Price : 5999 rs/-
Discount avail: 499 rs/-

```

**Ex:-2**

```
public class Student
```

```

{
public static void main(String arngs[])
{
String name = "Pooja",skl="St mary's high school",clg="Shantiniketan Jr Collge",
gradclg="CBIT Engg & Sciene";

int age=22,sscyop=2014,interyop=2016,gradyop=2020;

float sscper=98.78F,interper=96.78F,gradper=65.76F;

System.out.println("STUDENT DATA ACCORDING TO RECORDS : ");

System.out.println(name+"\n"+age+"\n"+skl+"\n"+sscyop+"\n"+sscper+"\n"+clg+"\n"+i
nteryop+"\n"+interper+"\n"+gradclg+"\n"+gradyop+"\n"+gradper);
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Student.java

C:\Users\user\Desktop\KCCM87>java Student
STUDENT DATA ACCORDING TO RECORDS :
Pooja
22
St mary's high school
2014
98.78
Shantiniketan Jr Collge
2016
96.78
CBIT Engg & Sciene
2020
65.76

```

## 5. RELATION OPERATOR:

*It is defines relationship b/w two operators and return the result in Boolean format.*

>  Greater Than

<  Lesser Than

$\geq$  → Greater Than

$\leq$  → Lesser Than

$\neq$  → Not Equal

## 6. LOGICAL OPERATOR:-

- It is defines logical relationship b/w two inputs and return result in Boolean.

### 1. LOGICAL AND (&&):-

- And operator return true if and only if both the inputs are true otherwise it return false.

IP-1	IP-2	IP-1 && IP-2
T	T	T
F	F	F
T	F	F
F	T	F

### 2. LOGICAL OR (||):-

- OR operator return true if any of the input is true if return false if both the input are false.

IP-1	IP-2	IP-1    IP-2
T	T	T
F	T	T
T	F	T
F	F	F

### 3. LOGICAL NOT(! ):-

- NOT operator make our result as opposite.

I/P	O/P
TRUE	FALSE
FALSE	TRUE

## EXAMPLE PROGRAMME:-

```
public class Op
{
    public static void main(String arngs[])
    {
    }
```

```

{
    int a=10,b=20,c=10,d=10;

    boolean e = a>b;

    boolean f = a<b;

    boolean h = a<b && c<d;

    boolean i = a>b || c>d;

    System.out.println(e+"\n"+f+"\n"+h+"\n"+i);
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Op.java
C:\Users\user\Desktop\KCCM87>java Op.java
false
true
false
false

```

## **7. INCREMENT OPERATOR:**

- *It increases the value by 1*
- *It represented as ++*
- *It is divided into 2 types*

### **1. PRE-INCREMENT:**

*Represented as ++ variable name;*

*Rule:*

- *Increment the Value.*
- *Assign the Value.*
- *Update the Value.*

### **2. POST INCREMENT:**

*Represented as variable name ++;*

*Rule:*

- *Assign the Value.*
- *Increment the Value.*

- *Update the Incremented Value.*

Ex -----► Program -----► Pre – Increment

Ex-1:

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 10,b;
        b = ++a;
        System.out.println(a+"\n"+b);
    }
}
```

Output:

11

11

Ex-2:

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 50,b,c,d;
        b = ++a;
        c = ++b;
        d = c;
        System.out.println(a+"\n"+b+"\n"+c+"\n"+d);
    }
}
```

```
 }  
 }
```

**Output:**

```
51  
52  
52  
52
```

**Ex-3:**

```
public class Opr  
{  
    public static void main(String arngs[])  
    {  
        int a = 30,b,c;  
        b = ++a + ++a;  
        c = ++b;  
        boolean d = c>b;  
        System.out.println(a+"\n"+b+"\n"+c+"\n"+d);  
    }  
}
```

**Output:**

```
32  
64  
64
```

**False**

**Ex**  **Program**  **Post – Increment**

**Ex-1:**

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 30,b;
        b = a++;
        System.out.println(a+"\n"+b);
    }
}
```

**Output:**

31

30

**Ex-2:**

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 20,b,c,d;
        b = a++;
        c = b++;
        d = c++;
        System.out.println(a+"\n"+b+"\n"+c+"\n"+d);
    }
}
```

**Output:**

**21**

**21**

**21**

**20**

**Ex-3:**

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 15,b,c;
        b = a++;
        c = b++;
        System.out.println(a+"\n"+b+"\n"+c);
        a++;
        b++;
        c++;
        System.out.println(a+"\n"+b+"\n"+c);
    }
}
```

**Output:**

**16**

**16**

**15**

**17**

17

16

#### 8. DECREMENT OPERATOR:

- *It increases the value by 1*
- *It represented as ++*
- *It is divided into 2 types*

##### 1. PRE-DECREMENT:

*Rule:*

- *Decrement the Value.*
- *Assign the Value.*
- *Update Decremented the Value.*

##### 2. POST DECREMENT:

*Rule:*

- *Assign the Value.*
- *Decrement the Value.*
- *Update the Decrement the Value.*

Ex -----> Program -----> Pre – Decrement

Ex-1:

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 100;
        int b = ++a + a++;
        int c = --b + b--;
        System.out.println(a+"\n"+b+"\n"+c);
    }
}
```

Output

102

200

402

Ex-2:

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 50;
        int b = --a + a--;
        int c = ++b + b++;
        System.out.println(a+"\n"+b+"\n"+c);
    }
}
```

Output:

48

100

198

### COMBINATIONAL OPERATOR:

*It is a combination of arithmetic operator and assignment operator.*

$+ = \rightarrow a = a + b \rightarrow a += b$

$- = \rightarrow a = a - b \rightarrow a -= b$

$* = \rightarrow a = a * b \rightarrow a *= b$

$/ = \rightarrow a = a / b \rightarrow a /= b$

$\% = \rightarrow a = a \% b \rightarrow a \% = b$

### **Examples:**

Count = Count+1;  Count += 1

i = i +1;  i += 1

a = b+c;  NA Because operand is not common in left and right

### **Ex-1:**

```
public class Opr
{
    public static void main(String arngs[])
    {
        int a = 50;
        int b = 100;
        b+= ++a;
        System.out.println(a+"\n"+b);
    }
}
```

### **Output:**

51

151

### **PRECEDENCE OF OPERATOR (Priority):**

- If two operands share a common operator then execution will take place based on priority.

### **LIST OF PRECEDENCE:**

- ✚ Post Increment and Post Decrement.
- ✚ Pre-Increment and Pre-Decrement.
- ✚ \*, %, /
- ✚ +, -
- ✚ Relational

-  Logical
-  Comparison
-  Combinational
-  Assignment

### Example-1:

```
public class Opr  
{  
    public static void main(String arngs[])  
    {  
        int a = 10,b = 5,c = 7;  
  
        int d;  
  
        d = a-++b-++c*2;  
  
        System.out.println(d);  
    }  
}
```

### Output:

-12

### Example-2:

```
public class Opr  
{  
    public static void main(String arngs[])  
    {  
        int x = 5;  
  
        x = x++ * 2 + 3 * - x;  
  
        System.out.println(x);  
    }  
}
```

}

## Output

-8

### **KEYWORDS:**

- ⊕ “Keyword are defined as pre-defined word or reserve word java”.
- ⊕ All the key all available in java library.
- ⊕ In java we have 58 keywords where every keyword will perform some task for a ‘Pr’.
- ⊕ So in simple keyword are given to make programmer task easier.
- ⊕ All the keyword in java must start with smaller case.

List of Java Keywords				
Primitive Types and void	Modifiers	Declarations	Control Flow	Miscellaneous
1.boolean	1.public	1.class	1.if	1.this
2.byte	2.protected	2.interface	2.else	2.new
3.char	3.private	3.enum	3.try	3.super
4.short	4.abstract	4.extends	4.catch	4.import
5.int	5.static	5.implements	5.finally	5.instanceof
6.long	6.final	6.package	6.do	6.null
7.float	7.transient	7.throws	7.while	7.true
8.double	8.volatle		8.for	8.false
9.void	9.synchronized		9.continue	9.strictfp
	10.native		10.break	10.assert
			11.switch	11._ (underscore)
			12.case	12.goto
			13.default	13.const
			14.throw	
			15.return	

### **IDENTIFIERS:**

*Identifier are the name given by the programmer as per convention.*

#### **Example:**

- ⊕ Class Name
- ⊕ Variable Name
- ⊕ Method Name
- ⊕ Package Name

-  Project Name
-  Interface Name

### **RULE FOR WRITING IDENTIFIER (COMPULSARY TO FOLLOW):**

1. An identifier can only be a combination of A-Z,a-z,0-9,\$.....
2. An identifier cannot start with a digit
3. An identifier cannot be a keyword
4. If an identifier has more than one word cannot use space between the

### **STANDARD FOR WRITING IDENTIFIER (OPTIONAL):**

1. An identifier length should not cross 15 characters.
2. A class name should start with capital.
3. A variable name should start with smaller.
4. If class name have more than one word for every word first letter keep capital.
5. If variable name have more than one word for starting word first letter keep small from second word every word first letter capital.

#### **RULE:**

- A-Z,a-z,0-9,\$,...  
 Class \$ample -----> Valid  
 Class S@mple -----> Invalid  
 Class S#ample -----> Valid
- Do not start with digit  
 Class 1sample -----> Invalid  
 Class s1ample -----> Valid
- Keyword  
 Class static -----> Invalid  
 Int if; -----> Invalid  
 Class String -----> Valid
- Does not allows spaces  
 Class My Program -----> Invalid  
 Class My\_Program -----> Valid

#### **STANDARD:**

- Class MyFirstProgramIsToDisplayDetails  
 Valid but not according to standard.
- Class Sample  
 Valid but not according to standard.
- Int Age;

Valid but not according to standard.

- Class MyDetails → acc to standard
- Class myDetails → acc to standard
- Int myAgeIs; → acc to standard
- Int MyAgeis; → acc to standard

### **TAKING THE I/P FROM THE USER WHILE EXECUTING THE PROGRAM:**

- In java we can take an I/P from the USER while running the program by using “SCANNER CLASS”
- Scanner is a pre-defined class which is there in java library using this we can take an input from the “USER”
- For using scanner class we have to follow following rules (or) steps

#### **STEP:1:- WRITE FIRST STATEMENT IN PROGRAM AS**

```
import java.util.Scanner;  
import java.util.*;
```

#### **STEP:2:- CREATE AN OBJECT OF SCANNER CLASS**

```
Scanner sc = new Scanner (System.in)
```

#### **STEP:-3: USING SCANNER CLASS REFERENCE CALL SCANNER CLASS METHOD**

Methods of Scanner Class are:

Byte Var = sc.nextByte() -----► For Taking Byte Value as I/P

Boolean Var = sc.nextBoolean()-----► For Taking Boolean Value as I/P

Short Var = sc.nextShort()-----► For Taking Short Value as I/P

**Int Var = sc.nextInt()** -----> For Taking Integer Value I/P

**Float Var = sc.nextFloat()** -----> For Taking Float Value I/P

**Double Var = sc.nextDouble()** -----> For Taking Double Value I/P

**Long Var = sc.nextLong()** -----> For Taking Long Value I/P

**String Var = sc.next()** -----> For Taking String Value I/P



(without space)

**String Var = sc.nextLine()** -----> For Taking String Value I/P



(with space)

#### Example:-

```
/* WAP TO ADD TWO NUMDERS */

import java.util.Scanner;

public class Added

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

        System.out.println("ENTER TWO NUMBERS");

        int a =sc.nextInt(),b =sc.nextInt(),c;
        c = a+b;

        System.out.println("Sum of two numbers : "+c);
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Added.java  
C:\Users\user\Desktop\KCCM87>java Added  
ENTER TWO NUMBERS  
10  
20  
Sum of two numbers : 30
```

### Example:2:-

```
/* WAP TO CALCULATE AREA OF RECTANGLE BY TAKING INPUT FROM THE  
USER */
```

```
import java.util.Scanner;  
  
public class Added  
{  
  
    public static void main(String args[])  
    {  
  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("ENTER LENGTH AND BREADTH ");  
  
        int a =sc.nextInt(),b =sc.nextInt(),c;  
  
        c = a*b;  
  
        System.out.println("Area of rectangle is : "+c);  
    }  
}
```

```
C:\Users\user\Desktop\KCCM87>java Added  
ENTER LENGTH AND BREADTH  
30  
45  
Area of rectangle is : 1350
```

### Example:3:-

```
/* WAP TO CALCULATE SIMPLE INTEREST BY TAKING THE INPUT BY USER*/
```

**FORMULA:-  $(P \cdot R \cdot T) / 100$**

P -----> Principle amt

R -----> Rate of Interest

T -----> Time in year

```
import java.util.Scanner;
```

```
public class SimpleInt
```

```
{
```

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

```
{
```

```
        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Enter the amount ");
```

```
        int p = sc.nextInt();
```

```
        System.out.println("Enter time duration");
```

```
        int t = sc.nextInt();
```

```
        System.out.println("Enter rate of interest");
```

```
        float r = sc.nextFloat();
```

```
        si = (p*r*t)/100;
```

```
        System.out.println("Simple rate of Interest : "+si);
```

```
}
```

```
}
```

```
C:\Users\user\Desktop\KCCM87>java SimpleInt
Enter the amount
500000
Enter time duration
5
Enter rate of interest
7.85
Simple rate of Interest : 196250.0
```

Example:4:-

```
/* WAP TO ADD TWO DECIMAL NUMBER BY TAKING INPUT FROM USER */
```

```

import java.util.Scanner;

public class Added

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

        System.out.println("ENTER TWO DECIMAL VALUE");

        Float a = sc.nextFloat(), b = sc.nextFloat(), c;

        c = a+b;

        System.out.println("SUM OF TWO DECIMAL NUMBER ARE: "+c);
    }
}

```

```

C:\Users\user\Desktop\KCCM87>java Added
ENTER TWO DECIMAL VALUE
56.99
67.88
SUM OF TWO DECIMAL NUMBER ARE: 124.869995

```

#### Example:-

```

/* WAP TO CALCULATE AREA OF CIRCLE, CIRCUMFERENCE A CIRCLE, AREA
OF SQUARE BY USING INPUTS FROM USER */

import java.util.Scanner;

public class Areas

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

        System.out.println("Enter the radius"+"\n"+"Enter side of square");

```

```

float pie =3.14f,a,c;

int r=sc.nextInt(),s=sc.nextInt(),area;

a= pie*r*r;

c= 2*pie*r;

area=s*s;

System.out.println("area of circle is:"+a+ "\n"+ "circumference of circles is:"+c+
"\n"+ "area of square is:"+area);

}

}

```

```

C:\Users\user\Desktop\KCCM87>java Areas
Enter the radius
Enter side of square

12
12
area of circle is:452.16
circumference of circles is:75.36
area of square is:144

```

## CONTROL FLOW SYSTEM:-

*These are system which controls the flow of execution.*

*It is classified into types:*

- ✚ CONDITIONAL STATEMENT
- ✚ SWITCH CASE STATEMENT

### CONDITIONAL STATEMENT:

- *These are statements which check condition and execute accordingly.*
- *Whenever we want to check any condition, then will go conditional statement.*

```
if (condition)
{
    // body of if
}
Else
{
    // body of else
}
```

**Syntax:**

**WORKING:**

- **JVM WILL CHECK CONDITION GIVEN AT if & INCASE IF CONDITION *if-body* WILL BE EXECUTED AND ELSE WILL BE SKIPPED**
- **INCASE IF CONDITION IS FALSE *if-body* WILL BE SKIPPED ELSE WILL BE EXECUTED.**

**/\* WAP TO CHECK WHETHER A PERSON IS ELIGIBLE TO CAST THE VOTE OR NOT TAKE THE AGE VALE THROUGH SCANNER CLASS \*/**

```
import java.util.Scanner;
public class Voting
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the age");
        int age = sc.nextInt();
        if(age>=18)
        {
            System.out.println(" person is eligible to vote");
        }
    }
}
```

```
else
{
    System.out.println("person is not eligible to vote");
}
}
```

```
C:\Users\user\Desktop\KCCM87>javac Voting.java
C:\Users\user\Desktop\KCCM87>java Voting
Enter the age
34
person is eligible to vote
```

#### **/\* WAP TO PRINT GREATEST OF TWO NUMBER \*/**

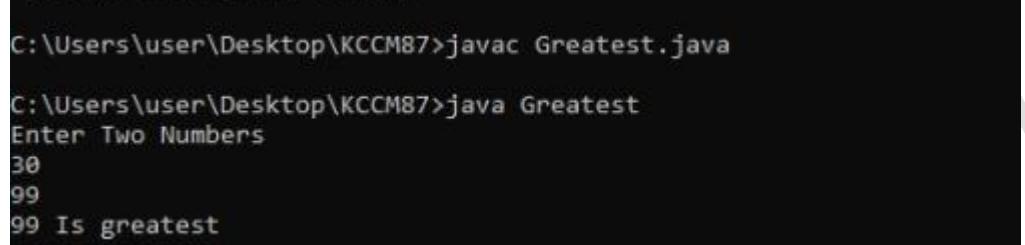
```
import java.util.Scanner;
public class Greatest
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Two Numbers");
        int a = sc.nextInt(), b = sc.nextInt();
        if (a>b)
        {
            System.out.println(a+" Is greatest");
        }
        else
        {
```

```
        System.out.println(b+" Is greatest");

    }

}

}
```



```
C:\Users\user\Desktop\KCCM87>javac Greatest.java
C:\Users\user\Desktop\KCCM87>java Greatest
Enter Two Numbers
30
99
99 Is greatest
```

```
/* WAP TO CHECK WHETHER THE NUMBER IS EVEN OR ODD NUMBER */

import java.util.Scanner;

public class Even

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

        System.out.println("Enter the Numbers");

        int a = sc.nextInt();

        if (a%2 == 0)

        {
            System.out.println( a+" Is EVEN");

        }

        else

        {
            System.out.println(a+" Is ODD");
        }
    }
}
```

```
}

}

}

C:\Users\user\Desktop\KCCM87>javac Even.java

C:\Users\user\Desktop\KCCM87>java Even
Enter the Numbers
24
24 Is EVEN
```

⊕ When we want check multiple conditions, then we will use.

*Syntax:*

*If/else if/else statement*

*If (condition)*

{

*// body of if*

}

*Else if (condition)*

{

*// body of else if* -----→1

}

*Else if (condition)*

{

*// body of else if* -----→2

}

*Else*

{

*// body of else*

```
}

/* WAP TO CHECK WHETHER A POSITIVE NUMBER, NEGATIVE NUMBER OR
ZERO*/

import java.util.Scanner;

public class Positive

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

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

        int a = sc.nextInt();

        if(a>0)
        {
            System.out.println(a+" Is Postive");
        }
        else if(a<0)
        {
            System.out.println(a+" Is Negative");
        }
        else
        {
            System.out.println(a+" Is Zero");
        }
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Positive.java
C:\Users\user\Desktop\KCCM87>java Positive
Enter Numbers
-333
-333  is negative

C:\Users\user\Desktop\KCCM87>java Positive
Enter Numbers
45
45  is positive

C:\Users\user\Desktop\KCCM87>java Positive
Enter Numbers
0
0  is Zero
```

## ASSIGNMENT

**/\*WAP TO CHECK FOLLOWING CONDITIONS IN SIMPLE PROGRAM\*/**

- Print Tom if number is even and in between 10 and 25.
- Print Jerry if number is odd and in between 10 and 25.
- Print Tom and Jerry if above two conditions are false

```
import java.util.Scanner;

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

        System.out.println("Enter a Number");

        int a = sc.nextInt();

        if (a%2==0 && a>10 && a<25)
            System.out.println(" IS TOM ");

        else if (a%2!=0 && a>10 && a<25)
            System.out.println(" IS JERRY ");

        else
            System.out.println(" TOM AND JERRY ");
```

```
}
```

```
C:\Users\user\Desktop\KCCM87>javac TomJerry.java
C:\Users\user\Desktop\KCCM87>java TomJerry
Enter a Number
21
IS JERRY

C:\Users\user\Desktop\KCCM87>java TomJerry
Enter a Number
17
IS JERRY

C:\Users\user\Desktop\KCCM87>java TomJerry
Enter a Number
37
TOM AND JERRY
```

/\*WAP TO CHECK WHETHER 23 IS DIVISIBLE IN 13 AND 17 (IF/ELSE) \*/

```
public class Div
{
    public static void main(String args[])
    {
        int a = 23;
        if (a%13==0 && a%17==0)
            System.out.println(a+ " IS Div ");
        else
            System.out.println(a+ " IS NOT DIV");
    }
}
```

```
C:\Users\user\Desktop\KCCM87>
C:\Users\user\Desktop\KCCM87>javac Div.java

C:\Users\user\Desktop\KCCM87>java Div
23 IS NOT DIV

C:\Users\user\Desktop\KCCM87>
```

/\*WAP TO PRINT GREATEST OF 3 NUMBER 33,66,17 (IF/ELSE IF/ELSE) \*/

```
public class GR
```

```

{
public static void main(String args[])
{
int a = 33,b = 17,c = 66;
if (a>b && a>c)
System.out.println(a+" Is greatest");
else if(b>a && b>c)
System.out.println(b+" Is greatest");
else
System.out.println(c+" Is greatest");
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac GR.java
C:\Users\user\Desktop\KCCM87>java GR
66 Is greatest
C:\Users\user\Desktop\KCCM87>_

```

**/\*WAP TO CHECK 25 IS DIVISIBLE IN 5 OR 3. (IF/ELSE)\*/**

```

public class Divisible
{
public static void main(String args[])
{
int a = 25;
if (a%5==0 || a%3==0)
System.out.println(a+ " IS DIVISIBLE ");
}

```

```
else  
    System.out.println(a+ " IS NOT DIVISIBLE ");  
}  
}
```

```
C:\Users\user\Desktop\KCCM87>javac Divisible.java  
C:\Users\user\Desktop\KCCM87>java Divisible  
25 IS DIVISIBLE  
C:\Users\user\Desktop\KCCM87>
```

**/\*WAP TO CHECK WEATHER A PERSON IS ELIGIBLE FOR IAS OR NOT CRITERIA IS AGE IS BETWEEN 22 TO 35. (IF/ELSE) \*/**

```
public class IAS  
{  
    public static void main(String args[])  
    {  
        int age = 25;  
        if (age>22 && age<35)  
            System.out.println("THE PERSON IS ELIGIBLE FOR IAS");  
        else  
            System.out.println("THE PERSON IS NOT ELIGIBLE FOR IAS");  
    }  
}
```

```
C:\Users\user\Desktop\KCCM87>javac IAS.java  
C:\Users\user\Desktop\KCCM87>java IAS  
THE PERSON IS ELIGIBLE FOR IAS  
C:\Users\user\Desktop\KCCM87>
```

*If we have single statement in (if or else) body then we can skip flower braces. But if we have more than one system it is compulsory flower braces.*

**SWITCH CASE STATEMENT:** When we want to test multiple condition we will go for switch case statement.

*Switch (Variable whose value you want to check)*

*Syntax:*

{

*Case Value 1 : Statement*

*Break;*

*Case Value 2 : Statement*

*Break;*

*Case Value 3 : Statement*

*Break;*

*Case Value 4 : Statement*

*Break;*

*Default    : Statement*

*Break;*

}

**BREAK:**

*It is a keyword which indicates us to stop the current operation and make JVM to come out of currently executing body.*

**What if we skip break key word?**

*Situation – 1: If value matches and skip break keyword then JVM will execute all remaining cases until it finds break keyword/statements.*

*Situation – 2 : If value does not matches and we skip break keyword it will not effect our output.*

**/\*WAP TO PRINT DAYNAME USING DAYNUMBER\*/**

DAYNUM	DAYNAME
1	MONDAY
2	TUESDAY
3	WEDNESDAY
7	SUNDAY
< 1 OR >7	INVALID DAY

```

import java.util.Scanner;

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

        System.out.println("Enter a Number");

        int dn = sc.nextInt();

        switch(dn)
        {
            case 1 : System.out.println("MONDAY");
                        break;
            case 2 : System.out.println("TUESDAY");
                        break;
            case 3 : System.out.println("WEDNESDAY");
                        break;
            case 4 : System.out.println("THURSDAY");
                        break;
            case 5 : System.out.println("FRIDAY");
                        break;
        }
    }
}

```

```

case 6 : System.out.println("SATURDAY");
    break;

case 7 : System.out.println("SUNDAY");
    break;

default : System.out.println("INVALID DAY");
    break;
}

}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Dayn.java

C:\Users\user\Desktop\KCCM87>java Dayn
Enter a Number
4
THURSDAY

C:\Users\user\Desktop\KCCM87>java Dayn
Enter a Number
67
INVALID DAY

```

**/\*WAP TO PRINT DAYTYPE USING DAYNUMBER\*/**

DAYNUM	DAYTYPE
1,2,3,4,5	WEEKDAY
6	WEEKDAY-1
7	WEEKDAY-2
>7 OR <1	INVALID DAY

```

import java.util.Scanner;

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

```

```

Scanner sc = new Scanner(System.in);

System.out.println("Enter a Number");

int dn = sc.nextInt();

switch(dn)

{

    case 1,2,3,4,5 : System.out.println("WEEKDAY");

        break;

    case 6 : System.out.println("WEEKDAY-1");

        break;

    case 7 : System.out.println("WEEKDAY-2");

        break;

    default : System.out.println("INVALID DAY");

        break;

}

}

}

```

```

C:\Users\user\Desktop\KCCM87>javac Dayn.java

C:\Users\user\Desktop\KCCM87>java Dayn
Enter a Number
5
WEEKDAY

C:\Users\user\Desktop\KCCM87>java Dayn
Enter a Number
6
WEEKDAY-1

C:\Users\user\Desktop\KCCM87>java Dayn
Enter a Number
7
WEEKDAY-2

C:\Users\user\Desktop\KCCM87>java Dayn
Enter a Number
45
INVALID DAY

```

**/\*WAP TO PRINT MONTHYPE USING DAYNUMBER\*/**

DAYNUM	DAYTYPE
3,4,5,6	SUMMER
7,8,9,10	RAINY
11,12,1,2	WINTER
12>OR <1	INVALID DAY

```

import java.util.Scanner;

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

        System.out.println("Enter a Number");

        int dn = sc.nextInt();

        switch(dn)
        {
            case 3,4,5,6 : System.out.println("SUMMER");
                            break;
            case 7,8,9,10 : System.out.println("RAINY");
                            break;
            case 11,12,1,2 : System.out.println("WINTER");
                            break;
            default : System.out.println("INVALID DAY");
                        break;
        }
    }
}

```

```
}
```

```
C:\Users\user\Desktop\KCCM87>javac Monthtype.java
C:\Users\user\Desktop\KCCM87>java Monthtype
Enter a Number
4
SUMMER

C:\Users\user\Desktop\KCCM87>java Monthtype
Enter a Number
9
RAINY

C:\Users\user\Desktop\KCCM87>java Monthtype
Enter a Number
2
WINTER

C:\Users\user\Desktop\KCCM87>java Monthtype
Enter a Number
34
INVALID DAY
```

### /\*WAP TO PRINT WHETHER A CHARACTER IS VOWEL OR NOT\*/

CHARACTER	TYPE
A,E,I,O,U	CAPTIAL VOWEL
a,e,i,o,u	SMALLER VOWEL
Other	CONSONANT

```
import java.util.Scanner;
public class Vowel
{
    public static void main(String args[])
    {
        //char ch = 'e';
        Scanner sc = new Scanner(System.in);
        System.out.println("ENTER CHARACTER");
        char ch = sc.next().charAt(0);
        switch(ch)
```

```
{  
    case 'A','E','I','O','U' : System.out.println(" CAPITAL VOWEL ")  
        break;  
  
    case 'a','e','i','o','u'   : System.out.println("SMALL VOWEL ");  
        break;  
  
    default : System.out.println("MAY BE CONSONANT OR SPECIAL SYMBOL");  
        break;  
}  
}  
}
```

```
C:\Users\user\Desktop\KCCM87>javac Vowel.java
C:\Users\user\Desktop\KCCM87>java Vowel
ENTER CHARACTER
A
CAPITAL VOWEL

C:\Users\user\Desktop\KCCM87>java Vowel
ENTER CHARACTER
e
SMALL VOWEL

C:\Users\user\Desktop\KCCM87>java Vowel
ENTER CHARACTER
W
MAY BE CONSONANT OR SPECIAL SYMBOL
```

## **NOTE:**

- *In switch case statement case value can't be Boolean, float, double & long.*
  - *In switch case statement case value can't be duplicate irrespective of its match (or) not that is all the case value must be unique.*

## **Difference b/w Switch Case and Conditional Statements**

CONDITIONAL	SWITCH
<p>1. Syntax:</p> <pre>If (condition) {     // body of if } Else</pre>	<p>1. Syntax:</p> <pre>Switch(variable whose value you want take) {     case value 1,2,3,4, : statement         break;</pre>

```
{
    // body of else
}
```

2. In conditional statements execution will happen based on condition.
3. If none of condition satisfies else will be executed.
4. Conditional statement supports all the data type.
5. Using conditional statement we can check multiple condition at same time.

```
default : statement
break;
}
```

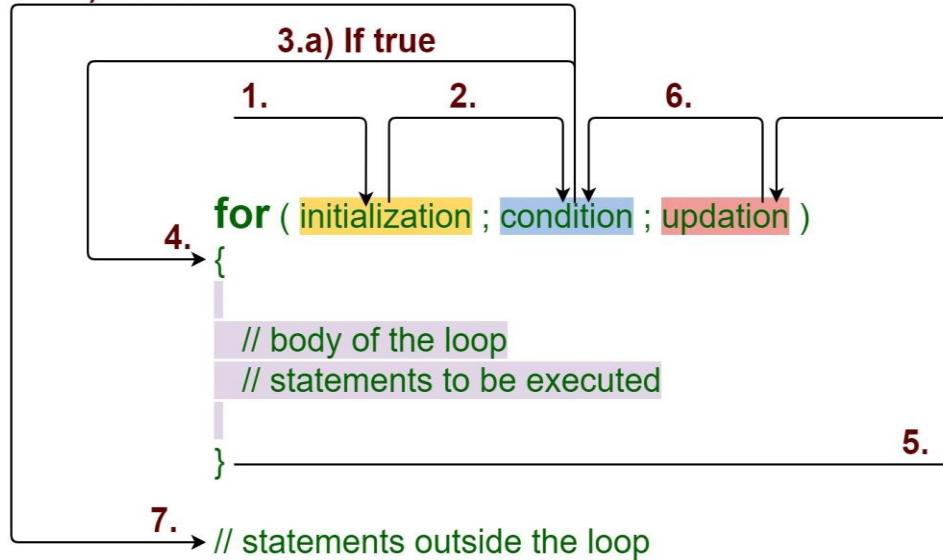
2. Switch case statement execution will happen based on switch expression.
3. If none if the case value match default will be executed.
4. Switch case support only byte, short,int,char & string data type.
5. Using switch case we can evaluate only one expression at a time

## LOOPING STATEMENT:

- ✚ If any part of a code is repeatedly executing than rather than developing it repeatedly we can keep it once in a loop & make it to run multiple times.
- ✚ Loop -----Repetition
- ✚ We have 4 types of loop in JAVA, FOR WHILE AND FOR EACH LOOP.

## For Loop

### 3.b) If false



**/\*WAP TO PRINT YOUR NAME 10 TIMES\*/**

```
public class Name
{
    public static void main(String args[])
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println("java");
        }
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Name.java
C:\Users\user\Desktop\KCCM87>java Name
java
```

**/\*CREATE A FOR LOOP TO PRINT THE NUMBERS FROM 1 TO 10 \*/**

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

```
 }
}
}
```

```
C:\Users\user\Desktop\KCCM87>javac Sri.java
C:\Users\user\Desktop\KCCM87>java Sri
1
2
3
4
5
6
7
8
9
10
```

**/\* CREATE A FOR LOOP TO PRINT THE NUMBER FROM 10 TO 1 \*/**

```
public class Sri
{
    public static void main(String args[])
    {
        for(int i=10;i>=1;i--)
        {
            System.out.println(i);
        }
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Sri.java
C:\Users\user\Desktop\KCCM87>java Sri
10
9
8
7
6
5
4
3
2
1
```

```

/* CREATE A FOR LOOP TO PRINT A TO Z CHARACTER */

public class Alphabet

{
    public static void main(String args[])
    {
        for(char i='A';i<='Z';i++)
        {
            System.out.print(i+" ");
        }
        System.out.println();
        for (char ch=97;ch<=122;ch++)
        {
            System.out.print(ch+" ");
        }
    }
}

```

```

C:\Users\user\Desktop\KCCM87>javac Alphabet.java
C:\Users\user\Desktop\KCCM87>java Alphabet
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
C:\Users\user\Desktop\KCCM87>

```

**/\* WAP TO PRINT ALL THE EVEN NUMBER FROM 1 TO N TAKE THE N VALUE FROM USER \*/**

```

import java.util.Scanner;

public class EN

{

```

```

public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter N-Value");
    int n = sc.nextInt();
    for(int i=1;i<=n;i++)
    {
        if(i%2==0)
        {
            System.out.print(i+" ");
        }
    }
}

```

```

C:\Users\user\Desktop\KCCM87>javac EN.java
C:\Users\user\Desktop\KCCM87>java EN
Enter N-Value
6
2 4 6
C:\Users\user\Desktop\KCCM87>java EN
Enter N-Value
48
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48

```

/\* WAP TO COUNT ALL THE EVEN NUMBERS FROM 1 TO N TAKE N VALUE FROM USER \*/

```

import java.util.Scanner;

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

```

```

{
Scanner sc = new Scanner(System.in);
System.out.println("Enter N-Value");
int n = sc.nextInt(), count=0;
for(int i=1;i<=n;i++)
{
if(i%2==0)
{
count++;
}
}
System.out.println("Final Count : "+count);
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac ENs.java
C:\Users\user\Desktop\KCCM87>java ENs
Enter N-Value
20
Final Count : 10

```

**/\* WAP TO PRINT SUM OF ALL THE EVEN NUMBERS FROM 1 TO N VALUE  
FROM USER \*/**

```

import java.util.Scanner;
public class Sum
{
public static void main(String args[])
{

```

```

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt(),sum=0;

for(int i=1;i<=n;i++)

{

if(i%2==0)

{

    sum=i+sum;

}

}

System.out.println("SUM OF EVEN IS : "+sum);

}

}

```

```

C:\Users\user\Desktop\KCCM87>javac Sum.java

C:\Users\user\Desktop\KCCM87>java Sum
ENTER NUMBER
10
SUM OF EVEN IS : 30

```

**/\* WAP TO PRINT PRODUCT OF EVEN NUMBERS FROM 1 TO N (N VALUE MUST BE LESS THAN 15) \*/**

```

import java.util.Scanner;

public class Sum

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

```

```

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

int n = sc.nextInt(), p=1;

for(int i=1;i<=n;i++)

{

if(i%2==0)

{

p=p*i;

}

}

System.out.println("PRODUCT OF EVEN IS : "+p);

}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Sum.java

C:\Users\user\Desktop\KCCM87>java Sum
ENTER NUMBER
15
PRODUCT OF EVEN IS : 645120

```

/\* WAP TO PRINT SUM OF EVEN AND PRODUCT OF ODD FROM 1 TO N (N VALUE MUST BE LESS THAN 15) \*/

```

import java.util.Scanner;

public class Sums

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter N-Value");

```

```

int n = sc.nextInt(),sum=0,p=1;

for(int i=1;i<=n;i++)
{
    if(i%2==0)
    {
        sum=i+sum;
    }
    else
    {
        p=i*p;
    }
}

System.out.println("sum of even numbers is:" +sum + "\n "+"product of odd numbers
is:" +p);
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Sums.java

C:\Users\user\Desktop\KCCM87>java Sums
Enter N-Value
15
sum of even numbers is:56
product of odd numbers is:2027025

```

/\* WAP TO PRINT COUNT OF EVEN AND COUNT OF ODD FROM 1 TO N \*/

```

import java.util.Scanner;

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

```

```

{
Scanner sc = new Scanner(System.in);
System.out.println("Enter Number");
int n = sc.nextInt(), count_e=0, count_o=0;
for(int i=1;i<=n;i++)
{
if(i%2==0)
    count_e++;
else
    count_o++;
}
System.out.println("COUNT OF EVEN = "+count_e);
System.out.println("COUNT OF ODD = "+count_o);
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac SumE.java
C:\Users\user\Desktop\KCCM87>java SumE
Enter Number
15
COUNT OF EVEN = 7
COUNT OF ODD = 8

```

**/\* WAP PROGRAM TO PRINT MUITPLE TABLE FROM N NUMBER TAKE N VALUE FROM USER \*/**

```

import java.util.Scanner;

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

```

```

{
Scanner sc = new Scanner(System.in);
System.out.println("Enter Number");
int n = sc.nextInt();
for(int i=1;i<=10;i++)
{
System.out.println(n+"*" +i+" ="+(n*i));
}
}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Table.java
C:\Users\user\Desktop\KCCM87>java Table
Enter Number
2
2*1 =2
2*2 =4
2*3 =6
2*4 =8
2*5 =10
2*6 =12
2*7 =14
2*8 =16
2*9 =18
2*10 =20

```

**/\*WAP TO SWAP TWO NUMBERS\*/**

—————> SWAP USING EXTRA VARIABLE  
 —————> SWAP USING WITHOUT EXTRA VARIABLE

**SWAP USING EXTRA VARIABLE**

```

import java.util.Scanner;
public class Swap1
{
public static void main(String args[])
{

```

```

Scanner sc = new Scanner(System.in);

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

System.out.println(" THE VALUE OF A: ");

System.out.println(" THE VALUE OF B: ");

int A = sc.nextInt();

int B = sc.nextInt();

int C ;

C = A;

A = B;

B = C;

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

System.out.println(" NOW THE VALUE OF A IS: "+A+"\n NOW THE VALUE OF B
IS : "+B);

}

}

```

```

C:\Users\user\Desktop\KCCM87>javac Swap1.java

C:\Users\user\Desktop\KCCM87>java Swap1
BEFORE SWAPPING
THE VALUE OF A:
THE VALUE OF B:
10
20
AFTER SWAPPING
NOW THE VALUE OF A IS: 20
NOW THE VALUE OF B IS : 10

```

## SWAP USING WITHOUT EXTRA VARIABLE

```

import java.util.Scanner;

public class Swap

{

public static void main(String args[])

{

```

```

Scanner sc = new Scanner(System.in);

System.out.println(" THE VALUE OF A: ");

System.out.println(" THE VALUE OF B: ");

int A = sc.nextInt();

int B = sc.nextInt();

A = A + B;

B = A - B;

A = A - B;

System.out.println(" NOW THE VALUE OF A IS: "+A+"\n NOW THE VALUE OF B
IS : "+B);

}

}

```

```

C:\Users\user\Desktop\KCCM87>javac Swap.java

C:\Users\user\Desktop\KCCM87>java Swap
THE VALUE OF A:
THE VALUE OF B:
12
25
NOW THE VALUE OF A IS: 25
NOW RHE VALUE OF B IS : 12

```

**/\* WAP TO PRINT FIBNOCCI SERIES \*/**

**FIBNOCCI SERIES:** A series of number whose first two terms are 0 & 1, from third term is every number sum of previous two number.

```

import java.util.Scanner;

public class Swap2

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

```

```

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int A = 0,B = 1,C;

System.out.print(A+" ");

System.out.print(B+" ");

for(int i=1;i<=n;i++)

{

    C = A + B;

    System.out.print(C+" ");

    A = B;

    B = C;

}

}

```

```

C:\Users\user\Desktop\KCCM87>javac Swap2.java

C:\Users\user\Desktop\KCCM87>java Swap2
19
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

```

**/\* WAP TO CHECK WEATHER A NUMBER IS PRIME OR NOT \*/**

**PRIME NUMBER:** *Prime number is a number which is divisible only in 1 & itself.*

```

import java.util.Scanner;

public class Prime

{

    public static void main(String args[])

    {

```

```

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int count = 0;

for(int i=1;i<=n;i++)

{

    if(n%i==0)

    {

        count++;

    }

}

if(count==2)

System.out.println(n+" IS PRIME");

else

System.out.println(n+" IS NOT PRIME");

}
}

```

```

C:\Users\user\Desktop\KCCM87>javac Prime.java

C:\Users\user\Desktop\KCCM87>java Prime
ENTER NUMBER
5
5 IS PRIME

C:\Users\user\Desktop\KCCM87>java Prime
ENTER NUMBER
14
14 IS NOT PRIME

```

**/\* WAP TO PRINT ALL THE PRIME NUMBERS FROM N1 TO N2 TAKE N1 & N2  
VALUE FROM USER \*/**

```
import java.util.Scanner;
```

```
public class Prime1
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Number");
        int n1 = sc.nextInt(),n2 = sc.nextInt();
        for(int i=n1;i<=n2;i++)
        {
            int count=0;
            for(int j=1;j<=i;j++)
            {
                if(i%j==0)
                    count++;
            }
            if(count==2)
                System.out.println(i+" IS A PRIME NUMBER");
        }
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Prime1.java
```

```
C:\Users\user\Desktop\KCCM87>java Prime1
```

```
Enter Number
```

```
5
```

```
50
```

```
5 IS A PRIME NUMBER
```

```
7 IS A PRIME NUMBER
```

```
11 IS A PRIME NUMBER
```

```
13 IS A PRIME NUMBER
```

```
17 IS A PRIME NUMBER
```

```
19 IS A PRIME NUMBER
```

```
23 IS A PRIME NUMBER
```

```
29 IS A PRIME NUMBER
```

```
31 IS A PRIME NUMBER
```

```
37 IS A PRIME NUMBER
```

```
41 IS A PRIME NUMBER
```

```
43 IS A PRIME NUMBER
```

```
47 IS A PRIME NUMBER
```

```
/* WAP SUM AND COUNT OF ALL THE PRIME NUMBERS FROM N1 TO N2 */

import java.util.Scanner;

public class Sum1

{

    public static void main (String args[])

    {

        Scanner sc = new Scanner(System.in);

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

        int n1=sc.nextInt(),n2=sc.nextInt();

        int countP=0,sum=0;

        for (int i=n1;i<=n2,i++)

        {

            int count=0;

            for(int j=1;j<=i;j++)

            {

                if(i%j==0)

                    count++;

            }

            if(count==2)

            {

                countP++;

                sum=sum+i;

            }

        }

        System.out.println("SUM OF PRIME : "+sum);

    }

}
```

```

        System.out.println("COUNT OF PRIME : "+countP);
    }
}

```

```

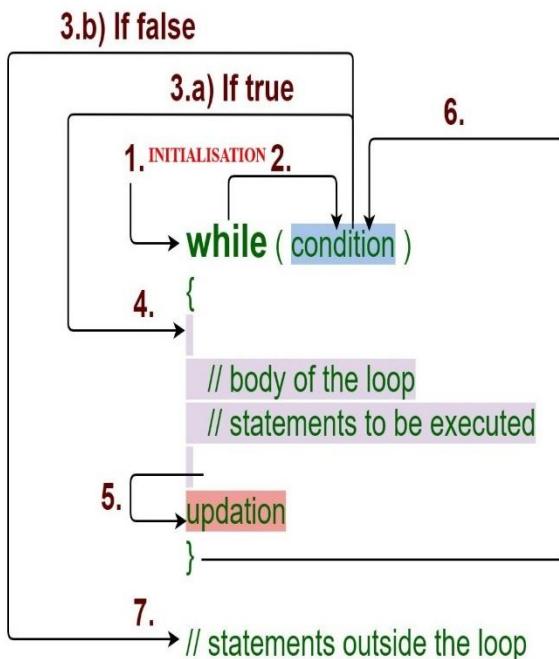
C:\Users\user\Desktop\KCCM87>javac Sum1.java
C:\Users\user\Desktop\KCCM87>java Sum1
ENTER NUMBER
15
29
SUM OF PRIME : 88
COUNT OF PRIME : 4

```

### WHILE LOOP:

Syntax:

## While Loop



### WORKING OF WHILE LOOP:

STEP – 1 : INITIALISATION

STEP – 2 : CHECKING OF CONDITION

→ IF CONDITION TRUE GO TO STEP – 3

→ IF CONDITION FALSE, COME OUT OF THE LOOP

STEP – 3 : EXECUTE BODY OF THE LOOP

#### **STEP – 4 : PERFORM UPDATION, GO TO STEP – 2**

```
/* CREATE A WHILE TO PRINT THE NUMBER FROM 1 TO 10 */

public class Loop2

{
    public static void main(String args[])
    {
        int i = 1;

        while(i<=10)
        {
            System.out.println(i);

            i++;
        }
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Loop2.java

C:\Users\user\Desktop\KCCM87>java Loop2
1
2
3
4
5
6
7
8
9
10
```

#### **/\* CREATE A WHILE TO PRINT THE NUMBER FROM 10 TO 1 \*/**

```
public class Loop3

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

```
{  
    int i = 10;  
    while(i<=1)  
    {  
        System.out.println(i);  
        i--;  
    }  
}
```

```
C:\Users\user\Desktop\KCCM87>java Loop3.java  
C:\Users\user\Desktop\KCCM87>java Loop3  
10  
9  
8  
7  
6  
5  
4  
3  
2  
1
```

**/\* CREATE A WHILE LOOP TO PRINT -11 TO +11 \*/**

```
public class Loop4  
{  
    public static void main(String args[])  
    {  
        int i = -11;  
        while(i<=11)  
        {  
            System.out.print(i+" ");  
        }  
    }  
}
```

```
i++;  
}  
}  
}
```

```
C:\Users\user\Desktop\KCCM87>javac Loop4.java  
C:\Users\user\Desktop\KCCM87>java Loop4  
-11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11
```

**/\* WAP TO PRINT A TO Z USING WHILE LOOP \*/**

```
public class Loop5  
{  
    public static void main(String args[])  
    {  
        char ch = 'A';  
        while(ch<='Z')  
        {  
            System.out.print(ch+" ");  
            ch++;  
        }  
    }  
}
```

```
C:\Users\user\Desktop\KCCM87>javac Loop5.java  
C:\Users\user\Desktop\KCCM87>java Loop5  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

**/\* WAP TO PRINT REVERSE OR NUMBER\*/**

```
import java.util.Scanner;  
public class Reverse
```

```

{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("ENTER A NUMBER");
        int num = sc.nextInt(), rem, rev=0;
        while(num>0)
        {
            rem = num%10;
            num = num/10;
            rev = rev*10+rem;
        }
        System.out.println("REVERSE OF NUMBER IS : "+rev);
    }
}

```

```

C:\Users\user\Desktop\KCCM87>javac Reverse.java
C:\Users\user\Desktop\KCCM87>java Reverse
ENTER A NUMBER
123
REVERSE OF NUMBER IS : 321

C:\Users\user\Desktop\KCCM87>java Reverse
ENTER A NUMBER
56678
REVERSE OF NUMBER IS : 87665

```

#### **NOTE:-**

*If we divided an ‘n’ digit number with 10 will we will remainder as last digit & remaining digits as Quotient.*

**PALINDROME NUMBER:-** *If original number & reverse of an number is same such number is called as palindrome number.*

**Ex** → **121,111,323,1221 ..etc.**

**/\* WAP TO CHECK WHETHER A NUMBER PALINDROME OR NOT \*/**

```
import java.util.Scanner;

public class Palindrome

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

        System.out.println("ENTER A NUMBER");

        int num = sc.nextInt(),rem,rev=0,temp=num;

        while(num>0)

        {
            rem = num%10;
            num = num/10;
            rev = rev*10+rem;
        }

        System.out.println("ACTUAL NUMBER IS : "+temp);

        System.out.println("REVERSE OF A NUMBER IS : "+rev);

        if(rev==temp)

            System.out.println(temp+" IS PALINDROME NUMBER ");

        else

            System.out.println(temp+" IS NON PALINDROME NUMBER");

    }
}
```

```
C:\Users\user\Desktop\KCCM87>java Palindrome
ENTER A NUMBER
121
ACTUAL NUMBER IS : 121
REVERSE OF A NUMBER IS : 121
121 IS PALINDROME NUMBER

C:\Users\user\Desktop\KCCM87>java Palindrome
ENTER A NUMBER
23
ACTUAL NUMBER IS : 23
REVERSE OF A NUMBER IS : 32
23 IS NON PALINDROME NUMBER
```

**/\* WAP TO CHECK WEATHER A NUMBER IS ARMSTRONG \*/**

**ARMSTRONG:-** A number is armstrong number if we individually cube every digits of a number & add them in case if we get a sum number such is called **armstrong number**.

Ex  $\rightarrow 153 = 1^3 + 3^3 + 5^3.$

```
import java.util.Scanner;

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

        System.out.println("ENTER A NUMBER");

        int num = sc.nextInt();

        int rem,arm=0,temp=num;

        while(num>0)
        {
            rem = num%10;

            num = num/10;

            arm = arm+rem*rem*rem;
        }

        if(arm == temp)

            System.out.println(temp+" IS AN ARMSTRONG NUMBER ");

        else
```

```

        System.out.println(temp+" IS NOT AN ARMSTRONG NUMBER");

    }

}

```

```

C:\Users\user\Desktop\KCCM87>javac Armstrong.java

C:\Users\user\Desktop\KCCM87>java Armstrong
ENTER A NUMBER
153
153 IS AN ARMSTRONG NUMBER

C:\Users\user\Desktop\KCCM87>java Armstrong
ENTER A NUMBER
371
371 IS AN ARMSTRONG NUMBER

C:\Users\user\Desktop\KCCM87>java Armstrong
ENTER A NUMBER
690
690 IS NOT AN ARMSTRONG NUMBER

```

**/\* WAP TO ALL THE ARMSTONG NUMBER FROM 1 TO 1000 \*/**

```

import java.util.Scanner;

public class Armstrong1

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

        System.out.println("ENTER FROM NUMBER AND TO NUMBER");

        int n1=sc.nextInt(),n2=sc.nextInt();

        for(int i=n1;i<=n2;i++)

        {
            int arm=0,temp=i,rem;

            while(temp>0)

            {
                rem = temp%10;

```

```

temp = temp/10;

arm = arm+rem*rem*rem;

}

if(arm==i)

System.out.println(i+" IS AN ARMSTRONG NUMBER ");

}

}

}

```

```

C:\Users\user\Desktop\KCCM87>javac Armstrong1.java

C:\Users\user\Desktop\KCCM87>java Armstrong1
ENTER FROM NUMBER AND TO NUMBER
1
1000
1 IS AN ARMSTRONG NUMBER
153 IS AN ARMSTRONG NUMBER
370 IS AN ARMSTRONG NUMBER
371 IS AN ARMSTRONG NUMBER
407 IS AN ARMSTRONG NUMBER

```

**/\* WAP TO ALL THE PALINDROME NUMBER FROM 1 TO 1000 \*/**

```

import java.util.Scanner;

public class Palindrome1

{

    public static void main(String args[])

    {

        Scanner sc = new Scanner(System.in);

        System.out.println("ENTER A NUMBER");

        int n1=sc.nextInt(),n2=sc.nextInt();

        for(int i=n1;i<=n2;i++)

        {

```

```

int rev=0,rem,temp=i;

while(temp>0)

{
    rem = temp%10;

    temp = temp/10;

    rev = rev*10+rem;

}

if(rev==i)

System.out.println(i+" IS PALINDORME NUMBER");

}
}
}

```

```

818 IS PALINDORME NUMBER
828 IS PALINDORME NUMBER
838 IS PALINDORME NUMBER
848 IS PALINDORME NUMBER
858 IS PALINDORME NUMBER
868 IS PALINDORME NUMBER
878 IS PALINDORME NUMBER
888 IS PALINDORME NUMBER
898 IS PALINDORME NUMBER
909 IS PALINDORME NUMBER
919 IS PALINDORME NUMBER
929 IS PALINDORME NUMBER
939 IS PALINDORME NUMBER
949 IS PALINDORME NUMBER
959 IS PALINDORME NUMBER
969 IS PALINDORME NUMBER
979 IS PALINDORME NUMBER
989 IS PALINDORME NUMBER
999 IS PALINDORME NUMBER

```

**/\* WAP TO SUM OF THE AMSTRONG NUMBER FROM 1 TO 1000 \*/**

```

import java.util.Scanner;

public class Armstrong

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

```

```

Scanner sc = new Scanner(System.in);

System.out.println("ENTER FROM NUMBER AND TO NUMBER");

int n1=sc.nextInt(),n2=sc.nextInt(),sum=0;

for(int i=n1;i<=n2;i++)

{

    int arm=0,temp=i,rem;

    while(temp>0)

    {

        rem = temp%10;

        temp = temp/10;

        arm = arm+rem*rem*rem;

    }

    if(arm==i)

        sum = sum +i;

}

System.out.println(" sum of armstrong numbers:"+sum);

}
}

```

```

C:\Users\user\Desktop\New folder>javac Armstrong.java

C:\Users\user\Desktop\New folder>java Armstrong
ENTER FROM NUMBER AND TO NUMBER
1
1000
SUM OF ARMSTRONG NUMBERS:1302

```

**/\* WAP TO SUM OF THE PALINDROME NUMBER FROM 1 TO 1000 \*/**

```

import java.util.Scanner;

public class Palindrome

```

```
{  
    public static void main(String args[])  
    {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("ENTER A NUMBER");  
        int n1=sc.nextInt(),n2=sc.nextInt(),sum=0;  
        for(int i=n1;i<=n2;i++)  
        {  
            int rev=0,rem,temp=i;  
            while(temp>0)  
            {  
                rem = temp%10;  
                temp = temp/10;  
                rev = rev*10+rem;  
            }  
            if(rev==i)  
            sum=sum+i;  
        }  
        System.out.println("SUM OF PALINDORME NUMBERS :" +sum)  
    }  
}
```

```
C:\Users\user\Desktop\New folder>javac Palindrome.java  
C:\Users\user\Desktop\New folder>java Palindrome  
ENTER A NUMBER  
1  
1000  
SUM OF PALINDORME NUMBERS :50040
```

**/\* WAP TO PRINT SUM OF EVEN DIGITS FROM THE GIVEN NUMBER \*/**

**Ex:- IF THE INPUT IS ‘54321’ OUTPUT SHOULD BE ‘6’ LOGIC SHOULD BE ‘4+2’**

```
import java.util.Scanner;

public class Digits

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

        System.out.println("ENTER A NUMBER");

        int n=sc.nextInt(),rem,sum=0;

        while(n>0)

        {
            rem = n%10;

            n = n/10;

            if(rem%2==0)

            {
                sum=sum+rem;
            }
        }

        System.out.println("SUM OF EVEN DIGITS IS :" +sum);
    }
}
```

```
C:\Users\user\Desktop\KCCM87>javac Digits.java
```

```
C:\Users\user\Desktop\KCCM87>java Digits
```

```
ENTER A NUMBER
```

```
54321
```

```
SUM OF EVEN DIGITS IS :6
```

```
C:\Users\user\Desktop\KCCM87>java Digits
```

```
ENTER A NUMBER
```

```
123456789
```

```
SUM OF EVEN DIGITS IS :20
```

```
/* WAP TO PRINT SUM OF ODD DIGITS FROM THE GIVEN NUMBER */

import java.util.Scanner;

public class Digits1

{

    public static void main(String args[])

    {

        Scanner sc = new Scanner(System.in);

        System.out.println("ENTER A NUMBER");

        int n=sc.nextInt(),rem,sum=0;

        while(n>0)

        {

            rem = n%10;

            n = n/10;

            if(rem%2!=0)

            {

                sum=sum+rem;

            }

        }

        System.out.println("SUM OF ODD DIGITS IS :" +sum);

    }

}
```

```
C:\Users\user\Desktop\KCCM87>javac Digits1.java

C:\Users\user\Desktop\KCCM87>java Digits1
ENTER A NUMBER
54321
SUM OF ODD DIGITS IS :9

C:\Users\user\Desktop\KCCM87>java Digits1
ENTER A NUMBER
123456789
SUM OF ODD DIGITS IS :25
```

**/\* WAP BY TAKING INPUT FROM USER AND CHECK WHETHER THAT NUMBER IS PALINDROME OR NOT \*/**

- a) IF IT IS PALINDROME , FIND SUM OF EVEN DIGITS.
- b) IF IT IS NON-PALINDROME , FIND SUM OF ODD DIGIT.

```
import java.util.Scanner;

public class PEO

{
    public static void main(String args[])

    {
        Scanner sc = new Scanner(System.in);

        System.out.println("ENTER A NUMBER");

        int n=sc.nextInt(),rem,rev=0,temp=n,evensum=0,oddsum=0;

        while(temp>0)

        {
            rem = temp%10;

            temp = temp/10;

            rev = rev*10+rem;

            if(rem%2==0)

                evensum = evensum+rem;

            else

                oddsum = oddsum+rem;
        }

        if(rev==n)

        {
            System.out.println(n+" IS A PALINDROME NUMBER");

            System.out.println("SUM OF EVEN :"+evensum);
        }
    }
}
```

```

    }

else
{
    System.out.println(n+" IS NON PALINDROME NUMBER");

    System.out.println("SUM OF ODD :" +oddsum);

}
}

}

```

```

C:\Users\user\Desktop\KCCM87>javac PEO.java
C:\Users\user\Desktop\KCCM87>java PEO
ENTER A NUMBER
1221
1221 IS A PALINDROME NUMBER
SUM OF EVEN :4

C:\Users\user\Desktop\KCCM87>java PEO
ENTER A NUMBER
1234567
1234567 IS NON PALINDROME NUMBER
SUM OF ODD :19

C:\Users\user\Desktop\KCCM87>java PEO
ENTER A NUMBER
12621
12621 IS A PALINDROME NUMBER
SUM OF EVEN :10

```

**/\* WAP TO CHECK COUNT OF PRIME DIGITS FROM GIVEN NUMBER \*/**

**For Example: Input is 12345678**

**Output -----> Cout is 4**

```

import java.util.Scanner;

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

        System.out.println("ENTER A NUMBER");

```

```
int num=sc.nextInt(),rem,countp=0;  
  
while(num>0)  
{  
    rem = num%10;  
  
    num = num/10;  
  
    int count=0;  
  
    for(int i=1;i<=rem;i++)  
    {  
        if(rem%i==0)  
            count++;  
    }  
  
    if(count==2)  
        countp++;  
}  
  
System.out.println(" THE COUNT OF PD IS : "+countp);  
}  
}
```

```
C:\Users\user\Desktop\KCCM87>javac PD.java  
C:\Users\user\Desktop\KCCM87>java PD  
ENTER A NUMBER  
123456789  
THE COUNT OF PD IS : 4
```

## DIFFERENCE B/W FOR LOOP & WHILE LOOP

### FOR LOOP

#### 1. Syntax:

```
For(initialization;condition;updation)
{
    // body of the loop
    // statement to be executed
}
//statement outside the loop.
```

#### 2. We will go for ‘for loop’ when we

Knows exact number of iteration.  
iteration

#### 3. In‘forloop’ initialization,condition & updation is not mandatory.

#### 4. If we didn’t give the condition,compiler will by default condition if as true.

**Ex:** `for(; ;)`

```
{
    System.out.println("JAVA");
}
```

**Output:**

JAVA

JAVA

Infinite Times

### WHILE LOOP

#### 1. Syntax:

```
initialization
While(condition)
{
    //body of whlie loop
    updation
}
```

#### 2. We will go for ‘while loop’ when we don’t know exact number of iteration.

#### 3. In while loop,Initialization is not a compulsory,Condition is compulsory, Updation is not compulsory.

#### 4. If we didn’t give the condition,we will get compile time error.

**Ex:** `int i = 1;`

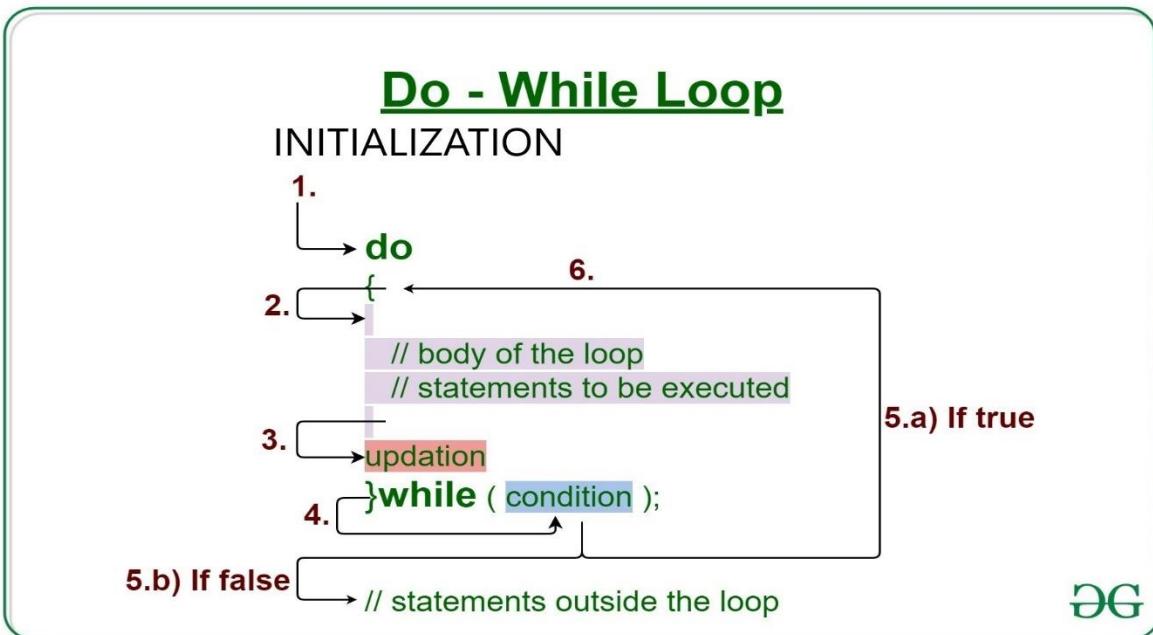
```
while()
{
    System.out.println("JAVA");
}
```

**Output:**

JAVA

Compile Time Error

## DO WHILE LOOP:



**NOTE:-** We will go for do while loop when we want our loop body to be executed at least once.

/\* WAP TO PRINT 1 TO 10 USING DO WHILE LOOP \*/

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

```
    }  
}  
}
```

## **COMPONENTS OF JDK:-**

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

- *It is a software which is responsible to develop and execute every java application.*

**JDK = JRE + DEV TOOLS**

### **JRE (JAVA RUN TIME ENVIRONMENT)**

- *It is a part of JDK which provide all the facilities to execute every java application.*

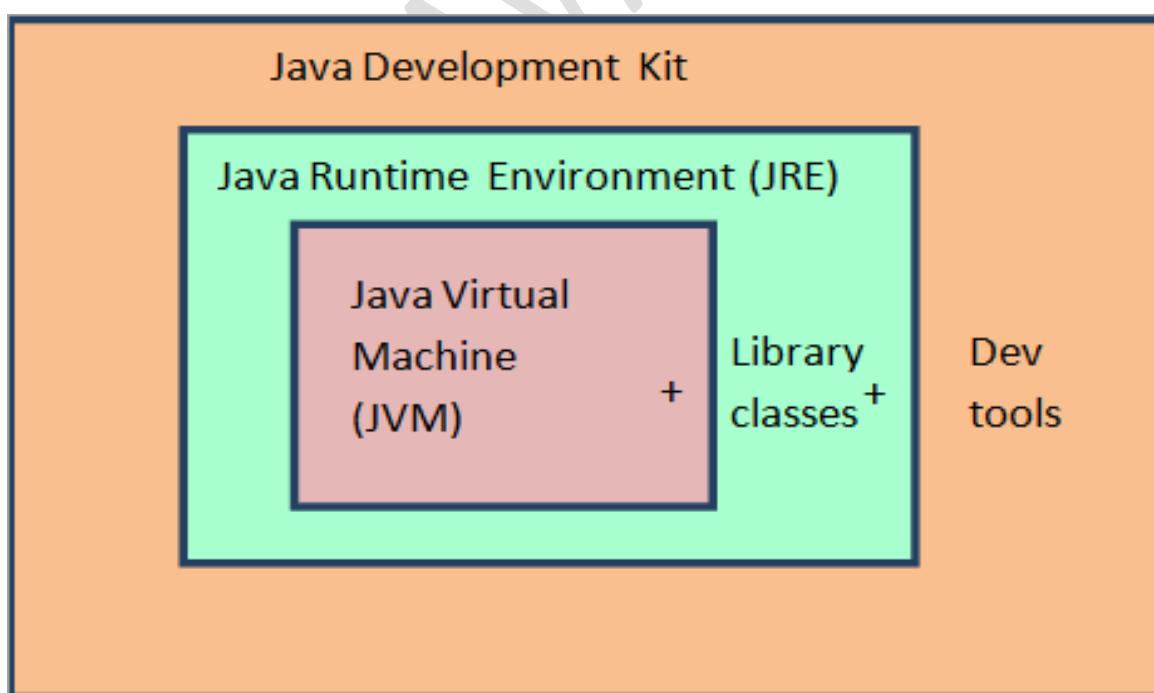
**JRE = JVM + PREDEFINE CLASSES / LIBRARY CLASSES**

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

- *It is a part of JRE, which is responsible to run every java application.*
- *Jvm having one inner thing – JIT.*

### **JIT (JAVA IN TIME)**

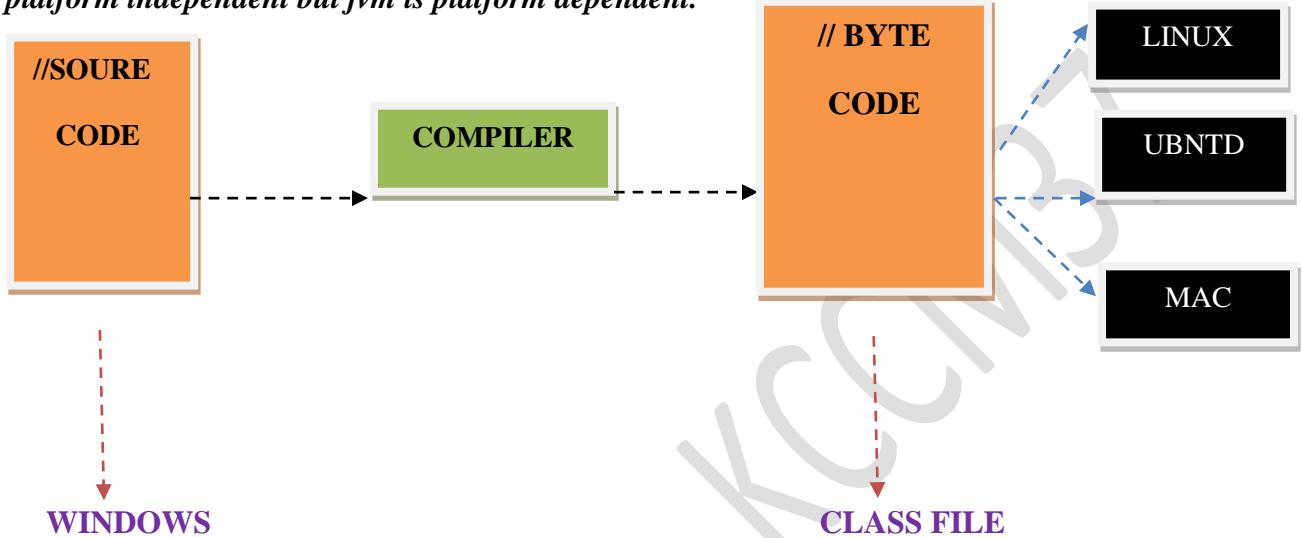
- *It is used to read one-one line of code and give it to operating system for execution*



## 1. IS JAVA AN PLATFORM INDEPENDENT LANGUAGE?

Yes ! Java is platform independent language i.e. program written under one operation system, can be run on any other operating system but to run them.

We need JVM of that particular operating system. Therefore, java language is platform independent but jvm is platform dependent.



### Ex:-1 - FIRST 50 PRIME NUMBERS

```
public class Prim3s
{
    public static void main(String args[])
    {
        int countp=0;
        for(int i=1;countp<50;i++)
        {
            int count=0;
            for(int j=1;j<=i;j++)
            {
                if(i%j==0)
                    count++;
            }
        }
    }
}
```

```

        }

    if(count==2)

    {

        countp++;

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

    }

}

}

}

```

```

E:\KCCM37>javac Prim3s.java
E:\KCCM37>java Prim3s
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
103
107
109
113
127
131
137
139
149
151
157
162

```

### **Ex:-2**

```

public class Prim2s

{

    public static void main(String args[])

    {

        int rem,count=0;

        for(int i=1;i<=100;i++)

```

```

{
    int temp=i;

    while(temp>0)
    {
        rem=temp%10;

        temp=temp/10;

        if(rem==2)

            count++;

    }

}

System.out.println("COUNT OF 2'S : "+count);

}
}

```

```

C:\Users\kssvi\Desktop\KCCM37>javac Prim2s.java

C:\Users\kssvi\Desktop\KCCM37>java Prim2s
COUNT OF 2'S : 20

```

### Ex:-3 /\* WAP TO PRINT FIRST 50 PALINDROME NUMBER \*/

```

public class Palindrome
{
    public static void main(String[] args)
    {
        int count=0;
        for (int i=1;count<50;i++)
        {
            int rev=0,rem,temp=i;
            while(temp>0)
            {
                rem = temp%10;
                temp = temp/10;
                rev = rev*10+rem;
            }
        }
    }
}

```

```
        if (rev==i)
    {
        count++;
        System.out.println(i+" IS A PALINDROME NUMBER");
    }
}
}
```

```
1 IS A PALINDROME NUMBER
2 IS A PALINDROME NUMBER
3 IS A PALINDROME NUMBER
4 IS A PALINDROME NUMBER
5 IS A PALINDROME NUMBER
6 IS A PALINDROME NUMBER
7 IS A PALINDROME NUMBER
8 IS A PALINDROME NUMBER
9 IS A PALINDROME NUMBER
11 IS A PALINDROME NUMBER
22 IS A PALINDROME NUMBER
33 IS A PALINDROME NUMBER
44 IS A PALINDROME NUMBER
55 IS A PALINDROME NUMBER
66 IS A PALINDROME NUMBER
77 IS A PALINDROME NUMBER
88 IS A PALINDROME NUMBER
99 IS A PALINDROME NUMBER
101 IS A PALINDROME NUMBER
111 IS A PALINDROME NUMBER
121 IS A PALINDROME NUMBER
131 IS A PALINDROME NUMBER
141 IS A PALINDROME NUMBER
151 IS A PALINDROME NUMBER
161 IS A PALINDROME NUMBER
171 IS A PALINDROME NUMBER
181 IS A PALINDROME NUMBER
191 IS A PALINDROME NUMBER
202 IS A PALINDROME NUMBER
212 IS A PALINDROME NUMBER
222 IS A PALINDROME NUMBER
232 IS A PALINDROME NUMBER
242 IS A PALINDROME NUMBER
252 IS A PALINDROME NUMBER
262 IS A PALINDROME NUMBER
272 IS A PALINDROME NUMBER
282 IS A PALINDROME NUMBER
292 IS A PALINDROME NUMBER
```

```
262 IS A PALINDROME NUMBER
272 IS A PALINDROME NUMBER
282 IS A PALINDROME NUMBER
292 IS A PALINDROME NUMBER
303 IS A PALINDROME NUMBER
313 IS A PALINDROME NUMBER
323 IS A PALINDROME NUMBER
333 IS A PALINDROME NUMBER
343 IS A PALINDROME NUMBER
353 IS A PALINDROME NUMBER
363 IS A PALINDROME NUMBER
373 IS A PALINDROME NUMBER
383 IS A PALINDROME NUMBER
393 IS A PALINDROME NUMBER
404 IS A PALINDROME NUMBER
414 IS A PALINDROME NUMBER
```

### **Differences between C and JAVA :**

<b>C</b>	<b>JAVA</b>
C is a Procedural Programming Language.	Java is an Object-Oriented Language.
C was developed by Dennis M.Ritchie in 1972.	Java language was developed by James Gosling in 1995.
In the C declaration variable are declared at the beginning of the block.	In Java, you can declare a variable anywhere.
C does not support multithreading.	Java has a feature of threading.
C supports pointers.	Java does not support pointers.
Memory allocation can be done by malloc.	Memory allocation can be done by new keyword.
Garbage collector needs to manage manually.	Garbage collector is managed automatically.
C does not have a feature of overloading functionality.	Java supports method overloading.
C is platform dependent language.	Java is platform independent language.
C is library Oriented language. #include<Stdio.h>	Java is package and class based language. package java.util;
Operating system is responsible to call main( ).	JVM is responsible to call main( ).

## **MODULE-2**

- **USER DEFINE METHODS**
- **METHOD OVER LOADING**
- **TYPES OF VARAIABLES**
  - Local, Static, Non-Static
- **EXECUTION PROCESS**
  - Multiple ways to access static
  - Accessing members of one class info another class
- **CONSTRUCIONS**
  - Types of constructions, This keyword, constructor overloading, constructor chaining.
- **UML**

### **USER DEFINE METHODS:-**

- *In java, we will never write a logic directly inside the class. In order to develop any logic, we will define main method then we will start developing the logic.*
- *In case, if the code is lengthly, it is recommended to create separate user define method and split the logic, which makes our program clear & more understandable.*

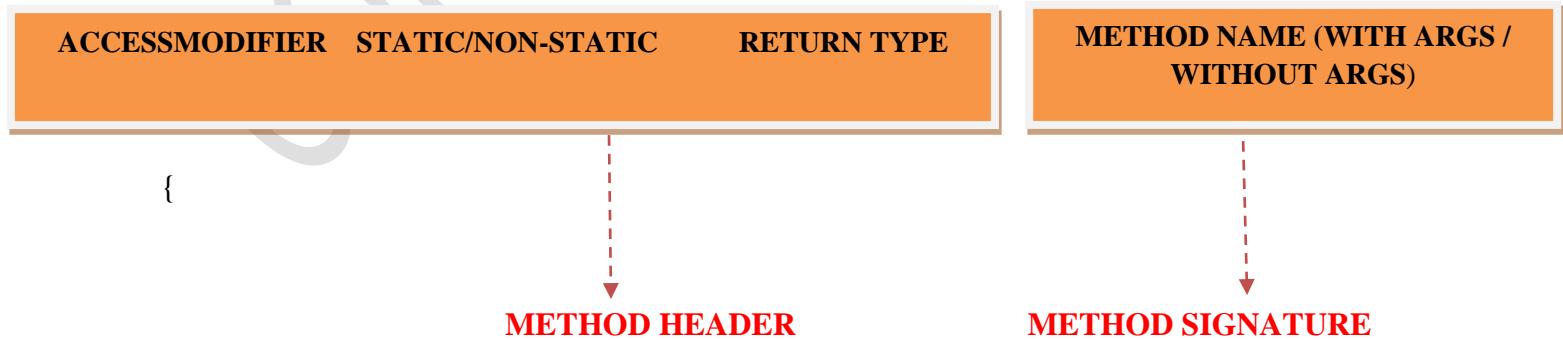
### **METHOD:-**

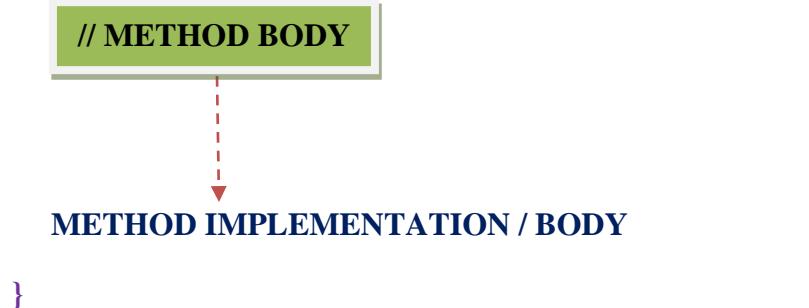
- *A method is define as a set of statement which is kept inside flower braces {}, which is having header, signature & body & which gets executed whenever we make a call to it.*

### **USER DEFINE METHODS HAS 3 JOB**

- *To break logic*
- *To make logic more clear*
- *Easy to understand*

### **SYNTAX OF CREATING USER DEFINE METHOD**





- ⊕ It is compulsory to call every user define method.
- ⊕ For calling purpose we use “Method Signature”.
- ⊕ In our program, we can create as many number method as we want.
- ⊕ We can not create me method inside another method.
- ⊕ A method can be develop once, and calls multiple times.

### **CREATING A USER-DEFINE METHOD WITHOUT ARGUMENTS:-**

```

public class Areas
{
    //user define method without args

    public static void areaCircle()
    {
        int r = 6;
        float areac = 3.14F*r*r;
        System.out.println("AREA OF CIRCLE IS : "+areac);
    }

    public static void areaRectangle()
    {
        int l=12,b=25,arear;
        arear = l*b;
        System.out.println("AREA OF RECTANGLE IS : "+arear);
    }

    public static void main(String args[])
}

```

```

{
    areaCircle();
    areaRectangle();
}
}

```

```

C:\Users\user\Desktop\KCCM37>javac Areas.java
C:\Users\user\Desktop\KCCM37>java Areas
AREA OF CIRCLE IS : 113.04
AREA OF RECTANGLE IS : 300

```

**NOTE:-** Using method without arguments we can write a method once & call it multiple times but the disadvantage is every time we call execution will happen in same values to over this we can, use method with argument.

- We can call user define method -----> in another User define method, like

```

public class Areas2
{
    public static void areaCircle()
    {
        areaRectangle();
        int r = 6;
        float areac = 3.14F*r*r;
        System.out.println("AREA OF CIRCLE IS : "+areac);
    }

    public static void areaRectangle()
    {
        int l=12,b=25,arear;
        arear = l*b;
    }
}

```

```

        System.out.println("AREA OF RECTANGLE IS : "+arear);
    }

    public static void main(String args[])
    {
        areaCircle();
    }

}

```

```

C:\Users\user\Desktop\KCCM37>javac Areas2.java
C:\Users\user\Desktop\KCCM37>java Areas2
AREA OF RECTANGLE IS : 300
AREA OF CIRCLE IS : 113.04

```

### **METHOD WITH SOME ARGUMENTS:-**

- *Arguments are define as input given to any method.*

#### **Syntax:**

METHODHEADER	METHODNAME(DATATYPE VAR, DATATYPE VAR.....DATATYPE VAR)
--------------	--

**Ex:- public static void add(int i, char c, String d)**

- *A method can have any number of inputs in the form of arguments.*
- *When we call any method with argument we have to pass that particular type of value.*
- *While passing arguments we have to make sure it will be as per sequence define in method declaration.*

```

public class Areas1
{
    public static void main(String args[])
    {
        areaCircle (7); areaCircle(24); areaCircle(67);

        areaRectangle(10,20); areaRectangle(12,24); areaRectangle(20,60);
    }
}
```

```

}

public static void areaCircle(int r)
{
    float areac = 3.14F*r*r;
    System.out.println("AREA OF CIRCLE IS : "+areac);
}

public static void areaRectangle(int l,int b)
{
    int arear;
    arear = l*b;
    System.out.println("AREA OF RECTANGLE IS : "+arear);
}
}

```

```

C:\Users\user\Desktop\KCCM37>java Areas1
AREA OF CIRCLE IS : 153.86002
AREA OF CIRCLE IS : 1808.64
AREA OF CIRCLE IS : 14095.46
AREA OF RECTANGLE IS : 200
AREA OF RECTANGLE IS : 288
AREA OF RECTANGLE IS : 1200

```

```

/* WAP TO SWAP TWO NUMBERS USING USER DEFINE METHOD WITH
ARGUMENTS */

public class Swap
{
    public static void main(String args[])
    {
        swap(10,20);swap(35,67);
    }
}
```

```

public static void swap(int a,int b)
{
    int c;
    c = a;
    a = b;
    b = c;

    System.out.println("AFTER SWAPPED VALUES ARE : "+a+" "+b);
}
}

```

```

C:\Users\user\Desktop\KCCM37>javac Swap.java
C:\Users\user\Desktop\KCCM37>java Swap
AFTER SWAPPED VALUES ARE : 20 10
AFTER SWAPPED VALUES ARE : 67 35

```

**/\* WAP TO CALCULATE FACTORIAL OF A NUMBER USING METHOD WITH ARGUMENT \*/**

```

public class Factorial
{
    public static void main(String args[])
    {
        factorial(2);
        factorial(8);
        factorial(9);
    }

    // user define method without args

    public static void factorial(int num)
    {

```

```

int fact = 1;

for(int i=num;i>=1;i--)
{
    fact = fact*i;
}

System.out.println("FACTORIAL OF A NUMBER IS : "+fact);
}
}

```

```

C:\Users\user\Desktop\KCCM37>javac Factorial.java

C:\Users\user\Desktop\KCCM37>java Factorial
FACTORIAL OF A NUMBER IS : 2
FACTORIAL OF A NUMBER IS : 40320
FACTORIAL OF A NUMBER IS : 362880

```

### **METHOD OVER LOADING:-**

- + *The process of developing multiple methods with same name but different arguments list is called as method over loading.*

### **RULES FOR DEFINING ARGUMENT LIST:**

- + **NO :: OF ARGUMENT MUST BE DIFFERENT**

Ex:- Swiggy()  
 Swiggy(String Name)  
 Swiggy(String Name, Int order id)

- + **TYPES (DATA TYPE) OF ARGUMENT MUST BE DIFFERENT**

Ex:- Add(int i, int j)  
 Add(Double d, Double D1)  
 Add(String S1, String S2)

- + **SEQUENCE (OR) POSITION OF ARGUMENT MUST BE DIFFERENT**

Ex:- log in (String url, int id)  
 log in (int id, String pwd)

**WE GO FOR METHOD OVER LOADING WHEN WE WANT TO PERFORM ONE TASK IN MULTIPLE WAYS:**

**EXAMPLES:**

 **NON-TECHNICAL**

Travelling ----- Hyd --- Bang

Bus

Train

Aeroplane

Bike

Car

 **KEEPING PHONE PASSWORD**

Number Lock

Drawing Pattern

Finger Print

Face Sensor

 **TECHNICAL EXAMPLE:**

-Payment in any Ecommerce

Credit

Debit

Wallets ( G-pay, P-pay, Amazon pay, Paytm)

Cash on Delivery

 **EXAMPLE -2**

-Banking

Online Banking

Physical Banking

ATM Banking

App Banking

## 1.CREATE A CLASS AS PAYMENT APP

## 2.CREATE AN OVER LOADED METHOD AS PAYMENT

Payment(String wallettype, String uid)

----- Print wallettype, uid -----

Payment(String cardtype, Long cardno, String expdate, int cvv)

----- Print cardtype, cardno, expdate, cvv -----

Payment(Long accno, String uname, String pwd)

----- Payment accno, uname, pwd ----- (related online baking)

## 3.CREATE MAIN () & MAKE A CALL TO ALL 3 PAYMENT METHOD'S

```
public class PaymentApp
```

```
{
```

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

```
{
```

```
        payment("GOOGLE PAY","vignesh@sbi");
```

```
        payment("CREDIT",62341205678L,"JAN-98",333);
```

```
        payment(38908235692L,"vignesh chowadary","VIGNE1234");
```

```
}
```

```
    public static void payment(String wallettype, String uid)
```

```
{
```

```
        System.out.println("WALLET TYPE:" + wallettype + "\nuid: " + uid);
```

```
}
```

```
    public static void payment(String cardtype, long cardno, String expdate, int cvv)
```

```
{
```

```

        System.out.println("CARD TPYE : " +cardtype +"\n card no: "+cardno+"\nexpdate:
"+expdate+"\ncvv:"+cvv);

    }

public static void payment(long accno,String uname,String pwd)

{
    System.out.println("ACC NO:"+accno+"\nuname:"+uname+"\npwd:"+pwd);

}
}

```

```

C:\Users\user\Desktop\KCCM37>javac PaymentApp.java

C:\Users\user\Desktop\KCCM37>java PaymentApp
WALLET TYPE:GOOGLE PAY
uid: vignesh@sbi
CARD TPYE :CREDIT
cardno:62341205678
expdate:JAN-98
cvv:333
ACC NO:38908235692
uname:vignesh
pwd:VIGNE1234

```

## 1.CREATE A CLASS AS TRAVELLING APP

## 2.CREATE AN OVER LOADED METHOD AS

**Travel (String Bustype, String Source, String Destination)**

----- Print Values -----

**Travel (String Flighttype, int Price, String Source, String Dest)**

----- Print Values -----

**Travel (String Traintype, int Price, int Platform, String Source, String Dest)**

----- Print Values -----

## 3.CREATE MAIN () & MAKE CALL TO ALL 3 TRAVEL'S.

```
public class TravelApp
```

```
{
```

```

public static void main(String args[])
{
    travel("AMARAVATHI A/C","VIJ","HYD");
    travel("OMANAIR",50000,"HYD-AIRPORT","MCT-AIRPORT");
    travel("DURONTO SPL",2000,1," MAS ","VIJ");
}

public static void travel(String bustype,String source, String destination)
{
    System.out.println("BUS TYPE:"+bustype+"\nsource: "+source+"\ndestination:
"+destination);
}

public static void travel(String flighttype,int price,String source,String dest)
{
    System.out.println("FLIGHT TPYE : "+flighttype+ "\nprice:"+price+"\nsource:
"+source+"\ndest:"+dest);
}

public static void travel(String traitype,int price,int platform,String source,String dest)
{
    System.out.println("TRAIN TYPE:"+traitype+ " \nprice:"+price+" \nplatform:
"+platform+"\nsource:"+source+"\ndest:"+dest);
}

```

```

C:\Users\user\Desktop\KCCM37>javac TravelApp.java
C:\Users\user\Desktop\KCCM37>java TravelApp
BUS TYPE:AMARAVATHI A/C
source: VIJ
destination: HYD
FLIGHT TPYE :OMANAIR
price:50000
source:HYD-AIRPORT
det:MCT-AIRPORT
TRAIN TYPE:DURONTO SPL
price:2000
platform:1
source: MAS
dest:VIJ

```

## **TYPES OF VARIABLES:**

- ⊕ Based on the place of declaration, variables are divided into 2 types

### **1. LOCAL VARIABLES**

### **2. GLOBAL VARIABLES:**

Global variables are again divided into two types:

- ⊕ STATIC VARIABLES
- ⊕ NON-STATIC VARIABLES

## **LOCAL VARIABLES:**

- ⊕ Variable which are declared inside a method and within the class, such variables are called as **Local variables**.
- ⊕ Local variables are accessible only inside a method in which they are present. If we try to access them outside of the method, we will get the **compilation error (compile time error)**
- ⊕ It is compulsory to initialise local variables before we use them.
- ⊕ All the local variable are present in '**Stack Memory Area**'.

## **GLOBAL VARIABLES:**

- ⊕ Variables which are declared outside a method and within the class, such variables are called **Global variables**.
- ⊕ Global variable are accessible anywhere within the scope of a class.
- ⊕ It is not compulsory to initialise global variables.
- ⊕ If we did not initialize, compiler, will consider default values based on datatype.
- ⊕ Global variables are present in '**Heap Memory Area**'.
- ⊕ Global variable are divided into two types

### **STATIC VARIABLES**

### **NON STATIC VARIABLES**

DATATYPE	DEFVALUE
BYTE,SHORT,INT,LONG	0
FLOAT,DOUBLE	0.0
CHAR	EMPTY SPACE
BOOLEAN	FALSE
STRING	NULL

## DIFFERENCE B/W LOCAL & GLOBAL VARIABLES

<b>LOCAL VARIABLES</b>	<b>GLOBAL VARIABLES</b>
<ol style="list-style-type: none"><li>1. Local variable are those, which are declared inside a method and within the class</li><li>2. Local variable are accessible only inside a method, in which they are present.</li><li>3. It is compulsory to initialise local variables before we use them.</li><li>4. There is no types in local variables</li><li>5. Local variables are present in Stack Memory Area.</li></ol>	<ol style="list-style-type: none"><li>1. Global variables are declared outside method and with in a class.</li><li>2. Global variables are accessible anywhere within the scop of a class.</li><li>3. It is not compulsory to initialise global variables.</li><li>4. There are 2 types in global variables 1.Static Variable 2.Non Static Variables</li><li>5. Global variables are present in Heap Memory Area.</li></ol>

### **STATIC VARIABLES:-**

- + A *variables which is present outside a method and with in the class and having static keyword such variables is Static Variables.*

**Syntax:**

STATIC DATATYPE VAR = VALUE;

### **1. CREATE A CLASS AS SHOPPING**

### **2. DECLARE SOME STATIC VARIABLE AS**

- + STORENAME
- + PRODUCT NAME
- + BRAND
- + COLOR
- + PRICE

### **3.CREATE A STATIC METHOD AS SHOPPING DETAILS () & PRINT ALL**

**STATIC VARIABLE.**

#### **4.CREATE MAIN () & MAKE A CALL TO SHOPPING DETAILS ()**

```
public class Shopping
{
    static String Storename = "APTRONIX";
    static String Productname = "IPHONE-14PLUS";
    static String Brand = "APPLE";
    static String Color = "RED";
    static int price = 75000;

    public static void main(String args[])
    {
        ShoppingDetails();
    }

    public static void ShoppingDetails()
    {
        System.out.println("STORE NAME: "+Storename+"\nproduct name: "+Productname
        +"\nbrand: "+Brand+"\ncolor: "+Color+"\nprice: "+price);
    }
}
```

```
C:\Users\user\Desktop\KCCM37>javac Shopping.java
C:\Users\user\Desktop\KCCM37>java Shopping
STORE NAME: APTRONIX
product name: IPHONE-14PLUS
brand: APPLE
color: RED
price: 75000
```

#### **1.CREATE A CLASS AS BOOK STORE**

#### **2.DECLEARE & INITIALSE STATIC VARIABLE AS**

⊕  **STORENAME**

- BOOKNAME**
- AUTHOR**
- PAGES**
- COLOR**
- PUBLISHER**
- PRICE**

**3.CREATE A STATIC METHOD AS BOOK DETAILS () & PRINT ALL STATIC VARIABLES HERE**

**4.CREATE MAIN () & MAKE A ALL TO BOOK DETAILS ()**

```
public class BookStore
{
    static String Storename = "LIBRARY";
    static String Bookname = "DMM-2";
    static String Author = "R.S.KHURMI";
    static String pages = "5000";
    static String Color = "BLUE";
    static String Publisher = "RAM NAGAR,NEW DELHI-110 055";
    static int Price = 670;

    public static void main(String args[])
    {
        BookStore();
    }

    public static void BookStore()
    {
        System.out.println("STORENAME: "+Storename+"\nbook name: "+Bookname+
                           "\nauthor:"+Author+"\npages: "+pages+"\ncolor: "+Color+"\npublisher: "+Publisher);
    }
}
```

```
c:\Users\user\Desktop\KCCM37>javac BookStore.java  
c:\Users\user\Desktop\KCCM37>java BookStore  
STORENAME: LIBRARY  
book name: DMM-2  
author: R.S.KHURMI  
pages: 5000  
color: BLUE  
publisher: RAM NAGAR,NEW DELHI-110 055
```

### NON STATIC VARIABLE:-

- ⊕ A variable which is present outside method with in the class & does't have static keyword, such variable are called as **Non Static Variable**.

**Syntax:**

DATATYPE VAR = VALUE;

### NON STATIC METHOD:-

- ⊕ If a method does not have static keyword such method are called as **Non Static Method**.

### NOTE:

- ⊕ Non static variables can not be accessible directly inside a static method, if we want them to access we have to create an object.
- ⊕ Non static method can not be call directly inside a static method, if we want them to call we have to create an object.

```
public class Addition  
{  
    int i = 100, j = 300; // non static variables  
  
    public static void Addition()  
    {  
        System.out.println(i+j); // (CTE as i and j cannot be referred directly inside static)  
    }  
  
    public static void main(String args[])  
    {
```

```
    Addition();  
}  
}
```

### SYNTAX OF OBJECT CREATION:-

CLASSNAME REFERENCE VARIABLE = NEW KEYWORD CLASS NAME ();

(OR)

CLASSNAME REFERENCE VARIABLE = NEW KEYWORD CONSTRUCTOR ();

- + *New is a keyword which creates an objects.*
- + *New keyword will loads all non static members ( non static method & non static variables inside an object)*
- + *After loading, reference variable will be assigned to that object.*
- + *Using reference variables & dot operator we can access all non static member present inside that object.*

### ACCESSING NON-STATIC VARIABLE INSIDE STATIC METHOD BY CREATING AN OBJECT

```
public class Addition  
{  
    int i = 100, j = 300; // non static variables  
    public static void Addition()  
    {  
        Addition A1 = new Addition();  
        System.out.println(A1.i+A1.j);  
    }  
    public static void main(String args[])  
    {  
        Addition();  
    }  
}
```

```
 }  
 }
```

## CALLING NON STATIC METHOD INSIDE STATIC METHOD THROUGH OBJECT

```
public class Addition  
{  
    public void Addition() // non static method  
    {  
        System.out.println("Addition Method")  
    }  
  
    public static void main(String args[])  
    {  
        Addition A1 = new Addition();  
        A1.Addition();  
    }  
}
```

```
C:\Users\user\Desktop\KCCM37>javac Addition.java  
C:\Users\user\Desktop\KCCM37>java Addition  
Addition Method
```

## CONCLUSION POINT:

### SITUATIONS WHERE WE NEED TO CREATE AN OBJECT:

- ✚ *Non static variables inside static method*
- ✚ *Non static method inside static method*

### SITUATIONS WHERE WE NEED NOT TO CREATE AN OBJECT

- ✚ *Static variable inside static method*
- ✚ *Static method inside static method*
- ✚ *Static variable inside non static method*
- ✚ *Non static method inside non static method*

- Non static variable inside non static method
- Static method inside non static method

**/\*WAP TO PRINT ADDITION USING STATIC VARIABLES & NON STATIC VARIABLES THROUGH OBJECT CREATION \*/**

```
public class Addition
{
    static int i = 100; // Static Variables
    int j = 200; // Non Static Variables
    public static void Addition() // Static Variabes
    {
        Addition A1 = new Addition(); // Non Static Var Inside Static Method Access Through Object
        System.out.println(i+A1.j); // Static & Non Static Inside Static
    }
    public void Subtraction() //Non Static Method
    {
        System.out.println(i-j); // Static & Non Static Inside Non Static
    }
    public static void main(String args[])
    {
        Addition();
        Addition A2 = new Addition();
        A2.Subtraction(); // Non Static Method Inside Static Call Through Object
    }
}
```

## OUTPUT

-100

**1.CREATE AN FOODORDER APP**

**2.DECLARE & INITIALISE SOME STATIC VARIABLES**

- Restaurant\_Name
- No\_of\_dishes\_order
- Price

**3.DECLARE & INITIALISE SOME NON STATIC VARIABES**

- Order\_id
- Discount\_amt

**4.CREATE A STATIC METHOD AS ORDER\_DETAILS ()**

**& PRINT ALL STATIC & NON STATIC VARIABLES**

**5.CREATE AN NON STATIC METHOD AS DELIVERY\_PERSON ( STRING ADDRESS, INT WAIT\_CHARGES)**

**AND PRNIT ADDRESS & WAIT\_CHARGES**

**AND MAKE A CALL TO ORDER\_DETILS ()**

**6.CREATE AN NON STATIC METHOD AS ZOMATO\_PARTNER ()**

**DECLARE AND INITIALISE SOME LOCAL VARIABLES AS**

**Food\_rating**

**Delivery\_rating**

**Hygenic\_rating**

**PRINT THESE LOCAL VARIABLES**

**MAKE A CALL TO DELIVERY\_PERSON METHOD**

**7.CREATE A MAIN () AND MAKE A CALL TO ZOMATO\_PATNER ()**

**public class FoodOrderApp**

**{**

**static String Restaurant\_Name = "NAIDU GARI KUNDA BIRYANI";**

**static int No\_of\_dishes\_order = 25;**

```
static int price = 20000;  
int order_id = 234518990;  
int discount_amt = 200;  
public static void OrderDetails()  
{  
    FoodOrderApp A1 = new FoodOrderApp();  
  
    System.out.println("Restaurant_Name:" + Restaurant_Name + "\nno_of_dishes_order: " +  
        No_of_dishes_order + "\nprice:" + price + " \norder_id:" + A1.order_id + " \ndiscount_amt:  
    "+A1.discount_amt);  
}  
  
public void delivery_person(String address,int wait_charges)  
{  
    System.out.println("Address: " + address + "\nwait_charges: " + wait_charges);  
    OrderDetails();  
}  
  
public void Zomto_panter()  
{  
    int food_rating = 3;  
    int delivery_rating = 5;  
    int hygenic_rating = 3;  
  
    System.out.println("food_rating: " + food_rating + "\ndelivery_rating: " + delivery_rating + "  
\nhygenic_rating: " + hygenic_rating);  
  
    delivery_person("VIJAYAWADA",50);  
}  
  
public static void main(String args[])  
{
```

```

FoodOrderApp A2 = new FoodOrderApp();

A2.Zomto_panter();

}

}

```

```

C:\Users\user\Desktop\KCCM37>javac FoodOrderApp.java

C:\Users\user\Desktop\KCCM37>java FoodOrderApp
food_rating: 3
delivery_rating: 5
hygenic_rating: 3
Address: VIJAYAWADA
wait_charges: 50
Restaurant_Name:NAIDU GARI KUNDA BIRYANI
no_of_dishes_order: 25
price:20000
order_id:234518990
discount_amt:200

```

## 1.CREATE AN GROCERY\_APP

## 2.DECLARE & INITIALISE SOME STATIC VARIABLES

- ✚ Item\_name
- ✚ Brand\_name

## 3.DECLARE & INITIALISE SOME NON STATIC VARIABLES

- ✚ Amount
- ✚ Quantity

## 4.CREATE A STATIC METHOD AS CUSTOMER\_ORDER()

& PRINT ALL DETAILS

## 5.CREATE A NON STATIC METHOD AS DELIVERY\_PARTNER()

& CREATE SOME LOCAL VARIABLES AS

Mask\_charges

Sanitise\_charges

PRINT ALL LOCAL VARIABLES VALUES.

## 6.CREATE MAIN () AND MAKE A CALL TO CUSTOMER\_ORDER ()

```
public class Grocery_App
{
    static String item_name = "BISCUIT";
    static String brand_name = "BRITANNIA";
    int Amount = 2000;
    int Quantity = 10;
    public static void customer_order()
    {
        System.out.println("ORDER DETAILS :");
        Grocery_App A1 = new Grocery_App();
        System.out.println("item_name:" + item_name + "\nbrand_name:" + brand_name + "\nAmount:" + A1.Amount + "\nQuantity:" + A1.Quantity);
    }
    public void delivery_panter()
    {
        int mask_charges = 245;
        int scanitise_charges = 65;
        System.out.println("mask_charges: " + mask_charges + "\nscanitise_charges: " + scanitise_charges);
    }
    public static void main(String args[])
    {
        customer_order();
        Grocery_App A2 = new Grocery_App();
        A2.delivery_panter();
    }
}
```

}

```
C:\Users\user\Desktop\KCCM37>javac Grocery_App.java
C:\Users\user\Desktop\KCCM37>java Grocery_App
ORDER DETAILS :
item_name:BISCUIT
brand_name:BRITANNIA
Amount:2000
Quantity:10
mask_charges: 245
sanitise_cahrges: 65
```

### **EXECUTION PROCESS:-**

-Whenever we execute any program there will be two memory blocks that gets created.

1. Stack area also called as *Execution Area*.

2. Heap area also called as *Storage Area*.

### **POINTS:-**

1. First JVM will enter into stack area and make a call to class Loader.

2. Class loader is like a developing tool whose job is to load all static members into *static pool area (SPA)*.

3. SPA is a part of heap area and its name is same as class name.

4. Once class loader loads all static members into SPA its job is completed so it will come out of stack area.

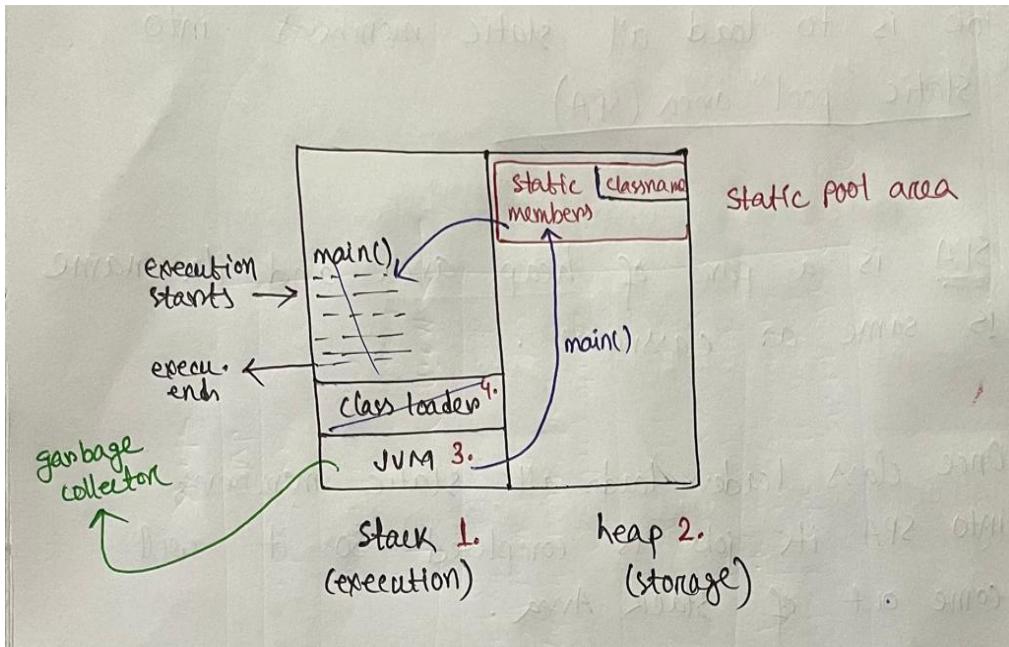
5. Next JVM will make a call to main method.

6. Upon call to anymethod, that method will get loaded in stack area.

7. Next Execution of main method will start

8. Once entire execution is completed main( ) also come out of stack area

9. Next JVM before coming out of stack area it will make a call to garbage collector whose job is to cleanup entire memory for next execution.



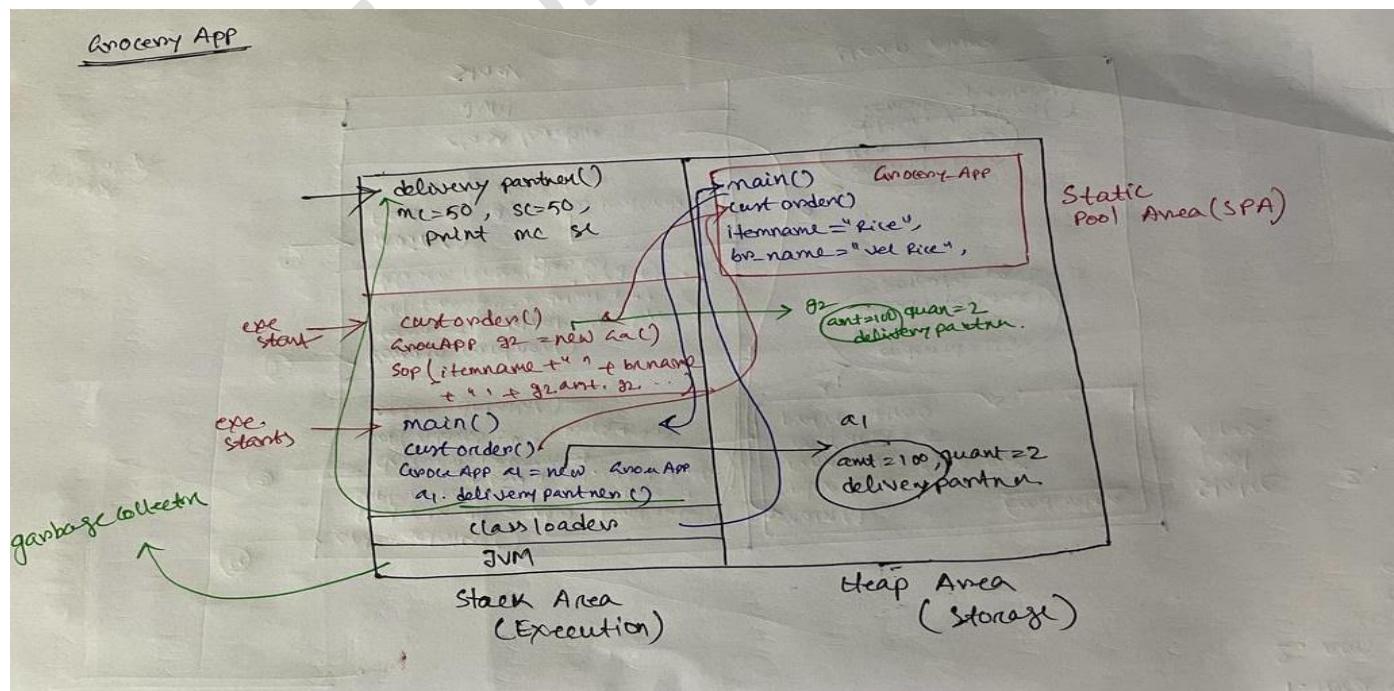
## GARBAGE COLLECTOR:

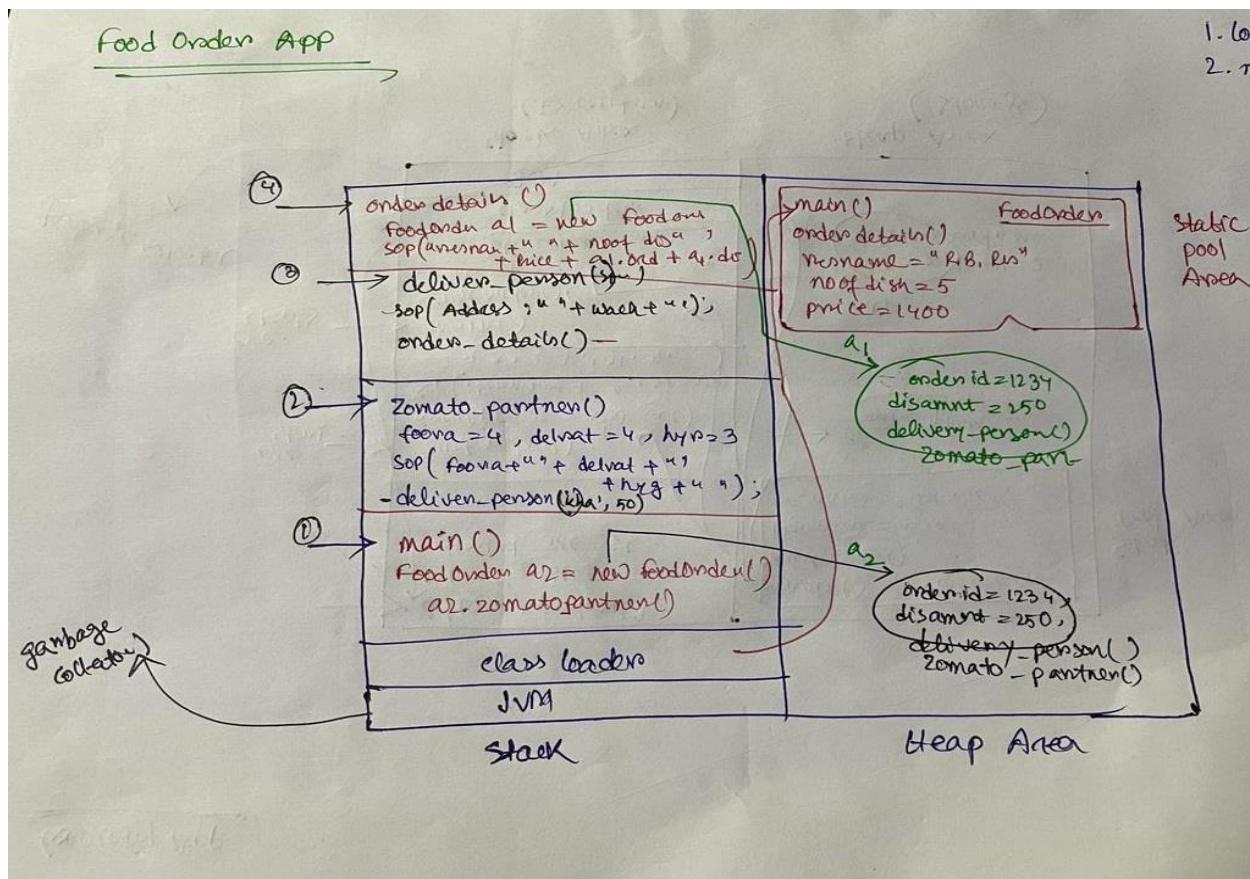
- Is a pre-defined method of system class JVM will call it internally.

**Syntax:** PUBLIC STATIC VOID GC();

- In case if we want to call garbage collector we can call it with the syntax as

SYSTEM.GC()





## CONCLUSION:-

- ⊕ Static methods & static variables are available (SPA) before execution starts that's why they are accessible without object creation.
- ⊕ Non static members are not available before execution so to access them we have to create an object. With new keyword object will get created & loads all non static members inside it.
- ⊕ All static members together present at one place and all non static members together present at one place so we can access directly.
- ⊕ Static member will get loaded only once in SPA that's why they are referred as single copy.
- ⊕ Since non static member will get loaded whenever we created an object that's why they referred as multiple copies.
- ⊕ Local variables are present in stack area & global variables present in heap area.
- ⊕ Static variables are present in SPA(Heap)
- ⊕ Non static variables are present in object(Heap)

## DIFFERENCE B/W STATIC & NON-STATIC

### **STATIC**

- 1.Static refers to single copy
- 2.Static members will be loaded only once in static pool area.
- 3.Static member are present in static pool area.
- 4.Class loader is responsible to load static members.
- 5.Static member can be accessed in 3 ways
  - ✚ Directly
  - ✚ Through object
  - ✚ Through class name
- 6.If I change the value of static variable, it will affect entire class that's why static variables are also called as class variables.
- 7.We will go for static, if the data is fixed.  
**Ex: PAN NO, PASSPORT NO**
- 8.Static members can not be inherited.
- 9.Static members can not be overridden.

### **NON STATIC**

- 1.Non static refers to multiple copies
- 2.Non static members will be loaded whenever we create an object.
- 3.Non static member are present inside an object.
- 4.New keyword is responsible to load non static member.
- 5.Non static can be accessed through object.
- 6.If I change the value of non static variable, it will affect only one object, that's why non static variable is also called as Instance variables.
- 7.We will go for non static, if the data is varying.  
**Ex: EMAIL ID (CHANGING)**
- 8.Non static members can be inherited.
9. Non static members can be overridden.

## **MULTIPLE WAYS TO ACCESS STATIC MEMBERS.**

- ➡ Static members can be accessed in 3 ways

### **DIRECTLY:-**

Because they will get loaded only once in static pool area (SPA).

### **THROUGH CLASS NAME:-**

Because class name and static pool area name are same.

### **THROUGH OBJECT:-**

Because there is a one way connection b/w object & static pool area (SPA).

**class A**

```
{  
    static int i = 100;  
  
    public static void main (String args[])  
    {  
        System.out.println(i);  
  
        System.out.println(A.i);  
  
        A a1 = new A();  
  
        System.out.println(a1.i);  
    }  
}
```

### **WORKING WITH ECLIPSE:**

- ➡ Install eclipse from google
- ➡ Click on icon & wait until it launches
- ➡ Provide workspace (place to store program)

### **STEP-1 CREACTION OF PROJECT:-**

**1. Click on file -> new -> java project -> provide project name -> click on finish -> click on don't create for module -> we can see one project folder is created left side.**

## STEP-2 CREACTION OF PACKAGE:-

Package → Is used to keep all classes together at one place

1.Click on file → new → package name → provide package name → verify source folder name → click on finish → we can see that under project folder one pakage folder is created.

## STEP-3 CREACTION OF CLASS :-

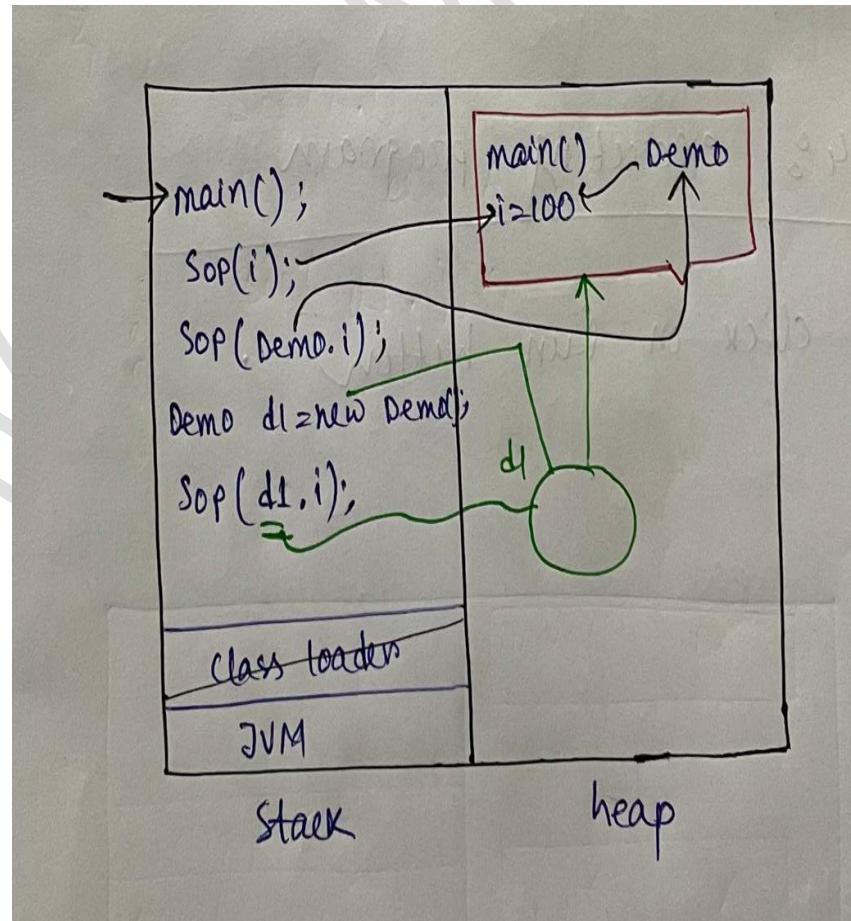
1.Click on file → new → class → provide class name → verify source folder name & package name → select checkbox for main method → click on finish → we can see that a class is created to develop logic.

## STEP-4 EXECUTION PROGRAM :-

Click on Run Button

Ex:-1

```
class Demo
{
    static int i = 100;
    public static void main (String args[])
    {
        System.out.println(i);
        System.out.println(Demo.i);
        Demo d1 = new Demo();
        System.out.println(d1.i);
    }
}
```



## ACCESSING MEMBERS OF ONE CLASS INTO ANOTHER CLASS:-

- We can access both static as well as non static members of one class into another class.
- Static members of one class can be accessed in another class through class name.

- Non static members of one class can be accessed in another class by *creating an object*.
- When we have more than one class, we have consider following points.
  - Only one class can have main method.*
  - File name must be same as class name of main method.*
  - The class which is having main method, only that class should be public.*

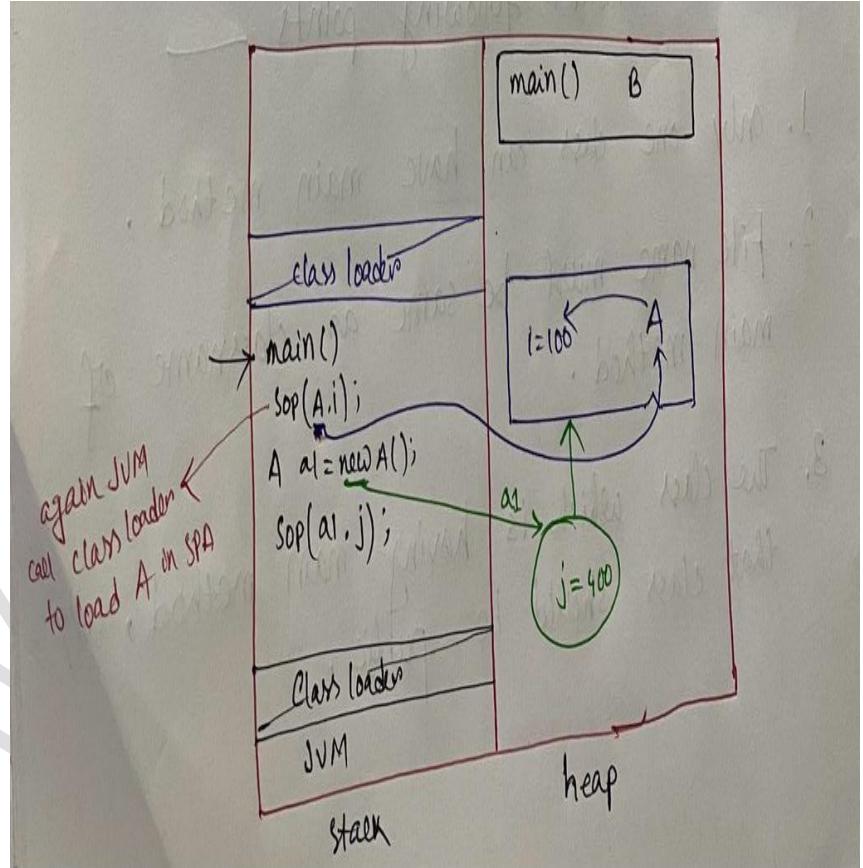
**Ex:2:-**

```

class A
{
    static int i = 100;
    int j = 400;
}

public class B
{
    public static void main (String args[])
    {
        System.out.println(A.i);
        A a1 = new A();
        System.out.println(a1.j);
    }
}

```



- In above program, classs loader will first load static member of 'B' class, because it contains main method & execution start with main method.
- Once, JVM notice 'A' class, again it makes a call to class loader & this time class loader will load static member of A class.

## NEED OF CONSTRUCTOR:

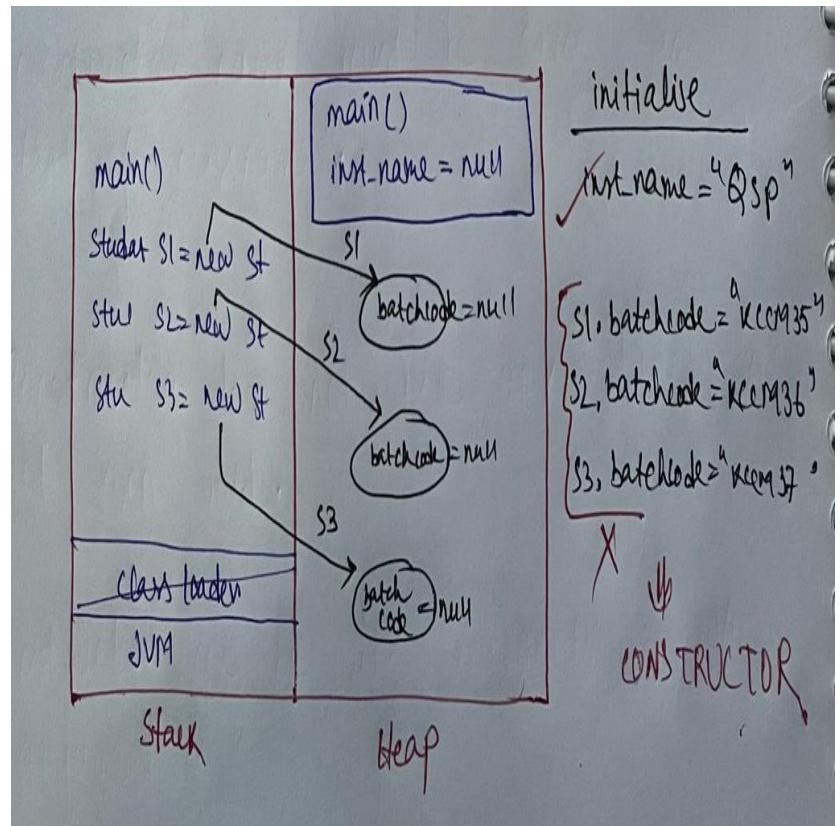
- Avoiding → For initialisation of non static variables in multiple times (through object creation), we need constructor.

```

public class student
{
    static String inst_name;
    String batchcode;

    public static void main(String args[])
    {
        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();
    }
}

```



- From the above memory diagram, we can observe that static variable have single memory. Where as, non static variable have separate memory for each object which means, we can initialise single value for static variable but multiple values for non static variable.
- Initialising separate value for each object is a greedy approach. Therefore to overcome this java has introduce constructors.

## CONSTRUCTOR:

### DEFINITION:

- Constructor is a special type of method which gets executed whenever we created an object.
- The main purpose of constructor is to initialise a non static variable.

### Syntax:

ACCESSMODIFIER CONSTRUCTOR NAME (WITH ARGUMENTS / WITH OUT ARUGUMENTS)

### RULES FOR DEFINING CONSTRUCTOR:

- Constructor can be public, private, protected or default.
- Constructor cannot be static, non static, final or abstract.

- Constructor name must be same as that of classname.
- Constructor does not have any return type not even void
- Constructor can be with arguments or without arguments.
- We have 3 types of constructors:
  1. NO ARGUMENTS CONSTRUCTOR
  2. WITH ARGUMENTS CONSTRUCTOR
  3. DEFAULT CONSTRUCTOR

### **1. NO ARGUMENTS CONSTRUCTOR:**

```
public class Dog
{
    String dname;
    public Dog()
    {
        dname = "Tommy";
    }
    public static void main(String args[])
    {
        Dog D1 = new Dog();
        System.out.println(D1.dname);
        Dog D2 = new Dog();
        System.out.println(D2.dname);
    }
}
```

```
C:\Users\user\Desktop\KCCM37>javac Dog.java
C:\Users\user\Desktop\KCCM37>java Dog
Tommy
Tommy
```

- If a constructor does not have arguments, it is called **No Argument Constructor**.

- With no argument constructor, one of the disadvantages is same value will be initialised for each object. To overcome this, we will use with argument constructor (or) Parameterized constructor.

## 2.WITH ARGUMENT CONSTRUCTOR:

- If a constructor have any arguments, it is called as with argument constructor.

```
public class Dog  
{  
    String dname;  
  
    public Dog(String DogName)  
    {  
        dname = DogName;  
    }  
  
    public static void main(String args[])  
    {  
        Dog D1 = new Dog("Tommy");  
        System.out.println(D1.dname);  
  
        Dog D2 = new Dog("Fluffy");  
        System.out.println(D2.dname);  
    }  
}
```

```
C:\Users\user\Desktop\KCCM37>javac Dog.java  
C:\Users\user\Desktop\KCCM37>java Dog  
Tommy  
Fluffy
```

## 2.CREATE A CLASS AS BOOK

-DECLARE SOME INSTANCE VARIABLE AS

- + bname
- + bcolor
- + bprice

-TO INITIALISE THEM USE WITH ARGUMENT CONSTRUCTOR

-CREATE MAIN () & MAKE A CALL TO CONSTRUCTOR ATLEAST 3 TIMES TO PRINT INFORMATION OF 3 BOOKS

```
public class Book

{
    String bname,bcolor;
    int bprice;

    public Book(String bookname,String bookcolor,int bookprice)
    {
        bname = bookname;
        bcolor = bookcolor;
        bprice = bookprice;
    }

    public static void main(String args[])
    {
        Book A1= new Book("JAVA","BULE",600);

        System.out.println("bname:"+A1.bname+"\nbcolor:"+A1.bcolor+"\nbprice:"+A1.bprice);

        Book A2 = new Book("SQL","RED",400);

        System.out.println("bname:"+A2.bname+"\nbcolor:"+A2.bcolor+"\nbprice:"+A2.bprice);

        Book A3= new Book("MANUAL","BULE",200);
```

```

System.out.println("bname:"+A3.bname+"\nbcolor:"+A3.bcolor+"\nbprice:"+A3.bprice);
}
}

```

```

C:\Users\user\Desktop\KCCM37>javac Book.java

C:\Users\user\Desktop\KCCM37>java Book
bname:JAVA
bcolor:BULE
bprice:600
bname:SQL
bcolor:RED
bprice:400
bname:MANUAL
bcolor:BULE
bprice:200

```

### **3.CREATE A CLASS AS EMPLOYEE**

**-DECLARE SOME INSTANCE VARIABLES AS**

- + Ename
- + Eid
- + Esal

**-TO INITIALISE THEM WE USE PARAMETENISED CONSTRUCTOR**

**-IN CONSTRUCTOR KEEP ARGUMENTS NAME SAME ACTUAL VARIARBLE  
NAME.**

**-CREATE MAIN METHOD & CREATE 3 OBJECTS TO CALL CONSTRUCTOR 3  
TIMES.**

```

public class Employee
{
    String ename;
    int eid,esal;
    public Employee(String ename,int eid,int esal)
    {

```

```

ename = ename;
esal = esal;
eid = eid;
}

public static void main(String args[])
{
Employee E1 = new Employee("SUNIL",7689,20000);
System.out.println(E1.ename+" "+E1.eid+" "+E1.esal);

Employee E2 = new Employee("MAHESH",7765,4000);
System.out.println(E2.ename+" "+E2.eid+" "+E2.esal);

Employee E3= new Employee("TARUN",7135,20023);
System.out.println(E3.ename+" "+E3.eid+" "+E3.esal);
}
}

```

```

C:\Users\user\Desktop\New folder>javac Employee.java
C:\Users\user\Desktop\New folder>java Employee
null 0 0
null 0 0
null 0 0

```

#### **NOTE:**

- Generally, we will keep different names for argument variables & actual variables.
- In case, if we keep same name we are getting default value as output.
- Inorder to overcome this,we will use '**This**' keyword.

#### **4.CREATE A CLASS AS EMPLOYEE**

##### **-DECLARE SOME INSTANCE VARIABLES AS**

- Ename
- Eid
- Esal

- TO INITIALISE THEM WE USE PARAMETENISED CONSTRUCTOR
- IN CONSTRUCTOR KEEP ARGUMENTS NAME SAME ACTUAL VARIARBLE NAME.
- CREATE MAIN METHOD & CREATE 3 OBJECTS TO CALL CONSTRUCTOR 3 TIMES.

```
public class Employee
{
    String ename;
    int eid,esal;
    public Employee(String ename,int eid,int esal)
    {
        this.ename = ename;
        this.esal = esal;
        this.eid = eid;
    }
    public static void main(String args[])
    {
        Employee E1 = new Employee("SUNIL",7689,20000);
        System.out.println(E1.ename+" "+E1.eid+" "+E1.esal);
        Employee E2 = new Employee("MAHESH",7765,4000);
        System.out.println(E2.ename+" "+E2.eid+" "+E2.esal);
        Employee E3= new Employee("TARUN",7135,20023);
        System.out.println(E3.ename+" "+E3.eid+" "+E3.esal);
    }
}
```

```
C:\Users\user\Desktop\KCCM37>javac Employee.java  
C:\Users\user\Desktop\KCCM37>java Employee  
SUNIL 7689 20000  
MAHESH 7765 4000  
TARUN 7135 20023
```

### **THIS KEYWORD:-**

- + This key word indicate current object reference name.
- + We will use 'This' Keyword to differentiate between argument variables and actual variables.
- + We will use 'This' Keyword inside a constructor and inside a method.

## **5.CREATE A CLASS AS SHOPPING**

### **-DECLARE INSTANCE VARIABLES AS**

- + Product
- + Brand
- + Price
- + Color

### **-TO INITIALISE USE CONSTRUCTOR WITH ARGUMEMT & ARGUMENT SAME AS ACTUAL VARIABLE NAME.**

### **-CREATE MAIN () & MAKE A CALL TO CONSTRUCTOR ATLEAST 3 TIMES.**

```
public class Shopping  
{  
    String product,brand,color;  
    int price;  
  
    public Shopping(String product,String brand,String color,int price )  
    {  
        this.product = product;  
        this.brand = brand;  
        this.color = color;
```

```

this.price = price;
}

public static void main(String args[])
{
    Shopping A1= new Shopping("I-PHONE","APPLE","BULE",75000);
    System.out.println(A1.product+" "+A1.brand+" "+A1.color+" "+A1.price);

    Shopping A2 = new Shopping("LAP-TOP-5","DELL","GRAY",50000);
    System.out.println(A2.product+" "+A2.brand+" "+A2.color+" "+A2.price);

    Shopping A3= new Shopping("SHIRT","US-POLO","BULE",2000);
    System.out.println(A3.product+" "+A3.brand+" "+A3.color+" "+A3.price);
}
}

```

```

C:\Users\user\Desktop\New folder>javac Shopping.java

C:\Users\user\Desktop\New folder>java Shopping
I-PHONE APPLE BULE 75000
LAP-TOP-5 DELL GRAY 50000
SHIRT US-POLO BULE 2000

```

### **CONSTRUCTOR OVER LOADING:-**

*This process developing multiple constructor with same name but different arguments is called as a **Constructor Over loading**.*

### **RULES FOR DEFINING ARGUMENT LIST:**

- + Between Constructor, number of arguments must be different.
- + Between Constructor, types of arguments must be different.
- + Between Constructor, sequence or position of argument must be different.

### **6.CREATE A CAR CLASS**

### **2.DECLAE 5 INSTANCE VARIABLE**

- + Name
- + Color

- Price
- Capacity
- Model

### 3.CREATE WITH ARGUMENT CONSTRUCTORS

**CAR(String Name, String Color)**

**CAR(String Name, String Color, Double Price)**

**CAR(String Name, String Color, Double Price, Int Capacitiy, String Model)**

```
public class Car
```

```
{
```

```
String name,color,model;
```

```
int capacity;
```

```
double price;
```

```
public Car(String name,String color)
```

```
{
```

```
this.name = name;
```

```
this.color = color;
```

```
}
```

```
public Car(String name,String color,double price)
```

```
{
```

```
this.name = name;
```

```
this.color = color;
```

```
this.price = price;
```

```
}
```

```
public Car(String name,String color,double price,int capacity,String model )
```

```
{
```

```
this.name = name;
```

```

this.color = color;
this.price = price;
this.model = model;
this.capacity = capacity;
}

public static void main(String args[])
{
    Car C1 = new Car("BMW","RED");
    System.out.println(C1.name+" "+C1.color);
    Car C2 = new Car("AUDI","BULE",6000000.75);
    System.out.println(C2.name+" "+C2.color+" "+C2.price);
    Car C3 = new Car("TATA","WHITE",2000000.35,4,"TATA NEXON");
    System.out.println(C3.name+" "+C3.color+" "+C3.price+" "+C3.capacity+
"+C3.model);
}
}

```

```

C:\Users\user\Desktop\KCCM37>javac Car.java
C:\Users\user\Desktop\KCCM37>java Car
BMW RED
AUDI BULE 6000000.75
TATA WHITE 2000000.35 4 TATA NEXON

```

## 7.CREATE A BOOK CLASS

### 2.DECLARE SOME INSTANCE VARIABLES

- Name
- Price
- Author
- Pages

### 3.CREATE OVER LOADED CONSTRUCTORS

**Book(String Name, Int Price)**

**Book(String Name, String Author, Int Price)**

**Book(String Name, String Author, Int Price, Int Pages)**

**4.CREATE DISPLAY-1( ), DISPLAY-2( ), DISPLAY-3( )**

**5.CREATE MAIN ( ) & CREATE 3 OBJECTS ON EACH OBJECT CALL DISPLAY ( ).**

⊕ The main purpose of constructor overloading is to create an object in multiple ways.

```
public class Book
```

```
{
```

```
    String name,author;
```

```
    int price,pages;
```

```
    public Book(String name,int price)
```

```
{
```

```
        this.name = name;
```

```
        this.price = price;
```

```
}
```

```
    public Book(String name,String author,int price)
```

```
{
```

```
        this.name = name;
```

```
        this.author = author;
```

```
        this.price = price;
```

```
}
```

```
    public Book(String name,String author,int price,int pages )
```

```
{
```

```
        this.name = name;
```

```
        this.author = author;
```

```
this.price = price;  
  
this.pages = pages;  
}  
  
public void display1()  
{  
    System.out.println(name+" "+price);  
}  
  
public void display2()  
{  
    System.out.println(name+" "+author+" "+price);  
}  
  
public void display3()  
{  
    System.out.println(name+" "+author+" "+price+" "+pages);  
}  
  
public static void main(String args[])  
{  
    Book B1 = new Book("KOM",565);  
  
    B1.display1();  
  
    Book B2 = new Book("DESIGN DATA BOOK","S.MD.JALALUDEEN",989);  
  
    B2.display2();  
  
    Book B3 = new Book("MACHINE DARWING","N.D BHATT",400,345);  
  
    B3.display3();  
}
```

```
C:\Users\user\Desktop\New folder>javac Book.java
C:\Users\user\Desktop\New folder>java Book
KOM 565
DESIGN DATA BOOK S.MD.JALALUDEEN 989
MACHINE DARWING N.D BHATT 400 345
```

## CONSTRUCTOR CHAINING:

The process of calling one constructor from another constructor is called as **Constructor Chaining**

## CONSTRUCTOR CHAINING CAN BE ACHIEVED IN TWO WAYS:

1. Call to This → Represented as This ()
2. Call to Super → Represented as Super ()

### CALL TO THIS-THIS ()

- + The process of calling one constructor from another constructor of same class is called as a **Call to This**.
- + The main rule for this () is – Call to this must be first statement of a constructor.

#### Example:-

```
public class Add
{
    public Add()
    {
        this(100,200); //call to this->call another constructor of same class with 2 int args
        System.out.println("CONSTRUCTOR WITHOUT ARGUMENTS");
    }

    public Add(int i,int j)
    {
        this("java"); // call to this->call another constructor of same class with String args
        System.out.println("CONSTRUCTOR WITH 2 INT ARGUMENTS");
    }
}
```

```
public Add(String s)
{
    System.out.println("CONSTRUCTOR WITH STRING ARGUMENTS");
}

public static void main(String args[])
{
    Add A1 = new Add();
}
}
```

```
C:\Users\user\Desktop\New folder>javac Add.java
C:\Users\user\Desktop\New folder>java Add
CONSTRUCTOR WITH STRING ARGUMENTS
CONSTRUCTOR WITH 2 INT ARGUMENTS
CONSTRUCTOR WITHOUT ARGUMENTS
```

### Example:2:-

```
public class Add
{
    public Add()
    {
        System.out.println("CONSTRUCTOR WITHOUT ARGUMENTS");
    }

    public Add(int i,int j)
    {
        this("java");

        System.out.println("CONSTRUCTOR WITH 2 INT ARGUMENTS");
    }
}
```

```

public Add(String s)
{
    this();
    System.out.println("CONSTRUCTOR WITH STRING ARGUMENTS");
}

public static void main(String args[])
{
    Add A1 = new Add(100,200);
}

```

```

C:\Users\user\Desktop\New folder>java Add
CONSTRUCTOR WITHOUT ARGUMENTS
CONSTRUCTOR WITH STRING ARGUMENTS
CONSTRUCTOR WITH 2 INT ARGUMENTS

```

### Example:3:-

```

public class Add
{
    public Add()
    {
        this(800,1000);
        System.out.println("CONSTRUCTOR WITHOUT ARGUMENTS");
    }

    public Add(int i,int j)
    {
        System.out.println("CONSTRUCTOR WITH 2 INT ARGUMENTS");
    }
}

```

```

public Add(String s)
{
    this();
    System.out.println("CONSTRUCTOR WITH STRING ARGUMENTS");
}

public static void main(String args[])
{
    Add A1 = new Add("java");
}

```

```

C:\Users\user\Desktop\New folder>javac Add.java
C:\Users\user\Desktop\New folder>java Add
CONSTRUCTOR WITH 2 INT ARGUMENTS
CONSTRUCTOR WITHOUT ARGUMENTS
CONSTRUCTOR WITH STRING ARGUMENTS

```

### **DEFAULT CONSTRUCTOR:-**

*If we created object & did not defined a constructor then in that situation compiler will add one constructor by itself such constructor is called as Default Constructor.*

#### **Ex: WHAT WE WRITE**

```

class A
{
    public static void main(String args[])
    {
        A a1 = new A();
    }
}

```

#### **WHAT IS ACTUALLY MEANS**

```

class A
{
//@ added by compiler----> default constructor
public Add
{
}
public static void main(String args[])
{
A a1 = new A();
}
}

```

### **METHOD WITH RETURN TYPE:-**

- ⊕ *Void means no specific return type – it indicates that method is not returning any value.*
- ⊕ *In place of void we can give any specific return type for returning specific value.*
- ⊕ *We can use any primitive or non primitive data type as a return specific value.*
- ⊕ *Whenever we give method with specific return type it is compulsory to add return statement.*

### **RETURN STATEMENT:-**

- ⊕ *Return is a keyword which is used to return specific value*
- ⊕ *Syntax is return value;*
- ⊕ *Return keyword will provide data to JVM and make it to exit from the method*
- ⊕ *In one method there should only be one return statement.*
- ⊕ *Return statement must be the last statement of a method.*

**// static method without return type without arguments**

```

public static void Add()
{
    Add();
}

```

**// static method without return type with arguments**

```
public static void Add(int i)
{
    Add(1000);
}

// static method with intger return type without arguments

public static int Add()
{
    int v = Add();
    return int value;
}

// static method with string return type with intger arguments

public static String Add(int i)
{
    String v = Add(1000);
    return string value;
}

// non static method without return type without arguments

class A
{
    public void Add() A a1 = new A();
    {
        a1.add();
    }
}

// non static method without return type with arguments

public void Add(String s)
{
    a1.add("java");
}

// non static method with boolean return type without arguments

public boolean Add()
```

```

{
    boolean v = a1.Add();

    return boolean value;
}

// non static method with float return type with arguments

public float Add(int i,int j)

{
    float v = a1.Add(100,200);

    return float value;
}

}

```

Example-1:-

1.CREATE A CLASS AS AREAS

2.CREATE A STATIC METHOD WTH ARGUMENTS & RETURN TYPE AS

Area of Cricle (int radius) : float → calculate area of circle

3.CREATE MAIN () & MAKE A CALL TO AREA OF CIRCLE METHOD

```

public class Areas

{
    public static float areaofcircle(int radius)

    {
        return 3.14F*radius*radius;
    }

    public static void main(String args[])
    {
        float areaR = areaofcircle(5);

        System.out.println("Area of circle is :" +areaR);
    }
}
```

```
}
```

```
C:\Users\user\Desktop\New folder>javac Areas.java
```

```
C:\Users\user\Desktop\New folder>java Areas
Area of circle is :78.5
```

### Example-2:-

**1.CREATE A CLASS AS FACTORIAL**

**2.CREATE A NON STATIC METHOD WITH ARGUMENT & WITH RETURN TYPE AS**

Fact (int num) : int → develop logic of factorial of number

**3.CREATE MAIN () & MAKE A CALL TO FACT METHOD**

```
public class Factorial
```

```
{
```

```
    public int fact(int num)
```

```
{
```

```
    int fact =1 ;
```

```
    for(int i=num;i>=1;i--)
```

```
{
```

```
    fact = fact*i;
```

```
}
```

```
    return fact;
```

```
}
```

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

```
{
```

```
    Factorial f = new Factorial();
```

```
    int ft = f.fact(5);
```

```
    System.out.println("Factroical of number is :" +ft);
```

}

}

```
C:\Users\user\Desktop\New folder>javac Factorial.java  
C:\Users\user\Desktop\New folder>java Factorial  
Factroical of number is :120
```

## UML:- (UNIFIED MODELING LANGUAGE)

It is a pictoral representation of a program

+ → Public

- → Private

# → Protected

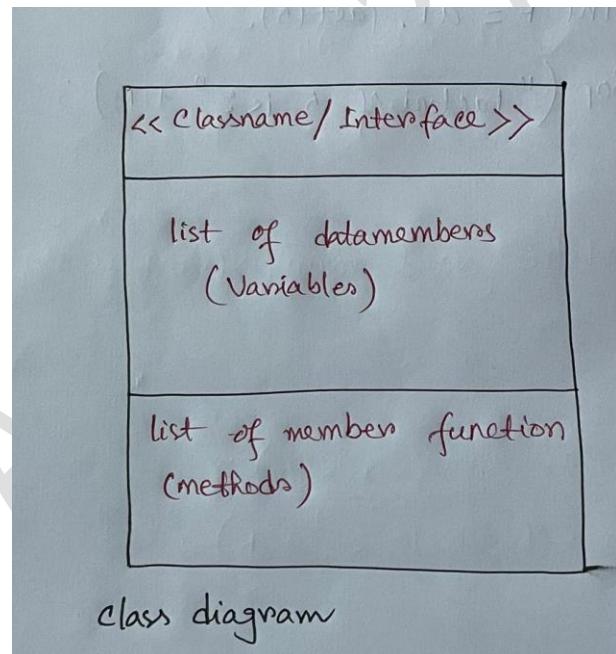
~ → Default



Extends



Implements



## **MODULE:-3 (OBJECT ORIENTED PROGRAMMING)**

- + OOPS
- + CLASS
- + OBJECT
- + INHERITANCE
- + SUPER()
- + METHOD OVER RIDING
- + SUPER AND FINAL KEYWORD
- + ENCAPSULATION
- + TYPE CASTING
- + OBJECT CASTING
- + ABSTRACTION
- + INTERFACE
- + POLYMORPHISM
- + METHOD BINDING

### **OOPS:- (OBJECT ORIENTED PROGRAMMING)**

- + It is a programming methodology or programming strategy which is required to fulfill every project requirement.
- + It has 4 pillars

PILLARS	PURPOSE
1.Inheritance	Code Reusability
2.Abstraction	Hiding Internal Details
3.Encapsulation	Securing Data
4.Polymorphism	One Thing Showing Multiple Behaviour

### **CLASS:-**

*Class is define as logical entity because it represent logic of an application.*

### **OBJECT:-**

*An object is defined as physical entity because physically it stores non static members in each object.*

<b>CLASS</b>	<b>OBJECT</b>
<ol style="list-style-type: none"> <li>1. logical entity</li> <li>2. Class is defined as blue print or Templet of an application.</li> <li>3. Class is created by using class key Word.</li> <li>4. Class contains any type of members (Static / Non Static).</li> <li>5. Class does not occpy any memory in heap area.</li> </ol>	<ol style="list-style-type: none"> <li>1.Physical entity</li> <li>2.Object is defined as reflection of a class.</li> <li>3.Object is created by using new key word.</li> <li>4.Object contains only Non-Static Members.</li> <li>5.With evey object memory will be reserved in heap area.</li> </ol>

### **INHERITANCE:**

- *It is the first pillar of oops concept.*
- *One class Acquiring properties of another class is called as Inheritance.*  
*(Or)*
- *One class containing members of another class without rewriting is called as Inheritance.*
- *Inheritance is also called is-a-relationship.*
- *Extends is a keyword which indicate that we are creating new class from an existing class.*
- *Extends keyword indicate about inheritance.*

### **Syntax:**

```

Class A
{
}
Class B Extends A
{
}
```

- *The class whose properties is are acquired is called as Parent class (or) Super class (or) Base class.*
- *The class who is acquiring the properties is called as Child class (or) Sub class (or) Drived class.*

- In the above syntax, “A” class is a parent class, “B” class is a child class.
- Parent class contains only its own property.
- Child class contains its own properties as well as properties of parent class.
- Therefore if we create an object of parent class, we can access only parent class properties, but if we create an object of child class, we can access both parent class as well as child class properties.

### NOTE:-

- Here properties are nothing but methods & variables.
- We can inherit only non static members.
- We can not inherit static member because they will always be located only once in static pool area (SPA).
- We can not inherit constructors because they are not a proper member of a class. They will just use for initialization purpose.

### Ex:-

```

class Youtub V1 --> Parent / Super / Base

{
    Search()
    Channels
    Downloads() 4
    Vidoes()
}

class Youtub V2 Extends Youtub V1 --> Child / Sub / Drive

{
    Subscription()
    Reaction()
}
  
```

is-a-relationship

6 -----> 4 (P) + 2 (Own)

- Every application can have two types of class
  1. BUSINESS LOGIC CLASS
  2. USER LOGIC CLASS

- The class which does not have main method is called as **Business logic class**.
- The class which have main method is called as **User logic class**.
- We should always save the application with **userlogic.java**

**Ex:-**

```
class op1
{
    public void Withdraw() -----> Business logic
}

class op2
{
    public void Deposit() -----> Business logic
}

public class ATM -----> User logic class -----> What user is expecting
{
    public static void main(String args[])
    {
        Withdraw();
        Deposit();
    }
}
```

-----> having main()

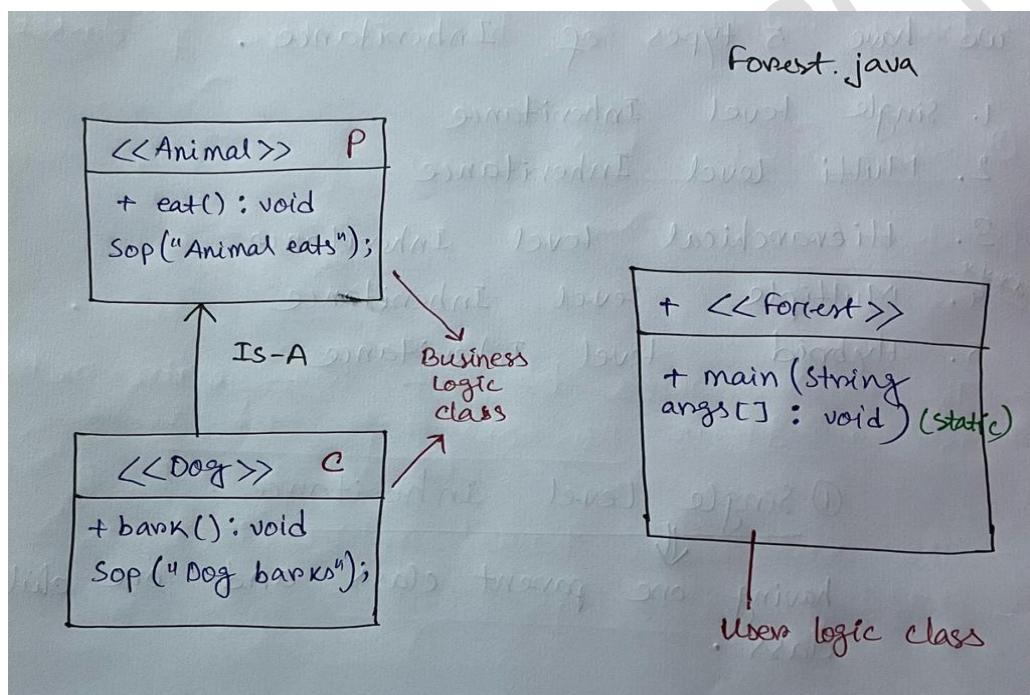
## **TYPES OF INHERITANCE:-**

*We have five types of inheritance.*

- 1.SINGLE LEVEL INHERITANCE
- 2.MULTI LEVEL INHERITANCE
- 3.HIERARCHICAL INHERITANCE
- 4.MULTIPLE INHERITANCE
- 5.HYBRID INHERITANCE

## **1.SINGLE LEVEL INHERITANCE:-**

*One parent class and one child class is called single level inheritance.*



## **PROGRAM:-**

```
class Animal  
{  
    public void Eat()  
    {  
        System.out.println("Animal Eats");  
    }  
}
```

```
}

}

class Dog extends Animal

{

public void Brak()

{



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

}

}

public class Forest

{

public static void main(String args[])

{

Dog F1 = new Dog();

F1.Eat();

F1.Brak();

}

}
```

```
C:\Users\user\Desktop\New folder>javac Forest.java

C:\Users\user\Desktop\New folder>java Forest
Animal Eats
Dog Brak
```

## 2. MULTI LEVEL INHERITANCE:-

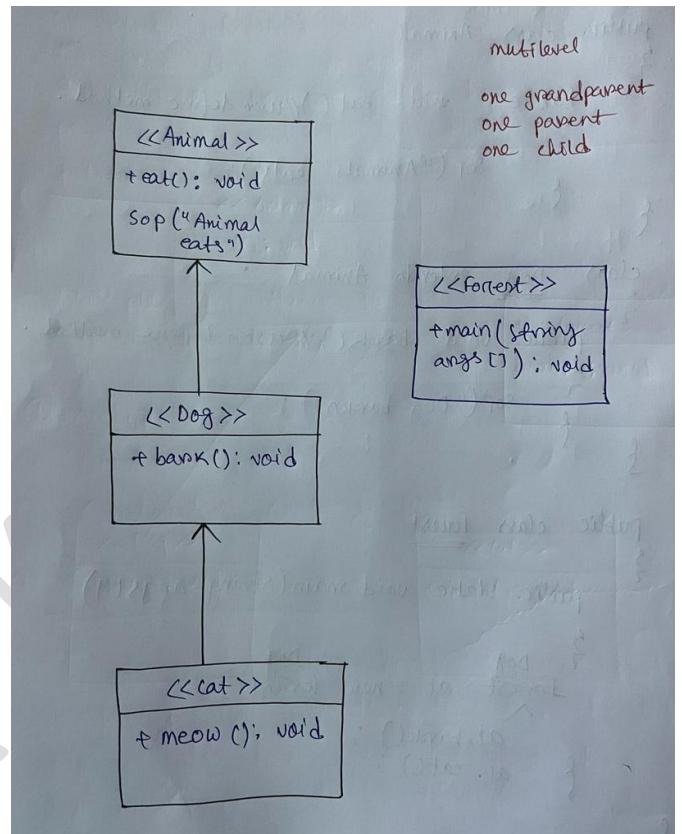
One grand parent, one parent, one child is called multi level inheritance.

### PROGRAM:

```
class Animal
{
    public void Eat()
    {
        System.out.println("Animal Eats");
    }
}

class Dog extends Animal
{
    public void Brak()
    {
        System.out.println("Dog Brak");
    }
}

class Cat extends Dog
{
    public void Meow()
    {
        System.out.println("Cat Meow");
    }
}
```



```

public class Forest
{
    public static void main(String args[])
    {
        Cat C1 = new Cat();
        C1.Eat();
        C1.Barak();
        C1.Meow();
    }
}

```

```

C:\Users\user\Desktop\New folder>javac Forest.java
C:\Users\user\Desktop\New folder>java Forest
Animal Eats
Dog Brak
Cat Meow

```

### 3.HIERARCHICAL INHERITANCE:-

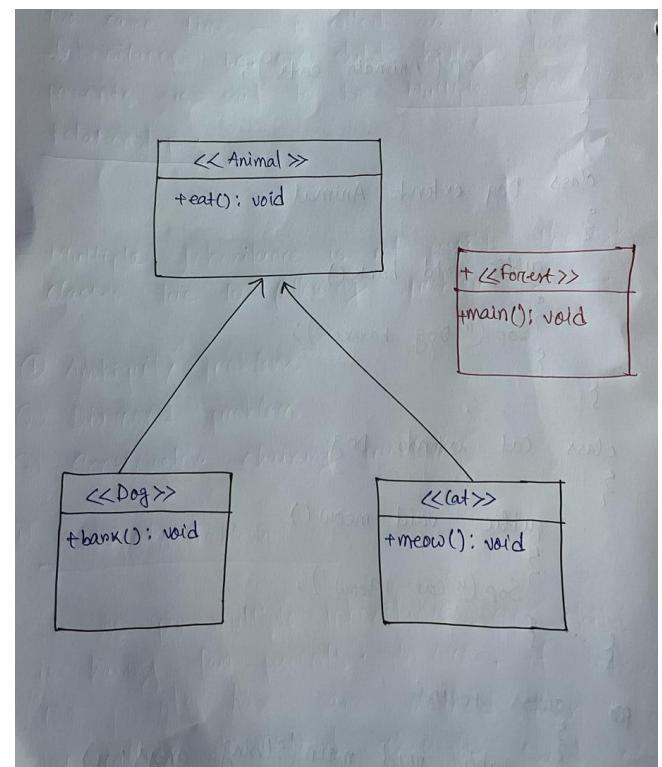
*One parent and two child class is called **hierarchical inheritance***

#### PROGRAM:-

```

class Animal
{
    public void Eat()
    {
        System.out.println("Animal Eats");
    }
}

```



```
class Dog extends Animal
{
    public void Brak()
    {
        System.out.println("Dog Brak");
    }
}

class Cat extends Dog
{
    public void Meow()
    {
        System.out.println("Cat Meow");
    }
}

public class Forest
{
    public static void main(String args[])
    {
        Dog D1 = new Dog();
        D1.Eat();
        D1.Brak();

        Cat C1 = new Cat();
        C1.Eat();
        C1.Meow();
    }
}
```

```
}
```

```
C:\Users\user\Desktop>New folder>javac Forest.java  
C:\Users\user\Desktop>New folder>java Forest  
Animal Eats  
Dog Brak  
Animal Eats  
Cat Meow
```

#### 4.MULTIPLE INHERITANCE:-

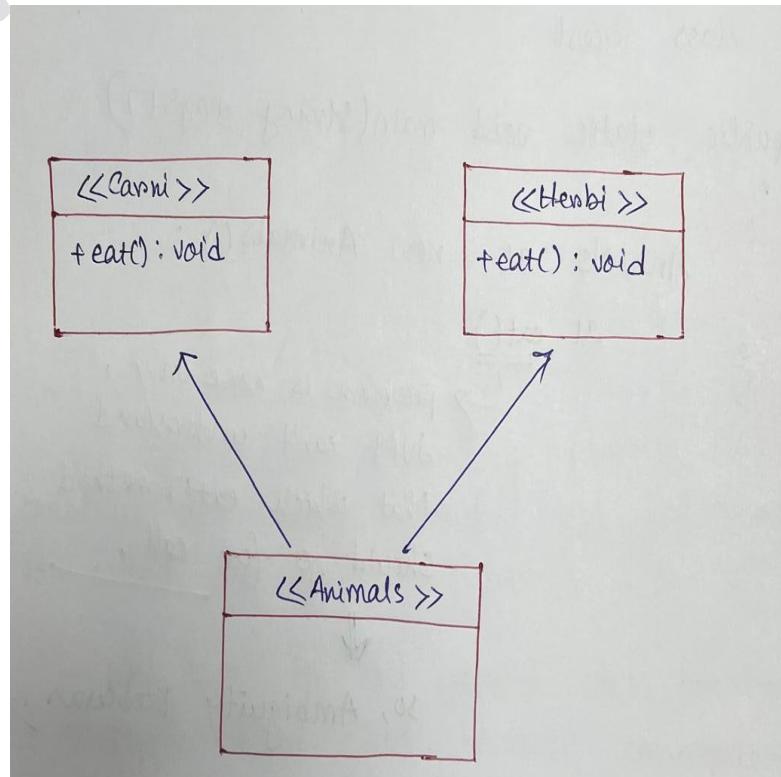
- + One class inheriting two parent class at the same time is called as **multiple inheritance** i.e. one child having two parent is said to be **Multiple Inheritance Relationship**.
- + Multiple inheritance is not possible through class due to following problems
  - 1.AMBIGUITY PROBLEM
  - 2.DIAMOND PROBLEM
  - 3.CONSTRUCTR CHAINING PROBLEM

#### 1.AMBIGUITY PROBLEM:-

- + As per multiple inheritance, one child is having two parent. In case, if both the parent have same method then while calling a method through child class there is a confusion which parent class method to call.
- + This problem is called as Ambiguity problem

#### Example:-

```
class Carni  
{  
    public void Eat()  
    {  
        System.out.println("Carni Eats");  
    }  
  
    class Herbi  
    {
```



```

public void Eat()
{
    System.out.println("Herbi Eat");
}

}

class Animals extends Carni,Herbi

{

}

public class Forest
{

    public static void main(String args[])
    {
        Animals A1 = new Animals();
        A1.Eat();
    }
}

```

problem is here only, JVM can't understand that which eat( ) method should go for call.

So, Ambiguity Problem

## 2.DIAMOND PROBLEM:-

- Since, the shape of a class diagram is in diamond structure, so due to this reason one of the problem is considered as Diamond shape class diagram problem.

## CALL TO SUPER / SUPER ():

- ⊕ The process of calling parent class constructor through child class constructor is called as **call to super**.
- ⊕ We have to call parent class constructor through child class because constructors can not be inherited.
- ⊕ Call to super, is implicit as well as explicit in nature.
- ⊕ Call to super will be implicit optional if class constructor does not have arguments (Here implicit means added by compiler).
- ⊕ Call to super, is explicit if parent class constructor have arguments (Here explicit means we should only added compulsorily).
- ⊕ Call to super, must be the first statement in a constructor. We can't write it in any other line except first line.

**//CALL TO SUPER IS IMPLICIT(OPTIONAL) CLASS CONSTRUCTOR DOES NOT HAVE ARGUMENTS.**

```

class A
{
    public A ()
    {
        System.out.println("A-class Constructor");
    }
}

class B extends A
{
    public B ()
    {
        //super() ---> default call to super added by compiler.

        System.out.println("B-class Constructor");
    }
}

public class C

```

```
{  
public static void main(String args[])  
{  
    B b1 = new B();  
}  
}
```

```
C:\Users\user\Desktop\New folder>javac C.java  
C:\Users\user\Desktop\New folder>java C  
A-class Constructor  
B-class Constructor
```

**//SUPER( ) IS EXPLICIT THAT MANDATORY SINCE PARENT CLASS CONSTRUCT  
OR HAVE ARGUMENT.**

```
class A  
{  
    public A (int i)  
    {  
        System.out.println("A-class Constructor");  
    }  
}  
  
class B extends A  
{  
    public B ()  
    {  
        super(10000);  
        System.out.println("B-class Constructor");  
    }  
}
```

```
}

public class C

{
    public static void main(String args[])
    {
        B b1 = new B();
    }
}
```

```
C:\Users\user\Desktop\New folder>javac C.java
C:\Users\user\Desktop\New folder>java C
A-class Constructor
B-class Constructor
```

### 3.CONSTRUCTOR CHAINING PROBLEM:

```
class A
{
    public A ()
    {
    }
}

class B
{
    public B ()
    {
    }
}
```

class C extends A,B //Class C having only one constructor i.e C( ) constructor. Because can't

```

{
    inherit.

public B ()
{
    super();

super();//Invalid -It is not in first statement constructor chaining problem.

}
}

```

- ⊕ As per multiple inheritance one child is having two parent & if both the parent have constructor, we need to call them by using multiple call to super, which is not possible & this lead to constructor chaining problem.
- ⊕ Therefore, due to ambiguity, diamond & constructor chaining problem, multiple inheritance is not possble through class. But, it is possible through interface.

## **5.HYBRID INHERITANCE:**

- ⊕ It is combination of hierarchical inheritance & multiple inheritance. Since, multiple inheritance is not possible, hyrid inheritance is also not possible.

## **ADVANTAGES OF INHERITANCE:**

- ⊕ Code Reusability
- ⊕ Connecting multiple logic at one place.
- ⊕ Easy to understand

## **REAL TIME EXAMPLES:**

- ⊕ Updating any application from one verison to another is an example of inheritance, where in without re-coding pervious verison features will be inherited.  
*(Facebook, Youtube App...)*
- ⊕ Back up data
- ⊕ Banking Application *(One Time You Give Your Details)*
- ⊕ E-commerce Application*(One Time You Fill Your Details)*

## **METHOD OVER RIDING:**

- ⊕ Parent class & child class are having same method but implementation is different process is called **Method Over Riding**.  
*(Or)*

- During inheritance, sub class have complete privilege to change method implementation of super class. This is called as a **Method Over Riding**.

## RULES FOR METHOD OVER RIDING:

- Inheritance is compulsory.**
- Method header of super & sub class is same.**
- Method signature of super & sub class is same.**
- Super class method should not be final.**

## WHY DO WE GO FOR OVER RIDING?

- When we want parent's property/feature but does not want there implementation rather we want to give our own implementation then we go for over riding.

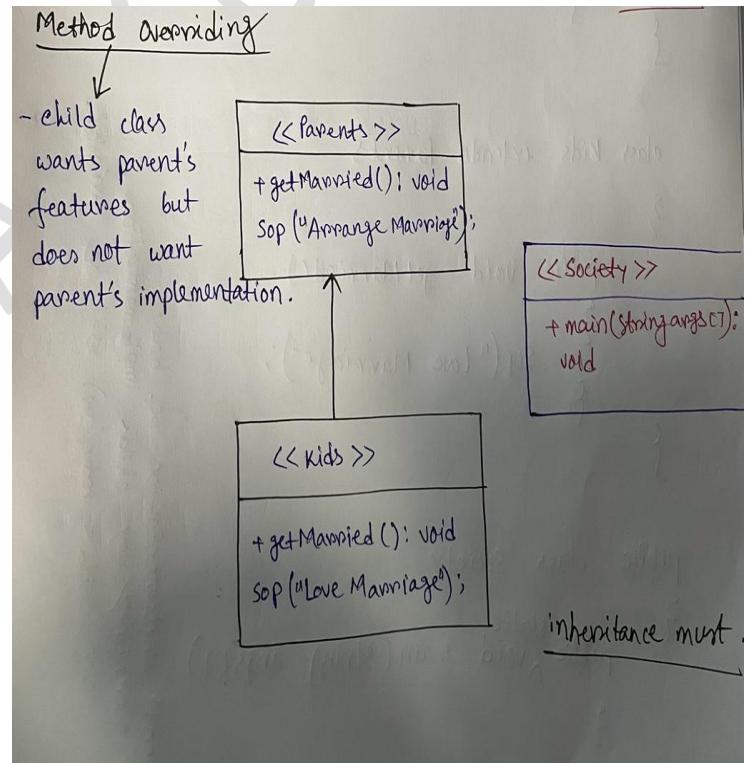
### PROGRAM:

```

class Parent
{
    public void getmarried()
    {
        System.out.println("arrange marriage");
    }
}

class Kids extends Parent
{
    public void getmarried()
    {
        System.out.println("Love marriage");
    }
}

```



```
public class Society  
{  
    public static void main(String args[])  
    {  
        Kids K1=new Kids();  
        K1.getmarried();  
    }  
}
```

```
C:\Users\user\Desktop\New folder>javac Socity.java  
C:\Users\user\Desktop\New folder>java Socity  
Love Marriage
```

### **METHOD RESOLUTION:**

- + When we call any method, method resolution will happen at two different stages compilation & execution
- + During compilation, compiler will see reference i.e. whatever class we give for reference variable, compiler will go to that class & just verify whether that method is present or not.

### **NOTE:**

- + Compiler will not decide body of the method.
- + During execution, JVM will see object type (right side whatever we gave after new keyword).

### **NOTE:**

- + JVM will decide body of the method.

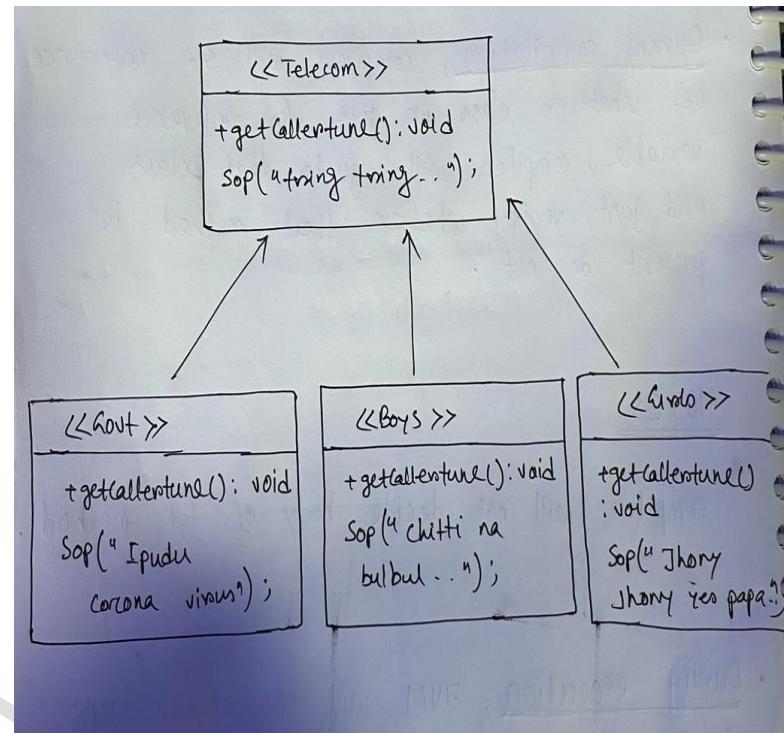
### **EXAMPLE:-1:**

```
class Telecom
{
    public void getcallertune()
    {
        System.out.println("TRING TRING .....");
    }
}

class Govt extends Telecom
{
    public void getcallertune()
    {
        System.out.println("IPPUDU CORANA VIRUS.....");
    }
}

class Boys extends Telecom
{
    public void getcallertune()
    {
        System.out.println("CHITTI NA BUL BUL CHITTI.....");
    }
}

class Girls extends Telecom
```



```

public void getcallertune()
{
    System.out.println("JHONY JHONY YES PAPA .....");
}

}

public class User
{
    public static void main(String args[])
    {
        Girls G1=new Girls();
        G1.getcallertune();

        Govt G2=new Govt();
        G2.getcallertune();

        Boys B1=new Boys();
        B1.getcallertune();
    }
}

```

```

E:\ccbp\New folder (2)>javac User.java
E:\ccbp\New folder (2)>java User
JHONY JHONY YES PAPA .....
IPPUDU CORANA VIRUS.....
CHITTI NA BUL BUL CHITTI.....

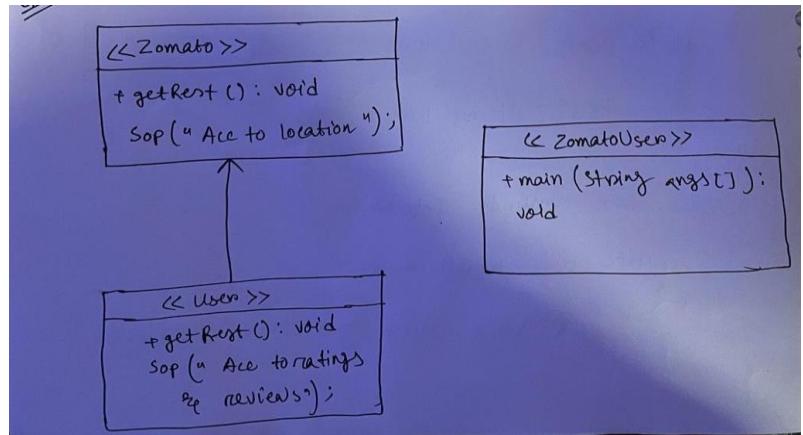
```

### EXAMPLE:-2:

```

class Zomato
{
    public void getresturant()

```



```

{
System.out.println("ACCORDING TO LOCATION");
}
}

class User extends Zomato

{
public void getresturant()
{
System.out.println("ACCORDING TO RATING AND REVIEWS");
}
}

public class Zomatouser

{
public static void main(String args[])
{
User U1=new User();
U1.getresturant();
}
}

```

```

C:\Users\user\Desktop\New folder>javac Zomatouser.java

C:\Users\user\Desktop\New folder>java Zomatouser
ACCORDING TO RATING AND REVIEWS

```

### **Example:-1**

```

-----
class A
{
```

```
public void M1()
{
    System.out.println("M1 method of A-class");
}

}

class B extends A
{
    public void M1()
    {
        System.out.println("M1 method of B-class");
    }

    public void M2()
    {
        M1();
    }
}

public class Demo
{
    public static void main(String args[])
    {
        B b1 = new B();
        b1.M2();
    }
}
```

```
C:\Users\user\Desktop>New folder>javac Demo.java  
C:\Users\user\Desktop>New folder>java Demo  
M1 method of B-class
```

### **Example:-2**

```
class A  
{  
    public void M1()  
    {  
        System.out.println("M1 method of A-class");  
    }  
}  
  
class B extends A  
{  
    public void M1()  
    {  
        System.out.println("M1 method of B-class");  
    }  
  
    public void M2()  
    {  
        super.M1(); //? Which M1 ()  
    }  
}  
  
public class Demo  
{  
    public static void main(String args[])
```

**if you add super keyword then parent class M1( ) first loaded.**

```
{  
    B b1 = new B();  
    b1.M2();  
}  
}
```

```
C:\Users\user\Desktop\New folder>javac Demo.java  
C:\Users\user\Desktop\New folder>java Demo  
M1 method of A-class
```

### **SUPER KEYWORD:-**

- ✚ Super keyword is used to call super class instance members.
- ✚ We will use super keyword in child class method.
- ✚ With the help of super keyword, we can get parent class implementation even after overriding happened.

### **PROGRAM:**

```
class A  
{  
    int i = 100;  
    public void M1()  
    {  
        System.out.println("M1 method of A-class");  
    }  
}  
  
class B extends A  
{  
    int i = 500;
```

```

public void M1()
{
    System.out.println("M1 method of B-class");
}

public void M2()
{
    M1(); // -----> Current Class M1()
    System.out.println(i); // -----> Current Class i value
    super.M1(); //-----> Parent class M1()
    System.out.println(super.i); //-----> Parent class i value
}
}

public class Demo
{
    public static void main(String args[])
    {
        B b1 = new B();
        b1.M2();
    }
}

```

```

C:\Users\user\Desktop\New folder>javac Demo.java

C:\Users\user\Desktop\New folder>java Demo
M1 method of B-class
500
M1 method of A-class
100

```

### **FINAL KEYWORD:**

- Final is keyword, which indicate no more changes are allowed.
- In a program, final keyword is applicable with
  - 1.VARIABLE
  - 2.METHOD
  - 3.CLASS

## FINAL VARIABLE:

If a variable is declared as final, two rules will be applicable.

### RULE:-1:

If a variable is declared as final, we should compulsorily initialize the value, before we use it.

### RULE:-2:

If a variable is declared as final, we can not change the value of it.

## SYNTAX:

FINAL DATATYPE VARIABLE = VALUE;

### EXAMPLE:-1:

```
class Sample
{
    final static int i;
    public static void main(String args[])
    {
        System.out.println(i);
    }
}
```

```
C:\Users\user\Desktop\New folder>javac Sample.java
Sample.java:3: error: variable i not initialized in the default constructor
  final static int i;
                           ^
1 error
```

### EXAMPLE:-2:

```
class Sample
```

```
{  
final static int i =500;  
  
public static void main(String args[])  
{  
    i = 1000; // CTE because i is a final variable  
  
    System.out.println(i);  
}  
}
```

```
C:\Users\user\Desktop\New folder>javac Sample.java  
Sample.java:6: error: cannot assign a value to final variable i  
    i = 1000;  
          ^  
1 error
```

### **FINAL WITH A CLASS:-**

*If a class is declared as final, we can not inherit that class, if we do we will get compile time error (CTE).*

### **SYNTAX:**

```
FINAL CLASS CLASSNAME
```

### **EXAMPLE:-1:**

```
final class A  
{  
}  
  
class B extends A  
{  
}  
  
public class Sample
```

```
{  
}
```

```
E:\ccbp\New folder>javac Sample.java  
Sample.java:4: error: cannot inherit from final A  
class B extends A  
      ^  
1 error
```

- A final class can inherit non-final class i.e. parent class should not be final but child class can be final.

#### EXAMPLE:-2:

Final class A

```
{  
}
```

Final class B extends A

```
{  
}
```

public class Sample

```
{  
}
```

```
E:\ccbp\New folder>javac Sample.java
```

```
E:\ccbp\New folder>java Sample  
Error: Main method not found in class Sample, please define the main method as:  
    public static void main(String[] args)  
or a JavaFX application class must extend javafx.application.Application
```

#### FINAL WITH A METHOD:-

If a method is declared as final, we can not over ride it, because final method can not be overridden.

#### SYNTAX:

FINAL METHOD HEADER	METHOD SIGNATURE
---------------------	------------------

### **EXAMPLE:-1:**

```
class Amazon
{
    final public void logo()
    {
        System.out.println("Amazon's logo");
    }
}

class Flipkart extends Amazon
{
    public void logo()
    {
        System.out.println("Flipkart's logo");
    }
}

public class Sample
```

```
E:\KCCM87>javac Sample.java
Sample.java:10: error: logo() in Flipkart cannot override logo() in Amazon
  public void logo()
                     ^
      overridden method is final
1 error
```

### **TYPE CASTING:-**

- *Converting one type of primitive data type into another type of primitive data type is called as **type casting**.*

*(OR)*

- Assigning one type of value into another type is called as **type casting**.
- It is divided into two types

## 1.WIDENING

## 2.NARROWING

### 1.WIDENING:-

- Converting smaller primitive data type into bigger primitive data type is called as **Widening**.
- Widening is also called as **Implicit Casting**.
- Byte → Short → Int → Long → Float → Double
- Since we are converting a smaller type into bigger type there is no loss of data.

### 2.NARROWING:-

- Converting bigger primitive data type into smaller primitive data type is called as **Narrowing**.
- Narrowing is also called as **Explicit Casting**.
- Byte ← Short ← Int ← Long ← Float ← Double
- Since we are converting a bigger type into smaller type there is loss of data.

### EXAMPLE:- PROGRAMME OF WIDENING

```
public class Widening
{
    public static void main(String args[])
    {
        byte b = 127;
        short s = b;
        int i = s;
        long L = i;
    }
}
```

Implicit in value, directly we can convert

```
System.out.println(b);
System.out.println(s);
System.out.println(i);
```

```
        System.out.println(L);
    }
}
```

```
C:\Users\user\Desktop\KCCM37>javac Widening.java
C:\Users\user\Desktop\KCCM37>java Widening
127
127
127
127
```

### EXAMPLE:-1:- PROGRAMME OF NARROWING

```
public class Narrowing
{
    public static void main(String args[])
    {
        long L = 76543211;
        int i = L ;
        short s = i;
        byte b = s;

        System.out.println(b);
        System.out.println(s);
        System.out.println(i);
        System.out.println(L);
    }
}
```

```
C:\Users\user\Desktop\KCCM37>javac Narrowing.java
Narrowing.java:6: error: incompatible types: possible lossy conversion from long to int
    int i = L ;
           ^
Narrowing.java:7: error: incompatible types: possible lossy conversion from int to short
    short s = i;
           ^
Narrowing.java:8: error: incompatible types: possible lossy conversion from short to byte
    byte b = s;
           ^
3 errors
```

## **EXAMPLE:-2:- PROGRAM OF NARROWING**

---

```
public class Narrowing
{
    public static void main(String args[])
    {
        long L = 76543211;
        int i = (int) L ; // Explicitly Converting Long to Int
        short s = (short) i; // Explicitly Converting Int to Short
        byte b = (byte) s; // Explicitly Converting Short to Byte

        System.out.println(b);
        System.out.println(s);
        System.out.println(i);
        System.out.println(L);
    }
}
```

### **OUT PUT**

---

-21-----> Byte (-21 means garbage value, because byte size small)  
-2837-----> Short (minimum because, size of short is small so it goes to garbage value)  
76543211-----> Int  
76543211-----> Long

### **OBJECT CASTING:**

---

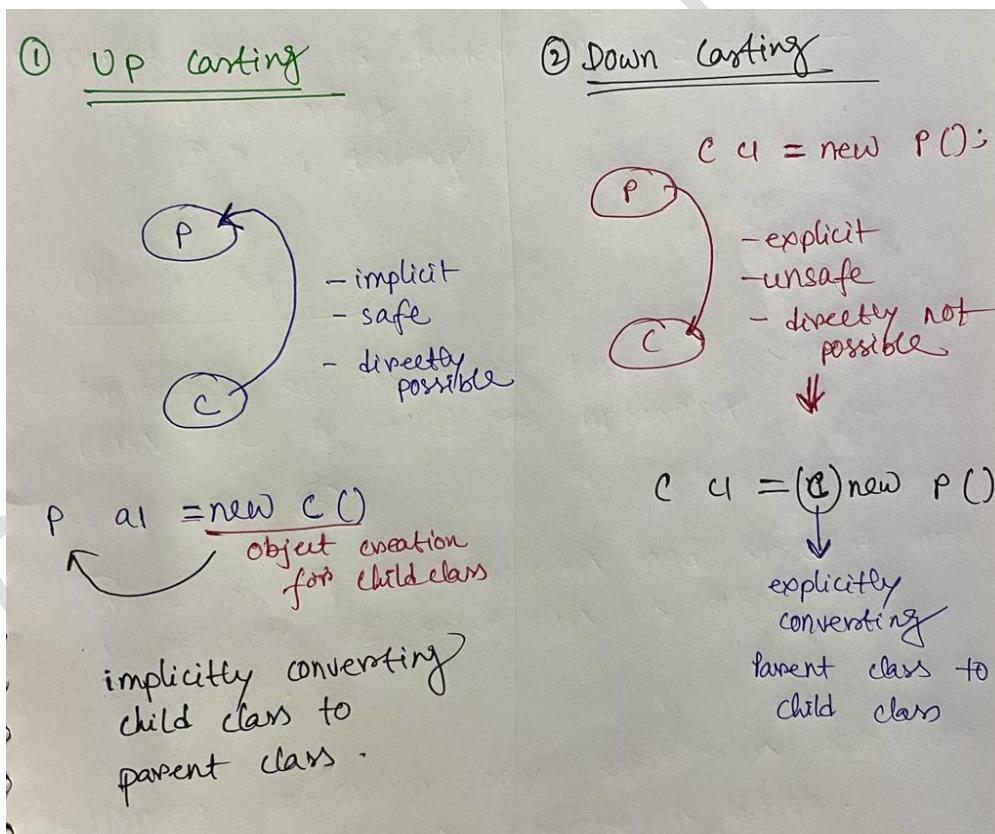
- ❖ Converting one class type of object into another class type of object is called as Object Casting.
- ❖ Object casting is divided into two types
  - 1.UP CASTING
  - 2.DOWN CASTING

## 1. UP CASTING

- Converting sub class object type into super class object type is called as **Up Casting**.  
**(OR)**
- Create an object of sub class and store it into a reference of super class is called as **Up Casting**.
- During **Up-Casting** only super class behaviour is visible sub class behaviour is hidden.

## 2. DOWN CASTING

- Converting super class object type into sub class object type is called as **Down Casting**.  
**(OR)**
- Converting an upcasted object into normal form is called as **Down Casting**.
- During **Down Casting** both sub class and super class behaviour is visible.
- Since super class does not contain property of sub class directly down casting is not possible rather explicitly we have to convert it.
- So that why down casting is explicit in nature.
- Only upcasted object is down casted, i.e direct down casted object creation is not possible if we do we will get **ClassCastException**.



## **EXAMPLE:-UPCASTING PROGRAM**

---

```
class A
{
    public void M1()
    {
        System.out.println("M1 METHOD");
    }
}

class B extends A
{
    public void M2()
    {
        System.out.println("M2 METHOD");
    }
}

public class UpCasting
{
    public static void main(String args[])
    {
        A a1 = new A(); // Parent Class Object
        a1.M1();

        B b1 = new B(); // Child Class Object
        b1.M1();
        b1.M2();

        A a2 = new B(); // UpCasted
    }
}
```

```
a2.M1(); // compiler check a2 which class reference then go 'A' class & M1() method  
is parent are not ?  
}  
}
```

```
C:\Users\user\Desktop\KCCM37>javac UpCasting.java  
C:\Users\user\Desktop\KCCM37>java UpCasting  
M1 METHOD  
M1 METHOD  
M2 METHOD  
M1 METHOD
```

### EXAMPLE:-DOWNCASTING PROGRAM

---

```
class A  
{  
    public void M1()  
    {  
        System.out.println("M1 METHOD");  
    }  
}  
  
class B extends A  
{  
    public void M2()  
    {  
        System.out.println("M2 METHOD");  
    }  
}  
  
public class DownCasting  
{
```

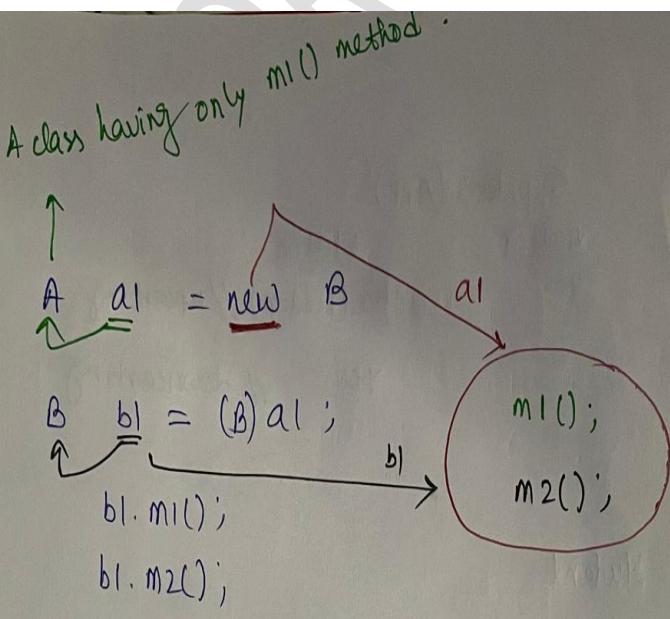
```

public static void main(String args[])
{
    A p = new A(); // Parent Class Object
    p.M1();
    B c = new B(); // Child Class Object
    c.M1();
    c.M2();
    A a1 = new B(); // UpCasted
    a1.M1();
    B b1 = (B) a1; // DownCasting → Take reference of upcasting & explicitly Converting to
                    child class type.
    b1.M1();
    b1.M2();
}
}

```

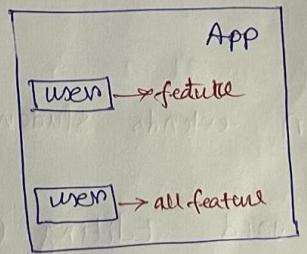
### NEED FOR OBJECT CASTING

- ⊕ If i want parent class properties & child class properties (both) then i should create 2times object for each parent & child.
- ⊕ But, here using object casting (UP+DOW N) i can access all into by using 1 object.



### \* Real Time example -

1. Hotstar app
  - ↳ subscribe user get all
  - but non-subscribers not get all



2. Banking app
  - ↳ Administration get another feature
  - ↳ User another feature

## **EXAMPLE:-1**

### **QspiderApp**

	<b>Viewing</b>	<b>Editing</b>
--	----------------	----------------

<b>Student</b>	<b>Yes</b>	<b>No(Hidden) // UpCasting</b>
----------------	------------	--------------------------------

<b>Admin</b>	<b>Yes</b>	<b>Yes // DownCasting</b>
--------------	------------	---------------------------

```
class Student
```

```
{
```

```
    public void Viewing()
```

```
{
```

```
        System.out.println("Student can view details");
```

```
}
```

```
}
```

```
class Admin extends Student
```

```
{
```

```
    public void Editing()
```

```
{
```

```
        System.out.println("Admin can edit student data");
```

```
}
```

```
}
```

```
public class QspiderApp
```

```
{
```

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

```
{
```

```
    Student A1 = new Admin();
```

```
    A1.Viewing();
```

```

Admin A2 = (Admin) A1;

A2.Viewing();

A2.Editing();

}

}

```

```

C:\Users\user\Desktop\KCCM37>javac QspiderApp.java
C:\Users\user\Desktop\KCCM37>java QspiderApp
Student can view details
Student can view details
Admin can edit student data

```

### **EXAMPLE:-2**

<b>Aeroplane</b>		
	<b>Tyres</b>	<b>Wings</b>
<b>Takeoff</b>	<b>Hidden</b>	<b>Visible // UpCasting</b>
<b>Landing</b>	<b>Visible</b>	<b>Visible // DownCasting</b>

```

class Takeingoff

{
    public void Tyres()
    {
        System.out.println("Tyers are Hidden");
    }
}

class Landing extends Takeingoff

{
    public void Wings()
    {

```

```

        System.out.println("Tyers and wings are visible");

    }

}

public class Aeroplane

{

    public static void main(String args[])

    {

        Takeingoff A1 = new Landing();

        A1.Tyres();

        Landing A2 = (Landing) A1;

        A2.Tyres();

        A2.Wings();

    }

}

```

```

C:\Users\user\Desktop\KCCM37>javac Aeroplane.java

C:\Users\user\Desktop\KCCM37>java Aeroplane
Tyers are Hidden
Tyers are Hidden
Tyers and wings are visible

```

### **EXAMPLE:-3**

<b>Hotstar</b>		
	<b>Tvshows</b>	<b>Liveshows</b>
<b>Normaluser</b>	<b>Visible</b>	<b>Hidden // UpCasting</b>
<b>Vipuser</b>	<b>Visible</b>	<b>Visible // DownCasting</b>

```

class Normaluser
{

```

```
public void Tvshows()
{
    System.out.println("Tvshows are Visible");
}

}

class Vipuser extends Normaluser
{
    public void Liveshows()
    {
        System.out.println("Tvshows and Liveshows both are Visible");
    }
}

public class Hotstar
{
    public static void main(String args[])
    {
        Normaluser N1 = new Vipuser();
        N1.Tvshows();
        Vipuser V1 = (Vipuser) N1;
        V1.Tvshows();
        V1.Liveshows();
    }
}
```

```
C:\Users\user\Desktop\KCCM37>javac Hotstar.java  
C:\Users\user\Desktop\KCCM37>java Hotstar  
Tvshows are Visible  
Tvshows are Visible  
Tvshows and Liveshows both are Visible
```

#### EXAMPLE:-4

LeaveApp		
	View	Edit
Member	Visible	Hidden // UpCasting
Manager	Visible	Visible // DownCasting

```
class Member  
{  
    public void View()  
    {  
        System.out.println("Employee apply a leave and view");  
    }  
}  
  
class Manager extends Member  
{  
    public void Edit()  
    {  
        System.out.println("Manager to edit and view a leave");  
    }  
}  
  
public class LeaveApp  
{
```

```

public static void main(String args[])
{
    Member M1 = new Manager();
    M1.View();

    Manager M2 = (Manager) M1;
    M2.View();

    M2.Edit();
}
}

```

```

C:\Users\user\Desktop\KCCM37>javac LeaveApp.java
C:\Users\user\Desktop\KCCM37>java LeaveApp
Employee apply a leave and view
Employee apply a leave and view
Manager to edit and view a leave

```

## **POLYMORPHISM:-**

### **DEFINITION:**

- It is a greek word
- Poly means many, morphism means forms.
- One thing showing multiple behaviour is called as polymorphism.  
*(Or)*
- One entity shows different behaviour is called as polymorphism.

PolyTechnique -----► Poly ---- Many  
Technique ----- Technologies

PolyClinic -----► Poly ---- Many  
Clinic ----- Treatment

### **TYPES:**

#### **COMPILE TIME POLYMORPHISM**

#### **RUN TIME POLYMORPHISM**

#### **COMPILE TIME POLYMORPHISM**

- During compile time one thing showing multiple behaviour is called as compile time polymorphism.

#### **EXAMPLE:**

Method over loading, where during compilation compiler will decide which behaviour Implemented. So here, we will call to same method but depending on arguments it shows different behaviour.

Public static void Add( )

Add( )

Public static void Add(int i)

Add(1000)

Public static void Add(Double d, String s)

Add(890.789, "JAVA")

Public static void Add(String s,int j)

Add(True)

CTE, because compiler will decide which method should be executed.

Add (True) there is boolean, so error occurred.

So, one compiler showing multiple behaviour so, it is polymorphism.

- It is also called as static polymorphism or compile time binding.

#### **RUN TIME POLYMORPHISM**

- During Run / Execution time one thing showing multiple behaviour, is called as Run Time polymorphism

#### **EXAMPLE:**

##### **Method Over Riding**

Where there are multiple methods with same name, during execution time only JVM will come to know which method to be executed depending on type of object.

class A

{

public void Run()

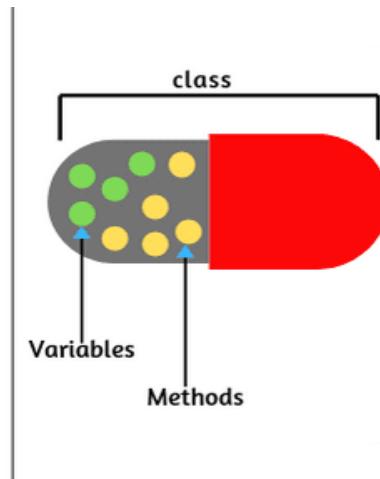
```
{  
    System.out.println("A class Method");  
}  
}  
  
class B extends A  
{  
    public void Run()  
    {  
        System.out.println("B class Method");  
    }  
}  
  
public class Main  
{  
    public static void main(String args[])  
    {  
        A a1 = new A();  
        a1.Run();  
        B b1 = new B();  
        b1.Run();  
        A a2 = new B();  
        a2.Run();  
    }  
}
```

```
C:\Users\user\Desktop\KCCM37>javac Main.java  
C:\Users\user\Desktop\KCCM37>java Main  
A class Method  
B class Method  
B class Method
```

## **ENCAPSULATION:**

**DEFINITION:** *The process of wrapping methods & variables into a single unit class in order protect them is called as **Encapsulation***

```
class  
{  
    data members  
    +  
    methods (behavior)  
}
```



## **PURPOSE:**

*Protecting the data members by keeping them private and accessing through some special method is nothing but **Encapsulation**.*

## **RULES FOR ACHIEVING ENCAPSULATION:**

**1.** Declare all the data members as private (if a data member is public it is not accessible outside of class)

**2.** Define separate setter and getter methods (it's not mandatory to define only setter and getter)

→ We can define any user defined methods

## **SETTER METHOD:**

- It is public in nature
- Setter method does not have specific return type.
- Setter method contains arguments same as data member
- Name of method is set followed by data member name.

## **GETTER METHOD:**

- It is public in nature
- Return type must be same as that of data member
- Name of method is get followed by data member name.

- *Getter method does not have arguments.*

**NOTE:**



- *For every data member we have define separate setter & getter method*

**PROGRAM OF ENCAPSULATION:**

```
package OOPS.Encapsulation;  
  
class Gmail  
{  
    private String uname;  
    private String pwd;  
    // setter and getter method  
  
    public void setUname(String uname)  
    {  
        this.uname = uname;  
    }  
  
    public String getUname()  
    {  
        //verification logic  
  
        if(uname=="qspkphb@gmail.com")  
            return "User Name is Verified";  
        else  
            return "User Not Found";  
    }  
  
    public void setPwd(String pwd)  
    {  
        this.pwd = pwd;  
    }  
  
    public String getPwd()  
    {  
        //verification logic
```

```

if(pwd=="kphb123")
    return "Password Verified";
else
    return "Invalid Password";
}
}

public class GmailUser
{
    public static void main (String[] args)
    {
        Gmail g1 = new Gmail ();
        g1.setUname ("qspresumes@gmail.com");
        g1.setPwd("kphb100");
        System.out.println (g1.getUname ());
        System.out.println (g1.getPwd ());
    }
}

```

Problems @ Javadoc Declaration Console X  
<terminated> GmailUser [Java Application] C:\Users\kssvi\p2\pool\plugins\org.eclipse.jdt.core\src  
User Not Found  
Invalid Password

### EXAMPLES:

1. Protecting bank details from an invalid user to access is an example of encapsulation.
2. Keeping a password for mobile to protect data is an example of encapsulation
3. Protecting an online documents by keeping password is an example of encapsulation.
4. Securing home by keeping lock is an example of encapsulation.

### EXAMPLE PROGRAM OF ENCAPSULATION

#### 2.CREATE AN ENCAPSULATED CLASS AS SCHOLARSHIP

##### 2.1 DECLARE PRIVATE VARIABLES AS APPID(STRING), SSCNO(STRING)

##### 2.2 CREATE SETTER FOR APPID & SSCNO

**2.3 CREATE GETTER METHOD FOR VERIFYING APPID ACTUAL APPID IS TS20224856**

**2.4 CREATE GETTER METHOD FOR VERIFYING SSCNO ACTUAL SSCNO IS 2022984563**

**2.5 CREATE USERSCHOLARSHIP CLASS**

**CREATE MAIN ()**

**CALL TO SETTER & GETTER**

```
package oops.Encapsulation;
```

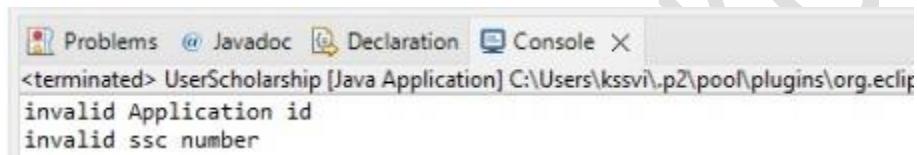
```
class Scholarship
{
    private String appid;
    private String sscno;
    public void setAppid(String appid)
    {
        this.appid = appid;
    }
    public String getAppid()
    {
        if(appid=="TS20224856")
            return "Application is verified";
        else
            return "invalid Application id";
    }
    public void setsscno(String sscno)
    {
        this.sscno = sscno;
    }
    public String getsscno()
    {
        if(sscno=="202245553962")
```

```

        return "ssc number is verified";
    else
        return "invalid ssc number";
    }
}

public class UserScholarship
{
    public static void main(String[] args)
    {
        Scholarship s1 = new Scholarship ();
        s1.setAppid ("TS4567899");
        s1.setsscno("2022456789");
        System.out.println (s1.getAppid ());
        System.out.println (s1.getsscno ());
    }
}

```



invalid Application id  
invalid ssc number

## METHOD BINDING:-

- ✚ The process of connecting method call with the method body is called as Method Binding
- ✚ Method binding is divided into 2 types
  - 1.COMPILE TIME BINDING
  - 2.RUN TIME BINDING

### 1. COMPILE TIME BINDING:

- ✚ During compilation, method will connecting to method body, is called as compile time binding
- ✚ Since the binding is happening at early stage it is also called as **early binding**.
- ✚ All the **static & final methods** will be binded during compile time.

### 2. RUN TIME BINDING:

- ✚ During execution, method call connecting with the method body is called as Run Time Binding
- ✚ Since the binding is happening at later stage it is called as **late binding**.

 All the non-static & abstract methods will be binded during run time.

### DIFFERENCE BETWEEN STATIC & FINAL

<b>STATIC</b>	<b>FINAL</b>
1.Static is Keyword which represent Single memory copy because it Loaded once in SPA	1.Final is Keyword which indicate no more changes are allowed.
2.Static keyword is applicable for variables, method, blocks & nested classes	2.Final keyword is applicable with variables, methods, classes.
3.We will go for static, if there is a single Value for particular situation  Ex: College name, University Name	3.We will go for final, if the value is constant every where.  Ex: $\pi = 3.14$
4.Static variables can be re-initialize	4. Final variables can not be re-initialize

### DIFFERENCE BETWEEN THIS, SUPER, SUPER (), THIS (), NEW

<b>THIS</b>	<b>SUPER</b>	<b>THIS()</b>	<b>SUPER()</b>	<b>NEW</b>
1. This keyword is used to different between actual variable & argument variable.	1. Super keyword is used to call super class member even after overriding happened.	1. Call to this is used to call a constructor From another constructor within the same class.	1.Call to super for the process of calling parent class constructor by using child class constructor.	1.New is a keyword to load all non-static member & create an object.
2.We can use this keyword at any statement inside a method & a constructor	2.We can use super keyword at any statement inside non-static method of child class.	2.Call to this must be the first statement of a constructor	2.Call to super must be the first statement of a child class constructor.	2.New keyword, we can use any statement whatever we want to create an object.
3.This keyword is explicit in nature	3.Super keyword is explicit in nature.	3.Call to this () is explicit in nature.	3.Call to super () is implicit & explicit in Nature.	3.New keyword is explicit in nature

## **ABSTRACTION:**

- ⊕ The process of hiding internal details & showing necessary data to the end user, is called as **Abstraction**.

(Or)

- ⊕ The process of showing what we are doing but not how we are doing is called as **Abstraction**.

(Or)

- ⊕ The process of giving method header & method signature, but not giving method implementation is called as **Abstraction**.

## **ABSTRACTION CAN BE ACHIEVED IN TWO WAYS**

### **1.USING ABSTRACT CLASS**

### **2.USING INTERFACE**

## **EXAMPLE OF ABSTRACTION:**

1.Sending an email where send button is visible but how the email is sending is not visible.

2.ATM application where in we enter the pin & choose the transaction. But, how the transaction is happening is not visible.

3.Driver knows only to apply breaks & accelerate but how it is working is hidden.

4.Remote of TV & AC, where we can operate through button, but how it is being operated is hidden.

## **ABSTRACT CLASS:**

- ⊕ Abstract is a keyword which indicates incompleteness.
- ⊕ Abstract class is a class which contains at least one abstract method.

**Ex:**

Abstract class Vechicle

{

    Abstract public void noofwheels();

}

- It is mandatory to mention abstract keyword for abstract class & abstract class.

### NORMAL/CONCRETE/COMPLETE METHOD:-

- If a method have method signature as well as method body method is called complete or concrete method.

**SYNTAX:** METHOD HEADER METHOD SIGNATURE

```
{  
    // METHOD BODY / IMPLEMENTATION  
}
```

**Ex:**

```
public void Run()  
{  
    System.out.println("In RUN");  
}  
  
public void Fly()  
{  
    System.out.println("FLY AWAY");  
}
```

### CONCRETE CLASS:-

A class with all the methods as concrete.

### ABSTRACT/INCOMPLETE METHOD:-

- If a method contains only method signature but not method implementation, is called as Abstract/Incomplete method.

**SYNTAX:** METHOD HEADER METHOD SIGNATURE

```
Ex: public void Run ()  
    public void Fly ()
```

- Abstract method has to be represented with **abstract keyword** & we have to add; in the end of method declaration.

**Ex:** **abstract public void Run ();**

**abstract public void Fly ();**

- If we didn't add **abstract keyword** or semicolon, we will get compile time error
- It is mandatory to mention the **abstract keyword** for **abstract method** & **abstract class**.

**Ex:-1**

**abstract class furit**

{

**Abstract public void taste();**

}

**Ex:-2**

**abstract class loans**

{

**abstract public void type();**

**abstract public void rate of intertest();**

}

## Q.CAN WE CREATE OBJECT OF ABSTRACT CLASS?

**A.No, we can not create an object because it contains abstract methods & abstract methods does not have any body to execute.**

**Ex:**

**Abstract class Vechicle**

{

**Abstract public void noofwheels();**

}

**public class A**

```

{
    Public static void main(Sting args[])
    {
        Vechicle V1 = new Vechicle(); // INSTANTIATION IS NOT POSSIBLE
        V1. noofwheels(); // Since Vechicle is a abstract class, we can't create object.
    }
}

```

## Q.CAN WE INHERIT ABSTRACT CLASS OR NOT?

A.**Yes, we can.** If any child classes are there for abstract class that child class has to undergo with any of the one rule.

1. Complete all abstract incomplete method of abstract class by means of overriding.

2. Declare your class also as abstract class.

 Abstract class can contains abstract method & concrete method (both).

**Ex:**

```

Abstract class A --> Abstract clas
{
    public void Add() --> Concrete method
    {
    }
    Abstract public void Add(); --> Abstract method
}

```

## PURPOSE OF ABSTRACT:-

1. To make simple information for end user to make user friendly (App should be simple in nature)

2. To make the application secure.

**Eg:**



*Suppose, one user send Hi (in whatup app) to another that other person get only 'Hi', not the coding of Hi*



*So, it is user friendly*

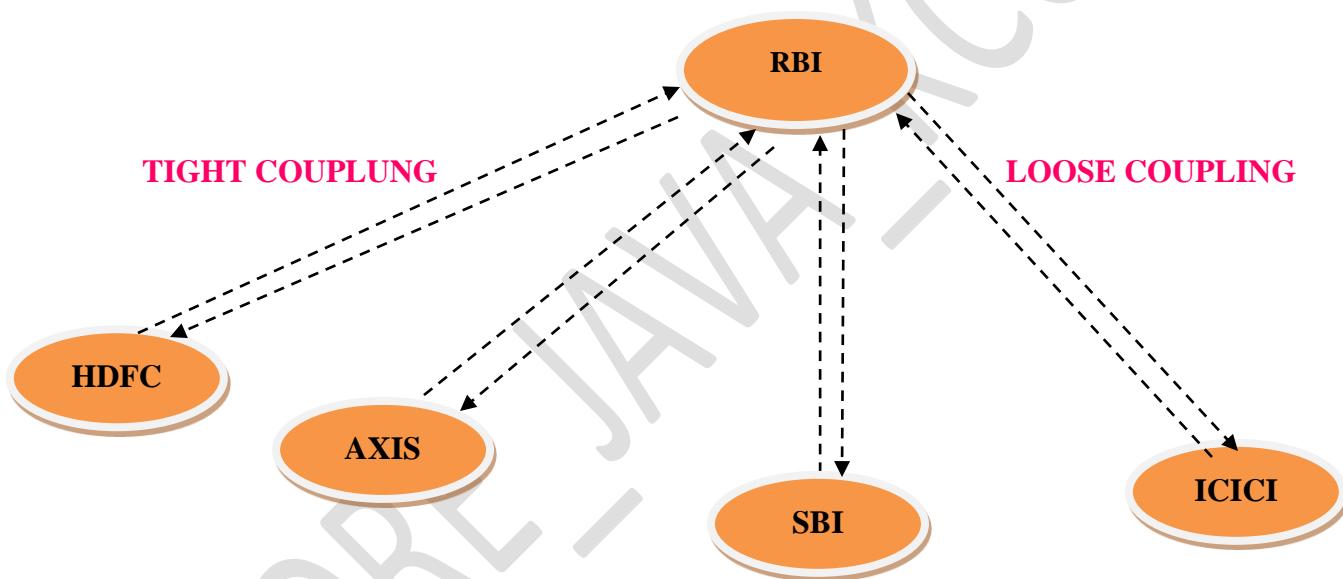
## **TIGHT COUPLING & LOOSE COUPING:-**

### **TIGHT COUPLING:-**

- + When one application is completely depends on another application for business purpose is called **Tight coupling**.

### **LOOSE COUPLING:-**

- + When one application is not completely depends on another application for business purpose, is called **Loose coupling**.



- + HDFC, AXIS...all are depends on RBI but RBI is not depends on HDFC.

## **STEPS SHOULD PERFORM FOR TIGHT & LOOSE COUPLING:-**

1. Create an abstract class with abstract method.
2. Create implementation class with required method implementations.

### **EXAMPLES:-**

*In this example we've created*

- a) Switch class (Abstract)

- b) Light class (Concrete)
- c) Fan class (Concrete)

Ex:

```
package practice;

abstract public class Switch
{
    abstract void on();
    abstract void off();
}

public class Fan extends Switch
{
    public void on()
    {
        System.out.println();
    }

    public void off()
    {
        System.out.println();
    }
}

public class light extends Switch
{
    public void on()
    {
        System.out.println ("light on");
    }

    public void off()
    {
        System.out.println ("light off");
    }
}
```

```

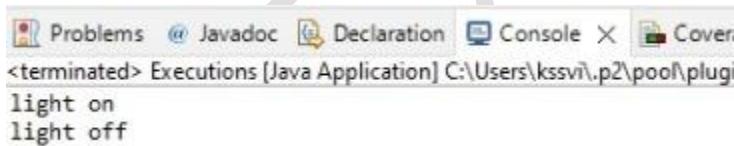
}

}

public class Executions
{
    public static void main(String[] args)
    {
        light l1 = new light();
        l1.on();
        l1.off();
    }
}

public class Execution
{
    public static void main(String[] args)
    {
        Fan f1 = new Fan();
        f1.on();
        f1.off();
    }
}

```



The screenshot shows the Eclipse IDE interface with the Java KCM37 watermark. In the bottom right corner, there is a screenshot of the Eclipse Console view. The console output shows the execution of the 'Executions' class, which prints 'light on' and 'light off' to the console.

```

Problems Javadoc Declaration Console × Coverage
<terminated> Executions [Java Application] C:\Users\kssvi\p2\pool\plugins
light on
light off

```

## 1. CREATE AN ABSTRACT CLASS AS CALCULATOR

## 2. CREATE ABSTRACT METHOD

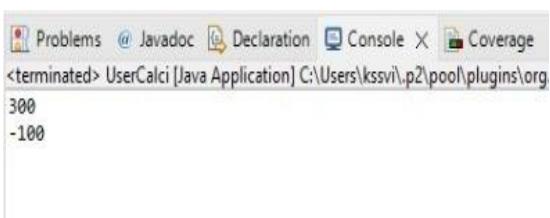
Add(int i, int j)

Sub(int i, int j)

## 3. CREATE CHILD CLASS AS CALCIDEV SHOULD IMPLEMENT ALL ABSTRACT METHODS.

#### 4. CREATE MAIN () IN USERCALCI & CREATE UPCASTED OBJECT TO ALL METHODS.

```
package Abstract;  
abstract public class Calculator  
{  
    abstract public void add(int i,int j);  
    abstract public void sub(int i,int j);  
}  
class DevCalci extends Calculator  
{  
    public void add(int i, int j)  
    {  
        System.out.println(i+j);  
    }  
    public void sub(int i, int j)  
    {  
        System.out.println(i-j);  
    }  
}  
public class UserCalci  
{  
    public static void main(String[] args)  
    {  
        Calculator c1 = new DevCalci();  
        c1.add(100,200);  
        c1.sub(100,200);  
    }  
}
```



Problems @ Javadoc Declaration Console X Coverage  
<terminated> UserCalci [Java Application] C:\Users\kssvi\p2\pool\plugins\org  
300  
-100

- 1. CRETAE AN ABSTRACT CLASS AS CAR**
- 2. CREATE ABSTRACT METHOD AS ENGINE(), COMFORTS(), DESIGN()**
- 3. CREATE CHILD CLASS AS AUDI & IMPLEMENT ENGINE() & COMFORT()**
- 4. CREATE ANOTHER CHILD FOR AUDI CLASS AS AUDI CLASS AS AUDIDESIGNER & IMPLEMENT DESGIN().**
- 5. CREATE USERLOGIC CLASS AS AUDICAR**

CARETE MAIN () & CREATE UPCASTED OBJECT

```
package Abstract;
abstract class Car
{
    abstract public void engine();
    abstract public void comforts();
    abstract public void design();
}

abstract class Audi extends Car
{
    public void engine() {
        System.out.println("AUDI ENGINE");
    }

    public void comforts()
    {
        System.out.println("AUDI IS COMFORTABLE");
    }
}

class AudiDesigner extends Audi
{
    public void design()
    {
        System.out.println("AUDI IS DESIGNED");
    }
}
```

```

public abstract class AudiCar
{
    public static void main(String[] args)
    {
        Car c1 = new AudiDesigner();
        c1.engine();
        c1.comforts();
        c1.design();
    }
}

```

```

@ Javadoc Declaration Console Coverage
<terminated> Audicar [Java Application] C:\Users\kssvi.p2\pool\plugins\org.eclipse.justj.openjdk.hots
AUDI ENGINE
AUDI OF COMFORTABLE
AUDI IS DESIGNED

```

*From the above program, we can observe that abstract class does not guarantees that it contains only abstract methods, due to this reason we cannot achieve 100% abstraction through abstract class in order to achieve 100% abstraction we will for interface.*

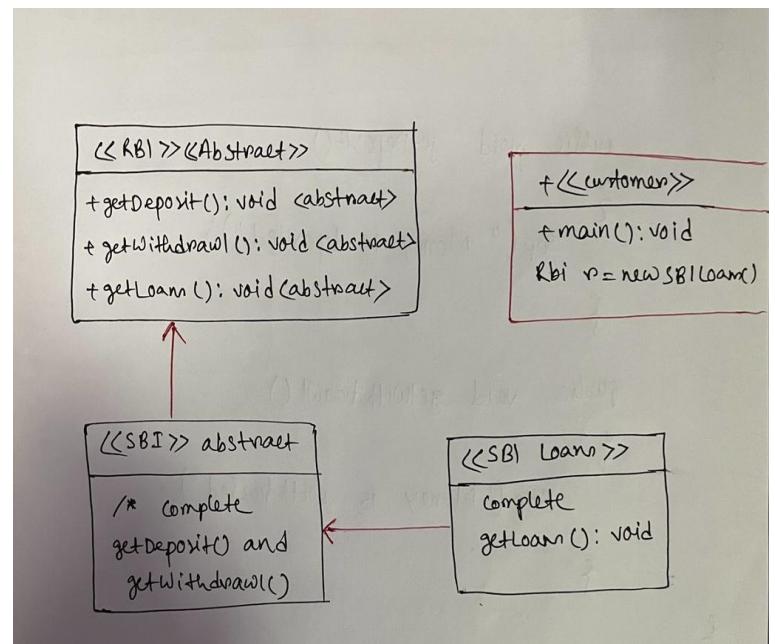
#### Example:-1

```

package Abstraction;
abstract class RBI
{
    abstract public void getdeposit();
    abstract public void getwithdraw();
    abstract public void getloans();
}

abstract class SBI extends RBI
{
    public void getwithdraw()
    {
        System.out.println("MONEY IS WITHDRAW");
    }
}

```



```

public void getdeposit()
{
    System.out.println("MONEY IS DEPOSITED");
}

}

class SBILoans extends SBI
{
    public void getloans()
    {
        System.out.println("LOANE IS DONE!");
    }
}

public class Customers
{
    public static void main(String args[])
    {
        RBI r = new SBILoans();
        r.getloans();
        r.getdeposit();
        r.getwithdraw();
    }
}

```

@ Javadoc Declaration Console Coverage  
<terminated> New\_configuration [Java Application] C:\Users\kssvi\p2\pool\plugins\org.ec  
LOANE IS DONE!  
MONEY IS DEPOSITED  
MONEY IS WITHDRAW

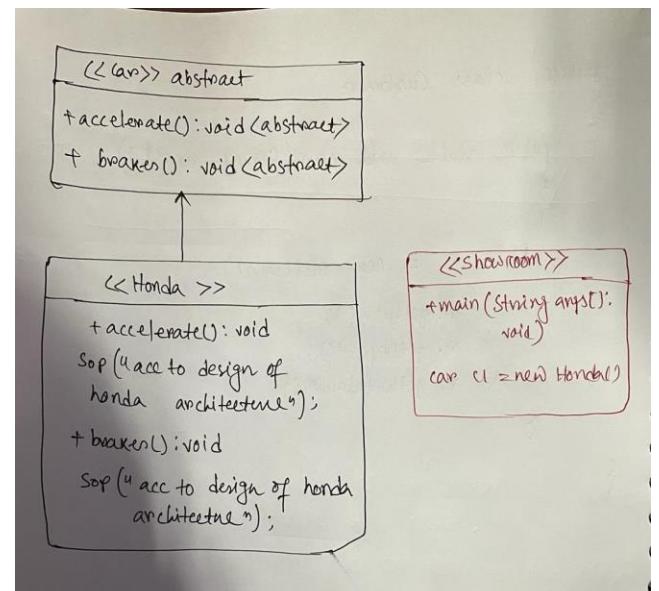
### Example:-2

```

package Abstraction;
abstract class Car
{
    abstract public void Accelerate();
    abstract public void Brakes();
}

class Honda extends Car

```



```

{
    public void Accelerate()
    {
        System.out.println("ACc TO DESIGN OF HONDA ARCHITECTURE");
    }

    public void Brakes()
    {
        System.out.println("ACC TO DESIGN OF HONDA ARCHITECTURE");
    }
}

public class Showroom
{
    public static void main(String args[])
    {
        Car c1 = new Honda();
        c1.Accelerate();
        c1.Brakes();
    }
}

```

The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The output window displays the following text:

```

@ Javadoc Declaration Console Coverage
<terminated> Showroom (1) [Java Application] C:\Users\kssvi\p2\pool\plugins\org.eclipse.justj.openjdk.hots
ACc TO DESIGN OF HONDA ARCHITECTURE
ACC TO DESIGN OF HONDA ARCHITECTURE

```

## CAN WE DECLARE A CLASS AS ABSTRACT EVEN THROUGH IT DOES NOT HAVE ANY ABSTRACT METHOD?

*Yes! We can declare the class as abstract even through it does not have any abstract method, in two different situation.*

### SITUATION:-1

*If all the members of a class is static we need not to create any object.*

*In that situation, we can declare the class as abstract. Because, abstract class says object creation is not possible. By doing this we can restricted the user from un-necessary object creation*

(we can save area of memory (Heap))

→ So, efficiency also

### abstract class Sample

```
{  
    static int i = 100;  
    public static void Add()  
    {  
        System.out.println(i);  
    }  
    public static void main(String args[])  
    {  
        Add();  
    }  
}
```

### SITUATION:-2

We can access parent class members through child class object. So, therefore we can declare parent class as abstract.

**CAN WE HAVE CONSTRUCTOR IN ABSTRACT CLASS SINCE WE CAN NOT  
CREATE AN OBJECT OF IT?**

Yes, we can create a constructor in abstract class even though we cannot create an object of it by using call to super i.e. through child class constructor, we can call to parent class constructor.

### abstract class A

```
{  
    A()  
    {  
        System.out.println("A-CLASS CONSTRUCTOR");  
    }  
}  
  
class B extends A  
{
```

```

B()
{
    System.out.println("B-CLASS CONSTRUCTOR");
}
}

```

```

public class User
{
    public static void main(String[] args)
    {
        A a1 = new B();
    }
}

```

```

@ Javadoc Declaration Console X Coverage
<terminated> User (1) [Java Application] C:\Users\kssvi\p2\pool\plugins\or
A-CLASS CONSTRUCTOR
B-CLASS CONSTRUCTOR

```

### INTERFACE:

**DEFINITION:** An interface is a type definition block (block of codes) whose type convention is same as that of class.

```

interface Animals
{
    void Eat();
    void Makenoise();
    void Color();
    void Species();
}

```

► An interface has to be represented with interface keyword.

### SYNTAX:-

<b>INTERFACE KEYWORD INTERFACE NAME</b>
{ // BODY OF INTERFACE }

- By default all methods of interface are public and abstract, whether we write or don't write.

**interface A**

```
{
void run();
public void fly();
abstract public void cry();
}
```

**interface A**

```
{
public abstract void run();
public abstract void fly();
public abstract void cry();
}
```

- By default all the variables of interface are public, static & final. Whether you write or you don't write it.

**interface A**

```
{
int i = 9; // public static final int i = 9;
static String s = "java"; // public static final String s = "java";
final int k = 88; // public static final int k = 88;
public static final float fq = 99.9f;
}
```

## Q.CAN WE INstantiate AN INTERFACE?

(Or)

## Q.CAN WE CREATE AN OBJECT OF INTERFACE?

No, we cannot create an object of interface because all methods are by default abstract, but we can create a reference of interface.

## IMPLEMENTATION:-

Implements is the keyword, we will use if any class want to form a relationship with Interface.

## SYNTAX:-

```
INTERFACE CAR
{
    VOID PRICE ();
}

CLASS AUDI IMPLEMENTS CAR
{}
```

## POSSIBLE RELATIONSHIP WITH KEYWORDS:

PARENT	CHILD	KEYWORD	SYNTAX
CLASS	CLASS	EXTENDS	CLASS B EXTENDS A
INTERFACE	INTERFACE	EXTENDS	INTERFACE B EXTENDS A
INTERFACE	INTERFACE	IMPLEMENTS	CLASS B IMPLEMENTS A
ABSTRACT	CLASS	EXTENDS	CLASS B EXTENDS A
CLASS, CLASS	CLASS	NOT POSSIBLE	NOT POSSIBLE
INTERFACE,INTERFACE	CLASS	IMPLEMENTS	CLASS C IMPLEMENTS A,B
INTERFACE,INTERFACE	INTERFACE	EXTENDS	INTERFACE C EXTENDS A,B

### EXAMPLE:-1

```

package practice;

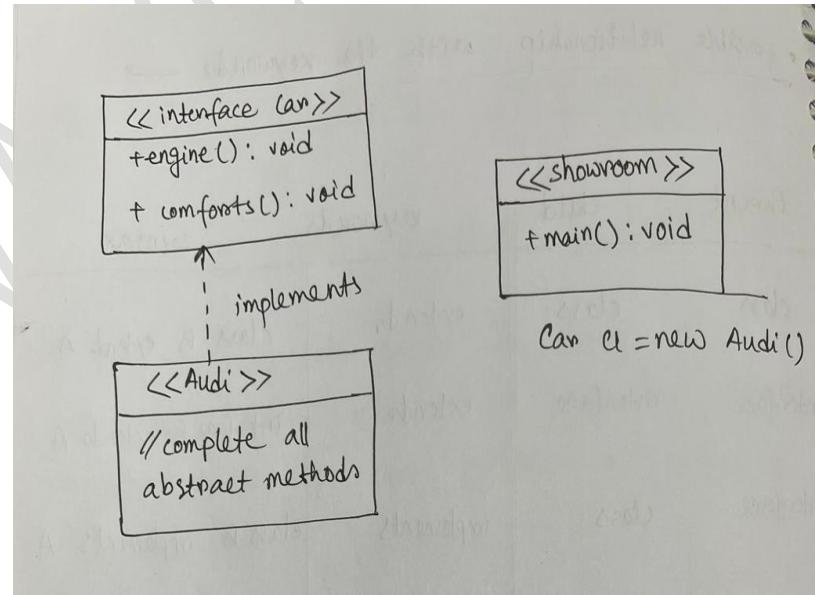
interface Car
{
    public void Engine();
    public void Comforts();
}

class Audi implements Car
{
    public void Engine()
    {
        System.out.println("AS PER AUDI STANDRAD");
    }

    public void Comforts()
    {
        System.out.println("AS PER AUDI DESIGN");
    }
}

public class Showroom

```



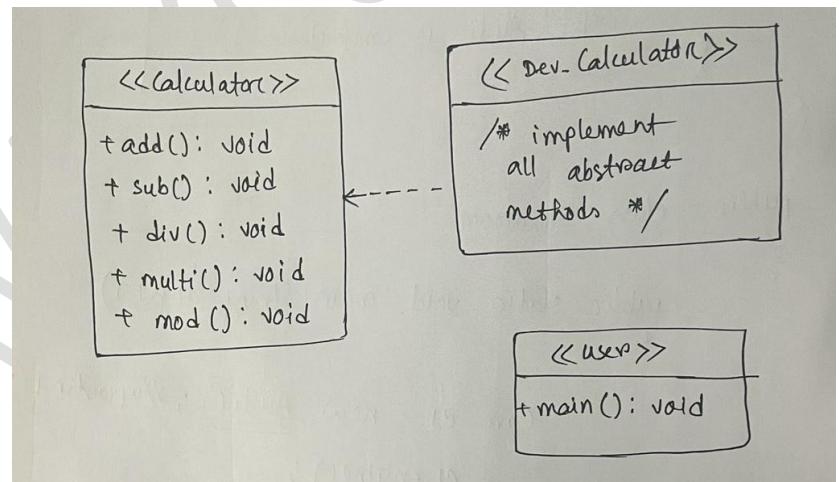
```
{
public static void main(String args[])
{
    Car c1 = new Audi();
    c1.Engine();
    c1.Comforts();
}
}
```

```
@ Javadoc Declaration Console X Coverage
<terminated> Showroom (2) [Java Application] C:\Users\kssvi.p2\pool\plugin:
AS PER AUDI STANDRAD
AS PER AUDI DESIGN
```

### EXAMPLE:-2

```
package Aneesh;
interface Calculator
{
    public abstract void Add();
    public abstract void Sub();
    public abstract void Div();
    public abstract void Multi();
    public abstract void Mod();
}

class Dev_calculator implements Calculator
{
    static int i = 900, j = 600;
    public void Add()
    {
        System.out.println(i+j);
    }
    public void Sub()
    {
```



```

        System.out.println(i-j);
    }

public void Div()
{
    System.out.println(i/j);
}

public void Multi()
{
    System.out.println(i*j);
}

public void Mod()
{
    System.out.println(i%j);
}

public class User
{
    public static void main(String args[])
    {
        Calculator c1 = new Dev_calculator();
        c1.Add();
        c1.Sub();
        c1.Div();
        c1.Multi();
        c1.Mod();
    }
}

```

@ Javadoc Declaration Console X Coverage  
<terminated> User [Java Application] C:\Users\kssv\l\p2\pool\plugins\org.eclipse.justj.openjdk.hot  
1500  
300  
1  
540000  
300

### EXAMPLE:-3

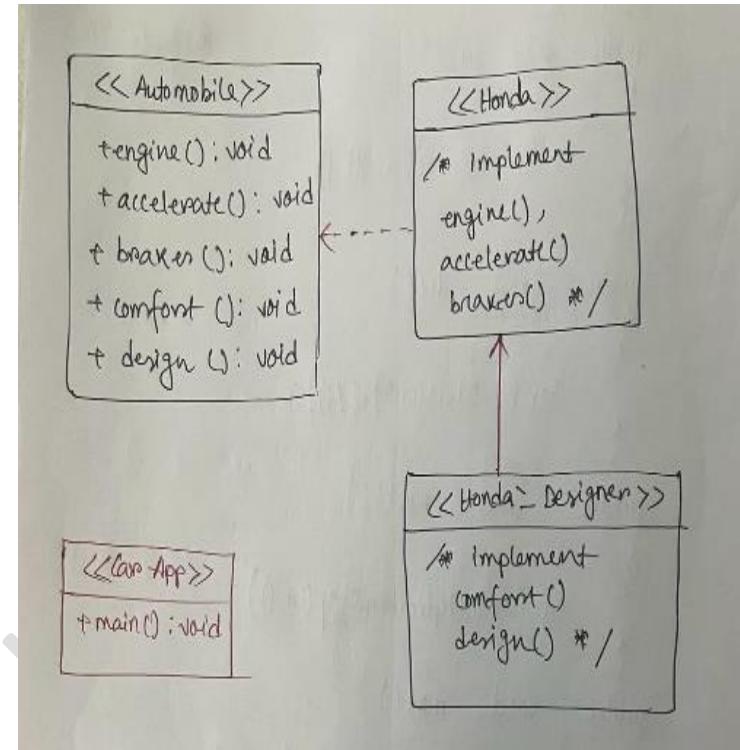
```
package Sri;
interface Automoblie
{
    public abstract void Engine();
    public abstract void Accelerate();
    public abstract void Brakes();
    public abstract void Comfort();
    public abstract void Design();
}

abstract class Honda implements Automoblie
{
    public void Engine()
    {
        System.out.println("ENGINE CONDITION IS GOOD");
    }

    public void Accelerate()
    {
        System.out.println("SPEED CONTROL");
    }

    public void Brakes()
    {
        System.out.println("BREAKS WILL WORKS");
    }
}

class Honda_Designer extends Honda
{
    public void Comfort()
    {
        System.out.println("COMFORTS ARE GOOD");
    }
}
```

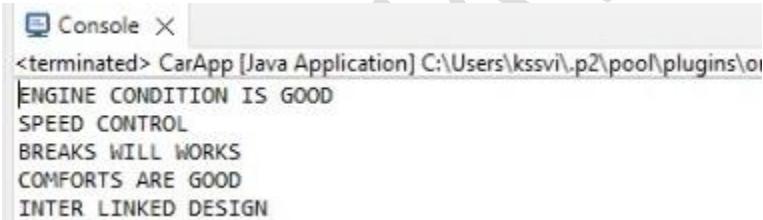


```
}

public void Design()
{
    System.out.println("INTER LINKED DESIGN");
}

}

public class CarApp
{
    public static void main(String args[])
    {
        Automoblie a1 = new Honda_Designer();
        a1.Engine();
        a1.Accelerate();
        a1.Brakes();
        a1.Comfort();
        a1.Design();
    }
}
```



A screenshot of a Java console window titled "Console X". The window shows the output of a Java application named "CarApp [Java Application]". The output consists of five lines of text: "ENGINE CONDITION IS GOOD", "SPEED CONTROL", "BREAKS WILL WORKS", "COMFORTS ARE GOOD", and "INTER LINKED DESIGN".

```
<terminated> CarApp [Java Application] C:\Users\kssvi\p2\pool\plugins\oi
ENGINE CONDITION IS GOOD
SPEED CONTROL
BREAKS WILL WORKS
COMFORTS ARE GOOD
INTER LINKED DESIGN
```

## **DIFFERENCE B/W ABSTRACT & INTERFACCE**

<b>ABSTRACT</b>	<b>INTERFACCE</b>
1. Abstract is class having abstract keyword word & having at least one abstract method.	1. Interface is block of codes represents as interface keyword.
2. In abstract class, we can have concrete as well as abstract method.	2. In interface, all the methods are public & abstract.
3. In abstract class, we can have any type of variables.	3. In interface, all the variables are public, static & final.
4. In abstract class, we can have constructor which is executed through child class constructor.	4. In interface, we can not have constructor because all the variables are static
5. We can achieve partial abstraction through abstract class.	5. We can achieve 100% abstraction through interface.
6. We can not achieve multiple inheritance through abstract class.	6. We can achieve multiple inheritance through interface.

## **ADVANCE ENHANCEMENT ALONG WITH INSIDE INTERFACE:-**

- From JDK 1.8, it starts.....

1. Complete static methods are allowed

`public static void M1()`

`{`

`}`

2. Default non-static methods are also allowed

`default void M2()`

`{`

`}`

- Object class is super most class in java because it by default present

*(In java. Language package)*

```

public class A extends object()
{
    }  

    
    By default it is parent  

    
    Constructor is responsible for inheritance.  

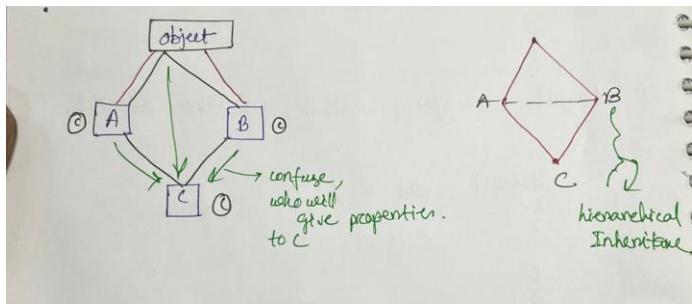
    One by default constructor always there.  

    
    Multiple inheritance not in java class.

```

### DIAMOND AMBIGUITY PROBLEM:-



- A, B are confuse that who will give properties to C → Multiple inheritance not possible.
- In Interface, no object class → So no diamond shape problem

 So, multiple inheritance possible

Compile Time Polymorphism → Method Over Loading

Run Time Polymorphism → Method Over Riding

4 class members → Constructor, Methods, Variables, Blocks

class A

```

{  

    A0 → Constructor  

    {  

}

```

public void M1() → Methods

{

}

int x = 100; → Variables

}  **Blocks**

**EXAMPLE:-1**

-----  
package Sri;

interface A

{

**void** Car();

}

interface B

{

**void** CarAudi();

}

interface C extends A,B

{

}

class D implements A,B

{

**public void** Car()

{

        System.out.println("CAR IS");

}

**public void** CarAudi()

{

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

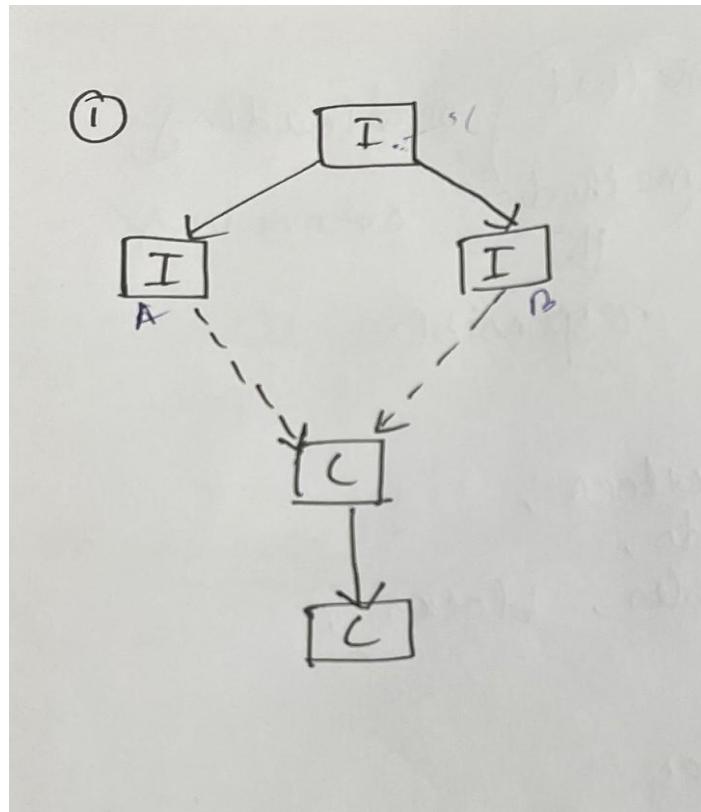
}

}

class E extends D

{

}



```

public class User
{
    public static void main(String args[])
    {
        D a1 = new E();
        a1.Car();
        a1.CarAudi();
    }
}

```

```

Console X
<terminated> User (3) [Java Application] C:\Users\kssvi\p2\p1>
CAR IS
AUDI IS

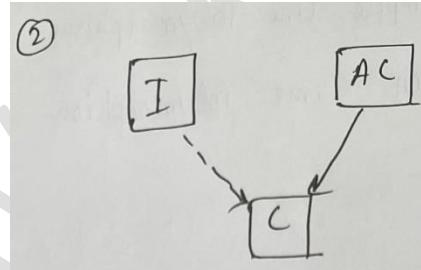
```

### EXAMPLE:-2

```

package Sri;
interface A
{
    void Car();
}
abstract class B implements A
{
    abstract public void CarAudi();
}
class C extends B implements A
{
    public void Car()
    {
        System.out.println("CAR IS HIGH COST");
    }
    public void CarAudi()
    {

```



```

        System.out.println("AUDI NEW MODEL");
    }
}

public class Sri
{
    public static void main(String args[])
    {
        C a1 = new C();
        a1.Car();
        a1.CarAudi();
    }
}

```

```

Console X
<terminated> Sri [Java Application] C:\User
CAR IS HIGH COST
AUDI NEW MODEL

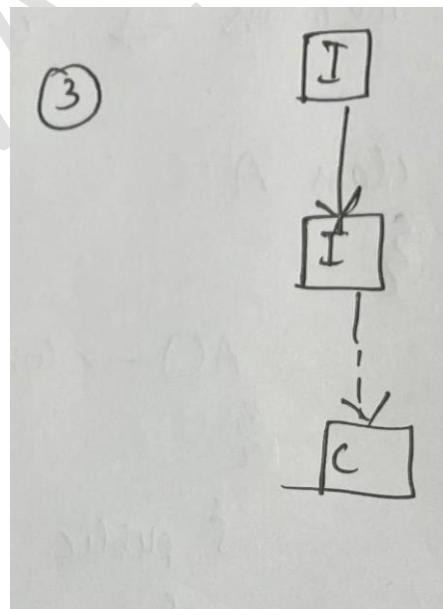
```

### EXAMPLE:-3

```

package practice;
interface VENU
{
    void Car();
}
interface BOBBY extends VENU
{
    void CarAudi();
}
class RAM implements BOBBY,VENU
{
    public void Car()
    {
        System.out.println("CAR IS WORKING");
    }
}

```



```

    }
public void CarAudi()
{
    System.out.println("AUDI IS NEW MODELS");
}
}

public class Armili
{
    public static void main(String args[])
    {
        BOBBY a1 = new RAM();
        a1.Car();
        a1.CarAudi();
    }
}

```

```

Console ×
<terminated> Armili [Java Application]
CAR IS WORKING
AUDI IS NEW MODELS

```

## MULTIPLE INHERITANCE THROUGH INTERFACE

---

```

package Sri;
interface RAM
{
    void M1();
}
interface SRI
{
    void M2();
}
interface VENKI extends RAM,SRI
{

```

```

}

class SRINU implements VENKI
{
    public void M1()
    {
        System.out.println("M1 IS METHOD");
    }

    public void M2()
    {
        System.out.println("M2 IS METHOD");
    }
}

public class Venu
{
    public static void main(String args[])
    {
        VENKI a1 = new SRINU();
        a1.M1();
        a1.M2();
    }
}

```

```

Console X
<terminated> SriDurga [Java Application] C:\Users\kssvi\p2\poo
M1 IS METHOD
M2 IS METHOD

```

## WHY MULTIPLE INHERITANCE POSSIBLE THROUGH INTERFACE?

*Through interface there is no ambiguity problem even though both the parents have same method because when class is going to implements those methods it will implements only once. Because in one class we can not have multiple methods same & arguments. Because it leads to duplicate method error.*

package Sri;

interface RAM

```

{
    void M1();
}
interface SRI
{
    void M2();
}
interface VENKI extends RAM,SRI
{
}

class SRINU implements VENKI
{
    public void M1()
    {
        System.out.println("M1 IS METHOD");
    }
}
public class Srikar
{
    public static void main(String args[])
    {
        VENKI a1 = new SRINU();
        a1.M1();
    }
}

```



*In interface there is no possibility of constructor chaining problem. Because interface does not allows constructor. When therefore ambiguity & constructor chaining it is possible to achieve multiple inheritance.*

### PRACTISING PROGRAMS

- ✚ CREATE A PROJECT AS “PRACTISINGSESSION”
- ✚ CREATE A PACKAGE AS “MODULE\_1”

#### P-1: CREATE A CLASS MULTIPLE LOGIC

- ✚ TAKE AN INPUT FROM USER USING SCANEER CLASS
- ✚ IF USER IS 1 → EXECUTE SWAP OF TWO NUMBERS
- 2 → EXECUTE FIBNOCCI SERIES

- 3 → EXECUTE FACTORIAL  
 4 → EXECUTE MULTIPLICATION TABLE OF 5  
 ANY OTHER IP → PRINT AS NO LOGIC FOUND

HINT: USE SWITCH CASE STATEMENTS

```

package Module_1;
import java.util.*;
public class MultipleLogic
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter a Number");
    int n = sc.nextInt();

    switch (n)
    {
      case 1 : System.out.println("BEFORE SWAPPING");
      int a = 20,b = 10, c;
      c=a;
      a=b;
      b=c;
      System.out.println("The Value Of a:"+a+"/nthe value of b:"+b);
      break;

      case 2: System.out.println("FIBNICCO SERIES");
      int x = 0, y = 1,z;
      System.out.println(x+"");
      System.out.println(y+"");
      for (int i=1;i<=5;i++)
      {
        z=x+y;
        System.out.println(z+"");
        x=y;
      }
    }
  }
}
  
```

```

y=z;
}
break;
case 3: System.out.println("FACTROIAL");
    int fact = 1;
    for (int k=5;k>=1;k--)
    {
        fact = fact * k;
    }
    System.out.println("FACTROIAL OF 5:"+fact);
    break;
case 4: System.out.println("MUTIPLATION TABLE");
    for (int r=1;r<=10;r++)
    {
        System.out.println(5+"*"+r+"="+(5*r));
    }
    break;
default : System.out.println("NO LOGIC FOUND");
    break;
}
}
}

```

### OUTPUT:-1

```

Console X
<terminated> MultipleLogic [Java Application] C:\Users\kssvi
Enter a Number
1
BEFORE SWAPPING
The Value Of a:10\nthe value of b:20

```

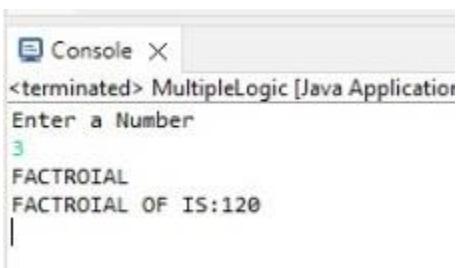
### OUTPUT:-2

```

Console X
<terminated> MultipleLogic [Java App]
Enter a Number
2
FIBNICO SERIES
0
1
1
2
3
5
8

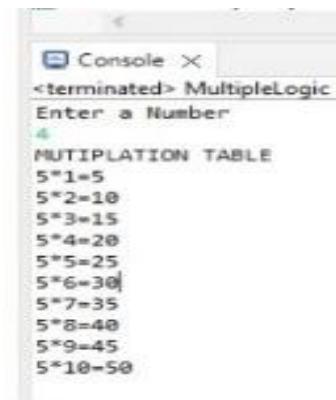
```

### OUTPUT:-3



```
Console X
<terminated> MultipleLogic [Java Application]
Enter a Number
3
FACTROIAL
FACTROIAL OF 3 IS:120
```

### OUTPUT:-4



```
Console X
<terminated> MultipleLogic
Enter a Number
5
MULTIPLICATION TABLE
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

## P-2: CREATE A CLASS MULTIPLE LOGIC\_2

CREATE FOLLOWING USER DEFINE METHODS

public static String is Prime(int num) → Develop prime number logic

public void is Armstrong(int num) → Develop armstrong logic

public String is Palindrome(int num) → Develop palindrome logic

CREATE MAIN() & MAKE A CALL TO USERDEFINE METHODS.

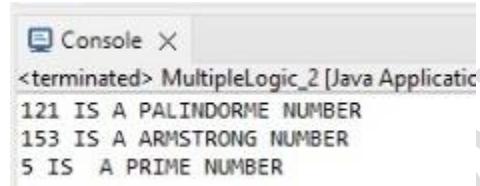
```
package Module_2;
public class MultipleLogic_2
{
    public static String prime(int n)
    {
        int count=0;
        for(int i=1; i<=n; i++)
        { if(n%i==0)
            {
                count++;
            }
        }
    }
}
```

```
        }
    }
if(count==2)
{
    return (n+" IS A PRIME NUMBER");
}
else
    return (n + " IS NOT A PRIME NUMBER");
}

public void Armstrong(int num)
{
    int rem,arm=0,temp=num;
    while(num>0)
    {
        rem=num%10;
        num=num/10;
        arm=arm+(rem*rem*rem);
    }
    if(arm==temp)
        System.out.println(temp+" IS A ARMSTRONG NUMBER");
    else
        System.out.println(temp+" IS NOT A ARMSTRONG NUMBER");
}

public String palendromic(int num)
{
    int rem,rev=0,temp=num;
    while(num>0)
    {
        rem=num%10;
        num=num/10;
```

```
        rev=rev*10+rem;
    }
if(temp==rev)
    return (temp+ "IS A PALINDORME NUMBER");
else
    return (temp+ " IS A NOT PALINDORME NUMBER");
}
public static void main (String args[])
{
    MultipleLogic_2 m1= new MultipleLogic_2();
    System.out.println( m1.palendromic(121));
    m1.Armstrong(153);
    System.out.println(m1.prime(5));
}
}
```



A screenshot of a Java application console window titled "Console X". The window shows the output of a program named "MultipleLogic\_2". The output consists of three lines of text: "121 IS A PALINDORME NUMBER", "153 IS A ARMSTRONG NUMBER", and "5 IS A PRIME NUMBER".

```
Console X
<terminated> MultipleLogic_2 [Java Application]
121 IS A PALINDORME NUMBER
153 IS A ARMSTRONG NUMBER
5 IS A PRIME NUMBER
```

## **MODULE:-4**

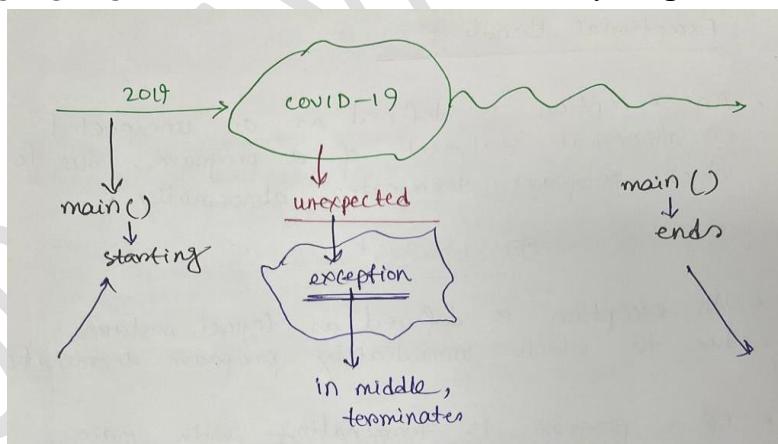
- OBJECT CLASS
- STRING CLASS
- ARRAY PROGRAMMING
- EXCEPTION HANDLING
- WRAPPER CLASSES
- COLLECTION FRAME WORK
- MAPS
- MULTI THREADING

### **EXCEPTIONAL HANDLING:-**

- An exception is defined as an unexpected (or) abnormal statement of a program, due to which program terminates abnormally.  
*(Or)*
- An exception is defined as logical mistake due to which immediately program terminates.
- If a program is terminating with main method, it is called as **Normal Termination**. But, in case if it is terminating anywhere in between it is called **Abnormal Termination**.

### **REAL TIME EXAMPLE:-**

- Everything is going well but COVID-19 came then everything went abnormally....



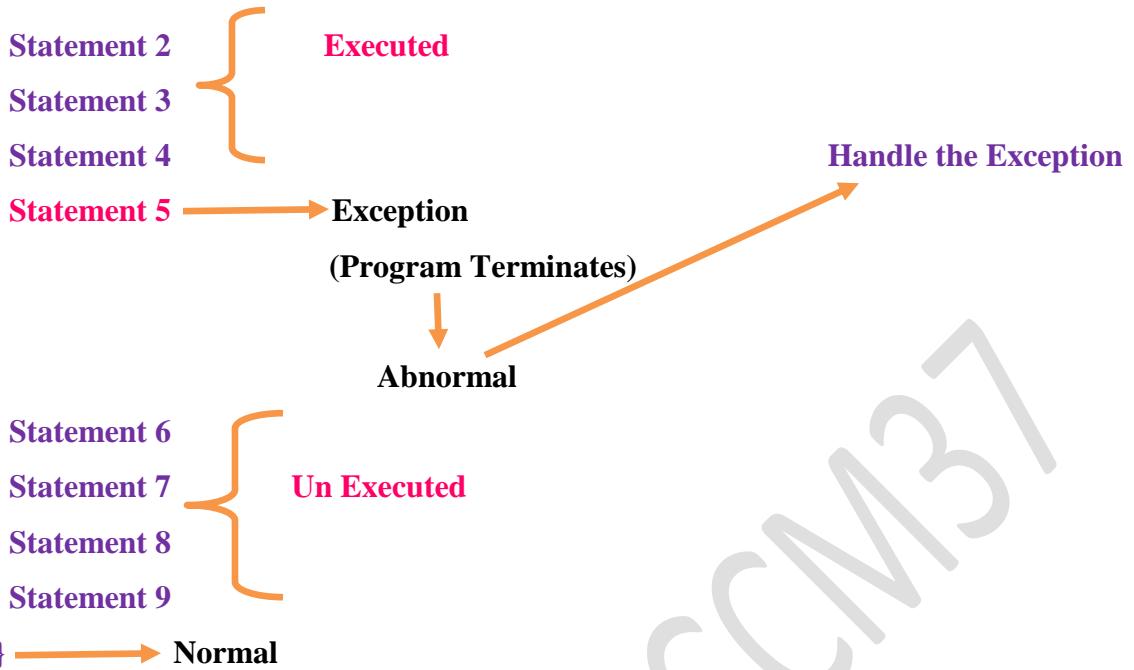
- Until exception occurs, everything will be executing. Once exception occurred, remaining part of a program will not be executed.

**Ex:-**

**class A**

{

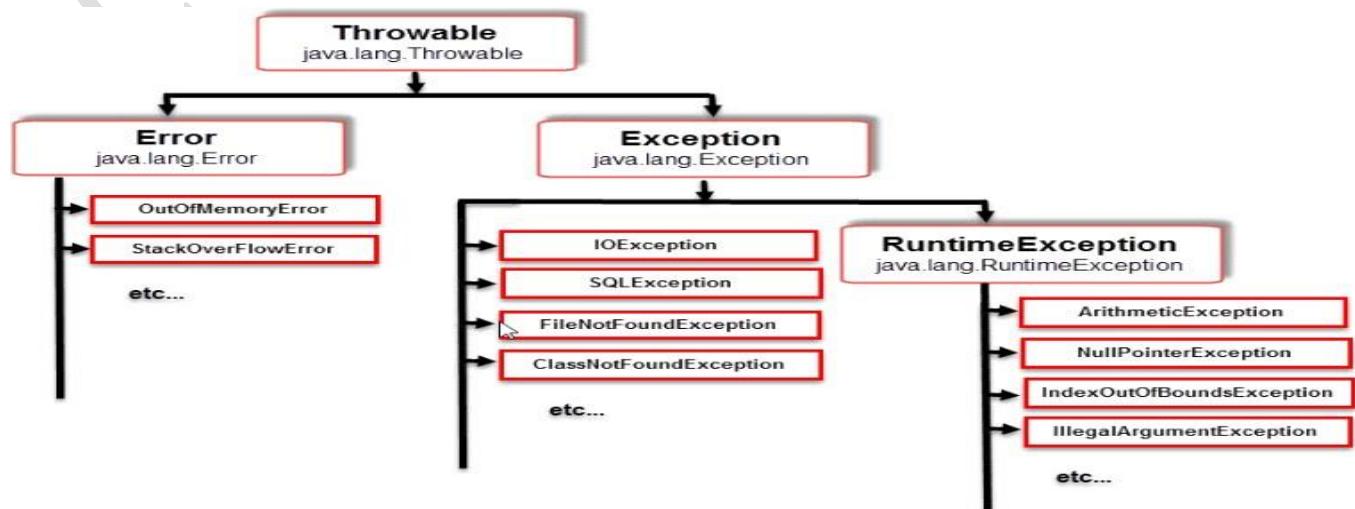
Statement 1



- ⊕ For executing all i.e. for normal termination, we should correct the exception (which is occurred for statement 5)
  - i.e. we handle the exception.
- ⊕ Once exception occurred, remaining program will not be executed untill we handle that exception.
- ⊕ In practically, we can not resolve the exception but we can find an alternative solution for that exception.
- ⊕ So, finding an alternate solution is called as **Exception Handling**.
- ⊕
- ⊕ In the above hierarchy, different class of exception is given, all these are pre-defined classes, which are present in **java.lang package**

## DIFFERENT STAGES OF EXCEPTION:-

### THROWABLE:-



- *Throwable class is a parent class for Error & Exception class.*

### **EXCEPTION:-**

- *Exception is a parent class of RuntimeException & all the checked type of Exception.*

### **RUN TIME EXCEPTION:-**

- *Run time exception is a parent class for all the checked type of Exception.*

### **TYPES OF EXCEPTION:**

Based on hierarchy, exception are divided into two types

- 1. CHECKED EXCEPTION**
- 2. UN-CHECKED EXCEPTION**

### **CHECKED EXCEPTION :-**

- *Exception which are checked (**identified or found out**) during compiletime by compiler, such type of exception are called as **Checked Exceptions**.*
- *Checked Exceptions are also called as **Compile time Exception**.*
- *For all checked Exception parent class is “Exception”*

### **EXAMPLES(CLASSES) OF CHECKED EXCEPTIONS ARE :-**

- *InterruptedException*
- *ClassNotFoundException*
- *SQLException*
- *FileNotFoundException*

### **UN-CHECKED EXCEPTION :-**

- *Exception which are checked (**identified or found out**) during Runtime or execution time, such type of exception are called as **Unchecked Exceptions**.*
- *Incase of Unchecked Exception our program will atleast compiles successfully.*
- *Unchecked Exceptions are also called as **Runtime Exceptions**.*
- *For all unchecked Exceptions parent class is “**RuntimeException**”*

### **EXAMPLES (CLASSES) OF CHECKED EXCEPTIONS ARE :-**

- *ArithmaticException*
- *ArrayIndexOutOfBoundsException*
- *NullPointerException*
- *StringIndexOutOfBoundsException*
- *ClassCastException*
- *NumberFormatException*

## WHAT HAPPENS WHEN EXCEPTION OCCURS:-

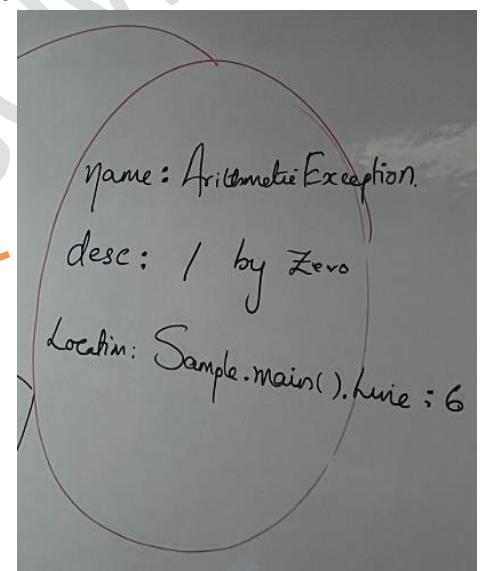
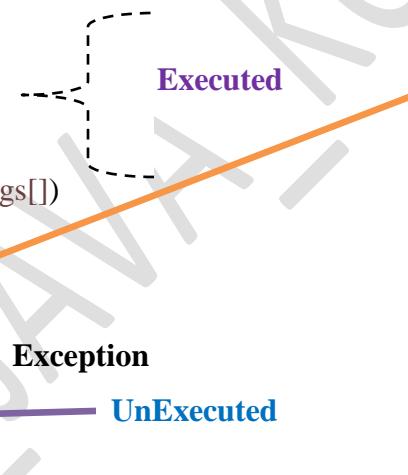
- Until exception occurs everything executed normally once exception occurred, an exception object gets created.
- Inside that object three things will be loaded name of an
  - 1. EXCEPTION
  - 2. DESCRIPTION
  - 3. LOCATION
- And that object will be hand over to JVM, now JVM looks for exception handling code. (Try / Catch / Block).
- In case JVM did not try / catch / block that exception object is hand over to default exception handler.
- Default exception handler is one of the predefined method of throwable class

## SYNTAX:-

PUBLIC VOID PRINTSTACKTRACE ()

Ex:

```
package Sravanthi;  
public class Sample  
{  
    public static void main(String args[])  
    {  
        int a = 10,b = 0,c;  
        c = a/b;  
        System.out.println(c);  
    }  
}
```



```
Console X  
<terminated> Sample [Java Application] C:\Users\kssvi.p2\pool\plugins\org.eclipse.justj.openjdk.ho  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at Sravanthi/Sravanthi.Sample.main(Sample.java:8)
```

## EXCEPTION CAN BE HANDLE BY USING TRY / CATCH BLOCK:-

### TRY BLOCK:-

Try block is used to keep a risky code (A code which causes Exception)

TRY  
{  
 // RISKY CODE  
}

## SYNTAX

*One Exception occurred in try block JVM will go to corresponding catch block. Therefore we should keep only risky code in try block & avoid writing normal code in try block.*

### CATCH BLOCK:-

*Catch block is used to caught the Exception in another words catch block is used to keep an alternate solution for the Exception.*

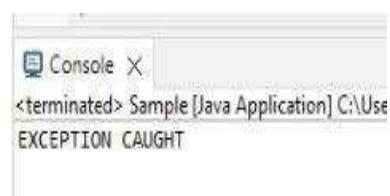
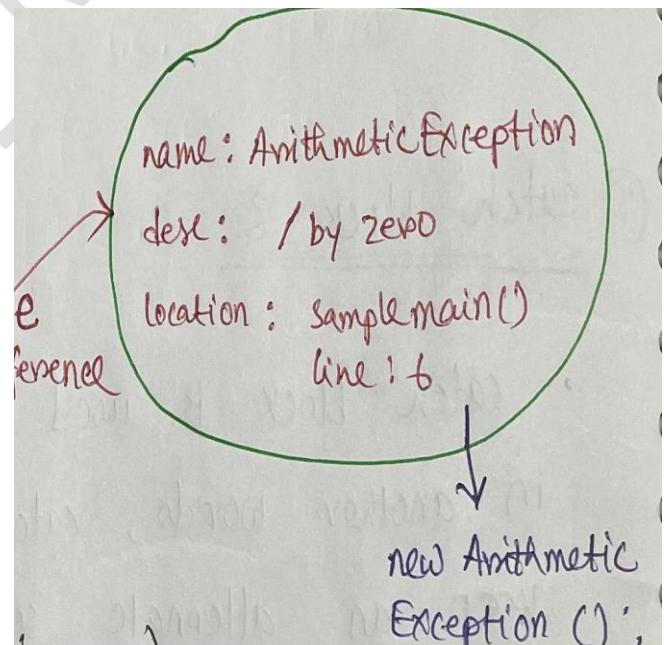
- In catch block we will also give reference for the exception object which is created in try block.

### SYNTAX:-

```
CATCH ( EXCEPTIONNAME REFERENCE VARIABLE)  
{  
    // ALTERNATE SOLUTION  
}
```

### PROGRAM ON EXCEPTION HANDLING:-

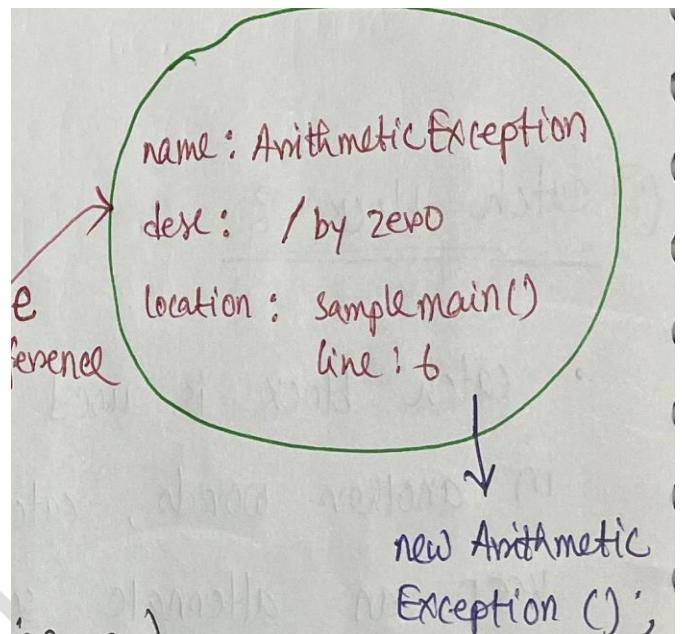
```
package Sri;  
  
public class Sample  
{  
    public static void main(String args[])  
    {  
        int a = 10,b = 0,c;  
        try  
        {  
            c = a/b;  
        }  
        catch(ArithmaticException e)  
        {  
            System.out.println("EXCEPTION CAUGHT");  
        }  
    }  
}
```



### Ex:-2

```
package Sri;  
public class Sample  
{  
    public static void main(String[] args)  
    {  
        System.out.println("MAIN START");  
        int a = 10,b = 0,c;  
        try  
        {  
            System.out.println("TRY START");  
            c = a/b;  
            System.out.println("TRY ENDS");  
        }  
        catch(ArithmaticException e)  
        {  
            System.out.println("CATCH BLOCK");  
        }  
        System.out.println("MAIN ENDS");  
    }  
}
```

```
Console X  
<terminated> Sample  
MAIN START  
TRY START  
CATCH BLOCK  
MAIN ENDS
```

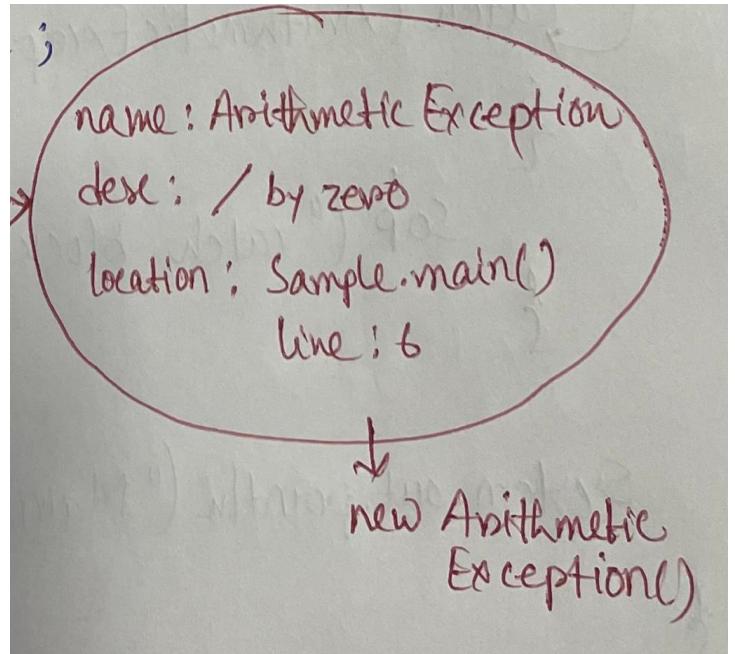


**Q. IN CASE IF WE DON'T KNOW WHAT TYPE OF EXCEPTION IS OCCURRING?  
WHICH CLASS REFERENCE WE WILL GIVE IN CATCH BLOCK?**

**A.** *In that situation, we will give the reference of parent class i.e. exception class, because parent class reference can handle any type of child object.*

### Example:-3

```
package Sri;  
public class Sample  
{  
    public static void main(String[] args)  
    {  
        int a = 10,b = 0,c;  
        try  
        {  
            c = a/b;  
        }  
        catch(Exception e)  
        {  
            System.out.println("EXCEPTION CATCH");  
        }  
    }  
}
```



```
Console X  
<terminated> Sample [Java Application] C:\Use  
EXCEPTION CAUGHT
```

EXCEPTION E = NEW ARITHMETIC EXCEPTION();

PARENT CLASS E = NEW CHILD CLASS();

→ UP-CASTED OBJECT

### Q.CAN WE WRITE SINGLE TRY WITH MULTIPLE CATCH BLOCK?

A. Yes, we can write single try with multiple catch block but the sequence must be specific catch block first, general catch block next.

Specific catch block can handle only one type of exception. General catch block can handle any type of exception.

## **SYNTAX:-**

```
TRY
{
}
CATCH (ARITHMETICEXCEPTION D)
{
}
CATCH (EXCEPTION D)
{
}
```

Only specific type of except

Any type of Exception

## **SINGLE TRY WITH MULTIPLE RISKY STATEMENT:-**

*In is possible that one try block can have multiple exception statements. Whichever exception occurred first only that exception will be handle. In case if we want, all the exceptions to be taken care then we should must write separate try & catch block for every exception.*

### **Example:-1**

```
package Exception;
class A
{
}
class B extends A
{
}
public class C
{
    public static void main(String[] args)
{
```

```

try
{
    int a = 100/0;
    B B1 = (B) new A ();
}

catch (ArithmaticException A)
{
    System.out.println("Exception Caught / by ZERO");
}

catch (ClassCastException A)
{
    System.out.println("ClassException Caught");
}

}
}

```

The screenshot shows a Java console window titled "Console X". The output text is:  
<terminated> C [Java Application] C:\Users\ks...  
Exception Caught / by ZERO

- ⊕ If we want both exception to be handle then we should write separate try & catch block like

### SINGLE PROGRAM WITH MULTIPLE TRY CATCH BLOCK:-

---

```

package Exception;
class A
{
}

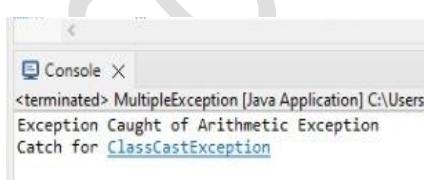
class B extends A
{
}

```

```
}
```

```
public class MultipleException
{
    public static void main(String[] args)
    {
        try
        {
            int a = 100/0;
        }
        catch (ArithmaticException A)
        {
            System.out.println("Exception Caught of Arithmatic Exception");
        }

        try
        {
            B B1 = (B) new A ();
        }
        catch (ClassCastException A)
        {
            System.out.println("Catch for ClassCastException");
        }
    }
}
```



```
Console X
<terminated> MultipleException [Java Application] C:\Users
Exception Caught of Arithmatic Exception
Catch for ClassCastException
```

**FINALLY BLOCK WILL BE EXECUTED IRRESPECTIVE OF**

- 
- 1. Exception did not occurred*

**2. Exception occurred & did not handle**

**3. Exception occurred & handle**

**NOTES:**

- ⊕ Finally block we can write only after try block or try catch block.
- ⊕ Finally block is used to keep the most important part of program which should be executed for sure.
- ⊕ For example, if we are using data base & fetching some records & in between exception occurs, in such situation if data base connection is not closed we will lose entire records.
- ⊕ Therefore, closing of data base logic we can keep in finally block so that it should be executed in every situation.

**EXCEPTION DID NOT OCCURRED:-**

```
package Exception;  
class A  
{  
}  
class B extends A  
{  
}
```

```
public class MultipleException
```

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

```
        try
```

```
{
```

**int a = 10/10; ➔ No Exception occur. So, no need of catch block**

```

        }
    catch (ArithmeticException A)
    {
        System.out.println("CATCH FOR ArithmeticException");
    }
    finally
    {
        System.out.println("FINALLY BLOCK");
    }
}

```

```

Console X
<terminated> MultipleException [Java Application]
FINALLY BLOCK

```

**EXCEPTION OCCURRED & DID NOT HANDLE:-**

```

package Exception;
class A
{
}

class B extends A
{
}

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

```

```

        int a = 10/0; ➔ No Exception occurred.
    }

/*catch (ArithmaticException A)
{
    System.out.println("CATCH FOR ArithmaticException");
}

finally
{
    System.out.println("FINALLY BLOCK");
}

}

```

```

Console X
<terminated> MultipleException [Java Application] C:\Users\kssvi\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jdk11\bin\java.exe
FINALLY BLOCK
Exception in thread "main" java.lang.ArithmaticException: / by zero
at Sravanthi/Exception.MultipleException.main(MultipleException.java:17)

```

### EXCEPTION OCCURRED & HANDLE:-

---

```

package Exception;
class A
{
}

class B extends A
{
}

public class MultipleException
{
}

```

```

public static void main(String[] args)
{
    try
    {
        int a = 10/0; ➔ Exception occurred
    }

    catch (ArithmaticException A) ➔ Exception Handled
    {
        System.out.println("CATCH FOR ArithmaticException");
    }

    finally
    {
        System.out.println("FINALLY BLOCK");
    }
}

```

```

Console X
<terminated> MultipleException [Java Application] C:\V
CATCH FOR ArithmaticException
FINALLY BLOCK

```

### THROW KEYWORD:-

- ✚ Throw keyword is used to create an exception object explicitly i.e. naturally exception is not occurring but we want an exception to occur based on our choice.

### SYNTAX:-

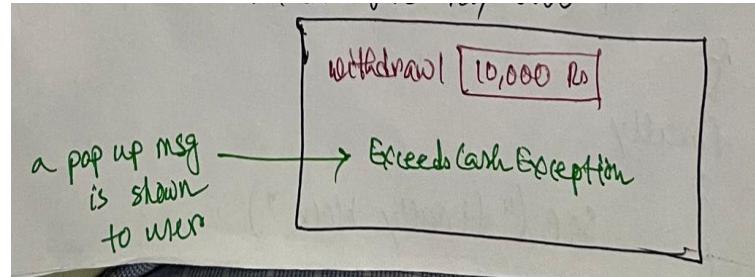
**THROW NEW EXCEPTIONNAME (DESCRIPTION);**

- ✚ Throw keyword is mainly used for operating a user defined exception
- ✚ (If we want immediate termination from current working like in ATM ➔ We use our wish able exception)

↓  
Intentionally, we want

## To create an exception

Suppose as an example in ATM, ATM can give only 5000



### Ex:-1

```
package Voting;  
public class Throwvoting  
{  
    public static void voting(int age)  
    {  
        if (age<=18)  
            throw new ArithmeticException("NOT ELIGIBLE TO VOTE");  
        else  
            System.out.println("eligible to vote");  
    }  
    public static void main(String[] args)  
    {  
        voting(15);  
    }  
}
```

```
Console X  
<terminated> Throwvoting (1) [Java Application] C:\Users\kssvi\p2\pool\plugins\org.eclipse.justj.openjdk.h  
Exception in thread "main" java.lang.ArithmeticException: NOT ELIGIBLE TO VOTE  
at practice/Voting.Throwvoting.voting(Throwvoting.java:7)  
at practice/Voting.Throwvoting.main(Throwvoting.java:14)
```

### Ex:-2

```
package Voting;  
public class Throwvoting  
{
```

```

public static void voting(int age)
{
    if (age<=18)
        throw new ArithmeticException("NOT ELIGIBLE TO VOTE");
    else
        System.out.println("eligible to vote");
}

public static void main(String[] args)
{
    voting(35);
}
}

```

```

Console X
<terminated> Throwvoting (1) [Java Application]
ELIGIBLE TO VOTE

```

### **THROWS KEYWORD:-**

- *Throws keyword is used to transfer exception from one method to another method.*
- *We will use throws keyword when current method cannot handle exception & transferring the exception to the caller of that method.*
- So, now it is responsibility of a caller to handle the exception.*
- *This process where exception is transferring from one method is called as Exception propagation.*
- *So, whenever exception, we have two choices*
  1. **WRITE TRY CATCH BLOCK**
  2. **USE THROWS KEYWORD**

### **SYNTAX:-**     METHODHEADER   METHODSIGNATURE   THROWS   EXCEPTIONSNAME

#### **Ex:-1**

```

public class Throwvoting
{
    public static void voting(int age) throws ArithmeticException

```

```

{
    if (age<18)
        throw new ArithmeticException("NOT ELIGIBLE TO VOTE");
    else
        System.out.println("eligible to vote");

}

public static void main(String[] args)
{
    try
    {
        voting(15);
    }
    catch(ArithmeticException e)
    {
        System.out.println("EXCEPTION CAUGHT");
    }
}

```

```

Console ×
<terminated> Throwvoting (1) [Java Application]
EXCEPTION CAUGHT

```

### **EXAMPLE FOR PRE DEFINED EXCEPTION:-**

#### **NullPointerException**

```
String s = null;
```

```
System.out.println (s.length());
```

#### **ArrayIndexOutOfBoundsException**

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

```
a [ 3 ] = 500;
```

### **StringIndexOutOfBoundsException**

```
String s = "java";
s.charAt(5);
```

### **ClassCastException**

```
Child c = (Child) new Parent();
```

### **InterruptedException**

```
Thread.sleep(5000);
```

### **OBJECT CLASS:-**

- ⊕ It is a pre-defined class which is present in java.lang Package.
- ⊕ Object class is a default parent class to the user define as well as pre-define class of java.

### **WHAT WE WRITE**

```
class A
```

```
{  
}  
}
```

### **WHAT ACTUALLY IT IS**

```
class A extends object
```

```
{  
}  
}
```

### **MEMBERS OF OBJECT CLASS:-**

- ⊕ getClass () : class → Final → Collection Frame Work
- ⊕ object1.equals(object2) : Boolean → String
- ⊕ toString () : String → Return Type
- ⊕ hashCode () : int → Return Type
- ⊕ notify () : void → Final
- ⊕ notifyAll () : void → Final
- ⊕ wait () : void → Final
- ⊕ wait(Long timeout) : void → Final
- ⊕ wait(Long timeout, int nanos) : void → Final
- ⊕ finalise () : void → String Program
- ⊕ clone () : void → Advance Java



Multi-Threading

### **toString () : String :-**

- + It is a method of object class
- + It return complete information of an object



**Packagenane.classname@object address**

**Example:-** myPackage.Sample@ZZh33453

**SYNTAX:-**

```
Public String toString ( )
{
    // return packagename.classname@objectaddress;
}
```

- + Object address is unique hexadecimal representation of hashCode of an object.
- + It cannot be same for two objects.
- + Whenever we print reference variable, internally it calls to `toString ()`.

**hashCode () : int :-**

- + It is a method of object class. It prints the hashCode number for given object.
- + HashCode number is 32 bit integer number
- + It is a unique number allocated to every object by JVM
- + If the object address are same, they will have same hashCode number or vice versa.

**SYNTAX:-**

```
public int hashCode ( )
{
    // return hashCode of object ;
}
```

### **PROGRAM OF TOSTRING ()**

**package** OBJECTCLASSEMTHODS;

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

```

{
    A a1 = new A();
    String s = a1.toString();
    System.out.println(s);
}
}

```

```

Console X
<terminated> A [Java Application] C:\User
OBJECTCLASSEMTHODS.A@2752f6e2

```

### **NOTE:-**

*Whenever we point reference variable, internally it will make a call to toString ()*

### **PROGRAM OF TOSTRING ()**

```

package OBJECTCLASSEMTHODS;
public class A
{
    public static void main(String[] args)
    {
        A a1 = new A();
        String s1 = a1.toString();
        A a2 = new A();
        String s2 = a2.toString();
        System.out.println(s1);
        System.out.println(s2);
    }
}

```

```

Console X
<terminated> A [Java Application] C:\Users\kss
OBJECTCLASSEMTHODS.A@2752f6e2
OBJECTCLASSEMTHODS.A@e580929
|

```

### **PROGRAM OF TOSTRING ()**

```

package OBJECTCLASSMETHODS;
public class A
{
    public static void main(String[] args)
    {
        A a1 = new A();
        //String s1 = a1.toString();
        A a2 = new A();
        //String s2 = a2.toString();
        System.out.println(a1); // internally call a1.toString();
        System.out.println(a2); // internally call a2.toString();
    }
}

```

The screenshot shows a Java console window titled "Console". The output is as follows:

```

Console X
<terminated> A [Java Application] C:\Users\DELL\IdeaProjects\PROJECTNAME\src\main\java\OBJECTCLASSMETHODS\A.java:10
OBJECTCLASSMETHODS .A@2752f6e2
OBJECTCLASSMETHODS .A@27f674d

```

### **CONCLUSION:-**

*The output we are getting from `toString()` there is no use of it in practical because if class name changes or package changes, it will effect our output to overcome this we have an option of overriding.*

### **Ex:-1**

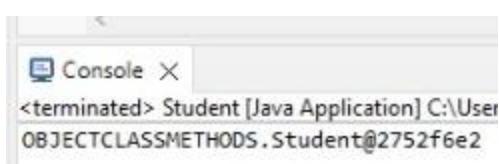
```

package OBJECTCLASSMETHODS;
public class Student
{
    String name; // Non - Static variables
    Student(String name)
    {
        this.name = name;
    }
    public static void main(String[] args)
    {
        Student s1 = new Student ("TARUN");
        String s = s1.toString();
        System.out.println(s);
    }
}

```

So, we use Constructor

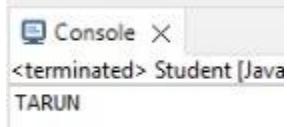
```
}
```



```
Console X
<terminated> Student [Java Application] C:\User
OBJECTCLASSEMTHODS.Student@2752f6e2
```

### Ex:-2

```
-----  
package OBJECTCLASSEMTHODS;  
  
public class Student  
{  
    String name;  
    Student(String name)  
    {  
        this.name = name;  
    }  
    // @overriding to toString()  
    public String toString() → In background (i.e. in Parent class) toString( ) consists package..  
    {  
        So, now as child class toString( ) is override that  
        return name;  
    }  
    public static void main(String[] args)  
    {  
        Student s1 = new Student ("TARUN");  
        String s = s1.toString();  
        System.out.println(s);  
    }  
}
```



```
Console X
<terminated> Student [Java]
TARUN
```

**-CREATE A CLASS AS PRODUCT**

**-DECLARE INSTANCE VARIABLES AS**

 Name, Color, Type

**-INITIALISE THEM USING CONSTRUCTOR**

**-OVER RIDE TOSTRING () & RETRUN Name, Color, Type**

**-CREATE MAIN () & CALL TOSTRING ()**

```
package OBJECTCLASSEMTHODS;
```

```
public class Product
```

```
{
```

```
    String name,color,type;
```

```
    public Product (String name,String color,String type)
```

```
{
```

```
        this.name = name;
```

```
        this.color = color;
```

```
        this.type = type;
```

```
}
```

```
    public String toString()
```

```
{
```

```
        return name+" "+color+" "+type;
```

```
}
```

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

```
{
```

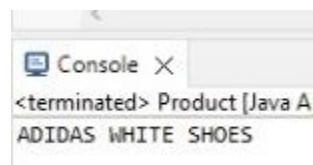
```
        Product p1 = new Product("ADIDAS","WHITE","SHOES");
```

```
        String s = p1.toString();
```

```
        System.out.println(s);
```

```
}
```

```
}
```

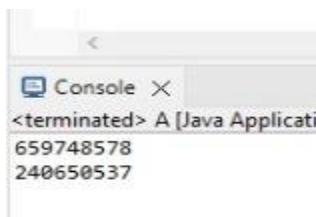


```
Console X
<terminated> Product [Java A]
ADIDAS WHITE SHOES
```

## PROGRAM OF HASHCODE()

Ex:-1

```
package OBJECTCLASSMETHODS;  
public class A  
{  
    public static void main(String[] args)  
    {  
        A a1 = new A();  
        int s1 = a1.hashCode();  
        A a2 = new A();  
        int s2 = a2.hashCode();  
        System.out.println(s1);  
        System.out.println(s2);  
    }  
}
```



*The output, we are getting from hashCode() there is no use of it in practical. To overcome this, we have option of Overriding.*

## PROGRAM OF HASHCODE()

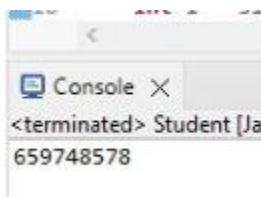
Ex:-2

```
package OBJECTCLASSMETHODS;  
public class Student  
{  
    int sid;  
    Student(int sid)  
    {  
        this.sid = sid;  
    }  
    /*@overriding to hashCode()  
    public int hashCode()
```

```

{
    return sid;
}
*/
public static void main(String[] args)
{
    Student s1 = new Student (45632179);
    int i = s1.hashCode();
    System.out.println(i);
}

```



### PROGRAM OF HASHCODE()

Ex:-3

```

package OBJECTCLASSEMTHODS;
public class Student
{
    int sid;
    Student(int sid)
    {
        this.sid = sid;
    }
    //@overriding to hashCode()
    public int hashCode()
    {
        return sid;
    }
    public static void main(String[] args)
    {

```

```

Student s1 = new Student (45632179);
int i = s1.hashCode();
System.out.println(i);
}
}

```

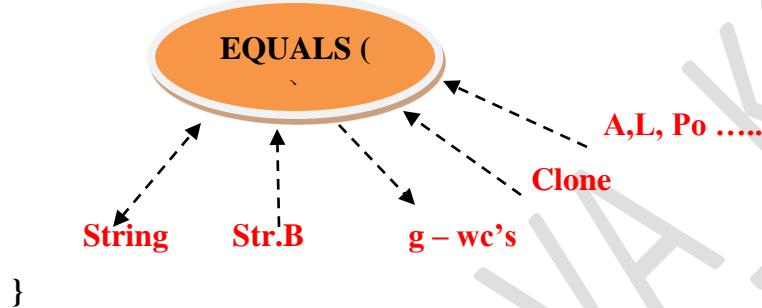
Console X  
<terminated> Student [Java]  
45632179

### WHY WE GO FOR OBJECT CLASS?

```

class Object
{

```



Suppose, equal ( ) is present in String, String.Buffer, Wrapper, clone..... Every class.

So, instead of studying all classes..... we should study only object class because object class contains 11 methods.

### MULTI-THREADING:-

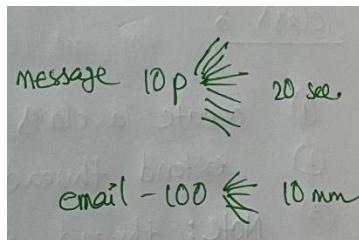
- + The process of executing more than one task parallelly, is called “Multi-Tasking”.
- + The process of executing two different paths of a program parallelly is called as **Thread Base Multi-Tasking**.
- + As an example, let's say we have 500 lines of code to execute, that JVM is taking 10 hrs of time. But, what we observe is if 250 & next 250 lines of code are independent on each other but then also next 250 lines will not be executed until 1<sup>st</sup> 250 lines completes its execution. Due to this execution time is more, efficiency is less.

Therefore to overcome that we can use **Thread Base Multi-Tasking**.

- For the above situation we can create a thread for first 250 lines & another thread for next 250 lines. And run both the threads parallelly with this we can reduce the execution time & increase the efficiency.

### **EXAMPLE:-**

If any user want to send a message to the 10 people then he will send that message by creating a group within 20 sec.



### **THREAD:-**

Thread is defined as a light weight process (or) A thread is a small part of our program.

- Creating multiple thread & executing them parallelly is called **Multi-Threading**.
- The main purpose of multi-threading is reducing the execution time & increasing the efficiency.
- A thread can be created in two ways
  - BY EXTENDING THREAD CLASS
  - BY IMPLEMENTING RUNNABLE INTERFACE.

### **PROCEDURE TO CREATE A THREAD BY EXTENDING THREAD CLASS:-**

- Create a class
- Extend thread class
- Note:- Thread is a pre-define class belongs to `java.lang` package
- Override public void run () of thread class
- develop some code in run ()
- run () is actually a our thread
- Start a thread by calling start () of thread class

### **PROCEDURE TO CREATE A THREAD BY IMPLEMENTING RUNNABLE INTERFACE:-**

- Create a class
- Implementing thread class

3. Note:-Runnable is pre-define interface belongs to `java.lang` package.
4. Override public void run () of runnable interface.
5. Develop some code in run ()
6. Run () is actually our thread
7. Create object of thread class
8. Start a thread by calling start () of thread class

### **EXAMPLE:-1**

```

package practice;
class Sample extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("SAMPLE THREAD");
        }
    }
}

public class UserLogic
{
    public static void main(String[] args)
    {
        Sample s1 = new Sample();
        s1.start(); -----> Create a thread by calling run ()
        for(int i=1;i<=5;i++)
        {
            System.out.println("USER LOGIC THREAD");
        }
    }
}

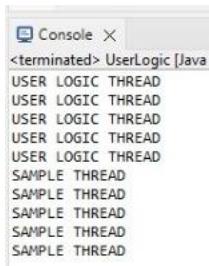
```

The diagram shows the inheritance path from the `java.lang` package to the `run()` and `start()` methods. It also highlights the creation of threads using the `start()` method and the logic contained within the `run()` method.

```
}
```

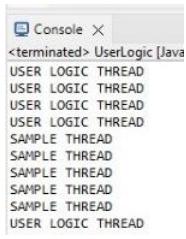
```
}
```

## OUT PUT:-1



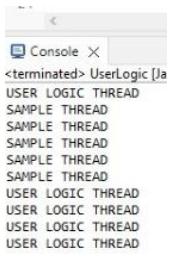
```
Console <terminated> UserLogic [Java]
USER LOGIC THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
```

## OUT PUT:-2



```
Console <terminated> UserLogic [Java]
USER LOGIC THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
```

## OUT PUT:-3



```
Console <terminated> UserLogic [Ja
USER LOGIC THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
USER LOGIC THREAD
USER LOGIC THREAD
USER LOGIC THREAD
USER LOGIC THREAD
```

- Thread is a pre-define class (Here it is parent class), present in java.lang package which has two methods.

### 1. PUBLIC VOID RUN ()

### 2. PUBLIC SYNCHRONISED VOID START ()

- Mythread is a child class here, want is-a-relationship with thread class.
- In this program, untill start () invokes there is only main () method under execution.
- Once start () method invoke, it internally creates a thread by calling run () .

Now, we have two thread



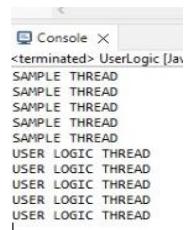
- Both the thread execute parallelly. Here, when 2 different parts are executing parallelly.  
We can not predict which one will execute first / last
- The decision is taken by thread scheduler. Suppose, time taken for execution of Thread 1 + Thread 2 (parallelly)

### GIVING PRIORITY TO THREAD:-

```

package practice;
class Sample extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("SAMPLE THREAD");
        }
    }
}
public class UserLogic
{
    public static void main(String[] args) throws InterruptedException
    {
        Sample s1 = new Sample();
        s1.start(); // Create a thread by calling to run & make it ready for execution.
        Thread.sleep(2000); // Current thread (Main()) goes to sleeping state for 2 sec
        for(int i=1;i<=5;i++)
        {
            System.out.println("USER LOGIC THREAD");
        }
    }
}

```



## REAL TIME EXAMPLE:-

When we click chrome browser.....it will take about 2 sec to open ( Depends on Internet Speed )



Actually here JVM will do **Thread.sleep(2000);**

## CREATING THREAD BY IMPLEMENTING RUNNABLE INTERFACE:-

```
package practice;

class Sample implements Runnable
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("SAMPLE THREAD");
        }
    }
}

public class UserLogic
{
    public static void main(String[] args) throws InterruptedException
    {
        Sample s1 = new Sample();
        Thread t1 = new Thread(s1);
        t1.start();
        for(int i=1;i<=5;i++)
        {
            System.out.println("USER LOGIC THREAD");
        }
    }
}
```

```
Console X
<terminated> UserLogic [Java]
USER LOGIC THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
SAMPLE THREAD
```

## ACCESS SPECIFIERS:-

- ⊕ It provides access permission to various fields like **class, method, variables, constructors & interface.**
- ⊕ We have **Four access specifiers**
  1. **Public**
  2. **Private**
  3. **Protected**
  4. **Default (No Modifier)**

### PUBLIC: *public fields are accessible*

- ⊕ Within the class
- ⊕ Within subclass of same package
- ⊕ Within other class of same package
- ⊕ Within subclass of another package but after writing import statement
- ⊕ Within nonsubclass of another package but after writing import statement

### PRIVATE: *private fields are accessible only within the class*

- ⊕ It is applicable for **method, variables & constructor only.**
- ⊕ Private members are accessible within the class
- ⊕ They are not accessible outside of the class
- ⊕ Private is only applicable for **method, variables & constructor**
- ⊕ Private is not applicable for class, why because class loader will loads all the static members of class, if we keep as private → class loader can't load

### PROTECTED:- *protected fields are accessible*

- ⊕ Within the class
- ⊕ Within subclass of same package
- ⊕ Within other class of same package
- ⊕ Within subclass of another package but after writing import statement
- ⊕ It is applicable with **method, variables & constructors**

### DEFAULT: *if we did not mention public, private, protected the access modifier is default*

- Within the class
- Within subclass of same package
- Within other class of same package

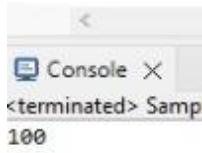
---

## PUBLIC:

### WITHIN THE SAME CLASS

---

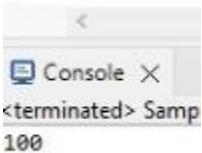
```
package abc;  
public class Sample  
{  
    public static int i = 100;  
    public static void main(String[] args)  
    {  
        System.out.println(i);  
    }  
}
```



### WITHIN SUBCLASS OF SAME PACKAGE

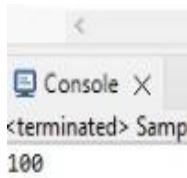
---

```
package abc;  
public class Sample1 extends Sample  
{  
    public static void main(String[] args)  
    {  
        System.out.println(i);  
    }  
}
```



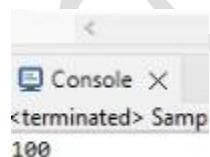
### WITHIN NON SUBCLASS OF SAME PACKAGE

```
-----  
package abc;  
  
public class Sample2  
{  
    public static void main(String[] args)  
    {  
        System.out.println(Sample.i);  
    }  
}
```



### WITHIN SUBCLASS OF ANOTHER PACKAGE

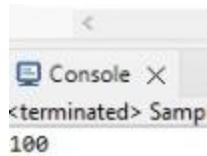
```
-----  
package def;  
import abc.Sample;  
  
public class Demo extends Sample  
{  
    public static void main(String[] args)  
    {  
        System.out.println(i);  
    }  
}
```



### WITHIN NONCLASS OF ANOTHER PACKAGE

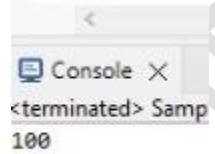
```
-----  
package def;  
import abc.Sample;
```

```
public class Demo1
{
    public static void main(String[] args)
    {
        System.out.println(Sample.i);
    }
}
```



#### **PRIVATE:-**

```
package abc;
public class Sample
{
    private static int i = 100;
    public static void main(String[] args)
    {
        System.out.println(i);
    }
}
```

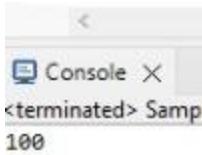


#### **PROTECTED:-**

##### **WITHIN THE SAME CLASS**

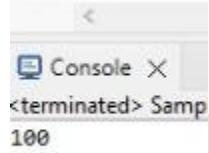
```
package abc;
public class Sample
{
    protected static int i = 100;
```

```
public static void main(String[] args)
{
    System.out.println(i);
}
```



### WITHIN SUBCLASS OF SAME PACKAGE

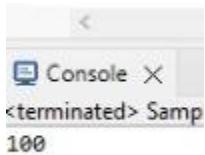
```
package abc;
public class Sample1 extends Sample
{
    public static void main(String[] args)
    {
        System.out.println(i);
    }
}
```



### WITHIN NON SUBCLASS OF SAME PACKAGE

```
package abc;
public class Sample2
{
    public static void main(String[] args)
    {
        System.out.println(Sample.i);
    }
}
```

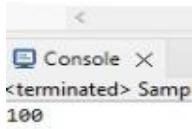
```
}
```



```
Console <terminated> Samp
100
```

### WITHIN SUBCLASS OF ANOTHER PACKAGE

```
package def;
import abc.Sample;
public class Demo extends Sample
{
    public static void main(String[] args)
    {
        System.out.println(i);
    }
}
```



```
Console <terminated> Samp
100
```

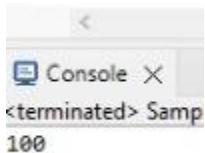
### WITHIN NON SUBCLASS OF DIFFERENT PACKAGE NOT ACCESSIBLE

**DEFAULT:**

### WITHIN THE SAME CLASS

```
package abc;
public class Sample
{
    static int i = 100;
    public static void main(String[] args)
    {
        System.out.println(i);
    }
}
```

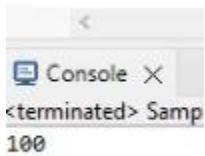
```
}
```



A screenshot of a Java console window titled "Console". The window shows the text "<terminated> Samp" and "100" on separate lines.

### WITHIN SUBCLASS OF SAME PACKAGE

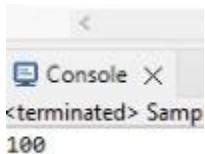
```
package abc;  
public class Sample1 extends Sample  
{  
    public static void main(String[] args)  
    {  
        System.out.println(i);  
    }  
}
```



A screenshot of a Java console window titled "Console". The window shows the text "<terminated> Samp" and "100" on separate lines.

### WITHIN NON SUBCLASS OF SAME PACKAGE

```
package abc;  
public class Sample2  
{  
    public static void main(String[] args)  
    {  
        System.out.println(Sample.i);  
    }  
}
```



A screenshot of a Java console window titled "Console". The window shows the text "<terminated> Samp" and "100" on separate lines.

- WITHIN SUBCLASS OF ANOTHER PACKAGE NOT ACCESSIBLE
- WITHIN NON SUBCLASS OF DIFFERENT PACKAGE NOT ACCESSIBLE

	WITH SAME PACKAGE			IN DIFFERENT PACKAGE	
	Withinclass	Subclass	NonSubclass	Subclass	NonSubclass
Public	Yes	Yes	Yes	Yes	Yes
Private	Yes	No	No	No	No
Protected	Yes	Yes	Yes	Yes	Yes
Default	Yes	Yes	Yes	No	No

### ARRAY PROGRAMMING:-

- An array is defined as an object which stores group of homogenous value in continuous manner.

(Or)

An array is defined as collection of homogenous values.

- Every value in array object will have an index position starting from 0 (0,1,2,3,4,...)
- An array represented with [ ] square bracket.
- Every array object is of finite size.
- When creating an array object, we should compulsory give the size.
- Size should must be numeric value.
- We can stores the value only upto given size if we crosses the size of an array, we will get *ArrayIndexOutOfBoundsException*.

### LENGTH VARIABLE:-

Length is a variable which returns number of values started in an array (Size of an array).

### USING ARRAY:-

For using array we need to follows three steps

#### 1. ARRAY CREATION

## 2. ARRAY INITIALISATION

### 3. ARRAY UTILIZATION

#### ARRAY CREATION:-

**SYNTAX:-**

**SIZE = SIZE OF AN ARRAY**

**ARRAY TYPE    ARRAY NAME [ ]    =    NEW ARRAY TYPE [SIZE];**

(Or)

**ARRAY TYPE [ ] ARRAY NAME = NEW ARRAY TYPE [SIZE OF ARRAY];**

#### ARRAY INITIALISATION:-

**SYNTAX:-**

**ARRAY NAME [INDEX] = VALUE;**

#### ARRAY UTILIZATION:-

**SYNTAX:-**

**System.out.println (Array name[index]);**

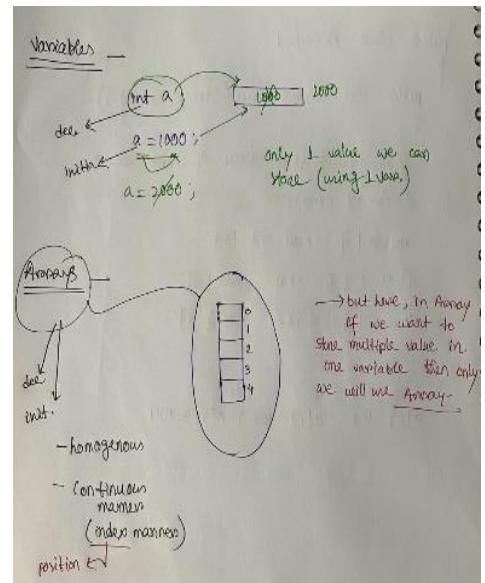
**NOTE:** We can declare & initialize our array in single statement also

**SYNTAX:-**

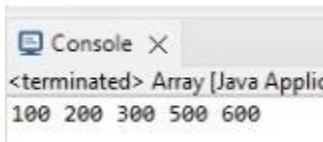
**ARRAY TYPE ARRAY NAME = {VALUE-1, VALUE-2, VALUE-3....VALUE-N}**

#### EXAMPLE:-1

```
package practice;
public class Array
{
    public static void main(String[] args)
    {
        int a [] = new int [5];
        a[0]= 100;
        a[1]= 200;
        a[2]= 300;
        a[3]= 500;
        a[4]= 600;
    }
}
```



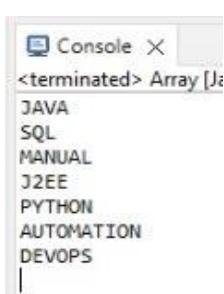
```
System.out.println(a[0]+" "+a[1]+" "+a[2]+" "+a[3]+" "+a[4]);
}
}
```



A screenshot of a Java IDE's console window titled "Console". The output shows the values of an array: "100 200 300 500 600".

**/\*WAP TO CERATE A STRING ARRAY OF SIZE 7 & PRINT ALL THE ARRAY VALUES\*/**

```
package practice;
public class Array1
{
    public static void main(String[] args)
    {
        String s [] = new String [7];
        s[0]= "JAVA";
        s[1]= "SQL";
        s[2]= "MANUAL";
        s[3]= "J2EE";
        s[4]= "PYTHON";
        s[5]= "AUTOMATION";
        s[6]= "DEVOPS";
        System.out.println(s[0]);
        System.out.println(s[1]);
        System.out.println(s[2]);
        System.out.println(s[3]);
        System.out.println(s[4]);
        System.out.println(s[5]);
        System.out.println(s[6]);
    }
}
```

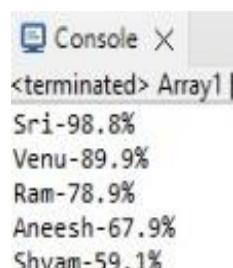


A screenshot of a Java IDE's console window titled "Console". The output shows the elements of a string array: "JAVA", "SQL", "MANUAL", "J2EE", "PYTHON", "AUTOMATION", and "DEVOPS".

**/\* WAP TO CERATE 2 ARRAY STRING & FLOAT OF SIZE 5. IN STRING ARRAYS  
ADD STUDENT NAME IN FLOAT ARRAY ADD THEIR PERCENTAGE & PRINT  
THEM TOGETHER (SIDE BY SIDE) \*/**

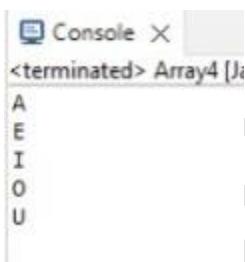
**Ex: STUDENT NAME ————— PERCENTAGE**

```
package practice;
public class Array2
{
    public static void main(String[] args)
    {
        String a [] = new String [5];
        a[0] = "Sri";
        a[1] = "Venu";
        a[2] = "Ram";
        a[3] = "Aneesh";
        a[4] = "Shyam";
        float f [] = new float [5];
        f[0] = 98.8f;
        f[1] = 89.9f;
        f[2] = 78.9f;
        f[3] = 67.9f;
        f[4] = 59.1f;
        System.out.println(a[0] + "-" + f[0] + "%");
        System.out.println(a[1] + "-" + f[1] + "%");
        System.out.println(a[2] + "-" + f[2] + "%");
        System.out.println(a[3] + "-" + f[3] + "%");
        System.out.println(a[4] + "-" + f[4] + "%");
    }
}
```



```
Console X
<terminated> Array1 |
Sri-98.8%
Venu-89.9%
Ram-78.9%
Aneesh-67.9%
Shyam-59.1%
```

```
/*WAP BY TASKING CHARACTER ARRAY, STORE VOWEL IN THAT ARRAY &
PRINT THEM USING FOR LOOP*/
package practice;
public class Array3
{
    public static void main(String[] args)
    {
        char Ch[] = new char[5];
        Ch[0] = 'A';
        Ch[1] = 'E';
        Ch[2] = 'T';
        Ch[3] = 'O';
        Ch[4] = 'U';
        for(int i=0;i<5;i++)
        {
            System.out.println(Ch[i]);
        }
    }
}
```



```
Console ×
<terminated> Array4 [Ja
A
E
I
O
U
```

```
/* WAP TO CREATE AN INTERGER ARRAY & PRINT THEM USING FOR LOOP*/
```

```
package practice;
public class Array4
{
    public static void main(String[] args)
    {
```

```
int a [] = {11,33,55,67,98,76};  
for (int i=0;i<a.length;i++)  
{  
    System.out.println(a[i]);  
}  
}
```

A screenshot of a Java IDE showing the output of the first program. The console window is titled 'Console' and shows the output: '11', '33', '55', '67', '98', and '76', each on a new line. The window title bar also says 'Console' and the status bar at the bottom says '<terminated>'.

/\* WAP TO REVERSE AN INTEGER ARRAY \*/

```
package practice;  
public class Array5  
{  
    public static void main(String[] args)  
    {  
        int a [] = {11,22,33,44,55,66};  
        int l = a.length-1;  
        for (int i=l;i>=0;i--)  
        {  
            System.out.println(a[i]);  
        }  
    }  
}
```

A screenshot of a Java IDE showing the output of the second program. The console window is titled 'Console' and shows the output: '66', '55', '44', '33', '22', and '11', each on a new line. The window title bar also says 'Console' and the status bar at the bottom says '<terminated>'.

```

/* WAP TO READ THE ARRAY VALUE FROM SCANNER CLASS & PRINT IT */

package practice;

import java.util.*;

public class Arrray6
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter Array Size");
        int size = s.nextInt(); // 5
        int a [] = new int [size]; // a[0] = 100 → Create Array
        for(int i=0;i<size;i++)
        {
            a [i] = s.nextInt(); // a[1] = 200
            // a[2] = 300
            // a[3] = 400
            // a[4] = 500
        }
        System.out.println("Array Value are:");
        for (int j=0;j<size;j++)
        {
            System.out.println(a[j]); → Printing
        }
        s.close(); → To Close Scanner Class
    }
}

```

```

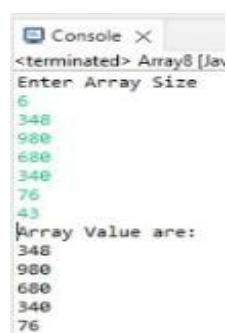
Console X
<terminated> Array7 [Ja
Enter Array Size
5
100
200
300
400
500
Array Value are:
100
200
300
400
500

```

**/\* WAP TO PRINT**  
**A. ONLY EVEN NUMBERS FROM THE ARRAY**  
**B. ONLY ODD NUMBERS FROM AN ARRAY**  
**BY READING THE ARRAY VALUE THROUGH SCANNER CLASS \*/**

**ONLY EVEN NUMBERS FROM THE ARRAY**

```
package practice;  
import java.util.*;  
public class Array7  
{  
    public static void main(String[] args)  
    {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter Array Size");  
        int size = s.nextInt();  
        int a [] = new int [size];  
        for(int i=0;i<size;i++)  
        {  
            a [i] = s.nextInt();  
        }  
        System.out.println("Array Value are:");  
        for (int j=0;j<size;j++)  
        {  
            if (a[j]%2==0)  
                System.out.println(a[j]);  
        }  
        s.close();  
    }  
}
```



```
Console >  
<terminated> Array7 [Java]  
Enter Array Size  
6  
348  
988  
688  
348  
76  
43  
Array Value are:  
348  
988  
688  
348  
76
```

## ONLY ODD NUMBERS FROM AN ARRAY

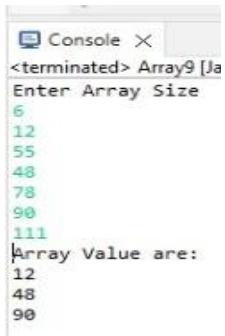
```
package practice;
import java.util.*;
public class Array7
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter Array Size");
        int size = s.nextInt();
        int a [] = new int [size];
        for(int i=0;i<size;i++)
        {
            a [i] = s.nextInt();
        }
        System.out.println("Array Value are:");
        for (int j=0;j<size;j++)
        {
            if (a[j]%2!=0)
                System.out.println(a[j]);
        }
        s.close();
    }
}
```

```
Console X
<terminated> Array8 [Ja
Enter Array Size
6
999
657
231
541
90
12
Array Value are:
999
657
231
541
```

```
/* WAP TO PRINT  
A. EVEN INDEX POSITION VALUE  
B. ODD INDEX POSITION VALUE */
```

### EVEN INDEX POSITION VALUE

```
package practice;  
import java.util.*;  
public class Array8  
{  
    public static void main(String[] args)  
    {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter Array Size");  
        int size = s.nextInt();  
        int a [] = new int [size];  
        for(int i=0;i<size;i++)  
        {  
            a [i] = s.nextInt();  
        }  
        System.out.println("Array Value are:");  
        for (int j=0;j<size;j++)  
        {  
            if (j%2==0)  
                System.out.println(a[j]);  
        }  
        s.close();  
    }  
}
```



```
Console <terminated> Array9 [Ja  
Enter Array Size  
6  
12  
55  
48  
78  
90  
111  
Array Value are:  
12  
48  
90
```

## ODD INDEX POSITION VALUE

```
package practice;
import java.util.*;
public class Array8
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter Array Size");
        int size = s.nextInt();
        int a [] = new int [size];
        for(int i=0;i<size;i++)
        {
            a [i] = s.nextInt();
        }
        System.out.println("Array Value are:");
        for (int j=0;j<size;j++)
        {
            if (j%2!=0)
                System.out.println(a[j]);
        }
        s.close();
    }
}
```

```
Console X
<terminated> Array9 [Ja
Enter Array Size
6
12
55
46
78
90
111
Array Value are:
55
78
111
```

```

/* WAP TO PRINT SUM & AVERAGE OF AN ARRAY BY READING THE ARRAY
VALUES THROUGH THE SCANNER CLASS */

package practice;
import java.util.*;
public class Array9
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter Array Size");
        int size = s.nextInt(),sum=0;
        int a [] = new int [size];
        for(int i=0;i<size;i++)
        {
            a [i] = s.nextInt();
        }
        System.out.println("Array Value are:");
        for (int j=0;j<size;j++)
        {
            sum = sum + a[j];
        }
        double avg = sum/size;
        System.out.println(sum+" "+avg);
        s.close();
    }
}

```

```

Console X
<terminated> Array10 [J
Enter Array Size
4
23
45
99
89
Array Value are:
256 64.0

```

```
/* WAP TO PRINT AVERAGE OF EVEN NUMBERS & ODD NUMBERS FROM THE  
GIVEN ARRAY */  
  
package practice;  
import java.util.*;  
public class Array10  
{  
    public static void main(String[] args)  
    {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter Array Size");  
        int size = s.nextInt(),sumE=0,sumO=0,countE=0,countO=0;  
        int a [] = new int [size];  
        for(int i=0;i<size;i++)  
        {  
            a [i] = s.nextInt();  
        }  
        System.out.println("Array Value are:");  
        for (int j=0;j<size;j++)  
        {  
            if (a[j]%2==0)  
            {  
                sumE = sumE+a[j];  
                countE++;  
            }  
            else  
            {  
                sumO = sumO+a[j];  
                countO++;  
            }  
        }  
        double avgE = sumE/countE;
```

```

        double avgO = sumO/countO;
        System.out.println(sumE+" "+avgE);
        System.out.println(sumO+" "+avgO);
        s.close();
    }
}

```

The screenshot shows a Java console window titled "Console X". It displays the output of a program named "Array11". The program asks for the array size (6), then prompts for six values: 12, 34, 56, 77, 9, and 4. It then prints "Array Value are:" followed by the array elements: 106 26.0 and 86 43.0.

```

Console X
<terminated> Array11 [Java]
Enter Array Size
6
12
34
56
77
9
4
Array Value are:
106 26.0
86 43.0

```

**/\* WAP TO PRINT MAXIMUM NUMBER FROM AN ARRAY**

**HINT : ASSUME FIRST NUMBER AS MAXIMUM**

**MAX = A [ 0 ]**

**COMPARE THAT WITH NEXT NUMBER IF YOU FIND ANY NEXT NUMBER**

**GREATEST THAN ASSUMED NUMBER ASSIGN IT \*/**

**IP → {12,33,45,1,78,20}**

**OP → 78**

**package practice;**

**public class Maximum**

**{**

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

**{**

**int a [] = {12,33,45,1,78,20};**

**int max = a [0];**

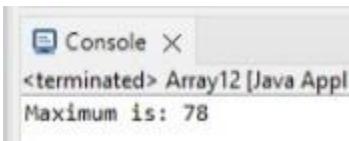
**for(int i=0;i<a.length;i++)**

**{**

```

        if(a[i]>max)
            max = a[i];
    }
    System.out.println("Maximum is: "+max);
}
}

```



**/\* WAP TO PRINT MINIMUM NUMBER FROM AN ARRAY**

**HINT : ASSUME FIRST NUMBER AS MINIMUM**

**MIN = A [ 0 ]**

**COMPARE THAT WITH NEXT NUMBER IF YOU FIND ANY NEXT NUMBER**

**GREATEST THAN ASSUMED NUMBER ASSIGN IT \*/**

**IP → {12,33,45,1,78,20}**

**OP → 1**

**package practice;**

**public class Minimum**

**{**

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

**{**

**int a [] = {12,33,45,1,78,20};**

**int min = a [0];**

**for(int i=0;i<a.length;i++)**

**{**

**if(a[i]<min)**

**min = a[i];**

**}**

**System.out.println("Minimum is: "+min);**

**}**

**}**

```
Console X
<terminated> Array12 []
Minimum is: 1
```

```
/* WAP TO PRINT MINIMUM & MAXIMUM NUMBER FROM GIVEN ARRAY */

package practice;

public class Array11

{
    public static void main(String[] args)

    {
        int a [] = {12,35,45,1,76,20};

        int max = a [0],min = a [0];
        for(int i=0;i<a.length;i++)

        {
            if(a[i]>max)
                max = a[i];

            else if(a[i]<min)
                min = a[i];
        }

        System.out.println("Maximum is: "+max);
        System.out.println("Minimum is: "+min);
    }
}
```

```
Console X
<terminated> Array13 []
Maximum is: 76
Minimum is: 1
```

```
/* WAP TO PRINT

    A. MAXIMUM & SECOND MAX FROM AN ARRAY
    B. MINIMUM & SECOND MIN FROM AN ARRAY */

    I/P → {12,33,45,1,76,20}
```

**MAXIMUM & SECOND MAX FROM AN ARRAY**

```

package practice;
public class SecondMaximum
{
    public static void main(String[] args)
    {
        int a [] = {10,21,30,45,4,25};
        int max = a [0],smax = a [1];
        for(int i=1;i<a.length;i++)
        {
            if(a[i]>max)
            {
                smax = max;
                max = a[i];
            }

            else if(a[i]>smax)
            {
                smax = a[i];
            }
        }

        System.out.println("Maximum is: "+max);
        System.out.println("Second Maximum is: "+smax);
    }
}

```

```

Console X
<terminated> Array14 [Java Appl]
Maximum is: 45
Second Maximum is: 30

```

## MINIMUM & SECOND MIN FROM AN ARRAY

```

package practice;
public class Array15
{

```

```

public static void main(String[] args)
{
    int a [ ] = {12,85,36,1,10,3};
    int min = a[0],smin = a[1];
    for(int i=1;i<a.length;i++)
    {
        if(a[i]<min)
        {
            smin = min;
            min = a[i];
        }

        else if(a[i]<smin)
        {
            smin= a[i];
        }
    }
    System.out.println("Minimum is: "+min);
    System.out.println("Second Minimum is: "+smin);
}
}

```

```

Console X
<terminated> Array15 [Java App]
Minimum is : 1
Second Minimum is : 3

```

**/\*WAP TO SORT THE ARRAY IN**  
**A. INCREASING ORDER**  
**B. DECREASING ORDER \*/**

**INCREASING ORDER**

```

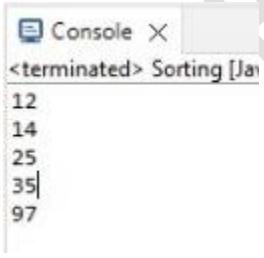
package practice;
public class SortIncreasing
{

```

```

public static void main(String[] args)
{
    int a[]={10,25,35,12,14,97};
    int temp;
    for (int i=0;i<a.length;i++)
    {
        for (int j=0;j<a.length-1;j++)
        {
            if (a[j]>a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1]= temp;
            }
        }
    }
    for (int k=0;k<a.length;k++)
    {
        System.out.println(a[k]);
    }
}

```



```

Console X
<terminated> Sorting [Java]
12
14
25
35
97

```

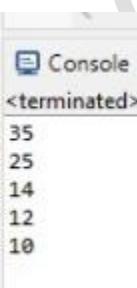
## DECREASING ORDER

```

package practice;
public class ScortDecreasing
{

```

```
public static void main(String[] args)
{
    int a[]={10,25,35,12,14,97};
    int temp;
    for (int i=0;i<a.length;i++)
    {
        for (int j=0;j<a.length-1;j++)
        {
            if (a[j]<a[j+1])
            {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1]= temp;
            }
        }
    }
    for (int k=0;k<a.length;k++)
    {
        System.out.println(a[k]);
    }
}
```



A screenshot of a Java IDE's console window. The title bar says "Console". The output area shows the following text:  
<terminated>  
35  
25  
14  
12  
10

```
/* WAP TO PRINT MAXIMUM, SECOND MAXIMUM, MINIMUM, SECOND  
MINIMUM FROM AN SORTED ARRAY */  
  
package array;  
  
public class Array11  
{  
    public static void main(String[] args)  
    {  
        int a[]={10,25,35,12,14,97};  
        int temp;  
        for (int i=0;i<a.length;i++)  
        {  
            for (int j=0;j<a.length-1;j++)  
            {  
                if (a[j]>a[j+1])  
                {  
                    temp = a[j];  
                    a[j] = a[j+1];  
                    a[j+1]= temp;  
                }  
            }  
        }  
        for (int k=0;k<a.length;k++)  
        {  
            System.out.println("MAXIMUM: "+a[a.length-1]);  
            System.out.println("SECOND MAXIMUM :" +a[a.length-2]);  
            System.out.println("MINIMUM :" +a[0]);  
            System.out.println("SECOND MINIMUM :" +a[1]);  
        }  
    }  
}
```

```
Console X
<terminated> Array1 (1) [J
MAXIMUM: 97
SECOND MAXIMUM :35
MINIMUM :10
SECOND MINIMUM :12
```

**/\* WAP TO DELETE DUPLICATE NUMBER FROM AN ARRAY \*/**

**package** practice;

**public class** DeleteDuplicate

{

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

{

**int** a [ ] = {2,3,4,2,3,1,1,5,6,5};

**int** temp;

**for** (**int** i=0;i<a.length;i++)

    {

**for** (**int** j=0;j<a.length-1;j++)

        {

**if** (a[j]>a[j+1])

            {

                temp = a[j];

                a[j] = a[j+1];

                a[j+1] = temp;

            }

        }

    }

**int** b [ ] = **new int**[a.length];

**int** k = 0;

**for** (**int** i=0;i<a.length-1;i++)

    {

**if**(a[i]!=a[i+1])

        {

            b[k] = a[i];

```

        k++;
    }
}
b[k] = a[a.length-1];
for (int j=0;j<=k;j++)
{
    System.out.println(b[j]);
}
}

```

```

Console X
<terminated> Delete
1
2
3
4
5
6

```

**/\* WAP TO PRINT PRIME NUMBER FROM AN ARRAY \*/**

```

package practice;
public class PrimeArray
{
    public static void isprime(int arr[])
    {
        for(int i=0;i<arr.length;i++)
        {
            int count=0;
            for (int j=1;j<=arr[i];j++)
            {
                if (arr[i]%j==0)
                    count++;
            }
            if(count==2)

```

```

        System.out.println(arr[i]+" IS A PRIME ");
    }
}

public static void main(String[] args)
{
    int a [] = {10,13,17,12,24,27};
    isprime(a);
    int b[] = {15,65,95,127,225,43};
    isprime(b);
    int c[] = {9,129,1001,2225,30};
    isprime(c);
}
}

```

```

Console X
<terminated> PrimeA1
13 IS A PRIME
17 IS A PRIME
127 IS A PRIME
43 IS A PRIME

```

**/\* WAP TO PRINT PALINDROME NUMBER FROM THE ARRAY USING USER**

**DEFINE METHOD \*/**

```

package practice;
public class PalindromeArray
{
    public static void main (String[] args)
    {
        int a [] = {10,111,121,125,55};
        ispalindrome(a);

        int b [] = {243432,1221,1664,5353};
        ispalindrome(b);
    }

    public static void ispalindrome(int a[])
    {

```

```

for(int i=0;i<a.length;i++)
{
    int rev=0,rem,temp=a[i];
    while(temp>0)
    {
        rem = temp% 10;
        temp = temp/10;
        rev = rev*10+rem;
    }
    if(rev==a[i])
        System.out.println(" IS A PALINDROME NUMBER : "+a[i]);
}
}

```

```

Console X
terminated> Array19 [Java Application]
IS A PALINDROME NUMBER : 111
IS A PALINDROME NUMBER : 121
IS A PALINDROME NUMBER : 55
IS A PALINDROME NUMBER : 1221

```

### **COMMAND LINE ARGUMENTS:-**

*Main method is containing one special type of arguments which is called as **command line arguments** i.e. during execution through command prompt we can enter sum inputs & those inputs will be stored in string type of array with the array name args*

#### **Example:-1**

```

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

```

```
}

}

C:\Users\kssvi\OneDrive\Desktop\K++KM>javac CommandLineArgs.java
C:\Users\kssvi\OneDrive\Desktop\K++KM>java CommandLineArgs 100 java True 55.65
100
java
True
55.65
```

### PASSING COMMAND LINE ARGUMENTS IN ECLIPSE:-

After completing logic click on run drop down select run configuration. Click on argument & provide values under program arguments, click on apply then click on run.

#### Example:-1

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

```
Consc
<terminat
100
CORE
JAVA
TRUE
55.90
```

#### Example:-2

```
package array;
public class ArraySum
{
    public static void main(String[ ] args)
```

```

{
    int sum=0;
    for(int i=0;i<args.length;i++)
    {
        int num = Integer.parseInt(args[i]);
        sum = sum + num;
    }
    System.out.println(sum);
}

```

Console  
<terminated>  
12678

### NOTE:-

Whatever argument we enter command prompt will be stored in string format if we want to convert string value to integer we have pre-define method as *Integer.parseInt*

### STRING PROGRAMMING:-

*String is the pre-defined class which is present in java.lang.package*

- ⊕ String class contains multiple pre-defined methods which is used to Manipulate / Modify string data.
- ⊕ We use those method, we have to create an object of string class
- ⊕ String class object can be created in two ways

### 1.USING NEW KEYWORD

**Ex:-** String s = new String (“Java”);

### 2.WITHOUT USING NEW KEYWORD

**Ex:-** String s = “Java”

### SOME OF THE IMPORTANT METHOD OF STRING CLASS

Method : returntype ----- Definition

1.length () : int – it return length of a string (no of character)

2.charAt(int index) : char – it return character at given index.

3.equals(String) : Boolean – it return true if string content matches

**4.equalsIgnoreCase(String) : Boolean** – it returns true string contents matches without Considering case sensitivity

**5.trim( ) : String** – it returns string after removing white spaces.

**6.subString(int startIndex):String** – it return subpart of a string

**7.subString(int startIndex,int endIndex) : String** – it returns subpart of a string from start From start index till end index-1.

**8.toUpperCase( ) : String** – convert string to upper case

**9.toLowerCase( ) : String** – convert string lower case

**10.indexOf(String) : int** – it returns index of given string

**11.indexOf(Char) : int** – it retuns index of given character

**12.indexOf(String,int from index) : int** – it returns index of given string but search it from given index.

**13.indexOf(Char,int fromindex) : int** – it returns index of given char but search it from given index.

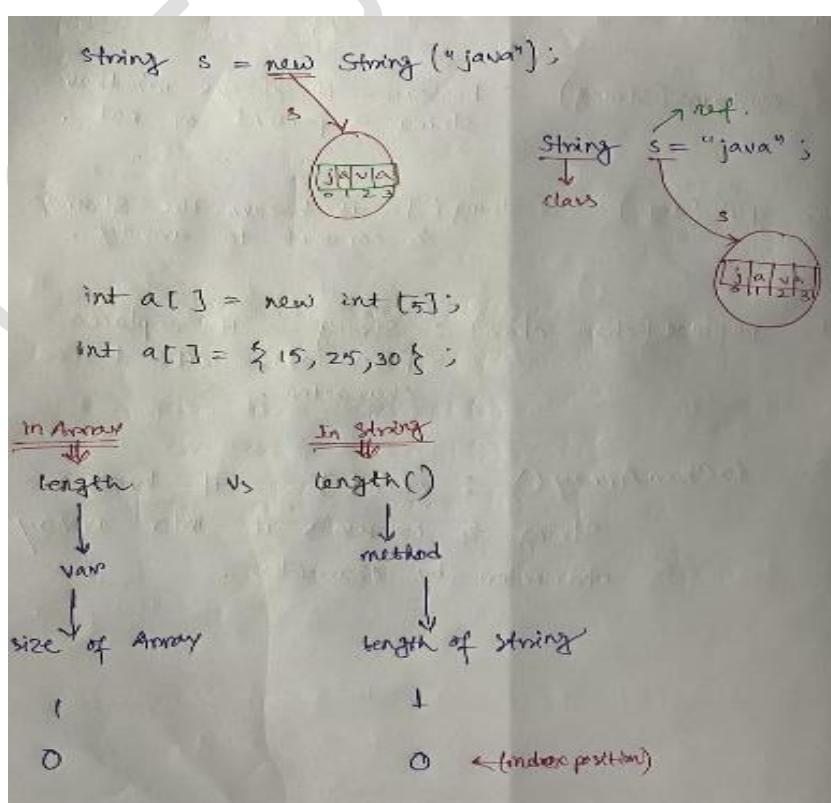
**14.concat(String) : String** – it adds string at the end.

**15.contains(String) : boolean** – it check whether string is present or not.

**16.split(args) : Sting [ ]** – it breaks the String & convert to array.

**17.replace(char, char) : String** – it replace given char with desire character.

**18.toCharArray( ) : char [ ]** – it breaks String & converts it into array character by character.



## DIFFERENCE B/W LENGTH VARIABLE & LENGTH METHOD

### LENGTH VARIABLE

1.Length is pre-defined variable, which is used to find out length of array object

Ex:

```
public int length()
{
    return length of String;
}
```

### LENGTH METHOD

1.Length( ) is pre-defined method, which is used to find out length of a string

Ex: int = s.length();

4

SOP(Length);

### 2.charAt(int index):-

```
public char charAt(int index)
{
    return char;
}
```

Char ch = s.charAt(0);

Char ch = s.charAt(1);

Char ch = s.charAt(2);

Char ch = s.charAt(3);

Char ch = s.charAt(4);

StringIndexOutOfBoundsException

/\* WAP TO READ THE STRING VALUE FROM THE USER & PRINT ALL THE CHARACTER USING FOR LOOP \*/ (length( ) + char (int index))

```
package program;
import java.util.Scanner;
public class String
```

```

{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("ENTER STRING VALUE: ");
        String ip = sc.next(); // String class object creation (using i/p from user)
        for(int i=0;i<ip.length();i++)
        {
            Take length of String i.e. 13
            char ch=ip.charAt(i);
            System.out.println(ch+ " ");
        }
    }
}

```

```

Console X
<terminated> Array2 [Java]
ENTER STRING VALUE:
JAVADEVELOPER
J
A
V
A
D
E
V
E
L
O
P
E
R

```

/\* WAP TO READ YOUR NAME THROUGH SCANNER CLASS & PRINT ONLY VOWELS FROM IT \*/

```

package program;
import java.util.Scanner;
public class String2
{
    public static void main(String[] args)
    {

```

```

Scanner sc = new Scanner(System.in);
System.out.println("ENTER STRING VALUE: ");
String ip = sc.next();
for(int i=0;i<ip.length();i++)
{
    char ch=ip.charAt(i);
    if (ch=='A'||ch=='E'||ch=='O'||ch=='U'||ch=='I')
        System.out.println(ch+" ");
}
sc.close();
}
}

```

```

Console X
<terminated> Array3 (1) [Java]
ENTER STRING VALUE:
VIGNESH
I
E

```

**/\* WAP TO PRINT COUNT OF VOWELS & COUNT OF CONSONANT FROM THE  
GIVEN STRING TAKE THE STRING VALUE FROM USER \*/**

```

package Program;
import java.util.Scanner;
public class String3
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("ENTER STRING VALUE: ");
        String ip = sc.next();int countV= 0,countC=0;
        for(int i=0;i<ip.length();i++)
        {
            char ch=ip.charAt(i);
            if (ch=='A'||ch=='E'||ch=='O'||ch=='U'||ch=='I')

```

```

        countv++;
    else
        countc++;
}
System.out.println(countV+" "+countC);
sc.close();
}
}

```

```

Console X
<terminated> String3 [Java]
ENTER STRING VALUE:
VIGNESH
2 5

```

```

/* WAP TO PRINT REVERSE OF A STRING */
package Program;
public class Reverse
{
    public static void main(String[] args)
    {
        String s = "Java Developer";
        String rev="";
        for(int i=s.length()-1;i>=0;i--)
        {
            char ch = s.charAt(i);
            rev = rev+ch;
        }
        System.out.println("Reverse"+ " of String is : "+rev);
    }
}

```

```

Console X
<terminated> Reverse [Java Application] C:\Users\k
Reverse of String is : repoleveD avaJ

```

```

/* WAP WHETHER THE STRING IS PALINDROME OR NOT */

package Program;

public class Palindrome
{
    public static void main(String[] args)
    {
        String s = "mam";
        String rev="";
        for(int i=s.length()-1;i>=0;i--)
        {
            char ch = s.charAt(i);
            rev = rev+ch;
        }
        if (rev.equals(s))
            System.out.println(s+" is a palindrome");
        else
            System.out.println(s+" is not a palindrome");
    }
}

```

Console X  
<terminated> Palindrome []  
mam is a palindrome

### Example:-2

```

package Program;

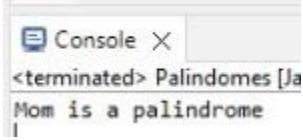
public class Palindomes
{
    public static void main(String[] args)
    {
        String s = "Mom";
        String rev="";
        for(int i=s.length()-1;i>=0;i--)
        {

```

```

char ch = s.charAt(i);
rev = rev+ch;
}
if (rev.equalsIgnoreCase(s))
    System.out.println(s+" is a palindrome");
else
    System.out.println(s+" is not a palindrome");
}
}

```



⊕ In String == not working properly. So, we should use equal(s) methods.

It return true if string content matches

like → "mom" (i/p) → "mom" (it will work)

⊕ But if input is Mom → in reverse → it will not work similar

So, we should use equalsIgnoreCase(String) method

It returns true if string content matches w/o considering case sensitivity

Line → I/p = Mom

O/p = Mom is palindrome

/\* WAP TO COUNT NUMBER OF WORDS PRESENT IN A STRING \*/

package Program;

public class CoutWord

{

public static void main(String[] args)

{

String s = "Manual Testing Eng";

```

int count=1;
s = s.trim()// Remove Space From Starting and ending
for(int i=0;i<s.length();i++)
{
    if (s.charAt(i)==' ' && s.charAt(i+1)!=' ')
    {
        count++;
    }
}
System.out.println("Count of Words :" +count);
}
}

```

```

Console X
<terminated> CoutNumber
Count of Words :3
|
```

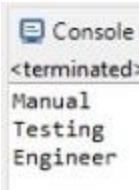
### **NOTE:-**

**trim()** → will reverse space from starting and ending of a string, not from the middle.

### **SPLIT (ARGS) : STRING [ ]**

```

package Program;
public class CountWords
{
    public static void main(String[] args)
    {
        String s = "Manual Testing Engineer";
        String str [ ] = s.split(" ");
        for(int i=0;i<str.length;i++)
        {
            System.out.println(str[i]);
        }
    }
}
```



**/\* WAP TO PRINT WORDS IN REVERSE ORDER FROM THE GIVEN STRING \*/**

Ex:- I/P → **MANUAL TEST ENGINEER**

O/P → **ENGINEER TEST MANUAL**

```
package Program;
```

```
public class ReverseWords
```

```
{
```

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

```
{
```

```
        String s = "Manual Testing Engineer";
```

```
        String str [] = s.split(" ");
```

```
        for(int i=str.length-1;i>=0;i--)
```

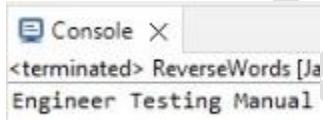
```
{
```

```
            System.out.print(str[i]+" ");
```

```
}
```

```
}
```

```
}
```



**/\* WAP TO PRINT FIRST & LAST WORD FROM STRING \*/**

```
package Program;
```

```
public class FLW
```

```
{
```

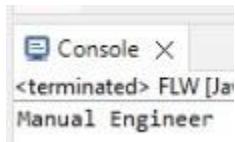
```
    public static void main(String[] args)
```

```
{
```

```
        String s = "Manual Testing Engineer";
```

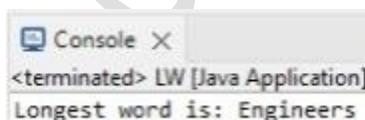
```
        String str [] = s.split(" ");
```

```
        System.out.println(str[0]+" "+str[2]);
    }
}
```



**/\* WAP TO PRINT LONGEST WORD FROM THEN GIVEN STRING \*/**

```
package Program;
public class LongestWord
{
    public static void main(String[] args)
    {
        String s = "We Are Test Engineers";
        String str[ ] = s.split(" ");
        String max = str[0];
        for(int i=1;i<str.length;i++)
        {
            if(str[i].length()>max.length())
            {
                max = str[i];
            }
        }
        System.out.println("Longest word is: "+max );
    }
}
```



**/\* WAP TO PRINT SMALLEST WORD FROM THEN GIVEN STRING \*/**

```
package Program;
public class SmallestWord
```

```

{
public static void main(String[] args)
{
String s = "We Are Test Engineers";
String str[ ] = s.split(" ");
String min = str[0];
for(int i=1;i<str.length;i++)
{
if(str[i].length()<min.length())
{
min = str[i];
}
}
System.out.println("Smallest word is: "+min );
}
}

```

```

Console X
<terminated> LW [Java App]
Smallest word is: We

```

**/\* WAP TO PRINT ALL THE WORDS FROM THE STRING USING FOR-EACH LOOP \*/**

**SYNTAX:- FOR-EACH LOOP**

```

[ ] --> Array
[ ] --> String
[ ] --> Collection
[ ] --> Map

```

**FOR (ARRAY TYPE    VARIABLE NAME : ARRAY NAME)**

```

{
System.out.println(VARIABLE NAME );
}

```

**package Program;**

```

public class Word
{
    public static void main(String[] args)
    {
        String s = "We Are Test Engineers";
        String str[ ] = s.split(" ");
        // foreach loop
        for(String v : str)
        {
            System.out.println(v);
        }
    }
}

```

 Console :  
<terminated>  
We  
Are  
Test  
Engineers

**/\* WAP TO PRINT FREQUENCY OF WORD FROM THE STRING \*/**

```

package Program;
public class FrequencyWord
{
    public static void main(String[] args)
    {
        String s = "we are happy we want be happy";
        String word = "happy";
        String str [] = s.split(" ");
        int count = 0;
        for(String v : str)
        {
            if (word.equalsIgnoreCase(v))
                count++;
        }
    }
}

```

```

        }
        System.out.println("Count of happy words: "+count);
    }
}

```

```

Console X
<terminated> FW [Java Application]
Count of happy words: 2

```

**/\* WAP TO PRINT COUNT OF PALINDROME NUMBER FROM THE STRING \*/**

```

package Program;
public class PalindromeCount
{
    public static void main(String[] args)
    {
        String s = "MOM IS A VERY PERTTY IN EACH LEVEL";
        String str [] = s.split(" ");
        int count = 0;
        for (String v : str)
        {
            String word = v, rev = "";
            for(int i=word.length()-1;i>=0;i--)
            {
                char ch = word.charAt(i);
                rev = rev+ch;
            }
            if(rev.equalsIgnoreCase(word))
                count++;
        }
        System.out.println("Count is :" + count);
    }
}

```

```

Console X
<terminated> PC
Count is :3

```

**/\* WAP TO DELETE DUPLICATE WORDS FROM A STRING \*/**

**STRING → “We Are Testing Engineers We Testing Application”**

```
package Program;
public class DeleteDuplicate
{
    public static void main(String[] args)
    {
        String s = "We Are Test Engineers We Test Application";
        String str [] = s.split(" ");
        for(int i=0;i<str.length;i++)
        {
            int count=0;
            for(int j=i+1;j<str.length;j++)
            {
                if (str[i].equalsIgnoreCase(str[j]))
                {
                    str [j] =" ";
                    count++;
                }
            }
            if (count==0 && str[i]!=" ")
                System.out.println(str[i]);
        }
    }
}
```

```
Console X
<terminated> Del
Are
Engineers
Application
```

**Example:-2**

```
package Program;
```

```

public class DeleteDuplicate
{
    public static void main(String[] args)
    {
        String s = "We Are Test Engineers We Test Application";
        String str [] = s.split(" ");
        for(int i=0;i<str.length;i++)
        {
            for(int j=i+1;j<str.length;j++)
            {
                if (str[i].equalsIgnoreCase(str[j]))
                {
                    str [j] = "";
                }
            }
            if (str[i]!=" ")
                System.out.println(str[i]);
        }
    }
}

```

```

Console X:
<terminated> DeleteDuplicate
We
Are
Test
Engineers
Application

```

### **TO CHARARRAY: : char [ ]**

*– it breaks String & converts it into array character by character.*

#### **Example:**

```

package Program;
public class CharArray
{

```

```

public static void main (String[] args)
{
    String s = "manual Testing";
    char ch[] = s.toCharArray();
    for(int i=0;i<ch.length;i++)
    {
        System.out.println(ch[i]);
    }
}

```

```

Console - terminated
m
a
n
u
a
l
T
e
s
t
i
n
g

```

**/\* WAP TO CALCULATE FREQUENCY OF EACH CHARACTER IN A GIVEN STRING \*/**  
**IF STRING → “MANUAL TESTING”, WE NEED TO DISPLAY HOW MANY TIMES,**  
**EACH CHARACTER OCCURS**

```

package Program;
public class CharArray
{
    public static void main(String[] args)
    {
        String s = "manual Testing";
        s = s.toUpperCase();
        char ch[] = s.toCharArray();
        for(char c='A';c<='Z';c++)

```

```

{
    int count = 0;
    for(int i=0;i<ch.length;i++)
    {
        if(c==ch[i])
            count++;
    }
    if(count>0)
        System.out.println(c+"-"+count);
}
}

```

```

<terminated>
A-2
E-1
G-1
I-1
L-1
M-1
N-2
S-1
T-2
U-1

```

### **INDEX OF (METHOD-STRING): INT**

*- It returns index of given string*

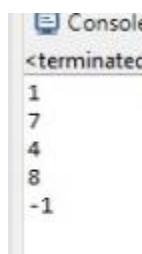
#### **Example**

```

package Program;
public class IndexOfDemo
{
    public static void main(String[] args)
    {
        String s = "manual testing";
        int i = s.indexOf('a');
        int j = s.indexOf("tes");
        int k = s.indexOf('a',2); //from '2' index , it will search where 'a' is
        int l = s.indexOf("est",4); //0,1,2,3 skipped
    }
}

```

```
int m = s.indexOf('z'); //int that string, no 'z' so will shows '-1'
```

```
System.out.println(i);//1
System.out.println(j);//7 (only first character's index)
System.out.println(k);//4
System.out.println(l);//8 ->index 0,1,2,3 will skip,count from 4
System.out.println(m);//1
}
}


```
Console
<terminated>
1
7
4
8
-1
```

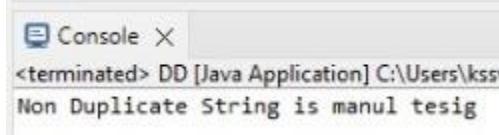

```

**/\* WAP TO DELETE DUPLICATE CHARACTER FROM A STRING \*/**

```
package Program;
public class DeleteDuplicate
{
    public static void main(String[] args)
    {
        String s = "manual testing";
        String un = "";
        for (int i=0;i<s.length();i++)
        {
            char ch = s.charAt(i);
            if(un.indexOf(ch)==-1)
            {
                un = un+ch;
            }
        }
        System.out.println("Non Duplicate "+ "String is "+un);
    }
}
```

```
}
```

```
}
```



```
Console X
<terminated> DD [Java Application] C:\Users\kss
Non Duplicate String is manul tesig
```

## USING ALL STRING METHODS:

```
package Program;

public class Methods
{
    public static void main(String[] args)
    {
        String s = "    Java Development";
        String s1 = s.trim();
        String s2 = s.toLowerCase();
        String s3 = s.concat(" is not soo easy");
        String s4 = s.toUpperCase();
        boolean s5 = s.contains("Testing");

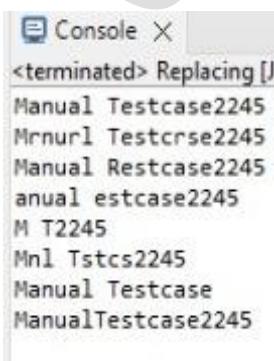
        System.out.println(s);
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
    }
}
```



```
Console X
<terminated> Methods [Java Application] C:\Users\kss
Java Development
Java Development
java development
Java Development is not soo easy
JAVA DEVELOPMENT
false
```

## **REPLACE (CHAR, CHAR):**

```
-----  
package Program;  
  
public class Replacing  
{  
    public static void main(String[] args)  
    {  
        String s = "Manual Testcase2245";  
  
        String s1 = s.replace('a','r');  
        String s2 = s.replace("Test","Rest");  
        String s3 = s.replaceAll("[A-Z]","");
        String s4 = s.replaceAll("[a-z]","");
        String s5 = s.replaceAll("[AEIOUaeiou]","");
        String s6 = s.replaceAll("[0-9]","");
        String s7 = s.replaceAll(" ", "");  
  
        System.out.println(s);
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
        System.out.println(s4);
        System.out.println(s5);
        System.out.println(s6);
        System.out.println(s7);
    }
}
```



The screenshot shows a Java console window titled "Console X". The output of the program is displayed, showing the original string and the results of various replace operations:

```
<terminated> Replacing []
Manual Testcase2245
Mrnurl Testcrse2245
Manual Restcase2245
annual estcase2245
M T2245
Mnl Tstcs2245
Manual Testcase
ManualTestcase2245
```

### **SUBSTRING (INT STARTINDEX) : String**

*- it return subpart of a string*

### **SUBSTRING(INT STARTINDEX, INT ENDINDEX) : String**

*- it returns subpart of a string from start From start index till end index-1.*

**package** Program;

**public class** SubStringDemo

{

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

{

    String s = "Manual Testing";

    String s1 = s.substring(4); *// it will give o/p from 4 index to last*

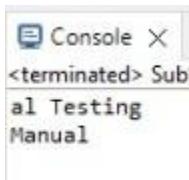
    String s2 = s.substring(0, 7); *// it will give o/p from 0 index to 7<sup>th</sup> index*

    System.out.println(s1);

    System.out.println(s2);

}

}



```
Console X
<terminated> Sub!
al Testing
Manual
```

### **STRING:**

- ❖ *String is predefine class which is present in java.lang package*
- ❖ *In java string is an -*
  - OBJECT
  - DATATYPE
  - CLASS
  - GROUP OF CHARACTERS

### **OBJECTS**

*String objects can be created in two ways-*

#### **1. USING NEW KEYWORD - two objects will get created**

- *one is under heap area - reference variable assigned to it.*
- *another is under string constant pool(SCP) - it is smallpart of heap area no reference assigned to it, it is only for backup.*

-String is created as Object.  
-String Object are immutable.

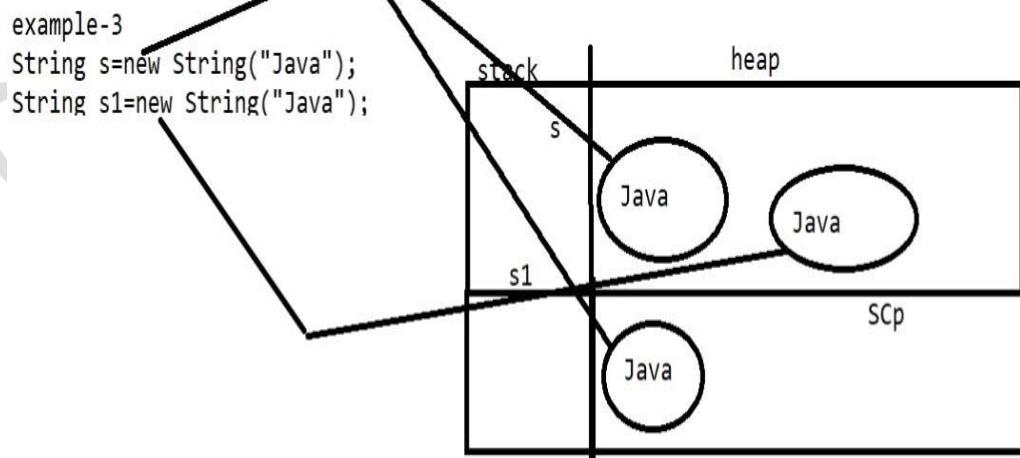
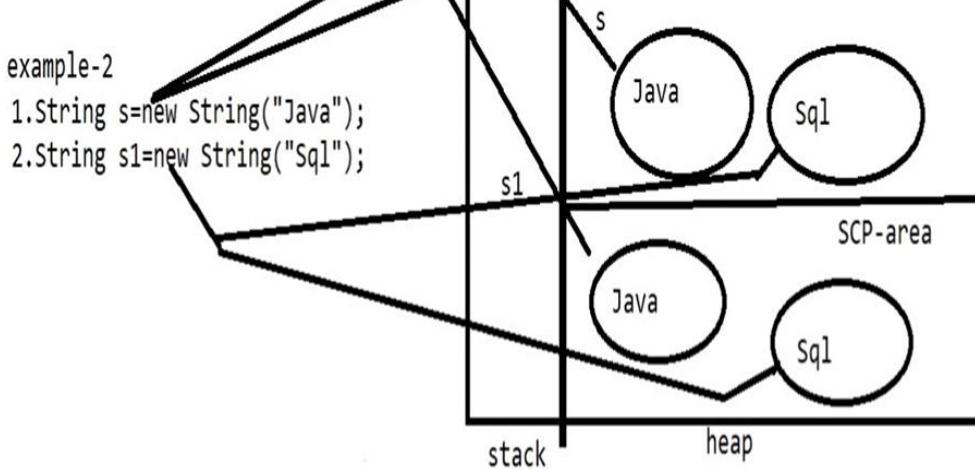
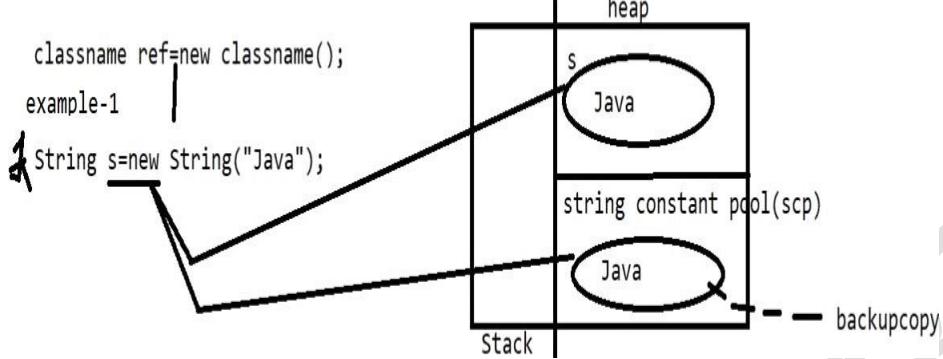
1 String s="java"; ✓

classname ref=new classname();  
example-1

1 String s=new String("Java");

example-2  
1.String s=new String("Java");  
2.String s1=new String("Sql");

example-3  
String s=new String("Java");  
String s1=new String("Java");

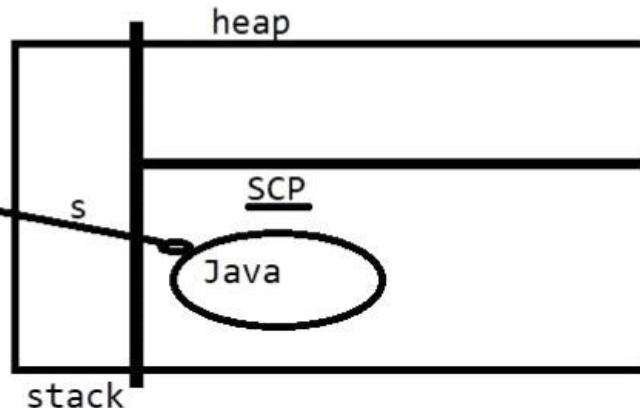


## 2. USING LITERAL

- first JVM will go to SCP and check if there is any object present with string data, if it is there it will assign reference to it, if it is not there it creates new object and assign reference.

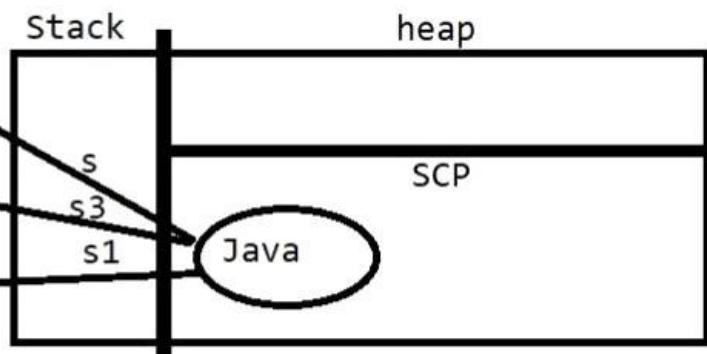
Example-1

```
String s="Java";
```



Example-2

```
String s="Java";
String s1="Java";
String s3="Java";
```



## IMMUTABILITY

For above examples, we understand that for one string object there can be multiple references assigned to it, if we change data of one string object it will effect to multiple references that's why java says that string objects are immutable i.e. once we created we cannot modify it.

### Example:-1

```
package Program;

public class SubStringDemo
{
    public static void main(String[] args)
    {
        String s = "Manually Testing";
```

```

        s.concat("the app");

    System.out.println(s);
}

}

```

```

Console X
<terminated> SubStringDemo
Manually Testing

```

### Example:-2

```

package Program;
public class SubStringDemo
{
    public static void main(String[] args)
    {
        String s = "Manually Testing";
        s = s.concat(" the app ");

        System.out.println(s);
    }
}

```

```

Console X
<terminated> SubStringDemo [Java]
Manually Testing the app

```

## STRING BUFFER & STRING BUILDER

- + These are classes which are present in *java.lang package*
- + *String buffer & String builder object can be created only by using new keyword*

**Ex :** String Buffer b1 = **new** String Buffer("Java");  
 ---- String Builder b2 = **new** String Builder("Java");  
 String Buffer b3 = "Java"; // invalid  
 String Builder b4 = "Java"; // invalid

- String Buffer & String Builder objects are multiple i.e. once we created we can modify it.
- String Buffer & String Builder objects will get created in heap area only.

### EQUALS ():

- In String class equal () compare based on string data.
- In String Buffer & String Builder class equals () compares based on reference
- In String class equals () is overridden from object class to compare two strings object based on content.
- In String Buffer and String Builder class equals () is not overridden so it compares based on reference (same like object class).
- == Operator can't be applied for string builder and string buffer objects.

### DIFFERENCES BETWEEN STRING, STRING BUFFER & STRING BUILDER

S.No	STRING	STRING BUFFER	STRING BUILDER
1.	String objects are immutable	String buffer objects are mutable	String builder objects are mutable
2.	Objects can be created in two ways 1.new 2.literal	Objects can be created only using new keyword	Objects can be created only using new keyword
3.	Objects will created in SCP or heap	Objects will created in heap	Objects will created in heap
4.	Thread safe i.e. at a time only one thread is allowed to operate on string object	Thread safe i.e. at a tie only one thread is allowed to operate on string buffer object	Not a Thread safe
5.	If context is fixed we will go for string	If context is varying we will go for string buffer	If context is varying we will go for string builder
6.	equals()-compares two String by seeing content	equals()-compares two String by seeing reference	equals()-compares two String by seeing reference
7.	performance is faster	performance is moderate	performance is faster

8.	For concatenation we have concat()	For concatenation we have append()	For concatenation we have append()
----	------------------------------------	------------------------------------	------------------------------------

## PATTERN PROGRAMMING

---

1. \*\*\*  
\*\*\*  
\*\*\*

```
package Program;
public class Pattern
{
    public static void main(String[] args)
    {
```

```
        for (int row=1;row<=3;row++)
        {
            for (int col=1;col<=3;col++)
            {
                System.out.print("*");
            }
            System.out.println( );
        }
    }
```

2. \*\*\*\*  
\*\*\*  
\*\*\*  
\*\*\*

```
package Program;
public class Patterns
{
    public static void main(String[] args)
    {
        for (int row=1;row<=4;row++)

```

```

{
    for (int col=1;col<=4;col++)
    {
        System.out.print("*");
    }
    System.out.println( );
}

}
3. ****
*****
**#*
****

package Program;
public class Pattern
{
    public static void main(String[] args)
    {
        for (int row=1;row<=4;row++)
        {
            for (int col=1;col<=4;col++)
            {
                if (row==3 && col==3)
                    System.out.print("#");
                else
                    System.out.print("*");
            }
            System.out.println( );
        }
    }
}
4. *#*#
*#*#

```

\*#\*#  
\*#\*#

```
package Program;  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for (int row=1;row<=4;row++)  
        {  
            for (int col=1;col<=4;col++)  
            {  
                if (col%2==0)  
                    System.out.print("#");  
                else  
                    System.out.print("*");  
            }  
            System.out.println();  
        }  
    }  
}
```

5. \*\*\*\*  
####  
\*\*\*\*  
####

```
package Program;  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for (int row=1;row<=4;row++)  
        {  
            for (int col=1;col<=4;col++)  
            {  
                if (col%2==0)  
                    System.out.print("#");  
                else  
                    System.out.print("*");  
            }  
            System.out.println();  
        }  
    }  
}
```

```

{
    if (row%2==0)
        System.out.print("#");
    else
        System.out.print("*");
    }
    System.out.println();
}
}

```

6. \*\*\*\*  
 \* \*  
 \* \*  
 \* \*  
 \*\*\*\*

```

package Program;
public class Parttern
{
    public static void main(String[] args)
    {
        for (int row=1;row<=5;row++)
        {
            for (int col=1;col<=5;col++)
            {
                if (row==1 || row==5 || col==1 || col==5)
                    System.out.print("*");
                else
                    System.out.print(" ");
            }
            System.out.println();
        }
    }
}

```

```
}
```

7.



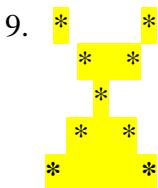
```
package Program;  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for (int row=1;row<=5;row++)  
        {  
            for (int col=1;col<=5;col++)  
            {  
                if (row==3 || col==3)  
                    System.out.print("*");  
                else  
                    System.out.print(" ");  
            }  
            System.out.println();  
        }  
    }  
}
```

8.



```
package Program;  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for (int row=1;row<=4;row++)  
        {
```

```
for (int col=1;col<=4;col++)
{
    if (row==col)
        System.out.print("*");
    else
        System.out.print(" ");
}
System.out.println();
}
```



```
package Program;  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for (int row=1;row<=5;row++)  
        {  
            for (int col=1;col<=5;col++)  
            {  
                if (row==col || row+col==6)  
                    System.out.print("*");  
                else  
                    System.out.print(" ");  
            }  
            System.out.println();  
        }  
    }  
}
```

10.\*

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

```
package Program;  
  
public class Pattern9  
{  
    public static void main(String[] args)  
    {  
        for (int row=1;row<=4;row++)  
        {  
            for(int col=1;col<=4;col++)  
            {  
                if (row>=col)  
                    System.out.print("*");  
                else  
                    System.out.print(" ");  
            }  
            System.out.println();  
        }  
    }  
}
```

11.\*\*\*\*

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

```
package Program;  
  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for (int row=4;row>=1;row--)  
        {  
            for(int col=1;col<=4;col++)  
            {  
            }  
        }  
    }  
}
```

```

{
    if (row>=col)
        System.out.print("*");
    else
        System.out.print(" ");
}
System.out.println();
}
}
}

12.


```

```

package Program;
public class Pattern
{
    public static void main(String[] args)
    {
        for (int row=1;row<=4;row++)
        {
            for(int col=4;col>=1;col++)
            {
                if (row>=col)
                    System.out.print("*");
                else
                    System.out.print(" ");
            }
            System.out.println();
        }
    }
}

```

13. \*\*\*\*  
  \*\*\*  
  \*\*  
  \*

```
package Program;  
  
public class Pattern12  
{  
    public static void main(String[] args)  
    {  
        for(int row=1;row<=4;row++)  
        {  
            for(int col=1;col<=4;col++)  
            {  
                if (row<=col)  
                    System.out.print("*");  
                else  
                    System.out.print(" ");  
            }  
            System.out.println();  
        }  
    }  
}
```

14. \*  
  \*\*  
  \*\*\*  
  \*\*\*\*

```
package Program;  
  
public class patterns14  
{  
    public static void main(String[] args)  
    {  
        for(int row=1;row<=4 ;row++)  
        {  
            for(int col=1;col<=4;col++)  
            {  
                if (row>col)  
                    System.out.print("*");  
                else  
                    System.out.print(" ");  
            }  
            System.out.println();  
        }  
    }  
}
```

```

    {
        if(row+col>=5)
            System.out.print("* ");
        else
            System.out.print(" ");
    }
    System.out.println();
}
}

```

**15.\* \* \* \***

```

package Program;
public class Pattern
{
    public static void main(String[] args)
    {
        for(int row=1;row<=4 ;row++)
        {
            for(int col=1;col<=4;col++)
            {
                if(row<=col)
                    System.out.print("* ");
                else
                    System.out.print(" ");
            }
            System.out.println();
        }
    }
}

```

## TYPE-2 (PATTERN PROGRAMMING)

```
1.1 1  
2 2 2  
3 3 3  
4 4 4  
package Type2;  
public class Pattern  
{  
    public static void main(String[] args)  
    {  
        for(int row=1;row<=4;row++)  
        {  
            for (int col=1;col<=3;col++)  
            {  
                System.out.print(row+" ");  
            }  
            System.out.println( );  
        }  
    }  
}
```

```
2. 1 2 3  
   1 2 3  
   1 2 3  
   1 2 3  
package Type2;  
public class Pattern2  
{  
    public static void main(String[] args)  
    {  
        for(int row=1;row<=4;row++)  
        {  
            for (int col=1;col<=3;col++)  
            {
```

```
        System.out.print(col+" ");
    }
    System.out.println( );
}
}
```

3.1 2 3 4  
5 6 7 8  
9 10 11 12

```
package Type2;
public class Pattern3
{
    public static void main(String[] args)
    {
        int count = 1;
        for(int row=1;row<=3;row++)
        {
            for (int col=1;col<=4;col++)
            {
                System.out.print(count+" ");
                count++;
            }
            System.out.println();
        }
    }
}
```

4.1  
2 2  
3 3 3  
4 4 4 4

```
package Type2;
public class Pattern4
{
```

```
public static void main(String[] args)
{
    for(int row=1;row<=4;row++)
    {
        for (int col=1;col<=row;col++)
        {
            System.out.print(row+" ");
        }
        System.out.println( );
    }
}
```

5.

1			
1	2		
1	2	3	
1	2	3	4

```
package Type2;
public class Pattern5
{
    public static void main(String[] args)
    {
        for(int row=1;row<=4;row++)
        {
            for (int col=1;col<=row;col++)
            {
                System.out.print(col+" ");
            }
            System.out.println( );
        }
    }
}
```

6. 

```
package Type2;

public class Pattern6
{
    public static void main(String[] args)
    {
        int count = 1;
        for(int row=1;row<=4;row++)
        {
            for (int col=1;col<=row;col++)
            {
                System.out.print(count+" ");
                count++;
            }
            System.out.println();
        }
    }
}
```

### TYPE-3 (PATTERN PROGRAMMING)

1. 

```
package Type3;

public class Pattern
{
    public static void main(String[] args)
    {
        for(char row='A';row<='D';row++)
        {
    }
```

```
for (char col='A';col<='C';col++)
{
    System.out.print(row+" ");
}
System.out.println( );
}
}
```

2. **A B C**  
**A B C**  
**A B C**  
**A B C**

```
package Type3;
public class Pattern2
{
```

```
public static void main(String[] args)
{
for(char row='A';row<='D';row++)
{
    for (char col='A';col<='C';col++)
    {
        System.out.print(col+" ");
    }
    System.out.println( );
}
}
```

3. **A B C**  
**D E F**  
**G H I**  
**J K L**

```
package Type3;
public class Pattern3
{
```

```
public static void main(String[] args)
{
    char c = 'A';
    for(char row='A';row<='D';row++)
    {
        for (char col='A';col<='C';col++)
        {
            System.out.print(c+" ");
            c++;
        }
        System.out.println( );
    }
}
```

4. A  
A B  
A B C  
A B C D

```
package Type3;
public class Pattern4
```

```
{

public static void main(String[] args)
{
    for(char row='A';row<='D';row++)
    {
        for (char col='A';col<= row;col++)
        {
            System.out.print(col+" ");
        }
        System.out.println( );
    }
}
```

```
}
```

5. A  
B B  
C C C  
D D D D

```
package Type3;
```

```
public class Pattern5
```

```
{
```

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

```
{
```

```
for(char row='A';row<='D';row++)
```

```
{
```

```
for (char col='A';col<= row;col++)
```

```
{
```

```
System.out.print(row+" ");
```

```
}
```

```
System.out.println( );
```

```
}
```

```
}
```

```
}
```

6. A  
B C  
D E F  
G H I J

```
package Type3;
```

```
public class Pattern6
```

```
{
```

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

```
{
```

```
char c = 'A';
```

```
for(char row='A';row<='D';row++)
```

```
{
```

```
for (char col='A';col<= row;col++)
```

```
{
```

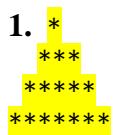
```

        System.out.print(c+" ");
        c++;
    }
    System.out.println();
}
}
}
}

```

#### **TYPE-4 (PATTERN PROGRAM)**

1.



```

      *
     ***
    *****
   *****
  *****
 *****

```

```

package Type4;
public class Pattern
{
    public static void main(String[] args)
    {
        int star = 1;
        int space = 3;
        for(int row=1;row<=4;row++)
        {
            for (int col=1;col<= space;col++)
            {
                System.out.print(" ");
            }
            for (int k=1;k<= star;k++)
            {
                System.out.print("*");
            }
            System.out.println();
            space = space-1;
        star = star+2;
    }
}

```

```
    }
}
}

2. *****
*****
 ***
 *
package Type4;
public class Pattern2
{
    public static void main(String[] args)
    {
        int star = 7;
        int space = 0;
        for(int row=1;row<=4;row++)
        {
            for (int col=1;col<= space;col++)
            {
                System.out.print(" ");
            }
            for (int k=1;k<= star;k++)
            {
                System.out.print("*");
            }
            System.out.println( );
            space = space+1;
            star = star-2;
        }
    }
}
```

3.

```
*  
***  
*****  
*****  
*****  
***  
*
```

```
package Type4;  
  
public class Pattern3  
{  
    public static void main(String[] args)  
    {  
        int star = 1;  
        int space = 3;  
        for(int row=1;row<=7;row++)  
        {  
            for (int col=1;col<= space;col++)  
            {  
                System.out.print(" ");  
            }  
            for (int k=1;k<= star;k++)  
            {  
                System.out.print("*");  
            }  
            if(row<=3)  
            {  
                star = star+2;  
                space = space-1;  
            }  
            else  
            {  
                star = star-2;  
                space = space+1;  
            }  
        }  
    }  
}
```

```
        System.out.println( );
    }
}
}

4. *****
 ****
 *** 
  *
 ***
 ****
 *****

package Type4;
public class Pattern4
{
    public static void main(String[] args)
    {
        int star = 7;
        int space = 0;
        for(int row=1;row<=7;row++)
        {
            for (int col=1;col<= space;col++)
            {
                System.out.print(" ");
            }
            for (int k=1;k<= star;k++)
            {
                System.out.print("*");
            }
            if(row<=3)
            {
                star = star-2;
                space = space+1;
            }
            else
            {
```

```
    star = star+2;
    space = space-1;

}
System.out.println( );
}

}

5. *
**
***
****
 ***
 **
 *
```

```
package Type4;
public class Pattern5
{
    public static void main(String[] args)
    {
        int star = 1;
        int space = 3;
        for(int row=1;row<=7;row++)
        {
            for (int col=1;col<= space;col++)
            {
                System.out.print(" ");
            }
            for (int k=1;k<= star;k++)
            {
                System.out.print("*");
            }
            if(row<=3)
            {
                star = star+1;
            }
        }
    }
}
```

```
        }
    else
    {
        star = star-1;
    }
    System.out.println( );
}
}
```

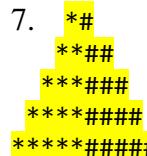
6.

```
*****
 ****
 ***
 **
 *
 *
```

```
package Type4;
public class Pattern6
{
    public static void main(String[] args)
    {
        int star = 1;
        int space = 3;
        for(int row=1;row<=7;row++)
        {
            for (int col=1;col<= space;col++)
            {
                System.out.print(" ");
            }
            for (int k=1;k<= star;k++)
            {
                System.out.print("*");
            }
            if(row<=3)
            {
```

```
        star = star-2;
        space = space+1;
    }
else
{
    star = star+2;
    space = space-1;
}
System.out.println( );
}
```

7.



```
 *#
 **##
 ***##
 ****##
 *****##
```

```
package Type4;

public class Pattern7
{
    public static void main(String[] args)
    {
        int star = 2;
        int space = 4;
        for(int row=1;row<=5;row++)
        {
            for (int col=1;col<= space;col++)
            {
                System.out.print(" ");
            }
            int count = 0;
            for (int k=1;k<= star;k++)
            {
                count++;
                if(count<=star/2)
                    System.out.print("*");
                else
                {
                    star = star-2;
                    space = space+1;
                }
            }
            System.out.println();
        }
    }
}
```

```

    else
        System.out.print("#");
    }
    star = star+2;
    space = space-1;
}
System.out.println( );
}
}

```

### **Example:-1**

```

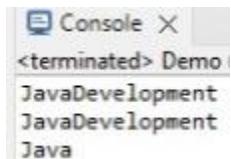
package Array;

public class Demo
{
    public static void main(String[] args)
    {
        StringBuffer s1 = new StringBuffer("Java"); // HEAP
        StringBuilder s2 = new StringBuilder("Java"); // HEAP
        String s3 = new String ("Java"); // 1.HEAP 2.SCP
        String s4 = "Java";

        s3.concat("Development");
        s1.append("Development");
        s2.append("Development");

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}

```



```

Console X
<terminated> Demo
JavaDevelopment
JavaDevelopment
Java

```

## **Example:-2**

```
package Array;  
  
public class Demo  
{  
    public static void main(String[] args)  
    {  
        StringBuffer s1 = new StringBuffer("Java"); // HEAP  
        StringBuilder s2 = new StringBuilder("Java"); // HEAP  
        String s3 = new String ("Java"); // 1.HEAP 2.SCP  
        String s4 = "Java";  
  
        s3.concat("Development");  
        s1.append("Development");  
        s2.append("Development");  
  
        System.out.println(s1);  
        System.out.println(s2);  
        System.out.println(s3);  
  
        boolean b1 = s1.equals("JavaDevelopment");  
        boolean b2 = s2.equals("JavaDevelopment");  
        boolean b3 = s3.equals(s4);  
  
        System.out.println(b1);  
        System.out.println(b2);  
        System.out.println(b3);  
    }  
}
```



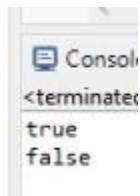
```
Console X  
<terminated> Demo ()  
JavaDevelopment  
JavaDevelopment  
Java  
false  
false  
true
```

## DIFFERENCE B/W EQUALS METHOD & EQUALS EQUALS TO METHOD

EQUALS METHOD	EQUALS EQUALS TO METHOD
1.Equals method compare values object	1. Equals equals to compare reference i.e. if two are referring to single object it returns true otherwise it returns false.

### Example:-3

```
package Array;  
public class Demo  
{  
    public static void main(String[] args)  
    {  
        String s1 = new String ("Java"); // 1.HEAP 2.SCP  
        String s2 = "Java";  
        boolean b1 = s1.equals("s2");  
        boolean b2 = s1==s2;  
        System.out.println(b1);  
        System.out.println(b2);  
    }  
}
```



## 2D-ARRAY PROGRAMMING:-

In 2d array array value are represented in terms of rows & columns.

### SYNTAX:-

Array type array name [ ] [ ] = new array type [row size] [col size];

(Or)

Array type [ ] [ ] array name = new array type [row size] [col size];

(Or)

Array type [ ] [ ] array name = new array type [row size] [col size];

## **ARRAY CREATION:-**

**FOR Ex:-** int a [ ] [ ] = new int [2] [2];

col-1	col-2
row-1	
row-2	

## **ARRAY INITIALISATION:-**

a[0][0] = 10

a[0][1] = 20

a[1][0] = 30

a[1][1] = 40

→ **Stored as 10 20  
30 40**

## **ARRAY UTILISATION:-**

**System.out.println(array name [row index] [col index]);**

**Ex:- System.out.println(a[0][0]);**

**/\*WAP TO CREATE 2X2 MATRIX STORE THE VALUES INTO IT AND PRINT THEM \*/**

```
package Array2D;
```

```
public class A2D
```

```
{
```

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

```
{
```

```
int a [] [] = new int [2] [2];
```

```
a[0][0] = 10;
```

```
a[0][1] = 20;
```

```
a[1][0] = 30;
```

```
a[1][1] = 40;
```

```
for(int row=0;row<2;row++)
```

```
{
```

```
for(int col=0;col<2;col++)
```

```
{
```

```
System.out.print(a[row][col]+ " ");
```

```
}

System.out.println( );

}

}

}

<terminate>
10 20
30 40
```

### EXAMPLE:-2

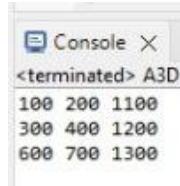
```
package Array;

public class A3D
{
    public static void main(String[] args)
    {
        int a [] [] = new int [3] [3];
        a[0][0] = 100;
        a[0][1] = 200;
        a[0][2] = 1100;
        a[1][0] = 300;
        a[1][1] = 400;
        a[1][2] = 1200;
        a[2][0] = 600;
        a[2][1] = 700;
        a[2][2] = 1300;
        for(int row=0;row<3;row++)
        {
            for(int col=0;col<3;col++)
            {
                System.out.print(a[row][col]+" ");
            }
            System.out.println();
        }
    }
}
```

```
}
```

```
}
```

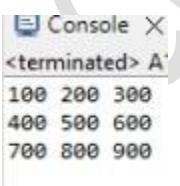
```
}
```



```
Console X
<terminated> A3D
100 200 1100
300 400 1200
600 700 1300
```

### EXAMPLE:-3

```
package Array;
public class A1D
{
    public static void main(String[] args)
    {
        int a [] [ ] = {{100,200,300},{400,500,600},{700,800,900}};
        for(int row=0;row<3;row++)
        {
            for(int col=0;col<3;col++)
            {
                System.out.print(a[row][col]+ " ");
            }
            System.out.println();
        }
    }
}
```



```
Console X
<terminated> A1D
100 200 300
400 500 600
700 800 900
```

**/\*WAP TO TAKE 2D ARRAY VALUES FROM SCANNER CLASS & PRINT THEN  
USING FOR LOOP \*/**

```

package Array;
public class A1D
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter row size & col size");
        int rowsize = sc.nextInt();
        int colsiz = sc.nextInt();
        int a [] [] = new int [rowsize][colsiz];
        for(int row=0;row<rowsize;row++)
        {
            for(int col=0;col<colsiz;col++)
            {
                a[row][col] = sc.nextInt();
            }
        }
        for(int row=0;row<rowsize;row++)
        {
            for(int col=0;col<colsiz;col++)
            {
                System.out.print(a[row][col]+ " ");
            }
        }
        System.out.println();
    }
    sc.close();
}

```

The screenshot shows a Java application window titled 'Console X' with the title bar '<terminated> a2 [Java Application]'. The console output is as follows:

```

Enter row size & col size
3
3
11
12
13
14
15
16
17
18
19
11 12 13
14 15 16
17 18 19

```

```
/* WAP TO PRINT SUM OF MARTICES */
package Array;
public class ArraySum
{
    public static void main(String[] args)
    {
        int a[ ][ ] = {{100,200},{300,400}};
        int b[ ][ ] = {{500,600},{700,800}};
        int s [ ][ ] = new int [2][2];
        for(int row=0;row<2;row++)
        {
            for(int col=0;col<2;col++)
            {
                s[row][col] = a[row][col]+b[row][col];
            }
        }
        for(int row=0;row<2;row++)
        {
            for(int col=0;col<2;col++)
            {
                System.out.print(s[row][col]+" ");
            }
            System.out.println();
        }
    }
}
```

```
Console X
<terminated> $1
600 800
1000 1200
```

## **DIFFERENCE BETWEEN FINAL, FINALLY & FINALIZE**

<b>FINAL</b>	<b>FINALLY</b>	<b>FINALIZE</b>
1.Final is keyword which indicate no more changes are allowed.	1.Finally is a block of code which will be executed for sure.	1. Finalize is method of object class.
2.Final is keyword is applicable with class, method & variable.	2.Finally block is used to keep an important code like closing of database connect, closing open file etc.. because it make sure that the code is executed irrespective of exception occurred, exception did not occurred & exception did not	2.Finalize method is get called by garbage collector just before it deletes the memory from heap area.
3.If a variable is final we cannot re-initialize.	Handle.	Finalize is used to keep the backup copy of delete data a in a heap area by calling garbage collector
4.If a method is final we cannot over ride it.		
5.If a class is final we cannot inherit		