write a Java program -

SROll No. Sname, Smarks, SAdress

```
Import java.lang.*;
class student
{
 int SRLNO;
 string Sname;
 float Smarks;
 string Sadress;
}
 void setdata()
 {
  SRLNO = 1903;
  Sname = "venky";
  Smarks = 69.43f;
  Saddress = "Nellore";
 }
 void display()
{
system.out.println("the student Roll no is "+ SRollNo);
system.out.println("the student name is "+ Sname);
system.out.println("the student marks is "+Smarks);
system.out.println("the student address is"+Saddress);
}
public static void main(string args[])
{
   student s=new student();
           s.setdata();
           s.display();
       } // end of main method.
      } // end of class
```

Compilation of App

Javac filename.java ⇒ Javac student.java

Run command.

output:-

SRLNO = 1903

Sname = venky

Smarks = 69.43

Sadress = Nellore

Write a java program to create employed details EmpNo. EmpName, Empdesignation, EmpID, Emp adress.

Write a java program to display Welcome to java lab.

2) Import java.lang.*;

```
class emp
{
 int EIDNO;
 string Ename;
 string Edesignation;
 string Eadress;
 void set data()
 {
 EIDNO = 2583
 Ename = "venky";
 Edesignation = "Manager";
Eadress = "Nellore";
}
void display()
{
system.out.println("the employee ID no is "+EIDNO);
system.out.println("the employee name is "+Ename);
system.out.println("the employee designation is "+Edesignation);
```

```
system.out.println("the employee adress is "+E.adress);
}
public static void main (string ArgsE])
{
  Employe E= New employe ();
        E.setdata();
        E.display ();
   }
}
```

output :-

The employee IDNO is 2583

The employee Name is venky

The employee designation is Manager

The employee adress is nellore.

3) Import.java.lang.*;

```
class Tab
{
public static void main(string ArgsE])
{
system.out.println(" welcome to Java Lab");
 }
}
```

output :-

Welcome to Java Lab

Basic Concepts / oops concept

Java is widely used, high level, object oriented programming language known for it's portability, performance. Here are some of basic concepts in Java.

1. class
2. object
3. Abstraction
4. encapsulation.
5. polymorphism
6. Message.
7. inheritance
8. Multithreading.

What is class:-

⇒ class is a blue print of object (or) class is a collection of objects
⇒ class is a keyword.

Syntax:- class classname
{
blobal variables;
own methods;
main method;
}

object :-

⇒ object is instance of a class
⇒ object means anything in real world.
eg:- car, cycle.

program:-

class lamp
{
boolean is on;
void turnon()

{

is on :true;

System.out.println("Light on?"+is on);

}

=> object is always maintain different attributes

=> Attribute means property of object.

=> According to Java means attribute means variable.

=> each and every objects are created using new operator

=> Different methods are associated to objects using C.).

Abstraction:-

=> Hiding of complexity information to user.

=> eg:- Abstract cls, Abstract Window.

Encapsulation:-

=> Encapsulation means combing of variables and methods.

=> eg:- method of parameters.

polymorphism:-

=> poly means many and morphism means terms.

=> According to polymorphism means different objects using same method (or) function

Eg:- Aeroplane fly in the sky, Birds fly in the sky.

Same method - fly : object - Aeroplane, Birds.

Message:-

=> TO exchang of any information of information from one object to another object.

Inheritance:-

=> Inheritance means acquiring the properties from one class to another class.

=> Acquiring the properties from one class to another class by

using extends keywords.

1. Single level
2. Multi level
3. Hybrid level

Multithreading :-

=> thread means single sequential of flow to control with application, Java supported inbuilt with multi concept.

=> Any application is more than one single sequential flow to control is called Multi threading.

=> To exchange any information from one thread to another thread using notify and notify all method.

* user thread created using 2 types :-

1) extends thread.
2) implements (runable) runable.

keywords :-

These are reserved words that in Java that have a specific meaning to compiler. for ex, the keywords class, public & static are used to declare class, and the keywords if, else and while are used to control to flow of a program.

Identifiers :-

these are names given to variables. But the only difference is their values cannot be modified by the program once they are defined. constants are fixed values. also called as literals.

Special operators :-

Brackets [] : opening and closing brackets are used in array element reference.

paranthesis ():- These special symbols are used to indicate function calls and function parameters.

Braces {}:- These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.

Comma (,):- It is used to operate more than one statements like seperating in parameters in function call.

Semicolon (;):- It is an operator that essentially invokes something called an initializing list.

Asterisk :- It is used create pointer variable.

Assignment :- It is used to assign the values.

Unit-2
operators :-

1) Arithmatic operators (+ - * / %)
2) unary operators
3) Assignment operator
4) Relational operators
5) Logical operators
6) Ternary operators.
7) Bitwise operators
8) shift operators.

Relational operator :-

```
import java.lang.*;
class Relational operator
{
public static void main (String[] args)
```

```
int a=10;
    b=20;
System.out.println("a==b:"+(a==b));
    "   "   "  ("a!=b:"+(a!=b));
    "   "   "  ("a>b:"+(a>b));
    "   "   "  ("a>=b:"+(a>=b));

int x=15;
    y=15;
System.out.println("x==y:"+(x==y));
    "   "   "  ("x!=y:"+(x!=y));
    "   "   "  ("x>y:"+(x>y));
    "   "   "  ("x>=y:"+(x>=y));
```

unary operators :-

```
import java.lang.*;
class unary operator
{
public static void main(String[] args)
{
  int a=10;
  int b=5;
  boolean c=true;
  int result=+a;
System.out.println("unary plus on a:"+result1);
  int result2=b++;
System.out.println("post-increment on b:"+result4);
  int result3=-a;
System.out.println("pre decrement on a:"+result5);
}}
```

output :-

unary plus on a : 10

post - increment on b : -5

pre decrement on a : 9.

Ternary operators :-

```
import .java.lang.*;

class ternary operator
{
public static void main (string[]args)
  {
   int a = 10;
   int b = 20;
   int max = (a>b) ? a : b;
  system.out.println("The maximum value is :"+ max);
  int min =(a<b)? a=b;
  system.out.println ("the minimum value is :"+min);
   }
}
```

output :-

The maximum value is :20

The minimum value is : 10

Arithmatic operators :-

```
import java.lang.*;
class Arthimatic operator
{
public static void main (string[]args)
 {
 int a=10;
 int b=3;
 system.out.println ("Arithmatic operators");
 system.out.println ("Addition :"+ (a+b));
 system.out.println ("substraction :"+ (a-b));
 system.out.println ("Multiplication :" (a*b));
```

```
System.out.println("division:"+(a/b));
System.out.println("matulus:"+(a%b));
  }
 }
```

output :-

Addition :13

subtraction :7

multiplication :30

division : 3

matulus :1.

Assignment operator :-

```
import java.lang.*;
class Assignment operator
 {
  int f=7;
  system.out.println("f+=3:"+(f+=3));
  system.out.println("f-=2:"+(f-=3));
  system.out.println("f*=4:"+(f*=4));
  system.out.println("f/=4:"+(f/=4));
  system.out.println("f%=2:"+(f%=2));
  }
 }
```

output :-

f+ 3 : 3:10

f- :2 :8

f* :4 :32

f/ :3:10

f%. :2:0

logical operator :-

```
import .java.lang.*;
class logical operator
```

```
{
  boolean x = true;
  boolean y = false;
  system.out.println("x&&y :" + (x&&y));
  system.out.println("x||y :" + (x||y));
  system.out.println("!x:" + (!x));
  }
}
```

output :-

x&&y = false

x||y = True

!x = false.

Bitwise operator :-

```
import java.lang.*;
class Bitwise operator
{
public static void main (string [] args)
{
  a:5
  b:2
system.out.println("value of a is :"+a);
system.out.println("value of b is :"+b);
system.out.println("Bitwise AND(&):"+(a&b));
system.out.println("Bitwise OR(|):"+(a|b));
system.out.println("Bitwise xOR(^):"+(a^b));
system.out.println("Bitwise compliment(~):"+(~a));
  }
}
```

shift operator :-

```
import java.lang.*;
```

```
System.out.println ("left shift (<<): "+ (a<<b));
System.out.println ("Right shift (>>): "+ (a>>b));
 }
}
```

JAVA statements

Statement is an executable instruction that tell the compiler what to perform. It forms a complete command to be executed and can include on or more expressions.

Control statements :-

1. If else - condition and mid-b
2. switch -
3. while
4. Do while
5. for
6. Break
7. continue
8. Return -
9. Labelled break, continue.

if else :-

Syntax :-

```
if (condition statement)
{
statement to be executed if condition becomes true.
}
else
{
statement to be executed if the above condition is false.
}
```

eg :-

```
int age=20;
if (age>=18)
{
S.O.P("age is greater than 18");
}
else
{
S.O.P("age is less than 18");
}
```

switch :-

Syntax

```
switch (bytes/short/int)
{
 case expression :
 statements
 case expression :
 statements
 default
 statement
}
```

While loop :- , mid-s

syntax -

While (condition statement → true)

{

statement to be executed when the condition becomes true and execute them repeatedly until condition becomes false.

}

Eg :-

```
int x=2;
while (x>5)
{
System.out.println("value of x:"+x);
  x++;
}
```

do while :- , mid-s

```
do
{ // code block to be executed
}
while (condition);
```

Eg :-

```
do
{
S.o.p ("Hello world");
i++;
}
while (i<0);
```

For loop :- , mids

⇒ for (initialization ; condition ; increment/decrement) {

statement to be executed until the condition is false

}

```
for (int x = 0; x < 10; x++)
{
System.out.println ("value of x: "+x);
}
```

break :-

=> Break is used in the loops and to skip the current executed & continues with nxt iteration

```
eg:- for (int i=0; i<50; i++)
     {
     if(i % 13 == 0)
     {
     continue;
     }
     System.out.println(" value of i: "+i);
     }
```

Continue :-

=> It makes the loops to skip the cured execution and continues with nxt iteration

```
eg:- for (int i = 0; i<50; i++)
     if (i % 13 == 0) {
     Continue;
     }
     System.out.println ("value of i: "+i);
     }
```

return :-

Return statement can be used to cause execution to branch back to the caller of the method.

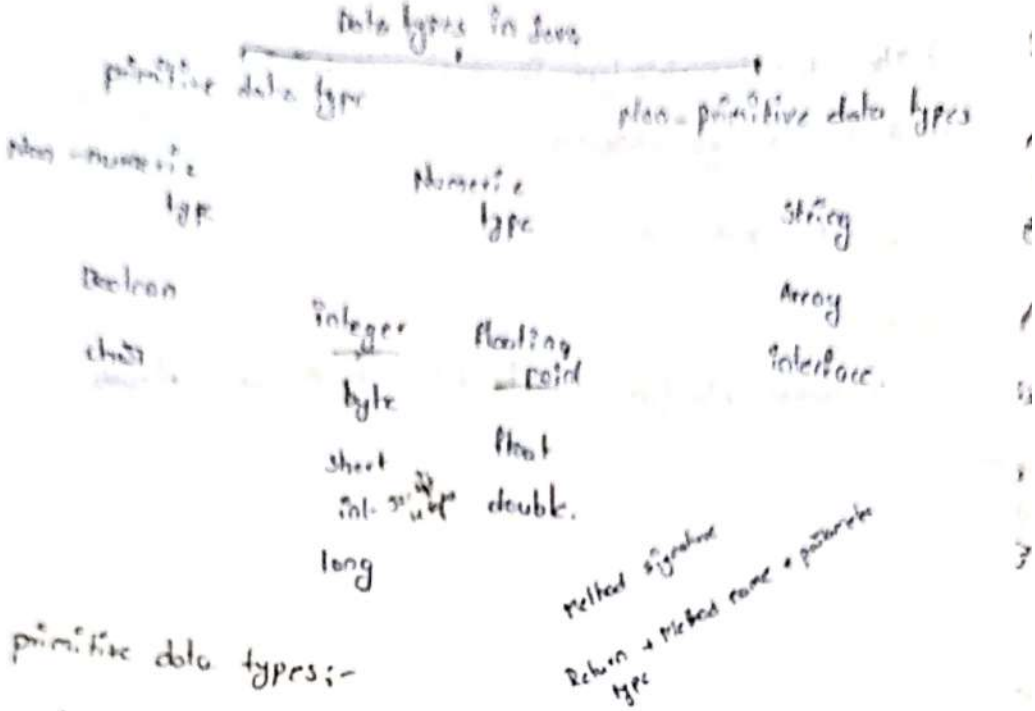
Labelled break, continue :-

=> Labelled break and continue statements will break or continue from the loop that is mentioned

created in nested loops.

JAVA data types :- to specify any type of data variable

1) or 2 types → Numeric 2) Non numeric



related signature
Return + method name + parameter type

primitive data types:-

Integer:-

1) A byte type variables takes 1byte of storage and can store values from -128 to 127.
2) The short data type is 16-bit (2 bytes). It has min value of -32,768 and max of 32,767.
3) The int data is 32-bit (4 bytes) signed two's complement integer, taking values from -2^{31} to $2^{31}-1$
4) The long data type is a 64-bit two's complement integer taking values from -2^{63} to $2^{63}-1$

Floating points:-

There are 2 types of floating point data types in Java - double and float.

float:-

1) A float gives you approximately 6 to 7 decimal digits
2) Needs 4 bytes of storage

Double :-

1) A double gives you approximately 15-16 decimal digits.
2) Needs 8 bytes of storage.

character :-

1) The character data type is single 16-bit uni-code char Text Format.
2) A char is single character.

eg:- char grade = 'A'

Boolean data type :-

A boolean data type will be declared by the union elements

Non-primitive data types :-

1) class :-

object is a data type that defines a template or blue print of an object.

2) object :-

object is a instance of class is used to specify.

3) strings :-

A string is a collection of characters. Java string is classified into two types.

1) Mutable
2) Immutable.

eg:- string a = "hello"

4) Array :-

Array is group of elements of same data type. Array specifies a common name.

eg:- string fruits [] = {"apple", "banana", "mango"};

5) Interface :- Interface is a collection of final variables and abstract method.

An interface in Java is used to specify the contract. or capability the implementing clss.

eg:- public interface Area {
public static compute (float r)
final float pi = 3.14f;

Type casting :-

=> Type casting is the process to converting one datatype to another datatype variable.

=> Type casting divide into 2 types.

1) Implicit
2) Explicit.

=> in general type casting supported internal.

=> Any user converted higher data variable to lower data var those of data is called narrow conversion.

Syntax:-

variable name = Data type (variable name)

eg:-

int i;
float a;
i = (int)a;

String Buffer Method

String buffer can be immutable means it can be altered any given string.

Approximately string buffer maintains 3 constructors and methods.

1) Sb = new string buffer();
2) Sb_1 = " " " (int index);
3) Sb_2 = " " " (string str);

The string buffer methods are of

1) length()
2) capacity()
3) Reversal()
4) Append()
5) insert()
6) delete()
7) charAt()

⇒ Method is a collection of variables & instructions.

⇒ Method signature.

returntype + method name + parameters

Eg:- void + venky (int a, int b)

ex:-

```
void display()
{
System.out.println("welcome");
System.out.println("Hp");
}
```

Global variables are used the entire the program.

Local variables are used in the same part of method(w) program.

Object:-

object is a any thing in real world

ex:- cycle, car

=> objects are created with the help of using new operator

eg:- Car C = New Car ( ) → constructor
↳ object.

=> Each and every thing considered as a object in the Java

=> object information reallocated from the memory using garbage collector.

=> garbage collector is a predefined mechanism in Java.

=> Garbage collector is taken care of automatically reallocated used object information from memory.

class:-

=> class is a blue print of object.

=> class is always act as a xerox copy of object.

=> class is a collection of variables & methods.

class structure

```
package packagename;
keyword. → class name   (class → user specific name.)
{
  global variables.
  user defined (pre-defined methods.
  Java main method
}// end of main class
```

open the class

Write a java program to implement quadratic equation using java.

```
import java.lang.*;
class quadratic
{
 public static void main (String [] args)
 {
  int a = 10;
  int b = 2;
  int c = 5;
  float root1, root2;
  float disc = b*b - 4*a*c
   if (disc>0)
  {
  root1 = (-b+ Math.sqrt (disc))/(2*a);
  root2 = (-b-b Math.sqrt (disc))/(2*a);
  }
  else if (disc == 0)
  {
   root1 = -b/(2*a);
   root2 = b/(2*a);
   }
  else
   {
    System.out.println ("No roots");
   }
   System.out.println ("root1:" + root1)
   System.out.println ("root2:" + root2)
  }
}
```

Write a java program of lecture details : Lecture name, Lecture salary, Lecture Adress, Lecture department.

```
import java.lang.*;
class Lecture
{
 string Lecture name;
 float Lecture salary;
 string lecture adress;
 string lecture department;
    void setdata ()
  {
   Lecture name = "venky";
   Lecture Salary = 15,000f;
   Lecture address = "Nellore";
   lecture department = "Data science";
  }
  void display()
  {
   system.out.println ("Lecture name is :"+Lecture name);
   System.out.println("Lecture salary is :" + lecture salary);
   system.out.println ("Lecture address :"+ Lecture.address);
   system.out.println("Lecture department is:" + lecture department);
  }
   public static void main (string[] args)
   {
    Lecture = new Lecture ();
        L.setdata ();
        L.display ();
    }
}
```

Write a Java program to implements given string is a palindrom or not.

```
import java.lang.*; (or) import java.util.*;
class palindrome test.
  {
    public static void main (string[] args)
     {
       string s = new string ("madam");
       string Buffer sb = new string buffer (s);
string buffer s1 = sb.reverse ();
       if (s.equals (s1.to string());
       {
    System.out.println ("Given string is a palindrome");
       }
    else
     {
      system.out.println ("Given string is not a palindrome");
        }
     }
  }
```

Constructor:-

=> Constructor is a hardware mechanism.

=> Hard ware mechanism is taken care of to prever any user application in the memory.

=> Constructors are created same as the class name.

=> Constructors are supported memory allocation and realleeation mechanisms (usage new and garbage collector)
allocation deallocation

=> Constructors are created using new operator.

=> Constructors are not required no return type.

=> Sometimes Constructors are associated public Modifier

Write a Java program to create student details using Constructor.

```
import java.lang.*;
class student <-- user specific
{
 string Sname;
  int SRNO;
  int Smarks;
 string Sadress;
 student ()
 {
 Sname = "venky";
 SRLNO = 2302;
 Smarks = 125;
 Sadress = "GIST";
 }
 void display()
 {
  system.out.println (" student name is ": + Sname);
  system.out.println ("student rollnumber is:" +SRLNO);
  system.out.println ("student marks is :" + Smarks);
  system.out.println ("student adress is :" + Sadress);
 }
```

```
public static void main (String args[])
{
    Student s = New student ()
        s.display();
}
```

Differences b/w Constructor & method

Constructor	Method.
1) Constructor is a collection of instructions	1) Method is a collection of variables & instructions.
2) Constructors are created same as the class name.	2) Methods are created using user-defined, pre defined, developers name.
3) Constructors are not required return type. eg:- No need of void keyword student()	3) Methods are required return type eg:- Need of void keyword void student();
4) Constructors are using New operator	4) Methods are not using New operator
5) Constructor is a Hardwaid mechanism	5) Method is not a Hardwaid mechanism.
6) Constructors supported over-loaded.	6) Methods are supported overloading & over riding mechanism.
7) Constructors supported passing parameters (or) Arguments. eg:- student() { Sname = "Lahari"; SRLNO = 254; Saddress = "Nellore"; Smarks = 100;	7) Methods are supported passing parameters or Args eg:- setdata() { Sname = "Lahari"; SRLNO = 254; Sadress = "Nellore"; Smarks = 100;

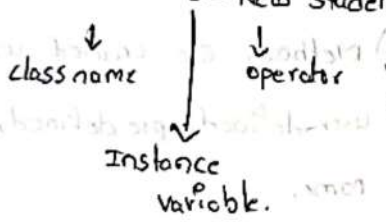
Types of Constructor :-

Constructors are different Constructors

1) default Constructor
2) Argument or parameter Constructor
3) Copy Constructor

1) Default Constructor :-

Default Constructor automatically provided by the system or Java software.



⇒ Here automatically initialized variables.

Eg :- Refer student details program.

2) parameter constructor or Argument Constructor.

parameter constructor should maintain atleast one parameter or Argument variable.

⇒ parameter Constructors are very useful for users.

⇒ parameter Constructors are supported Constructor overloading mechanism.

mid-12.

Constructor overloading :-

1) Constructors are should maintain same name.
2) Java class supported more than one Constructors
3) Constructor overloading should maintain different properties are

⇒ No. of parameters

⇒ order of parameters

4) Sometimes constructor overloading mechanism is implemented with the help of using 'this' keyword.

Eg:- write a Java program to implement student details using constructor overloading mechanism.

```
import java.lang.*;
class student
{
   string Sname;
    int SRLNO;
   string Sadress;
    int Smarks;
   student (string sn)
   {
    This.Sname = sn;
   }// end of first constructor
   student (string sn, int SRL)
   {
    This.Sname = sn;
    This.SRLNO = SRL;
   }
   student (string sn, int SRL, string SA)
   {
     This.Sname = sn;
     This.SRLNO = SRL;
     This.Sadress = SA;
   }
  student (string sn, int SRL, string SA, int SM)
   {
     This.Sname = sn;
     This.SRLNO = SRL;
     This.Sadress = SA;
     This.Smarks = SM;
   }
   void display()
```

```
{
System.out.println("the student name is:"+sn);
system.out.println("the student roll no is:"+sRL);
system.out.println("the student adress is:" sa);
system.out.println("the student marks is:"+sm);
}
public static void main (string [] args)
{
student s1= New student ("[redacted]");
student s2= New student ("[redacted]", 1902);
student s3= New student ("[redacted]", 1902, "Nellore");
student s4= New student ("[redacted]", 1902, "Nellore", 190);
        s1.display()
        s2.display()
        s3.display()
        s4.display()
    }
}
```

This keyword :-

⇒ This keyword to the users java providers

→ This keyword is used to differentiating of instance/variable / global variables, local variables

⇒ This keyword is always referring a current object

⇒ This keyword is applied on local variables / global variables using dot operator.

write a java program to implement Blue, Red green to using This keyword.

```
import java.lang.*;
class colourtest
{
string Red;
string Blue;
string Green;
void setdata (string R, string B, string G)
{
 This. Red=R;
 This. Blue = B;
 This. Green = G;
}
void display()
{
 System.out.println("the red colour is:"+Red);
 System.out.println("the blue colour is:"+Blue);
 System.out.println("the green colour is:"+Green);
}
public static void main (string[]args)
        colour test c = New colour test
          c.setdata ("Red","Blue","Green")
          c.display();
        }
}
```

To implements student's data :-

```
import java.lang.*;
class (this& test)
{
String Sname;
int Sid;
int Rollno;
void setdata (string, int si, int SRL)
{
  This.S Name =sn;
  This.Sid =si;
  This.SRollNO =SRL;
}
void display()
{
system.out.println("the student name is:"+Sname);
system.out.println("the student id is ="+Sid);
system.out.println("the student Roll No is:"+SRL);
}
public static void main (string args[])
{
  this& test T= new this& test();
      T.setdata ("kumar", 1234, 1901);
       T.display();
   }
}
```

Access specifier :-

Access specifier is given addition meaning to the class, Method and variables.

Access specifier are divided into 3 types.

1. Abstract
2. final
3. static

1. Abstract :-

=> Abstract is a access specifier

=> Access specifier means is given additional meaning to the class, Methods of variables.

=> Abstract can be associated with class.

=> Abstract can not be associated with variables.

=> Abstract class is a collection of Abstract methods and general variables.

```
Abstract class
{
 Abstract methods;
 Normal variables;
}
```

=> Abstract method is a collection of variables & instructions.

=> Abstract class are used to creating of a users classes.

=> Abstract method are not having any body - every abstract method end with a semicolon(;).

=> Abstract methods are implemented in feature classes.

Abstract class syntax

```
Abstract no return value.
 + method name + parameter list;
 sai (int a)
 //This method does not having no body.
```

2. final :-

⇒ It is a access specifier

→ It is a keyword.

⇒ Access specifiers means giving additional meaning to class, variables and methods.

→ final can be associated with class

→ final can be associated with variable.

→ final can not be associated with methods.

→ sometimes final can be associated with methods.

Static :-

→ It is a access specifier

→ Access specifiers means gives additional meaning to the class, variables and methods.

→ static is a keyword.

→ static can be associate with, method, variable.

→ static can not be associated with class.

→ sometimes static methods refer non static variable.

→ static method is always required static variables.

eg:- Java main method

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

eg:- import java.lang.*;

```
class static test
{
static int count=0;
static int x()
{
system.out.println("The Incrementation Count Value is :"+count++);
}
```

```
public static void main (String [] Args)
{
    static test st= New static test();
        st.x
    }
}
```

Modifier :-

⇒ Modifier is given additional meaning to the class, packages, interfaces.

⇒ Modifiers are mainly divided into 4 types.

1) public
2) private
3) protected
4) default (or) friendly

public :-

1) public is a modifier
2) public can be associated with class, subclass, package classes and interfaces.
3) public classes is access to other classes (or) package classes is access to other package classes.

private :-

1) private is a modifier
2) private classes are accessible to other classes within the same package.
3) " " is always requires private methods only.

protect :-

1) protect is a modifier
2) " classes is possible to access into subclasses only
3) " modifier applicable for within the classes and within packages.

Default :-

1) It is a modifier

2) Default classes is accessible to other classes (Default modifier same as public)

3) Default classes are accessible to within same package classes or different package classes.

write a Java program to implements multiple inheritance / interface final and abstract keywords

(or)

write a Java program to find area of circle and semicircle.

Interface is a collection of Abstract methods & final variables

interface act as both keyword & technique. inter

Interface structure

Interface created same as class construction

```
interface interface structure.
{
  Abstract methods;
   final variables;
}
```

Interface technique provides implements keywords to the user.

```
Interface Area
{
  public float compute (float x);
     final float pi = 3.14f;
  }
     class circle Area implements Area
     {
```

```
public float compute (float r)
{
 return (pi*r*r)
}
class semicircle area implements Area
{
 public float compute (float r)
 {
 return (0.5 * pi *r*r)
  }
}
class main
{
public static void main (String[] Args)
{
 ~~Area~~ A: Area A;
  circle Area CA = New circle Area();
          A=CA;
  System. out. println ("The circle Area is:" + CA. Compute (5.34f));
   Semi circle. Area SA = New semi semicircle Area()
          A=SA;
  System. out. println ("The semicircle Area is :" +SA . compute(5.34f));
 }
}
```

To save this application using file name. java ⇒ main. java

To compile - Javac. Main. java

run - Java. file name.

output:-

circle Area

Semi circle Area

outer and inner classes

1) outer & inner classes supported inheritance properly of java.

2) Inheritance means aquaring the properties from super class to sub class

(or)

one class to another class.

3) super classes is technically class called Base class. (or) outer class

4) sub class is technically called derived class (or) inner class

5) To exchange of any amount of information from outer to inner classes with the help of using Nested classes.

6) To exchanging any amount of information from one class to another class using extends keyword.

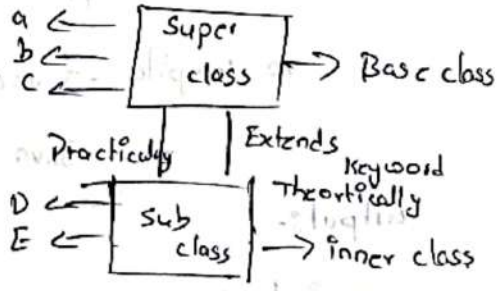
7) Nested classes are outer and inner supported over riding Mechonism.

8) over riding mechanism implemented with the help of using super keyword

Eg:- write a program for inner & outer classes.

```
class Super class
{
    String a;
    int b;
    int c;

Super class()
{
    a = 12;
    b = 14;
    c = 24;
}
```



```
void display()
{
System.out.println("the value of a:"+a);
System.out.println("the value of b is:"+b);
System.out.println("the value of c is:"+c);
 }
}// end of super class
 class subclass extends superclass
 {
   int d;
    int c;
  subclass()
  {
   d=55;
   c=2;
  }
  void display()
  {
   super.display()
   System.out.println("the value of d is:"+d);
   System.out.println("the value of c is:"+c);
  }
 } //end of sub class
class main
{
 public static void main(String Args[])
 {
  super class, sp = New super class.();
     sp.display();
     subclass sc = New subclass();
              sc.display();
     }
  }
```

Inheritance :-

Aquiring the properties from one class to another class are. to exchanging to the any amount of information from superclass to sub class.

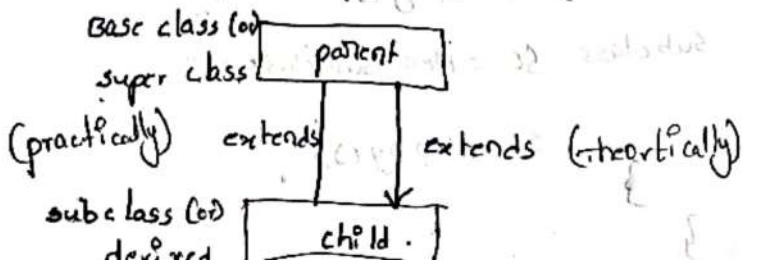
Inheritance are divided into 4 types.

1) single inheritance (or) single level
2) Multi level inheritance.
3) Multiple inheritance
4) Hiarachical inheritance.

1. Single (or) single level inheritance :-

- singk level inheritance is used only two classes
- the two class names are
  1) super class
  2) subclass.
- TO exchanging any information from super class to subcla using extends keyword
- super class is techically called as base class
- sub class is basically called as derived class
- users are same theoritically super class extends subcl
- single level inheritance practically implemented using bottom to top up approach in (subclass extends Superc

eg:-



program to implementing single level inheritance using parent & child classes.

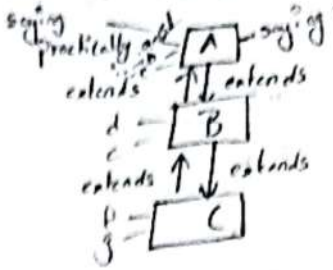
```
import java.util
import.java.lang.*;
class Parent
{
  string Name;
  int    age;
  string address;
  Parent()
  {
    name = "venky";
    Age = "48";
    Address = "Nellore";
  }
  void display(){
  system.out.println("parent Name is :"+name);
  system.out.println("parent Age is :"+age);
  system.out.println("parent Address is :"+address);
  }
}
class child extends parent
{
  string childname;
  int Age;
  child(){
    child name = "prabha";
    cAge = 17;
  }
  void display(){
  system.out.println("child name is :"+childname);
  system.out.println("child Age is :"+cAge);
```

```
    }
}
class main{
    public static void main(String[] args)
        parent p = new parent();
        p.display();
        child c = new child();
        c.display();
    }
}
```

Multi level inheritance:-

- Inheritance means aquiring the properties from one class to another class.
- Inheritance multilevel inheritance require more than two classes.
- Multilevel inheritance is used 2 keywords name:-
  1. extends keyword.
  2. super keyword.
- Extends keyword is used to exchanging of any information from one class to another class.
- Super keyword is used to method overriding purpose.
- Method overriding supported to retriving of information from upper level class method to lower level class method
- Method overriding is always maintain two methods are having same name
- Super key associated with methods using dot operator
- super key worked at only one level inheritance.

example for multi level inheritance :-




```
import java.lang.*;
class A
{
 string a;
  int b;
  int c;
  A()
  {
  a= "venky";
   b=20;
   c=30;
  }
  void display()
   {
   system.out.println("value of a:"+a);
   system.out.println("value of b:"+b);
   system.out.println("value of c:"+c);
      }
  } // end of class A
class B extends A
{
   string d;
     int e;
    B()
    {
     d="chinnu";
     e=" 40;
    }
    void display()
    {
     super.display();
     System.out.println(" value of d:"+d);
     system.out.println(" value of e:"+e);
    } // end of class B
```

```
class C extends B
{
 int f;
 int g;
 C()
 {
   f=50;
   g=60;
   }
  void display() void show()
  {
   super.display(); super.show();
   system.out.println("value of f:"+f);
   system.out.println("value of g:"+g);
    }
  } // end of class c
class main
 {
  public static void main (String args)
  {
    C c1 = new C();
       c1.display();
       }
 }
```

output:-

```
The value of a is venky
The value of b is 20
The value of c is 30
The value of d is chinnu
The value of e is 40
The value of f is 50
The value of g is 60
```

- to save this application filename.
- to compile this using javac filename.
- to run this appl using java filename

Multiple inheritance :-

- Multiple inheritance :-
- Inheritance means aquiring the properties from one class to another class.
- Interface is a collection of abstract methods and final variables
- Interface act as both keyword and technique
- Interface constructed using class constructor.

Interface structure

Interface created as same as class construction.

```
interface interface structure
{
  Abstract Methods;
  final variables :
}
```

- Interface technique provides implements keywords to the user
- Method overriding is always maintain two methods are having same name.
- super key associated with methods using dot operator (.).
- Method overriding support to relation of inherit.upper level class to lower level class method.

Ex:-

Program :-

```
interface area
{
public float compute (float r);
  final float pi = 3.14f;
}
class circle area implements Area
{
```

```
public float compute (float r)
{
 return (pi * r * r)
}
class semiCircleArea implements area
{
public float compute (float r)
{
 return (0.5f * pi * r * r)
 }
}
class main
{
 public static void main(String [] args)
 {
  AreaA;
   circleArea CA = new circleArea();
          A=CA;
  System.out.println("The circle area is:"+CA.compute(5.34f));
   semiCircleArea SA = new semiCircleArea();
          A=SA;
  System.out.println("The semi circle area is:"+SA.compute(5.34f));
   }
 }
```

output:-

Circle Area is 89.538994

semi Circle area is 44.769497

write a java program to find the area of sphere & hemi sphere using multiple inheritance.

```
interface area
{
public float compute(float r);
final float pi=3.14f;
}
class sphere_area implements area
{
public float compute(float r)
{
return (4*pi*r*r);
}
}
class hemi_sphere area implements area
{
public float compute(float r)
{
return (3*pi*r*r);
}
}
class main
{
public static void main(string args[])
{
area A;
sphere_area S=new sphere_area();
A=S;
System.out.println("the sphere area is:"+S.compute(6.14f));
hemi_sphere area SA=new hemi_sphere area();
A=SA;
```

```
System.out.println("the hemi sphere area is :"+SA.compute(6.14));
    }
}
```

outputs:-

the sphere area is :473.50696

the hemi sphere area is :355.13022.

Array :-

- Array is a group of same datatype elements.
- Array declaration supported two types. that are:

syntax:-

1. Datatype. arrayname [ ] , Ex:-int S[];
   ex2:- int SRLNO[];
2. Arrayname Datatype [] , Ex:-Sint[];
   ex2:- SRLNO int[];

- Create memory locations for array using objects a
- Combining of two and three points.

syntax :-

1st declaration Datatype arrayname [] = new Datatype [];

for ex:- int SRLNO[] = new int [];

2nd declaration arrayname [] Datatype = new datatype [];

- put data values into memory locations.

Datatype arrayname [] = new Datatype [Data values]

Ex:- int SRLNO [] = new int [5]

- Array is always maintain length property.
- length property sending into arrays using dot operator

write a java program to create student rlno using array.

```
import java.lang.*;
import java.util.*;
class arrayTest
{
int SRLNO[] = new int[5];
Scanner S=new Scanner(System.in);
int i;
public static void main(String args[])
{
for (i=0;i<length;i++)
{
SRLNO[i] = S.nextInt();
}
 for (i=0;i<length;i++)
{
System.out.println("The student Rollnumber is:"+SRLNO[i]);
   }
  }
 }
```

write a java program to create student name using array.

```
import java.lang.*;
import java.util.*;
class array test
{
String Sname[] =new String[0]
Scanner S= new Scanner(System.in)
public static void main (String[] args){
for (int i=0; i<length ;i++){
    Sname[i] = Snext.str();
   }
   for (int i=0;i<length();i++){
   syst
```

```
System.out.println("student name :"+Sname(i));
  }
 }
}
```

write a java program to create student roll number & student name using array.

```
import java.lang.*;
import java.util.*;
class array test
{
 int SRLNO
 public static void main (string[]args)
 {
  for(int i=0;i<length;i++)
  {
int SRLNO [] = new int [10]
String Sname [] = new string [10]
Scanner s= new Scanner (system.in);
int i;
 for (i=0;i<length;i++)
{
SRLNO [i] = S.next int ();
}
Sname [i] = S.next string ();
}
system.out.println("the student roll number is :"+ SRLNO);
system.out.println("the student name is :"+ Sname);
 }
}
```

Two dimensional array :-

Syntax :-

Datatype arrayname[][] = new Datatype [][]
operator size size.

Ex:- int a[][] = new int [3][3];

W.A.P to implements matrix multiplication.

```
import java.lang.*;
class matrix multiplication test
{
public static void main (String[] args)
{
 int a[][] = {{10,20,30},{4,5,6},{7,8,9}};
 int b[][] = {{1,2,3},{4,5,6},{7,8,9}};
 int c[][] = new int [3][3];
//
 for (int i=0; i<3; i++) {
 for (int j=0; j<3; j++){
      c[i][j] = 0;
  for (int k=0; k<3; k++){
  c[i][j] = c[i][j] + a[i][k] * b[k][j]
   }
  }
 } System.out.println("the matrix is :");
  for ( int i=0; i<3; i++){
  for (int j=0; j<3; j++)
  {
  System.out.println ("The A matrix is"+a[i][j]);
  }
  System.out.println(" ");
}
```

```
// print B matrix is
for (int i=0; i<3; i++) {
for(int j=0; j<3; j++){
System.out.println("the B matrix is :"+b[i][j]);
}
System.out.println();
}
}
// C matrix is
for (int i=0; i<3;i++) {
for (int j=0; j<3; j++){
System.out.println("c matrix is:"+c[i][j]);
}
System.out.println();
} //for loop
} //main method
} // class
```

output :-

A is :

B is :

C is :

w.A.p to implements addition, subtraction and multiplication of matrix using switch case.

```
import java.lang.*;
class matrix{
public static void main(String[]args){
```

```
import java.lang.*;
import java.util.Scanner;
class matrix_operations{
 static Scanner sc = new Scanner(System.in);
 //creation method
 static void creation(int[][]arr, int a, int b){
  System.out.println("enter element for matrix :");
 for(int i=0; i<a; i++){
 for(int j=0; j<a; j++){
    arr[i][j]=sc.nextInt();
     }
    }
 }// Display method.
 static void display(int[][]arr, int a, int b){
 for(int i=0; i<a; i++){
 for(int j=0; j<b; j++){
    System.out.println("arr[i][j]+ ");
 }
  System.out.println();
 }
}
//matrix addition
static int[][] add(int()() arr, int a, int b, int(x) arr1, int z, int n),
  int res[][]=new int[a][n];
  for(int i=0; i<a; i++){
  for(int j=0; j<a; j++){
```

```
res[i][j] = arr[i][j] + arr1[i][j];
 }
}
return res;
}
Matrix Subtraction
static int[][] sub(int()() arr, int a, int b, int()() arr1, int m, int n){
 int res()() = new int[a][n];
 for(int i=0; i<a; i++){
 for(int j=0; j<a; j++){
   res[i][j] = arr[i][j] - arr[i][j];
  }
 }
  return res;
 }
  // matrix multiplication
  static int()() mul(int()() arr, int a, int b, int()() arr1, int m, int n){
    int res[][] = new int[b][n];
    for(int i=0; i<a; i++){
    for(int j=0; j<b; j++){
    for(int k=0; k<m; k++){
      res[i][j] += arr[i][k] * arr1[k][j];
      }
    }
   }
   return res;
 }
// main method
public static void main(String[] args){
//matrix one
int a,b;
```

```
System.out.println("enter no of rows of matrix A:");
a = sc.nextInt();
System.out.println("enter no of columns for matrix A:");
b = sc.nextInt();
int arr[][] = new int[a][b];
Creation(arr, a, b);
// matrix two
int m, n;
System.out.println("enter no of rows for matrix B:");
m = sc.nextInt();
System.out.println("enter no of columns for matrix B:");
n = sc.nextInt();
int arr1[][] = new int[m][n];
Creation(arr1, m, n);
// Display first matrix
System.out.println("matrix A:");
  display(arr, a, b);
// Display second matrix
System.out.println("matrix B:");
display(arr1, m, n);
System.out.println("enter matrix operations" +
                   "1. Addition \n" +
                   "2. Subtraction \n" +
                   "3. multiplication \n");
int choice = sc.nextInt();
switch(choice){
    case 1:
     if(a==m && b==n){
```

```
System.out.println("matrix Addition");
System.out.println(" Resultant matrix");
int [][] res =add (arr, a,b, arr1, m,n);
}
display(res,a,n);
else {
System.out.println("invalid matrix");
}
break;
case (2):
if (a==m && b==n){
System.out.println("matrix Subtraction");
System.out.println(" Resultant matrix ");
int[][] res1=sub (arr, a,b, arr1, m,n);
display (res1, a,n);
}
else {
System.out.println("invalid matrix");
}
break;
case (3):
if (b==m){
System.out.println("matrix multiplication");
System.out.println("Resultant matrix ");
int [][] res = mul (arr, a,b, arr1, m,n);
display (res,b,n);
}
else
{
```

```
System.out.println("invalid matrix");
}
break;
default:
System.out.println("exit");
}
}
}
```

output:-

1. Addition
2. Subtraction.
3. Multiplication.

UNIT - IV

Introduction of packages:-

- package is a collection of classes, methods and interfaces.
- class is a collection of variables and methods.
- Interface is a collection of final variables and abstract methods.
- packages are again divided into two types.
  1. System defined / pre defined.
  2. User defined

1. System defined packages :-

- System defined packages are

1. Lang	2. AWT	3. Net	4) SQL
5. string	6. IO	7. util	8. Math.
9. applet	10. awt.event.	11. javax.string.	

1. Language (lang)

- System provides a language package to the user ;
- It is a default package
- Any user not properly using language package at the time system provides language package to the user
- Language package retriving memory into user applica using import statement

import syntax:-

import java software.packge.name.*;/

import java sw name.package.name.classname;

Ex:- statement

1. import java.lang.*;
   s/w → package name.
   operator

2. import java.lang.class A;

- Language package provides different information to the user that are.
  1. class template information
  2. string
  3. string buffer
  4. threads

2. AWT package :-

- AWT stands for (Abstract window Toolkit)
- Abstract window act as a super window or superframe
- Any user one's to creating of a own window using features of superframe (or) super window
- To retriving features from super window to user windows is possible using extends keyword.
- Abstract window hiding of complexity components information to the user
- Abstract window supported different components are

1. button	2. Textfield	3. Text area	4. Radio button
5. label	6. choice box	7. check box	8. list
9. Scrollbar	10. menu		

- In additionally AWT support layouts and listeners
  1. Layouts
     Border layout
  2. Flow layout
  3. Grid layout.
  4. Grid bag layout.
  5. Card layout.
  6.

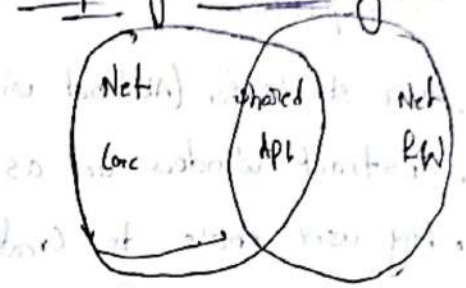
Listeners :-

1) Action Listener
2) Key Listener
3) mouse Listener.
4) mouse motion Listener
5) Item Listener.

write a java program to create a own frame by using super key.

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

Net package blocks diagram:-



3. Net package :-

- It is a communication package
- To transfer any amount of data from client to server (or) Sender to receiver
- Net package supported different types of protocols. they are:

1. UDP (User datagram protocol)
   i) It is a connection less protocol.
   ii) (It is a connection). protocal means framing of set of rules
   iii) Some times not properly transfer data from client to server

2. TCP/IP.
   i) TCP stands for (Transmission Control protocol)
   ii) IP stands for (Internet protocol)
   iii) These protocols supported connection oriented mechanism
   iv) Connection oriented means safely transfer any data from client to server.

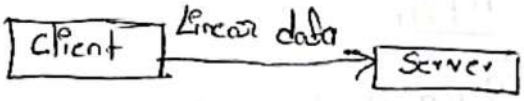
3. HTTP.
   i) HTTP stands for (Hyper Text Transfer protocol)
   ii) It is a connection oriented protocol
   iii) HTTP protocol initially established communication path b/w client & server.
   iv) It is taken care of package information safely transfer from client to server.

- Net package additionally supported different technologies.
  1) LAN (Local area N/w)
  2) MAN (Metropolitan Area N/w)
  3) WAN (wide Area N/w)

4. IO package :-

- IO stands for Input output streams
- Stream is a linear flow of data that travel from safely from client to server.
- IO streams are supported either char streams (or) byte streams
- Any data converted to another datatype is possible with the help of using type conversion (or) type casting

ex:-



- IO streams are supported is also supported different types of streams that are
  1) Data I/p stream
  2) O/Data o/p stream
  3) File i/p stream
  4) file writer stream
  5) Buffer o/p stream

5. SQL package :-

- SQL stands for (Structure query Language)
- It is a backend language.
- SQL supports differents commands versions ex:

1. Create
2. insert
3. delete.
4. update.
5. Select

Create :-

Syntax :-

Create table tablename (att1 .. Dt (size). Att2 .. Dt(size) ... Attn Dt. size());

Insert :-

Ex: Create table IIDs (Student rollno. number type(2) .. Sname varchar(6) Smarks no(5));

Insert:

insert into table name values(att1, att2 - ... attn)

Select: Syntax:

Select * from table name.

ex:

Select * from tab IIDS and cs table, here retriving all attribute values from the IIds & cs.

S.NO	S Name	S. Marks
1901	[illegible]	12
1902	chinnu.	13
1903	2	14

String & package :-

- Its extended package of swings.
- It is supported light weight components (button, choice box, list etc).
- swings components are identify 'J'.

Math package:-

- Math package supports different mathematical functions are square root, sine, tan, cosine.

UTIL package:-

- UTIL package supported scanner object & array list
- UTIL package is also supported:
  - lovectors

2. Random numbers.

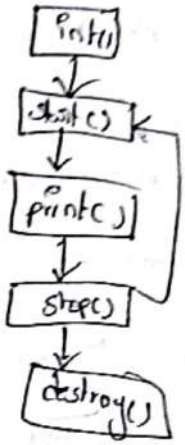
3. Linked list

4. Date.

5. Time.

Applet:

- Applet is a small Java application program.
- Applets are supported web applications
- Applets are supported different types of html tags & Creating of a graphics



ex:

<applets> Begin
</applets> end.

/* <Applet code = "sai.class" Height =50 width =50>
</Applet>
*/

Applet supported 3 attributes.

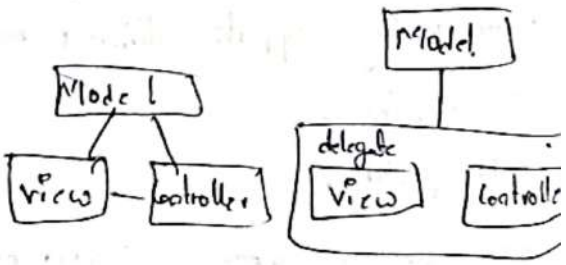
1. code attribute. supported . (dot) class file.

String:

- It is the extension package of AWT.
- String package supported light weight Components.
- String package is supported J Components.

ex:

1. button.
2. label.
3. text field
4. JText Area,
5. choice box
6. checkbox
7. J Scroll box
8. list
9. J Radio button.
10. J Combo box.



Enumeration :-

Enumeration are a special data type in java that enable for a variable to be a set of perdefined constants it includes compass directions values of NORTH, SOUTH, EAST and WEST.

program :-

```
import java.lang.*;
class enumeration
{
Declare enum type.
enum day {
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
}
Create a Day object
  Day day = Day.MONDAY;
/print the day
System.out.println(day)
}
```

Output :-

MONDAY

Math class :-

Java.lang.math class methods help to perform numeric operations like square, square root, cube, cube root, exponential & trigonometric operations.

Declaration :-

```
final class math
  extends object
```

program :-

```
class Javamath Example {
public static void main (String[] args) {
double x=28;
```

```
double y = 4;
//return the maximum of two numbers
System.out.println("maximum no of x and y is:"+math.max(x,y));
//return the square root of y
System.out.println("square root of y is:"+math.sqrt(r));
//return 28 power of 4 i.e, 27 + 28*28*28 + 27
System.out.println("power of x and y is:"+math.pow(x,y));
}
}
```

Output:-

maximum number of x and y is 28.0

Square root of y is:2.0

power of x and y is: 614656.0

WRAPPER class :-

wrapper classes are classes that encapsulate primitive datatypes.

program:-

```
public class main
{
public static void main (String[] args) {
integer myint=5;
Double mydouble = 5.99;
character mychar = 'A';
System.out.println(myint);
System.out.println(mydouble);
System.out.println(mychar);
}
}
```

Output:-

5

5.99

A

Auto Boxing:

In Autoboxing the java compiler automatically converts primitive types into the corresponding wrapper class objects.

program:-

```
import java.util.ArrayList;
class mainclass
{
public static void main (String [] args){
ArrayList <Integer> list= new ArrayList <>();
//Auto boxing
 list.add (5)
 list.add (6)
System.out.println (" ArrayList :"+list);
 }
}
```

- output

Array list (5,6)

Auto unboxing :-

In autobox : the java compiler automatically converts wrapper class objects into their corresponding primitive types.

program:-

```
import java.util.ArrayList
class mainclass
{
public static void main (String [] args){
 ArrayList <Integer> list = new arrayList<>();
// Autoboxing
  list.add (5);
  list.add (6);
System.out.println("Array list :"+list);
```

```
// unboxing
int a = list.get(0);
System.out.println("value of index 0:"+a);
}
}
```

Output:

Array list [5,6]

value at index 0:5

* ~~Formation~~

formatter.class :-

The formatter is a built in class in Java used for layout justification & alignment, common formats, for numeric, string & date/time data, and locale-specific output in Java programming.

programs:-

```
import java.util.formatter;
class example {
public static void main (String [] args){
formatter formatter = new formatter();
// format a string
formatter.~~string~~ format ("Hello, %-s!", "word");
// format a number
formatter.format ("%d", 100);
//format.date
formatter.format("%tF", new date());
//Get the formatted string
String formattedString: formatter.to.string();
System.out.println (formatted-string);
}
}
```

* Random class :-

Random class is used to generate pseudo-random numbers in Java. An instance of this class is thread-safe.

Program :-

```
// A Java program to demonstrate random number generation.
// using java.util.Random:
import java.util.Random;
class GenerateRandom {
public static void main(String[] args) {
// Create instance of Random class
Random rand = new Random();
// Generate random integers in range 0 to 999
int rand_int1 = rand.nextInt(1000);
int rand_int2 = rand.nextInt(2000);
```

Exception Handling:-

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

fat program:-

```
import java.lang.*;
class importance test
{
public static void main (String [] args) {
int a;
int b;
int c;
a=5;
b=0;
c= a/b;
}
}
```

output:-

Exception in thread "main" java.lang.ArithmaticException: /by zero
at importance.test.main(importancetest.java:23)

Exception Hierarchy:-

- The Exception classes has two direct subclasses: Error & exception
- The exception class is used to indicate errors that can be handled by the program, such as invalid input or file not found errors.
- The exception classes has two further subclasses:
  checked exceptions & unchecked exceptions

program:-

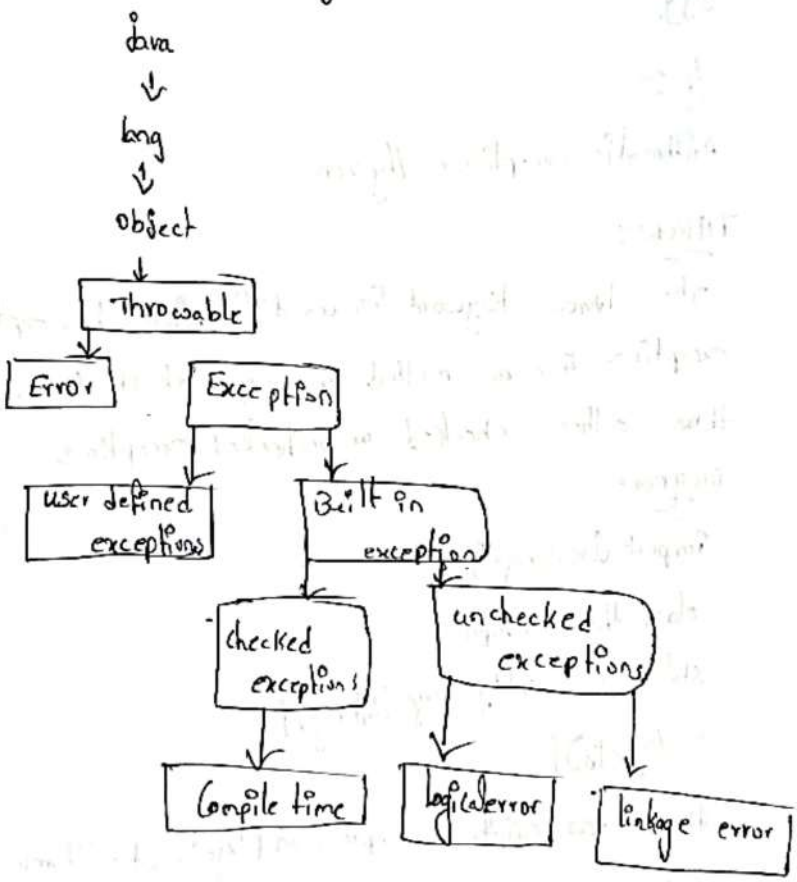
```
import java.lang.*;
class importance test{
public static void main (String [] args) {
try{
```

```
}
int a=5;
int b=0;
int z;
z = a/b;
}
catch (ArithmeticException Ae) {
System.out.println ("Runtime divide by zero error");
}
}
}
```

Output:-

Runtime divide by zero error.

block diagram of exception Hierarchy :-



Try :- Scope of any exception associated with it. It detects the exception.
Second use for try catch blocks :-

Program :-

```
import java.lang.*;
class main {
public static void main (String [] args) {
try {
int divide by zero = 5/0;
System.out.println ("first of. code in try block");
}
catch (Arithmetic exception e)
{
System.out.println ("Arithmetic exception =>" +e.getmessage 0);
}}}.
```

o/p :-

Arithmetic exception => /by zero.

THROW :-

The throw keyword is used in java to explicitly throw an exception from a method or any block of code. It can be used to throw either checked or unchecked exceptions.

program :-

```
import java.lang.*;
class throw example {
static void validate Age. (int age) {
if (age<18) {
throw new Arithmetic expression ("Age is not valid, must be 18 or older");
}else {
System.out.println ("welcome to the club!!");
}
public static void main (String [] args)
{
```

```
try {
validate Age(15);
} catch (Arithmetic exception e){
System.out.println("exception caught :"+e.getmessage());}
validate Age(20);
}}
```

O/P:-

Exception Caught : Age is not valid, must be 18 or older
welcome to the club!

Throws Keyword:-

The throws keyword in Java is used in method signature to indicates that a method might throw one or more exceptions

Syntax:-

```
throw new exception-type("error message");
```

Program:-

```
class Throwkey1 {
public static void main (String args){
try{
checkAge (15) ;{
Catch(Exception e) {
System.out.println (" Caught an exception: "+e.getmessage());
}
}
static void checkAge (int age){
if (age<18) throw new IllegalArgument exception ("Age must be 18 or older.");
}
}.
```

O/P:-

Caught an exception : Age must be 18 or older.

finally :-

The finally block is used to put important codes such as cleanup code e.g closing the file or closing the connection.

A finally contains all the crucial statement regardless of the exception occurs or not

Program :-

```
import java.io.*;
class UFO {
public static void main(String [] args) {
try {
System.out.println("inside try block");
System.out.println(34/0);
}
Catch (Arithmetic exception e){
System.out.println("catch: exception handled.");}
finally {
System.out.println("finally: i excute always.");}
}
}
```

o/p :-

inside try block

Catch: exception handled

finally: I excute always.

Creation of package Creation

step 1: Declare the package.

syntax:

package package name

keyword — depends on you.

ex:- package sai;

step 2: Declare the class.

syntax:

class class name.

keyword — depends on you.

ex: class sai1.

step 3: class Declared as public.

Syn:-

public class sai1

{

:

}

modifier

Step:- Create subdirectory under the main directory.

ex:- C:>My pict> cc6/cc1 /cc2 >MD subdirectory name

MD - make directory

Note:- packagename (or) Same as the subdirectory name.

(packagename and subdirectory names are Same).

C:>Mypic.>cc1>MD sai;

C:>Mypic.>cc1>MD Sai;

Save the package class under the subdirectory save: sai1.java.

Compile package file under the subdirectory

ex:- javac filename.java javac sai1.java

check atleast once the created or not created dot class file under the subdirectory.

write a java program to create a own package.

```
import java.lang
package sai;
public class sai1
{
public static void main (String [] args)
{
void display ()
{
System.out.println ("hai");
System.out.println ("welcome");
System.out.println ("javalab");
}
}
}
```

How to access own package into user classes

to access own package information to using import ownpackage

syntax:

import own packagename.package classname;

ex:

import sai.sai1;

write a java program to implements own package accessed into user classes

```
import sai.sai1;
class sai2
{
public static void main (String [] args)
```

```
sai1 s1 = new sai1();
     s1.display();
          ↓
     dot operator
}
}
```

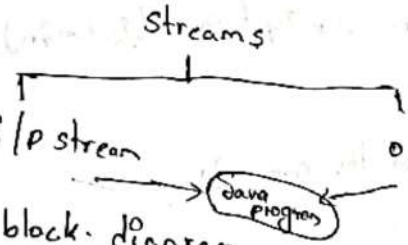
- TO. mandatory shift from subdirectory to main directory to save the sai2 application under the MTO sai2.java.
- Compile the sai2 application under the main directory javac sai2.java.
- Run the sai2 application under the MTO. java filename java sai2.

o/p:-

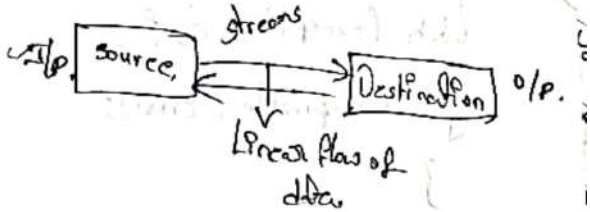
Hi welcome javalab.

Stream:-

- Stream is a linear flow of data.
- Linear flow of data travel from source to destination
- streams are again divided into two types.
  1. I/p stream.
  2. o/p. stream



- Consider a block diagram representation of a stream.



- Create. streams are created with the help of three criterias
1) Create the stream.
2) Call the methods. [readline → methods calling integer.parseInt() for calling values
3) Close the stringstream

Streams are accepted in two ways.

* byte stream - Byte stream provide a convenient means for handling i/p & o/p of byte. byte streams are used when reading or writing binary data.
* character streams - character streams are more efficient than byte streams. character streams provide a convenient means for different streams are converted into either byte stream (or) character stream using type conversion (or) type casting.

Classes: byte streams classes are defined two abstract classes i/p & o/p different types. Reader & writer streams are different types.

		character streams. class	
1) Data i/p	1) Data o/p	1. buffer reader	8. filter writer
2) Buffered i/p	2) byte array i/p	2. buffer writer	9. i/p stream reader
3) Buffered o/p	3) byte array o/p	3. char array reader	10. o/p stream writer
4) file i/p		4. char array writer	11. Line number Reader
5) file o/p		5. file reader	
6) push back streams		6. file writer	
		7. filter reader	

write a java program to enter your name from keyboard using streams.

```
import java.lang.*;
import java.io.*;
class Streamtest
{
Public static void main(String [] args) {
try{
Data Input stream d = new Data Input Stream (System.in);
String Sname;
System.out.println("enter the name");
Sname = d.read line();
System.out.println("The name is" + Sname);
}
Catch (exception e) {
System.out.println("error");
}
}
}
```

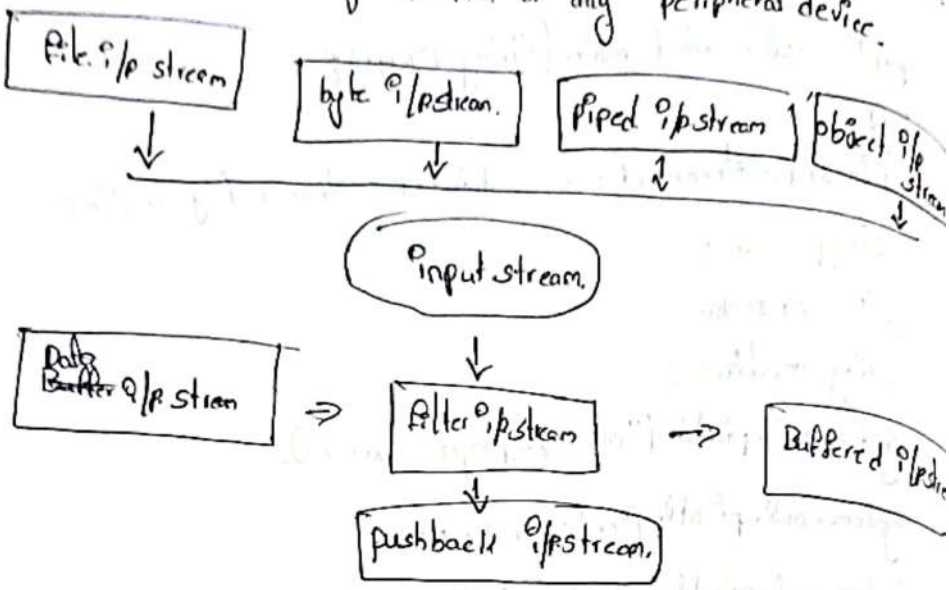
write a java program to Create employee details using Streams.

```
import java.lang.*;
import java.io.*;
class lokesh {
public static void main (String [] args) {
try {
Data.InputStream d = new dataInput.stream (System.in);
String ename;
Int EIDNO;
String eaddress;
System.out.println ("enter employee name");
System.out.println ("enter EIDNO:");
System.out.println ("enter eaddress:");
        Ename = d.readline ();
        EIDNo : d.Readline ();
        Eaddres = d.readline ();
System.out.println ("The employee name is :"+ename);
System.out.println ("The employee ID is :"+EID);
System.out.println ("The employee address is :"+Eaddres);
}
Catch (Exception e)
{
System.out.println ("error");
  }
 }
}
```

.T

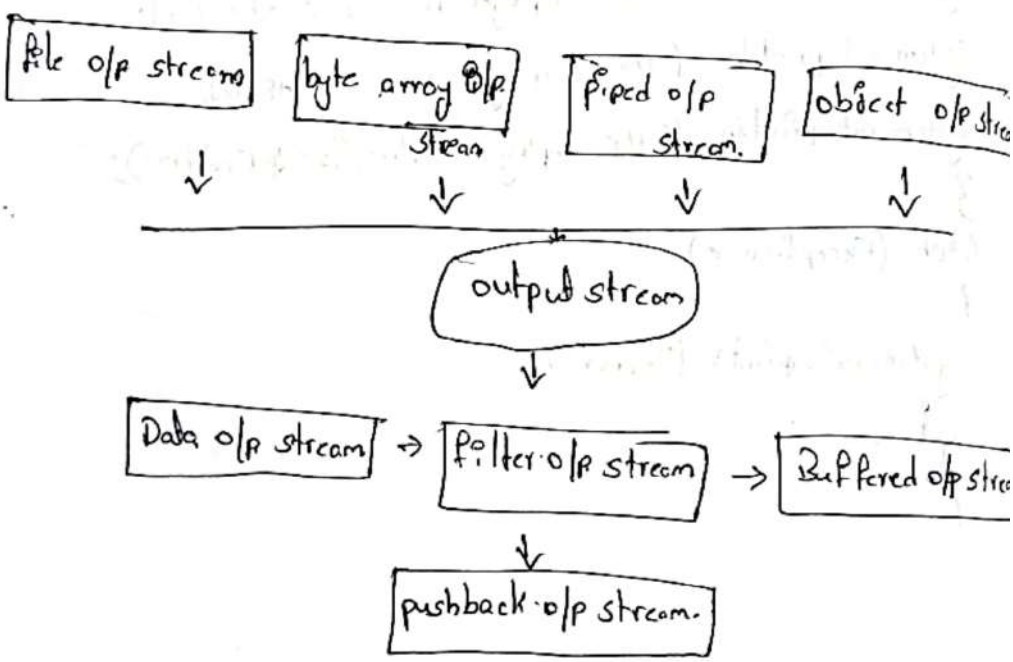
I/P stream :-

These streams are used to read data that must be taken as input from a source array of file or any peripheral device.



O/P streams :-

These streams are used to write data as o/p into an array or file or any i/p peripheral device.



A.P implements buffer reader.

Program:

```
import java.io.*;
class Nameteste {
public static void main (String [] args) {
try {
BufferedReader reader = new BufferedReader (new InputStreamReader (System.in));
System.out.println ("enter name:");
String name = reader.readline ();
System.out.println ("Hi!!" + Name);
} Catch (Exception e)
{
System.out.println ("error");
}
}
```

o/p:-

```
Enter name:
Alice
Hi Alice
```

File:-

- File is a collection of logical attributes and records.
- File is a collection of interrelated collection of information.
- Record is a collection of attributes.
- Record r1 = A1+A2+A3+A4
- Attribute is a property of an entity.
- Entity is anything in real world.
- File (String path, String filename)

eg:- File f = new file ("c:\", "filename.java")

File (String path);

eg:- File f1 = new file.

File methods:-

getName():-

This method is used to retriving of any filename from the directory

getLength():

It is used to finding the length of a file.

exists():

It is used to finding of any file available or not available under the directory.

lastModified():

It is used to finding of the any file is modified or not modified specifies a size of the file.

~~System for.~~ ~~exception~~ of multiple Catch blocks.

```
try
  {
    Loop 1/Statement 1
//First Catch block} catch (Exception e)
  {
  System.out.println ("error");
  }
    Loop 2/Statement 2.
//Second Catch block } catch (Exception e)
  {
   System.out.println ("error 1");
  }
```

-> try block supported more than one Catch blocks

-> try block is always maintain open flower bracket & close flower bracket

-> ~~Catch~~ block is ~~always~~ also supported one open flower bracket & close flower bracket

* Example program for one try block & multiple Catch blocks.

```
import java.lang.*;
class classCatchblocktest
{
public static void main(String[]args){
int a[]=new int[3];
try{
for(int i=1;i<3;i++){
a[i]=i*i;
}
}catch(Exception e){
System.out.println("Arithmatic Exception");
}

for(int i=0;i<3;i++){
    a[i]=i/i;
  }catch(Exception e)
  {
   System.out.println("Array out of bounds exception");
  }
 }// end of main method.
}// end of main class
```

output:-

Array out of bounds Exception.

Finally block:-

→ Finally block executes after 1 try ↑ Catch block execution.

→ finally block is always requires atleast 1 try catch block.

→ Finally block is always specified at the end of the application.

W.A.J.P implements finally block.

```
import java.lang.*;
class finally blktest
{
public static void main(String[]args){
```

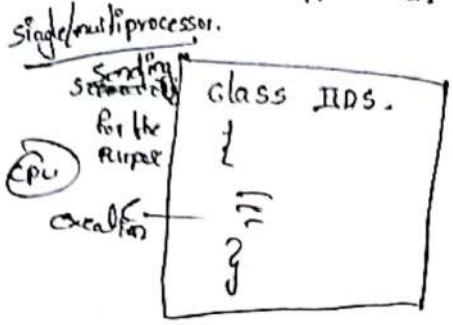
```
int a=4;
int b=-1;
try {
  c=a/b;
} Catch (Arithmatic Exception e)
{
System.out.println("Divide by zero");
}
finally {
if (b!=-1)
{
System.out.println ("with error");
else
Systemout.println ("without error");
}
}
```

UNIT-V Introduction of Multithreading

Process is a execution of entire application for the single processor or multiprocessor

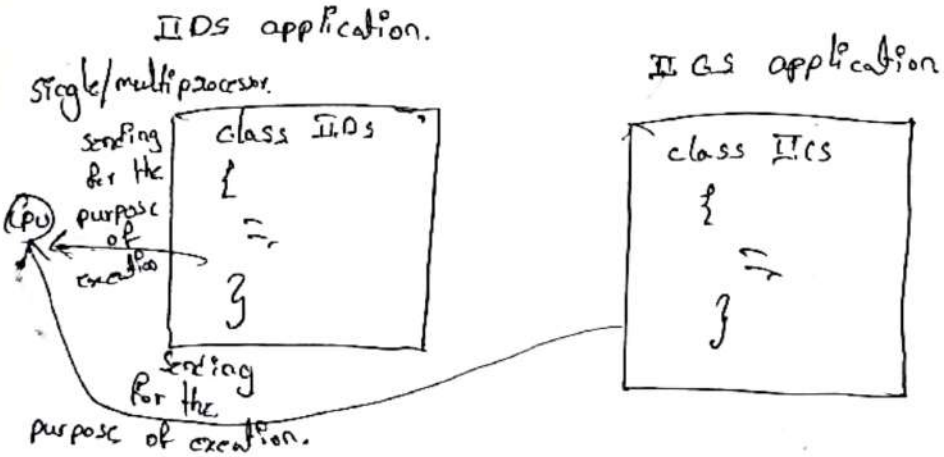
Consider the IIDS application.

Ex:- IIDS application



Multiprocess any user executes more than one process successfully under the single/multiprocessor is called multiprocess

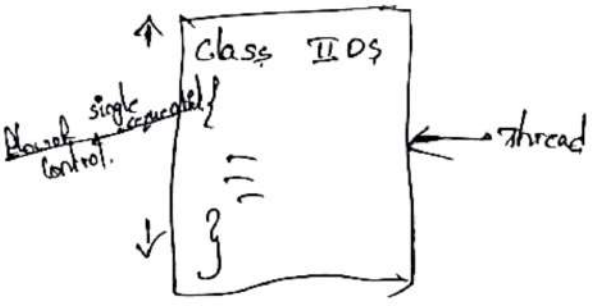
Ex:- Consider a IIDS and IICS application.



Thread:-

thread is single sequential flow of control within a application

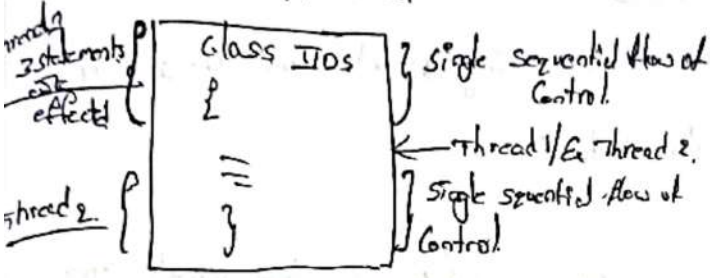
ex:- IIDS application.



Multithreading:-

Any user application using more than one single sequential flow of control is technically as multithreading (or) any user using more than one single sequential flow of control within application is technically called Multithreading.

Ex:- Consider a IIOs application.

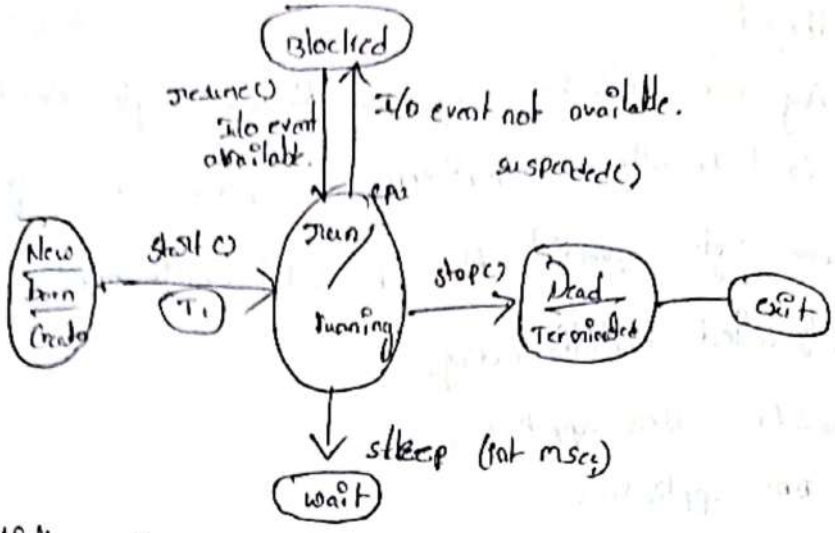


- Threading Concept throughly as business operating system
- OS threading is maintain some drawback. (any thread is not properly executed at that time restart the user application (or) reboot the user application)
- To overcome above drawback using Java multithreading Concepts to
- According to multithreading threads are created using 2 criteria's.
  1. Extends thread. → Inheritance
  2. Implements Runable. → Interface.
- All threads are created using objects in Java.
- Objects are Created using new operator.

Ex:-

```
Thread T1 = new Thread();
Thread T2 = new Thread();
```

- Thread (Classes / Concepts) are available in language packages
-



- Initially created no. of threads using private area. $T_i = 1 \ldots n$

here, I: no. of required thread iterations

Step 1:-

Create/born

- this state accepted no. of initialized threads from private area
- No. of threads information needed to send into 2nd state (Run/Running) using start() method.

ex:-

```
Thread T1
        =new Thread()
    T1.start();
```

- Run/Running state accepted different threads from the create state.
- Run/Running state act as a processor
- Different threads executed successfully under the Run/Running state with the help of using run method.

ex:-

```
public void run()
{
=
}
```

- Every thread implementation is required to write under the run method only.
- Successfully executed threads needed to send from run/Running state to terminated state (or) dead state.

• In case of any threads are not properly executed under the Run/Runing state those threads needed to send from Run/Running state to blocked state

ex:- any threads are not properly received i/o event from I/o devices to Running state (or) I/o devices to processor.

• Above threads are sending from Run/Running state to blocked state. using suspend().

Blocked state:-

• Blocked state accepted different threads from the Run/Running state.

• In case of any I/o event is available at the time blocked state sending different threads to the Run/Running state using resume method.

• Timed wait state:- accepted differents threads from the Run/Running state.

• In case of any user wants to seeing of particular thread information & Console at the time different threads are sending from Run/Running state wait state.

• Timed wait state supported sleep method

ex:-

```
Thread T1 = new thread().
        T1.sleep(500);
```

• Timed wait method is also supported wait().

• ~~Timed~~ ex:-

```
Thread T1 = new thread()
        T1.wait();
```

Terminated states:-

• Terminated state accepted ~~diffe~~ successfully executed threads from the Run/Running state. with the help of using stop method.

ex:-

```
Thread T1 = new thread()
        T1.stop().
```

• To remove any threads from the memory is possible with the help of using kill() method.

Write a J.A.P. to create 3 threads using multithreading concepts.

```
import java.lang.*; // creation of 1st thread
class thread1 extends Thread
{
public void run()
{
for(int i=1; i<3; i++){
System.out.println("thread1");
}
}
class Thread2 to extends Thread1
{
public void run(){
for(int i=1; i<5; i++){
System.out.println("thread2");
}}
class Thread3 extends thread2.
{
public void run(){
for(int i=1; i<7; i++){
System.out.println("Thread 3");
}
} class 3threads test {
public static void main(String []args){

    Thread T1 = new thread();
    Thread T2 = new thread();
    Thread T3 = new Thread();

        T1.display();
        T2.display();
        T3.display();
    }
    T1.start();
    T2.start();
    T3.start();
}
}
}
```

write a java program to create 3 threads using Runnable interface

```
import java.lang.*;
class threeThreadsTest implements Runnable
{
Thread T1 = new Thread(this);
Thread T2 = new Thread(this);
Thread T3 = new Thread(this);
   T1.start();
   T2.start();
   T3.start();
public void run()
{
If (Thread.currentThread() == T1)
{
for(int i=1; i<3; i++) {
System.out.println("Hi");
}
}
if (Thread.current Thread() == T2)
{
for(int i=1; i<5; i++){
System.out.println("welcome");
}
}
if (Thread.currentThread() == T3)
{
for(int i=1; i<=3; i++){
System.out.println("Java lab");
}
}
} //end of run method.
public static void main(String []args){
threeThreadsTest TT = new threeThreadsTest();
}//enclose main method.
}//enclose main class
```

```
    T3.sleep(500);
  } catch (exception e)
  {
  System.out.println("error");
  }
  public void run() {
    if (Thread.currentThread() == T1) {
    for (int i=1; i<9; i++) {
  System.out.println("good morning");
  }
  }
    if (Thread.currentThread() == T2) {
  for (int i=1; i<5; i++) {
  System.out.println("good afternoon");
  }
  }
  If (Thread.currentThread() == T3) {
  for (int i=1; i<4; i++) {
  System.out.println("good evening");
  }
  }
  public static void main (String[] args) {
    priority Test PT = new priority Test();
  }}
```

Thread priority Synchronization

Thread priority:-

Thread priority determines the order in which threads are exec. by the JVM. Java threads have a priority range of 1 to 10.

Thread priority levels

1. MIN_PRIORITY [1]: lowest priority.
2. NORM_PRIORITY [5]: Normal priority [default]
3. MAX_PRIORITY [10]: Highest priority.

Thread synchronization:-

Synchronization that ensures only one thread can execute a blo of code at a time.

Synchronization Methods:

1. Synchronized keyword: Locks a block of code or method.
2. Lock Interface:- provides more flexibility than synchronized key
3. Atomic variables: updates variables Automatically.

Deadlock & race situation:-

A deadlock occurs when two or more threads are blocked indefinitely, waiting for each other to release resources.

Deadlock Conditions:-

1. Mutual exclusion:- Two or more threads require exclusive access to a common resource.
2. Hold & wait:- one threads holds a resource and waits and ano resource held by another thread.
3. No preemption:- The OS does not prompt one thread to give resources to another.
4. Circular wait:- threads from a circular chain, each waiting for resources held by the next thread.

Q:-

```
public class Deadlock {
    public static void main (String args){
        object lock1 = new object();
        object lock2 = new object();
    Thread thread1 = new Thread (()->{
        Synchronized (lock1){Synchronized (lock2){}}
    });
    Thread thread2 = new thread (()->{
        Synchronized (lock2){synchronized (lock1){}}
    });
    thread1.start(); thread2.start();
    }
}
```

Expected o/p:-

program hangs with no. o/p.

Race situation:-

A Race situation occurs when multiple threads access & shared resources simultaneously leading to inconsistent results.

```
class Counter {
    static Count = 0;
public static void main (String [] args) throws InterruptedException {
    Thread thread1 = new thread (()-> Count++);
    Thread thread2 = new Thread (()-> Count++);
    thread1.start(); thread2.start();
    thread1.join(); thread2.join();
    System.out.println (Count); }}
```

2

Inter thread Communication - Suspending:-

Inter-thread Communication in Java allows threads to coordinate the exchange information. Suspending a thread temporarily pauses its execution.

Methods for Inter-thread communication.

1. wait():- Causes the current thread to wait until another thread calls notify() or notifyall().
2. notify():- wakes up a single thread waiting in the object's monitor.
3. notifyall():- wakes up all threads waiting on the object's monitor.
4. suspend():- Thread temporarily pauses a thread execution (deprecated).
5. Resume():- Resumes a suspend thread (deprecated)

Resuming & Stopping threads:-

1. Resume() method (deprecated):- Restores a suspend thread.
2. notify() or notifyall() methods:- wake up a waiting thread.

Stopping threads:-

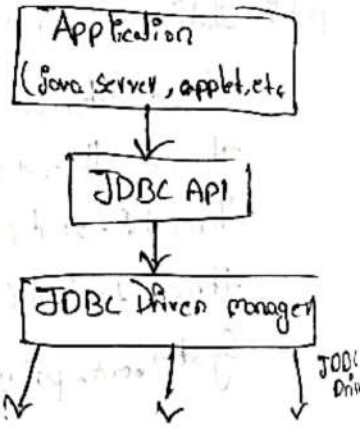
1. stop() method (deprecated):- immediately terminates the thread
2. interrupt() method:- Requests the thread to stop.

Java Database Connectivity (JDBC):-

JDBC (Java Database Connectivity) is a Java API for connecting to relational db, executing SQL statements & retrieving results.

Key features:-

1. platform independence
2. DB "
3. SQL support
4. Transaction management.
5. Connection pooling.



JDBC Architecture:-

1. JDBC Driver:- Translates JDBC calls into db-specific calls.
2. JDBC driver manager:- Manages multiple JDBC drivers.
3. Application:- uses JDBC API to

Q:-
Use assume a table as my-table for creating a JDBC program.
write a program for creating JDBC program?

ID	Name
1	Ramu
2	Sita
3	Ali

```
import java.sql.*;
public class SimpleJDBC {
    public static void main(String[] args){
try {Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test",
"user", "pass");
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT FROM
my-table"))
{
while (rs.next()) System.out.println(rs.getString(1));
}catch (SQLException e){ e.printStackTrace();}
  }
}.
```

Expected o/p:-

1
2
3.

Installing MySQL:-

On windows:

Download MySQL installer:-

Visit the MySQL Community Downloads page & download the MySQL installer.

Run the Installer:-

Launch the installer & select the installation type. (Developer default is recommended)

Configurations:-

Follow the prompts to configure the server. Set a root PW & create any necessary user accounts.

Complete installation:

Finish the setup, you can start the MySQL server from the installer manually via the Services management tool.

On Linux (ubuntu / debian)

Update package index: sudo apt-get update.

Install MySQL server: sudo apt install mysql-server.

Secure installation:- sudo mysql_secure_installation.

Start MySQL: sudo systemctl start mysql.

Installing MySQL Connector/J:-

For both windows & linux:-

- Download Connector/J: Go to the MySQL Connector/J page & download the latest version (zip for windows TAR for Linux)
- Extract the archives:-

  Windows: use a tool like win RAR or 7-zip to extract.

  Linux: use the command.

  tar -xvf mysql-connector-java-x.x.x.tar.gz.

Event Handling:-

W.J.P to create a frame (or) window using abstract window toolkit.

```
import java.lang.*;
import java.awt.*;
class myframe
{
public static void main (String[] args){

  frame F= new frame ();
Set size F (200,200);
Set title ("venky");
Set variable (true);
}
```

w.J.P to create a window of frame using super window (or) super class

```
import java.lang.*;
import java.awt.*;
class myframe2 extends Frame
{
public static void main(String[] args){
Set size (100,100);
Set Size;
Set title ("Venky");
Set visible (True);
  }
}
```

w.J.P to create two buttons & two text fields & using flow layout

```
import java.lang.*;
import java.awt.*;
class Button Text field extends frame
{
    Button B1, B2;
    Text field TF1, TF2;
    Flow Layout FL;

BTF()
{
    B1 = new Button ("Set 1");
    B2 = new Button ("Set 2");

    TF1 = new Text field();
    TF2 = new Text field();
    FL = new Flow layout();
  add (B1);
  add (B2);
  add (TF1);
  add (TF2);
  add (FL);
  Set layout (FL);

 Set Size (200,200);
 Set title ("FRAME");
 Set visible (true);
}
```

public static void main(String[] args) {

BTF btf = new BTF();

}

}