OOPS

OOPS: These concepts aim to implement real-world entities in programs.

Inheritance:

→ A class is created from another class is called Inheritance

→ The class which was created is called a child class

→ The child class acquires all the properties of the parent class.

→ Inheritance is declared using the "extends" keyword.

Types:

Single Inheritance:

→ A class is created from only one parent class is called single Inheritance.

multiple Inheritance:

→ A class is created from two or more parent classes.

→ Due to ambiguity problem java does not support multiple inheritance

→ To achieve this multiple Inheritance we can use Interfaces.

Hirarical Inheritance:

→ Two or more classes are created from only one parent class is called hirarical Inheritance.

Keywords:

this:

→ "this" is a reference variable that refers to the current object.

→ using "this" we can invoke current class method. (this.methodname();)

→ using "this" we can invoke current class constructor (this(parameters))

→ If no parameters are there in the constructor what we are trying to pass parameters then use "this();".

Super:

→ "super" is a reference variable which is used to refer immediate parent class object.

Example:

```
class A
{
 void run()
  {
   s.o.p.ln("Hello");
}}
class B extends A
{
 void display()
  {
  super.run();
  s.o.p.ln("elisha");
}}
```

```
class C
{
 p.s.v.m(String[] args)
 {
  B b = new B();
  b.display();
 }
}
```

Outputs:

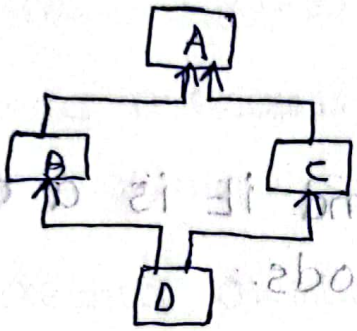
Hello
elisha

multilevel Inheritance:

→ A class is created from one class which is already derived from another class.

hybrid Inheritance:

→ Hybrid Inheritance is a combination of more than one types of Inheritance. (Simple, multiple, hierarchical)



→ Any two Inheritance exist then it is a hybrid Inheritance.

Real time Example:

→ The real time example of inheritance is child and parents.

→ All the properties of either father or mother are inherited by his son.

Advantages:

→ Code reusability is the advantage of inheritance.

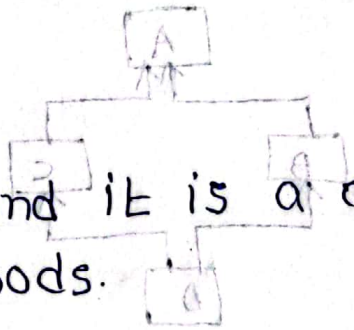
→ Due to this redundancy of the application is also reduced

this & super difference:

→ "this" keyword can call current class as well as parent class method also.

→ But child class and parent class methods should be different.

→ When a parent class and child class have same named methods then to resolve ambiguity we use "super" keyword.



Class:

→ class is a keyword and it is a collection of objects and its methods.

→ class is ~~nothing but~~ where we can model an object.

Object:

→ object is a real time entity or it is an instance of class.

Some points:

→ javac converts source code into java byte code.

→ "Object" class is the super class of all the classes.

Polymorphism:

→ Polymorphism is solving problems in different ways (or) many ways.

→ There are two types:

(i) Static Polymorphism:

→ This is also called compile time polymorphism

→ The methods to be called at compile time so this is also called compile time polymorphism.

→ method overloading is the example of static polymorphism.

→ method overloading is retaining the multiple methods with the same name but with the different parameters.

Example:

```
class A
{
void square(int a)
{
 S.O.println(a*a);
}
void square(int a,int b)
{
 S.O.println(a+b);
}
 P.S.V.m(String[] args)
 {
  A Pace = new A();
  Pace.square(5);
} } Pace.square(5,6);
```

(ii) Dynamic polymorphism:

→ This is also called run time polymorphism.

→ The methods to be called at run time so this is also called dynamic polymorphism.

→ Method overriding is the example of dynamic polymorphism

→ Method overriding means no of methods are declared in the no of classes with the same name but using one class object can access all the classes methods.

Example:

```
class fruits
{
  void msg()
  {
    S.o.pln("fruits");
  }
}
class mango extends fruits
{
  void msg()
  {
    S.o.pln("mangoes");
  }
}
class Bananas extends fruits
{
  void msg()
  {
```

```
        S.o.pln("Bananas");
    }
}
public class Poly
{
    p.s.v.m(String[] args)
    {
        Fruits f = new Fruits();
        f.msg();
        Mango m = new mango();
        f = m;
        f.msg();
        Banana b = new Banana();
        f = b;
        f.msg();
    }
}
```

Polymorphism real time example:

→ A person at the same time can have different characteristic.

→ like a man at the same time is a father, a husband, an employee.

Advantages:

→ code reusability

→ single variable can be used to store multiple data types.

Encapsulation:

→ Wrapping or binding or combining of data variables with the data methods in a single unit

→ There are two points in the encapsulation
(i) Declare the data variables as private
(ii) If any class access that private variables using set and get methods.

→ Data hiding is ~~nothing but~~ means when the variables are declared as private we cannot access it from outside the class.

→ Encapsulation can be achieved by declaring all the variables in the class as private and write public methods in the class to set and get the values of private variables.

→ Encapsulation = data hiding + data abstraction

Example:

```
class student
{
 private int RollNo;
 private String Name;
 private String age;

 Public Int getName()
 {
  return RollNo;
 }
```

```
public void SetRollNo(int RollNo)
{
this.RollNO = RollNO;
}
class hello
{
P.S.V.m(String[] args)
{
Student ed = new Student();
S.o.println(ed.getName());
ed.SetRollNo(123);
System.out.println(ed.getName());
}
}
```

Output:

NULL
123

Advantages:

→ The encapsulated code is more flexible and easy to change with new requirements.

→ It prevents the other classes to access the private variables.

Real time Example:

→ Every java class is an example of encapsulation.

→ The "Java Bean" class is the example of a fully encapsulated class.

Abstraction:

→ The process of hiding the implementation details to the user but provides the functionality to the user.

→ A class declared with "abstract" keyword is called abstract class

→ Abstract class is a class which with zero or more abstract methods.

→ Abstract classes may or may not contain abstract methods.

→ A method declared with "abstract" keyword is called abstract method.

→ When a method is declared as abstract method then that class must be an abstract class.

→ Abstract class can't be instantiated means we can't create object for abstract classes.

→ In order to convert the abstract class into concrete class we must and should override all the methods in the parent class using child class methods.

→ The process of hiding the details and highlighting the set of services that provided is also called abstraction.

→ We can achieve abstraction using abstract classes and interfaces.

Example:

```
abstract class Vehicle
{
abstract void findrate(int a);
abstract void findbrand(String b);
}
class car extends Vehicle
{
public void findBrand(String a)
{
S.o.pln(a);
}
public void findRate(int b)
{
S.o.pln(b);
}
public class demo
{
P.S.V.m(String[] args)
{
Car C = new Car();
C.findBrand("elisha");
C.findRate(1000000);
}
}
```

Output:

elisha

Advantages

-> security
-> easily Enhancable

Constructor:

→ Constructor is a special method but the class name is equals to the constructor name.

→ The constructor is used to allocate the memory locations to an object.

Default Constructor:

→ Default constructor is created by the java compiler when an object was created in the main method then java compiler create a constructor for allocating the memory locations to the object.

Example:

```
class hello
{
public void show()
 {
  S.o.pln("Hello");
 }
}

p.s.v.m(String[] args)
{
hello h=new hello();
h.show();
}
}
```

Output:

Hello

No argument constructor:

→ A constructor is declared without argument is known as No argument constructor.

Example:

```
class pace
{
 pace()
 {
  S.O.Pln("Hello");
 }

class college

P.S. V.m(String[] args)
{
pace p = new pace();
}
}
```

Output:

Hello

Parameterised Constructor:

→ A constructor is declared with arguments is called as parameterised constructor.

Example:

```
class loading
{
loading(int a, String b)
{
S.o.Pln(a+" "+b);
}
P.s.v.m(String[] args)
{
loading l = new loading(42, "Elisha");
}
}
```

Output:

42 Elisha

Constructor Overloading:

→ No of constructors are declared with the different arguments is known as constructor overloading.

Example:

```
class Student
{
Student()
{
S.O.println("Nothing");
}
Student(int a,int b)
{
S.O.println(a*b);
}

Student(String name)
{
S.O.println(name);
}
P.S.V.m(String[] args)
{
Student S1=new Student();
Student S2=new Student(10,20);
Student S3=new Student("Hello");
}
}
```

Output: Nothing
200
Hello

Buffered Reader

—> It reads input from either keyboard or file.
—> Input Stream reader read the bytes and be converted into the character set.
—> File reader reads data from files
—> All the bytes which are taken from "System.in" or "FileReader" will be buffered.
—> So in order to read the data from this buffer we will use "BufferedReader".
—> Buffered Reader only reads characters or strings.
—> In order to read another we need to typecast
—> read() reads method reads single character
—> readLine() method reads multiple characters or string

Example

```
import java.io.*;
class loading
{
P.S.V.m(String[] args) throws IOException
{
FileReader fr=new FileReader("Location");
BufferedReader br=new BufferedReader(fr);
```

```
String str = br.readLine();
S.O.println(str);
InputStreamReader ir=new InputStreamReader
                        (System.in);
}
}
```

Example :

```
import java.io.*;
class loading
{
p.s.v.m(String[] args) throws Exception
{
InputStreamReader ir=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(ir);
String str=br.readLine();
S.O.println(str);
}
}
```

Combining both (FileReader and InputStreamReader)

```
BufferedReader br= new BufferedReader(fr);
BufferedReader kr= new BufferedReader(ir);
```

→ By creating the different objects for the BufferedReader class we can perform both FileReader and InputStream reader.

Console:

→ The console is a class it is used to read the user input data.

→ This class lies in the package .util.

```
import java.util.console;
```

Example:

```
import java.util.*;
class hello
{
P.S.V.m(String[] args)
{
Console c = System.console();
```

↓ 'c' is capital

↓ 'c' is small

```
S.o.pln("enter any string");

String s = c.readLine();

}
}
```

Output:

```
enter any string
Hello Hill
Output:
Hello Hii.
```

Threads:

→ Whenever we are performing a thread operations we should extends the "Thread" class. (using inheritance but not interface)

→ we must use "start()" method to start the process of a thread.

→ Whenever we call "start()' method it implicitly calls the "run()" method which is already present in "Thread" class.

→ So we are overriding run() method in the "Thread" class..

Example: (using inheritance)

```
class hello extends Thread
{
public void run()
{
 S.o.Pln("Hello");
}
}
class B
{
 P.S.V.m(String[] args)
 {
  hello h=new hello();
  h.start();
}}
```

output:

Hello.

* Example using Interface:

```
class Single implements Runnable
{
public void run()
{
  S.o.pln("Hello");
}
}
class hello
{
P.s.v.m(String[] args)
{
  Single S = new Single();
    Thread t = new Thread(S);
    t.start();
}
}
```

Output:

Hello

methods of Thread class

start() -* The thread execution starts when we called this method.

run() -* Implicity called by start().

sleep(milliseconds) -* suspends the thread for given milliseconds of time

* Sleep may throw an exception so we should write this is in "try and catch block".
* It is a static method so we can call directly using "Thread.sleep(1000)"

join() -* waits the thread to complete its process.

* used in multithreading
* It is written in "try and catch" block.
* It is a static method means we cannot create object for this we directly call this using class called "Thread"..

getId() -* Gives the Id of a thread.

getName() - * Gives the thread name
* Always start from thread-0

setName(String) - * Thread name will be replaced with given string.

getPriority() - * Priority ranges from 1 to 10
* MIN-PRIORITY - 1
* NORM-PRIORITY - 5 → default
* MAX-PRIORITY - 10

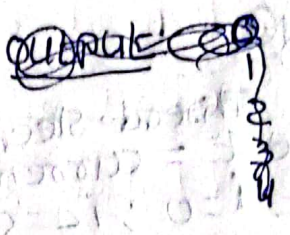
setPriority(Integer) - * setPriority(10)
↓ 1 to 10 ↓ Here the priority of thread is 10.

isAlive() - * Returns true if thread is still running
* Returns false if thread complete its execution

Example:

```
class Th extends Thread
{
  public void run()
  {
    S.O.Pln("Hello");
  }
}
class T
{
  try
  {
    Thread.sleep(1000);
  }
  catch(Exception e)
  {
  }
  Thread t = currentThread();
  for(int i=0 ; i<=5 ; i++)
  {
    try
    {
      Thread.sleep(1000); (or) t.sleep(1000);
    }
    catch(Exception e)
    {
    }
    S.O.Pln(i);
  }
}
class TMethods
{
  P.S.V.m(String[] args)
  {
    Th t1 = new Th();
```

```
S.o.pln(t1.getID());
S.o.pln(t1.getName());
t1.setName("elisha");
S.o.pln(t1.getPriority());
t1.setPriority(1);
t1.start();
}
}
```



→ To use join we need multithreading so create one more thread to write this method.

```
class Th extends Thread
{
public void run()
{
Thread t = currentThread();
S.o.pln(isAlive());
for(int i=0; i<=5; i++)
{
try
{
t.sleep(1000);
}
```

```
        catch(Exception e)
        {
        }
        System.out.println(i);
    }
}

class TMethods
{
    p s v m(String[] args)
    {
        Th t1 = new Th();
        Th t2 = new Th();
        t1.start();
        try
        {
            t1.join();
        }
        catch(Exception e)
        {
        }

        t2.start();
    }
}
```

output:

```
0
1
2
3
4
5
0
1
2
3
4
```

→ without using "join" the output will be like below.

```
0 0
1 1
2 2
3 3
4 4
5 5
```

Thread synchronization:

→ When more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time this is done by thread synchronization.

→ It is achieved using 3 ways

(i) Synchronized keyword for method

(ii) Synchronized block inside the method.

(iii) Synchronized static block applied for method.

→ we cannot write synchronized inside the run() method

Example with Synchronized keyword:

```
import java.io.*;
class Example
{
synchronized void display()
{
 Thread g = Thread.currentThread();
 for(int i=0;i<=5;i++)
 {
  try
  {
   Thread.sleep(1000);
   S.o.pln(g.getName()+" "+i);
  }
  catch(Exception e)
  {
  }
```

```
    }
  }
}
class T extends Thread
{
  Example e;
  T(Example e)
  {
    this.e = e;
  }
  public void run()
  {
    e.display();
  }
}
class TSynch
{
  P.S.V.M(String[] args)
  {
    Example ex = new Example();
    T t1 = new T(ex);
    T t2 = new T(ex);
    T t3 = new T(ex);

    t1.start();
    t2.start();
    t3.start();
  }
}
```

Example with synchronized block inside the method

→ same above program but slight modifications in example class.

```
import java.io.*;
class Example
{
void display()
{
 Thread t = Thread.currentThread();
 synchronized(this)
 {
 for(int i=0;i<=5;i++)
 {
 try
 {
  Thread.sleep(1000);
  S.o.pln(t.getName()+" "+i);
 }
 catch(Exception e)
 {
  }
 }
 }
}
}
```

Example with static block applied for method:

→ We need to modify ~~the~~ entire above program.

```
import java.io.*;
class Example
{
 synchronized static void display()
 {
  Thread g = Thread.currentThread();
  for(int i=0; i<=5; i++)
  {
   try
   {
    Thread.sleep(1000);
    S.O.Pln(g.getName()+" "+i);
   }
   catch(Exception e)
   {
   }
  }
 }
}
class T extends Thread
{
 Public void run()
 {
   Example.display();
```

```
    }
}
class TSynch
{
 p.s.v.m(String a[] args)
 {
    T t1 = new T();
    T t2 = new T();
    T t3 = new T();

    t1.start();
    t2.start();
    t3.start();
  }
}
```

→ Here we are using static so need to create object.

Output:

```
Thread -1  0
    :
Thread -2  0
    :
    :
Thread -0  0
    :
    :
    }
```

Interfaces

Multithreading:

→ The process of executing multiple threads simultaneously.

→ A thread is a lightweight process

Life cycle of Thread:

(i) New:

→ The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

→ In this state the thread is newly created.

(ii) Runnable:

→ The thread is in runnable state after invocation of start() method.

→ In this state the thread is ready to run or ready to execute.

(iii) Running State:

→ In this the thread is in running or executing.

(iv) Block State:

→ In this state the thread is invoked by sleep (or) wait methods.

→ Here thread is still live but is currently not eligible.

(v) Terminated State (or) dead state:

→ In this state the destroy method is invoked by the thread.

→ A thread is in terminated state when its run() method exits.

Interface:

→ Interface contains abstract methods and static variables or final variables.

→ multiple inheritance is achieved using interface

Example:

```
Interface A
{
   void display();
}
interface B
{
void show();
}
class AB implements A,B
{
public
  void display()
  {
    s.o.pln("A");
  }
public
  void show()
  {
    s.o.pln("B");
  }
}
```

```
class interface AB
{
  p.s.v.m(String[] args)
  {
    AB obi = new A();
    obi.display();
    obi.showC();
  }
}
```

output;

A

B

Private variable:

→ It cannot be accessed from outside of the class.

→ In order to give input to private variable from outside of the class below is the example.

→ It is also an example for encapsulation.

Example:

```
import java.io.*;
import java.util.Scanner;
class demo
{
private String name;
public void action(String movie)
{
name = movie;
}
public void func()
{
S.O.PLn(name);
}
}
class checkprivate
{
P.S.V.m(String[] args)
{
Scanner s = new Scanner(System.in);
String a = s.nextLine();
hello h = new hello();
h.action(a);
h.func();
}
}
```

output: Elisha
Elisha

import java.lang.Math Package;

→ It consists of several functions

Math.max(a,b);
Math.min(a,b);

Math.sqrt(a);

Math.pow(a,2);

Math.abs(-25.8);

Math.ceil(5.2)

Math.floor(5.2)

Math.round(a)

Math.random() → (0 to 1 values in float)

Math.exp(a)

Math.log(a)

Math.cbrt(a)

Math.signum(a)

Math.sin(a)

Math.cos(a)

→ Trignometric values

Math.tan(a) → trignometric values

Math.asin(a)
Math.acos(a)
Math.atan(a)
→ trignometric arc values

Math.sinh(a)
Math.cosh(a)
Math.cosh(a)
→ hyperbolic values

Math.log() → returns natural logarithm of a double value.

Math.log10() → returns the base 10 logarithm of a double value.

Math.toDegrees() → Radians to degree

Math.toRadians() → Degree to radians

Exception:

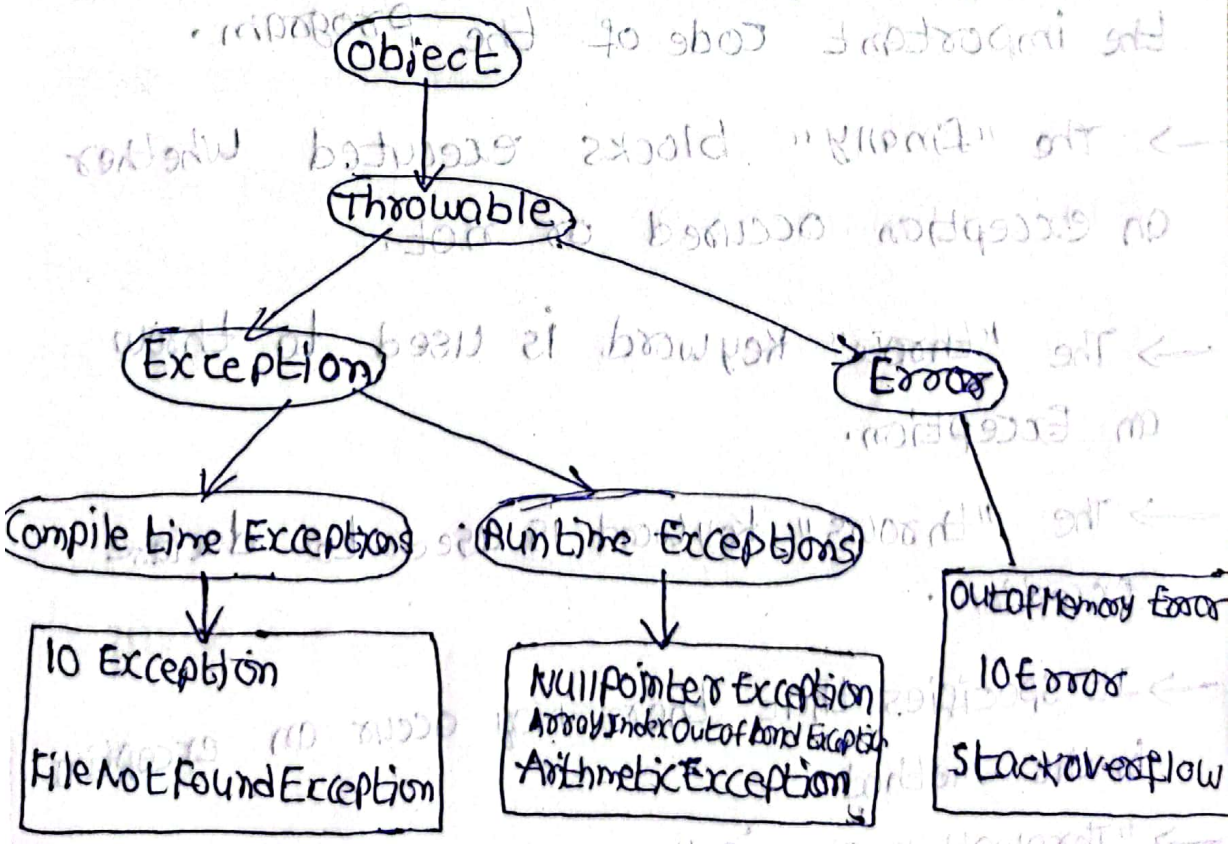
→ Exception is an abnormal condition.

→ When the exception occurs in a method, the process of creating the exception object and handling it over to runtime environment (JVM) is called "Throwing the exception".

→ When exception object received by JVM then it search for the block of code that can process the exception object (Exception handler)

→ Here the Exception handler is said to be "Catching the exception".

→ Java exception handling is a framework that is used to handle runtime errors only, we cannot handle compile time errors using Exception handling



→ Exceptions occured at compiled time are called Checked exceptions.

→ Exceptions occured at run time are called Unchecked exceptions.

→ "Run time Exceptions" or "unchecked Exceptions" are controlled JVM.

Exception handling keywords:

→ The "try" keyword is used to specify a block where we should place exception code.

→ The "try" block must be followed by either catch or finally.

→ The "catch" block is used to handle the exception.

→ The "finally" block is used to execute the important code of the program.

→ The "finally" blocks executed whether an exception occured or not.

→ The "throw" keyword is used to throw an Exception.

→ The "throws" keyword is used to declare exception.

→ It specifies that there may occur an exception in the method.

→ "Throwable" class is the super class of all the exceptions and errors.

→
```
try
{
 a = 5
 b = 0
 c = a/b;
}
catch(ArithmeticException e)
{
  e.printStackTrace(); (or) e.toString());
}
```

→ "printStackTrace()" method is used to give full details where exception occurs.

→ "toString" method only gives one Exception detail.

→ "getMessage()" is used to give only why Exception occurs.

→ "toString()" and "getMessage()" should return in s.o.println().

→ We can create multiple catch blocks for a single try block.

→ When method is declared with "static" so there is no need to call this method using object.

→ * The throw keyword is mainly used to arise custom exceptions.

Throw keyword Example:

```
Class A
{
static void validate(int age)
{
if(age<18)
{
throw new ArithmeticException("Not Eligible");
}
else
{
S.O.Pln("eligible");
}
}

P.S.V.m(String[] args)
{
validate(13);
S.O.Pln("program");
}
}
```

Output:

eligible

Throws keyword Example:

```
class A
{
 static void display() throws ArithmeticException
 {
 int a=3/0;
 S.O.pln(a);
 }

 P. S. V. m(String[] args)

{
 try
 {
 display();
 }
 catch(ArithmeticException ae)

 {

 ae.printStackTrace();
 }

 S.o.pln("Success");

 }
}
```

→ we are given there may be a chance of ArithmeticException.

→ In real time we are using "throws" keyword only for checked Exceptions.
(IO, FileNotFound, ClassNotFound, SQL etc exceptions)

→ We cannot throw multiple exceptions using "throw" keyword.

→ We can throw multiple exceptions using "throws" keyword.

User defined Exception:

→ Exception created by user is called user defined Exception.

Example:

```
class ExceptionDemo extends Exception
{
public Exceptiondemo(String str)
{
 super(str);
}
p.s.v.m(String[] args) throws ExceptionDemo
{
 try
 {
  java.util.Scanner sc = new java.util.Scanner
                          (System.in);
int age = sc.nextInt();
```

```
if (age<18)
{
throw new ExceptionDemo(" Not eligible");
}
else
{
 S.O.Pln("eligible");
}
}
catch(ExceptionDemo ed)
{
 ed.printStackTrace();
}
}
}
```

Graphical User Interface (GUI):

→ GUI is made up of graphical components (buttons, labels, windows) through which the user can interact with the page or application.

→ "import.javax.swing.JOptionPane" package has facilities for creating a dialog box that can appear on the computer's desktop to request input from or display messages to the user

→ "import java.awt.FlowLayout;" package is used to create own window layout.

→ "import javax.swing.JFrame;" package is used for title bar, minimize, maximize purpose.

→ "import javax.swing.JLabel;" to store text and simple images.

Dialog box Example:

```
import java.swing.JOptionpane;
class graph
{
P. S. V. m(String [] args)
{
String x,y;
x = JOptionPane.showInputDialog("enter any");
y = JOptionpane.showInputDialog("enter any");
```

```
int a = Integer.parseInt(x);
int b = Integer.parseInt(y);
int c = a+b;
JOptionPane.showMessageDialog(null, "Total is",
    "No title", JOptionPane.PLAIN_MESSAGE);
}
}
```

Window Layout Example:

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
class graph extends JFrame
{
private JLabel item1;
{
super("The title bar");
setLayout(new FlowLayout());
item1 = new JLabel("Hello");
item1.setToolTipText("Hii");
add(item1);
}
P.S.V.m(String[] args)
{
graph g = new graph();
g.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
g.setSize(250, 200);
g.setVisible(true);
}
}
```

Main method (public static void main(String[] args))

Public

→ To call by JVM from anywhere

→ When we install JVM in c drive or d drive so in order to call by the JVM the main method should be declared public.

Static

→ Without existing object also JVM has to call this method if main method no way related to any object.

~~→ Here JVM is a etc.~~

Void

→ main() method won't return anything to JVM.

main

→ Name of the method. ~~which is configured~~

String[] args:

→ It means an array of String type is used to store Java command line arguments.

→ Since java accepts only String type as a command line argument.

→ In order to change to other data types we have to parse it.

System.out.println();

System

→ System is a class present in java.lang package.

out

→ out is a static variable present in System class of type PrintStream.

println()

→ println() is a non-static method present in PrintStream class. for writing or printing data.

Example:

```
import java.io.PrintStream;
class System
{
public static final PrintStream out;
}
class PrintStream
{
public void println(java.lang.String);
}
```

Java Virtual machine (JVM):

→ JVM acts as a run-time engine to run java applications.

→ JVM is a part of java runtime environment.

→ JVM calls the main method present in the source code.

→ When we compile the java source file the java compiler converts the source code into byte code (or) ".java" file into ".class" file with the same name.

→ After this ".class" file is loaded into the JVM. JVM creates an object of type class to represent this file in the heap memory.

→ Finally JVM converts byte code into machine code.

→ ~~JVM two primary functions~~

→ JVM allows java programs to run on any device or operating system (write once run anywhere)

→ JVM is used to manage and optimize program memory.

→ JVM acts as a
- class loader
- various memory areas
- execution units

} → these are present in JVM architecture

Java data base connectivity (JDBC):

→ JDBC is an application programming interface (API) for the programming language Java. It provides methods to query and update data in a database.

Steps for executing JDBC program:

* ⇒ Load the JDBC Driver using forName() from class.

ex: Class.forName("Driver.Name");

Class.forName("oracle.jdbc.driver.OracleDriver");

* ⇒ Write a connection statement to connect with database using getConnection() method having 3 arguments
  - (i) Jdbc_url
  - (ii) Username
  - (iii) Password

ex:
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system", "Pace");

* ⇒ create statement class object inorder to execute queries in java Applications.
  - (i) Select queries
  - (ii) Non Select queries

→ for select queries use executeQuery() method

→ for non select queries use executeUpdate() method

Ex:

```
Statement stmt = con.createStatement();
int r = stmt.executeUpdate("Any DML Query");
```

(or)

```
ResultSet rs = stmt.executeQuery("Any select Query");
```

→ Statement object is used to execute queries in java Application.

* ⟹ close the connection using close() method.

ex: con.close();

Example program:

```
import java.io.*;
import java.sql.*;
class jdbcdemo
{
 P.S.V.m(String[] args)
 {
 try
 {
 class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection(
   "jdbc:oracle:thin:@localhost:XE","System","Pace");
Statement stmt = con.createStatement();
int r = stmt.executeUpdate("insert into student
        values(111,'Smith')");
System.out.println(r+" row is inserted");
con.close();
 }
}
}
```