DATE

file handeling in Java

Create a file

```
import java.io.file;
import java.io.IOException
Pubic class GfG{
    Pubic static void main(String[] args){
        try{
            file obj=new file("myfile.txt");
            if (obj.createNewfile()){
                System.Out.println("created:"+obj.getname
            else{
                System:Out.println("file already exists");
        Catch(IOException e){
            System:Out.println("An error has occured");
            e.printStackTrace();
        }
    }
}
```

Read from file:

```
file Obj=new file("hi.txt");
Scanner read=new Scanner(obj);
While(read.hasNextLine()){
    String data=read.nextLine();
    System.Out.println(data);
}
read.close();
```

PAGE

DATE

3) Write to file :-

```
file obj = new file("hello.txt");
FileWriter writer = new FileWriter(obj);
writer.write("Hi, I am Nisha");
writer.close();
```

4 WAP to read from a file abc.txt and write it to xyz.txt

```
Public class files {

Public static void main(string args[]){
  file abc = new file("abc.txt");
  FILEReader reader = new Filereader(abc);
  file xyz = new file("xyz.txt");
  int i;
  while((i = reader.read()) != -1) {
    String s += (char)i
  }
  FileWriter writer = new FileWriter(xyz)
  writer.write(s);
  reader.close();
  writer.close();
}
```


DATE

WAP to check Palindrome where client sends a message to the server and server responds with yes or no.

SERVER

```
Class server {
  Public Static void main (String [] args) {
    try {
      ServerSocket ss = new ServerSocket (5000);
      Socket s = ss.accept(); //accept client conn
      DataInputStream din = new DataInputStream(s.getInputStr
      DataOutputStream dout = new DataOutputStream(s.getOutputStrea
      String input = din.readUTF();
      dout.writeUTF(new String("\n"+checkPalindrome(inpu
      dout.close();
      ss.close();
    } Catch (IOException e) {
      System.out.println(e);
    }
  }

  Public Static Boolean checkPalindrome(String s){
    String check = "";
    for (int i = 0; i < s.length(); i++){
      check = s.charAt(i)+check;
    }
    if (check.equalsIgnoreCase(s){
      return true;
    } return false; }
```

```
import java.util.Scanner Client
class client {
public static void main (String [] args) {
try {
Socket S = new Socket ("localhost", 5000);
DataInputStream din = new DataInputStream (S.getInputStream());
DataOutputStream dout = new DataOutputStream
(S.getOutputStream());
Scanner S = new Scanner (System.in);
String input = "";
while (true) {
System.out.println ("Enter the string");
input = S.next();
if (input.equalsIgnoreCase ("exit")) {
break;
}
dout.writeUTF (input);
String output = din.readUTF();
System.out.println (output
dout.close();
din.close();
S.close();
} catch (IOException e) {
System.out.println (e);
}}}
```

Multithreading:

multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called thread. So threads are lightweight process within a process.

when we create an object of our new class and call start() method to start the execution of a thread, Start invokes the run() method on thread object.

```
class multithreadingdemo extends Thread {
public void run() {
try {
System.out.println ("Thread" + Thread.currentTh
().getId() + "is running");
} catch (Exception e) {
System.out.println ("Exception is caught");
}
}
public class multithread {
public static void main (String [] Args) {
int n = 8;
for (int i = 0; i < n; i++) {
Multithreading_demo object = new multithreadingdemo ();
object.start(); //invokes run funcn
}
}
}
```

DATE

Multithreaded Server can handle multiple Client requests

```
Public multithreaded_demo extends thread{
    Socket socket;
    Public multithreaded(socket S){ //Constructor
      this.socket = S
    }
    Public void run(){
    Socket S = Socket.accept();
    DataInputStream din = new DataInputStream(S.getInputStream());
    DataOutputStream dout = new DataOutputStream(S.getOutputStream());
    String input = din.readUTF();
     String reverse = "";
    for(int i=0; i< reverse.length(); i++){
      reverse = (char)input.charAt(i)+reverse;
    }
    dout.writeUTf(reverse);
    }
}
Public class Server {
        Public Static void main(String args[]){
        ServerSocket SS = new ServerSocket(5000);
         while (true){
         Socket S = SS.accept();
         multithread_demo handler = new multithread_demo(S);
           handler.start(); }
```

PAGE

DATE

PAGE

DATE

Socket Programming:-

- Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end and attempts to connect that socket to a server.

  When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by reading and writing from & to the socket when establishing connection

- The server instantiates a ServerSocket object denoting port number communication to occour on.

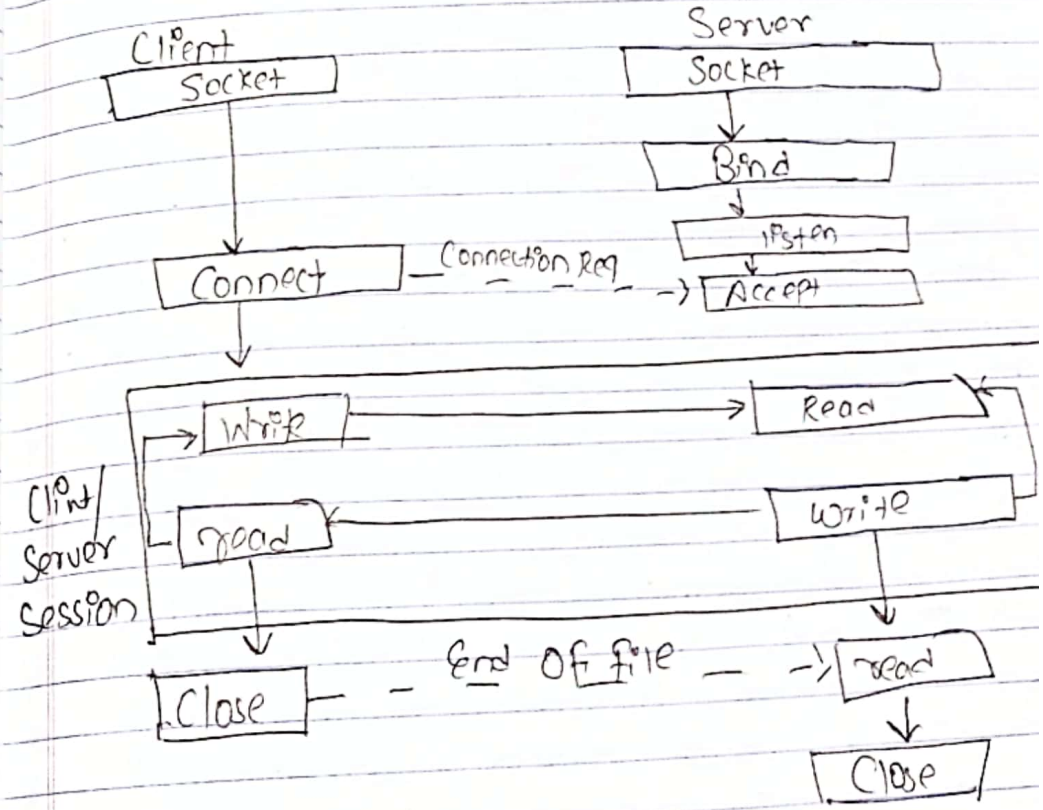
- The server invokes the accept() method of server socket class. This method waits until a client connection is established on the port

- The constructor of socket class attempts to connect the client with server and port. If communication is established then client now has a socket capable of communicating with server

DATE

- On the server side accept() method returns a reference of new socket that is connected to client socket.

The client's OutputStream is connected to server's input stream and vice-versa.



[illegible]

TCP	UDP
[illegible]	[illegible]
Transmission control protocol	User datagram protocol
[illegible] connection [illegible]	Connectionless or no such [illegible]
- Connection closed once transmission is complete	No requirement for opening, maintaining or terminating a connection
- Guarantees delivery	no guarantee
- Retransmission of lost packets is possible so reliable	No retransmission so unreliable
- Slower than UDP	faster than UDP
- Used for applications where reliability is priority	where latency is priority and occasional data loss is bearable
Doesnot support broadcasting	support broadcasting
Used by HTTPS, SMTP, FTP, POP etc	Videocalling, streaming etc

A Socket is a software component that establishes a bidirectional communication channel between two programs running on a network. It allows the programs to communicate over a network, either on same machine or different machine.

In client server model, a socket is used to establish connection by client and to listen to incoming requests by server. Used for network applications.

Ports: Ports are used to uniquely identify a network protocol connection. Ports are essential in network communication as they allow multiple network applications to run on same computer without interfering each other. Each network service is assigned a unique port number and OS uses it to route incoming network traffic to correct service.

Ports are used in conjunction with sockets to establish network connection. A Java program uses socket to open a connection to a specific port on remote computer and then communicates with network service running on that port.

The computer is determined by IP address and the application for which data is to be delivered is determined by port number.

DATE

JDBC API

The JDBC API is designed to allow developers to create database connections and perform activities on the database. Such as mySQL, SQLServer it provides set of interface and classes that allows java program to connect to a database, send SQL statements, retrive result.

1) Load the JDBC Driver class.forName.
2) Establish Connection DriverManager.getConnection()
3) Create a Statement Conn.prepareStatement
4) Retrive result. st.executeQuery(sql);

There are two distinct layer within the JDBC API

1) The driver layer : loading and unloading drivers. DriverManager, Class.forName interfaces are provide .getConnection()

2) Application layer: Allows writing and executing database query. eg. prepareStatement(), resultset class lies under application layer.

PAGE

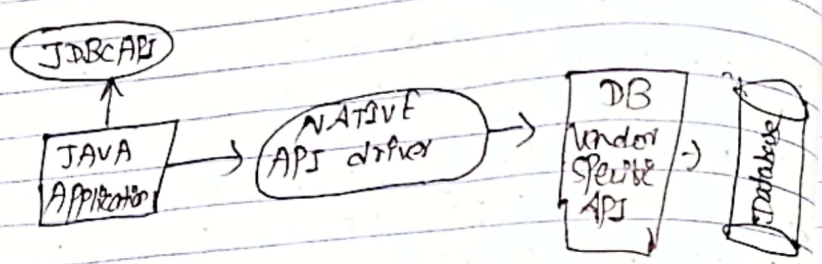
DATE

ResultSetMetadata:

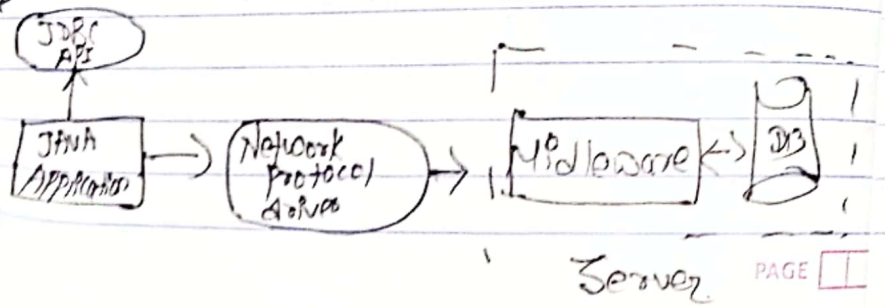
It is used to get metaData of the resultset object. Every resultset object is associated with one ResultSetMetaData object. This object will have all the meta data about a resultset that contains metadata like: Schema name, tablename, Column name, number of columns, data type of columns.

```
ResultSetMetaData = rs.getMetaData();
```

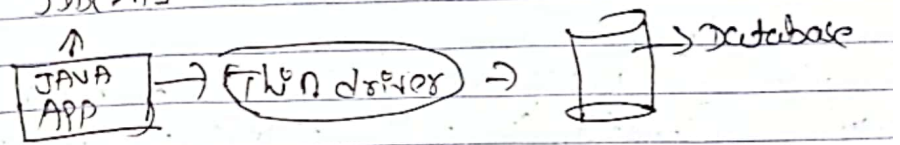
2) Native API driver:- This driver uses database specific native library to connect with the database. This has removed OS dependency. The database vendor provides the API to connect to database. The drawback is that the API should be installed on the system.



3) Network Protocol driver:- Network protocol driver is just a small modification on Native API driver. Network protocol driver uses a middleware on the server side that converts JDBC calls into vendor-specific database protocols. When database responds the middleware translates it into JDBC. There is no need for client side installation. Network support is required on client machine.



4) Thin driver:- This driver uses a JDBC url provided by the database to connect to the database. It eliminates the need of any sort of middleware. This driver is typically used in web apps. The thin driver interprets the URL and a network connection is established. Then the application sends it's credentials for authentication to access the database. Once it has been established the Java application can now send SQL statements.



for mysql "com.mysql.jdbc.Driver"

Example to connect Java application with mysql database:

```
Create database sonoo
Create table emp(id int(10), name varchar(20),
age int(3))
```

```
import java.sql;
class mysqlcon {
public static void main(String args[]){
try {
Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/database","username"
,"password");

String sql = "SELECT * from emp";
ResultSet res
PreparedStatement Statement = Conn.prepareStatement(sql)
ResultSet res = Statement.executeQuery();
while(res.next()){
System.out.println(res.getInt(1) + res.getString(2)
+res.getInt(3));
}
conn.close();
} Catch (Exception e) {
sout (e);
}
}
```

WAP to ask user to enter a name & then query the database and display total number of employes with that name.

```
import java.sql;
class mysql {
public static void main(String[]args){

Class.forName("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection("jdbc:
mysql://localhost:3000/PU","username","password

String sql = "SELECT * FROM PU WHERE
Name=
Scanner S = new Scanner(System.in);
String name = S.nextLine();
String sql = "SELECT COUNT(*) FROM PU WH
Name = ?";//where?= parameter
SetString(1,name);
PreparedStatement Statement = Conn.prepareStatement(S
Statement.setString(1,name);
Resultset result = Statement.executeQuery();
result.next(); // move cursor to only row returned by qu
int count = result.getInt(1);
System.out.println(count);
Conn.close();
}
}
```

A database "testdb" contains a table "student" with fields id, name and adress. Write a program that asks user ~~to enter the information~~ that displays records whose salary is more than 25000.

```
class db {
Public static void main(String[] args) {
  try {
  Class.forName("com.sql.jdbc.Driver");
Connection con = DataManager.getConnection("jdbc
:mysql://localhost:3000/pu", "username", "password")
String sql = "SELECT * FROM pu WHERE
  salary > 25000";
PreparedStatement st = con.prepareStatement(sql);
ResultSet res = st.executeQuery();
while (res.hasnext()) {
System.out.println(res.getString("name") +
res.getString("adres" + res.getInt("salary"));
}
} catch (IOException e) {
System.out.println(e);
}
```



```
CREATE TABLE student (name VARCHAR(255), roll no);
```

Ask user for id and name and save it to the database

```
class db {
Public static void main(String[] args) {
  try {
  Class.forName("com.sql.jdbc.Driver");
Connection conn = DriverManager.getConnection("jdbc:
mysql://localhost:3000/pu", "username", "password")
String Boolean continue = "yes"; Scanner s = new Scanner(Syst
loop;
while (continue.equalsIgnoreCase("yes")) {
System.out.println("Enter name");
  String name = s.nextLine();
System.out.println("Enter id");
  int id = s.nextInt();
String sql = "INSERT INTO COSTUMER (id, username
VALUES (?, ?)";
PREPared statement st = Prepare statement(sql)
  st.setInt(1, id);
  st.setString(2, name);
  st.executeUpdate();
System.out.println("yes or no");
  continue = s.nextLine();
  }
} Ceten(SQL Exception e) {
    System.out.println(e);
  }
}
}
```

Update salary of employees whose post is manager.

```
UPDATE EMPLOYEES SET Salary = 5000 WHERE
Post = 'manager';
Public Class db {
  Public static void main (String [] args) {
    Class.forName ("com.mysql.jdbc.Driver");
Connection conn = DriverManager.getConnection (
jdbc:mysql://localhost:3306/DB", "username", "password");
Scanner S = new Scanner
System.out.println ("enter Salary");
  int Salary = S.nextInt();
String SQL = "UPDate EMPLOYEE SET
Salary = ? where post = "manage" ";
Prepared Statement St = Conn.prepareStatement(SQL);
  St.setInt (1, Salary);
  St.executeUpdate();
  } catch (SQLException e) {
  Sout (e.printStackTrace);
  }
}
```

Java applet code to allow user to input two numbers and calculate

```
Java.awt.textfield
Class appletdemo extends Applet {
  Textfeild t1 = new Textfeild (10);
    t2
  label l1 = new label ("first number");
  Button add = new button ("Add")
  Textfeld t3 = new Textfeild (10);
  Public void init () {
    Add (l1)
    add (t1)
    add (l2)
    add (t2)
    Add (Add);
    add (t3);
  Add. addActionListner () } new ActionListner () {
    Public void action (ActionEvent e) {
      int a = Integer.parseInt (t1);
      int b =
      c = a + b.
      t3.setText (c) }
        }
}
```

<applet code = "appletDemo.class"> </applet>

getheight() * data[i]
maxvalue.

DATE

INAP in applet to play, pause and restart audio.

```
Public class audio extends Applet implements ActionListner {

    Public void init() {
    Button play = new Button("play");
    Button pause = new Button ("pause");
    Button restart = new Button ("restart");
    AudioClip clip = new getAudioclip(getCodeBase(), "music.wav"
        add(play);
        add (pause);
        add (restart);
        play.addActionListner (this);
        pause.addActionListner (this);
        restart.addActionListner (this);
    }
    Public void actionPerformed (Action event e) {
        If (e.getSource() == play) {
            clip.play();
        }
        If (e.getSource() == pause) {
            clip.stop()
        }
        elseif (e.getSource == stop) {
            clip.setframePosition (0);
            clip.stop();
        }
    }
}
```

PAGE

Q In a class of 100 students, 20 of them Prefer Java while 35 students use PHP, 30 students use Python and remaining use Go lang as their Preferred backend language. Illustrate this use using bar diagram:

```
Public class diagram extends JPanel {
    int[] data = {20, 35, 30, 15};
    Public diagram() {
        JFrame frame = new JFrame();
        frame.setSize(200, 400);
        frame.setVisible(true);
        int max = data[0];
        for (int i=0; i<data.length; i++) {
            if (data[i] > max) {
                max = data[i];
            }
        }
    }
    Public void paintComponent(Graphics g) {
        int x=0;
        for (int i=0; i<data.length; i++) {
            int width = (int)(getWidth()/data.length);
            int height = (int)(((double)(getHeight()/maxvalue))*data[i]);
            int y = getHeight() - height; // drawn from top to bottom
            g.setColor(Color.blue);
            g.fillRect(x, y, width, height);
            x += barwidth;
        }
    }
}
```

```
Public static void main(String[] args) {
    diagram d = new diagram();
    JFrame frame = new JFrame();
    frame.setSize(400, 400);
    frame.setVisible(true);
    frame.add(d);
}
```

100% correct

fillArc (x value for top left ... end angle);

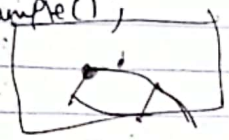
DATE

Draw a pie chart for given data:

```
Public class PiechartExample extends JPanel {
    Public PiechartExample () {
        Random random = new random ()
    Int[] data = { 11, 12, 15, 16, 10 };
    Color[] Colors = new Color[5];
    for(Int i=0; i<5; i++) {
        Colors[i] = new Color ( random.nextFloat(),
    random.nextFloat(), random.nextFloat() );
    }
    JFrame frame = new JFrame();
    frame.setSize (200, 200);
    frame.setVisible (true);
}
Public void PaintComponent (Graphics g)
    Int centerX = getWidth()/2;
    Int centerY = getHeight()/2;
    Super.PaintComponent(g);
int radius = (int)(Math.min(width, height) * 0.4)
    Int startAngle = 0;
    for(Int i=0; i<data.length; i++) {
int endangle = (int) ( (data[i] / total ) * 360 )
    g.setColor (colors[i]);
g.fillArc (centerX - radius, centerY - radius, radius*2, radius*2,
    StartAngle, endangle );
startangle = endangle
}
```

DATE

```
//draw a circle in the center of pie chart.
    g.setColor (color.WHITE)
    g.fillOval ((centerX - radius)/2 (centerY - radius)/2
    radius, radius);
}
Public static void main (String[] args) {
    PieChartExample p = new PiechartExample();
    frame.add(p);
}
```



Difference Between Panel	Frame
It is an AWT component that can attach other GUI containers including other panels	It is a top level window that has border and title
- Subclass of container Public class Panel extends Container implements Accessible{	Subclass of window Public class frame ex window {
- It doesnot have a title bar	It has a title bar.
- It doesnot have a border	It has a border
- Panel is contained inside frame	It contains Panels.
- Uses flowlayout as default	uses border layout default.

PAGE

```
Public class Myframe() extends JFrame(a){

Public myframe(){

JPanel Panel = new JPanel();
JFrame frame = new JFrame();
JPanel.setLayout(new BoxLayout(Panel, BoxLayout.Y_A
Box topBox = CreateVerticalBox();
topBox.SetAlignmentX (CENTER_ALIGNMENT);
topBox.add(new JButton("Button 1"));
topBox.add (new JButton ("Button 2"));
topBox.add (new JButton ("Button 3"));
Box bottomBox = CreateVerticalBox();
bottomBox.SetAlignmentX(CENTER_ALIGNMENT);
bottom box.add ("A very long name button");
panel.add (topBox);
Panel.add (Box.CreateHorizontalGlue);
Panel.add (bottomBox);
}
```

Align box to horizontal Center

Layouts in Java:

1) Grid layout: This type of layout arranges elements in grid with certain number of rows and columns. In Java swing

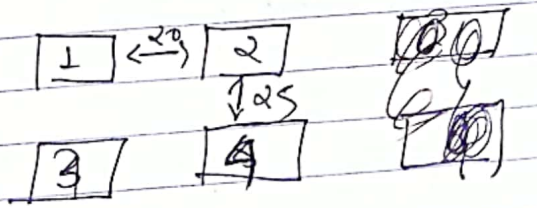
The functions associated with it are.

i) GridLayout(): There is only one row and the number of column is the number of elements added.

ii) GridLayout(int rows, int col);

iii) GridLayout(int rows, int col, int horizontalgap, int verticalgap);

```
Public myframe(){..
    JFrame frame = new JFrame();
    frame.setSize(200,200);
    Frame.setLayout(new GridLayout(2,2,20,25))
    frame.add(new JButton("1"));
    frame.add(new JButton("2"));
    frame.add(new JButton("3"));
    frame.add(new JButton("4"));
    frame.setVisible(true);
}
```



DATE

Box layout :- It is a layout manager that arranges components in single row or column. On BoxLayout you can specify alignment for each component either by setting alignment for each component or by setting the alignment for a container as whole. We can also specify the amount of space that should be placed

- CreateVerticalStrut(x) :- add x spacing up and down
- CreateVerticalBox() :- creates a box that aligns elements vertically
- box.setAlignmentX(CENTER_ALIGNMENT), aligns box in the center.

PAGE

WAP to display only those record whose ~~Employee~~ adress is Kathmandu.

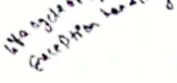
```
Public Class Employee {

    Public Static void main (String [] args) {

        FileInputStream fis = new FileInputStream ("file.txt");
        DataInputStream din = new DataInputStream (fis);
while((i=din.read())!=-1)while ( din.available() > 0 ) {
                String name = din.readUTF();
                String adress = din.readUTF();
                int salary = din.readInt();
                if ( adress.equalsIgnoreCase ("Kathmandu")
                    System.out.println ("");
                }
            }
        }
```

Generate the following using Box layout

```
Public oldisgood (){
JFrame frame= new JFrame();
JPanel panel= new JPanel();
panel.setLayout (new BoxLayout (Panel, BoxLayout.Y_AXIS));
Box top= Box.createHorizontalBox();
top.setAlignmentX (CENTER_ALIGNMENT);
JButton b1 = new JButton ("Hi");
top.add (b1);
Box mid= Box.createHorizontalBox();
mid.add (new JButton ("1"));
mid.add (new JButton ("2"));
mid.add (new JButton ("n"));
Box b3 =Box.createHorizontalBox();
b3.add (new JButton ("3"));
Panel.add (Box.createHorizontalGlue()); to spread
Panel.add (top);
Panel.add (Box.createHorizontalGlue());
Panel.add (Box.createVerticalGlue());
Panel.add (mid);
Panel.add (Box.createVerticalGlue());
b3.setAlignmentX (CENTER_ALIGNMENT);
Panel.add (Box.createHorizontalGlue());
Panel.add (b3);
Panel.add (Box.createHorizontalGlue());
}
```

DATE

Exception handeling in Java

Exception handeling is the process of managing and responding to errors and exceptional conditions that occur during the execution of program. When an exception is thrown in Java, the normal flow of the program is disrupted and control is transferred to an exception handler that can take appropriate action to handle the exception. A personalized error handeling class can be created using by extending the Exception class

```
Public customexception extends Exception {

    Public Customexception (){
        Super ();
    }
    Public  Customexception (String message) {
            Super (message);
    }
    Public Customexception (String message, Throwable ca
            Super (message, cause.);
    }
}
Public class demo {
  Public static void main (String[] args) {
try{ if (true) throw new customexception ("error");

} catch (custom exception error){
    System.out.println (error.getMessage());
}
```

DATE

URL And URL Connection Classes In Java

=) In Java URL class is a class that represents a uniform resource identifier reference. A URL is used to specify the location of a resource on the internet.

```
Public class ex {
    Public static void main (String [] args) {
    URL url = new URL ("http://www.send.com/hi?");
    System.out.println (url.getHost() + url.getQuery()
    + url.getPort() + url.getProtocol() + url.getPort()
    );
}
```

URL Connection class :- It is an abstract class that represents a connection of a resource as specified by the corresponding URL. It provides a control over interaction with server. From URL class we can also download binary files by using URLConnection.

DATE

```
Public static void main (String [] args) {
try {
URL u = new URL ("www.website.com");
URLConnection urlconnect = u.openConnection();
// HttpURLConnection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("GET");
DataInputStream din = (Url)connect.new DataInputStre
    (Url)connect.getInputStream());
while ((i = din.read()) != -1) {
    System.out.print((char) i);
}
} catch (exception e) {
}
}
```

DATE

What makes Java platform independent?

Java is considered platform independent because Java code can be compiled to bytecode which is an intermediate code that can run on any platform that has Java virtual machine. The JVM is a software that interprets and executes bytecode, so as long as the platform has JVM installed it can run Java.

The key-feature that makes Java platform-independent

1) Bytecode: Java is compiled into bytecode and they can be compiled by any platform that has JVM

2) JVM: A JVM is platform specific implementation of Java virtual machine. The JVM is responsible for interpreting the bytecode and executing it on the host platform.

3) Standard Libraries: It is a collection of classes and interfaces that provide a wide range of functionalities including I/O, networking, concurrency etc. The standard library provides a consistent set of APIs that can be used on any platform.

while (str.available() > 0)
String name = din.readUTF();
int age = din.readInt();

DATE

Method overloading

Method overloading feature allows a class to have multiple methods with the same name but different parameters.

```
Public class Overloaded {

    Public void display (int x) {
        System.out.println("Integer is: " + x);
    }
    Public void display (String s) {
        System.out.println("String is " + s);
    }
    Public static void main(String[] args) {
        Overloaded O = new Overloaded();
        O.display(10);
        O.display("Hello");
    }
```

Multiple Inheritance

DATE

Inheritance in Java:

The property of a class to inherit properties of other subclass is called as inheritance

```
Public class Bicycle{
  Public int gear;
  public int speed;

Public Bicycle(int gear, int speed){
  this.gear=gear;
  this.speed=Speed;

}
Public void brake(int decrement){
      speed -=decrement;
}
Public void accelerate(int increment){
      Speed+=increment;
}
Public void toString(){

System.Out.println("speed":+speed+"gear":+gear);
}
}
```

PAGE

DATE

```
Public class MountainBike extends Bicycle{
  Public int seatheight;

Public MountainBike(int speed, int gear, int seatheight){
    Super(Speed, gear);
    This.seatheight=seatheight;
}
Public void Seatadjust(int new){
    seatheight=new;
}
@override public void toString(){
  System.Out.println(gear+speed+seatheight);
}
}
Public class test{
Public static void main String[] args){
  MountainBike m= new MountainBike(0,10,15,
  & m.toString();
```

PAGE

Why multiple inheritance is not supported in Java?

Java doesnot support multiple inheritance because it can lead to several problems, such as the diamond problem. The diamond problem occurs when a class inherits from two or more classes and the class attempts to call a method that defined in both the super classes. In this scenario, the compiler cannot determine which version of the code to call it leads to ambiguity and confusion in the code

To avoid this Java uses single inheritance, where a class can inherit from one superclass. However Java provides an alternative to multiple inheritance with the help of interface and implementation

```
Public Interface A {
    Public void foo();
}
Public Interface B {
    Public void Bar();
}
Public class myclass implements A, B {
    Public void foo() {
        System.out.println("foo");
    }
    Public void bar() {
        System.out.println("Bar");
    }
}
```

using interface in this way provides a form of multiple inheritance, where a class can inherit behaviours from multiple sources. However it doesnot suffer from the diamond problem since there are no conflicting implementations to resolve.

Interface and Implementation in Java

In Java, an interface is a collection of abstract methods that define a set of behaviours. An interface specifies a set of methods that a class must implement but it doesnot provide implementation for these methods. By defining only abstract methods, an interface achieves 100% abstraction. Abstraction is the process of hiding the implementation details and exposing only the necessary information to the user. An interface does this by providing clear and consise definition of a methods that a class must implement without exposing details of how they are implemented.

```
Interface A {
    Public int Area();
}
Class Circle implements A {
    Public int Area() {
        return 2*3.14*4*4;
    }
}
```

Packages in Java

Packages is a grouping mechanism that allows us to organize related classes and interface. A package is a way to organize classes and interface into seperate namespace. Packages enable us to avoid naming conflict and make it easier to manage code.

Consider two classes with same name Myclass. If these classes are in different packages then it helps us to avoid naming conflicts. Since one class is referred as "com.example.package1.myclass" and another is referred as "com.example.package2.myclass".

A package is simply a directory on the filesystem that contains set of related classes and interface. The package name corresponds to folder name & classname corresponds to file name.

Package com.example

```
Public Class Myclass {
  // code here
}
```

In general package are used to group related classes and interface together making it easier organize and manage code.

JAVA Inner class:-

In java it is possible to nest classes (a class within a class) for the purpose of grouping classes that belong together which makes the code more readable and maintainable.

To access the inner class create an object of outerClass and with the help of outerClass create an object of inner class.

```
class outerclass {
    int x=10;
  class innerClass {
      int y=5
  }
}
Public class main {
    Public Static void main(String[] args) {
      Outerclass myOuter = new OuterClass();
      Outerclass.innerclass inner = myouter.new innerclass
        System.out.println(inner.y + myouter.x);
```

DATE

Access Specifiers in Java

In Java, Access specifiers control the visibility or accessibility of classes, methods and variables. These access specifiers are

i) Public :- They can be accessed from any class or package

ii) Private :- classes, methods and variables that are declared private can be accessed only within the same class

iii) Protected :- The classes, memod and variables declared as protected can be accessed within the same class or by subclass or by other classes in same package. not outside package. if not inheritave

(4) Default (or Package Private) :- Can be accessed only within the same package

PAGE

Convert Applet to App
clock using Applet
Reflection API
TCP vs UDP

DATE

	Acc. by classes in same Package	classes in oth pack	Subclass in same pack	Subclass in oth p
Public	✓	✓	✓	✓
protected	✓	X	✓	✓
Private	~~✓~~ ~~✓~~ X	~~✓~~ X	~~✓~~ X	X
Defaut	✓	X	✓	X

Font f0 = new Font ("Arial", Font.Bold, 15);

PAGE

clock using Applet

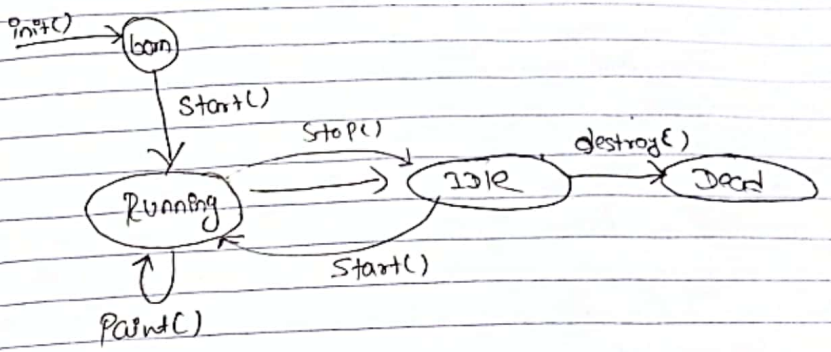
```
import java.awt;
import javax.swing;

Public class DigitalApplet extends JApplet {
    Public void init() {
    JLabel label = new JLabel();
    label.setFont (new Font ("Arial", Font.BOLD, 16));
    add (label);
    }

    Public void start() {

    while (true) {
    Calendar cal = Calendar.getInstance();
    SimpleDateFormat f = new SimpleDateFormat ("hh:mm:ss");
    String time = f.format (cal.getTime());
    label.setText();
try { Thread.sleep(1000);
    } catch (InterruptedException e) {
        System.out.println(e.getMessage());
    }
    }
}

/applet code = DigitalApplet.class width = "200" height = "200"
    /applet>
```

init - loaded, setups in browser, see clears.
start: starts the applet shows necessary items.
stop: pauses.
destroy: clears the memory kills applet.

JMenuBar
JMenu
JMenuItem
addSeparator();

Create a Menu in Swing.

```
Public class menu extends JFrame {
    Public menu(){
    JFrame frame = new frame("frame");
    JMenuBar menubar = new JMenuBar();
    JMenu menu = new JMenu();
    JMenuItem J1 = new JMenuItem();
    JMenuItem J2 = new JMenuItem()
    menu.add(J1);
    menu.add(J2);
    menu.add Seperator();
    menubar.add(menu);
    frame.add(menubar);
    frame.setVisible(true);
    frame.setSize(200,200);
}
}
```

jdbc:mysql://localhost:3306/database

Q WAP to insert data in following table.

```
Public class db {
    Public static void main(String []args){

load driver     Class.forname("com.sql.jdbc.driver");
    Connection con = DriverManager.getConnection
("sql.jdbc://Port3306/pu","username","password");

    Scanner S = new Scanner();
    System.out.println("enter id, NAme &ROll");
      int id = S.nextInt();
       Put name = S.nextString();
       int Roll = S.nextInt();
    String sql = "INSERT INTO Student(id, name ROLL(INT)
    VALUES (?, ?, ?)";

    PreparedStatement S = con.PrepareStatement(sql);
       S.setInt(1, id));
       S.setString(2, name))
       S.setInt(3, Roll));
    Resultset S =  S.execute Query();
}
}
```

Reflection API

The reflection API in Java provides a way to inspect and manipulate classes, interface, objects, methods, fields and other components of Java at runtime. It allows us to perform various operations such as obtaining information about class, examining its constructor and method, invoking methods and creating new instances of class.

1) Class.forName("string"): Returns a class for the given name in string.

2) obj.getClass(): Returns class of that object

3) isInterface(): This method returns boolean value.

) getArray():

Class.getConstructor():

Constructor.newInstance():

ethod.Invoke():

Why is Reflection API important?

1) Getting access to classes methods at runtime.

2) Debugging

3) Building frameworks: where new classes are added at runtime

4) Code generation

5) Testing

Cons

i) Breaks encapsulation as it is possible to access private methods and fields with reflection

ii) Overhead in performance since types in reflection are resolved dynamically, JVM optimizations cannot take place

iii) Security is compromised.

Event handling mechanism

DATE

Delegation event model

Delegation event model is a programming paradigm used in programming languages to handle events generated by user actions such as mouse clicks, button presses. In this model, an object that generates an event doesnot handle the event itself but delegates the task to another object ("Listener") that is registered to handle events

```
Public class ButtonDemo extends JFrame{

    Public ButtonDemo(){
    JLabel label = new JLabel();
    JButton button = new JButton();
    Button.addActionListener(new ActionListenerX());
    frame.add(label);
    frame.add(button);
    frame.setVisible(true);
    }
}
public class ActionListenerX implements ActionListener{

    Public void actionPerformed(ActionEvent e){
        label.setText("Clicked");
    }
}
Public static void main(String[] args){
    ButtonDemo();
```

PAGE

We register ActionListenerX class's object as an event listener to for Button. The "ActionListenerX" implements ActionListener interface and register it as a button listner by calling actionPerformed method of button when button is clicked actionPerformed() method is called that handles the event.

PAGE

DATE

Create a frame with two text fields, one of which should show whether the mouse pointer is inside or outside the frame the other text field should display the x and y coordinates of the pointer when the user moves the pointer inside frame area.

```
Public class mouseDemo extends JFrame implements ActionListener {

    Public mouseDemo() {

    JFrame frame = new JFrame();
    JLabel label1 = new JLabel();
    JLabel label2 = new JLabel();
    frame.addMouseListener(this);
    frame.setVisible(true);
    }

    Public void mouseEntered(MouseEvent e) {
        label1.setText("Entered");
        label2.setText("X :" + e.getX() + "Y :" + e.getY());

    Public void mouseExited(MouseEvent e) {
        label1.setText("Exited");
        label2.setText(" ");
    }
    Public void mouseMoved(MouseEvent e) {
        label2.setText(e.getX(), e.getY());
    }
}
```


DATE

Various event listners in Java

→ Inheritance and Composition

Inheritance is a mechanism in which a class acquires property of another class. The class that inherits property and behaviour is called subclass and it inherits from superclass. Inheritance in Java is achieved using extends.

Composition is method for reuse of code achieved by creating object of a class inside another class.

```
class A {
    public void display() {
        System.out("Hi");
    }
}
class B {
    A a = new A();
    public void show() {
        a.display();
    }
}
```

Is a describes inheritance relationship eg Dog is a Animal. Animal is a superclass and Dog is a subclass. Has a is a composition where one class has instance of another class. Car has a engine. Car has a feature of class engine.

Advantages of Java's layout manager over traditional windowing systems

(i) Platform independent
(ii) consistent
(iii) Dynamic adjusts to change in size
(iv) Easy to use

Singelton class : A singelton class is a class in Java that is designed to restrict instance of an object to single object and to provide global point of access to that object. This means only one object can exist at a point.

```
class Singelton {
    private Singelton() {} // cannot access con

    public static createInstance() {
        if (instance == null) {
            Singelton instance = new singelton();
        }
        return Singelton;
    }
}
```

Useful for resource management, logging, caching configuration setting. useful to ensure only single object of a class exists.

DATE

BufferWriter: It writes data in Bulk to a file. It writes data to buffer which is flushes to the file when buffer is full or when the flush() method is called.

```
BufferWriter buff = new BufferWriter(new FileWriter(
buff.write("Hello world");
buff.write("Hi");
buff.flush();
buff.close();
```

PAGE

Reflection Api methods:

```
ClassDemo c = new classDemo();
```

Using class: `Class cls = c.getClass();`

```
Field[] fields = cls.getDeclaredFields()
                 //returns all fields
```

GetConstructor:

```
Class[] parametertype = null;
myclass Obj = new Obj();
Constructor cons = Obj.getConstructor(Parametertype);
```

GetDeclaredMethods(): returns all private & public methods

Using class:

```
myclass m = Class.forName("myclass");
Method[] met = m.getDeclaredMethods();
```

- getfield(String): Returns a field represented by the String for that object.

```
field f = Obj.getfield("data");
```

- getmethod(String, class[]):

```
Obj.getmethod("func", null);
```

DATE

Client/Server Interaction via Java

Converting Applications to Applet

i) make an html page with an applet tag to load applet page

ii) Eliminate the main method which calls the Constructor, You don't need it as applet is displayed in the browser

iii) Extend the main class from Applet instead of JFrame

iv) move application code from frame window constructor to init() method

v) Remove all setSize, setVisible methods

vi) Remove frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)

vii) Remove any sort of HTML

Multithreaded

```
pubic multithreadDemo extends Thread {
    Socket s
    Pubic multithreadDemo (Socket s) {
      this.s = s
    }
    Pubic void run() {
    DataInputStream din = new DataInputStream(s.getInpu
    DataInputstream dout = new
      String s = din.readUTF();
    dow.writeUTF (s);
    }
}
Pubic class multi {
    Pubic static void main(String [] args) {
     ServerSocket ss = new ServerSocket (5000);

     While (true) {
      Socket s = ss.accept();
      multithreadDemo A = new multithreadDemo (s
          s.start();
     }
```

Client

```
Class client {
  Public static void main (String[] args) {
    Socket s = new Socket (" ",
    DataInputStream din = new DataInputStream(s.getInputS...
    ");
    Scanner s = new Scanner (System.in);
    while (true) {
      String str = s.nextLine();
      dout.writeUTF();
      if (str = "Bye") {
        break
      }
      System.out.println(din.readUTF());
    }
  }
```

Bar diagram

```
x=0
width = getWidth() / data.length
for (int i=0; i<N; i++) {
  int height = (getHeight() / maxvalue) * data[i]
  int y = getHeight - height
  g.fillRect(x, y, width, height);
  x = x + width;
```

DATE

Why do we need to serialize an Object?

Serialization is the process of converting an Object's state into a format that can be stored, transmitted or reconstructed later. The primary reason for serializing an object is to be able to transmit or store it in a way that can be easily reconstructed later

(i) Persistance:- Serialization is used to store objects in a persistant storage medium such as a file or database so it can be retrived later.

(ii) Communication:- It transmits Objects over a network two applications can communicate with the help of serialized Objects that contain app

(iii) Caching:- Serialization is used to cache objects that are frequently used

(iv) Copying:- to make a copy of object

) Security:- encrypting an Object

DATE

Wrapper class

Wrapper class provides a object oriented representation of primitive data types (Boolean, byte, short, int, long etc) and their corresponding wrapper class (Boolean, Byte, Short, Integer, Long, float & Double) and provides useful methods and additional functionality beyond what is available for primitive types. They can be used to convert primitive type and objects as well as to manipulate primitive types using object oriented methods

Consider ArrayList<Integer> obj = new ArrayList<>();

Integer Wrapper class is used to represent integer values as objects which can be added to arraylist

Wrapper class provide methods like
parseXXX() : Integer.parseInt()
Double.parseDouble()

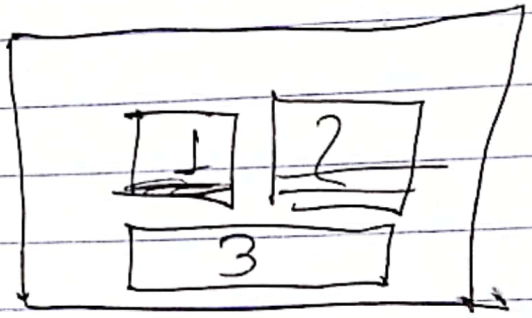
Integer.valueOf();
~~Double~~ Integer.MAX_VALUE

GridBag Layout:

```
Public class GridBag extends JFrame {

Public GridBag () {

JFrame frame = new JFrame ();
   frame.setLayout (new GridBagLayout()
GridBag Constraints c = new GridBag(c
   c.gridX = 0;
   c.gridY = 0;
frame.add (new JButton("1"), c);
   c.gridY = 0;          X means column
   c.gridX = 1;          Y means row
Frame.add( new JButton("2"), c);
   c.gridX = 1;
   c.gridY = 1
   c.fill (g: GridBagConstraints.HORIZONTAL
   c.gridWidth (2);
   Frame.add (new JButton ("3"); c);
   }
PSVM( String [] args){
GridBag a = new GridBag();
}
```



applies to panel

Box layout: It allows us to arrange components in single dimension, either row or column

panel.setLayout(new BoxLayout(Panel, BoxLayout.Y_

Public class Boxlayout {

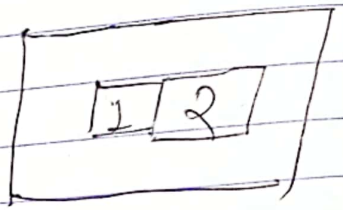
Public Boxlayout() {

Jframe frame = new Jframe();
JPanel panel = new JPanel();
frame.setSize(200,200);
panel.setLayout(new BoxLayout(Panel, Box.X_
panel.add(CreateHorizontalGlue());
panel.add(CreateVerticalGlue());
Panel.add(new JButton("1"));
Panel.add(new JButton "2");
panel.add(CreateVerticalGlue());
panel.add(CreateHorizontal Glue());
frame.setVisible(true);
frame.add(Panel);

}

}

psvm(){
Boxlayout();
}



Another way:

Public class Boxlayout() {

Jframe frame = new Jframe();
JPanel panel = new JPanel();
frame.setSize(200,200);
Box b = Box.createHorizontalBox();
b.setAlignmentX(CENTER_ALIGNMENT);
b.add(new JButton("1"));
b.add(new JButton("2"));
Panel.add(b);
frame.add(panel);
frame.setVisible(true);

}

Public Static void main(String[] args){
boxlayout b = new boxlayout();

Java foundation class: It is a part that encompasses a group of features to help people build graphical User interfaces GUI.

It contains AWT, Swing, Java 2D API

AWT: Abstract window toolkit is part of Java designed for creating user interfaces and painting graphics and images. It is a set of classes intended to provide everything a developer needs to create GUI most AWT components are derived from java.awt.component.

Swing: Swing is an extention to AWT that contains larger sets of components ranging from table, trees etc. The swing components start with J.

The Swing components are light weights while AWT components are referred as heavyweights. The difference between lightweight and heavyweight components is Z-order i.e the notion of depth or layering. Swing components are written 100% on Java thus they are platform independent.

flow layout: It arranges the components in a left to right flow much like lines of text in paragraph. It there is no extra space to fit all components the components will be placed in a new row.

```
frame.setLayout(new FlowLayout(FlowLayout.CENTER,
    10,5)) // 10 is hgap & 5 is v gap
```

Card layout: The CardLayout class manages the components in such a manner that one component is visible at a time

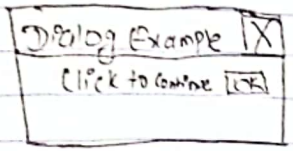
```
C = getContentPane();
frame.setLayout(new CardLayout(40,30))
b1 = new JButton("Apple");
b2 = new JButton("Ball");
b3 = new JButton("cat");
frame.add("a", b1);
frame.add("b", b2);
frame.add("c", b3)
frame.add(new JButton("next"));
b1.addActionListner(actionO{
    public void ActionPerformed(ActionEv
        card.next(C);
```

~~# Chapter 5~~

~~Event handling Events & Actionlistener~~

JDialog: It is a top level window with a border and title used to take some form of input from the user. It inherits Dialog class.

```
Public class Dialog Example {
  private static JDialog d;
    DialogExample () {
      JFrame f = new JFrame();
      d = new JDialog (f, "Dialog Eg", true);
      d.setLayout (new FlowLayout());
      JButton b = new JButton ("ok");
      b.addActionListener (new ActionListener()
Public void actionPerformed (ActionEvent e) {
            d.setVisible (false);
        }
      });
      d.add (new JLabel ("click to continue"));
        d.add (b);
        d.setSize (500, 500);
        d.setVisible (true);
        b.setSize (1000, 1000);
Public static void main () {
    DialogExample e = new DialogExample();
  }
```

DATE

Event Listner for checkbox that turns text into bold when checked.

```
JTextfield text = new JTextfield();
JCheckBox b = new JCheckBox("Bold");
b.addItemlistner(new Itemlistner(){
    public void itemstatechanged(Itemfvent e,
    mytext.setfont(new font("serif", font.Bol
    .3
}
```

Write a Program to display Pokhara university inside an ellipse

~~JFrame~~ JPanel.

```
Public class Demo extends ~~Applet~~ {
    @Override
    ~~JFrame~~ frame = new ~~JFrame~~();
    Public void paint(Graphics g){
        Super.paintComponent(g);
        g.drawOval(10, 10, 400, 200);
        g.drawString("pokhara university", 20, 2
    }

    Public static void main(String[] args){
        Demo d = new Demo();
        JFrame frame = new JFrame();
        frame.add(d);
    }
```

PAGE

Sandbox security model with regard to Java Applet

The sandbox is a set of rules that are used when creating an applet that prevents certain function when an applet is sent as a part of webpage. It the applet is allowed unlimited access to memory and Operating System resources. It can do harm in the hands of someone with malicious intent. The sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandbox is used to when executable code comes from unknown an untrusted sources and allow user to run untrusted code safely.

It uses three tier architecture.

- Byte code verifier:- It verifies the Java code compiled down as bytecode.

- Applet class loader: It determines how the Java class is loaded into applet and performs all security measures.

- Security manager: The security manager has the option to veto the operation by generating security exception.

Displaying a picture.

```
Picture = getImage(getDocumentBase(), "img.jpg");
g.drawImage(picture, 30, 30);
```

Event models in Java

i) Java 1.0 Event model:- In this approach, when a user initiated event is generated, it automatically propagated to container of that component again to the container of that container until the root container is not figured out. Then the event is handled there. The major drawback is the events have to be passed through containers that were not capable of handling it. thus wasting a lot of CPU cycles.

ii) Event delegation model :- Explained ahead.

WAP to write bytes of to a file called "Java-Ex... and reading it uptill end of file is obtained.

```
Public class Byte {
    Public static void main (String[] args) {
        FileInputStream fis = new FileInputStream("...");
        FileOutputStream fout = new FileOutputstream("...Java...");
        byte[] datatowrite = {10,20,30,40,50};
        try {
            fout.write(datatowrite);

            while ((i = fis.read()) != -1) {
                System.out.println(i);
            }
            fin.close();
            fout.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```


Types of streams in Java:

(i) Byte stream:

- fileInputstream: used to read bytes from file
- fileOutputstream: used to write bytes to a file.
- ObjectInputStream: used for reading objects from...
- ObjectOutput stream.

(ii) character streams:

fileReader: read characters from file
fileWriter: read write characters
BufferedWriter: used to write characters wi... buffering.
Reader: abstract class as superclass for all ...
writer: ...

#

```
InetAddress addr = InetAdress.getLocalHost();
System.out.println(addr.getHostAdress());
System.out.println(addr.getHostName());
```

Q. 2018 fall:
How is interface different from abstract class?

1) Variables declared in interface are final i.e they cannot be modified. An abstract class may contain non-final variables.

2) Abstract class doesn't provide 100% abstraction as it has implementation of certain methods.

3) Java interface is used using implements keyword while abstract class is used using extends keyword.

4) A Java class can implement multiple interface but can only extend single abstract class.

```
Abstract class employee{
    abstract void display();
}
class developer extends employee {
void display() {
    System.out.println("developer");
}
}
```


Q. 2017 fall:
Explain with example: static block, static variable and static method.

Static block: It is used for static initialization of a class. The code inside static block is executed only once.

```
class test {
    static int i;
    int j;
    static {
        i=10;
        System.out.println("static block called");
    }
}
```

Static variable: All the objects of the class share static variable. Static int i;

Static method: Static methods can be called without having an object. These methods belong to a class rather than to any object.

Instance Variable hiding

If there is a local variable with same name as instance variable then the local variable hides instance variable. However the instance variable can be achieved using this keyword.

```
class demo {
    int value=10;

    public method () {
        int value=20
        System.out.println (value); //20
        System.out.println (this.value); //10
    }
}
```

Errors and Exception in Java:

An error is deviation of program from it's expected behaviour. Errors can occour for a variety of reasons such as Input incorrect, System failure or software bugs. Errors can result in one system or program producing incorrect or unexpected result or crashing altogether.

When an error is predicted and dealt using try catch block. It becomes an exception. Errors are mostly caused by the factors out of program control. Errors are broader category of issues that can affect a system or program as a whole. exceptions are specific type of error that occur during execution and can be handled by programmer.

Write a program to write name, roll and address in record.txt file

```
class file {
    Public Static void main(String[] args){
    Scanner S=new Scanner (System.in);
    FileOutputStream fos=new FileOutputStre
    DataOutputstream dos=new DodoOutPutStre
      While (true){
      System.out.println("Enter name");
      name=S.nextLine();
      dos.writeUTF(name,roll,adress)
        If (name.equalsIgnoreCase("
            break;
        }dos.close;
    }    fos.close();
```

Copy file using BufferedStream

```
class Copy {
  Public Static void main (String[] args){
  FileInputStream fis=new FileInputStream("
  BufferedInputstream bis=new BufferedInput
  BufferedOutputStream bout=new BufferedOutputStre
    int data;
    While(data=bin.read()!=-1){
        bos.write(data);
    }
}
}
```

PAGE

Multithreaded Server

DATE