

The solution is downloaded and managed from

<http://www.rajanyal.com.np/>

<http://hutrajshrestha.com.np/>

Visit site For bsc csit syllabus/old question/notes/solution

You can follow the site for more notes.

Our Social Links

[Rajan Facebook](#)

[Rajan youtube channel](#)

[Hutraj Facebook](#)

[Hutraj youtube channel](#)

[Like Our FaceBook Page](#)



[Bsc Csit Nepal \(Official\)](#)

Note : if you found any mistake on solution then please send us a message .

(You don't need to write long answer. Some of our answer is very long for better understanding so summarize yourself)

Tribhuvan University
Institute of Science and Technology
B.Sc. CSIT, 7th Semester
Model Question Paper

Subject: Advance Java Programming

FM: 60 PM: 24

Course No.: CSC-403

Time: 3 Hrs.

Section A

Attempt all Questions. [8×5=40]

1. Why concept of interface is important? How multiple-inheritance is supported by interface? Explain with suitable example.

Answer:

Refer to TU Exam Solution 2070.

2. When the exception handling constructs throws and finally is important? Explain both of them with suitable example.

Answer:

Refer to TU Exam Solution 2069.

3. Write a program to read ID, Name, address, salary of twenty employees from keyboard and write in into the file “emp.doc”. Again read records of the employees and display records of those students whose salary is more than 25000.

Answer:

```
//Employee.java
```

```
import java.io.RandomAccessFile;
```

```
import java.util.Scanner;

public class Employee {

    private int id;

    private String name;

    private String address;

    private double salary;

    public int getId() {

        return id;

    }

    public void setId(int id) {

        this.id = id;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getAddress() {

        return address;

    }

    public void setAddress(String address) {

        this.address = address;

    }

    public double getSalary() {
```

```

    return salary;
}

    public void setSalary(double salary) {
this.salary = salary;

    }

    public static void main(String []args)    {

final int size = 20;

    int i;

    Employee e[] = new Employee[size];

    Scanner sc = new Scanner(System.in);

    int id;

    String n;

    String a;

    double s;


    //setting student objects

    for(i=0; i<size; i++) {

        e[i] = new Employee();

        System.out.println("Enter id of emp "+(i+1)+":");

        id = sc.nextInt();

        System.out.println("Enter name:");

        n = sc.next();

        System.out.println("Enter address:");

        a=sc.next();

        System.out.println("Enter salary:");

```

```
s=sc.nextDouble();  
e[i].setId(id);  
e[i].setName(n);  
e[i].setAddress(a);  
e[i].setSalary(s);  
}
```

```
//write to file
```

```
try {  
    RandomAccessFile f = new RandomAccessFile("D:\\emp.doc","rw");  
    for(i=0;i<size;i++) {  
        f.write(e[i].getId());  
        f.writeUTF(e[i].getName());  
        f.writeUTF(e[i].getAddress());  
        f.writeDouble(e[i].getSalary());  
    }  
    f.close();  
}  
catch(Exception ex) {  
    ex.printStackTrace();  
}
```

```
//read from file and display salary greater than 25000
```

```
int id1[] = new int[size];
```

```
String naam[] = new String[size];
```

```

String add[] = new String[size];

double sal[] = new double[size];

try {

    RandomAccessFile f = new RandomAccessFile("D:\\emp.doc","r");

    for(i=0;i<size;i++) {

        id1[i] = f.read();

        naam[i] = f.readUTF();

        add[i] = f.readUTF();

        sal[i] = f.readDouble();

        if(sal[i]>25000) {

            System.out.println("ID:"+id1[i]+"\\tName:"+naam[i]+"\\tAddress:"+add[i]+"\\tSalary"+sal[i]);

        }

    }

}

catch(Exception ex) {

    ex.printStackTrace();

}

}

```

4. Why adapter classes are important? Compare it with listener interface with suitable example.

Answer:

The Adapter class provides the default modification of all methods of an interface; we don't need to modify all the methods of the interface so we can say it reduces coding burden. Sometimes or often we need a few methods of an interface. For that the Adapter class is very helpful since it already modifies all

the methods of an interface and by implementing the Adapter class, we only need to modify the required methods.

Advantages of the Adapter class

- Assists unrelated classes to work together.
- Provides a way to use classes in multiple ways.
- Increases the transparency of classes.
- Its provides a way to include related patterns in a class.
- It provides a pluggable kit for developing applications.
- It makes a class highly reusable.

Comparison between adapter class and listener interface

In an adapter class, there is no need for implementation of all the methods presented in an interface. It is used when only some methods of defined by its interface have to be overridden.

In a listener, all the methods that have been declared in its interface have to be implemented. This is because these methods are final.

5. Write a program using RMI to find sum and difference of two numbers. Methods sum and difference should be invoked from some remote machine.

Answer:

Here, the remote machine will contain three things: first one is the Message interface, second one is the MessageImplementation class which implements the Message interface and third one is the Server class which creates Registry for lookup by the client. The client machine will contain two things: one is the Message interface and another is the Client class. The complete code is given below:

```
//Message.java
```

```
import java.rmi.Remote;  
import java.rmi.RemoteException;
```

```
public interface Message extends Remote {  
    double sum(double num1, double num2) throws RemoteException;  
    double difference(double num1, double num2) throws RemoteException;  
}
```

```
//MessageImplementation.java
```

```
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;
```

```
public class MessageImplementation extends UnicastRemoteObject implements  
Message {
```

```
    public MessageImplementation() throws RemoteException {  
    }  
}
```

```
@Override
```

```
    public double sum(double num1, double num2) throws RemoteException {  
        return num1 + num2;  
    }  
}
```

```
@Override
```

```
    public double difference(double num1, double num2) throws  
RemoteException {
```



```
        return num1 + num2;
    }
}
```

//Server.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Server {
    private void startServer() {
        try {
            // create registry on port 1099
            Registry registry = LocateRegistry.createRegistry(1099);

            // create a new service named myMessage
            registry.rebind("myMessage", new MessageImplementation());
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        Server main = new Server();
        main.startServer();
    }
}
```

```
    }  
}
```

//Client.java

```
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
public class Client {  
  
    private void doTest() {  
        try {  
            // register to localhost port 1099  
            Registry myRegistry =  
LocateRegistry.getRegistry("127.0.0.1", 1099);  
  
            // search for myMessage service  
            Message implementation = (Message)  
myRegistry.lookup("myMessage");  
  
            // call server's method  
            double s = implementation.sum(1, 2);  
            double d = implementation.difference(3, 4);  
  
            System.out.println("Sum:" + s + "\tDifference:" + d);  
        }  
        catch (Exception e) {
```

```

        e.printStackTrace();
    }
}

public static void main(String[] args) {
    Client main = new Client();
    main.doTest();
}
}

```

6. How java beans are different from java classes? Explain bean writing process.

Answer:

Refer to TU Exam Solution 2070.

7. Write a java program using TCP that enables chatting between client and server.

Answer:

Here we create two classes ServerChat and ClientChat that utilize ServerSocket and Socket classes. Sockets use the TCP/IP protocol by default. Below is the code that performs the actual chatting:

```

//ServerChat.java

import java.io.*;

import java.net.*;

import java.util.Scanner;

```

```

public class ServerChat {

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

        //create a server socket

        ServerSocket serverSocket = new ServerSocket(8000);

        //listen for client request

        Socket socket = serverSocket.accept();

        //create data input and output streams

        DataInputStream inputFromClient = new
DataInputStream(socket.getInputStream());

        DataOutputStream outputToClient = new
DataOutputStream(socket.getOutputStream());

        Scanner sc = new Scanner(System.in);

        String msg;

        while (true) {

            msg = inputFromClient.readUTF();

            System.out.println("Client says:" + msg);

            System.out.println("(From Server) Input message to client:");

            msg = sc.nextLine();

            outputToClient.writeUTF(msg);

        }

    }
}

```

```
//ClientChat.java

import java.io.*;

import java.net.Socket;

import java.util.Scanner;

public class ClientChat {

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

        //create a socket to connect to the server

        Socket socket = new Socket("localhost", 8000);

        //create an input stream to retrieve data from the server

        DataInputStream fromServer = new DataInputStream(socket.getInputStream());

        //create an output stream to send data to the server

        DataOutputStream toServer = new
DataOutputStream(socket.getOutputStream());

        Scanner sc = new Scanner(System.in);

        String msg;

        while (true) {

            System.out.println("(From Client)Input message to server:");

            msg = sc.nextLine();

            toServer.writeUTF(msg);
```

```

        msg = fromServer.readUTF();

        System.out.println("Server:" + msg);

    }

}

}

```

8. What is internal frame? Write a program that displays an internal frame within some parent frame. Answer:

Internal frame is a frame that is inside another frame. The `JInternalFrame` class is used to display a `JFrame`-like window within another window. Usually, we add internal frames to a desktop pane. The desktop pane, in turn, might be used as the content pane of a `JFrame`. The desktop pane is an instance of `JDesktopPane`, which is a subclass of `JLayeredPane` that has added API for managing multiple overlapping internal frames.

In the program below, we are just displaying five frames inside another frame.

```

//MyInternalFrame.java

import javax.swing.*.*;

public class MyInternalFrame extends JFrame {

    public MyInternalFrame() {

        for(int i=0; i<5; i++) {

            JInternalFrame frame      = new JInternalFrame("Internal Frame
" + i);

            frame.setLocation(i*50+10, i*50+10);

            frame.setSize(200, 150);

            this.add(frame);

            frame.setVisible(true);

```

```

    }

    setVisible(true);

    setSize(800, 600);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    public static void main(String[] args) {

        new MyInternalFrame();

    }

}

```

Section B

Attempt all Questions. [10×2=20]

9. Write a program to design an interface containing fields User ID, Password and Account type, and buttons login, cancel, edit by mixing border layout and flow layout. Add events handling to the button login and cancel such that clicking in login checks for matching user id and password in the database and opens another window if login is successful and displays appropriate message if login is not successful. Clicking in cancel terminates our program.

Answer:

Let us suppose we have a database named user and a table named user_table with fields user_id and password. Now the complete code for the above task is given below:

```

//UserInterfaceDesign.java

import java.awt.*;

import java.awt.event.*;

import java.sql.*;

import javax.swing.*;

```

```
public class UserInterfaceDesign extends JFrame {

    JLabel user_id_label = new JLabel("User ID:");

    JTextField user_id_field = new JTextField(10);


    JLabel password_label = new JLabel("Password:");

    JPasswordField password_field = new JPasswordField(10);


    JLabel account_type_label = new JLabel("Account Type:");

    JRadioButton account_type_radio_button1 = new JRadioButton("Savings");

    JRadioButton account_type_radio_button2 = new JRadioButton("Current");

    ButtonGroup account_type_button_group = new ButtonGroup();


    JButton login_button = new JButton("LogIn");

    JButton cancel_button = new JButton("Cancel");

    JButton edit_button = new JButton("Edit");


    public UserInterfaceDesign() {

        account_type_button_group.add(account_type_radio_button1);

        account_type_button_group.add(account_type_radio_button2);


        JPanel p1 = new JPanel(new FlowLayout(10));

        p1.add(user_id_label);

        p1.add(user_id_field);

        p1.add(password_label);
```



```
p1.add(password_field);  
p1.add(account_type_label);  
p1.add(account_type_radio_button1);  
p1.add(account_type_radio_button2);
```

```
add(p1, BorderLayout.NORTH);  
  
JPanel p2 = new JPanel(new FlowLayout(50));  
p2.add(login_button);  
p2.add(cancel_button);  
p2.add(edit_button);
```

```
add(p2, BorderLayout.SOUTH);
```

```
login_button.addActionListener(new ActionListener() {
```

```
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        try{
```

```
            Class.forName("com.mysql.jdbc.Driver");
```

```
            Connection c =  
            DriverManager.getConnection("jdbc:mysql://localhost:3306/user", "root", null);
```

```
            Statement s = c.createStatement();
```

```
            int id = Integer.parseInt(user_id_field.getText());
```

```
            char [] pass = password_field.getPassword();
```

```
            String passwd = String.valueOf(pass);
```

```

        System.out.println("ID:"+id+"\tPassword:"+passwd);

        String sql = "SELECT * FROM user_table WHERE user_id = "+id+" AND
password='"+passwd+"'";

        ResultSet r = s.executeQuery(sql);

        if(r.first()) {

            JOptionPane.showMessageDialog(new JFrame(), "Log In
Successful!!!");

        }

        else {

            JOptionPane.showMessageDialog(null, "Incorrect
UserID/Password");

        }

    }

    catch (Exception ex){

        System.out.println("Exception occurred");

    }

}

});

cancel_button.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

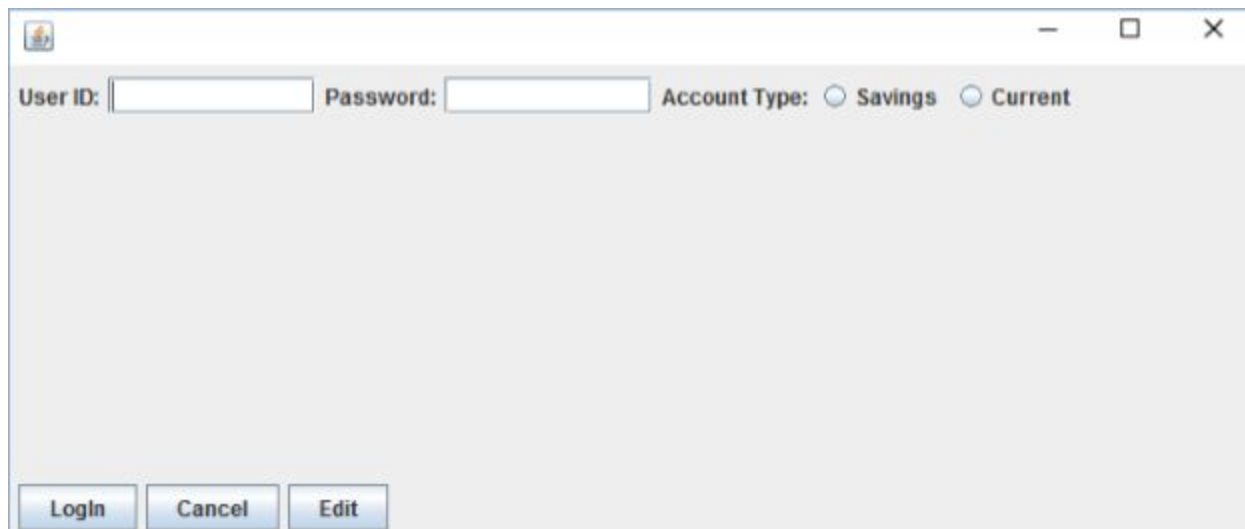
    System.exit(0);

}

```

```
});  
setSize(700,300);  
setVisible(true);  
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
  
public static void main(String []args) {  
    new UserInterfaceDesign();  
}  
}
```

A snapshot of the above program is:

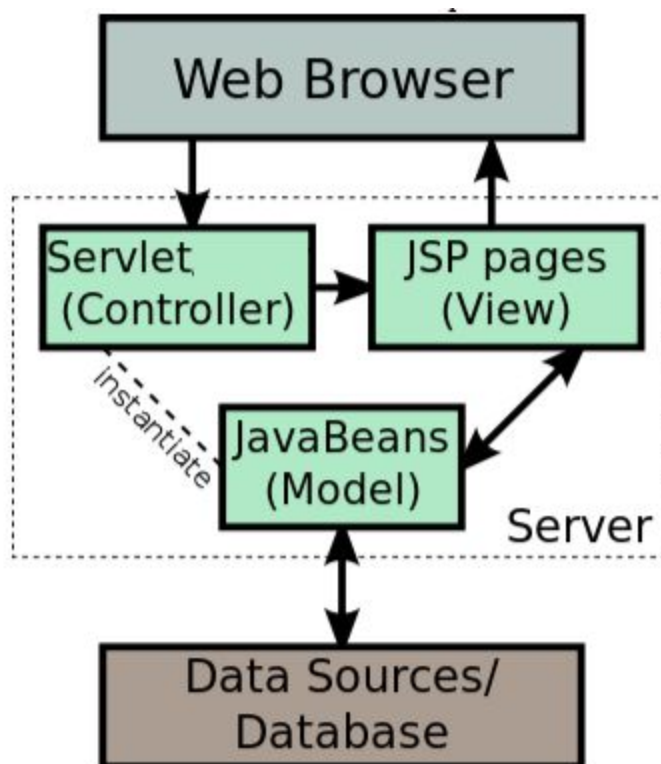


10. Compare and contrast between Servlets and Java Server Pages? How web server responds to servlets and JSP page? Write down sample example handling HTTP request and response using servlets.

Answer:

JavaServer Pages (JSP) is a technology that helps software developers create dynamically generated web pages normally based on HTML documents. Released in 1999 by Sun Microsystems, JSP is similar to PHP, but it uses the Java programming language. Servlets are small Java programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server. Two packages contain the classes and interfaces that are required to build servlets. These are `javax.servlet` and `javax.servlet.http`. They constitute the Servlet API. Actually, both servlets and JSP are server side components. In servlets, to add HTML components, we write the HTML tags in `out.println()` method of a servlet. In JSPs, to add server side logics, we write the codes in a scriptlet (`<% ... %>`).

In web Model-View-Controller design, normally JSP pages are used as view components and servlets are used as controller components with Java Beans as the model.



If the request is for a static web page, then whenever a browser generates an HTTP request to a web server, the web server maps this request to a specific file and the file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multipurpose Internet Mail Extensions (MIME) are used for this purpose. For example, ordinary ASCII text has a MIME type of text/plain. The HTML source code of a web has a MIME type of text/html.

If the request is for dynamic content, then whenever a browser generates an HTTP request to a web server, the web server maps this request to a particular servlet. The servlet then processes the HTTP request: it can read data available with the HTTP request, communicate with the model and can also formulate an HTTP response whereby it may redirect to another JSP page for the client.

The following program uses index.jsp page and ServletTest.java servlet. The index.jsp file contains a form whose contents are sent to a servlet by GET method. Here, the servlet just catches the form's contents and displays it, however the servlet may have taken the form's contents, saved to database and then redirected to another view.

```
//index.jsp

<!DOCTYPE html>

<html>

    <head>

        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">

        <title>JSP Page</title>

    </head>

    <body>

        <form action="ServletTest">

            Username:<input type="text" name="user"/>

            Password:<input type="password" name="pass"/>

            <input type="submit" value="Submit"/>

        </form>

    </body>

</html>
```

```
        </form>
    </body>
</html>
```

```
//ServletTest.java
```

```
    import java.io.*;
    import javax.servlet.*;
    import javax.servlet.http.*;

    public class ServletTest extends HttpServlet {

        @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

            response.setContentType("text/html;charset=UTF-8");

            PrintWriter out = response.getWriter();

            out.println("<h1>Username is: " + request.getParameter("user")
+ "</h1>");

            out.println("<h1>Password is: " + request.getParameter("pass")
+ "</h1>");

        }
    }
```

Some other important questions (1)

Explain GridBagLayout with an example.

Answer:

GridBagLayout is one of the most flexible — and complex — layout managers the Java platform provides. A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns. Not all rows necessarily have the same height. Similarly, not all columns necessarily have the same width. Essentially, GridBagLayout places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be. The way the program specifies the size and position characteristics of its components is by specifying constraints for each component using GridBagConstraints object, as demonstrated below:

```
//GridBagLayout.java

import java.awt.*;

import javax.swing.*;

public class GridBagLayoutTest extends JFrame {

    public GridBagLayoutTest() {

        GridBagLayout gridbag = new GridBagLayout();

        GridBagConstraints c = new GridBagConstraints();

        setLayout(gridbag);

        c.fill = GridBagConstraints.BOTH;

        c.weightx = 1.0;

        Button b1 = new Button("Button 1");

        gridbag.setConstraints(b1, c);
```

```
Button b2 = new Button("Button 2");  
gridbag.setConstraints(b2, c);
```

```
c.gridwidth = GridBagConstraints.REMAINDER;  
Button b3 = new Button("Button 3");  
gridbag.setConstraints(b3, c);
```

```
Button b4 = new Button("Button 4");  
gridbag.setConstraints(b4, c);
```

```
c.gridwidth = GridBagConstraints.RELATIVE;  
Button b5 = new Button("Button 5");  
gridbag.setConstraints(b5, c);
```

```
add(b1);  
add(b2);  
add(b3);  
add(b4);  
add(b5);
```

```
setSize(300, 100);
```

```
setVisible(true);  
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



```

    }

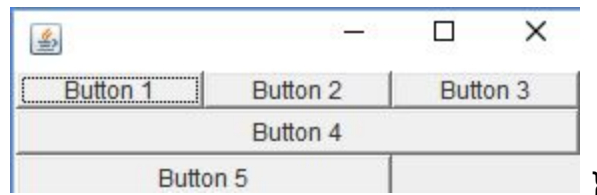
    public static void main(String args[]) {

        new GridBagLayoutTest();

    }

```

A snapshot of the above program is:



(2) What are prepared statements? Explain with example.

Answer:

In database management systems, a prepared statement or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with SQL statements such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution. The typical workflow of using a prepared statement is as follows:

(a) Prepare:

The statement template is created by the application and sent to the database management system (DBMS). Certain values are left unspecified, called parameters, placeholders or bind variables (labelled "?" below): INSERT INTO PRODUCT (name, price) VALUES (?, ?)

(b) The DBMS parses, compiles, and performs query optimization on the statement template, and stores the result without executing it.

(c) Execute:

At a later time, the application supplies (or binds) values for the parameters and the DBMS executes the statement (possibly returning a result). The application may execute the statement as many times as it wants with different values. In this example, it might supply 'Bread' for the first parameter and '50.00' for the second parameter.

As compared to executing SQL statements directly, prepared statements offer two main advantages:

(i) The overhead of compiling and optimizing the statement is incurred only once, although the

statement is executed multiple times.

(ii) Prepared statements are resilient against SQL injection, because parameter values, which

are transmitted later, need not be correctly escaped.

Example: This example uses Java and the JDBC API:

```
java.sql.PreparedStatement stmt = connection.prepareStatement(  
    "SELECT * FROM users WHERE USERNAME = ? AND ROOM =  
    ?");  
  
    stmt.setString(1, username);  
  
    stmt.setInt(2, roomNumber);  
  
    stmt.executeQuery();
```

The Java PreparedStatement provides "setters" (setInt(int), setString(String), setDouble(double), etc.) for all major built-in data types which are used to set the values of the parameters.

(3) Explain inheritance and its types with example.

Answer:

Tribhuvan University
Institute of Science and Technology
B.Sc. CSIT, 7th Semester
2069

Subject: Advance Java Programming

FM: 60 PM: 24

Course No.: CSC-403

Time: 3 Hrs.

**Candidates are required to give their answers in their own words as far as practicable.
(NEW COURSE)**

Section A

**Attempt any two questions.
(2*10=20)**

1. What are exceptions? Why is it important to handle exceptions? Discuss different keywords that are used to handle exception.

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

Importance of handling exceptions.

- First, it lets the user know, in a relatively friendly manner, that something has gone wrong and that they should contact the technical support department or that someone from tech support has been notified. As we all know there's a HUGE difference between receiving a rather nasty, tech riddled notice that says something like "Object not set to reference of an object" etc. ... and receiving a nice popup type window that says "There has been an issue. Please contact the helpdesk".
- Second it allows the programmer to put in some niceties to aid in the debugging of issues. For instance ... in my code, I typically write a custom error handler that takes in a number of parameters and spits back a nice, formatted message that can either be emailed to the helpdesk, stashed in an event log, written to a

log file etc.. The error message will contain as much info as I can cram in there to help me figure out what happened, stack traces, function parameters, database calls ... you name it. I like verbose error messages to help me figure out what actually happened. The user never has to see any of it, they get the nice, friendly message above, letting them know that someone can figure out what's going on.

There are five keywords used in Java for exception handling. They are:

- i. try
- ii. catch
- iii. throw
- iv. throws
- v. finally

The try block

The try block contains the code that might throw an exception. The try block contains at least one catch block or finally block.

The catch block

A catch block must be declared after try block. It contains the error handling code. A catch block executes if an exception generates in corresponding try block.

The throw keyword in Java

- The throw keyword in Java is used to explicitly throw our own exception.
- It can be used for both checked and unchecked exception.

The throws keyword in Java

- The throws keyword is generally used for handling checked exception.

- When you do not want to handle a program by try and catch block, it can be handled by throws.
- Without handling checked exceptions program can never be compiled.
- The throws keyword is always added after the method signature.

The finally block

- The code present in finally block will always be executed even if try block generates some exception.
- Finally block must be followed by try or catch block.
- It is used to execute some important code.
- Finally block executes after try and catch block

2. When the exception handling constructs throws and finally is important? Explain both of them with suitable example.)

Answer:

An exception is a problem that arises during the execution of a program. An exception can occur for many different reasons, including the following:

- A user has entered an invalid data
- A file that needs to be opened cannot be found
- A network connection has been lost in the middle of communications or

the JVM

has run out of memory

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner. There are three categories of exceptions in Java:

a) Checked exceptions:

A checked exception is an exception that is typically a user error or a problem that cannot be foreseen by the programmer. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot be simply ignored at the time of compilation.

b) Runtime exceptions:

A runtime exception is an exception that occurs where that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.

c) Errors:

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in our code because we can rarely do anything about an error. For example, if JVM is out of memory, an error will arise. They are also ignored at the time of compilation.

It is important to handle exceptions because otherwise the program would terminate abnormally whenever an exceptional condition occurs. Exception handling enables a program to deal with exceptional situations and continue its normal execution. There are five keywords in Java that are used to handle exceptions. They are

- a) try
- b) catch
- c) finally
- d) throws
- e) throw

The try and catch keywords are used in combination to handle exceptions. A try/catch block is placed around the code that might generate an exception. Exceptions are also objects of some class in Java derived from `java.lang.Throwable`. If an exception occurs in the try block, the catch block (or blocks) that follow the try block is checked and exception is handled there. The finally keyword is used to create a block of code that follows a try block. There may be a try/finally

block to handle exception in place of a try/catch block. However, a finally block of code always executes whether or not an exception has occurred. Normally, finally block is used to run any cleanup-type statements that we wish to execute no matter what happens in the try block. Thus we will normally have try/catch/finally blocks. If a method does not handle an exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature and is followed by class name(s). The throw keyword is used to throw an exception explicitly whenever an exceptional condition occurs. It is followed by only a single instance of the exception class we want to throw and is used inside method body.

The code example to show usage of the keywords for exception handling is given below:

```
import java.util.Scanner;

public class ExceptionHandlingTest {

    public static int quotient(int number1, int number2) throws ArithmeticException
    {

        if (number2 == 0) {

            throw new ArithmeticException("Divisor cannot be zero");

        }

        return number1 / number2;

    }

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        System.out.print("Enter two integers for division: ");

        int number1 = input.nextInt();

        int number2 = input.nextInt();

        int result;
```

```

try {
    result = quotient(number1, number2);
    System.out.println(number1 + " / " + number2 + " is " + result);
}
catch (ArithmeticException ex) {
    System.out.println(ex.getMessage());
}
finally {
    if(number2 == 0) {
        while(number2 == 0) {
            System.out.println("Execution continues ...");
            System.out.println("Enter non-zero divisor:");
            number2 = input.nextInt();
        }
        result = quotient(number1, number2);
        System.out.println(number1 + " / " + number2 + " is " + result);
    }
}
}
}

```

Here, when the divisor will be non-zero, the program will run normally i.e. the program will run its try block where it computes the quotient and then end in finally block. However when the divisor will be zero then the method quotient will throw an exception which is caught by the catch block and a message will be printed. After that the finally block will be executed where the divisor will be re-entered by the user till a non-zero value is given and the quotient will be calculated again.

2. Write a program using components to add two numbers. Use text fields for inputs and output. Your program should display the result when the user presses a button. (10)

Answer:

```
package classPractise;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

class Add extends JFrame{
    private JLabel lblFirstNum;
    private JLabel lblSecondNum;
    private JLabel result;
    private JTextField firstNum;
    private JTextField secondNum;
    private JButton btnAdd;
    public Add(){
        setSize(400,300);
        setLayout(new FlowLayout());
        lblFirstNum=new JLabel("Enter First Number:");
        firstNum=new JTextField(10);
        add(lblFirstNum);
        add(firstNum);
        lblSecondNum=new JLabel("Enter First Number:");
```

```

secondNum=new JTextField(10);

add(lblSecondNum);

add(secondNum);

btnAdd= new JButton("Add");

add(btnAdd);

result=new JLabel();

add(result);

btnAdd.addActionListener(new ActionListener(){

    @Override

    public void actionPerformed(ActionEvent e) {

        int a,b,sum;

        a=Integer.parseInt(firstNum.getText());

        b=Integer.parseInt(secondNum.getText());

        sum=a+b;

        result.setText("Sum="+sum);

    }

});

}

}

public class AddNum {

    public static void main(String[] args) {

        Add add = new Add();

        add.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        add.setVisible(true);

    }

}

```

}

3. What is Java bean? Differentiate it with Java class. Discuss bean writing process with suitable examples. (2+2+6)

Answer:

Refer to Solution of TU Exam Question 2070.

Steps for writing a bean are:

1. create a directory for the new Bean
2. create the java source files
3. compile the source file
4. Create manifest file
5. Generate a JAR file
6. Start the BDK
7. Test the bean

Section B

Attempt any eight questions.

(8*5=40)

4. Discuss the use of interfaces to achieve multiple inheritance.

Answer:

Refer to Solution of TU Exam Question 2070.

5. Discuss the use of multithreading with suitable example.

Answer:

Refer to Solution of TU Exam Question 2070.

6. What is JDBC? How do you execute SQL statements in JDBC?

Answer:

Look at Q.no 5 2071

7. Discuss grid layout with suitable example.

Answer:

The GridLayout class is a layout manager that lays out a container's components in a rectangular grid. The container is divided into equal-sized rectangles, and one component is placed in each rectangle. Thus we can say that the components will be arranged in the form of rows and columns like a table with equal sized cells. The constructors for the GridLayout class are:

- a) GridLayout()
- b) GridLayout(int rows, int cols)
- c) GridLayout(int rows, int cols, int hgap, int vgap)

A suitable example of using GridLayout class is given below:

```
import java.awt.GridLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

public class GridLayoutTest extends JFrame {

    public GridLayoutTest() {

        setLayout(new GridLayout(2,3,10,20));

        JButton one = new JButton("1");

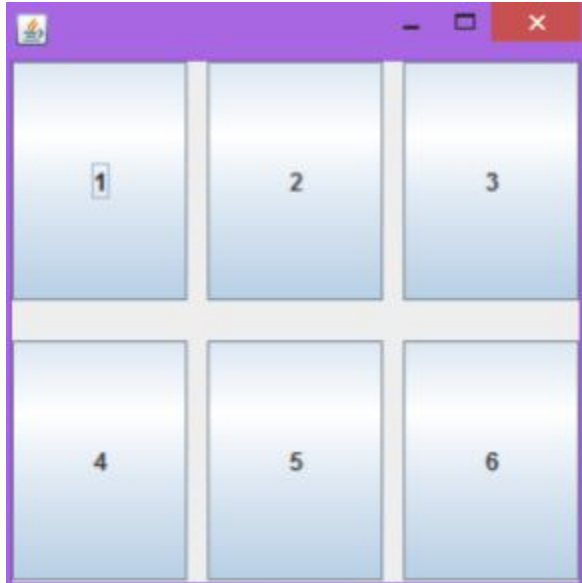
        JButton two = new JButton("2");

        JButton three = new JButton("3");
```

```
        JButton four = new JButton("4");
        JButton five = new JButton("5");
        JButton six = new JButton("6");
        add(one);
        add(two);
        add(three);
        add(four);
        add(five);
        add(six);
        setSize(300, 300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

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

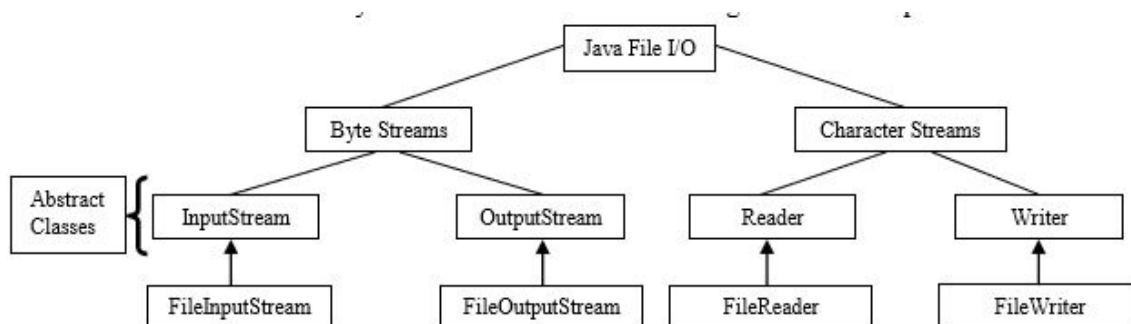
The output of the above program will be as:



8. Discuss any five classes to handle files in java.

Answer:

File handling implies performing I/O on files. In Java, file I/O is performed through streams. A stream means flow of data irrespective of the device. There are two kinds of streams: byte streams and character streams. `InputStream` and `OutputStream` are two abstract classes in Java that are provided to read/write in byte streams i.e. they perform read/write as a sequence of 8-bit bytes. `FileInputStream` and `FileOutputStream` are two concrete classes derived from `InputStream` and `OutputStream` that are used to perform the actual read/write on files. Similarly, there are `Reader` and `Writer` named abstract classes in Java that are provided to read/write in character streams i.e. they perform read/write in 16-bit characters. `FileReader` and `FileWriter` are two concrete classes derived from `Reader` and `Writer` that are used to perform the actual read/write on files. Further there is a separate class named `RandomAccessFile` which allows file to be read/written randomly. The classes used for file handling in Java can be pictured as:



Some classes to handle files in Java are discussed below:

(i) **FileInputStream:**

The **FileInputStream** class creates an **InputStream** that we can use to read bytes from a file. Its two most common constructors are shown here:

`FileInputStream(String filepath)` throws

`FileNotFoundException` `FileInputStream(File fileObj)` throws `FileNotFoundException` Here, `filepath` is the full path name of a file, and `fileObj` is a `File` object that describes the file.

Code example:

```
import java.io.*;

public class FileInputStreamTest {

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

        FileInputStream fin = new FileInputStream("welcome.txt");

        int i = 0;

        while ((i = fin.read()) != -1) {

            System.out.print((char)i);

        }

    }

}
```

(ii) **FileOutputStream:**

FileOutputStream creates an **OutputStream** that we can use to write bytes to a file. Its most commonly used constructors are: `public FileOutputStream(String name)` throws `FileNotFoundException` `public FileOutputStream(String name, boolean append)` throws `FileNotFoundException` `public FileOutputStream(File file)` throws `FileNotFoundException` `public FileOutputStream(File file, boolean append)` throws `FileNotFoundException`

Here, filePath is the full path name of a file, and fileObj is a File object that describes the file. If append is true, the file is opened in append mode.

Code example:

```
import java.io.*;

class FileOutputStreamDemo
{
    public static void main(String args[])
    {
        try {
            FileOutputStream fout = new FileOutputStream("hello.txt");

            String s = "Hello World!!!";

            byte b[]=s.getBytes();

            fout.write(b);

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

(iii) FileReader:

The **FileReader** class creates a **Reader** that you can use to read the contents of a file. Its two most commonly used constructors are: public FileReader(String fileName) throws

FileNotFoundException public FileReader(File file) throws FileNotFoundException Here, filePath is the full path name of a file, and fileObj is a **File** object that describes the file.

Code example:

```
import java.io.FileReader;

import java.io.IOException;

public class FileReaderDemo {

    public static void main(String[] args) {

        try {

            FileReader fr = new FileReader("test.txt");

            int c;

            while ((c = fr.read()) != -1) {

                System.out.print((char) c);

                }

            }

        catch (IOException ex) {

            ex.printStackTrace();

            }

    }

}
```

(iv) FileWriter:

FileWriter creates a **Writer** that you can use to write to a file. Its most commonly used constructors are:

```
public FileWriter(String fileName) throws IOException
```

```
public FileWriter(String fileName, boolean append) throws IOException
```

```
public FileWriter(File file) throws IOException
```

```
public FileWriter(File file, boolean append) throws IOException
```

Here, **filePath** is the full path name of a file, and **fileObj** is a File object that describes **the file**. If **append** is true, then output **is appended** to the end of the file.

Code example:

```
import java.io.FileWriter;
```

```
import java.io.IOException;
```

```
public class FileWriterDemo {
```

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

```
        String src = "Welcome to Java World!!!";
```

```
        char buffer[] = src.toCharArray();
```

```
        try {
```

```
            FileWriter f0 = new FileWriter("file0.txt");
```

```
            FileWriter f1 = new FileWriter("file1.txt");
```

```
            for (int i = 0; i < buffer.length; i = i + 2) {
```

```
                f0.write(buffer[i]);
```

```
            }
```

```
            f1.write(buffer);
```

```

        f0.close();

        f1.close();

    } catch (IOException e) {

        System.out.println(e);

    }

}

```

This program creates two files named file0.txt and file1.txt. The first file contains every alternate characters of the buffer array and the second file contains the whole buffer.

(v) **RandomAccessFile:**

RandomAccessFile encapsulates a random-access file. It is not derived from **InputStream** or **OutputStream**. Instead, it implements the interfaces **DataInput** and **DataOutput**, which defines the basic I/O methods. It also supports positioning requests i.e., we can position the file pointer within the file. It has these two constructors:

RandomAccessFile(File fileObj, String access) throws FileNotFoundException

RandomAccessFile(String filename, String access) throws FileNotFoundException

In the first form, fileObj specifies the name of the file to open as a **File** object. In the second form, the name of the file is passed in filename. In both cases, access determines what type of file access is permitted. If it is “r”, then the file can be read, but not written. If it is “rw”, then the file is opened in read-write mode. The most important method of this class is **seek()**. This method is used to set the current position of the file pointer within the file: void seek(long newPos) throws IOException Here, newPos specifies the new position, in bytes, of the file pointer from the beginning of the file. After a call to **seek()**, the next read or write operation will occur at the new file position.

Code example:

```
import java.io.IOException;
```

```

import java.io.RandomAccessFile;

public class TestRandomAccessFile {

    public static void main(String []args)

    {

        try

        {

            RandomAccessFile r = new RandomAccessFile("abc.dat", "rw");

            r.writeChar('a'); //2 bytes

            r.writeInt(555); //4 bytes

            r.writeDouble(3.14); // 8 bytes

            String s = "Random";

            r.writeBytes(s); //6 bytes

            r.seek(0);

            System.out.println(r.readChar());

            System.out.println(r.readInt());

            System.out.println(r.readDouble());


            r.seek(2);

            System.out.println(r.readInt());

            r.close();

        }

        catch(IOException e)

        {

            System.out.println(e);

        }

    }

}

```

```
}  
  
}
```

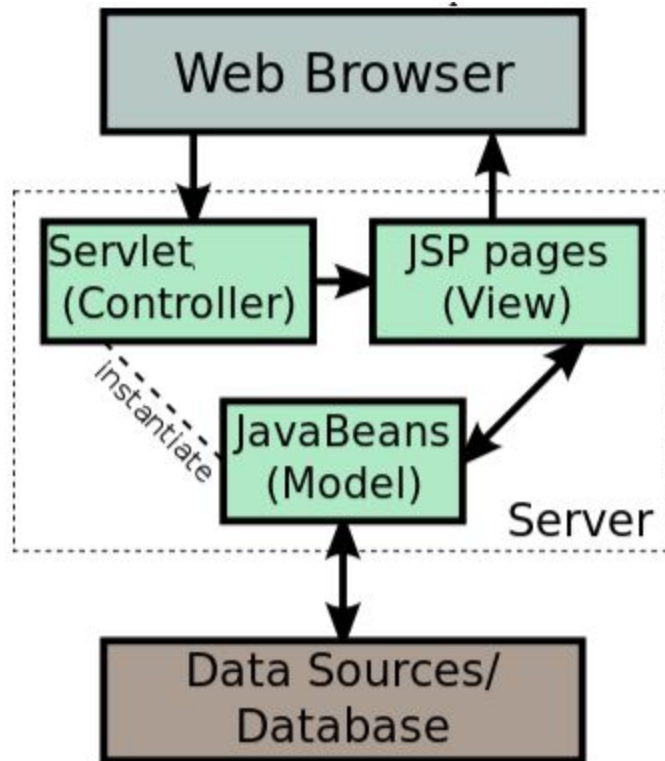
9. What is JSP? Differentiate it with servlet.

Answer:

JavaServer Pages (JSP) is a technology that helps software developers create dynamically generated web pages normally based on HTML documents. Released in 1999 by Sun Microsystems, JSP is similar to PHP, but it uses the Java programming language. To deploy and run JavaServer Pages, a compatible web server with a servlet container such as Apache Tomcat is required.

Servlets are small Java programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server. Two packages contain the classes and interfaces that are required to build servlets. These are **javax.servlet** and **javax.servlet.http**. They constitute the Servlet API.

JSP is normally used as the view component of a server side model–view–controller design, normally with JavaBeans as the model and Java servlets as the controller.



Key Differences :

- **Servlet** is html in java whereas **JSP** is java in html.
- Servlets run faster compared to JSP
- JSP can be compiled into Java Servlets
- It's easier to code in JSP than in Java Servlets
- JSP is a webpage scripting language that can generate dynamic content while Servlets are Java programs that are already compiled which also creates dynamic web content
- In MVC, **jsp** acts as a view and **servlet** acts as a controller.
- JSP are generally preferred when there is not much processing of data required. But servlets are best for use when there is more processing and manipulation involved.
- The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans. There is no such facility in servlets.

- We can achieve functionality of JSP at client side by running JavaScript at client side. There are no such methods for servlets.
- A servlet is like any other Java class. You put HTML into print statements like you use `System.out` or how JavaScript uses `document.write`. A JSP technically gets converted to a servlet but it looks more like PHP files where you embed the Java into HTML.

10. What is UDP socket? Differentiate it with TCP socket.

Answer:

Sockets are the endpoints of logical connections between two hosts and can be used to send and receive data. Java supports stream sockets and datagram sockets. Stream sockets use TCP (Transmission Control Protocol) for data transport, thus they are also called TCP sockets. Datagram sockets use UDP (User Datagram Protocol) for data transport, thus they are also called UDP sockets. Since TCP can detect lost transports and resubmit them, the transports are lossless and reliable. UDP, in contrast, cannot guarantee lossless transport and so is unreliable.

9

When using a TCP socket, the tcp stack takes care of the data being sent to the network and being delivered to the receiver, retransmitting it until acknowledged by the receiver. TCP also takes care of flow control, i.e. transmitting the data at a suitable rate for the network connection and receiver. Last, TCP ensures that the receiver gets the data exactly once and in the correct order.

With UDP, the programmer manages the transmission to network directly, and has to take care of lost and out-of order packets as well as flow control and fragmenting data to packets that can be transmitted over the network connection.

Last, as udp and tcp are different protocols, they require different settings in firewalls to allow passing through to the server if the server is behind a firewall. Also you can't send data from a connected (or non-connected) UDP socket to a TCP socket or vice versa.

Connecting the UDP socket just means that send() and recv() can be used to send data to and from the connected address, but it is still UDP data being sent and received so all of the above differences apply. Calling connect() on a UDP socket is something you would do if you do only point-to-point communications using the socket.

TCP/IP Vs UDP :

1. Connection oriented & other is Connection less
2. Will have Acknowledgment & other one don't
3. Performance slow & other one fast
4. More secure & other is not much

11. How can you handle events using adapter classes? Discuss.

Answer:

Adapter classes such as KeyAdapter, MouseAdapter, MouseMotionAdapter, WindowAdapter, etc provides the default (generally empty) implementation of all methods for many of the event listener interfaces such as KeyListener, MouseListener, MouseMotionListener, WindowListener, etc. Adapter classes are very useful when you want to process only few of the events that are handled by a particular event listener interface. For example, the KeyListener interface contains three methods KeyPressed(), KeyReleased() and KeyTyped(). If we implement this interface, we have to give implementation of all these three methods. However, by using the KeyAdapter class, we can override only the method that is needed. This can be shown in the code example given below. Here we are writing a GUI program that accepts user input in a text field and displays the key typed in another text field when the pressed key is released. The first program uses KeyListener interface and the second program uses KeyAdapter class.

First program

```
//KeyListenerTest.java

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class KeyListenerTest extends JFrame {

    JTextField t1 = new JTextField(10);
    JTextField t2 = new JTextField(10);

    public KeyListenerTest() {
        setLayout(new FlowLayout());
        setSize(200, 200);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(t1);
        add(t2);
        t2.setEditable(false);

        t1.addKeyListener(new KeyListener() {

            @Override
            public void keyTyped(KeyEvent e) { }
```

```

        @Override
        public void keyPressed(KeyEvent e) { }

        @Override
        public void keyReleased(KeyEvent e) {

            String copy = t1.getText();

            t2.setText(copy);

        }
    });
}

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

```

Second program

```

//KeyAdapterTest.java

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

public class KeyAdapterTest extends JFrame {

    JTextField t1 = new JTextField(10);

```

```
TextField t2 = new TextField(10);
```

```
public KeyAdapterTest() {  
    setLayout(new FlowLayout());  
    setSize(200, 200);  
    setVisible(true);  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    add(t1);  
    add(t2);  
    t2.setEditable(false);  
  
    t1.addKeyListener(new KeyAdapter() {  
  
        @Override  
        public void keyReleased(KeyEvent e) {  
  
            String copy = t1.getText();  
            t2.setText(copy);  
        }  
    });  
}
```

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

```
    new KeyAdapterTest();
```

```
}  
  
}
```

12. What is RMI? Discuss architecture of RMI in detail.

Answer:

Refer to Solution of TU Exam Question 2070.

13. Write a simple JSP program to display "Kathmandu, Nepal" 10 times.

Answer:

```
<!DOCTYPE html>  
  
<html>  
  
  <head>  
  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
  
    <title>A simple JSP program</title> </head>  
  
</head>  
  
<body>  
  
  <h1>Displaying "Kathmandu, Nepal" 10 times!</h1>  
  
  <table>  
  
    <%  
  
      for(int i=1; i<=10; i++) {  
  
        %>  
  
        <tr><td>Kathmandu, Nepal</td></tr>  
  
        <% } %>  
  
  </table>  
  
</body>
```

</html>

Tribhuvan University
Institute of Science and Technology
B.Sc. CSIT, 7th Semester
2070

Subject: Advance Java Programming

FM: 60 PM: 24

Course No.: CSC-403

Time: 3 Hrs.

**Candidates are required to give their answers in their own words as far as practicable.
(NEW COURSE)**

Section A

**Attempt any two questions.
(2*10=20)**

- 1. What is interface? How can you use the concepts of interface to achieve multiple inheritances? Discuss with suitable example.
(2+2+6)**

Answer:

An interface in the Java programming is an abstract type that is used to specify a boundary that classes must implement. It contains only method signatures and static final variable declarations. It is declared using the keyword "interface". A class that implements the interface must define all of the interface's methods. An example of interface declaration is:

```
interface MyInterface {  
    static final int var = 5;  
    public void myMethod();  
}
```

In Java, there is no multiple inheritance. This is because Java tries to eliminate ambiguous things. So we cannot extend more than one class to get multiple inheritance. However, by using the concept of interface, we can achieve multiple inheritance whenever required. It can be done

in two ways: (a) by implementing more than one interfaces (b) by extending one class and implementing another interface.

Now, we give a suitable example of multiple inheritance in Java utilizing the second method. Here, we have an interface AcademicActivities and a class ExtraCurricularActivities. The interface is implemented and the class is extended by another class named Result to find out the student's total performance.

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

```
interface AcademicActivities
```

```
{
```

```
    float getAcademicMarks();
```

```
    void setAcademicMarks(float a);
```

```
}
```

```
class ExtraCurricularActivities
```

```
{
```

```
    float extra_curricular_marks;
```

```
    float getExtraCurricularMarks(){
```

```
        return extra_curricular_marks;
```

```
    }
```

```
    void setExtraCurricularMarks(float e) {
```

```
        extra_curricular_marks = e;
```

```
    }
```

```
}
```

```
public class Result extends ExtraCurricularActivities implements AcademicActivities
```

```

{
    float academic_marks;

    float total;

    @Override
        public float getAcademicMarks() {
            return academic_marks;
        }

    @Override
        public void setAcademicMarks(float a) {
            academic_marks = a;
        }

    void displayResult() {
        total = (academic_marks + extra_curricular_marks)/2;
        System.out.println("Student total:"+total);
    }

    public static void main(String[] args) {
        float a,e;

        Result r = new Result();

        System.out.println("Enter marks in academic activities:");

        Scanner sc = new Scanner(System.in);

        a = sc.nextFloat();

```

```

        r.setAcademicMarks(a);

        System.out.println("Enter marks in extra curricular activities:");

        e = sc.nextFloat();

        r.setExtraCurricularMarks(e);

        r.displayResult();

    }

}

```

2. Write a program using swing components to multiply two numbers. Use text fields for inputs and output. Your program should display the result when the user presses a button. (10)

Answer:

The code for the Java program is given below:

```

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

public class MultiplyUsingSwing extends JFrame implements ActionListener {

    JLabel l1 = new JLabel("First No:");

    JTextField t1 = new JTextField(10);

    JLabel l2 = new JLabel("Second No:");

    JTextField t2 = new JTextField(10);

    JTextField t3 = new JTextField(10);

    JButton b = new JButton("Add");

    String fno = "";

    String sno = "";

```



```

public MultiplyUsingSwing() {
    setLayout(new FlowLayout());
    setSize(250, 250);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    add(l1);
    add(t1);
    add(l2);
    add(t2);
    t3.setEditable(false);
    add(t3);
    add(b);
    b.addActionListener(this);
}

```

@Override

```

public void actionPerformed(ActionEvent e) {
    int f = Integer.parseInt(t1.getText());
    int s = Integer.parseInt(t2.getText());
    int r = f + s;
    t3.setText("" + r);
}

```

```

public static void main(String[] args) {

```

```
new MultiplyUsingSwing();  
}}
```

3. what is RMI ? How can you use RMI to develop a program that runs in different machine? Discuss with suitable example.

3) How can you use RMI to develop a program that runs in different machine? Discuss with suitable example.
→ There are six steps to write the RMI program.

i) Create the remote interface
We extend the Remote interface and declare the RemoteException with all the methods of the remote interface.

```
import java.rmi.*;  
public interface Adder extends Remote  
{  
    public int add(int x, int y) throws RemoteException;  
}
```

ii) Provide the implementation of the Remote interface
For providing the implementation of the Remote interface...

need to either extend the `UnicastRemoteObject` class, or use the `exportObject()` method of the `UnicastRemoteObject` class

```
import java.rmi.*;
```

```
import java.rmi.server.*;
```

```
public class AdderRemote extends UnicastRemoteObject  
implements Adder
```

```
{
```

```
    AdderRemote() throws RemoteException
```

```
{
```

```
    super();
```

```
}
```

```
    public int add(int x, int y)
```

```
{
```

```
        return x+y;
```

```
}
```

```
}
```

iii) Create the stub and skeleton objects using the `rmic` Tool
The `rmic` tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

iv) Start the Registry Service by the `rmi registry` Tool

The port number (default) that we use is 5000 .

```
rmi registry 5000
```

v) Create the and Run the Server Application

Now rmi services need to be hosted in a server process.

```
import java.rmi.*;
```

```
import java.rmi.registry.*;
```

```
public class MyServer
```

```
{
```

```
try {
```

```
    Adder add adderobject = new AdderRemote();
```

```
    Naming.rebind("rmi://localhost:5000/rmiserver", adderobject);
```

```
    }
```

```
catch (Exception e)
```

```
{
```

```
    System.out.println(e);
```

```
    }
```

```
    }
```

```
}
```

vi) Create and run the client application

At the client we are getting the adder object by the lookup() method of the Naming class and invoking the method on this object.

```
import java.rmi.*;
public class MyClient {
    public static void main(String args[]) throws Exception
    {
        Adder a = (Adder) Naming.lookup("rmi://localhost:5000/as");
        int s = a.add(34, 4);
        System.out.println("sum = " + s);
    }
}
```

Section B

Attempt any eight questions.

(8*5=40)

4. What is JDBC? How do you execute SQL queries in JDBC?

(2+3) Answer:

JDBC is a Java database connectivity technology from Oracle Corporation. This technology is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases such as MySQL. The JDBC classes are contained in the java.sql package.

To work with database from Java programs, we need to first load the appropriate driver and then get a connection to the database via JDBC APIs. The JDBC connection supports creating and executing SQL statements. These may be update statements such as INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. When SQL queries are

executed in JDBC, they return a JDBC result set and we manipulate this result set to present a tabular view to the user. A complete example showing the execution of SQL query in JDBC is given below:

```
import java.sql.*;

public class ExecuteSQLInJDBC {

    public static void main(String []args) {

        try{

            Class.forName("com.mysql.jdbc.Driver");

            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/library",
"root",

            "vertrigo");

            Statement stat = conn.createStatement();

            String sql = "select * from book";

            ResultSet rs = stat.executeQuery(sql);

            ResultSetMetaData rsmd = rs.getMetaData();

            int no_of_columns = rsmd.getColumnCount();

            for(int i=1; i<=no_of_columns;i++) {

                System.out.print(rsmd.getColumnName(i)+"\t");

            }

            while(rs.next()) {

                System.out.println();

                for(int i=1; i<=no_of_columns;i++)
```

```

        System.out.print(rs.getObject(i)+"\t\t");

    }

}

catch(Exception e){

    e.printStackTrace();

}

}

```

In this program we pre-suppose that we have a database named “library” and within that database we have a table named “book”. When the SELECT query is executed, we get a ResultSet object and by manipulating it, we have displayed the “book” table.

5. What is a Java bean? How is it different from Java class? (1+4)

Answer:

A Java bean is a Java class that should follow the following conventions:

- a. It should have a no argument constructor.
- b. It should be serializable i.e. it must implement the java.io.Serializable interface. Serializability ensures that the object’s state can be written into streams such as files (called serialization) and can be restored later into object as well (called deserialization).
- c. It should provide methods to set and get the values of the properties, known as getter and setter methods.

A Java bean is different from a Java class in the sense that a Java class definition does not have any restriction whereas a Java bean class definition has. Thus a Java bean class is just a Java class with the properties mentioned above. An example of a Java bean class is:

```
package bsccsit;
```

```
import java.io.Serializable;

public class JavaBeanClassEmployee implements Serializable {

    private int id;

    private String name;


    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Now, to access this Java bean class (from another package), we should use its getter and setter methods as:


```

package seventhsemester;

import bsccsit.JavaBeanClassEmployee;

public class Test {

    public static void main(String []args) {

        JavaBeanClassEmployee e = new JavaBeanClassEmployee();

        e.setId(1);

        e.setName("Lok");

        System.out.println(e.getId() + " " + e.getName());

    }

}

```

6. Write a simple Java program to read from and write to files.

(5)

Answer:

Let us suppose we have a file named “test.txt” in D-drive. Now we first read from this file character-by-character and later we write the contents of this file to another file say “testwrite.txt” in E-drive. For these tasks, we use the character stream classes namely FileReader and FileWriter. The code is given below:

```

import java.io.*;

public class FileReadWrite {

    public static void main(String []args) {

        try

        {

            FileReader fr = new FileReader("D:\\test.txt");

            FileWriter fw = new FileWriter("E:\\testwrite.txt");

            int c;

```

```

while((c=fr.read())!=-1) {
    fw.write(c);
    System.out.print((char)c);
}
fr.close();
fw.close();
}
catch (IOException ex)
{
    ex.printStackTrace();
}
}
}

```

7. Discuss different methods used in the life cycle of servlet.

(2+3) Answer:

A Java servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. To run a servlet, a web container must be used. One of the best known open source web container for servlet is Tomcat.

All servlets implement the Servlet interface, which defines its life-cycle methods. A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet during its life cycle:

- a. The servlet is initialized by calling the init() method.
- b. The servlet calls service() method to process a client's request.
- c. The servlet is terminated by calling the destroy() method.

d. Finally, the servlet is garbage collected by the garbage collector of the JVM.

The different methods used in the life cycle of servlet are:

(a) The `init()` method:

The `init()` method is designed to be called only once during the life cycle of a servlet. It is called when the servlet is first created, and not called again for each user request.

The servlet is normally created when a user first invokes a URL corresponding to the servlet. When a user invokes a servlet, a single servlet instance of the servlet gets created and is initialized using the server's configuration. After the `init()` method is completed (i.e. servlet has been initialized if it has not been initialized before), the `service()` method is called. The `init()` method looks like this:

```
public void init(ServletConfig config) throws ServletException
```

(b) The `service()` method:

The `service()` method is the main method to perform the actual task. The servlet container (i.e. web server) calls the `service()` method to handle requests coming from the client (browsers) and to write the formatted response back to the client. Each time the server receives a request for a servlet, the server spawns a new thread and calls `service()`. The `service()` method checks the HTTP request type (GET, POST, etc) and calls `doGet()`, `doPost()`, etc methods as appropriate. The signature of the `service()` method is:

```
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
```

The `doGet()` or `doPost()` methods are invoked by the `service()` method and thus they should be defined by us depending on what type of request is received from the client. If it is a normal

request for a URL or a request from an HTML form that has no METHOD specified, then `doGet()`

method is called by `service()`. If it is a request from an HTML form that specifically lists POST as

the METHOD, then doPost() method is called by service(). Their signatures are:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException
```

(c) The destroy() method:

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, and perform other such cleanup activities. Its signature is: public void destroy()

After the destroy() method is called, the servlet object is marked for garbage collection.

8. Discuss border layout with suitable example.

(5)

Answer:

A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region contains no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER. When adding a component to a container with a border layout, we have to use one of these five constants. A suitable example utilizing border layout is given below:

```
import java.awt.BorderLayout;
```

```
import javax.swing.*;
```

```
public class BorderLayoutTest extends JFrame {
```

```
    JButton north, south, east, west, center;
```

```
public BorderLayoutTest()
{
    setLayout(new BorderLayout());

    north = new JButton("NORTH");

    south = new JButton("SOUTH");

    east = new JButton("EAST");

    west = new JButton("WEST");

    center = new JButton("CENTER");


    add(north, BorderLayout.NORTH);
    add(south, BorderLayout.SOUTH);
    add(east, BorderLayout.EAST);
    add(west, BorderLayout.WEST);
    add(center, BorderLayout.CENTER);

    setSize(300, 300);

    setVisible(true);

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

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

9. Why multithreading is important in programming? Discuss.

(5)

Answer:

Multithreading is the ability to run multiple threads concurrently. Thus multithreaded programs can do many things at once. A thread is a separate unit of execution performing a particular task in a multithreaded program. A thread is implemented by a particular method in programming language such as Java. With multithreading, we can achieve multitasking. If we have multiple CPUs, then each thread of a program can run on different CPUs allowing parallelism. If we have only one CPU, then the threads take turns in their run and this context switching of threads takes less time than multiprocessing systems. For example, in word processors such as MS-Word, one thread may be receiving input, another thread may be performing grammar check, and another thread may be auto-saving the data, and so on. While one thread is waiting, another thread is scheduled to run in single CPU systems. Further, in multithreading, all the threads of a program share the same memory space whereas if it has been multiprocessing system, then each process would have its own memory space. Thus, multithreading is important in programming for achieving multitasking and for optimizing resource use.

In the code example below, we have two threads Thread 1 and Thread2. Thread1 prints odd numbers from 1 to 10 after 1 second and Thread2 prints even numbers in the same range after ½ second.

```
class Thread1 extends Thread {  
    @Override  
    public void run() {  
        try {  
            for (int i = 1; i <= 10; i = i + 2) {  
                Thread.sleep(1000);  
                System.out.println(i);  
            }  
        } catch (InterruptedException e) {
```

```
        e.printStackTrace();
    }
    System.out.println("Exiting Thread1");
}
}
```

```
class Thread2 extends Thread {
    @Override
    public void run() {
        try {
            for (int i = 2; i <= 10; i = i + 2) {
                Thread.sleep(500);
                System.out.println(i);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Exiting Thread2");
    }
}
```

```
public class ThreadTest {
    public static void main(String[] a) {
        Thread1 t1 = new Thread1();
```

```

        t1.start();

        Thread2 t2 = new Thread2();

        t2.start();

    }

}

```

**10. Discuss any five exception classes in Java.
(5)**

Answer:

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the Exception class, there is another subclass called Error which is derived from the Throwable class. The Exception class has two main subclasses: IOException class and RuntimeException class. There are many subclasses of RuntimeException class such as ArithmeticException, NullPointerException, IndexOutOfBoundsException, IllegalArgumentException, ClassCastException, etc. The ArrayIndexOutOfBoundsException class extends IndexOutOfBoundsException class. The exception hierarchy is shown below:

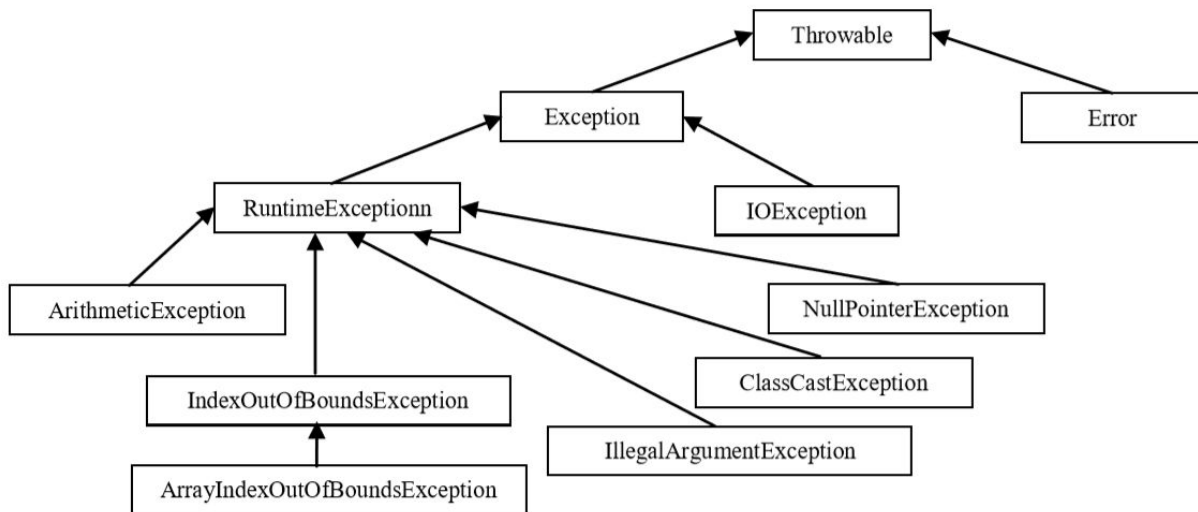


Fig: Exception Hierarchy

Some of the exception classes in Java are discussed below:

1.ArithmeticException class:

This class deals with exceptional arithmetic conditions such as the divide-by-zero exception.

The constructors for this class are:

- a. `public ArithmeticException()`
- b. `public ArithmeticException(String s)`

```
//Code Example public class ExceptionTest {  
  
    public static void main(String[] args) {  
  
        try {  
  
            int a = 10;  
  
            int b = 0;  
  
            int c = a / b;  
  
        } catch (ArithmeticException e) {  
  
            System.out.println(e);  
  
        }  
  
    }  
  
}
```

ii. ArrayIndexOutOfBoundsException class:

This class is a subclass of `IndexOutOfBoundsException` class which is a subclass of `RuntimeException` class. This class is used to deal with the accessing of an illegal index of an array. The illegal index means either negative value or a value greater than or equal to the size of the array. The constructors for this class are:

- a. `public ArrayIndexOutOfBoundsException()`
- b. `public ArrayIndexOutOfBoundsException(int index)`
- c. `public ArrayIndexOutOfBoundsException(String s)`

```
//Code Example public class ExceptionTest {
```

```

public static void main(String[] args) {

    try {

        int a[] = new int[10];

        int size = -10;

        if (size < 0) {

            throw new ArrayIndexOutOfBoundsException("Negative Index");

        }

    } catch (ArrayIndexOutOfBoundsException e) {

        System.out.println(e.getMessage());

    }

}

}

```

3.NullPointerException class:

This class is used to deal with the cases where there should be an object instead of null.

For example, when we try to find the length of string when the string contains null, then this exception occurs. The constructors for this class are:

- a. public NullPointerException()
- b. public NullPointerException(String s)

//Code Example public class ExceptionTest {

```

public static void main(String[] args) {

    try {

        String s = null;

        int len = s.length();

    } catch (NullPointerException e) {

        System.out.println(e);

    }

}

```

```

    }
}
}

```

4. **ClassCastException class:**

This class deals with the code that attempts to cast an object to a subclass of which it is not an instance. For example, when we try an integer object to be casted to a string object, then this exception occurs. The constructors for this class are:

- a. `public ClassCastException()`
- b. `public ClassCastException(String s)`

```

//Code Example public class ExceptionTest {

    public static void main(String[] args) {

        try {

            Object o = new Integer(5);

            System.out.println((String)o);

        } catch (ClassCastException e) {

            System.out.println(e);

        }

    }

}

```

v. **IllegalArgumentException class:**

This class deals with methods that are passed an illegal or inappropriate argument. The constructors for this class are:

- a. `public IllegalArgumentException()`

b. `public IllegalArgumentException(String s)`

```
//Code Example public class ExceptionTest {  
  
    static void myAge(int age){  
  
        if(age<0 || age>99){  
  
            throw new IllegalArgumentException("Invalid Age");  
  
        }  
  
    }  
  
    public static void main(String[] args) {  
  
        try {  
  
            myAge(111);  
  
        } catch (IllegalArgumentException e) {  
  
            System.out.println(e);  
  
        }  
  
    }  
  
}
```

**11. Discuss the use of event listeners to handle events with suitable example.
(5)**

Answer:

When keys are pressed or mouse buttons are clicked, some events are generated. If we specify what to do when an event is generated, then it is known as event handling. Event handling is basically an automatic notification that some action has occurred. There are three things that are done in event handling:

- i. Create a class that represents the event handler.
- ii. Implement an appropriate interface called “event listener interface” in the above class.

iii. Register the event handler

The event listeners listen for a particular event and whenever the event occurs they fire the action that is registered for them. A suitable example of using event listener to handle mouse click event is given below:

```
import java.awt.event.*;
```

```
import javax.swing.*;
```

```
public class EventListener extends JFrame implements ActionListener {  
    public EventListener(){  
        JButton btn = new JButton("Click Me");  
        add(btn);  
        btn.addActionListener(this); //registering the event handler  
  
        setSize(100, 100);  
        setVisible(true);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}
```

@Override

```
    public void actionPerformed(ActionEvent e) {  
        JOptionPane.showMessageDialog(null, "Hello");  
    }  
  
    public static void main(String []a) {  
        new EventListener();  
    }  
}
```

```
}
```

Here, whenever the “Click Me” button is clicked, an event is generated and a particular action of opening a dialog box and displaying message “Hello” happens.

12. What is socket? How can you communicate two programs in network using TCP sockets? (5)

Answer:

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. This way we can have multiple connections between our host and the server.

Normally, a server (one program) runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client (another program) to make a connection request. On the client side, the client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to make contact with the server on the server's machine and port. If everything goes well, the server accepts the connection. On the client side, if the connection is accepted, a socket is successfully created with its own local port and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets. The java.net package contains classes and interfaces required to do network programming. There are two major classes used: Socket class and ServerSocket class. The ServerSocket class is used to make a socket that server can use to listen for and accept connections to clients on a specific port. The Socket class is used to make a client socket to connect to the server using the server's hostname and port number. In the code example below, we are creating two programs: Server and Client. The Server program is run first so that it creates a socket on port 8000 to which any client may connect to. The Server's hostname is localhost. Next the Client program is run which connects to the Server. Here the Client program sends some radius value to the socket via an OutputStream which is received by the Server via an InputStream and the Server sends back the area to the Client via OutputStream. The complete code is:

```
//Server.java
```

```
package clientserverprogram;

import java.io.*;

import java.net.*;

public class Server {

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

        //create a server socket

        ServerSocket serverSocket = new ServerSocket(8000);

        //listen for client request

        Socket socket = serverSocket.accept();

        //create data input and output streams

        DataInputStream inputFromClient = new
DataInputStream(socket.getInputStream());

        DataOutputStream outputToClient = new
DataOutputStream(socket.getOutputStream());

        while(true) {

            double radius = inputFromClient.readDouble();

            System.out.println("Radius sent from client:"+radius);

            double area = Math.PI*radius*radius;

            outputToClient.writeDouble(area);

        }

    }

}
```

```
}
```

```
//Client.java package
```

```
clientserverprogram;
```

```
import java.io.*;
```

```
import java.net.Socket;
```

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

```
public class Client {
```

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

```
        //create a socket to connect to the server
```

```
        Socket socket = new Socket("localhost", 8000);
```

```
        //create an input stream to retrieve data from the server
```

```
        DataInputStream fromServer = new DataInputStream(socket.getInputStream());
```

```
        //create an output stream to send data to the server
```

```
        DataOutputStream toServer = new DataOutputStream(socket.getOutputStream());
```

```
        double radius;
```

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

```
        while(true) {
```

```
            System.out.println("Enter radius:");
```



```

radius = sc.nextDouble();

toServer.writeDouble(radius);

System.out.println("Area from Server:"+fromServer.readDouble());    }   }}

```

**13. Write a simple JSP program to display "Lalitpur, Nepal" 10 times.
(5)**

Answer:

The JSP program code is given below. It utilizes scriptlets to perform the task.

```

<!DOCTYPE html>

<html>

    <head>

        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

        <title>A simple JSP program</title>

    </head>

    <body>

        <h1>Displaying "Lalitpur, Nepal" 10 times!</h1>

        <table>

            <%

                for(int i=1; i<=10; i++) {

                    %>

                                <tr><td>Lalitpur, Nepal</td></tr>

            <% } %>

```

</table>
</body>
</html>

Tribhuvan University
Institute of Science and Technology
2071
Advanced Java Programming

Full Marks: 60

Pass Marks: 24

Time: 3 hours

New Course

Candidates are required to give their answers in their own words as far as practicable. The figures in the margin indicate full marks.

Section A (2*10=20)

Attempt any two questions.

1.) What is multithreading? Why is it important to develop computer programs? Discuss life cycle of thread in detail.

(2+2+6) Answer:

Multithreading is the ability to run multiple threads concurrently. Thus multithreaded programs can do many things at once. A thread is a separate unit of execution performing a particular task in a multithreaded program. A thread is implemented by a particular method in programming language such as Java. With multithreading, we can achieve multitasking. If we have multiple CPUs, then each thread of a program can run on different CPUs allowing parallelism. If we have only one CPU, then the threads take turns in their run and this context-switching of threads takes less time than multiprocessing systems. For example, in word processors such as MS-Word, one thread may be receiving input, another thread may be performing grammar check, and another thread may be autosaving the data, and so on. While one thread is waiting, another thread is scheduled to run in single CPU systems. Further, in multithreading, all the

threads of a program share the same memory space whereas if it has been multiprocessing system, then each process would have its own memory space. Thus, multithreading is important in programming for achieving multitasking and for optimizing resource use.

The life cycle of a thread in java is controlled by JVM. The life cycle of a java thread involves the following states in which the thread goes through.

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

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

2) Runnable: The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running: The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked): This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated: A thread is in terminated or dead state when its run() method exits.

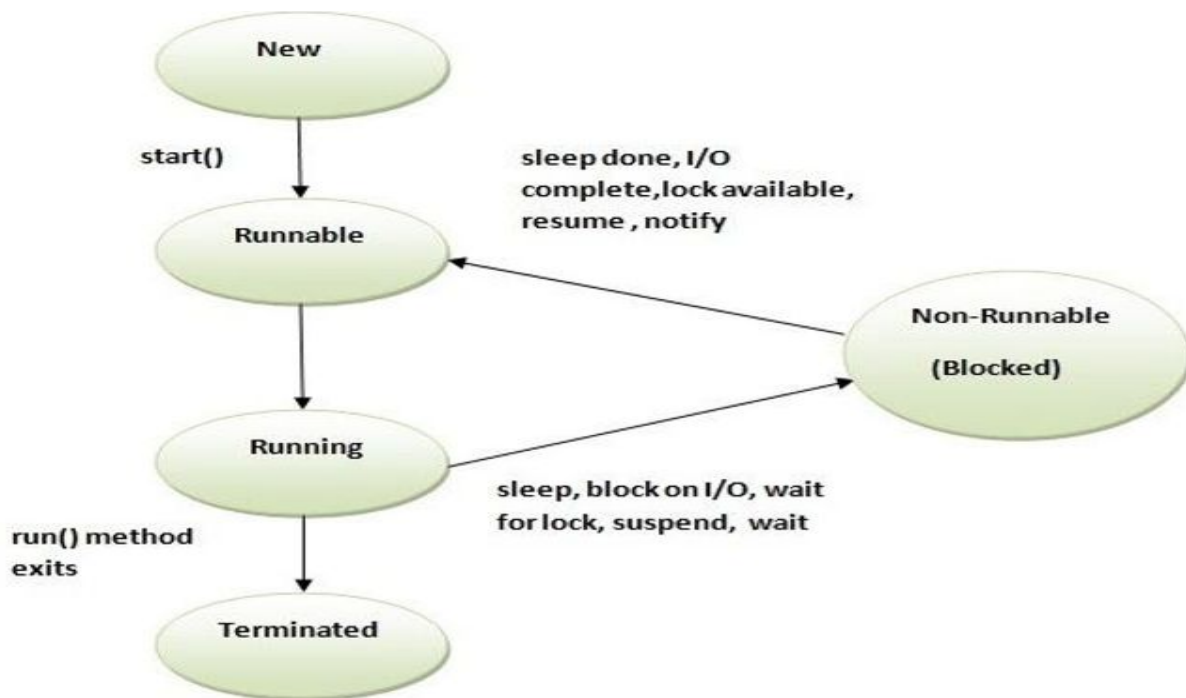


Fig: Thread States

2.) Write a program using swing components to find simple interest. Use text fields for inputs and output. Your program should display the result when the user presses a button.
(10)

Answer:

```
//SimpleInterest.java

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SimpleInterest extends JFrame implements ActionListener {
    JLabel l1 = new JLabel("Principal:");
    JTextField t1 = new JTextField(10);
```

```
JLabel l2 = new JLabel("Time:");  
JTextField t2 = new JTextField(10);  
JLabel l3 = new JLabel("Rate:");  
JTextField t3 = new JTextField(10);
```

```
JTextField t4 = new JTextField(10);  
JButton b = new JButton("Interest");
```

```
public SimpleInterest() {  
    setLayout(new FlowLayout());  
    setSize(200, 200);  
    setVisible(true);  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    add(l1);  
    add(t1);  
    add(l2);  
    add(t2);  
    add(l3);  
    add(t3);  
    t4.setEditable(false);  
    add(t4);  
    add(b);  
    b.addActionListener(this);  
}
```

@Override

```
public void actionPerformed(ActionEvent e) {  
    double pr = Double.parseDouble(t1.getText());  
    double ti = Double.parseDouble(t2.getText());  
    double ra = Double.parseDouble(t3.getText());  
    double in = (pr*ti*ra)/100;  
    t4.setText("" + in);  
}  
  
public static void main(String[] args) {  
    new SimpleInterest();  
}  
}
```

**3.) What is servlet? Differentiate it with JSP. Discuss life cycle of servlet in detail.
(2+3+5)**

Answer:

3) What is servlet? Differentiate it with JSP. Discuss life cycle of servlet in detail. (2+3+5)

→ Servlets are small programs that execute on the server side of a web connection. It is a java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets.

The differences between JSP and servlet are as follows:

- i) Servlet is a server-side program where as JSP is a webpage scripting language.
- ii) It's easier to code in JSP but a bit difficult to write in servlet because it involves http request and response.
- iii) JSP run slower compared to servlet as it takes compilation time to convert into java servlets. Servlets run faster than JSP.
- iv) Servlets are best for use when there is more processing and manipulation involved where as JSP is preferred when

there is not much processing.

v) The advantage of JSP over servlets is that we can build custom tags which can directly call Java beans where as there is no such custom tag facility in servlets.

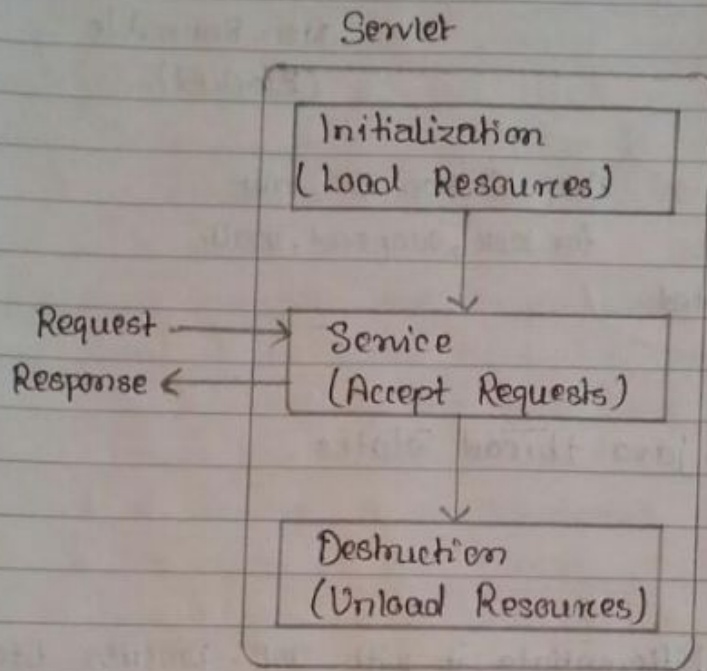


fig: Diagram of the Servlet Life cycle

Servlets follow a three-phase life: initialization, service and destruction.

i) Initialization

is the first phase of the servlet life cycle and represents the creation and initialization of resources the servlet may need to service requests. All servlets must implement the `javax.servlet.Servlet` interface which defines the `init()` method.

ii) Service

This phase represents all interactions with requests until the servlet is destroyed. The `service()` method is invoked once per a request and is responsible for generating the response to that request.

iii) Destruction

represents when a servlet is being removed from use by a container. The `Servlet` interface defines the `destroy()` method to correspond to the destruction life cycle phase.

Source: [Dipak Timilsaina](#)

Section B (8*5=40)

Attempt any eight questions.

4.) How do you achieve multiple inheritance in java? Discuss.
(5)

Answer:

4) How do you achieve multiple inheritance in java? Discuss. (6)

→ Multiple inheritance is the ability of a single class to inherit from multiple classes. Java does not have this capability. Multiple inheritance can cause the diamond problem.

What a Java class does have is the ability to implement multiple interfaces - which is considered a reasonable substitute for multiple inheritance, but without the added complexity.

Example:

```
interface X
```

```
{
```

```
    public void myMethod();
```

```
}
```

```
interface Y
```

```
{
```

```
    public void myMethod();
```

```
}
```

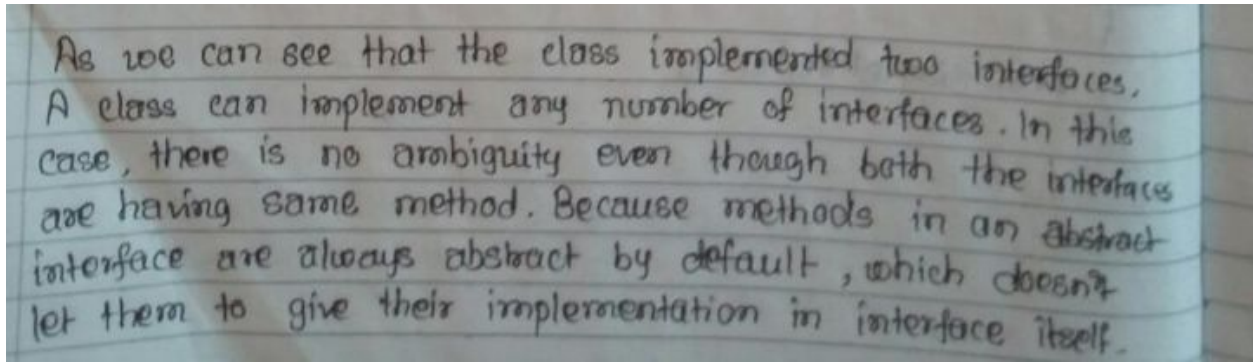
```
class Demo implements X, Y
```

```
{
```

```
    public void myMethod() {  
        System.out.println("Multiple Inheritance example using  
        interfaces");
```

```
    }
```

```
}
```



Source: [Dipak Timilsaina](#)

5.) What is JDBC? Discuss different driver types in JDBC.

(1+4) Answer:

JDBC is a Java database connectivity technology from Oracle Corporation. This technology is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases such as MySQL.

A JDBC driver is a set of Java classes that implement the JDBC interfaces, targeting a specific database. The JDBC interface comes with standard Java, but the implementation of these interfaces is specific to the database you need to connect to. Such an implementation is called a JDBC driver. There are 4 different types of JDBC drivers:

Type 1: JDBC-ODBC bridge driver

Type 2: Java + Native code driver

Type 3: All Java + Middleware translation driver

Type 4: All Java driver (Today, most drivers are type 4 drivers).

Type 1 JDBC Driver

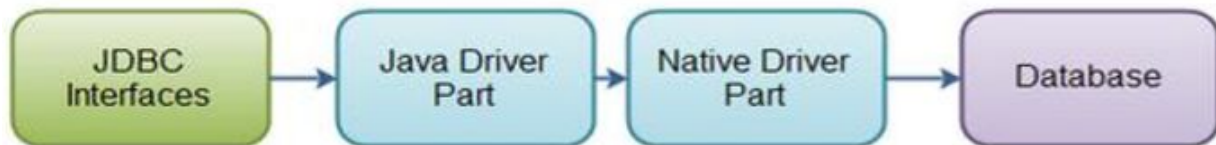
A type 1 JDBC driver consists of a Java part that translates the JDBC interface calls to ODBC calls. An ODBC bridge then calls the ODBC driver of the given database. Type 1 drivers are

(were) mostly intended to be used in the beginning, when there were no type 4 drivers (all Java drivers). Here is an illustration of how a type 1 JDBC driver is organized:



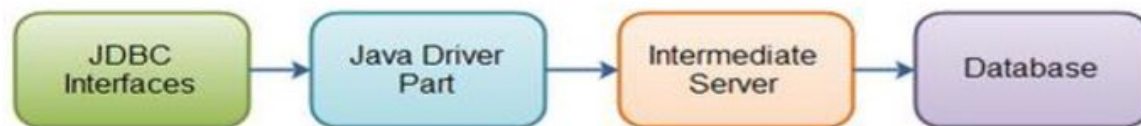
Type 1 JDBC driver.

Type 2 JDBC Driver A type 2 JDBC driver is like a type 1 driver, except the ODBC part is replaced with a native code part instead. The native code part is targeted at a specific database product. Here is an illustration of a type 2 JDBC driver:



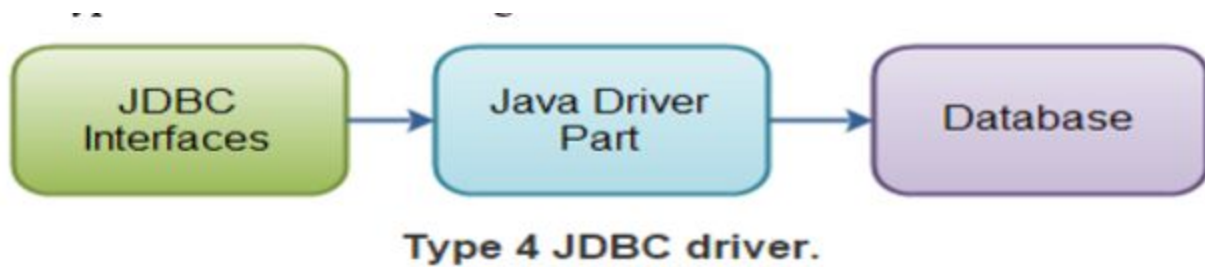
Type 2 JDBC driver.

Type 3 JDBC Driver A type 3 JDBC driver is an all Java driver that sends the JDBC interface calls to an intermediate server. The intermediate server then connects to the database on behalf of the JDBC driver. Here is an illustration of a type 3 JDBC driver:



Type 3 JDBC driver.

Type 4 JDBC Driver A type 4 JDBC driver is an all Java driver which connects directly to the database. It is implemented for a specific database product. Today, most JDBC drivers are type 4 drivers. Here is an illustration of how a type 4 JDBC driver is organized:



6.) Explain the importance of exception handling with suitable example.

(5) Answer:

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. So we have to handle exceptions. Exception handling is the ability of a program to intercept run-time errors, take corrective measures, and then continue. Java exception handling enables our Java applications to handle errors sensibly. Exception handling is a very important yet often neglected aspect of writing robust Java applications or components. When an error occurs in a Java program it usually results in an exception being thrown. How we throw, catch and handle these exception matters. There are several ways to do so. Exception handling is important for a program because it signals that some error or exceptional situation has occurred, and that it doesn't make sense to continue the program flow until the exception has been handled.

Advantages of Exception Handling:

- Exception handling allows us to control the normal flow of the program by using exception handling in program.
- It throws an exception whenever a calling method encounters an error providing that the calling method takes care of that error.
- It also gives us the scope of organizing and differentiating between different error types using a separate block of codes. This is done with the help of try catch blocks.

The following program throws an exception whenever the user of the program gives an invalid age. The invalid age condition is age being negative or age greater than 150. If exception has not been handled here, then the user may possibly have provided the wrong input.

```
class InvalidAgeException extends Exception {
```

```

        public InvalidAgeException(String msg) {
            System.out.println(msg);
        }
    }

class Person {
    private int age;

    public void setAge(int age) throws InvalidAgeException {
        if (age < 0 || age > 150) {
            throw new InvalidAgeException("Invalid Age");
        }
        this.age = age;
    }
}

public class ExceptionHandling {
    public static void main(String[] args) {
        Person p = new Person();
        try {
            p.setAge(-20);
        } catch (InvalidAgeException e) {
        }
    }
}

```

7.) Discuss group layout with suitable example.

(5) Answer:

```
import javax.swing.*;

public class GroupLayoutTest extends JFrame {

    public GroupLayoutTest() {

        JPanel panel = new JPanel();

        GroupLayout layout = new GroupLayout(panel);

        JButton one = new JButton("Button 1");

        JButton two = new JButton("Button 2");

        JButton three = new JButton("Button 3");

        JButton four = new JButton("Button 4");

        JButton five = new JButton("Button 5");

        JButton six = new JButton("Button 6");

        layout.setHorizontalGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)

            .addGroup(layout.createSequentialGroup()

                .addComponent(one)

                .addComponent(two)

                .addComponent(three)) //first group
```



```
.addGroup(layout.createSequentialGroup()  
    .addComponent(four)  
    .addComponent(five)) //second group  
.addComponent(six));
```

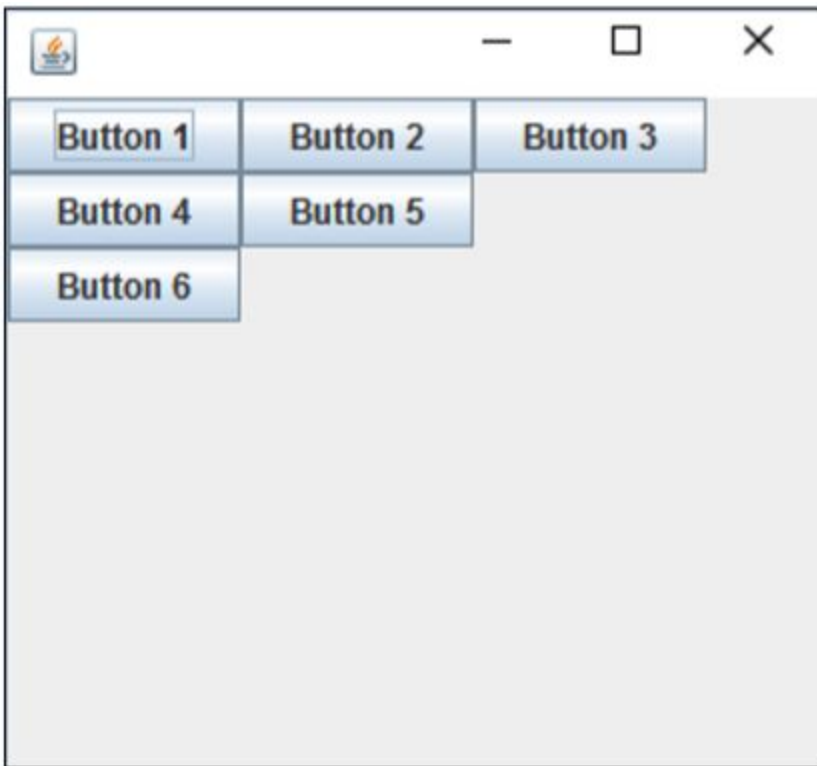
```
layout.setVerticalGroup(layout.createSequentialGroup()  
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)  
        .addComponent(one).addComponent(two).addComponent(three))  
    .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)  
        .addComponent(four).addComponent(five))  
    .addComponent(six));
```

```
panel.setLayout(layout);  
add(panel);
```

```
setSize(400, 400);  
setVisible(true);  
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```

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


The output of the above program will be:



8.) Write a simple java program to read from and write to files.
(5)

Answer:

```

→
import java.io.*;

public class FileReadwrite
{
    public static void main(String a[]) throws IOException
    {
        FileWriter writer = new FileWriter("D:/hello.txt");
        writer.write("This\n is\n an\n example");
        writer.close();
        FileReader fr = new FileReader("D:/hello.txt");
        char a[] = new char[50];
        fr.read(a);
        for(char c:a)
            System.out.println(c);
        fr.close();
    }
}

```

Output :

```

This
is
an
example

```

Source: [Dipak Timilsaina](#)

9.) What is TCP socket? Differentiate it with UDP socket.

(2+3) Answer:

9) What is TCP socket? Differentiate it with UDP socket. (4+3)

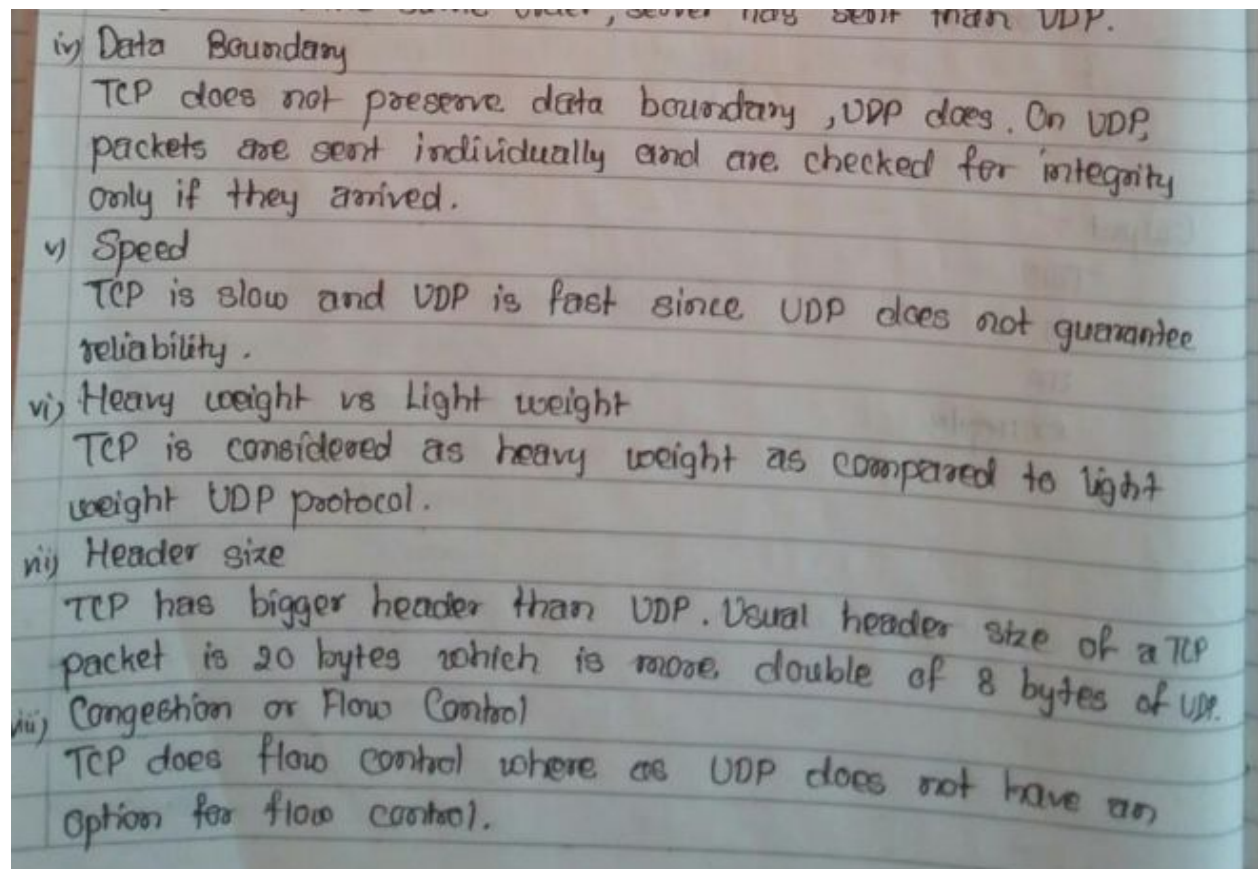
→ A TCP socket is that Transmission Control Protocol suite of communication flow between two programs running over a network. It is used to create a new communication endpoint.

The differences between TCP socket and UDP socket can be illustrated below:

i) Connection oriented vs Connection less
TCP socket is a connection-oriented where connection is established between client and server, where as UDP socket is connection less.

ii) Reliability
TCP provides delivery guarantee, which means a message sent using TCP protocol is guaranteed to be delivered to client. UDP is unreliable.

iii) Ordering
TCP also guarantees order of message. Message will be delivered to client in the same order, server has sent than UDP.



Source: [Dipak Timilsaina](#)

10.) Discuss any five event classes in java.

(5) Answer:

Every time a user interacts with a component on the GUI, events are generated. Events are objects that store information like the type of event that occurred, the source of the event, etc. Once the event is generated, then the event is passed to other objects, which handle or react to the event, called event listeners. Some event classes that represent the event and their corresponding listeners are:

Events	Listeners
FocusEvent	FocusListener
MouseEvent	MouseListener, MouseMotionL
WindowEvent	WindowListener
TextEvent	TextListener
KeyEvent	KeyListener
ItemEvent	ItemListener
ActionEvent	ActionListener

Five of the many important event classes are:

(1) **ActionEvent**: In Java, most components have a special event called an **ActionEvent**. This is loosely speaking the most common or canonical event for that component. A good example is a click for a button. To have any component listen for an **ActionEvent**, we must register the component with an **ActionListener** as:

```
component.addActionListener(new MyActionListener());
```

and then write the

```
public void actionPerformed(ActionEvent e);
```

method for the component to react to the event.

(2) **ItemEvent**: It is generated in case of **JRadioButton**, **JCheckBox**, **JCheckBoxMenuItem**, **JComboBox** and **JList** components. To have any component listen for an **ItemEvent**, we must register the component with an **ItemListener** as:

```
component.addItemListener(new MyItemListener);
```

and then write the

```
public void itemStateChanged(ItemEvent e);
```

method for the component to react to the event.

(3) **MouseEvent**:

It is generated in case of mouse clicks and mouse motion when the cursor is over Swing components such as JFrame. To have any component listen for a MouseEvent, we must register the component with either MouseListener or MouseMotionListener as:

```
component.addMouseListener(new MyMouseListener);
```

OR

```
component.addMouseMotionListener(new MyMouseMotionListener);
```

and then write the

```
public void mouseDragged(MouseEvent e);
```

```
public void mouseMoved(MouseEvent e);
```

OR

```
public void mouseClicked(MouseEvent e);
```

```
public void mousePressed(MouseEvent e);
```

```
public void mouseReleased(MouseEvent e);
```

```
public void mouseEntered(MouseEvent e);
```

```
public void mouseExited(MouseEvent e);
```

methods for the component to react to the event.

(4) KeyEvent:

It is generated in case of key presses and key depresses on text fields such as JTextField and JTextArea. One example of KeyEvent is user typing in a textfield. To have any component listen for a KeyEvent, we must register the component with KeyListener as:

```
component.addKeyListener(new MyKeyListener);
```

and then write the

```
public void keyPressed(KeyEvent e);
```

```
public void keyTyped(KeyEvent e);
```

```
public void keyReleased(KeyEvent e);
```

methods for the component to react to the event.

(5) WindowEvent:

It is generated in case of Swing window and its derivative components such as JFileDialog and JFrame. One example of WindowEvent is user closing a frame. To have any component listen for a WindowEvent, we must register the component with WindowListener as:

```
component.addWindowListener(new MyWindowListener);
```

and then write the

```
public void windowOpened(WindowEvent e);
```

```
public void windowClosing(WindowEvent e);
```

```
public void windowClosed(WindowEvent e);
```

```
public void windowIconified(WindowEvent e);
```

```
public void windowDeiconified(WindowEvent e);
```

```
public void windowActivated(WindowEvent e);
```

```
public void windowDeactivated(WindowEvent e);
```

methods for the component to react to the event.

However, we also do have the option to use corresponding Adapter classes instead of these Listener interfaces by which we can implement only the methods that are needed for the component and not define all the methods that are not necessary.

11.) What is java beans? Differentiate it with java classes.

(1+4)

Answer:

A java bean is reusable software component written in java that can manipulated visually in an application builder tool .This idea is that one can start with a collection of such components , and quickly wire them together to form complex programs without actually writing any new code.

A JavaBean is a software component that is written in java programming language. It is similar to a java except following differences:

- A JavaBean must adhere to certain conventions regarding property and event interface definitions whereas java classes need to be.
- A builder tool must be able analyze how a Bean works through introspection process . This is not necessary java classes.
- Java Beans must be customizable . It should allow its state to manipulated at design time. This is not necessary to be true in case of java classes .
- JavaBeans must implement serializable interface which is not necessary for java classes.
- Every JavaBean must have one public no-argument constructor which is not true for java classes .

12.) What is RMI? Differentiate it with CORBA.

(2+3) Answer:

Refer 2072 Q no 12

13.) Write a simple JSP program to display “Tribhuvan University” 10 times.

(5) Answer:

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
    <title>A simple JSP program</title> </head>
```

```
  </head>
```

```
  <body>
```

```
    <h1>Displaying " Tribhuvan University " 10 times!</h1>
```

```
    <table>
```



```
<%  
    for(int i=1; i<=10; i++) {  
        %>  
        <tr><td> Tribhuvan University </td></tr>  
    <% } %>  
</table>  
</body>  
</html>
```

Tribhuvan University
Institute of Science and Technology
2072
Advanced Java Programming

Pass Marks : 24

Time : 3 hours

New Course

***Candidates are required to give their answers in their own words as far as practicable.
The figures in the margin indicate full marks.***

Section A (2*10=20)

Attempt any two questions.

1) What is exception handling? Discuss exception handling in detail with suitable example.

Answer:

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

There are two types of exceptions in Java:

- 1)Checked exceptions
- 2)Unchecked exceptions

Checked exceptions

All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

Unchecked Exceptions

Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it's the responsibility of the programmer to handle these exceptions and provide a safe exit. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Exception Handling Example Program

```
class ExceptionHandlingExample{  
    public static void main(String[] args){  
        int num1,num2,num3;  
        num1=20;  
        num2=0;  
        try{  
            num3=num1/num2;  
            System.out.println("Result is "+num3);  
        }catch(ArithmeticException ae){
```

```
        System.out.println("Numbers cannot be divided by zero");
    }
    num3=num1+num2;
    System.out.println("Result after addition is "+num3);
}
}
```

Sample Output

Output is:

Numbers cannot be divided by zero

Result after addition is 20

2) What is layout management? Discuss any three layout management classes with example of each.

Layout management is the process of determining the size and position of components. By default, each container has a *layout manager* -- an object that performs layout management for the components within the container. Components can provide size and alignment hints to layout managers, but layout managers have the final say on the size and position of those components.

The Java platform supplies five commonly used layout managers: BorderLayout, BoxLayout, FlowLayout, GridBagLayout, and GridLayout. These layout managers are designed for displaying multiple components at once, and are shown in the preceding figure. A sixth provided class, CardLayout, is a special-purpose layout manager used in combination with other layout managers. You can find details about each of these six layout managers, including hints for choosing the appropriate one, in Using Layout Managers.

3) Write a Java program using JDBC to extract name of those students who live in Kathmandu district, assuming that the student table has four attributes (ID, name, district, and age).

```
package JDBC;

import java.sql.*;

public class Q3_2072 {

    static final String db_url="jdbc:mysql://localhost:3306/students";

    public static void main(String[] args) {

        Connection conn=null;

        Statement statement=null;

        ResultSet resultSet=null;

        try{

            //connection to database

            conn=DriverManager.getConnection(db_url,"root","");

            //sql query

            String sql="Select *from student where address='dhading'";

            //executing sql statement

            resultSet=statement.executeQuery(sql);

            ResultSetMetaData metaData=resultSet.getMetaData();

            int numOfCols=metaData.getColumnCount();

            System.out.println("Students-----");

            //getting column names

            for(int i=1;i<=numOfCols;i++){
```

```

        System.out.printf("%-8s\t", metaData.getColumnName(i));

        System.out.println();
    }

    //executing query
    while(resultSet.next()){
        for(int i=1;i<=numOfCols;i++){
            System.out.printf("%-8s\t", resultSet.getObject(i));

            System.out.println();
        }
    }

}

}catch(SQLException e){
    e.printStackTrace();
}finally{
    try
    {
        if(statement!=null){
            statement.close();

            resultSet.close();

            conn.close();
        }
    }

    catch(SQLException ex){
        ex.printStackTrace();
    }
}

```

```
}  
  
}  
  
}
```

Section B (8*5=40)

Attempt any eight questions .

4) Write an object oriented program in Java to find area of circle.

```
import java.util.Scanner;  
  
/*  
 * Java Program to calculate area of circle  
 */  
  
public class Circle  
{  
    public static void main(String args[]) {  
        // creating scanner to accept radius of circle  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("Welcome in Java program to calculate area of  
circle");  
        System.out.println("Formula for calculating area of circle is 'PI*R^2'  
");  
  
        System.err.println("Please enter radius of circle:");  
        float radius = scanner.nextFloat();  
  
        double area = area(radius);  
        System.out.println("Area of Circle calculate by Java program is :  
" + area);  
    }  
}
```

```

        scanner.close();
    }

    public static double area(float radius) {
        double interest = Math.PI * radius * radius;
        return interest;
    }
}

```

Output

Welcome in Java program to calculate area of circle

Formula for calculating area of circle is 'PI*R^2'

Please enter radius of circle:

2

Area of Circle calculate by Java program is : 12.566370614359172

5) Write a simple Java program that reads data from one file and writes the data to another file.

Answer:

We can do this by reading the file using **FileInputStream** object and write into another file using **FileOutputStream** object.

Here is the sample code

```

package java_io_examples;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Vector;

```

```

public class Filetest {

    public static void main(String[] args) {

        try {

            FileInputStream fin = new FileInputStream("D:\\testout.txt");

            int i = 0;
            String s = "";

            while((i=fin.read())!=-1) {

                s = s + String.valueOf((char)i);

            }

            FileOutputStream fout = new
            FileOutputStream("D:\\newtestout1.txt");
            byte[] b = s.getBytes();

            fout.write(b);
            fout.close();

            System.out.println("Done reading and writing!!!");

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

    }

}

```

#ALTERNATIVE

```

public void readwrite() throws IOException
{
    // Reading data from file
    File f1=new File("D:/read.txt");
    FileReader fr=new FileReader(f1);
    BufferedReader br=new BufferedReader(fr);

    String s = br.readLine();

    // Writing data
    File f2=new File("D:/write.txt");
    FileWriter fw=new FileWriter(f2);
    BufferedWriter bw=new BufferedWriter(fw);
    while(s!=null)

```



```
{
    bw.write(s);
    bw.newLine();
    System.out.println(s);
    s=br.readLine();

}
bw.flush();
bw.close();
}
```

6) What are the benefits of using swing components? Explain.

Answer:

There are a few other advantages to Swing over AWT:

- Swing provides both additional components and added functionality to AWT-replacement components
- Swing components can change their appearance based on the current "look and feel" library that's being used. You can use the same look and feel as the platform you're on, or use a different look and feel
- Swing components follow the Model-View-Controller paradigm (MVC), and thus can provide a much more flexible UI.
- Swing provides "extras" for components, such as:
 - Icons on many components
 - Decorative borders for components
 - Tooltips for components
- Swing components are lightweight (less resource intensive than AWT)
- Swing provides built-in double buffering
- Swing provides paint debugging support for when you build your own components

7) Discuss any three event classes in Java.

Event Class	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.
AdjustmentEvent	Generated when a scroll bar is manipulated.
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.
ContainerEvent	Generated when a component is added to or removed from a container.
FocusEvent	Generated when a component gains or loses keyboard focus.
InputEvent	Abstract super class for all component input event classes.
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.

8) Discuss different driver types on JDBC.

JDBC drivers are classified into the following types:

📖 A type 1 driver translates JDBC to ODBC and relies on an ODBC driver to communicate with the database. Sun includes one such driver, the JDBC/ODBC bridge, with the JDK. However, the bridge requires deployment and proper configuration of an ODBC driver. When JDBC was first released, the bridge was handy for testing, but it was never intended for production use. At this point, plenty of better drivers are available, and is advised against using the JDBC/ODBC bridge.

📖 A type 2 driver is written partly in Java and partly in native code; it communicates with the client API of a database. When you use such a driver, you must install some platform-specific code in addition to a Java library.

📖 A type 3 driver is a pure Java client library that uses a database-independent protocol to communicate database requests to a server component, which then translates the requests into a database-specific protocol. This can simplify deployment since the database-dependent code is located only on the server.

📖 A type 4 driver is a pure Java library that translates JDBC requests directly to a database-specific protocol.

Most database vendors supply either a type 3 or type 4 driver with their database. Furthermore, a number of third-party companies specialize in producing drivers with better standards conformance, support for more platforms, better performance, or, in some cases, simply better reliability than the drivers that are provided by the database vendors.

In summary, the ultimate goal of JDBC is to make possible the following: Programmers can write applications in the Java programming language to access any database, using standard SQL statements or even specialized extensions of SQL while still following Java language conventions. Database vendors and database tool vendors can supply the low-level drivers. Thus, they can optimize their drivers for their specific products

9) What is socket? Differentiate TCP socket from UDP socket.

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The java.net package in the Java platform provides a class, Socket, that implements one side of a two way connection between your Java program and another program on the network. The Socket class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the java.net.Socket class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

TCP	UDP
-----	-----

<ul style="list-style-type: none"> • Transfer control protocol 	<ul style="list-style-type: none"> • User Datagram protocol
<ul style="list-style-type: none"> • Is connection oriented protocol 	<ul style="list-style-type: none"> • Connection less protocol
<ul style="list-style-type: none"> • Data is read as “stream” with nothing distinguishing where one packets ends and other begins. There is guarantee of transferring data in order. 	<ul style="list-style-type: none"> • Data Is send independently in the form of packets called datagrams. There is no guarantee of transferring data in order.
<ul style="list-style-type: none"> • Used by other protocols such as HTTP, HTTPS ,FTP , SMTP , telnet 	<ul style="list-style-type: none"> • By DNS , DHCP
<ul style="list-style-type: none"> • Speed of transfer is slow 	<ul style="list-style-type: none"> • fast
<ul style="list-style-type: none"> • Guarantee of transfer of data 	<ul style="list-style-type: none"> • Not guarantee
<ul style="list-style-type: none"> • Header size is 20 bytes. 	<ul style="list-style-type: none"> • Header size is 8 bytes.
<ul style="list-style-type: none"> • Heavy weight 	<ul style="list-style-type: none"> • Light weight
<ul style="list-style-type: none"> • Consist acknowledgement segment 	<ul style="list-style-type: none"> • No acknowledgement
<ul style="list-style-type: none"> • Hand shake SYN 	<ul style="list-style-type: none"> •

10)What is servlet? Discuss its life cycle.

Answer:

Servlets are small programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server.

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. All servlets must implement the `Servlet` interface, which defines life-cycle methods.

The Life Cycle of a Servlet

Three methods are central to the life cycle of a servlet. These are

1. `init()`,
2. `service()`, and
3. `destroy()`.

The `init()` Method

The `init` method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the `init` method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

The `service()` Method

The `service()` method is the main method to perform the actual task. The servlet container (i.e. web server) calls the `service()` method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls `service`. The `service()` method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls `doGet`, `doPost`, `doPut`, `doDelete`, etc. methods as appropriate.

The `destroy()` Method

The `destroy()` method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection.

11)What is Java Bean? How is it different from other Java classes?

Answer:

A java bean is reusable software component written in java that can manipulated visually in an application builder tool .This idea is that one can start with a collection of such components , and quickly wire them together to form complex programs without actually writing any new code.

A JavaBean is a software component that is written in java programming language. It is similar to a java except following differences:

- A JavaBean must adhere to certain conventions regarding property and event interface definitions whereas java classes need to be.
- A builder tool must be able analyze how a Bean works through introspection process . This is not necessary java classes.
- Java Beans must be customizable . It should allow its state to manipulated at design time. This is not necessary to be true in case of java classes .
- JavaBeans must implement serializable interface which is not necessary for java classes.
- Every JavaBean must have one public no-argument constructor which is not true for java classes .

12)What is RMI? How is it different from CORBA?

Java's Remote Method Invocation (commonly referred to as RMI) is used for client and server models. RMI is the object oriented equivalent of RPC (Remote procedure call). The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM. RMI provides for remote communication between programs written in the Java programming language. RMI differs from CORBA (Common Object Request Broker Architecture) in the sense that CORBA is language/machine independent whereas RMI is designed for machines running JVM, i.e. JVM

dependent. The advantage of using RMI is that some operations can be executed significantly faster on the remote system than on the local client compute.

RMI	COBRA
RMI is java specific technology.	CORBA has implemented for many language.
RMI use java interface.	CORBA uses IDL to separate interface from implementation.
RMI program can download new class.	CORBA does not have this code sharing mechanism.
RMI is machine dependent architecture.	CORBA is machine independent architecture
RMI application are slower than CORBA.	CORBA application are faster than RMI.

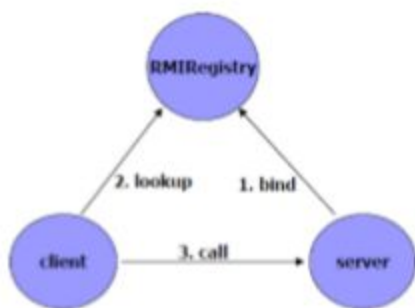
13) Write Short Notes on:

a) Multithreading

Multithreading is the ability to run multiple threads concurrently. Thus multithreaded programs can do many things at once. A thread is a separate unit of execution performing a particular task in a multithreaded program. A thread is implemented by a particular method in programming language such as Java. With multithreading, we can achieve multitasking. If we have multiple CPUs, then each thread of a program can run on different CPUs allowing parallelism. If we have only one CPU, then the threads take turns in their run and this context-switching of threads takes less time than multiprocessing systems. For example, in word processors such as MS-Word, one thread may be receiving input, another thread may be performing grammar check, and another thread may be autosaving the data, and so on. While one thread is waiting, another thread is scheduled to run in single CPU systems. Further, in multithreading, all the threads of a program share the same memory space whereas if it has been multiprocessing system, then each process would have its own memory space. Thus, multithreading is important in programming for achieving multitasking and for optimizing resource use.

b) RMI architecture

To send a message to a remote server object, the client object has to find the remote object. Remote objects are listed in the RMI Registry. The Registry is a program that binds remote objects with a textual name and runs on a known port (default 1099). The RMI Registry service can be started within the server JVM, via the `LocateRegistry.createRegistry()` API. The client, before performing invoking a remote method, must first lookup the registry to obtain access to the remote object. After that, the client can call methods on the remote objects.



A remote object is one whose instances can be used remotely. There are two parts to defining a remote class: its interface and the actual class itself. The interface definition must be public and it must extend the interface `java.rmi.Remote`. Further every method in the interface must declare that it throws `java.rmi.RemoteException`. The Remote class must implement a Remote interface and it should extend `java.rmi.server.UnicastRemoteObject` class. Objects of this class exist in the address space of the server and can be invoked remotely. Objects in the remote class may have methods that are not in its remote interface. However, these can only be invoked locally. The interface must be available to both client and server. The class should only be on the server.

Tribhuvan University
Institute of Science and Technology
2073
Advanced Java Programming

Pass Marks : 24)

Time: 3 hours.

(NEW COURSE)

Candidates are required to give their answers in their own words as far as practicable.
Group A

Attempt any Two Questions (2*10=20)

1. What is multithreading? How can you write multithreaded programs in java? Discuss with suitable example. (2+8)

Answer:

Multithreading is the ability to run multiple threads concurrently. Thus multithreaded programs can do many things at once. A thread is a separate unit of execution performing a particular task in a multithreaded program. A thread is implemented by a particular method in programming language such as Java. With multithreading, we can achieve multitasking. If we have multiple CPUs, then each thread of a program can run on different CPUs allowing parallelism. If we have only one CPU, then the threads take turns in their run and this context-switching of threads takes less time than multiprocessing systems. For example, in word processors such as

MS-Word, one thread may be receiving input, another thread may be performing grammar check, and another thread may be autosaving the data, and so on. While one thread is waiting, another thread is scheduled to run in single CPU systems. Further, in multithreading, all the threads of a program share the same memory space whereas if it has been multiprocessing system, then each process would have its own memory space. Thus, multithreading is important in programming for achieving multitasking and for optimizing resource use.

Creating a thread in Java

There are two ways to create a thread in Java:

- 1) By extending Thread class.
- 2) By implementing Runnable interface.

2. What is Java Beans? How is it different from other Java classes? Discuss property design patterns with suitable example of each. (1+2+7)

Answer:

A java bean is reusable software component written in java that can manipulated visually in an application builder tool .This idea is that one can start with a collection of such components , and quickly wire them together to form complex programs without actually writing any new code.

A JavaBean is a software component that is written in java programming language. It is similar to a java except following differences:

- A JavaBean must adhere to certain conventions regarding property and event interface definitions whereas java classes need to be.
- A builder tool must be able analyze how a Bean works through introspection process . This is not necessary java classes.
- Java Beans must be customizable . It should allow its state to manipulated at design time. This is not necessary to be true in case of java classes .
- JavaBeans must implement serializable interface which is not necessary for java classes.
- Every JavaBean must have one public no-argument constructor which is not true for java classes .

Bean Properties:-

A bean property is a named attribute of a bean that can affect its behavior or appearance. examples of bean properties include color, font, font size, label.

Types of Bean properties:-

① Simple property:-

A bean property with a single value whose changes are independent of changes in any other property. To add simple properties to a bean, add appropriate `getXxx` and `setXxx` methods.

② Indexed Property:-

A bean property that supports a range of values instead of a single value is called indexed property. An indexed property is an array of properties or objects that support a range of values and enables the access or to specify an element of a property to read or write.

③ Bound Property:-

A bean property for which a change to the property results in a notification being sent to some other bean is called bound property. Whenever a bound property changes, notification of the change is sent to interested listeners. The access methods for a bound property are defined in the same way as those for simple properties.

④ Constrained property:-

A bean property for which a change to the property results in validation by other bean. The other bean may reject the change if it is not appropriate.

3. What is socket? How can you write java programs that communicate with each other using TCP sockets? Discuss with suitable example. (2+8)

Answer:

Socket is an interface for programming networks at the transport layer. Network communication using sockets is very much similar to performing the file I/O. In fact, socket handle is treated like file handle.

Program for chatting between client and server

//Server.java

```
import java.io.*;

import java.net.*;

public class Server

{

public static void main(String a[])throws IOException

{

try

{

System.out.println("SERVER:.....\n");

ServerSocket s=new ServerSocket(95);

System.out.println("Server Waiting For The Client");

Socket cs=s.accept();

InetAddress ia=cs.getInetAddress();

String cli=ia.getHostAddress();

System.out.println("Connected to the client with IP:"+cli);

BufferedReader in=new BufferedReader(new

InputStreamReader(cs.getInputStream()));

PrintWriter out=new PrintWriter(cs.getOutputStream(),true);
```

```

do
{
BufferedReader din=new BufferedReader(new    InputStreamReader(System.in));

    System.out.print("To Client:");

    String tocl=din.readLine();

    out.println(tocl);

    String st=in.readLine();

    if(st.equalsIgnoreCase("Bye")||st==null)break;

    System.out.println("From Client:"+st);

}while(true);

in.close();

out.close();

cs.close();

}

catch(IOException e) { }

}

}

```

//Client.java

```

import java.io.*;

import java.net.*;

public class Client

{

public static void main(String a[])throws IOException

{

```

```

try
{
System.out.println("CLIENT:.....\n");

Socket con=new Socket("localhost",95);

BufferedReader in=new BufferedReader(new    InputStreamReader(con.getInputStream()));
PrintWriter out=new PrintWriter(con.getOutputStream(),true);

while(true)
{
String s1=in.readLine();

System.out.println("From Server:"+s1);

System.out.print("Enter the messages to the server:");

BufferedReader din=new BufferedReader(new    InputStreamReader(System.in));

String st=din.readLine();

out.println(st);

if(st.equalsIgnoreCase("Bye")||st==null)break;
}

in.close();

out.close();

con.close();

}

catch(UnknownHostException e){ }

}

}

```

Group B

Attempt any Eight questions. (8*5=40)

4. Write an object oriented program to find the area and perimeter of rectangle .

To **calculate the perimeter and area of rectangle**, we need the dimensions of two adjacent sides(length and width) of a rectangle. The **area of a rectangle** is the amount of two-dimensional space inside the boundary on rectangle. The **perimeter of a rectangle** is the linear distance around the boundary of the rectangle.

Java program to calculate area and perimeter of a rectangle

```
package com.tcc.java.programs;

import java.util.*;

public class RectangleAreaPerimeter {
    public static void main(String args[]) {
        int length, width, area, perimeter;

        Scanner in = new Scanner(System.in);
        System.out.println("Enter length of Rectangle");
        length = in.nextInt();
        System.out.println("Enter width of Rectangle");
        width = in.nextInt();
        // Area of rectangle = length X width
        area = length*width;
        // Perimeter of rectangle = 2 X (length X width)
        perimeter = 2*(length + width);
        System.out.println("Area of Rectangle : "+ area);
        System.out.println("Rectangle of Rectangle : "+ perimeter);
    }
}
```

Output:

```
Enter length of Rectangle
4
Enter width of Rectangle
5
```

Area of Rectangle : 20

Rectangle of Rectangle : 18

5. Write the simple java program that reads data from one file and writes data to another file.

We can do this by reading the file using **FileInputStream** object and write into another file using **FileOutputStream** object.
Here is the sample code

```
package java_io_examples;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Vector;

public class Filetest {

    public static void main(String[] args) {

        try {

            FileInputStream fin = new FileInputStream("D:\\testout.txt");

            int i = 0;
            String s = "";

            while((i=fin.read())!=-1) {

                s = s + String.valueOf((char)i);

            }

            FileOutputStream fout = new
            FileOutputStream("D:\\newtestout1.txt");
            byte[] b = s.getBytes();

            fout.write(b);
            fout.close();

            System.out.println("Done reading and writing!!!");

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

    }

}
```



```
}
```

#ALTERNATIVE

```
public void readwrite() throws IOException
{
    // Reading data from file
    File f1=new File("D:/read.txt");
    FileReader fr=new FileReader(f1);
    BufferedReader br=new BufferedReader(fr);

    String s = br.readLine();

    // Writing data
    File f2=new File("D:/write.txt");
    FileWriter fw=new FileWriter(f2);
    BufferedWriter bw=new BufferedWriter(fw);
    while(s!=null)
    {
        bw.write(s);
        bw.newLine();
        System.out.println(s);
        s=br.readLine();
    }
    bw.flush();
    bw.close();
}
```

6. Discuss border layout with example.

BorderLayout is the default **layout** manager for the content pane of a JFrame, JWindow, JDialog, JInternalFrame, and JApplet. Place components against any of the four **borders** of the container and in the center. The component in the center fills the available space.

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**

4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:

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

public class Border {
    JFrame f;
    Border(){
        f=new JFrame();

        JButton b1=new JButton("NORTH");
        JButton b2=new JButton("SOUTH");
        JButton b3=new JButton("EAST");
        JButton b4=new JButton("WEST");
        JButton b5=new JButton("CENTER");

        f.add(b1, BorderLayout.NORTH);
        f.add(b2, BorderLayout.SOUTH);
        f.add(b3, BorderLayout.EAST);
        f.add(b4, BorderLayout.WEST);
        f.add(b5, BorderLayout.CENTER);

        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
}
```

OUTPUT



#ALTERNATIVE

```
package borderlayout;
```

```
import java.awt.BorderLayout;
```

```
import javax.swing.JButton;
```

```
import javax.swing.JFrame;
```

```
public class BorderLayoutTest extends JFrame {
```

```
    /**
```

```
     * @param args the command line arguments
```

```
     */
```

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

```
        new BorderLayoutTest();
```

```
}
```

```
public BorderLayoutTest() {  
    this.setTitle("BorderLayout");  
    this.setSize(300, 300);  
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    this.setLayout(new BorderLayout());  
    JButton north = new JButton("North");  
    JButton south = new JButton("South");  
    JButton east = new JButton("East");  
    JButton west = new JButton("West");  
    JButton center = new JButton("Center");  
  
    this.add(north, BorderLayout.NORTH);  
    this.add(south, BorderLayout.SOUTH);  
    this.add(east, BorderLayout.EAST);  
    this.add(west, BorderLayout.WEST);  
    this.add(center, BorderLayout.CENTER);  
  
    this.setVisible(true);  
}
```

}

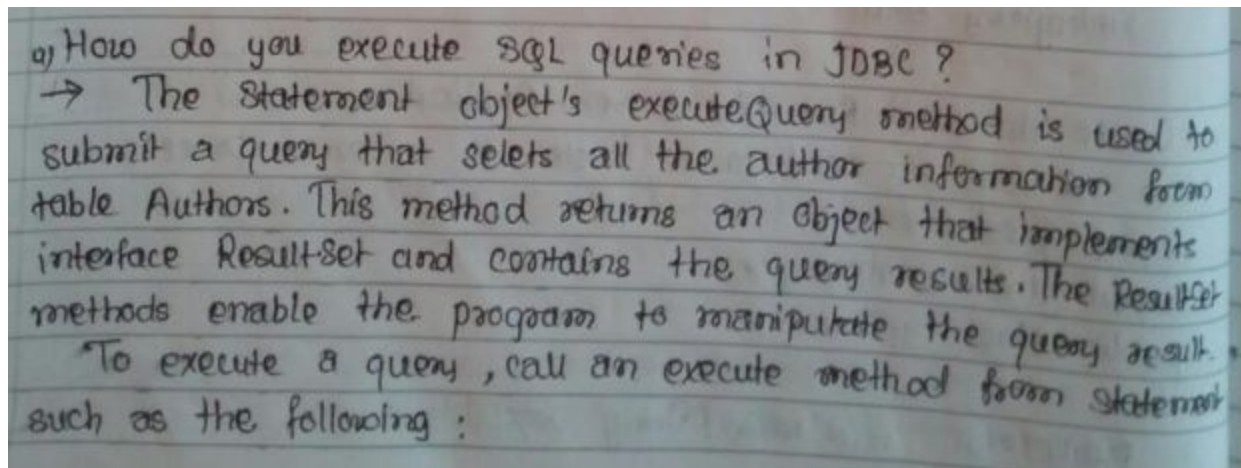
7. Why is it important to handle events? List any five event classes .

8. What is JDBC? How do you execute SQL statements in JDBC?

Java programs communicate with databases and manipulate their data using the Java Database Connectivity(JDBC). A JDBC driver enables Java applications to connect to a database in a particular DBMS and allows you to manipulate that database using the JDBC API. The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.

JDBC helps to write Java applications that manage these three programming activities:

- Connect to a data source, like a database
- Send queries and update statements to the database
- Retrieve and process the results received from the database in answer to your query



Q) How do you execute SQL queries in JDBC ?
→ The Statement object's executeQuery method is used to submit a query that selects all the author information from table Authors. This method returns an object that implements interface ResultSet and contains the query results. The ResultSet methods enable the program to manipulate the query result. To execute a query, call an execute method from Statement such as the following :

.execute : Returns true if the first object that the query returns is a ResultSet object. Use this method if the query could return one or more ResultSet objects. Retrieve the ResultSet objects returned from the query by repeatedly calling Statement.getResultSet.

.executeQuery : Returns one ResultSet object

.executeUpdate : Returns an integer representing the number of rows affected by the SQL statement.

for example,

CoffeeTables.viewTable executed a Statement object with the following code :

```
ResultSet rs = stmt.executeQuery(query);
```

Source : [Dipak Timilsaina](#)

9. Discuss the process of sending email messages using Java.

```
package com.company.bhishanwork;
```

```
import java.net.PasswordAuthentication;
```

```
import java.util.Properties;
```

```
import javax.mail.Session;
```

```
import javax.mail.PasswordAuthentication;
```

```
import javax.mail.Message;
```

```
import javax.mail.Transport;
```

```
import javax.mail.internet.InternetAddress;
```

```
import javax.mail.internet.MimeMessage;
```

```
public class EMAILJAVA {
```

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

```

        final String username = "bbhishan@mail.com" ;

        final String password = "pass" ;

        Properties props = new properties();

        props.put("mail.smtp.auth", true);

        props.put("mail.smtp.starttls.enable", true);

        props.put("mail.smtp.host", "smtp.google.com");

        props.put("mail.smtp.port", "587");


        Session session = Session.getInstance(props, new javax.mail.Authenticator{

            protected PasswordAuthentication getPasswordAuthentication(){

                return new PasswordAuthentication(username, password);

            }

        });


        Message message = new MimeMessage(session);

        message.setFrom(new InternetAddress("a@a.com"));

        message.setRecipients(Message.RecipientType.To, InternetAddress.parse("b@b.com"));

        message.setSubject("Hello");

        message.setText("Hey test");

        Transport.send(message);

    }

}

//ALTERNATE

import java.util.*;

import javax.mail.*;

```

```
import javax.mail.internet.*;

import javax.activation.*;

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

        String to = "sonoojaiswal1988@gmail.com";//change accordingly

        String from = "sonoojaiswal1987@gmail.com";change accordingly

        String host = "localhost";//or IP address


        //Get the session object

        Properties properties = System.getProperties();

        properties.setProperty("mail.smtp.host", host);

        Session session = Session.getDefaultInstance(properties);


        //compose the message

        try{

            MimeMessage message = new MimeMessage(session);

            message.setFrom(new InternetAddress(from));

            message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));

            message.setSubject("Ping");

            message.setText("Hello, this is example of sending email ");


            // Send message

            Transport.send(message);
```



```
System.out.println("message sent successfully....");
```

```
}catch (MessagingException mex) {mex.printStackTrace();}  
}  
}
```

10. What is JSP? Discuss with suitable example.

JavaServer Pages simplify the delivery of dynamic Web content. They enable Web application programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting. Custom-tag libraries are a powerful feature of JSP that allows Java developers to hide complex code for database access and other useful services for dynamic Web pages in custom tags. Web sites use these custom tags like any other Web page element to take advantage of the more complex functionality hidden by the tag. Thus, Web-page designers who are not familiar with Java can enhance Web pages with powerful dynamic content and processing capabilities.

The classes and interfaces that are specific to JavaServer Pages programming are located in packages

javax.servlet.jsp and javax.servlet.jsp.tagext.

A Simple JSP Example

JSP expression inserting the date and time into a Web page.

```
//test.jsp
```

```
<html>
```

```
    <head>
```

```
        <meta http-equiv = "refresh" content = "60" />
```

```
        <title>A Simple JSP Example</title>
```

```
        <style type = "text/css">
```

```
            .big { font-family: helvetica, arial, sans-serif;
```

```

        font-weight: bold;

        font-size: 2em; }

</style>

</head>

<body>

    <p class = "big">Simple JSP Example</p>

    <table style = "border: 6px outset;">

        <tr>

            <td style = "background-color: black;">

                <p class = "big" style = "color: cyan;">

                    <!-- JSP expression to insert date/time -->

                    <%= new java.util.Date() %>

                </p>

            </td>

        </tr>

    </table>

</body>

</html>

```

11. What is InetAddress class? Discuss.

InetAddress class is used to represent an IP address. IP address can be 32 or 128 bits.

Methods:

public static InetAddress getByName(String host): returns an instance of InetAddress for the given host. public static InetAddress getByName(byte[] addr): returns an instance of InetAddress for the given addr. public String getHostName(): returns host name.

public String getHostName(): returns host address.

public static InetAddress getLocalhost(): returns an instance of InetAddress representing localhost.

Example:

```
import java.net.*;

public class InetAddressClass {

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

        InetAddress inetAddress = InetAddress.getByName("www.google.com");

        System.out.println("Host name = " + inetAddress.getHostName());

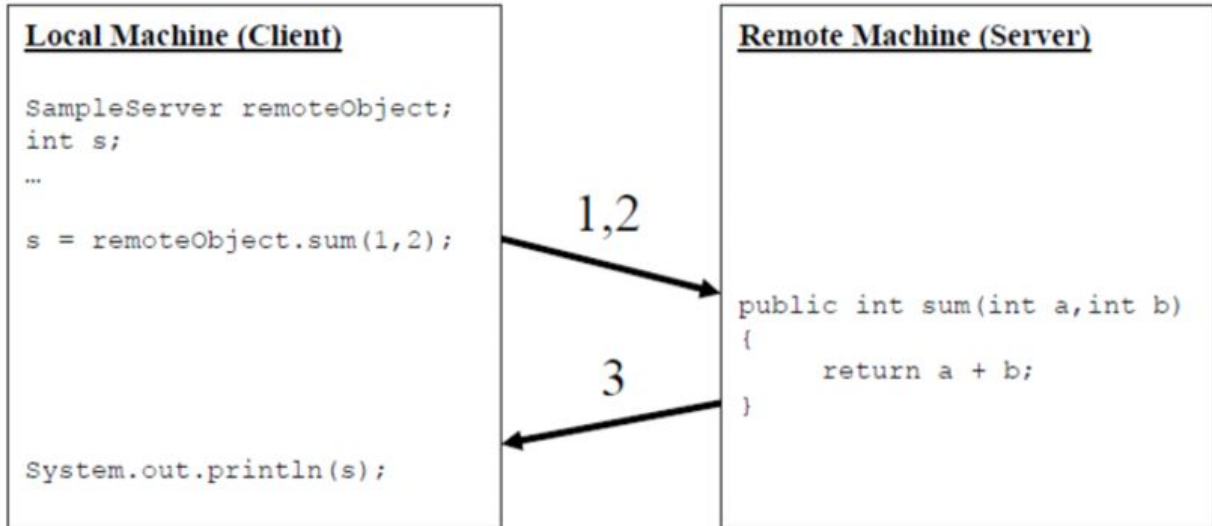
        System.out.println("IP address = " + inetAddress.getHostAddress());

    }

}
```

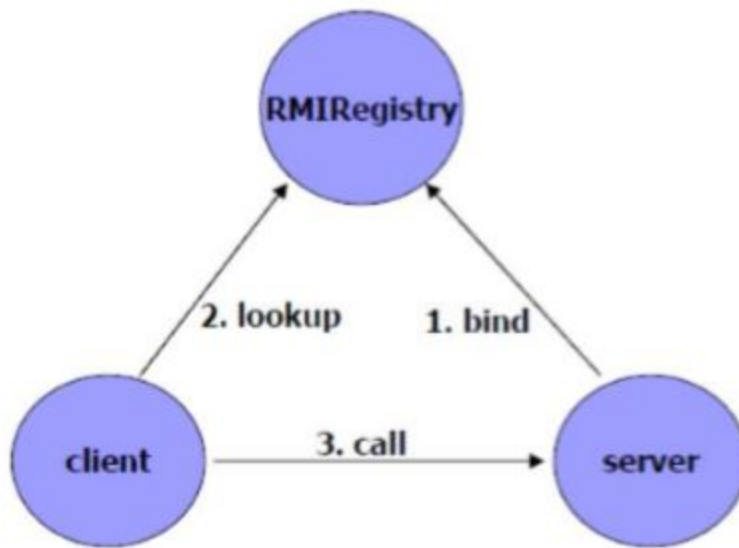
12. What is RMI? Discuss architecture of RML.

Java's Remote Method Invocation (commonly referred to as RMI) is used for client and server models. RMI is the object oriented equivalent of RPC (Remote procedure call). The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM. RMI provides for remote communication between programs written in the Java programming language. RMI differs from CORBA (Common Object Request Broker Architecture) in the sense that CORBA is language/machine independent whereas RMI is designed for machines running JVM, i.e. JVM dependent. The advantage of using RMI is that some operations can be executed significantly faster on the remote system than on the local client computer



RMI Architecture

To send a message to a remote server object, the client object has to find the remote object. Remote objects are listed in the RMI Registry. The Registry is a program that binds remote objects with a textual name and runs on a known port (default 1099). The RMI Registry service can be started within the server JVM, via the `LocateRegistry.createRegistry()` API. The client, before performing invoking a remote method, must first lookup the registry to obtain access to the remote object. After that, the client can call methods on the remote objects



A remote object is one whose instances can be used remotely. There are two parts to defining a remote class: its interface and the actual class itself. The interface definition must be public and it must extend the interface `java.rmi.Remote`. Further every method in the interface must declare that it throws `java.rmi.RemoteException`. The Remote class must implement a Remote interface and it should extend `java.rmi.server.UnicastRemoteObject` class. Objects of this class exist in the address space of the server and can be invoked remotely. Objects in the remote class may have methods that are not in its remote interface. However, these can only be invoked locally. The interface must be available to both client and server. The class should only be on the server.

13. Write short notes on:

a. Interface

Interfaces are syntactically similar to classes but they lack instance variables, and their methods are declared without any body. In practice, this means you can define interfaces which do not make assumptions about how they are implemented. Once it is defined, any number of classes can implement any number of interfaces. Using interface, you can specify what a class must do, but now how it does it.

To implement an interface, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation. By providing the interface keyword, Java allows you to fully utilize 'one interface, multiple methods' aspect of polymorphism. Interfaces are designed to support dynamic method resolution at run time. Normally, in order for a method to be called from one class to another,

both classes need to be present in compile time so that Java compiler can check and ensure that the method signatures are compatible.

An interface is defined much like a class. This is the general form of an interface.

```
access interface name{

    return-type method-name1(parameter-list);

    return-type method-name2(parameter-list);

    type final-varname1 = value; type final-varname2 = value;

    //..

    return-type method-nameN(parameter-list);

    type final-varnameN = value;

}
```

Here, access is either public or, not used. When no access specifier is included, then default access results, and the interface is only able to other members of the package in which it is declared. When it is declared as public, the interface can be used by any other code. name is the name of the instance, and can be any valid identifier.

Notice that the methods which are declared have no bodies. They end with a semicolon after the parameter list. They are, essentially, abstract methods; there can be no default implementation of any method specified within an interface. Each class that includes an interface must implement all of the methods.

Variables can be declared inside of interface declarations. They are implicitly final and static, meaning they cannot be changed by implementing class. They must also be initialized with a constant value. All methods and variables are implicitly public if the interface, itself, is declared as public.

b. Servlet

Servlets are small programs that execute on the server side of a Web connection. Just as applets dynamically extend the functionality of a Web browser, servlets dynamically extend the functionality of a Web server. A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. The `javax.servlet` and `javax.servlet.http` packages provide

interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines lifecycle methods.