

Tribhuvan University
Institute of Science and Technology

2069



Bachelor Level/ Fourth Year/ Seventh Semester/ Science
Computer Science and Information Technology (CSc. 401)
(Advanced Java Programming)

Full Marks: 60

Pass Marks: 24

Time: 3 hours.

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.

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: Refer to Solution of TU Exam Question 2070.

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.

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: 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. The following code example demonstrates the mechanism of executing SQL statements in JDBC:

```
import java.sql.*;

public class ExecuteSQLStamentsInJDBC {

    public static void main(String[] args) throws SQLException, ClassNotFoundException {

        //load the driver
        Class.forName("com.mysql.jdbc.Driver");

        //create the connection
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost/library", "root",
"vertrigo");

        //create the statement
        Statement stat = conn.createStatement();

        //SQL string formation for INSERT
        String sql1 = "INSERT INTO BOOK VALUES (5, 'JAVA', 360)";
        //INSERT into database
        stat.executeUpdate(sql1);

        //SQL string formation for UPDATE
        String sql2 = "UPDATE BOOK SET BOOKNAME='ADVANCED JAVA' WHERE
BOOKID=5";
        //UPDATE the database
        stat.executeUpdate(sql2);

        //SQL string formation for DELETE
        String sql3 = "DELETE FROM BOOK WHERE BOOKID=3";
        //DELETE from database
        stat.executeUpdate(sql3);

        //SQL string formation for SELECT query
        String sql4 = "SELECT * FROM BOOK";
        //Procedure for SELECTing data from database
        ResultSet rs = stat.executeQuery(sql4);
        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\t");
}

while (rs.next()) {
    System.out.println();
    for (int i = 1; i <= no_of_columns; i++) {
        System.out.print(rs.getObject(i) + "\t\t");
    }
}
}
}

```

Here, we suppose that we have an existing table named “book” in a database named “library”. Also we suppose that the table “book” has three fields named bookID, bookName and bookPrice of data types integer, varchar and float respectively and we also assume that the table contains some records. Here, bookId will serve as a unique key for the table “book”.

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");
    }
}

```

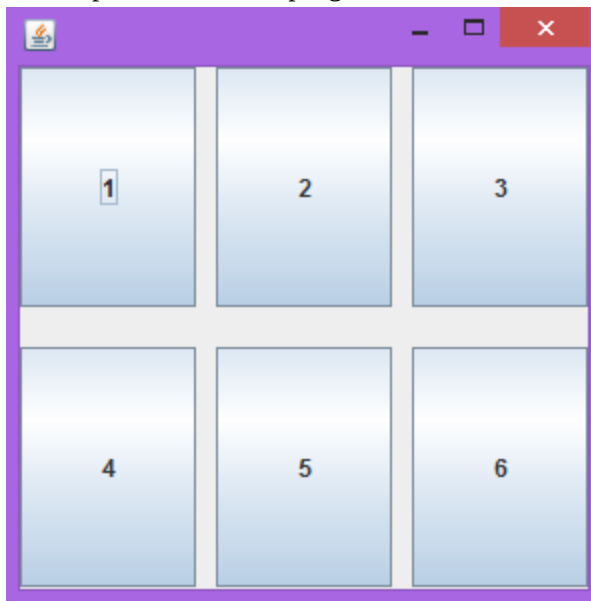
```

        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();
    }
}

```

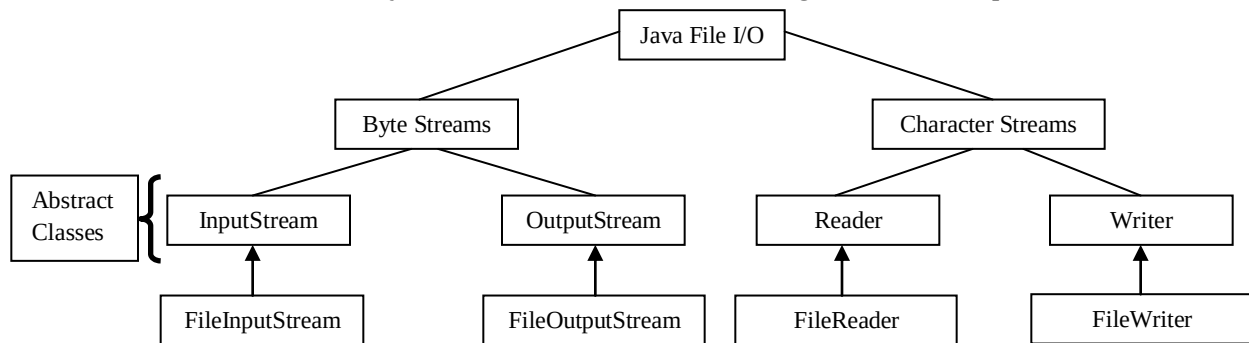
The output of the above program will be as:



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);
        }
    }
}
```

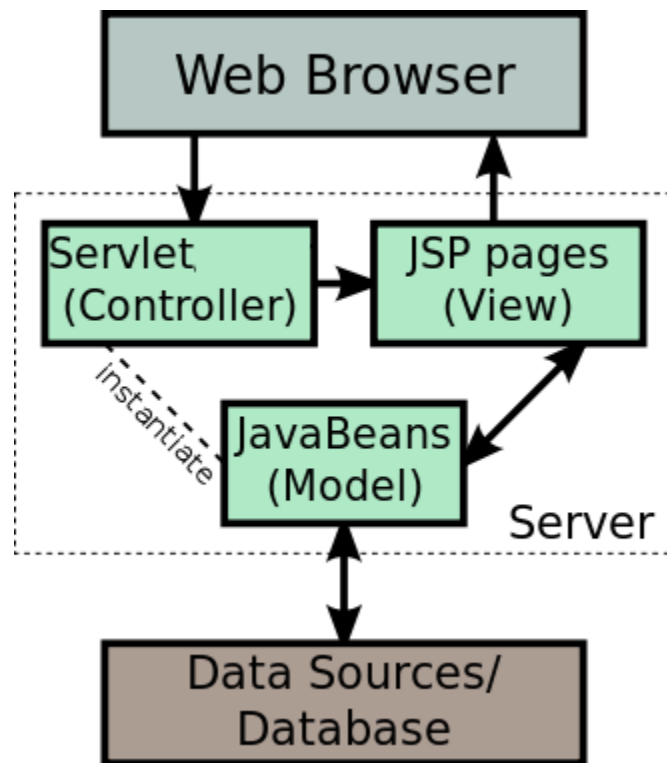
9. What is JSP? Differentiate it with servlet.

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.



In short, we can say 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 (`<% ... %>`).

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.

The Java code to perform client-server communication using UDP sockets is given below:

```

//TheServer.java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class TheServer {
    static int serverPort = 998;
    static int clientPort = 999;
    static final int buffer_size = 1024;
    static DatagramSocket ds;
    static byte buffer[] = new byte[buffer_size];

    public static void Server() throws Exception {
        int length = 0;
        while (true) {
            int c = System.in.read();
            if (c != '\n') {
                buffer[length++] = (byte) c;
            } else {
                ds.send(new DatagramPacket(buffer, length, InetAddress.getLocalHost(), clientPort));
                length = 0;
            }
        }
    }

    public static void main(String[] args) throws Exception {
        ds = new DatagramSocket(serverPort);
        Server();
    }
}

```

```

//TheClient.java
import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class TheClient {
    static int clientPort = 999;
    static final int buffer_size = 1024;
    static DatagramSocket ds;
    static byte buffer[] = new byte[buffer_size];

    public static void Client() throws Exception {
        while (true) {
            DatagramPacket p = new DatagramPacket(buffer, buffer.length);
            ds.receive(p);
        }
    }
}

```

```

        System.out.println(new String(p.getData()));
    }
}

public static void main(String[] args) throws Exception {
    ds = new DatagramSocket(clientPort);
    Client();
}
}

```

Here the server can send a datagram packet of maximum 1024 bytes length to the client via UDP socket from port 998 to the localhost's port 999. The client can receive the datagram packet from server on its client port 999 and display on the screen.

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);
    JTextField t2 = new JTextField(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: Refer to Solution of TU Exam Question 2070.

Tribhuvan University
Institute of Science and Technology



2070

Bachelor Level/ Fourth Year/ Seventh Semester/ Science

Full Marks: 60

Computer Science and Information Technology (CSc. 401)

Pass Marks: 24

(Advanced Java Programming)

Time: 3 hours.

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 = "";

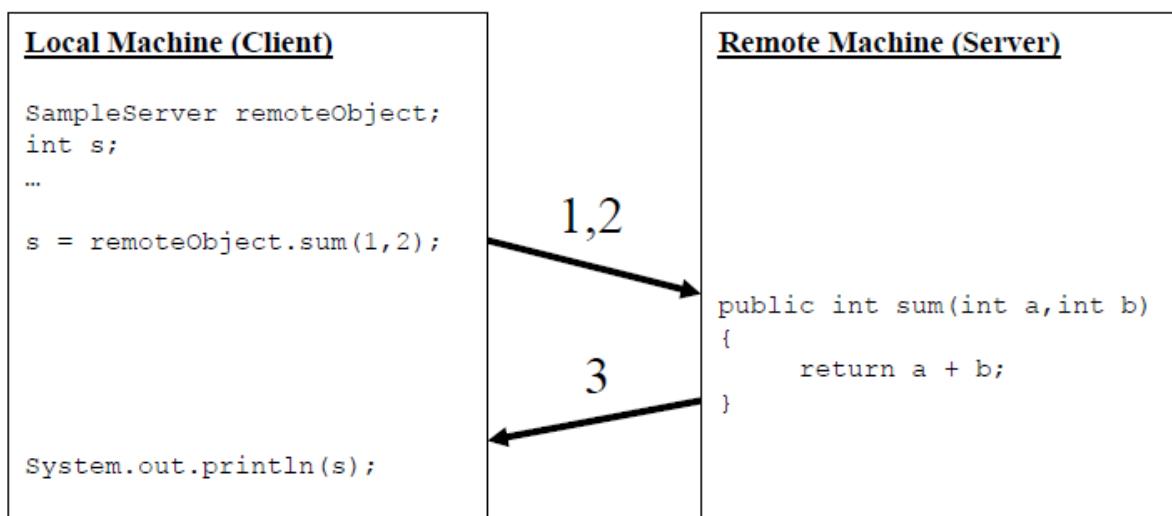
    public MultiplyUsingSwing() {
        setLayout(new FlowLayout());
        setSize(250, 250);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(l1);
        add(t1);
        add(l2);
        add(t2);
        t3.setEditable(false);
        add(t3);
        add(b);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

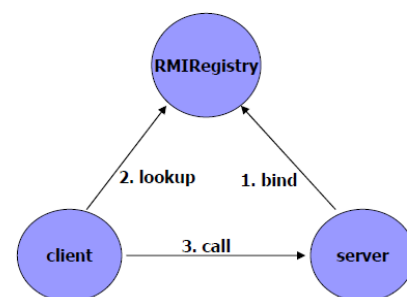
3. What is RMI? How can you use RMI to develop a program that runs in different machines? Discuss with suitable example. (2+2+6)

Answer: 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.

In the code example below, ClientMain is a program that invokes methods on a remote object and ServerMain is a program that owns the remote object (local object to the server). The ClientMain program invokes two remote methods with strings as parameters and a ServerMain program returns one remote method with hello appended to the first string and another remote method giving the square root of the given number provided as string. The name of the project folders, name of packages inside project folders and name of classes/interfaces are as given in the table below:

Name of project folder (machine)	Name of package	Name of Interface/Class
SimpleRMIClient	com.san.rmi	Message.java
SimpleRMIClient	com.san.clientmain	ClientMain.java
SimpleRMIServer	com.san.rmi	Message.java
SimpleRMIServer	com.san.rmi	MessageImpl.java
SimpleRMIServer	com.san.servermain	ServerMain.java

The contents of Message.java interface are:

```
//Message.java
package com.san.rmi;

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

public interface Message extends Remote {
    String sayHello(String name) throws RemoteException;
    double saySquareRoot(String name) throws RemoteException;
}
```

The contents of MessageImpl.java class are:

```
//MessageImpl.java
package com.san.rmi;

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

public class MessageImpl extends UnicastRemoteObject implements Message {

    public MessageImpl() throws RemoteException { }

    @Override
    public String sayHello(String name) throws RemoteException {
        return "Hello " + name;
    }
}
```

```

@Override
public double saySquareRoot(String name) throws RemoteException {
    int i = Integer.parseInt(name);
    return Math.sqrt(i);
}
}

```

The contents of ServerMain.java class are:

```

//ServerMain.java
package com.san.servermain;

import com.san.rmi.MessageImpl;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ServerMain {

    private void startServer() {
        try {
            // RMI registry is RMI server //create registry on port 1099
            Registry registry = LocateRegistry.createRegistry(1099);

            // create a new service named myMessage with remote object
            registry.bind("myMessage", new MessageImpl());
        } catch (Exception e) {
        }
        System.out.println("system is ready");
    }

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

```

The contents of ClientMain.java class are:

```

//ClientMain.java
package com.san.clientmain;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import com.san.rmi.Message;

public class ClientMain {

```

```

private void doTest() {
    try {
        // fire to localhost port 1099
        Registry myRegistry = LocateRegistry.getRegistry("127.0.0.1", 1099);

        // search for myMessage service in the registry
        //gets remote object i.e. instance of MessageImpl
        Message impl = (Message) myRegistry.lookup("myMessage");

        // call server's method
        String s = impl.sayHello("Lok Prakash Pandey");

        double d = impl.saySquareRoot("5");

        System.out.println("Message Sent and Response Received");

        System.out.println(s + "\n" + d);
    } catch (Exception e) {
    }
}

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

```

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:

- The servlet is initialized by calling the **`init()`** method.
- The servlet calls **`service()`** method to process a client's request.
- The servlet is terminated by calling the **`destroy()`** method.
- 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 Thread1 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:

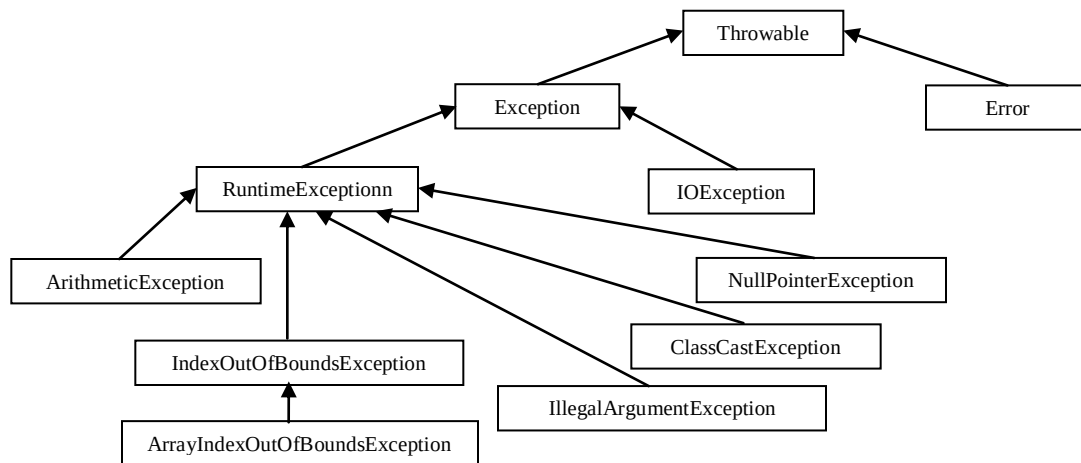


Fig: Exception Hierarchy

Some of the exception classes in Java are discussed below:

- i. **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:
- `public ArrayIndexOutOfBoundsException()`
 - `public ArrayIndexOutOfBoundsException(int index)`
 - `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());
        }
    }
}
```

- iii. `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:
- `public NullPointerException()`
 - `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);
        }
    }
}
```

- iv. `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 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.

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.

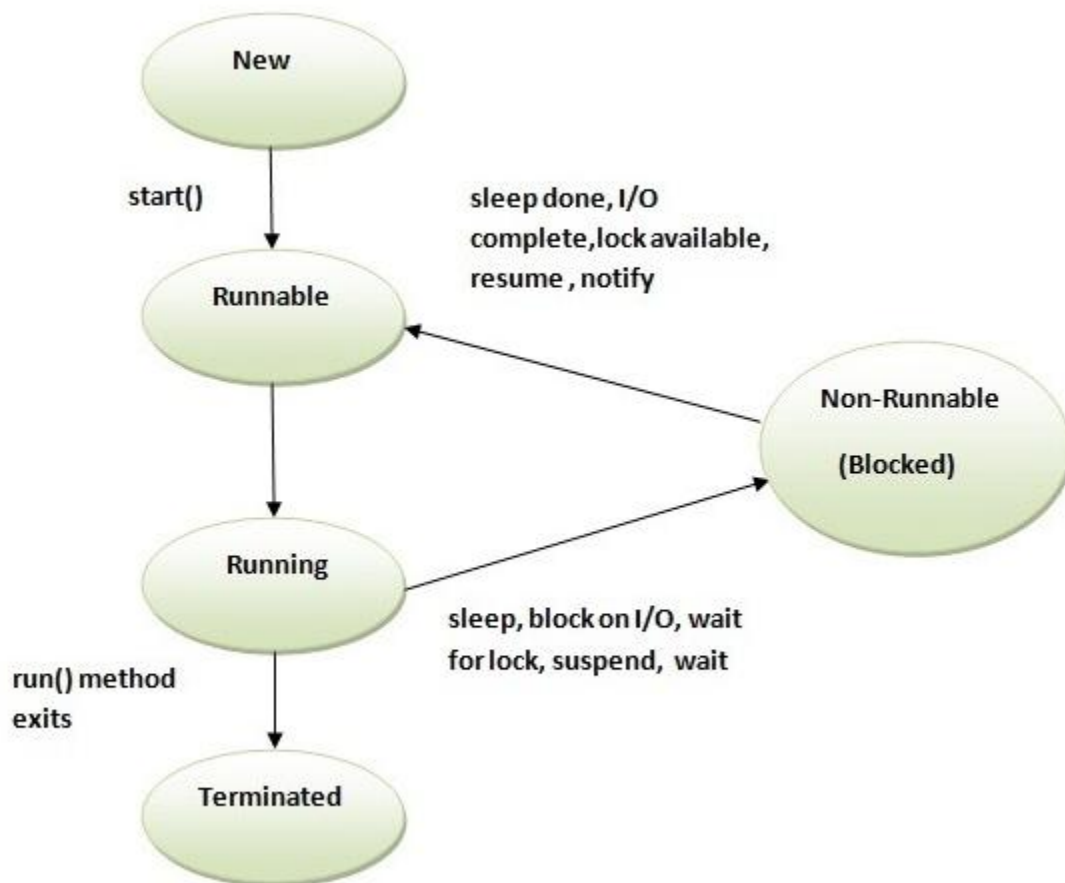


Fig: Thread States

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: Refer to TU Exam Solution 2069 and 2070.

Section B (8*5=40)

Attempt any eight questions.

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

Answer: Refer to TU Exam Solution 2070.

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

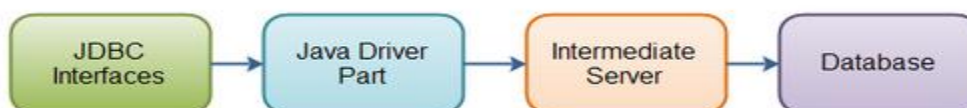
A type 2 JDBC driver is like a type 1 driver, except the ODBC part is replaced with a native code part instead. The native code part is targeted at a specific database product. Here is an illustration of a type 2 JDBC driver:



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:



Type 4 JDBC driver.

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.

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: GroupLayout is a layout manager that hierarchically groups components in order to position them in a Container. GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently. We do not need to worry about the vertical dimension when defining the horizontal layout, and vice versa, as the layout along each axis is totally independent of the layout along the other axis. When focusing on just one dimension, we only have to solve half the problem at one time. This is easier than handling both dimensions at once. This means that each component needs to be defined twice in the layout. If we forget to do this, GroupLayout will generate an exception. The following program demonstrates using GroupLayout.

```
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);
```

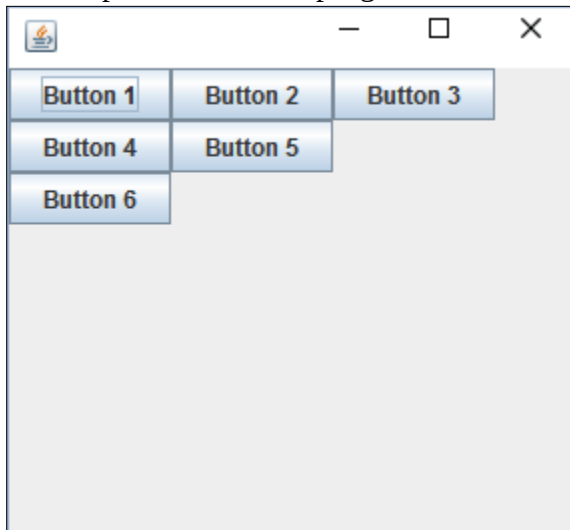
```

        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: Refer to TU Exam Solution 2070.

9.) What is TCP socket? Differentiate it with UDP socket. (2+3)

Answer: Refer to TU Exam Solution 2070.

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, MouseMotionListener
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: Refer to TU Exam Solution 2070.

12.) What is RMI? Differentiate it with CORBA. (2+3)

Answer: 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 i.e. RMI is a Java-specific technology. However, CORBA (Common Object Request Broker Architecture) has implementations for many languages. We can use CORBA to share objects between programs written in different languages (e.g. C++ and Java). Further, CORBA uses IDL (Interface Definition Language) to separate interface from implementation whereas RMI just uses Java interfaces. Also, since CORBA is not tied to a particular language, the data types do not always map exactly to the types used by our programming language (e.g. a long in IDL is an int in Java), however with RMI the data types are exactly mapped. These are the differences between competing distributed systems technologies: RMI and CORBA.

13.) Write a simple JSP program to display "Tribhuvan University" 10 times. (5)

Answer: Refer to TU Exam Solution of 2070.

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

Subject: Advance Java Programming
Course No.: CSC-403

FM: 60
PM: 24
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: Refer to TU Exam Solution 2069.

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 form 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() {

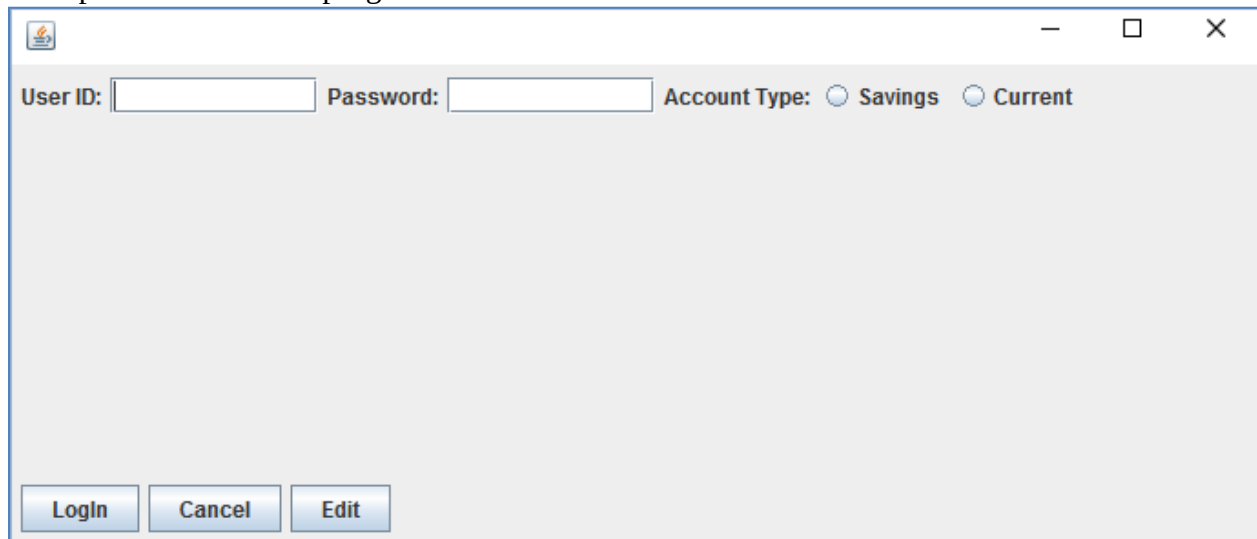
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

setSize(700,300);
setVisible(true);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

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

```

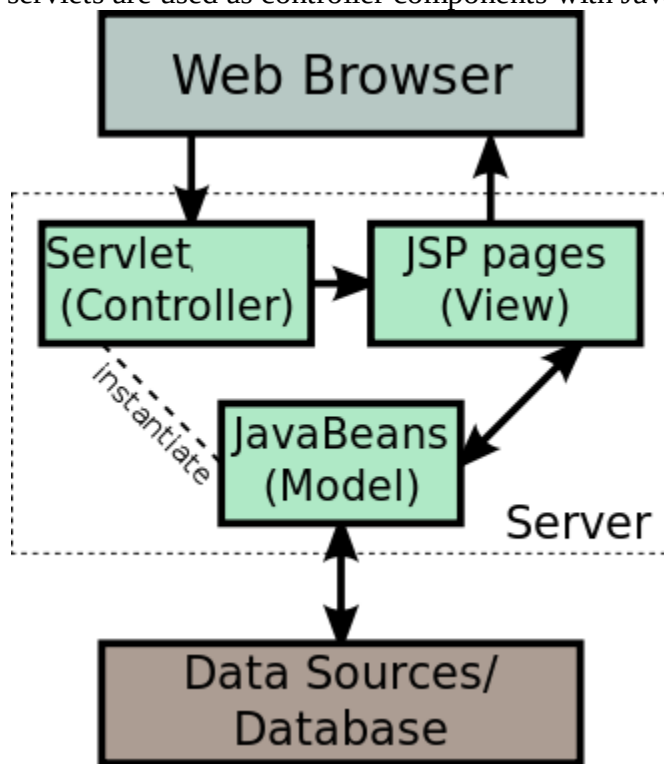
A snapshot of the above program is:



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>

//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);
    }
}
```

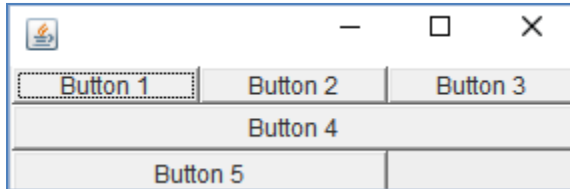
```

        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: DO IT YOURSELF!!!