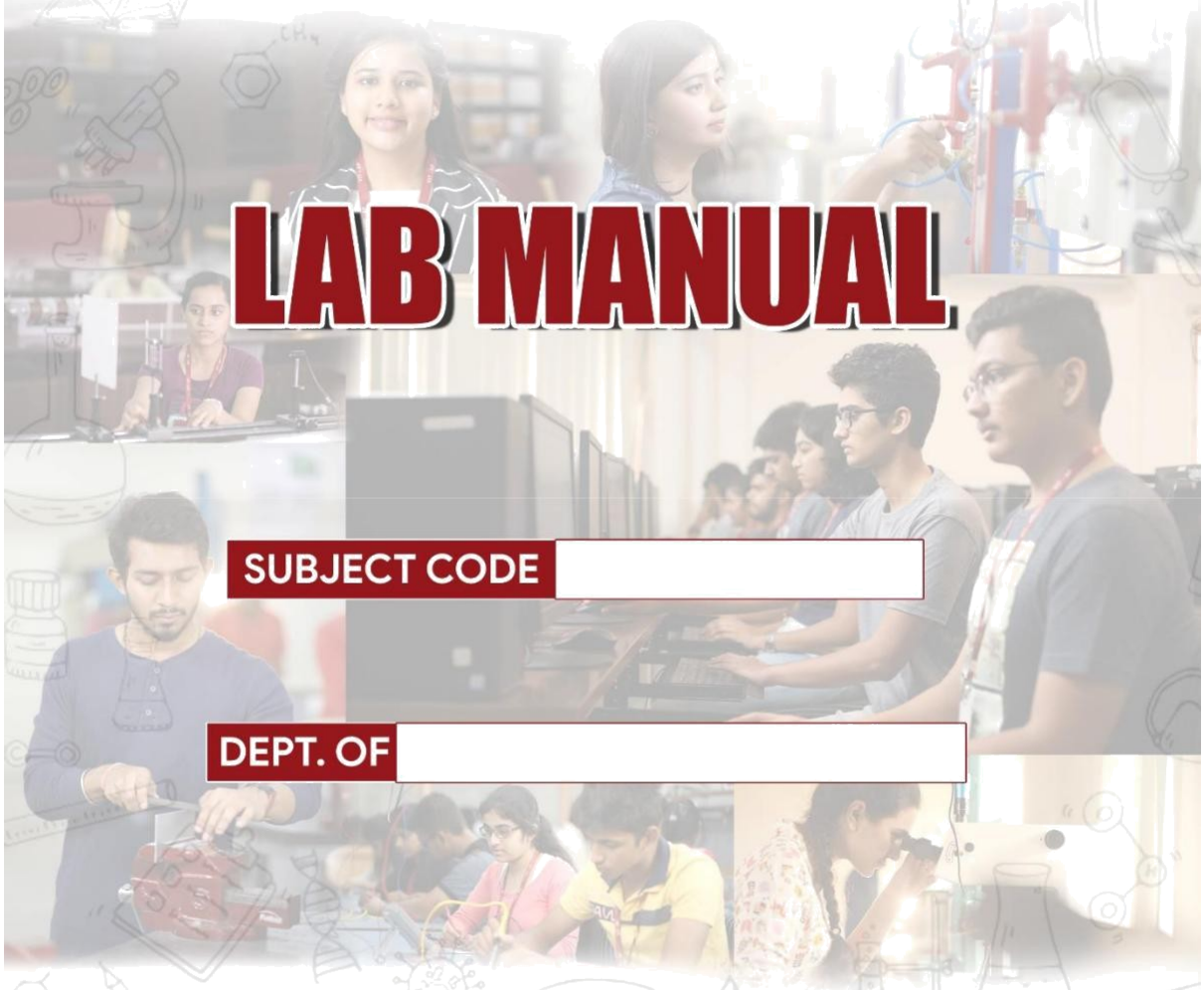
VIT AP AMARAVATI



LAB MANUAL

SUBJECT CODE

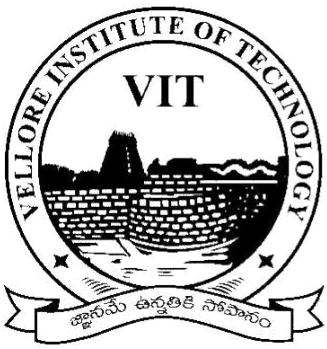
DEPT. OF

VIT-AP UNIVERSITY

Beside AP Secretariat, Amaravati - 522 237, Andhra Pradesh

Phone : 0863-2370444, Email : info@vitap.ac.in

Apply Knowledge. Improve Life!™



VIT®
AP

AMARAVATI

School of Computer Science and Engineering

B.Tech (Computer Science and Engineering)

CSE3011 – Network Programming

Lab Manual

2019-20

List of Exercises

1. Write a Java program to read the IP address and Host Name of the machine, Collect the IP address of the machines connected to the LAN.
2. Demonstrate Remote method Invocation for simple arithmetic operations.
3. Write a java program to get the Host name of the IP address, Display the data of the given URL in command prompt.
4. Write a program to get the complete URL path of the given URL, Read the information of the URL file and Get the information of the given URL file.
5. Retrieve the details of the given URL Using GET method using java.
6. Exhibit using Java to read the data from the HTTP server.
7. Illustrate using Java to write the data to the HTTP server.
8. Program to implement all HTTP methods.
9. Demonstration of Least Recently Used(LRU) Cache using Java.
10. Write a Socket program for client / server to upload the file from the client and to retrieve from server by any client.
11. Programs to exhibit three classes for Events handling in java.
12. Justify through java code: Secure Client / Server communication.
13.Program to demonstrate session management using cookies through servlets.
14.UDP Client / Server chat.
15.Multicasting a message communication using UDP Client/Server.

1. Write a Java program to read the IP address and Host Name of the machine, Collect the IP address of the machines connected to the LAN.

Java.net.Inet4Address class in Java:

This class extends InetAddress class and represents an IPv4 address. It provides methods to interpret and display useful information about ip addresses. Methods of this class takes input in 4 formats:

1. **d.d.d.d :** When this format is used as input, each of the given values are assigned to 4 bytes of the IP address from left to right.
2. **d.d.d :** When this format is used as input, the last part is interpreted as 16 bit number and assigned to the rightmost 2 bytes as the host address. This is generally used for specifying class-B address.
3. **d.d :** When this format is used as input, the last part is interpreted as 24 bit number and assigned to the rightmost 3 bytes as the host address. This is generally used for specifying class-A address.
4. **d :** When this format is used as input, the given value is directly stored as network address without any rearrangement.

Methods:

1. getAddress() : returns raw IP address of this InetAddress object as an array. The order in which bytes appear in array are same as in IP address i.e. getAddress[0] will contain highest order byte.

   ```
   Syntax : public byte[] getAddress()
   ```

2. getHostAddress() : returns IP address in textual form.

   ```
   Syntax :public String getHostAddress()
   ```

3. isAnyLocalAddress() : returns true if this address represents a local address.

   ```
   Syntax :public boolean isAnyLocalAddress()
   ```

4. isLinkLocalAddress() : returns true if this address is a link local address.

   ```
   Syntax :public boolean isLinkLocalAddress()
   ```

5. isLoopbackAddress() : returns true if this address is a loopback address.

   ```
   Syntax :public boolean isLoopbackAddress()
   ```

6. isMCGlobal() : returns true if this multicast address has global scope.

   ```
   Syntax :public boolean isMCGloabal()
   ```

7. isMCLinkLocal() : returns true if this multicast address has link scope.

   ```
   Syntax :public boolean isMCLinkLocal()
   ```

8. isMCNodeLocal() : returns true if this multicast address has node scope.

```
Syntax :public boolean isMCNodeLocal()
```

9. isMCOrgLocal() : returns true if this multicast address has organisation scope.

```
Syntax :public boolean isMCOrgLoacal()
```

10. isMCSiteLocal() : returns true if this multicast address has site scope.

```
Syntax :public boolean isMCSiteLocal()
```

11. isMulticastAddress() : returns true if this address is an IP multicast address. Multicast addresses have 1110 as their first 4 bits.

```
Syntax :public boolean isMulticastAddress()
```

12. isSiteLocalAddress(): returns true if this address is a site local address.

```
Syntax :public boolean isSiteLocalAddress()()
```

13. hashCode() : returns the hashcode associated with this address object.

```
Syntax : public int hashCode()
```

14. equals() : returns true if this ip address is same as that of the object specified. Equals() method don't consider host names while comparing and only consider IP address associated.

```
Syntax : public boolean equals(Object obj)
Parameters :
obj : object to compare with
```

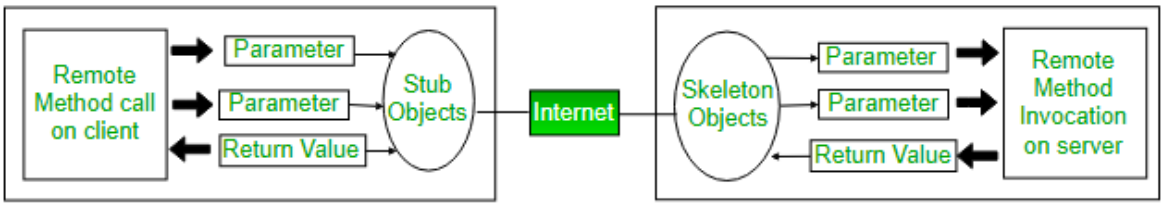
Sample Code :

```
import java.net.*;
import java.io.*;
import java.util.*;
public class GetIPAddress{
  public static void main(String[] args )throws IOException {
    BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
    System.out.print("Enter the host name: ");
    String hostname = input.readLine();
          ServerSocket s=new ServerSocket(6125);
    try {
      InetAddress inetadd = InetAddress.getByName(hostname);
      System.out.println("IP address is: " + inetadd.getHostAddress());
          Socket soc=s.accept();
          DataOutputStream out=new DataOutputStream(soc.getOutputStream());
          out.writeBytes("Server Date: " + (new Date()).toString() + "\n");
BufferedReader in=new BufferedReader(new InputStreamReader(soc.getInputStream()  ));
       System.out.println(in.readLine());
          out.close();
          soc.close();
    }
    catch(UnknownHostException e ){
```

```
      System.out.println("Could not find IP address for: " + hostname);
    }
  }
}
```

2. Demonstrate Remote method Invocation for simple arithmetic operations.

Remote Method Invocation (RMI) is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side). RMI creates a public remote server object that enables client and server side communications through simple method calls on the server object.



Working of RMI

Working of RMI:

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server side).

Stub Object:

The stub object on the client machine builds an information block and sends this information to the server. The block consists of

- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

Skeleton Object:

The skeleton object passes the request from the stub object to the remote object. It performs following tasks

- It calls the desired method on the real object present on the server.
- It forwards the parameters received from the stub object to the method.

Steps to implement Interface:

- Defining a remote interface
- Implementing the remote interface

- Creating Stub and Skeleton objects from the implementation class using rmic (rmi complier)
- Start the rmiregistry
- Create and execute the server application program
- Create and execute the client application program.

Sample Code:

Interface:

```
import java.rmi.*;
 import java.rmi.server.*;
 public interface servinterface extends Remote
 {
     public double add(double a,double b) throws RemoteException;
     public double sub(double a,double b) throws RemoteException;
     public double mul(double a,double b) throws RemoteException;
     public double div(double a,double b) throws RemoteException;
     public double mod(double a,double b) throws RemoteException;
 }
```

Implementation Class:

```
 import java.rmi.*;
  import java.rmi.server.*;
  public class servimpl extends UnicastRemoteObject implements servinterface
  {
    public servimpl() throws RemoteException
    {
      super();
    }
    public double add(double a,double b)
    {
      return(a+b);
    }
    public double sub(double a,double b)
    {
      return(a-b);
    }
    public double mul(double a,double b)
    {
      return(a*b);
    }
    public double div(double a,double b)
    {
      return(a/b);
    }

    public double mod(double a,double b)
```

```
  {
    return(a%b);
  }
 }
```

Server Class:

```
import java.rmi.*;
 import java.rmi.server.*;
 import java.rmi.registry.*;
 public class server extends servimpl
 {
     public server() throws RemoteException
     {
     }
     public static void main(String args[]) throws RemoteException
     {
         try
         {
             Registry reg = LocateRegistry.createRegistry(9999);
             servimpl s = new servimpl();
             reg.rebind("Server",s);
             System.out.println("Server is ready!");
         }
         catch(Exception e)
         {
             System.out.println(e);
         }
     }
 }
```

Client Class:

```
 import java.rmi.*;
 import java.rmi.server.*;
 import java.rmi.registry.*;
 public class client
 {
     public static void main(String args[])throws RemoteException
     {
         try
         {
             int a = Integer.parseInt(args[0]); int b = Integer.parseInt(args[1]);
             Registry reg=LocateRegistry.getRegistry("127.0.0.1",9999);
             servinterface i = (servinterface)reg.lookup("Server");
             System.out.println("First number: "+a);  System.out.println("Second number: "+b);
             System.out.println("Add:\t"+i.add(a,b));
```

```
            System.out.println("Sub:\t"+i.sub(a,b));
            System.out.println("Mul:\t"+i.mul(a,b));
            System.out.println("Div:\t"+i.div(a,b));
            System.out.println("Mod:\t"+i.mod(a,b));
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

3. Write a java program to get the Host name of the IP address, Display the data of the given URL in command prompt.

The URL class is the gateway to any of the resource available on the internet. A Class URL represents a Uniform Resource Locator, which is a pointer to a “resource” on the World Wide Web. A resource can point to a simple file or directory, or it can refer to a more complicated object, such as a query to a database or to a search engine

Components of a URL:

1. Protocol: HTTP is the protocol here
2. Hostname: Name of the machine on which the resource lives.
3. File Name: The path name to the file on the machine.
4. Port Number: Port number to which to connect (typically optional).

Constructors for URL class:

1. URL(String address) throws MalformedURLException: It creates a URL object from the specified String.
2. URL(String protocol, String host, String file): Creates a URL object from the specified protcol, host, and file name.
3. URL(String protocol, String host, int port, String file): Creates a URL object from protocol, host, port and file name.
4. URL(URL context, String spec): Creates a URL object by parsing the given spec in the given context.
5. URL(String protocol, String host, int port, String file, URLStreamHandler handler):- Creates a URL object from the specified protocol, host, port number, file, and handler.
6. URL(URL context, String spec, URLStreamHandler handler):- Creates a URL by parsing the given spec with the specified handler within a specified context.

Sample Code:

```
import java.io.*;
import java.net.*;
        public class DisplayData
        {
                public static void main(String[] args)
                {
                        try
                        {
```

```
                    URL url=new URL("http://www.google.co.in");
                    URLConnection urlcon=url.openConnection();
                    InputStream stream=urlcon.getInputStream();
                    int i;
                    while((i=stream.read())!=-1){
                    System.out.print((char)i);
                }
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
}
```

4. Write a program to get the complete URL path of the given URL, Read the information of the URL file and Get the information of the given URL file.

Refer No.3 for explanations.

Sample Code:

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

public class URLReadFile
{
     public static void main(String[] args) throws IOException
    {
            try
            {
             URL url = new URL("file:///home/Gokul/Course.html");
  BufferedReader buff = new BufferedReader(new InputStreamReader(url.openStream()));
        String str;
        while((str = buff.readLine()) != null)
            {
             System.out.println("\n");
             System.out.println(str);
            }
          buff.close();
         }
         catch(IOException e)
        {
          System.err.println(e);
        }
    }
}
```

5. Retrieve the details of the given URL Using GET method using java.

Refer No. 3 for explanations.

Sample Code:

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class URLInformation
{
	public static void main(String args[]) throws Exception
	{
		int c;
		URL url = new URL("https://vitap.ac.in/academic-calendar/");
		URLConnection con = url.openConnection();
		System.out.println("");
		System.out.println("Date: " + new Date(con.getDate()));
		System.out.println("Content-Type: " + con.getContentType());
		System.out.println("Expires: " + con.getExpiration());
		System.out.println("Last-Modified: " + new Date(con.getLastModified()));
		int len = con.getContentLength();
		System.out.println("Content-Length: " + len);
		if (len > 0)
		{
			System.out.println("=== Content ===");
			InputStream input = con.getInputStream();
			int i = len;
			while(((c = input.read()) != -1) && (-i > 0))
			{
				System.out.print((char) c);
			}
			input.close();
```

```
                }
                else
                {
                        System.out.println("No Content Available");
                }
        }
}
```

6. Exhibit using Java to read the data from the HTTP server.

Refer No.3 for basics.
Procedure to read flat text database files from a web server by looking at portions of my "To-Do List" code is listed below. Also, to read and display it in an AWT List component.

Write to server directly to the filesystem of the web server is not possible, following code shows how to simulate an old-fashioned HTML Post method by writing data to a CGI program on the HTTP server. Then, let the CGI program write the data to the Server filesystem.

Sample Code:

```
void getToDoList ()
{
  try
  {
    URL     url;
    URLConnection      urlConn;
    DataInputStream    dis;
    url = new URL("http://webserver.our-intranet.com/ToDoList/ToDoList.txt");
    urlConn = url.openConnection();
    urlConn.setDoInput(true);
    urlConn.setUseCaches(false);
    dis = new DataInputStream(urlConn.getInputStream());
```

```
   String s;
   toDoList.clear();
   while ((s = dis.readLine()) != null)
   {
     toDoList.addItem(s);
   }
     dis.close();
   }
   catch (MalformedURLException mue) {}
   catch (IOException ioe) {}
  }
}
```

Applet Code:

```
import java.awt.*;
import java.applet.*;

public class ToDoList extends Applet
{

  java.awt.Button      addButton;
  java.awt.Label       newLabel;
  java.awt.TextField   addTextField;
  java.awt.Label       toDoLabel;
  java.awt.List        toDoList;
  java.awt.Button      refreshButton;

  public void init() {

  super.init();
  setLayout(null);
  addNotify();
  resize(410,336);

  addButton = new java.awt.Button("Add to List");
  addButton.reshape(290,222,71,23);
  addButton.setFont(new Font("Dialog", Font.PLAIN, 12));
  add(addButton);

  newLabel = new java.awt.Label("New To-Do Item");
  newLabel.reshape(25,202,100,22);
  newLabel.setFont(new Font("Dialog", Font.BOLD, 12));
  add(newLabel);

  addTextField = new java.awt.TextField();
  addTextField.reshape(24,225,243,21);
  add(addTextField);
```

```
  toDoLabel = new java.awt.Label("To-Do List");
  toDoLabel.reshape(24,24,69,22);
  toDoLabel.setFont(new Font("Dialog", Font.BOLD, 12));
  add(toDoLabel);

  toDoList = new java.awt.List(0,false);
  add(toDoList);
  toDoList.reshape(25,48,238,123);

  refreshButton = new java.awt.Button("Refresh");
  refreshButton.reshape(287,49,71,23);
  refreshButton.setFont(new Font("Dialog", Font.PLAIN, 12));
  add(refreshButton);

  // "getToDoList()" not created yet:
  // getToDoList();

  }  // end of init() method

}  // end of ToDoList class
```

7. Illustrate using Java to write the data to the HTTP server.

Refer No.6 for explanations

Sample Code:

```
void postNewItem () {
  try {

    URL             url;
    URLConnection     urlConn;
    DataOutputStream   dos;
    DataInputStream    dis;

    url = new URL("http://webserver.our-intranet.com/cgi-bin/AddToDoItem");
    urlConn = url.openConnection();
    urlConn.setDoInput(true);
    urlConn.setDoOutput(true);
    urlConn.setUseCaches(false);
    urlConn.setRequestProperty ("Content-Type", "application/x-www-form-urlencoded");

    dos = new DataOutputStream (urlConn.getOutputStream());
    String message = "NEW_ITEM=" + URLEncoder.encode(addTextField.getText());
    dos.writeBytes(message);
    dos.flush();
    dos.close();

    // the server responds by saying
```

```
    // "SUCCESS" or "FAILURE"

    dis = new DataInputStream(urlConn.getInputStream());
    String s = dis.readLine();
    dis.close();

    if (s.equals("SUCCESS")) {
      toDoList.addItem(addTextField.getText());
      addTextField.setText("");
    } else {
      addTextField.setText("Post Error!");
    }

  } // end of "try"

  catch (MalformedURLException mue) {
    addTextField.setText("mue error");
  }
  catch (IOException ioe) {
    addTextField.setText("IO Exception");
  }

}  // end of postNewItem() method
```

8. Program to exhibit the capabilities of all HTTP methods.

The set of common methods for HTTP/1.1 is defined below and this set can be expanded based on requirements. These method names are case sensitive and they must be used in uppercase.

S.N.	Method and Description
1	GET The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2	HEAD Same as GET, but transfers the status line and header section only.
3	POST A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.
4	PUT

Replaces all current representations of the target resource with the uploaded content.

DELETE

5 Removes all current representations of the target resource given by a URI.

CONNECT

6 Establishes a tunnel to the server identified by a given URI.

OPTIONS

7 Describes the communication options for the target resource.

TRACE

8 Performs a message loop-back test along the path to the target resource.

Sample Code:

```
import apache.http.client.methods;

import java.net.URI;

import org.apache.http.annotation.NotThreadSafe;

/**
 * HTTP PUT method.
 * <p>
 * The HTTP PUT method is defined in section 9.6 of
 * <a href="http://<any>">:
 * <blockquote>
 * The PUT method requests that the enclosed entity be stored under the
 * supplied Request-URI. If the Request-URI refers to an already
 * existing resource, the enclosed entity SHOULD be considered as a
 * modified version of the one residing on the origin server.
 * </blockquote>
 * </p>
```

```
 *
 * @since 4.0
 */

public class HttpPut extends HttpEntityEnclosingRequestBase {

    public final static String METHOD_NAME = "PUT";

    public HttpPut() {
        super();
    }

    public HttpPut(final URI uri) {
        super();
        setURI(uri);
    }

    /**
     * @throws IllegalArgumentException if the uri is invalid.
     */
    public HttpPut(final String uri) {
        super();
        setURI(URI.create(uri));
    }

    @Override
    public String getMethod() {
        return METHOD_NAME;
    }

}
```

9. Demonstration of Least Recently Used(LRU) Cache using Java.

The LRU caching scheme is to remove the least recently used frame when the cache is full and a new page is referenced which is not there in cache.

Two data structures shall be used to implement an LRU Cache.

1. Queue which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size). The most recently used pages will be near front end and least recently pages will be near the rear end.
2. A Hash with page number as key and address of the corresponding queue node as value.

When a page is referenced, the required page may be in the memory. If it is in the memory, we need to detach the node of the list and bring it to the front of the queue. If the required page is not in memory, we bring that in memory. In simple words, we add a

new node to the front of the queue and update the corresponding node address in the hash. If the queue is full, i.e. all the frames are full, we remove a node from the rear of the queue, and add the new node to the front of the queue.

Sample Code:

```
import java.util.Deque;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Iterator;
public class LRUCache {
   // store keys of cache
   static Deque<Integer> dq;
   // store references of key in cache
   static HashSet<Integer> map;
   // maximum capacity of cache
   static int csize;

   LRUCache(int n)
   {
      dq = new LinkedList<>();
      map = new HashSet<>();
      csize = n;
   }

   /* Refers key x with in the LRU cache */
   public void refer(int x)
   {
      if (!map.contains(x)) {
         if (dq.size() == csize) {
            int last = dq.removeLast();
            map.remove(last);
         }
      }
      else {
         /* The found page may not be always the last element, even if it's an
            intermediate element that needs to be removed and added to the start
            of the Queue */
         int index = 0, i = 0;
         Iterator<Integer> itr = dq.iterator();
         while (itr.hasNext()) {
            if (itr.next() == x) {
               index = i;
               break;
            }
            i++;
         }
         dq.remove(index);
      }
      dq.push(x);
```

```
        map.add(x);
    }

    // display contents of cache
    public void display()
    {
        Iterator<Integer> itr = dq.iterator();
        while (itr.hasNext()) {
            System.out.print(itr.next() + " ");
        }
    }
     public static void main(String[] args)
    {
        LRUCache ca = new LRUCache(4);
        ca.refer(1);
        ca.refer(2);
        ca.refer(3);
        ca.refer(1);
        ca.refer(4);
        ca.refer(5);
        ca.display();
    }
}
```

10. Write a Socket program for client to upload the file to server.

The client program has three responsibilities which must be fulfilled in the code. To make the code simple, instead of try-catch blocks, Exception is thrown in the main() method signature.

Following are the three responsibilities of the client.

- To take input for file name from keyboard. Remember, this file should exist on server. For this, it uses input stream.

- The file read from keyboard should be sent to the server. For this client uses output stream.

- The file contents sent by the server, the client should receive and print on the console.

Sample Code:

```
import java.net.*;
import java.io.*;
public class ContentsClient
{
  public static void main( String args[ ] ) throws Exception
  {
    Socket sock = new Socket( "127.0.0.1", 4000);
    // reading the file name from keyboard. Uses input stream
    System.out.print("Enter the file name");
    BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
    String fname = keyRead.readLine();
    // sending the file name to server. Uses PrintWriter
    OutputStream  ostream = sock.getOutputStream( );
    PrintWriter pwrite = new PrintWriter(ostream, true);
    pwrite.println(fname);
    // receiving the contents from server.  Uses input stream
    InputStream istream = sock.getInputStream();
    BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));
    String str;
    while((str = socketRead.readLine())  !=  null) // reading line-by-line
    {
       System.out.println(str);
    }
    pwrite.close(); socketRead.close(); keyRead.close();
  }
}

import java.net.*;
import java.io.*;
public class ContentsClient
{
  public static void main( String args[ ] ) throws Exception
  {
    Socket sock = new Socket( "127.0.0.1", 4000);
    // reading the file name from keyboard. Uses input stream
    System.out.print("Enter the file name");
    BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
    String fname = keyRead.readLine();
     // sending the file name to server. Uses PrintWriter
    OutputStream  ostream = sock.getOutputStream( );
    PrintWriter pwrite = new PrintWriter(ostream, true);
    pwrite.println(fname);
      // receiving the contents from server.  Uses input stream
    InputStream istream = sock.getInputStream();
```

```
    BufferedReader socketRead = new BufferedReader(new InputStreamReader(istream));
     String str;
    while((str = socketRead.readLine())  !=  null) // reading line-by-line
    {
       System.out.println(str);
    }
    pwrite.close(); socketRead.close(); keyRead.close();
  }
}
```

11. Programs to exhibit three classes for Events handling in java.

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- Button
    - public void addActionListener(ActionListener a){}
- MenuItem
    - public void addActionListener(ActionListener a){}
- TextField
    - public void addActionListener(ActionListener a){}
    - public void addTextListener(TextListener a){}
- TextArea
    - public void addTextListener(TextListener a){}
- Checkbox
    - public void addItemListener(ItemListener a){}
- Choice
    - public void addItemListener(ItemListener a){}
- List
    - public void addActionListener(ActionListener a){}
    - public void addItemListener(ItemListener a){}

Sample Code:

```
import java.awt.*;
import java.awt.event.*;
class AEvent extends Frame implements ActionListener{
TextField tf;
AEvent(){

//create components
tf=new TextField();
tf.setBounds(60,50,170,20);
Button b=new Button("click me");
b.setBounds(100,120,80,30);

//register listener
b.addActionListener(this);//passing current instance

//add components and set size, layout and visibility
add(b);add(tf);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
tf.setText("Welcome");
}
public static void main(String args[]){
new AEvent();
}
}
```

12. Justify through java code: Secure Client / Server communication.

Java chat application over a network. The key features are to be:

Confidentiality: Information to be secure during transfer.

Integrity: Ensure data not tampered during the transfer.

Authentication: Ensure the identity of sender.

Sample Server Code:

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.nio.file.*;
import javax.crypto.Mac;
//Server side of the chat
public class Server {
    //reads users input and listens on the specific port.
    public static void main(String[] args) throws Exception {
        int portNumber = 8080;
        Security security = null;
        if (args.length != 1) {
            System.out.println("Usage: java server <port>");
            return;
        }
        security = new Security();
        //If they want authentication, verify their password.
        if (security.authentication) {
            PasswordTools.verifyPassword(Paths.get("server_private", "pass"));
        }
        portNumber = Integer.parseInt(args[0]);
        //Setup server
        boolean isOver = false;
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Binding to port " + portNumber);
        ServerSocket server = new ServerSocket(portNumber);
        System.out.println("Server started: " + server);
        //Loops and creates new ClientHandler objects, allows for server to persist even when
client closes connection.
        while (!isOver) {
            Socket socket = server.accept();
            System.err.println("Open session message recieved, comparing protocol.");
            ClientHandler handler = new ClientHandler(socket, input, security);
            handler.handleClient();
        }
        server.close();
        input.close();
```

```
    }
}
//Class used to handle new clients accessing the socket
class ClientHandler {
    //Needed variables
    static BufferedReader input;
    static OutputStream serverStream;
    static InputStream clientStream;
    static Socket socket;
    static boolean connected;
    static Security security;
    static byte[] key = null;
    static Cryptography crypto;
    static Communication communication;
    //Constructor for established socket and I/O
    public ClientHandler(Socket socket, BufferedReader input, Security security) {
        this.socket = socket;
        this.input = input;
        this.security = security;
        this.connected = true;
    }
    //Actually read and write to the client
    public static void handleClient() throws IOException {
        serverStream = socket.getOutputStream();
        clientStream = socket.getInputStream();
        crypto = new Cryptography();
        communication = new Communication();
        try {
            if (invalidProtocol(clientStream, serverStream)) {
                return;
            }
        } catch (IOException ioe) {
            System.out.println("Client closed connection.");
        }
        if (security.confidentiality || security.integrity) {
            key = ServerDiffie.doDiffie(clientStream, serverStream);
        }
        //Spins up a thread for reading from input and sending to outputstream
        Thread sendMessage = new Thread(new Runnable() {
            @Override
            public void run() {
                String send;
                byte[] encrypted;
                try {
                    while (connected) {
                        send = input.readLine();
                        if (!socket.isClosed()) {
                            //Format the message.
                            byte[] signature = crypto.sign(send.getBytes(),
                                    Paths.get("server_private", "private.der"),
```

```
                                    security.authentication);
                            byte[] mac = crypto.generateMAC(send.getBytes(), key,
security.integrity);
                            byte[] message = crypto.encrypt(send.getBytes(), key,
                                    security.confidentiality);
                            byte[] finalMessage = communication.format(message, signature,
mac);
                            serverStream.write(finalMessage);
                        }
                    }
                } catch (IOException ioe) {
                    System.out.println("Client closed connection.");
                }
            }
        });
        //Spin up a thread to read from inputstream and write to command line.
        Thread readMessage = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    while (true) {
                        byte[] msg = new byte[16 * 1024];
                        int count = clientStream.read(msg);
                        msg = Arrays.copyOf(msg, count);
                        String message = communication.handleMessage(msg,
                                Paths.get("server_private", "publicClient.der"), crypto, key,
security);
                        System.out.println("client: " + message);
                        if (message.equals("bye")) {
                            System.out.println("Client closed connection.");
                            disconnect();
                            connected = false;
                            break;
                        }
                    }
                } catch (IOException ioe) {
                    System.out.println("Client closed connection.");
                }
            }
        });
        sendMessage.start();
        readMessage.start();
    }
    //Clean up streams.
    public static void disconnect() {
        try {
            serverStream.close();
            clientStream.close();
            socket.close();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
    //Recieve protocol from client and validate that it is the same as servers.
    public static boolean invalidProtocol(InputStream clientStream, OutputStream
serverStream) throws IOException {
        byte[] msg = new byte[16 * 1024];
        int count = clientStream.read(msg);
        String s = new String(msg, 0, count, "US-ASCII");
        String errorLog = "invalid security protocol, dropping connection.";
        if ((s.contains("a") && !security.authentication) || (!s.contains("a") &&
security.authentication)) {
            serverStream.write("invalid security protocol, authentication not matching. re-
establish connection."
                    .getBytes());
            System.out.println(errorLog);
            return true;
        }
        if ((s.contains("i") && !security.integrity) || (!s.contains("i") && security.integrity)) {
            serverStream.write(
                    "invalid security protocol, integrity not matching. re-establish
connection.".getBytes());
            System.out.println(errorLog);
            return true;
        }
        if ((s.contains("c") && !security.confidentiality) || (!s.contains("c") &&
security.confidentiality)) {
            serverStream.write("invalid security protocol, confidentiality not matching. re-
establish connection."
                    .getBytes());
            System.out.println(errorLog);
            return true;
        }
        String reply = "Valid protocol.";
        if (security.confidentiality || security.integrity) {
            reply += "Beginning DH.";
        }
        System.out.println(reply);
        serverStream.write(reply.getBytes());
        return false;
    }
}
```

13.Program to demonstrate session management using cookies through servlets.

Session simply means a particular interval of time.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

1. public void addCookie(Cookie ck):method of HttpServletResponse interface is used to add cookie in response object.

2. public Cookie[] getCookies():method of HttpServletRequest interface is used to return all the cookies from the browser.

Sample Code:

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response
    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
    out.close();
    }catch(Exception e){System.out.println(e);}
  }
}
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    Cookie ck[]=request.getCookies();
    out.print("Hello "+ck[0].getValue());
   out.close();
        }catch(Exception e){System.out.println(e);}
    }
}
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
```

```
  <servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
 <servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
  <servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
  </web-app>
```

14.UDP Client / Server chat.

DatagramSockets are Java's mechanism for network communication via UDP instead of TCP. Java provides DatagramSocket to communicate over UDP instead of TCP. It is also built on top of IP. DatagramSockets can be used to both send and receive packets over the Internet.

One of the examples where UDP is preferred over TCP is the live coverage of TV channels. In this aspect, we want to transmit as many frames to live audience as possible not worrying about the loss of one or two frames. TCP being a reliable protocol add its own overhead while transmission.
Another example where UDP is preferred is online multiplayer gaming. In games like counter-strike or call of duty, it is not necessary to relay all the information but the most important ones. It should also be noted that most of the applications in real life uses careful blend of both UDP and TCP; transmitting the critical data over TCP and rest of the data via UDP.

Java Datagram programming model Steps:

Creation of DatagramSocket:- First, a datagramSocket object is created to carry the packet to the destination and to receive it whenever the server sends any data. To create a datagramSocket following constructors can be used:

protected DatagramSocket DatagramSocket():

Syntax: public DatagramSocket()

throws SocketException

Creates a datagramSocket and binds it to any available

port on local machine. If this constructor is used, the

OS would assign any port to this socket.

protected DatagramSocket DatagramSocket(int port):-

Syntax: public DatagramSocket(int port)

throws SocketException

Parameters:

port - port to which socket is to be bound

Throws:

SocketException - If the socket cannot be bound to the

specific local port. Creates a DatagramSocket and binds

to the specified port on the local machine.

protected DatagramSocket DatagramSocket(int port, InetAddress inetaddress):-

Syntax: public DatagramSocket(int port,

InetAddress inetaddress)

throws SocketException

Parameters:

port - port to which socket is to be bound.

inetaddress - local address to which socket is to be bound.

Throws:

SocketException - If the socket cannot be bound to the

specific local port. It creates a DatagramSocket and

binds it to specified port and ip-address.

Creation of DatagramPacket: In this step, the packet for sending/receiving data via a datagramSocket is created.

Constructor to send data: DatagramPacket(byte buf[], int length, InetAddress inetaddress, int port):-

Syntax: public DatagramPacket(byte[] buf,

int offset,

int length,

SocketAddress address)

Parameters:

buf - the packet data.

offset - the packet data offset.

length - the packet data length.

address - the destination socket address.

Constructs a DatagramPacket for sending data at specified address and specified port.

Constructor to receive the data:

DatagramPacket(byte buf[], int length):-

Syntax: public DatagramPacket(byte buf[],
int length)

Parameters:

buf - the packet data.

length - the packet data length.

Constructs a DatagramPacket for recieving the data of length length in the byte array buf.

Invoke a send() or receive() call on socket object

Syntax: void send(DatagramPacket packet)
throws SocketException

Parameters:

packet - Datagrampacket to send.

Throws:

SocketException - If there is an error in binding.

IllegalArgumentException - if address is not supported by the socket.

Syntax: void receive(DatagramPacket packet)
throws SocketException

Parameters:

packet - Datagrampacket to receive from this socket.

Throws:

SocketException - If there is an error in binding.

IllegalArgumentException - if address is not supported by the socket

Sample Code:

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class udpBaseClient_2
{
	public static void main(String args[]) throws IOException
	{
		Scanner sc = new Scanner(System.in);
		// Step 1:Create the socket object for
		// carrying the data.
		DatagramSocket ds = new DatagramSocket();
		InetAddress ip = InetAddress.getLocalHost();
		byte buf[] = null;
		// loop while user not enters "bye"
		while (true)
		{
			String inp = sc.nextLine();
			// convert the String input into the byte array.
			buf = inp.getBytes();
			// Step 2 : Create the datagramPacket for sending
			// the data.
			DatagramPacket DpSend =
				new DatagramPacket(buf, buf.length, ip, 1234);
			// Step 3 : invoke the send call to actually send
			// the data.
			ds.send(DpSend);
			// break the loop if user enters "bye"
			if (inp.equals("bye"))
				break;
		}
	}
}
// Java program to illustrate Server side
// Implementation using DatagramSocket
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
public class udpBaseServer_2
{
	public static void main(String[] args) throws IOException
	{
		// Step 1 : Create a socket to listen at port 1234
		DatagramSocket ds = new DatagramSocket(1234);
```

```
            byte[] receive = new byte[65535];

            DatagramPacket DpReceive = null;
            while (true)
            {
                // Step 2 : create a DatgramPacket to receive the data.
                DpReceive = new DatagramPacket(receive, receive.length);

                // Step 3 : revieve the data in byte buffer.
                ds.receive(DpReceive);

                System.out.println("Client:-" + data(receive));

                // Exit the server if the client sends "bye"
                if (data(receive).toString().equals("bye"))
                {
                    System.out.println("Client sent bye.....EXITING");
                    break;
                }

                // Clear the buffer after every message.
                receive = new byte[65535];
            }
        }
        // A utility method to convert the byte array
        // data into a string representation.
        public static StringBuilder data(byte[] a)
        {
            if (a == null)
                return null;
            StringBuilder ret = new StringBuilder();
            int i = 0;
            while (a[i] != 0)
            {
                ret.append((char) a[i]);
                i++;
            }
            return ret;
        }
}
```

15.Multicasting a message communication using UDP Client/Server.

A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet.

Sample Code:

```
import java.net.*;
import java.io.*;
import java.util.*;
public class GroupChat
{
	private static final String TERMINATE = "Exit";
	static String name;
	static volatile boolean finished = false;
	public static void main(String[] args)
	{
		if (args.length != 2)
```

```
                System.out.println("Two arguments required: <multicast-host> <port-
number>");
        else
        {
                try
                {
                        InetAddress group = InetAddress.getByName(args[0]);
                        int port = Integer.parseInt(args[1]);
                        Scanner sc = new Scanner(System.in);
                        System.out.print("Enter your name: ");
                        name = sc.nextLine();
                        MulticastSocket socket = new MulticastSocket(port);

                        // Since we are deploying
                        socket.setTimeToLive(0);
                        //this on localhost only (For a subnet set it as 1)

                        socket.joinGroup(group);
                        Thread t = new Thread(new
                        ReadThread(socket,group,port));

                        // Spawn a thread for reading messages
                        t.start();

                        // sent to the current group
                        System.out.println("Start typing messages...\n");
                        while(true)
                        {
                                String message;
                                message = sc.nextLine();

        if(message.equalsIgnoreCase(GroupChat.TERMINATE))
                                {
                                        finished = true;
                                        socket.leaveGroup(group);
                                        socket.close();
                                        break;
                                }
                                message = name + ": " + message;
                                byte[] buffer = message.getBytes();
                                DatagramPacket datagram = new
                                DatagramPacket(buffer,buffer.length,group,port);
                                socket.send(datagram);
                        }
                }
                catch(SocketException se)
                {
                        System.out.println("Error creating socket");
                        se.printStackTrace();
                }
```

```
                        catch(IOException ie)
                        {
                                System.out.println("Error reading/writing from/to socket");
                                ie.printStackTrace();
                        }
                }
        }
}
class ReadThread implements Runnable
{
        private MulticastSocket socket;
        private InetAddress group;
        private int port;
        private static final int MAX_LEN = 1000;
        ReadThread(MulticastSocket socket,InetAddress group,int port)
        {
                this.socket = socket;
                this.group = group;
                this.port = port;
        }

        @Override
        public void run()
        {
                while(!GroupChat.finished)
                {
                        byte[] buffer = new byte[ReadThread.MAX_LEN];
                        DatagramPacket datagram = new
                        DatagramPacket(buffer,buffer.length,group,port);
                        String message;
                    try
                    {
                        socket.receive(datagram);
                        message = new
                        String(buffer,0,datagram.getLength(),"UTF-8");
                        if(!message.startsWith(GroupChat.name))
                                System.out.println(message);
                    }
                    catch(IOException e)
                    {
                        System.out.println("Socket closed!");
                    }
                }
        }
}
```