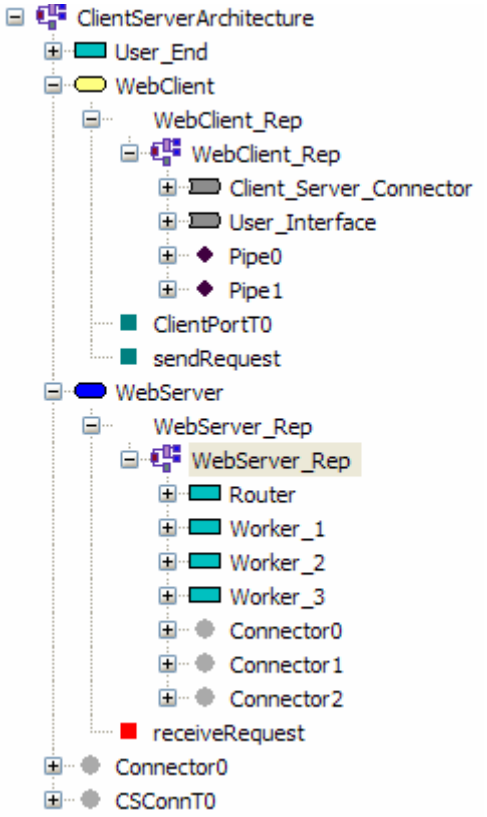
HomeWork 3

In this homework, an ArchJava application that allows one to query one's scores in a networked environment is presented.

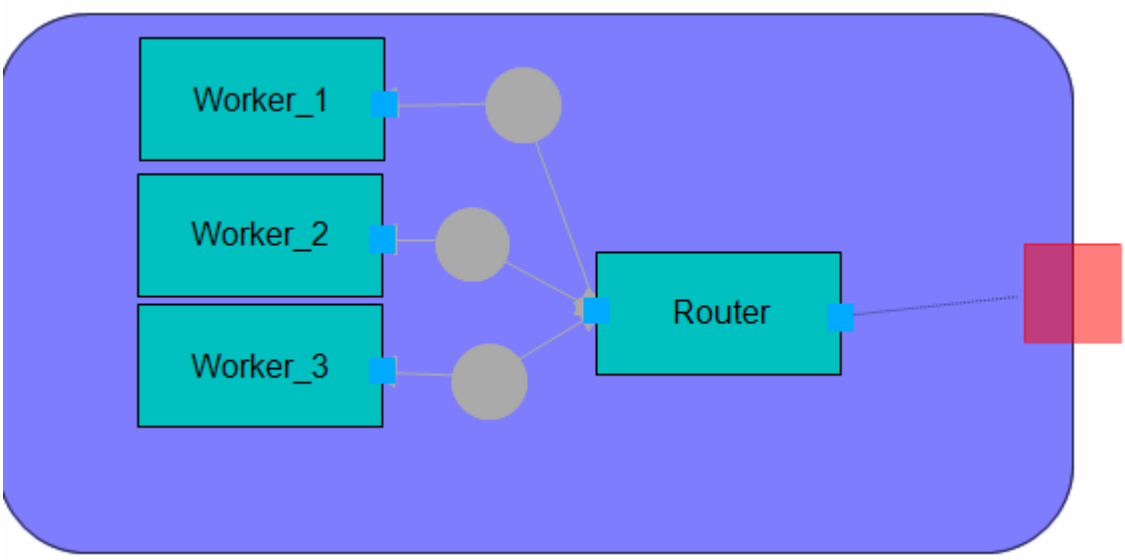
[1] Acme Description

The architecture that has been implemented using ArchJava has been documented using Acme. The following few pages illustrate the description of the architecture.

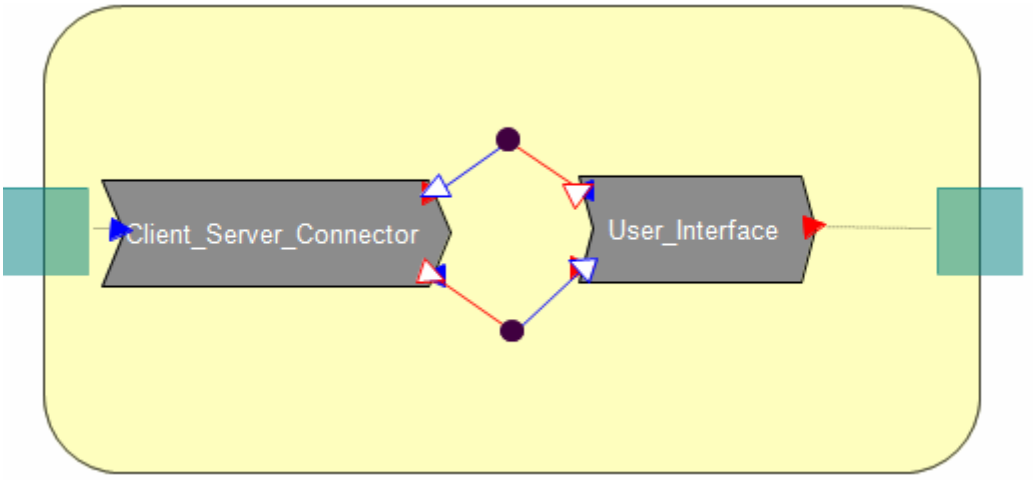


The diagram above illustrate the hierarchical structure of the ClientServerArchitecture. It is built primarily on Client-Server Model/Family. It contains the WebServer and the WebClient which constitutes the Client-Server model. It also contains an user end component.

This illustrates how the top level design looks like. The WebServer is connected to the WebClient via a single connector.



This figure shows the inner view of the WebServer. Here we see a router and how it is dynamically being attached to worker components. The creation of these worker components is the job of the router and not the WebServer.



This is the innermost layer of the WebClient. It is built on PipesAndFilter family. It contains a component that binds the address between a server and the client. And, user interface displays the input and output to the outside world.

Code for the description of the architecture using Acme:

```
import $AS_GLOBAL_PATH/families/ClientAndServerFam.acme;
import $AS_GLOBAL_PATH/families/PipesAndFiltersFam.acme;
```

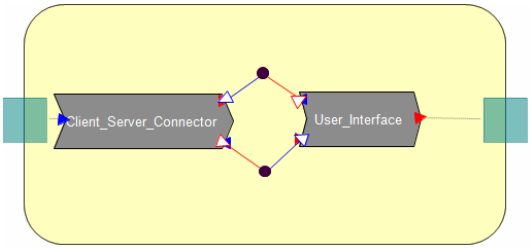
```
System ClientServerArchitecture : ClientAndServerFam = new
ClientAndServerFam extended with {
 Component WebServer : ServerT = new ServerT extended with {
  Representation WebServer_Rep = {
   System WebServer_Rep : PipesAndFiltersFam = new PipesAndFiltersFam
extended with {
    Component Router  = {
     Port Port0  = {
     }
     Port Port1  = {
     }
    }
    Component Worker_1  = {
     Port Port0  = {
     }
    }
    Component Worker_2  = {
     Port Port0  = {
     }
    }
    Component Worker_3  = {
     Port Port0  = {
     }
    }
    Connector Connector0  = {
     Role role0  = {
     }
     Role role1  = {
     }
    }
    Connector Connector1  = {
     Role role0  = {
     }
     Role role1  = {
     }
    }
    Connector Connector2  = {
     Role role0  = {
     }
     Role role1  = {
     }
    }
    Attachment Router.Port1 to Connector0.role0;
    Attachment Worker_1.Port0 to Connector0.role1;
    Attachment Worker_3.Port0 to Connector2.role0;
    Attachment Router.Port1 to Connector2.role1;
    Attachment Router.Port1 to Connector1.role1;
    Attachment Worker_2.Port0 to Connector1.role0;
   }

   Bindings {
    WebServer.receiveRequest to Router.Port0;
   }
  }
  }
  Component WebClient : ClientT = new ClientT extended with {
   Port ClientPortT0 : ClientPortT = new ClientPortT extended with {
```

```
   }
   Representation WebClient_Rep = {
    System WebClient_Rep : PipesAndFiltersFam = new PipesAndFiltersFam
extended with {
     Component Client_Server_Connector : BinaryFilter = new
BinaryFilter extended with {
      Port inputT0 : inputT = new inputT extended with {
      }
     }
     Component User_Interface : BinaryFilter = new BinaryFilter
extended with {
      Port outputT0 : outputT = new outputT extended with {
      }
     }
     Connector Pipe0 : Pipe = new Pipe extended with {
     }
     Connector Pipe1 : Pipe = new Pipe extended with {
     }
     Attachment Client_Server_Connector.output to Pipe0.source;
     Attachment User_Interface.input to Pipe0.sink;
     Attachment Client_Server_Connector.inputT0 to Pipe1.sink;
     Attachment User_Interface.outputT0 to Pipe1.source;
    }
   Bindings {
    WebClient.sendRequest to Client_Server_Connector.input;
    WebClient.ClientPortT0 to User_Interface.output;
   }
  }
 }
 Component User_End  = {
  Port Port0  = {
  }
 }
 Connector CSConnT0 : CSConnT = new CSConnT extended with {
 }
 Connector Connector0  = {
   Role role0  = {
   }
   Role role1  = {
   }
  }
 Attachment WebClient.ClientPortT0 to Connector0.role1;
 Attachment User_End.Port0 to Connector0.role0;
 Attachment WebServer.receiveRequest to CSConnT0.serverSide;
 Attachment WebClient.sendRequest to CSConnT0.clientSide;
}
```

[2] Implementation using ArchJava:

Informal reasoning: *How the implementation preserves the architectural properties and constraints*

To allow programmers to describe software architecture, ArchJava has added new language constructs to support *components*, *connections*, and *ports*. The rest of this section describes how to use these constructs to express software architectures. Throughout the discussion, we also show how the constructs work together to enforce communication integrity.



A graphical WebClient architecture is shown above that has been implemented using ArchJava. The WebClient component class contains two subcomponents—a Client_Server_Connector, and a User_Interface. This client architecture follows the well-known pipeline compiler design. The connector, and user_interface components are connected in a linear sequence, with the out port of one component connected to the in port of the next component and vice-versa for two-way or bidirectional communication.

```
public component class WebClient{
  /* Defining two components and instantiating them */
  private final component UserInterface ui =new UserInterface();
  private final component ClientServerConnector conn = new
                ClientServerConnector();
  /* Connecting the ports */
  // port out of ui is connected to port in of conn
  connect ui.out, conn.in;
  // port out of conn is connected to port in of ui
  connect conn.out, ui.in;
  public void run() {
  /* Calling the function run of component ui */
  ui.run();
  }
}
```

In ArchJava, hierarchical software architecture is expressed with *composite components*, which are made up of a number of subcomponents connected together. A *subcomponent* is a component instance nested within another component. Singleton subcomponents are typically declared with **final** fields of component type. The figure above shows how a WebClient's architecture can be expressed in ArchJava. The code shows that the user_interface communicates with the clientServerConnector using one protocol, and also implies that the userinterface does *not* communicate directly with the

clientserverconnector. A primary goal of ArchJava is to ease program understanding tasks by supporting this kind of reasoning about program structure.

The WebClient architecture shows that while the userinterface communicates with the serverconnector through ports, they do not directly communicate with each other. If this2 represented an abstract architecture to be implemented in Java code, it might be difficult to verify the correctness of this reasoning in the implementation. For example, if the serverconnector obtained a reference to the userinterface, it could invoke any of the serverconnector's methods, violating the intuition communicated by the architecture. In contrast, programmers can have confidence that an ArchJava architecture accurately represents communication between components, because the language semantics enforce communication integrity.

Communication integrity in ArchJava means that components in an architecture can only call each other's methods along declared connections between ports. Each component in the architecture can use its ports to communicate with the components to which it is connected. However, a component may not directly invoke the methods of components other than its own subcomponents, because this communication may not be declared in the architecture—a violation of communication integrity.

```
public component class WebServer {
        private final Router r = new Router();
        connect r.request, create;
        connect pattern Router.workers, Worker.serve;
        private port create {
                provides r.workers requestWorker() {
                        final Worker newWorker = new Worker(++countWorker);
                        r.workers connection = connect(r.workers,
        newWorker.serve);
                        return connection;}
        }
public void run() { r.listen(); }
        public static void main(String[] args){
                new WebServer().run();}
}
component class Router {
        public port interface workers {
                requires void httpRequest(InputStream in,OutputStream out); }
        public port request {requires this.workers requestWorker(); }
        public void listen() {
                try{    ServerSocket server = new ServerSocket(6000);
                        while (true) {Socket sock = server.accept();
                                this.workers conn = request.requestWorker();
        conn.httpRequest(sock.getInputStream(),sock.getOutputStream());
                        }
                }
                catch(Exception e) { System.out.println(e.getMessage());}
        }
}
component class Worker extends Thread {
        public port serve {
                provides void httpRequest(InputStream in,OutputStream out)
        {pBr = new BufferedReader(new InputStreamReader(in));
                 pPs = new PrintStream(out);
                 start();}
        }
        public void run() {
                try{while(true){
                        query = pBr.readLine();
                        pPs.println(answer);
                        pPs.flush();
```

```
                        answer = null;}
                }
                catch(Exception e) {
                        System.out.println(e.getMessage() + " Client No :: " + workerNo + "
has exited");}
        }
        /* other method declarations */
}
```

A snapshot of the code of a web server architecture is shown below. The Router subcomponent accepts HTTP requests and passes them on to a set of Worker components that respond. When a request comes in, the Router requests a new worker connection on its request port. The WebServer then creates a new worker and connects it to the Router. The Router assigns requests to Workers through its workers port.

The following code is the WebServer code for the implementation in ArchJava.

```
import java.awt.BorderLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.*;

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.event.*;
import java.awt.*;
import java.util.Scanner;

public component class WebServer extends JFrame{

      private int countWorker = 0;
      private final Router r = new Router();

      connect r.request, create;
      connect pattern Router.workers, Worker.serve;

      private port create {
            provides r.workers requestWorker() {
                  final Worker newWorker = new Worker(++countWorker);
                  System.out.println("Client " + countWorker + " is
attempting to connect.");
                  System.out.println("Attempt Success, connected!!");
                  System.out.println("-------------------------------
------------------------------");
      r.workers connection = connect(r.workers, newWorker.serve);
                  return connection;
            }
      }

      public WebServer(){
```

```
            System.out.println("-------------------------------------
------------------------");
            System.out.println("Server has been setup...");
            System.out.println("Waiting for connections...");
            System.out.println("-------------------------------------
------------------------");
      }

public void run() { r.listen(); }

      public static void main(String[] args){
            new WebServer().run();
      }
}

component class Router {

      public port interface workers {
            requires void httpRequest(InputStream in,OutputStream out);
      }

      public port request {
            requires this.workers requestWorker();
      }

      public void listen() {
            try{
                  ServerSocket server = new ServerSocket(6000);
                  while (true) {
                        Socket sock = server.accept();
                        this.workers conn = request.requestWorker();

      conn.httpRequest(sock.getInputStream(),sock.getOutputStream());
                  }
            }
            catch(Exception e) {
                        System.out.println(e.getMessage());
            }
      }
}

component class Worker extends Thread {

      private int workerNo;
      private int noOfQuery = 0;
      private BufferedReader pBr;
      private PrintStream pPs;
      private String query, answer;
      public port serve {
            provides void httpRequest(InputStream in,OutputStream out)
{
                  pBr = new BufferedReader(new InputStreamReader(in));
                  pPs = new PrintStream(out);
                  // this.in = in; this.out = out;
                  start();
            }
      }
```

```
	public Worker(int workerNo){
		this.workerNo = workerNo;
	}

	public void run() {

		try{
			while(true)
			{
			query = pBr.readLine();
			System.out.println("Servicing Client " + workerNo);
			System.out.println("No. of times it has accessed
Server :: " + (++noOfQuery));
			System.out.println("Received request. PIN no: " +
query);
			processLineByLine();
			System.out.println("Sending response.");
			System.out.println("Response is :: " + answer);
			System.out.println("-------------------------------
------------------------------");
			pPs.println(answer);
			pPs.flush();
			answer = null;
			}
		}
		catch(Exception e) {
			System.out.println(e.getMessage() + " Client No :: "
+ workerNo + " has exited");
			System.out.println("-------------------------------
------------------------------");
			}

	}

	public final void processLineByLine(){
		try {
			//first use a Scanner to get each line
			Scanner scanner = new Scanner( new File(
"C:\\text.txt" ));
			while ( scanner.hasNextLine() ){
			  if (processLine( scanner.nextLine() ) == 1) return;
			}
			scanner.close();
		}
		catch (IOException ex){
			System.out.println(ex.getMessage());
			answer = "Database connectivity problem. Please try
again, after some time";
		}
	}

	protected int processLine(String aLine){
		//use a second Scanner to parse the content of each line

		Scanner scanner = new Scanner(aLine);
		  scanner.useDelimiter(",");
```

```
              if ( scanner.hasNext() ){
                  String name = scanner.next();
                  String pin = scanner.next();
                  String marks = scanner.next();
                  if( pin.equals(query) ){
                  answer = aLine;
                  return 1;
                  }
            }
            else {
                  answer = "Empty or invalid line. Unable to process.";
            }
            scanner.close();
      return 0;
      }
}
```

The following code is the WebClient code for the implementation in ArchJava.

```
import java.awt.BorderLayout;
import java.text.NumberFormat;
import javax.swing.JButton;
import javax.swing.JFormattedTextField;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.*;

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.event.*;
import java.awt.*;
import java.lang.*;

/* The main component class WebClient */
public component class WebClient{

  /* Defining two components and instantiating them */

  private final component UserInterface ui =new UserInterface();
  private final component ClientServerConnector conn = new
ClientServerConnector();

  /* Connecting the ports */

  // port out of ui is connected to port in of conn
  connect ui.out, conn.in;
  // port out of conn is connected to port in of ui
  connect conn.out, ui.in;

  public void run() {

  /* Calling the function run of component ui */
  ui.run();
```

```
  }

  public static void main(String[] args){

  /* Instantiating a client object and calling the run method */
  new WebClient().run();
  }
}

/* Defination of component class UserInterface */
component class UserInterface extends JFrame implements ActionListener{

  /* Defining variables to be used */
  JFormattedTextField tf;
  JButton button;
  JTextField pintf,nmetf,mrk1,mrk2,mrk3,mrk4,mrk5;;
  JPanel panel,panel1,panel2,panel3;
  JLabel label1,label2,label3,label4,label5,label6,label7;

   /* Defination of port out */
   public port out{

   /* The sendQuery method of ClientServerConnector component is
accessed through port out */
   requires void sendQuery(String querry);

   /* The isConnected method of ClientServerConnector component is
accessed through port out of this component*/
   requires String isconnected();
   }

   /* Defination of port in */
   public port in{

   /* Function defination of answerOfQuery. This method is accessed
      from the ClientServerConnector component through its out port */
   provides void answerOfQuery(String answer,boolean isans){

      /* Splitting the answer into different strings */
      String ans[] = answer.split(",");

      /* If the answer is NULL then dialogBox is shown */
      if(ans[0].equals("null")){
         dialogBox("PIN value does not match. Please verify and re-
enter","Database Error",
JOptionPane.WARNING_MESSAGE);
         setDisable();
         repaint();
      }

      /* if the isans is FALSE meaning a server side error */
      else if (isans == false){
         dialogBox(ans[0],"Server- Client Error",
JOptionPane.ERROR_MESSAGE);
      System.exit(0);
      }
```

```
      /* if the file to be searched is not found */
        else if (answer.equals("Database connectivity problem. Please
try again, after some time")){
         dialogBox(answer,"Database Connectivity Error",
JOptionPane.WARNING_MESSAGE);
         setDisable();
         repaint();
      }

      /* if the server returns the right data set */
      else{
         setEnable();
         displayResult(ans);
      }
     }
  }

  /* Function for displaying the dialog box */
  private void dialogBox(String msg, String header, int type){

    JOptionPane.showMessageDialog(this, msg, header, type);
  }

  /* Disabling all the elements */
  private void setDisable(){

    pintf.disable();
    nmetf.disable();
    mrk1.disable();
    mrk2.disable();
    mrk3.disable();
    mrk4.disable();
    mrk5.disable();

    label1.disable();
    label2.disable();
    label3.disable();
    label4.disable();
    label5.disable();
    label6.disable();
    label7.disable();
  }

  /* Enabling all the elements */
  private void setEnable(){

    label1.enable();
    label2.enable();
    label3.enable();
    label4.enable();
    label5.enable();
    label6.enable();
    label7.enable();

    pintf.enable();
    nmetf.enable();
    mrk1.enable();
```

```
    mrk2.enable();
    mrk3.enable();
    mrk4.enable();
    mrk5.enable();
  }

  /* Displaying the Result set */
  private void displayResult(String ans[]){

    pintf.setText(ans[1]);
    nmetf.setText(ans[0]);
    mrk1.setText(ans[2]);
    mrk2.setText(ans[3]);
    mrk3.setText(ans[4]);
    mrk4.setText(ans[5]);
    mrk5.setText(ans[6]);
  }

  /* The run method */
  public void run(){

    /* Setting the properties */
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setVisible(true);

    /* Checking if connection is done properly */
    String connected = this.out.isconnected();

    /* If there is connection failure */
    if(!connected.equals("OK")){
      dialogBox(connected,"Server-Client Connection Error",
JOptionPane.ERROR_MESSAGE);
      System.exit(0);
    }
  }

  /* Setting the layout */
  private void setTheLayout() {

    getContentPane().setLayout(new GridLayout(4,1));

    /* Setting the layout */
    panel = new JPanel();
    panel1 = new JPanel();
    panel2 = new JPanel();
    panel3 = new JPanel();
    JLabel label = new JLabel("Enter the PIN :");

    pintf = new JTextField(3);
    label1 = new JLabel("PIN :");

    nmetf = new JTextField(10);
    label2 = new JLabel("NAME :");

    mrk1 = new JTextField(3);
    mrk2 = new JTextField(3);
    mrk3 = new JTextField(3);
```

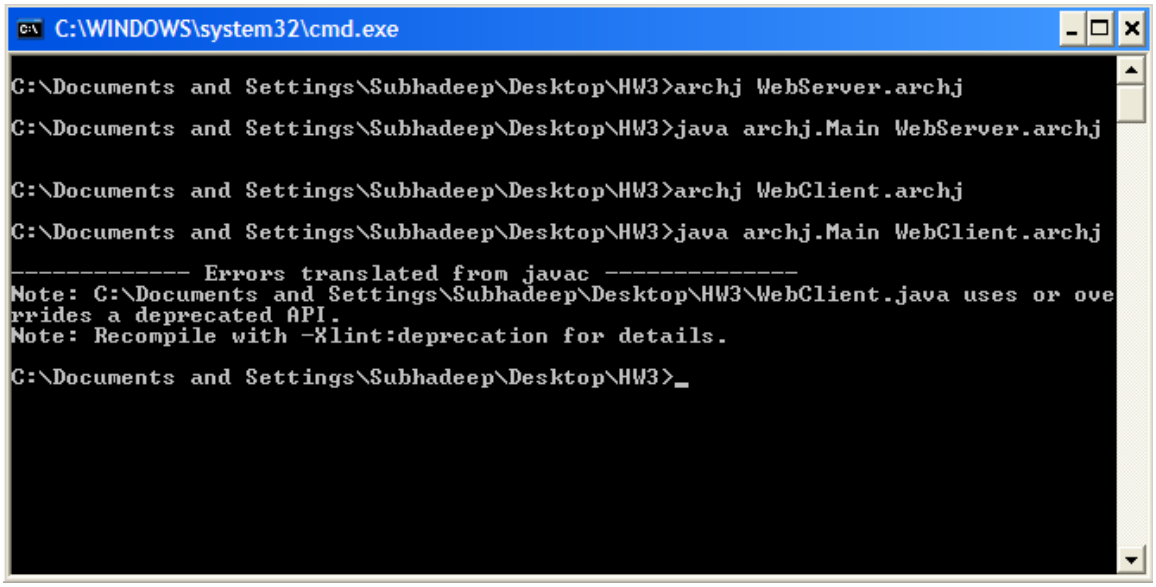
```
    mrk4 = new JTextField(3);
    mrk5 = new JTextField(3);
    label3 = new JLabel("PHY :");
    label4 = new JLabel("CHEM :");
    label5 = new JLabel("BIO :");
    label6 = new JLabel("MATHS :");
    label7 = new JLabel("TOTAL :");
    tf = new JFormattedTextField(NumberFormat.getIntegerInstance());
    tf.setColumns(10);
    panel.add(label);
    panel.add(tf);
    button = new JButton();
    button.setLabel("Search");

    button.addActionListener(this);
    panel.add(button);

    getContentPane().add(panel, BorderLayout.WEST);

    panel1.add(label1);
    panel1.add(pintf);
    getContentPane().add(panel1, BorderLayout.WEST);

    panel2.add(label2);
    panel2.add(nmetf);
    getContentPane().add(panel2, BorderLayout.WEST);

    panel3.add(label3);
    panel3.add(mrk1);
    panel3.add(label4);
    panel3.add(mrk2);
    panel3.add(label5);
    panel3.add(mrk3);
    panel3.add(label6);
    panel3.add(mrk4);
    panel3.add(label7);
    panel3.add(mrk5);
    getContentPane().add(panel3, BorderLayout.WEST);

    setDisable();
  }

  /* Default constructor */
  public UserInterface(){

    setSize(600,300);
    setTitle("Server Client Interaction - Amritam Sarcar");
    setTheLayout();
  }

  /* Action Listener */
  public void actionPerformed(ActionEvent ae) {

    /* Accessing the sendQuery method through port out */
    this.out.sendQuery(tf.getText());
  }
}
```

```
/* Defination of the ClientServerConnector */
component class ClientServerConnector{

  /* Defining the temporary values */
  private BufferedReader br;
  private PrintStream ps;
  private Socket socket = null;
  private String error = "OK";
  private String answer = null;

  /* Defining the default constructor */
  public ClientServerConnector(){

    try{
      /* Establishing the TCP/IP protocol connection */
      byte[] ipAddr = new byte[]{127, 0, 0, 1};
      socket = new Socket();
      InetAddress addr = InetAddress.getByAddress(ipAddr);
      socket.connect(new InetSocketAddress(addr, 6000));
      br = new BufferedReader(
           new InputStreamReader(socket.getInputStream()));
      ps = new PrintStream(socket.getOutputStream());
    }
    catch(Exception e) {
      error = new String(e.getMessage());
    }
  }

  /* Defining the port out */
  public port out{

    /* Accesses answerOfQuery of UserInterface component through its in
port */
    requires void answerOfQuery(String answer,boolean isans);
  }

  /* Defining the port in */
  public port in{

    /* Provides a method to check whether connected which is accessed
by
       the UserInterface component through its out port */
     provides String isconnected(){
      return error;
     }

    /* Provides a method to send the query which is accessed by
       the UserInterface component through its out port */
    provides void sendQuery(String query){
      try{
      ps.println(query);
      ps.flush();
      answer= br.readLine();
      this.out.answerOfQuery(answer,true);
      answer = null;
      }
```

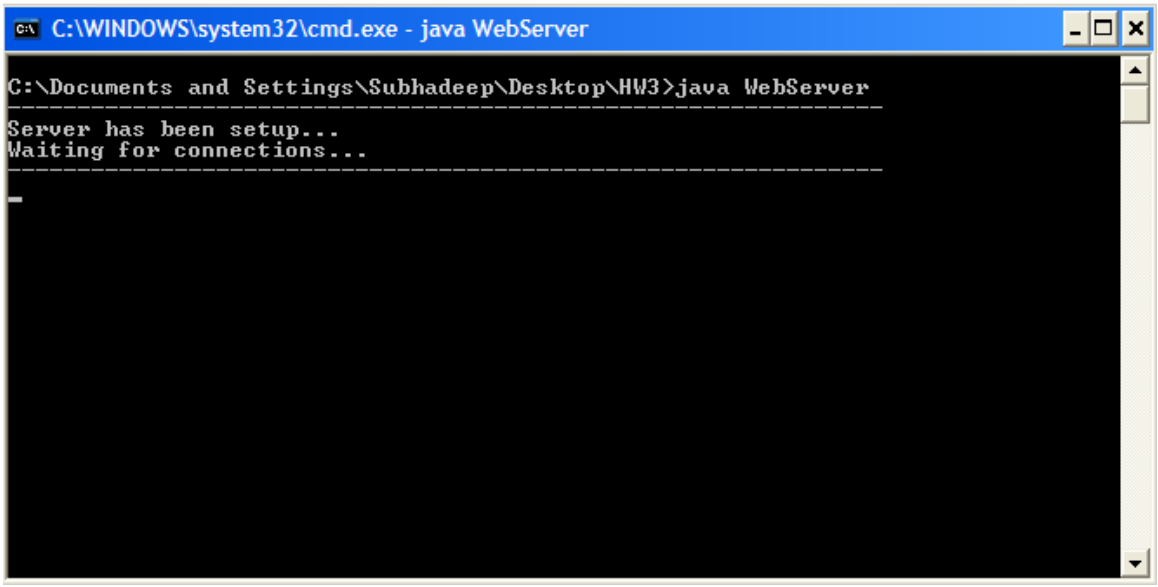
```
      catch(Exception e) {
            this.out.answerOfQuery(e.getMessage(),false);
      }
    }
  }
}
```

Output

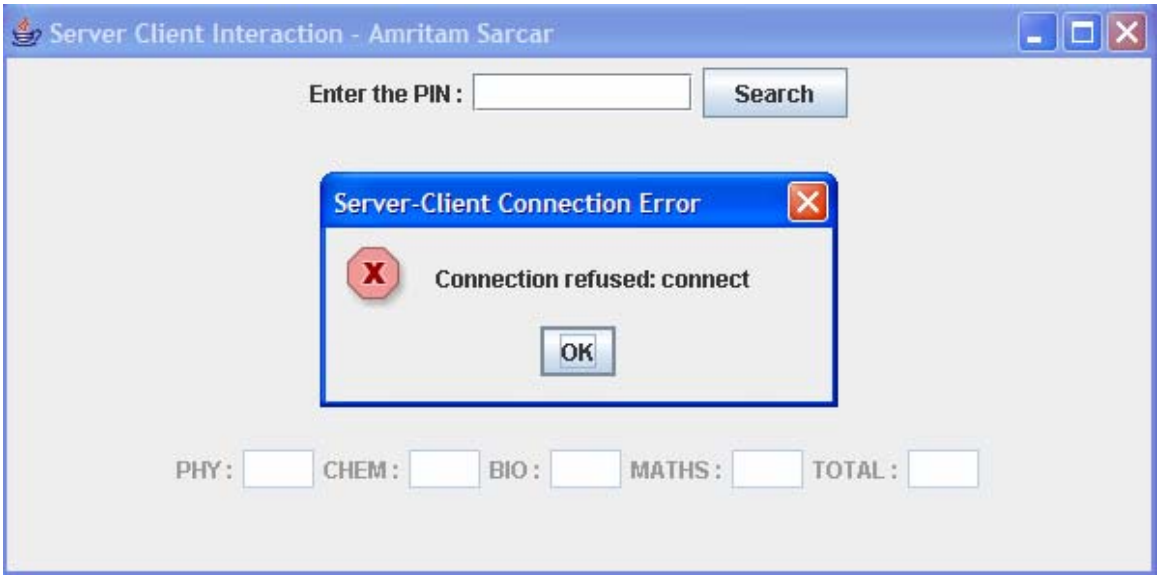
The following snapshots were taken in running the server and the client.



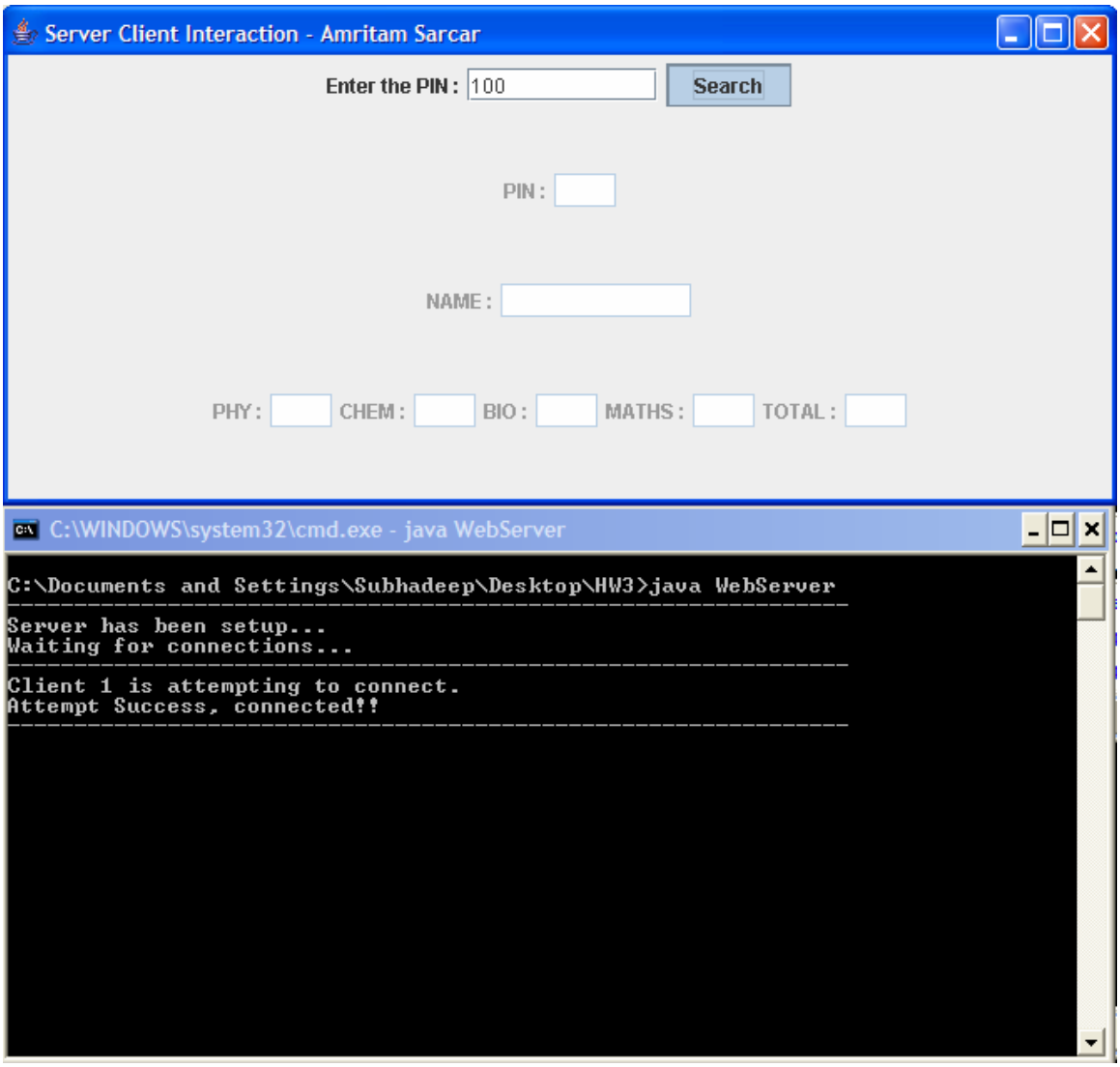
On successful compilation of WebServer.archj and WebClient.archj files.



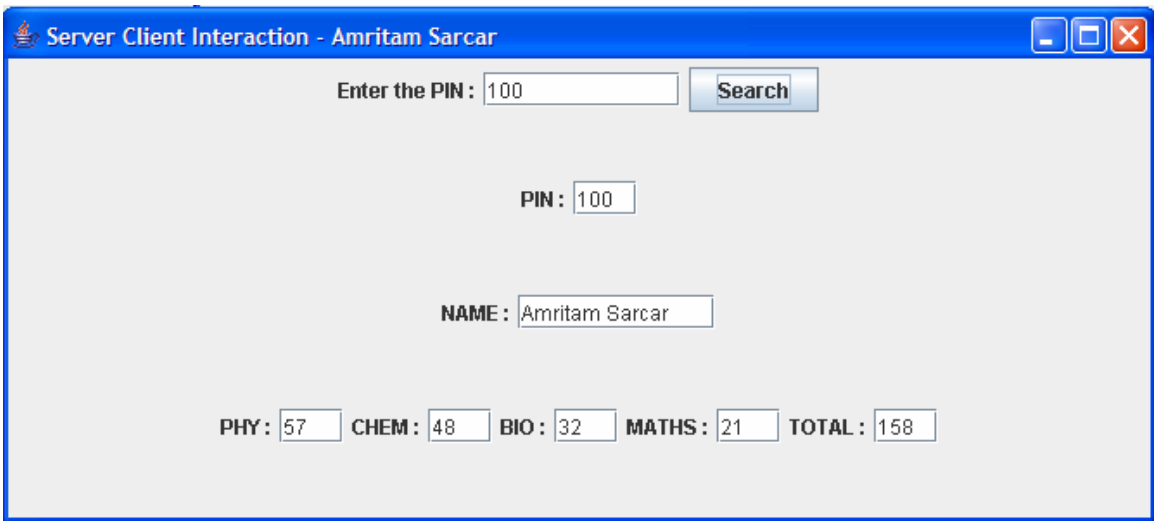
Running the WebServer without starting the client program



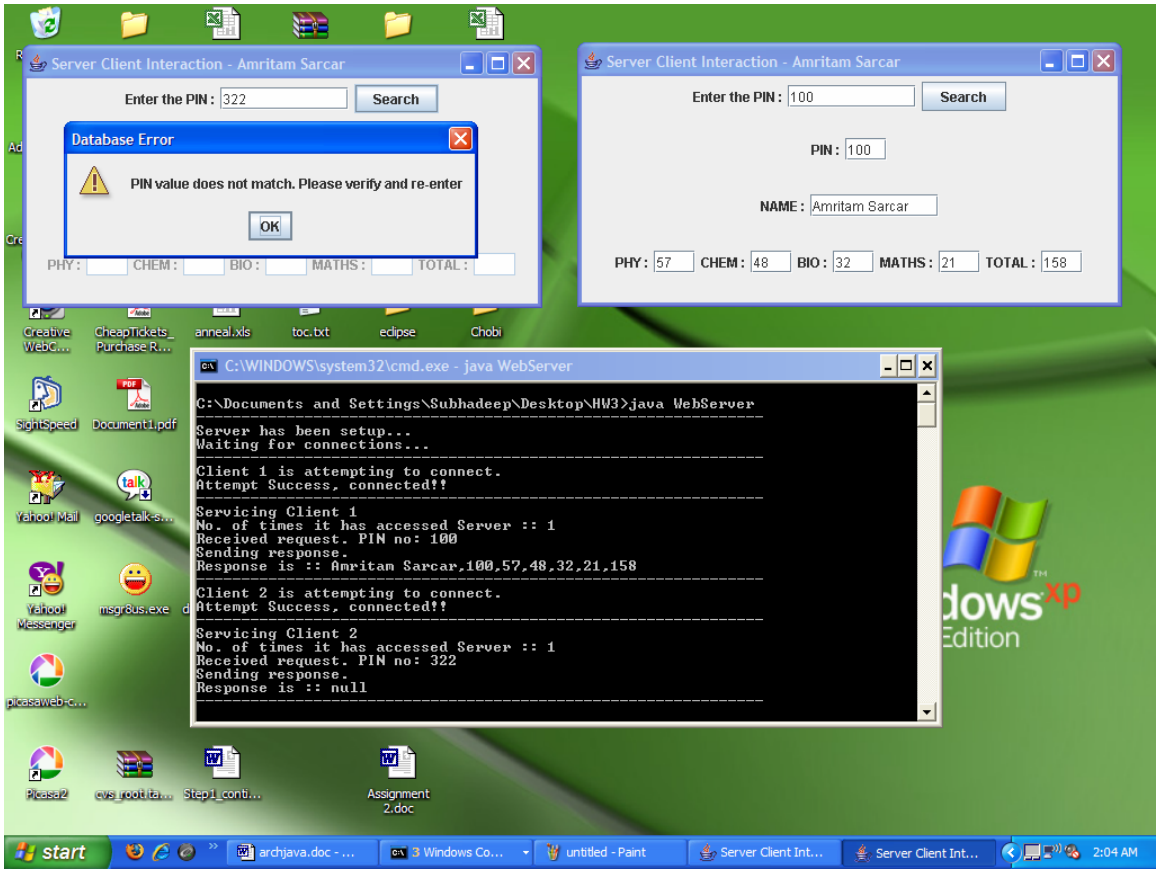
Running the WebClient without starting the server program



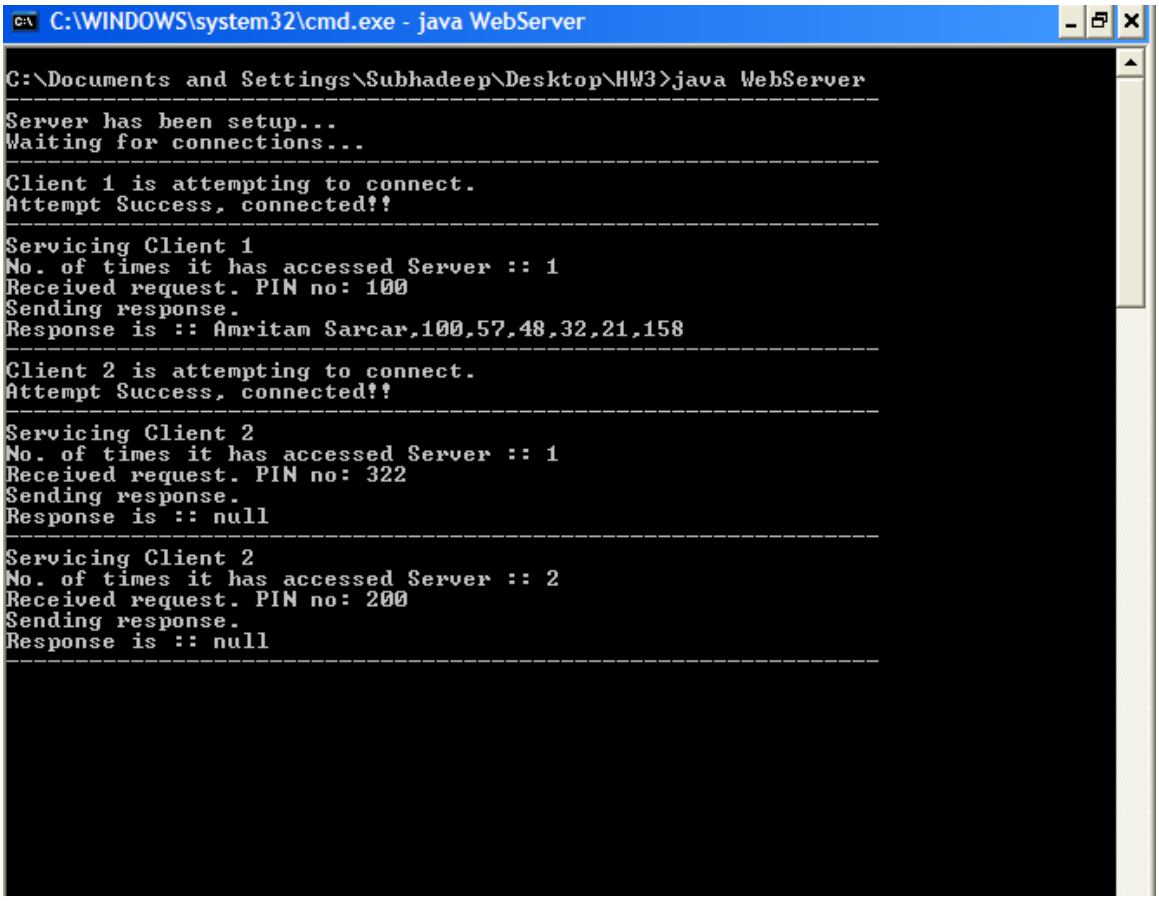
Client and Server interacting with each other.



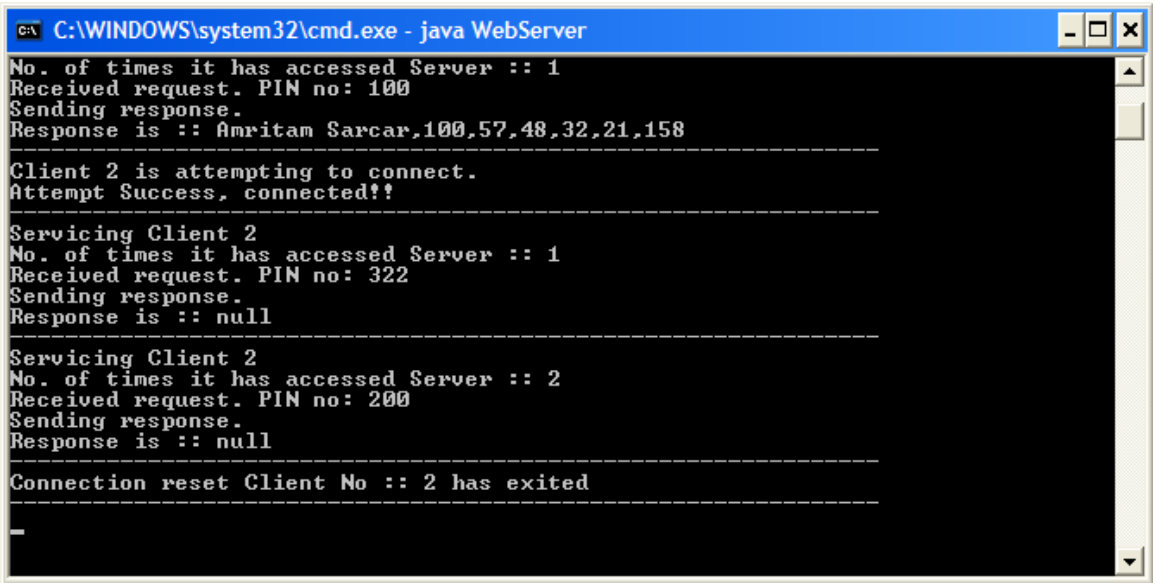
Corresponding marks and name for pin 100



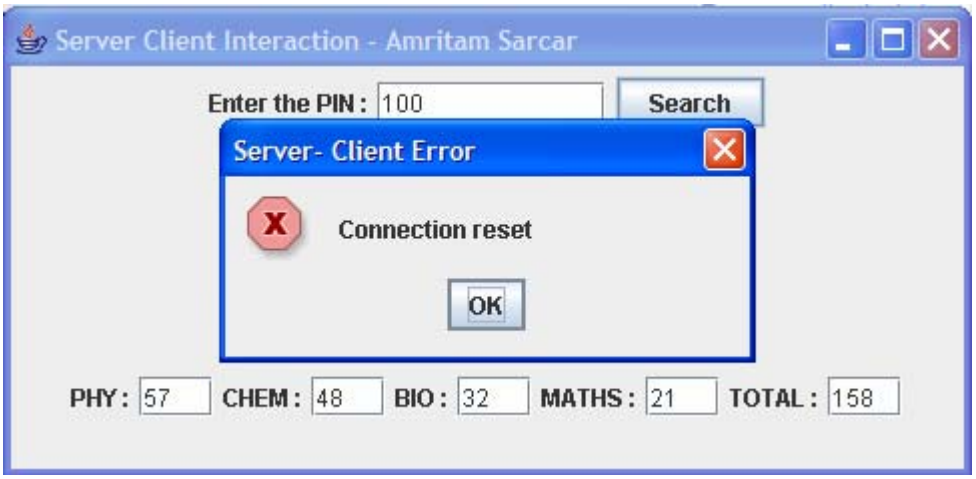
Two clients interacting and one client enters the wrong pin



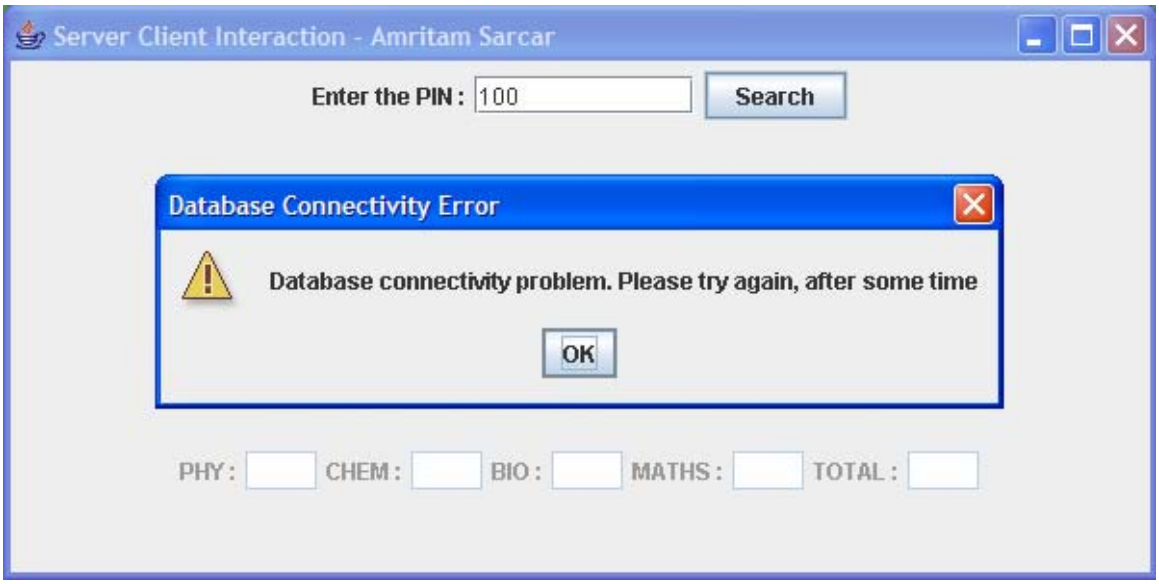
Keeping count of multiple clients and their individual access counts.



One of the connected clients being disconnected.



After successful connection, the server disconnects… client trying to access the server



The file/ database to be accessed cannot be found.

Reference:

1. http://www.cs.utep.edu/~cheon/cs5381/homework/network.ppt#1
2. ArchJava:. Connecting Software Architecture to Implementation. Jonathan Aldrich Craig Chambers. David Notkin. Department of Computer Science and Engineering, University of Washington, Seattle, WA USA
3. http://java.sun.com/docs/books/tutorial/uiswing/