Question 2

StudentDetails.java

```
package question_2;

import java.io.Serializable;

/**
 *
 * @author Donovan van Heerden | EL2014-0043
 */
public class StudentDetails implements Serializable {

    /**
     * Constructor of StudentDetails, takes in all required parameters to
initialise
     * the class
     *
     * @param Id            int
     * @param FirstName     String
     * @param LastName      String
     * @param ContactNumber String
     * @param Address       String
     */
    public StudentDetails(int Id, String FirstName, String LastName, String
ContactNumber, String Address) {
        this.Id = Id;
        this.FirstName = FirstName;
        this.LastName = LastName;
        this.ContactNumber = ContactNumber;
        this.Address = Address;
    }

    private final int Id;
    private final String FirstName;
    private final String LastName;
    private final String ContactNumber;
    private final String Address;

    /**
     * Returns the Id of the StudentDetails instance
     *
     * @return Id
     */
    public int getId() {
        return this.Id;
    }

    /**
```

```
     * Returns the FirstName of the StudentDetails instance
     *
     * @return FirstName
     */
    public String getFirstName() {
        return this.FirstName;
    }

    /**
     * Returns the LastName of the StudentDetails instance
     *
     * @return LastName
     */
    public String getLastName() {
        return this.LastName;
    }

    /**
     * Returns the ContactNumber of the StudentDetails instance
     *
     * @return ContactNumber
     */
    public String getContactNumber() {
        return this.ContactNumber;
    }

    /**
     * Returns the Address of the Student Details instance
     *
     * @return Address
     */
    public String getAddress() {
        return this.Address;
    }
}
```

Client.java

```
package question_2;

import java.awt.event.ActionEvent;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JButton;
```

```
import javax.swing.JLabel;
import javax.swing.SwingConstants;

/**
 *
 * @author Donovan van Heerden | EL2014-0043
 */
public class Client {

    private JFrame frmMain;
    private Socket socket;
    private ObjectOutputStream out;

    // This labels string array is used when positioning and laying out the
    // components
    private final String[] labels = new String[] { "Student ID:", "First Name:",
"Last Name:", "Contact Number:",
            "Address:" };

    private JTextField txtId;
    private JTextField txtFirstName;
    private JTextField txtLastName;
    private JTextField txtContactNumber;
    private JTextField txtAddress;

    private JButton btnRegister;

    /**
     * Entry point of file. Creates a new instance of the Client.
     *
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        Client client = new Client();
    }

    /**
     * Constructor for the Client class, initialises the JFrame and creates the
     * initial Socket connection
     *
     * @throws IOException
     */
    public Client() throws IOException {
        createForm();
        connect();
    }

    /**
     * Initialises the JFrame and some components
     *
     */
    private void initialise() {
        frmMain = new JFrame();
```

```
        txtId = new JTextField(5);
        txtFirstName = new JTextField(50);
        txtLastName = new JTextField(50);
        txtContactNumber = new JTextField(10);
        txtAddress = new JTextField(250);

        btnRegister = new JButton("Register");
    }

    /**
     * Used to setup and position all components on the JFrame
     *
     */
    private void createForm() {
        initialise();

        frmMain.setTitle("Student Details");
        frmMain.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frmMain.setResizable(false);
        frmMain.getContentPane().setLayout(null);

        frmMain.setBackground(new java.awt.Color(255, 255, 255));
        frmMain.setSize(320, 205);
        frmMain.setLocationRelativeTo(null); // Center JFrame

        frmMain.setVisible(true);

        txtId.setBounds(160, 10, 140, 25);
        txtFirstName.setBounds(160, 10 + (1 * 25), 140, 25);
        txtLastName.setBounds(160, 10 + (2 * 25), 140, 25);
        txtContactNumber.setBounds(160, 10 + (3 * 25), 140, 25);
        txtAddress.setBounds(160, 10 + (4 * 25), 140, 25);

        btnRegister.setBounds(160, 10 + (5 * 25), 140, 25);
        // Add the action listener to the button for handling the registration
function
        btnRegister.addActionListener((ActionEvent event) -> {
            try {
                handleRegister();
            } catch (IOException e) {
                System.out.println(e.toString());
            }
        });

        // Iterate over the labels string array to set the label value and
position the
        // labels dynamically
        for (int index = 0; index < labels.length; index++) {
            JLabel lbl = new JLabel(labels[index], SwingConstants.RIGHT);
            lbl.setBounds(35, 10 + (index * 25), 120, 25);
            frmMain.add(lbl);
        }
```

```
        // Add all the components to the JFrame
        frmMain.add(txtId);
        frmMain.add(txtFirstName);
        frmMain.add(txtLastName);
        frmMain.add(txtContactNumber);
        frmMain.add(txtAddress);
        frmMain.add(btnRegister);
    }

    /**
     * Initialises the socket connection on localhost:5000
     *
     * @throws UnknownHostException
     * @throws IOException
     */
    private void connect() throws UnknownHostException, IOException {
        socket = new Socket("127.0.0.1", 5000);
        out = new ObjectOutputStream(socket.getOutputStream());
    }

    /**
     * Handles the registration functionality. Fetches each textbox value, binds
the
     * value to an instance of the StudentDetails class and sends it over the
socket
     * connection.
     *
     * @throws IOException
     */
    private void handleRegister() throws IOException {
        try {

            int id = Integer.parseInt(txtId.getText());
            String firstname = txtFirstName.getText();
            String lastname = txtLastName.getText();
            String contactnumber = txtContactNumber.getText();
            String address = txtAddress.getText();

            if (firstname.equals("") || lastname.equals("") ||
contactnumber.equals("") || address.equals("")) {
                return;
            }

            StudentDetails student = new StudentDetails(id, firstname, lastname,
contactnumber, address);

            out.writeObject(student);
            out.flush();

            txtId.setText("");
            txtFirstName.setText("");
            txtLastName.setText("");
            txtContactNumber.setText("");
            txtAddress.setText("");
```

```
            } catch (Exception ex) {
                System.out.println(ex.toString());
            }
        }
    }
```

Server.java

```
package question_2;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.*;
import java.util.ArrayList;

/**
 *
 * @author Donovan van Heerden | EL2014-0043
 */
public class Server {
  private static final int PORT_NUMBER = 5000;
  private static ServerSocket serverSocket;

  // Connections arraylist is used to keep track of the existing connections,
  // later on this allows the server to dispose or disconnect each connection
  // gracefully before exiting
  private static ArrayList<Socket> connections;

  /**
   * Start of the server, creates a ServerSocket instance on port 5000. Waits for
   * incoming connections and creates a new thread for each new connection,
   * storing that connection into an array to keep track of.
   *
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    connections = new ArrayList<>();
    serverSocket = new ServerSocket(PORT_NUMBER);

    while (true) {
      Socket socket = serverSocket.accept();

      connections.add(socket);

      Thread thread = new Thread(() -> {
        try {
          handleConnection(socket);
```

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

      thread.start();
    }
  }

  /**
   * Used to handle each client connection. Waits for data being sent from the
   * client and casts the object to a StudentDetails instance. Which it then tries
   * to save to the database.
   *
   * @param client
   * @throws SQLException
   */
  private static void handleConnection(Socket client) throws SQLException {
    ObjectInputStream input = null;
    try {
      input = new ObjectInputStream(client.getInputStream());

      while (client.isConnected()) {
        StudentDetails student = (StudentDetails) input.readObject();
        saveToDatabase(student);
      }

      input.close();
      client.close();

    } catch (IOException | ClassNotFoundException ex) {
      // exit silently
    } finally {
      try {
        input.close();
        client.close();
      } catch (IOException e) {
        System.out.println(e.toString());
      }
    }
  }

  /**
   * Saves the StudentDetails data into the database, provided the student
   * parameter is not null
   *
   * @param student
   * @throws ClassNotFoundException
   * @throws SQLException
   */
  private static void saveToDatabase(StudentDetails student) throws
ClassNotFoundException, SQLException {
    try (Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/PIHE2019", "root",
```

```
"root")) {
      PreparedStatement statement = conn.prepareStatement(
          "INSERT INTO details (Id, FirstName, LastName, ContactNumber, Address)"
+ " VALUES(?, ?, ?, ?, ?);");

      statement.setInt(1, student.getId());
      statement.setString(2, student.getFirstName());
      statement.setString(3, student.getLastName());
      statement.setString(4, student.getContactNumber());
      statement.setString(5, student.getAddress());

      statement.executeUpdate();
    } catch (SQLException e) {
      System.out.println(e.toString());
    }
  }

  /**
   * Allows for the server to shutdown gracefully and closes any open socket
   * connections to the server, before stopping the server
   *
   * @throws IOException
   */
  protected void finalise() throws IOException {

    for (int i = 0; i < connections.size(); i++) {
      Socket socket = connections.get(i);
      if (socket.isClosed()) {
        continue;
      }

      socket.shutdownOutput();
      socket.close();
    }

    serverSocket.close();
  }
}
```