


National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Software Construction & Development	Course Code:	CS-3001
	Degree Program:	BS(SE)	Semester:	Fall 2023
	Exam Duration:	180 minutes (3 hours)	Total Marks:	80
	Paper Date:	29 - Dec - 2023	Weight:	40.00%
	Section:	ALL	Pages:	8
	Exam Type:	Final	Questions:	5

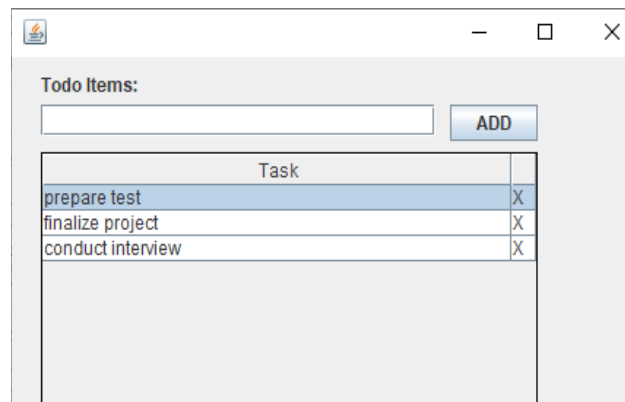
Student Name:_____ **Roll No.**_____ **Section:**_____

Instruction/Notes: Attempt all questions. Do not use pencil or red ink. In case of confusion or ambiguity make a reasonable assumption. **Do not attach any extra sheet.** Use extra sheet for rough work only. A **double-sided hand-written** cheat sheet is allowed but it shouldn't be photo-copied.

Question 1 [CLO-1]

20 points

Consider a simple Java Swing Application that tracks a list of To-do items using a JTable component, as illustrated below:



Partial code is given below. Complete the blank portions to ensure that Todo items can be added and removed.

```
public class TodosUI extends javax.swing.JFrame{
    private javax.swing.JButton addButton;
    private javax.swing.JLabel label;
    private javax.swing.JTextField todoTextField;
    private javax.swing.JPanel todosTablePanel;

    private TodosTableModel model;
    private JTable todosTable;
    private JScrollPane scroll;

    public TodosUI(ArrayList<Todo> d) {
        initComponents(); // add components and set layout
        // relevant event handlers creation to be done here ...
        todosTable.getSelectionModel().addListSelectionListener(
            new RowSelectionListener());
        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                model.add(new Todo(todoTextField.getText()));
                todoTextField.setText("");
            }
        });
    }
}
```

```

private class TodosTableModel extends AbstractTableModel{
    private ArrayList<Todo> todos;
    private final String[] columnNames = {"Task", " "};

    public TodosTableModel(ArrayList<Todo> t){
        todos = t;
        if (todos == null){
            todos = new ArrayList<>();
        }
    }

    @Override
    public int getRowCount() {
        return todos.size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public Object getValueAt(int r, int c) {
        Todo todo = todos.get(r);
        if (c == 0){
            return todo.getTask();
        }

        return "X";
    }

    public void add(Todo t){
        // write code to add a todo item
        todos.add(t);
        fireTableDataChanged();
    }

    public void delete(int index){
        // write code to delete a todo item
        if (index >= 0 && index < todos.size()){
            todos.remove(index);
        }
        fireTableDataChanged();
    }
}

private class RowSelectionListener implements ListSelectionListener{

    @Override
    public void valueChanged(ListSelectionEvent evt) {
        // write code to handle row manipulation in the table
        if (evt.getSource() == todosTable.getSelectionModel()) {
            if (todosTable.getSelectedColumn() > 0){
                model.delete(todosTable.getSelectedRow());
            }
        }
    }
}
}

```

Consider a Java based client-server application for monitoring patient oxygen levels in a hospital ICU. Oxygen saturation is measured in percentage and a reading below 80% is problematic in which case nursing staff needs to be notified. The application is implemented using socket programming where each client measures oxygen saturation level every second for the relevant patient and transmits this information (bed number and saturation level) to the server. The task of server is to monitor all attached clients (beds) and in case the level of any patient goes below threshold (80%) then display flashing red alarm and bed number on the attached screen (assuming it is a shared resource where only a single bed information can be shown at a time).

The client code is given as follows:

```
public class OximeterClient {

    public static void main(String[] args) {
        Socket socket;

        try{

            socket = new Socket("localhost",4444);

            PrintWriter socketwriter = new PrintWriter(socket.getOutputStream());

            Thread thread = new Thread(new Runnable(){
                public void run(){
                    try{
                        while(true){
                            int value = measureO2level();
                            socketwriter.println("" + value);
                            socketwriter.flush();
                            Thread.sleep(1000);
                        }
                    } catch(Exception ex){
                        System.out.println("Exception:" + ex.getMessage());
                    }
                }
            });
            thread.start();
            thread.join();
            socket.close();

        }
        catch(Exception ex){
            System.out.println("Exception:" + ex.getMessage());
        }

    }

    private static int measureO2level(){
        return ((int)( Math.random() * 30)) + 69;
    }

}
```

Write corresponding server code that can handle multiple clients (beds) simultaneously and raise alarm if the level goes below the threshold at any bed. Take care of necessary threading and concurrency issues.

```

public class IcuServer {

    public static void main(String[] args) {
        ServerSocket server;
        Display display;

        try {
            server = new ServerSocket(4444);
            display = new Display();

            Socket client = null;
            do {
                client = server.accept();

                if (client != null){
                    OximeterTask task = new OximeterTask(client,display);
                    Thread thread = new Thread(task);
                    thread.start();
                }

            } while (client != null);

        } catch (Exception ex) {
            log(ex.toString());
        }
    }

    private static void log(String message){
        System.out.println(message);
    }
}

public class Display{

    // other attributes and functions

    public synchronized void raiseAlarm(String bedNumber){
        System.out.println("Alarm at " + bedNumber);
        // flash alarm signal
    }
}

```

```

public class OximeterTask implements Runnable {

    Socket client;
    Display display

    public OximeterTask(Socket client, Display display) {
        this.client = client;
        this.display = display;
    }

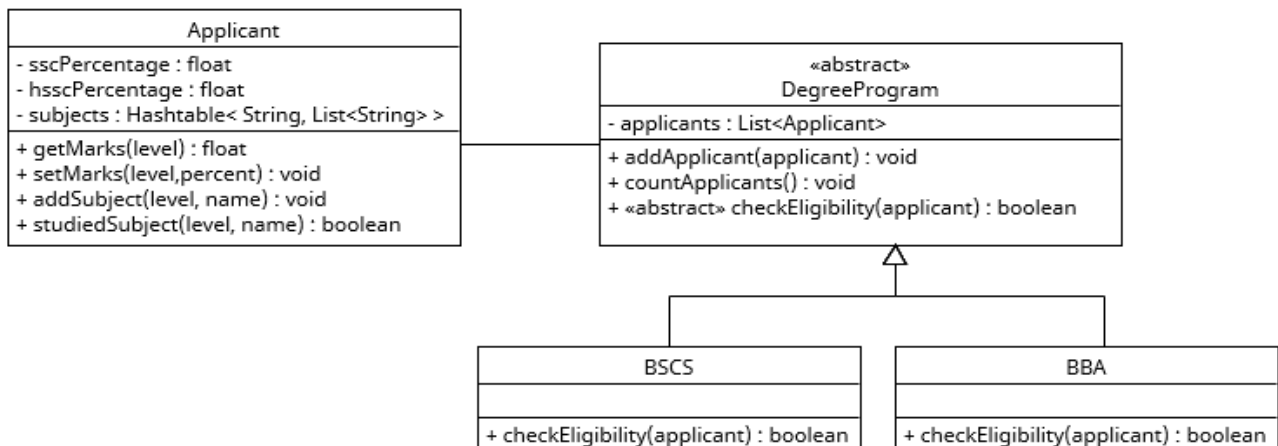
    public void run() {

        try {
            PrintWriter writer = new PrintWriter(client.getOutputStream());
            BufferedReader reader = new BufferedReader(new
                InputStreamReader(client.getInputStream()));

            String message = "";
            while((message = reader.readLine()) != null){
                System.out.println("bed " + client.getPort() + ":" + message);
                Integer value = Integer.parseInt(message);
                if (value < 80){
                    display.raiseAlarm("" + client.getPort());
                }
            }
        } catch(Exception ex){
            ex.printStackTrace();
        }
    }
}

```

Consider the following class diagram of an admission management system:



Details of functions are described as under:

- **setMarks** and **getMarks** set and retrieve marks in percentage respectively for the given level (SSC i.e. matric or equivalent, and HSSC i.e. intermediate or equivalent)
- **addSubject** adds a subject studied at given level e.g. Mathematics at HSSC
- **studiedSubject** checks whether the subject was studied at the given level or not
- **addApplicant** adds an applicant to the list of applicants in a degree program if the applicant is eligible (through **checkEligibility**) and **countApplicants** determine the number of eligible applicants added to the list
- **checkEligibility** for BSCS returns true if applicant has: i) 60% or above marks at SSC level, ii) 50% marks or above at HSSC level, iii) studied Mathematics at HSSC level
- **checkEligibility** for BBA returns true if applicant has: i) 60% or above marks at SSC level, ii) 50% marks or above at HSSC level

Write unit tests (using JUnit) for the following functions:

(a) Applicant.studiedSubject(level, name)

```

Applicant app = new Applicant();
assertFalse(app.studiedSubject("HSSC", "Mathematics"));
app.addSubject("HSSC", "Mathematics");
assertTrue(app.studiedSubject("HSSC", "Mathematics"));
assertFalse(app.studiedSubject("SSC", "English"));
app.addSubject("SSC", "English");
assertTrue(app.studiedSubject("SSC", "English"));
  
```

(b) DegreeProgram.addApplicant(applicant)

```
DegreeProgram prog = new BBA(); // or any other concrete subclass e.g. BSCS
assertEquals(prog.countApplicants(), 0);

Applicant app = new Applicant();
app.setMarks("SSC", 65);
app.setMarks("HSSC", 65);
prog.addApplicant(app);

assertEquals(prog.countApplicants(), 1);
```

(c) BSCS.checkEligibility(applicant)

```
DegreeProgram prog = new BSCS();
Applicant app = new Applicant();
assertFalse(prog.checkEligibility(app));

app.setMarks("SSC", 65);
assertFalse(prog.checkEligibility(app));

app.setMarks("HSSC", 65);
assertFalse(prog.checkEligibility(app));
```

(d) BBA.checkEligibility(applicant)

```
DegreeProgram prog = new BBA();
Applicant app = new Applicant();
assertFalse(prog.checkEligibility(app));

app.setMarks("SSC", 65);
assertFalse(prog.checkEligibility(app));

app.addSubject("HSSC", "Mathematics");
assertFalse(prog.checkEligibility(app));

app.setMarks("HSSC", 65);
assertTrue(prog.checkEligibility(app));
```

Question 4 [CLO-4]

5+5=10 points

Answer the following questions:

(a) What is the difference between a Centralized and a Distributed Version Control System?

In a centralized system, repository is maintained centrally – clients only have their working copies.

In a distributed system, repository is distributed / replicated among all clients i.e. each client has a working copy as well as the snapshot of repository.

(b) A developer using Git committed changes to the repository but forgot to push them. What can possibly go wrong in such a situation and what is the purpose of push command?

If the changes are committed but not pushed, then they are tracked locally on the client machine only. If the client machine is corrupted, information loss may occur. The purpose of push command is to synchronize the changes with a remote repository as well.

Question 5 [CLO-5]

5+5=10 points

A development team has completed work on a new educational application (built in JAVA) for secondary school students and intends to release the corresponding JAR file on Microsoft Store:

(a) Advise them on the necessary steps for a successful release:

Develop and test the release, run QA procedures, prepare all the artifacts (executables, source-code, relevant docs such as user manuals and API docs, etc.) , apply version control labels

(b) What issue will occur if the JAR is not signed?

If JAR is not signed then its integrity (correct JAR prepared by trusted vendor) cannot be verified and Microsoft Store will not accept it for publishing.