Event-driven Programming
Project (60%)

Due: **Two Deadlines: Week 6, Week 11**

This project consists of two parts:

- **Part 1: A practical component 1 is due to be submitted online on Brightspace by Week 6 Sunday 11:55 PM. Interviews and Presentations will be held during week 7 Labs (you MUST appear in your scheduled labs). No late submissions are allowed.**
- **Part 2: A practical component 2 is due to be submitted online on Brightspace by Week 11 Sunday 11:55 PM. You must submit a video presentation covering ALL assessment criteria. No late submissions are allowed.**
- **Bear in mind that practical component 2 is the extension of practical component 1. This means you won't be able to work on Component 2 before completing Component 1.**

OVERALL PROJECT MARKING SCHEME

FINAL MARK = PART1 marks * 50% + PART2 marks * 50%

PART 1 - Practical Component 1 - 30%

This is a group-based project that will be individually assessed during the midterm and final. Each student will be assessed for the full project no matter what part of the project he/she has implemented. No more than 2 students are allowed in each group.

Each group is required to submit the solution for the practical task on Brightspace. The submission must include both the client and the server NetBeans projects to allow for compilation and testing.

Description

Develop a TCP protocol-based Client-server service that consists of two applications. Where the client application will be a GUI application whereas the server is a console application.
The main communication protocol for this service is that the client and the server must alternate between sending and receiving messages. The client initiates the process by sending the first message request and the server replies with a new message response. A *STOP* button should be pressed by the client when it wishes to close the connection/communication with the server. When the server receives the *STOP* message, it will confirm the termination of the communication by sending back a message with the *TERMINATE* keyword response and closing its connection to the client. From the client side, all this communication should be performed using appropriate GUI controls (e.g., text fields and buttons) and event-handling mechanisms.

A message sent by the Client application to the server consists of *action* and *event description* information read from the GUI controls. The server application acts and responds to the client with a message.

Your applications **MUST USE** the **Exceptions Handling.** You must define a new exception called **IncorrectActionException** that produces a message error that will be sent by the server. This **exception** will be thrown if the user provides an incorrect action format. Your application must then react to this exception.

For a connection to be established with a client, the server application must:

- receives the message sent by the client application.
- calls the appropriate method to perform the action indicated in the message.
- replies to the client with a piece of information according to the action performed.

Class Scheduler Application

Following is the detail of the client and server applications:

Client App: The client sends to the server app information regarding a class to schedule and displays on the proper GUI control the response messages received from the server. Before sending a message, the client selects an action to be requested from the server using an appropriate GUI control. The client may request to perform the following operations (1). Add Class, (2). Remove Class, and (3). Display Schedule. To add or remove a class, the client selects a date, time, and room number and then provides the name of the class to schedule. All this information should be provided using appropriate GUI controls. When requesting to display schedule, the client should again be using a proper GUI control to send a display request for a specific class (e.g., LM051-2022) to the server.
On getting a response from the server, the client should also display the response using a GUI control.

Server App: The server application has a memory-based data collection (e.g., ArrayList, HashMap, etc.) that stores class's (e.g., LM051-2022) schedules that were received from a client. The server performs the following actions:

1. **Add Class**: Check if there is any clash in scheduling the requested class. If a clash is found notify the client. If no clash is found, then book that class at the requested time and notify the client about successful scheduling.
2. **Remove Class**: The server removes a particular class from the server's data collection. The server removes the record from its data structure (e.g., ArrayList) and will reply with the freed time slot with room information.
3. **Display Schedule**: In case the client requests to display the complete schedule. The server will display the full schedule of the requested class (e.g., LM051-2022). Assume there will be no more than 5 modules a class can study in a semester; the server will display the schedule in console application and **NOT** in GUI.

Submission Format

Name your applications with your student IDs. For example, if two members of a group have the following IDs 21237333 and 21222344, they will name their applications as follows:

- 21237333_21222344_Server
- 21237333_21222344_Client
- They will zip both applications into a folder and will name that folder as follows: 21237333_21222344.zip. Such a zip file should (only) be submitted on Brightspace.

REMEMBER!

- **In case of plagiarism found, both plagiarised projects will get zero marks and your case will be forwarded to the disciplinary committee. Hence, it is your responsibility not to copy or disclose your project in class.**
- **A new exception called IncorrectActionException must be defined and caught by the application.**
- **Client and Server exchange messages until the client presses STOP and the connection will be terminated.**
- **Test the application for all scenarios and the assessment criteria provided below.**

Assessment Criteria	Marks
Right selection of the overall GUI controls for client application	**10%**
Successful Connection and communication between Client-server	**10%**
Scheduling clash testing	**10%**
Class Add test – with all events handling	**20%**
Class remove test – with all events handling	**20%**
Display test – with all events handling	**10%**
Implementation of IncorrectActionException	**10%**
Code structure and modularization	**10%**
BONUS: Any additional convincing features	**Upto 40%**

SPECIAL NOTES!

1. **As a starting point, one working but rough example has been provided in the Brightspace. This example contains both server and client programs. The client is using very simple and eccentric GUI controls to get a message from the user and send that message to the server. The server, on the other hand, receives the message capitalizes the message, and echoes the same message back to the client. The client receives a response and displays using a Label GUI control and terminates.**
2. **The Description of Practical Component Part 2 will be made available in Week 8**