Event-driven Programming
Project (60%)

Due: **Two Deadlines: Week 6, Week 12**

This project consists of two parts:

- **Part 1: A practical component 1 is due to be submitted online on Brightspace by Week 6 Monday 11:55 PM. No late submissions are allowed.**
- **Part 2: A practical component 2 is due to be submitted online on Brightspace by Week 12 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.

Introduction and High-level Requirements

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. You will develop this application using the Model-View-Controller (MVC) style.

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.

- On 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.

Project Description: Lecture Scheduler Application

Following is the detail of the client and server applications:

Client App: The client sends to the server app information regarding a lecture 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 a Lecture, (2). Remove a Lecture, (3). Display Schedule, (4). Others. To add or remove a lecture, the client selects a date, time, and room number from it's GUI controls and then provides the name of the module for which lecture needs to be scheduled. All this information should be provided using appropriate GUI controls. On getting a response from the server, the client should also display the response using a GUI control. Supposing that application will only allow to schedule the lectures for 1 course (e.g., LM051-2026), client may request scheduling lectures for upto five modules.

Server App: The server application will receive 4 types of requests ((1). Add a Lecture, (2). Remove a Lecture, (3). Display Schedule, (4). Others) from client and will process those requests. Server app will have a memory-based data collection (e.g., ArrayList, HashMap, etc.) that stores one course (e.g., LM051-2026) schedule by adding/removing lectures for modules based on the requests from client.

The following demonstrate the 4 actions and the tasks to perform by client and server applications:

1. **Add a Lecture**: Client using its proper GUI will gather information for lecture to schedule and will send that information to the server with an action identifier (e.g., Add). Server will check if there is any clash in scheduling the requested lecture. A clash can occur if the room

is occupied at requested time and-or course student (LM051-2026) have another module's lecture at that time. If a clash is found notify the client with a proper message. If no clash is found, then book that class at the requested time and room, add this information in memory-based data structures (e.g., HashMap) and notify the client about successful scheduling of lecture. Client once receive a failure/success information will use a proper GUI to display the error/success message.

2. **Remove a Lecture**: Similar to the "Add a lecture", client will use proper GUI to select the lecture to remover and send that request to the server with an action identifier (e.g., Remove). The server removes a particular class from the server's data collection. That is, the server will remove the record from its data structure (e.g., ArrayList) and will reply with the freed time slot with room information to client. Client will display the response from server using proper GUI controls.
3. **Display Schedule**: In case the client requests to display the complete schedule of a course (e.g., LM051-2026). The server will send the information of full schedule of the course (e.g., LM051-2026). Assume there will be no more than 5 modules a course can study in a semester; the server don't need to display the schedule rather it will send the schedule information to client (what format server will send to client and how client will parse that information?). The client will receive that information and then will display the full course schedule using a **proper** GUI control (Hint: You may consider the design of your own course's GUI on web portals: https://www.timetable.ul.ie/). The schedule display will be for per week (as it is for your course).
4. **Other**: Using GUI control, the client will ask a request that server is not providing service for. The server will throw an IncorrectActionException on server side on receiving such an information and will send a proper message to client, letting him know that client ask a service that server don't provide and it has caused an exception on server side. Client after receiving that information will use a proposer GUI to display such a message on client side.

Assumptions: Assume that this full application is for managing the timetable only for 1 semester, only for maximum 5 modules, only for one course where display is only for 1 week, and GUI is only on client side. ***You are free to assume the other information about the application that you think is still missing.***

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 server 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%**
Add a Lecture test – with all events handling	**10%**
Remove a Lecture test – with all events handling	**20%**
Display test – with all events handling	**20%**
Implementation of IncorrectActionException	**10%**
Code structure and modularization – Implementation of MVC	**10%**
BONUS: Any additional convincing features	**Upto 40%**

SPECIAL NOTES!

1. **As a starting point, one working but rough example will be provided through Brightspace after your first hard lab is completed. 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**

PART 2 - Practical Component 2 - 30%

The same group will extend the already developed project for the following features. Each groupis required to submit on Brightspace the solution for the practical task. The submission must include both the client and the server NetBeans projects to allow for compilation and testing.

Description

Suppose that multiple clients can now schedule the timetable at the same time. Provide all the functionalities in Part 1 by adding the multi-processor architecture support using the Multithreadingconcept. To implement this, the server application must start a new Thread for each connection established with a client. Each thread on the server side will then:

1. Receives the requests/messages sent by the client application.
2. Calls the appropriate method to perform the action indicated in the message.
3. Replies to the client with a piece of information according to the action performed.

Your applications must ensure synchronization and control access to the memory-based data collection (e.g., ArrayList, HashMap object) managed on the server side. This time your server will also be a GUI application.

In addition to this, implement a new feature in your applications where clients can request '***early lectures***' and the server brings the classes towards morning timings. For example, if two classes are scheduled on Tuesday between 13:00 –15:00 and 15:00 – 17:00 and three classes are scheduled on Wednesday between 11:00 – 13:00, 13:00 –15:00, and 15:00 – 17:00, then an '***early lectures***' request from client will cause the server to shift those classes in early mornings (e.g., between 9:00 and 13:00 for Tuesday classes and between 9:00 and 15:00 for Wednesday classes) ONLY if the early morning timeslots are not occupied.
You are expected to implement this functionality using a divide-and-conquer algorithm, where a different thread is shifting the classes to early mornings for each day (i.e., different threads created todeal with separate days in a week).

Assessment Criteria	Marks
Implementation of multi-threading	**10%**
'early lectures' test – with all events	**30%**
Implementation of Synchronization and control access to data/methods	**20%**
Implementation of Fork-Join Rule using Divide and Conquer	**20%**
Implementation of the javafx.cuncurrent	**10%**
Code structure and modularization	**10%**
BONUS: Any additional convincing feature	**Upto 40%**

Server must decouple the JavaFX part from '*early lectures*' feature using Task and Worker phenomenon of javafx.cuncurrent package. This decoupling should allow the server to handle the requests from other clients to perform Addition, Removal, and Display tasks for other classes (e.g.,LM110-2022).

Submission Format

Video Presentation:

You must record a video of 7-8 minutes of your projects demonstrating the 3 features and their code.

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
- 21237333_21222344_video
- They will zip both applications and video files 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 the 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.***
- **Each Client-Server communication is managed by a thread.**
- ***The Server application deals with a shared resource, the memory-based data collection (e.g., ArrayList, HashMap object). You must implement a mechanism that ensures threads synchronization.***
- **The task of 'early lectures' must be decoupled and performant.**
- ***Test the application by launching in NetBeans multiple executions of the Client application to simulate multiple clients that exchange messages with the same server concurrently.***