# National College of Ireland

**BSHC3A_BSHC3B**

**Release Date: Monday 11th Oct, 2024**
**Due Date: Monday 4th Nov, 2024**
**Dr. Abdul Razzaq**

---

## Advanced Programming

## Continuous Assessment (CA) Type: Project and Assignment

**Weight:** The weight is 50% and the assignment will be marked out of 100.

## Instructions:

You are required to answer ALL questions.
This is a practical project-based assessment to be done individually.
**Please do the version based on the last digit of your Student ID.**

**SUBMISSION DETAILS**:
Each student is required to submit the solution for the practical task online on Moodle. The submission must include both the client and the server NetBeans projects to allow for compilation and testing.
**IMPORTANT: ONLY one Zip file contains client and server applications. You will name your client project as Id_name_client, server project as Id_name_server, and main zip file (that contains both client-server applications) as Id_Name (e.g., 6735921_Martin)**

**TURNITIN**: All report submissions will be electronically screened for evidence of academic misconduct (i.e., plagiarism and collusion)

**Duration of Continuous Assessment:** 22 days for project

# Project Description

**Description**
Develop a Java TCP protocol-based Client - Server service that consists of two applications. The main communication protocol for this service is that client and server must alternate between sending and receiving. The client initiates the process by sending the first message and the server replies with a new message. A message with the *STOP* keyword should be sent by the client when it wishes to close down 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 messagewith the *TERMINATE* keyword and close its connection to this client.

A message sent by the Client application to the server consists of *action* and *event description* information read from the keyboard. The server application performs the *action* and responses back to the client with a message.

Your applications demonstrate the use of the following concepts:

- Threads
- Sockets
- Exceptions

You must define a new exception called **IncorrectActionException** that produces an errormessage that will be sent by the server. This **IncorrectActionException** will be thrown if the user provides an incorrect action. Your application must then react to this exception.

For each connection established with a client, the server application must start a new thread that

- 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

Based on the last digit in a student ID's, the information to be exchanged between client and server should be as follows:

## Students ID's with a last digit of 0, 1, 2, 3 or 4:
## *Develop a Dublin Events Calendar*

The server application has a memory-based data collection (e.g., ArrayList) that stores event descriptions that were received from clients. The client sends to the server app information regarding an event and displays on the screen the message received from the server.

The client application sends to the server messages read from the keyboard. One message consists of an *action* and *event description*.

The **action** can be one of the following key words:

- ***add*** - to add a new event with the provided description to the server data collection. The server will reply with a list of all events due on the new event's date.
- ***remove*** - to remove a particular event from the server's data collection. The server will reply with a list of all events that are still due on the date of the removed event.

The **event** description consists of date, time and brief text description of the event.

Example 1:
Client sends the message: *add; 2 November 2024; 6 pm; Fireworks Dublin City Centre*.

Server replies with the message: 2 *November 2024; 12 pm, Food Market IFSC Square; 6 pm, Fireworks Dublin City Centre; 7.30 pm, Jersey Boys Bord Gais Energy Theatre*

Example 2:

Client sends the message: *remove; 2 November 2024; 6 pm; Fireworks Dublin City Centre*.

Server replies with the message: 2 *November 2024; 12 pm, Food Market IFSC Square; 7.30 pm, Jersey Boys Bord Gais Energy Theatre*

## Students ID's with a last digit of 5, 6, 7 or 9:

***Develop a To-Do List for New Hopes Charity Association***

The server application has a memory-based data collection (e.g ArrayList) that stores TODO tasks that were received from end-users (remote client app) of the server application. Any new task description sent by the client app will be stored in the TODO data collection. The server replies with a message as described below. The client application displays on the screen the message received.

The client application sends to the server messages read from the keyboard. One message consists of an *action* and *event description*.

If the action is ***add***, then the text description represents details about the new TODO task such as date and a brief text description. The server adds the task to memory-based data collection and sends back a list with all tasks that currently exist for that particular day.

If the action is ***list***, then then the text description represents a date for which the client wants to see all the TODO tasks that exist. The server sends back a message with the TODO tasks list. If there are no TODO tasks activities for that date, a message to indicate that is sent by the server.

*Example 1:*
Client sends the message: *add; 20 December 2024; Distribute Christmas presents to St Mary School*.
Server replies with the message: *20 December 2021, Distribute Christmas presents to St Mary School; Sing Christmas Carols in Dundrum Shopping Center*.

*Example 2:*

Client sends the message: *list; 20 December 2024*.

Server replies with the message: *20 December 2024; Distribute Christmas presents to St Mary School; Sing Christmas Carols in Dundrum Shopping Center*.

## Your application must demonstrate the following:

- Make sure that exception handling is addressed in your program.
- A new exception called IncorrectActionException must be defined and caught by the application.
- Client and Server exchange messages until the client sends STOP and the connection will be terminated.
- 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 object). You must implement a mechanism that ensures threads synchronisation.
- The server runs on the local machine. Do not include your machine IP address in the code.
    - Use the getHostAddress() method
- Test the application in NetBeans by launching multiple executions of the Client application to simulate multiple clients that exchange messages with the same server concurrently.

Suggestion:
*StringTokenizer* class or the *String.split* method may be used to break a String type message consisting of words (tokens) and then be able to process the extracted words. https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html

# CA- MARKING SCHEME

## IMPLEMENTATION 75%

| | |
|---|---|
| Client App - Connection to Server establishment and mechanism for sending and receiving multiple messages over the network connection | 10% |
| Server App - Internet connection establishment and mechanism for receiving and sending messages over the network connection to a client | 10% |
| Use of Threads for concurrent Client-Server communications | 10% |
| Identify Action 1 and provide functionality to perform action 1 | 10% |
| Identify Action 2 and provide functionality to perform action 2 | 10% |
| Define IncorrectActionException error and correct usage in the application | 10% |
| Exceptions are properly handled in the apps. | 5% |
| Mechanism to handle shared resource(s) among Threads | 5% |
| Use of a suitable data collection to store and manipulate events/todo tasks received from clients | 5% |
| **TESTING 25% (Tasks are working correct during app running)** | |
| Client app reads correct messages from keyboard and transmits them to server | 5% |
| Server app processes the received message and identifies corect the action | 5% |
| Action 1 is executed correct ( e.g. add/list/remove) | 5% |
| Action 2 is executed correct ( e.g. add/list/remove) | 5% |
| Concurent Client-Server communications | 5% |