Assignment 1

CSC241- Object Oriented Programing

CLO3

Due Date: 15-11-24 **Maximum Marks:15**

Console-Based Messaging Application

The goal of this assignment is to implement object-oriented programming (OOP) principles in designing and developing a console-based messaging application. This project will introduce you to **socket programming** and **multithreading**, enabling two PCs to communicate with each other. You will learn and apply these concepts as part of this assignment. The application will simulate a real-time messaging environment using `ArrayList` to store and manipulate data, without relying on file handling.

Assignment Overview

You are tasked to create a console-based messaging application in Java that facilitates communication between two PCs. The application will be implemented using **basic socket programming**, **threads**, and **Java Collections Framework**. The focus will be on designing meaningful classes to represent key components in a messaging system, such as contacts, messages, senders, receivers, and message operations (e.g., send, receive, search, delete). The application should be user-friendly, interactive, and robust, with proper exception handling.

Requirements:

1. Functional Requirements:

Your application must support the following operations:

1. **Contacts Management**:
    - Create an initial list of contacts stored in an `ArrayList` (e.g., names and IDs).
    - Display all available contacts.
2. **Messages Management**:
    - Populate an initial `ArrayList` of messages to simulate message history.
    - Each message should have attributes like:
        - Sender ID
        - Receiver ID

        - Message text
        - Timestamp
        - Status (e.g., read/unread)

3. **Message Operations**:
    - Send a message to a contact.
    - Receive a message from a contact.
    - Mark messages as read/unread.
    - Search for a message by text or sender.
    - Delete all messages for a specific contact.
    - Display all messages for a specific contact.
4. **Thread and Socket Programming**:
    - Implement basic socket programming to enable two-way communication between two PCs.
    - Use threads to handle sending and receiving messages concurrently.
5. **Application Workflow**:
    - Start with a **welcome message** and guide users through the available options.
    - Provide a text-based menu for sending, receiving, and managing messages.
    - End with a **goodbye message** upon exiting the application.

2. Non-Functional Requirements:

1. **ArrayLists**:
    - Use `ArrayList` to store and manage contacts and messages.
    - Populate these lists with initial data for simulation purposes.
2. **Exception Handling**:
    - Ensure the application does not crash due to invalid inputs or network errors.
    - Provide clear and meaningful error messages (e.g., "Contact not found," "Message sending failed").
3. **User Interface**:
    - Design a text-based, menu-driven interface.
    - Provide clear instructions for each operation (e.g., "Enter contact ID to send a message").

Key Tools and Concepts:

1. **Java Socket Programming**:
    - Learn to create server and client sockets to enable two-way communication between machines.
2. **Multithreading**:
    - Use threads to handle simultaneous sending and receiving of messages.
3. **Collections Framework**:
    - Use `ArrayList` to store and manage data like contacts and messages.
4. **Object-Oriented Design**:
    - Create the following suggested classes and design them meaningfully:

        - **Contact**:
            - Attributes: Contact ID, name.
            - Methods: Display contact details.
        - **Message**:
            - Attributes: Sender ID, receiver ID, text, timestamp, status (read/unread).
            - Methods: Update status, display message details.
        - **Sender**:
            - Methods: Send a message to a contact.
        - **Receiver**:
            - Methods: Receive a message from a contact.
        - **MessageManager**:
            - Methods: Search messages, delete messages, mark messages as read/unread, display all messages.

5. **Exception Handling**:
    - Handle invalid inputs, socket connection failures, and unexpected errors gracefully.

Group Work and Submission Instructions:

1. **Group Composition**:
    - Work in groups of **two students**.
    - Each student must upload the code to their **own GitHub repository**.
2. **GitHub Requirements**:
    - Initialize a GitHub repository and upload your project code.
    - Do **not** update your repository after the submission deadline.
3. **Submission Link**:
    - Submit the following details via the provided Google Form:
        - GitHub repository link
        - Registration number
        - Group member IDs
    - The Google Form link will be shared in the classroom.

Deliverables:

1. **Codebase**:
    - A well-structured Java project folder containing:
        - Class files for all key components.
        - A `main` class to start and manage the application.
2. **GitHub Repository**:
    - Each student must upload their own code to a GitHub repository.
    - Clearly document your code and repository with a `README.md` file.

Learning Objectives:

1. Understand the basics of **socket programming** to enable machine-to-machine communication.
2. Learn **multithreading** to manage concurrent tasks.
3. Use Java’s **Collections Framework** to manage data.
4. Implement robust error handling using Java’s **exception handling mechanisms**.

Submission Deadline:

The assignment must be submitted by **15-112024**.

https://forms.gle/dXtAUNnaX1seGn4D6