

**Group Members: Fatima Khalid (SP24-BSE-132)**

**Ayesha Arshad (SP24-BSE-019)**

**Course name: Object Oriented Programming**

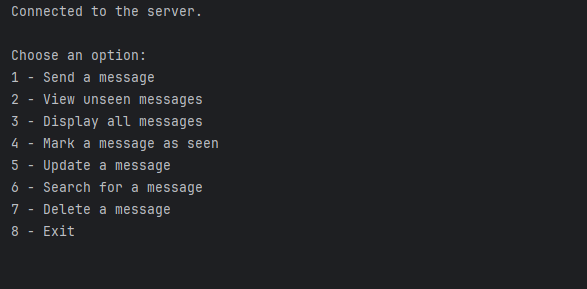
**Submitted to: Muhammad Shahid Bhatti**

**Assignment no: 02**

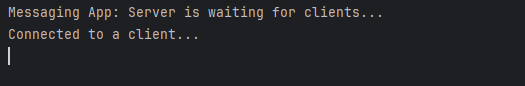
**Due Date: 14-11-2024**

**Messaging App:**

**Client Console:**

****A console is an interface that allows users to interact with a computer system or application by typing text commands and receiving text-based feedback. It is often used for debugging, managing systems, or running command-line programs. In programming, the console is typically accessed through a terminal or command-line interface (CLI), where users can input commands and see outputs directly. In languages like Java, the console is often used to capture user input (via classes like Scanner) and display output (using System.out.println()). The console provides a direct, no-graphical way of interacting with programs, especially during development.

**Server Console:**

A console is an interface that allows users to interact with a computer system or application by typing text commands and receiving text-based feedback. It is often used for debugging, managing systems, or running command-line programs. In programming, the console is typically accessed through a terminal or command-line interface (CLI), where users can input commands and see outputs directly. In languages like Java, the console is often used to capture user input (via classes like Scanner) and display output (using System.out.println ()).

**Main Functions in My App:**

Following are the functions in my code:

1. **Receive Message:**

**receiveMessages ()**

* **Purpose**: This is the main method that waits for client connections and handles various user interactions (choices like sending, viewing, updating, and deleting messages).
* **How it works**:
  + It creates a ServerSocket that listens for client connections on port 1234.
  + Once a client connects, it creates a BufferedReader to read input and a PrintWriter to send output (both using UTF-8 encoding).
  + It then enters a loop where it continuously reads the client's choice (1-7) and processes the request by calling corresponding methods (e.g., receiveMessage, viewUnseenMessages, etc.).

**receiveMessage (BufferedReader in)**

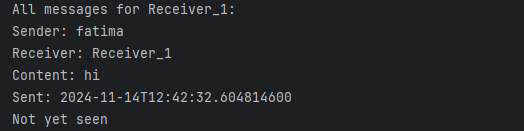
* **Purpose**: Receives a new message from the client and stores it in the appropriate message slot.
* **How it works**:
  + The method reads the sender's name, the receiver's index (to identify the recipient), and the message content from the client.
  + It checks if the message box for the receiver is full (messageCounts[receiverIndex] < MAX\_MESSAGES). If there’s space, it creates a new Message object and stores it in the messages array. The count of messages for that receiver is updated.
  + If the receiver's inbox is full, it prints a message indicating that no more messages can be added.

1. **Display all Messages:**

**displayAllMessages(BufferedReader in)**

* **Purpose**: Displays all messages for a specific receiver, including both seen and unseen messages.
* **How it works**:
  + It reads the receiver index from the client input.
  + It then loops through all the messages stored for that receiver and prints each one to the console.

**Server’s console:**

****

**Client’s Console:**

****

1. **View unseen messages:**

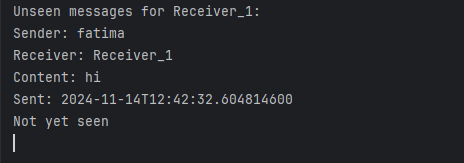
**viewUnseenMessages(BufferedReader in)**

* **Purpose**: Displays the unseen (unread) messages for a specific receiver.
* **How it works**:
  + The receiver index is read from the client input.
  + It loops through the messages for that receiver and checks if each message is marked as unseen (!messages[receiverIndex][i].isSeen()).
  + Unseen messages are printed to the console.

**Client’s Console:**

****

**Server’s Console:**

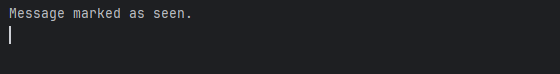
****

1. **Mark message as seen:**

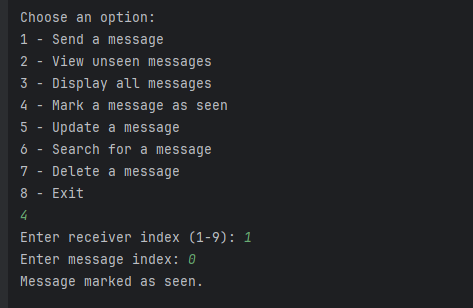
**markMessageAsSeen(BufferedReader in)**

* **Purpose**: Marks a specific message as "seen."
* **How it works**:
  + It reads the receiver index and message index from the client input.
  + It checks if the specified indices are valid (i.e., if the message exists for the given receiver).
  + If valid, it calls markAsSeen() on the specified message, which sets the message’s seen status to true and records the time it was marked as seen.

**Server’s Console:**

****

**Client’s Console:**

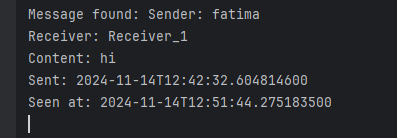
****

1. **Search Message:**

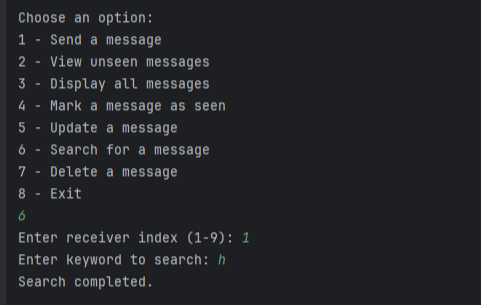
**searchMessage(BufferedReader in)**

* **Purpose**: Searches for messages containing a specific keyword within a specific receiver's messages.
* **How it works**:
  + It reads the receiver index and keyword from the client.
  + It loops through all the messages for that receiver and checks if the message content contains the specified keyword.
  + If any messages match, they are printed to the console. If no matches are found, a message is printed indicating no messages were found with the given keyword.

**Server’s Console:**

****

**Client’s Console:**

****

1. **Delete Message:**

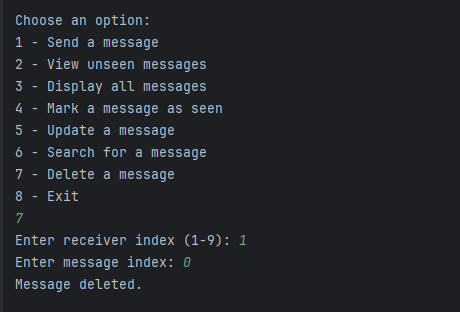
**deleteMessage(BufferedReader in)**

* **Purpose**: Deletes a specific message from a receiver's inbox.
* **How it works**:
  + It reads the receiver index and message index from the client.
  + It checks if the provided indices are valid.
  + If valid, it shifts the messages after the deleted message one position to the left in the messages array, effectively removing the specified message. The count of messages for the receiver is then updated.

**Server’s Console:**

****

**Client’s Console:**

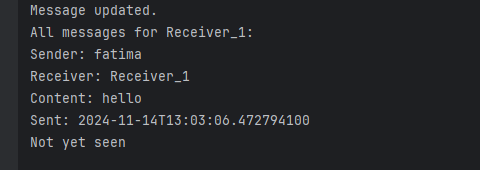
****

1. **Update Message:**

**updateMessage(BufferedReader in)**

* **Purpose**: Updates the content of an existing message.
* **How it works**:
  + It reads the receiver index, message index, and new content from the client.
  + It checks if the provided message index is valid for the receiver.
  + If valid, it calls the setContent() method on the specified message, which updates the message content to the new one provided by the client.

**Server’s Console:**

****

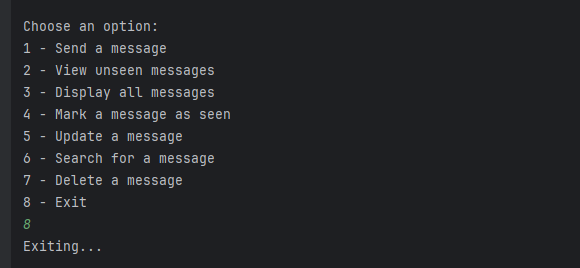
**Client’s Console:**

****

1. **Exit Program:**

**Server’s Console:**

****

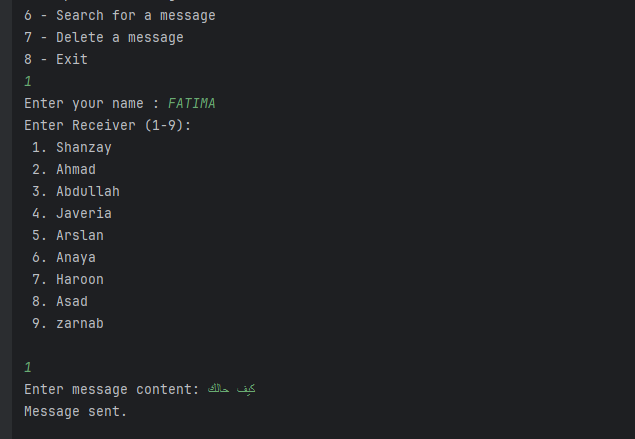
**Client’s Console:**

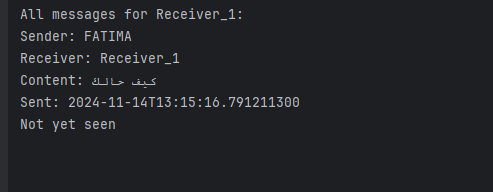
**Language Selection:**

UTF-8 is a character encoding standard that is widely used to ensure compatibility with a variety of languages and special characters across different systems. It can encode all characters in the Unicode character set, making it versatile and suitable for international text processing.

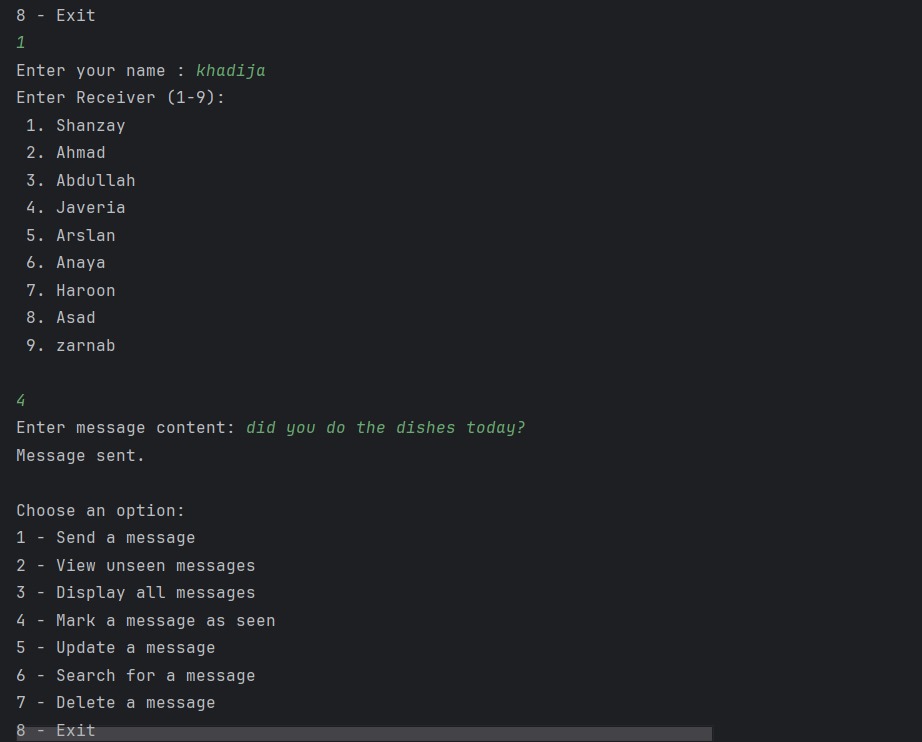
In the context of your Java server application, using UTF-8 is important because it ensures that special characters (like accents, non-Latin characters, and symbols) are properly transmitted and displayed, both in client-server communications and in console outputs.

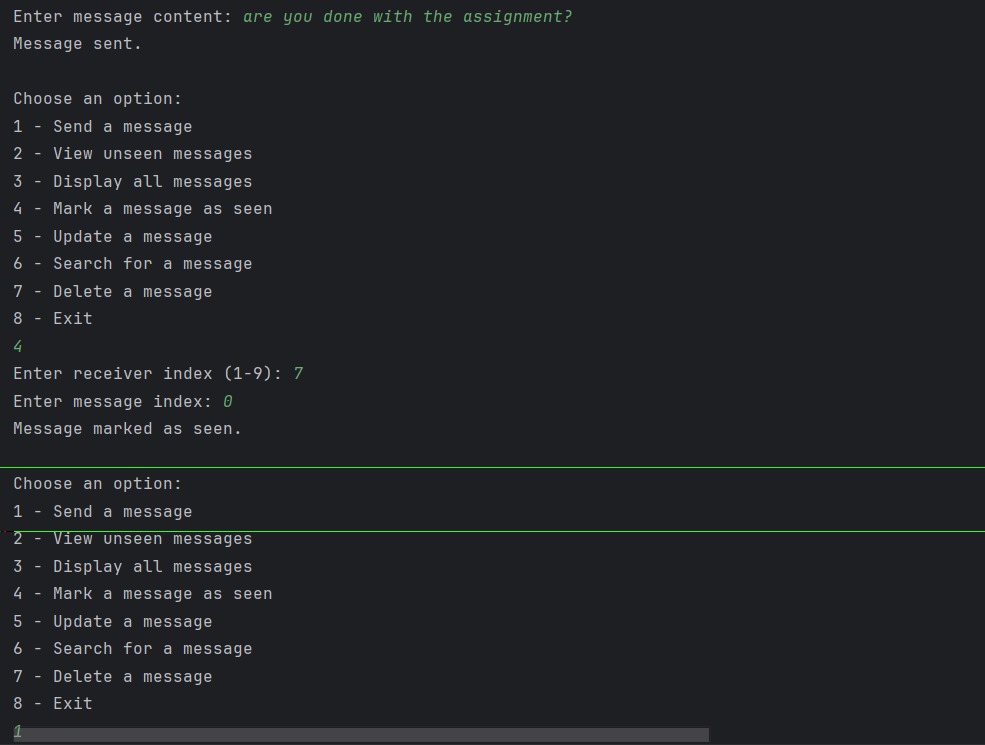
**Console:**

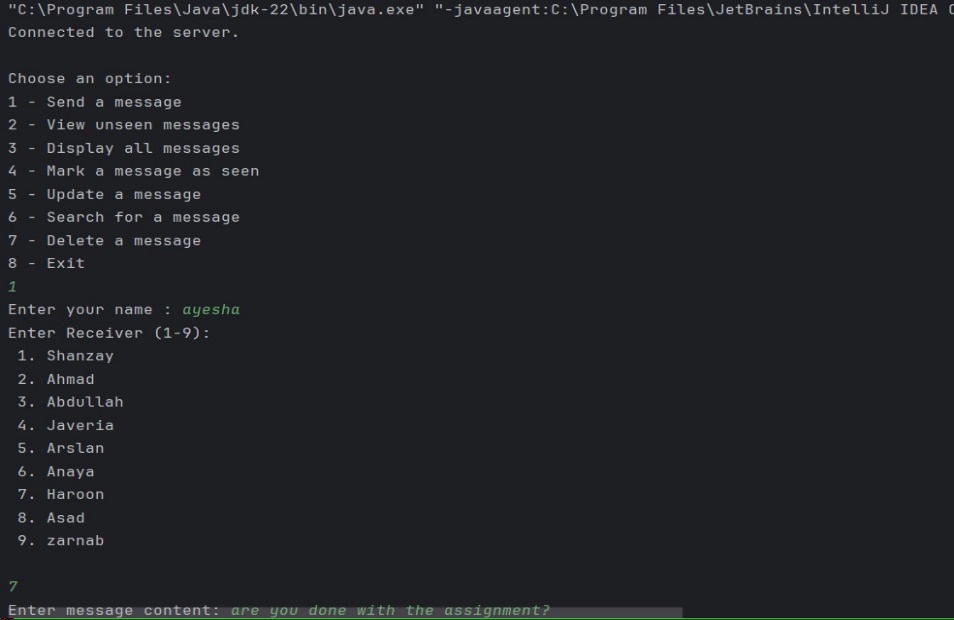
****

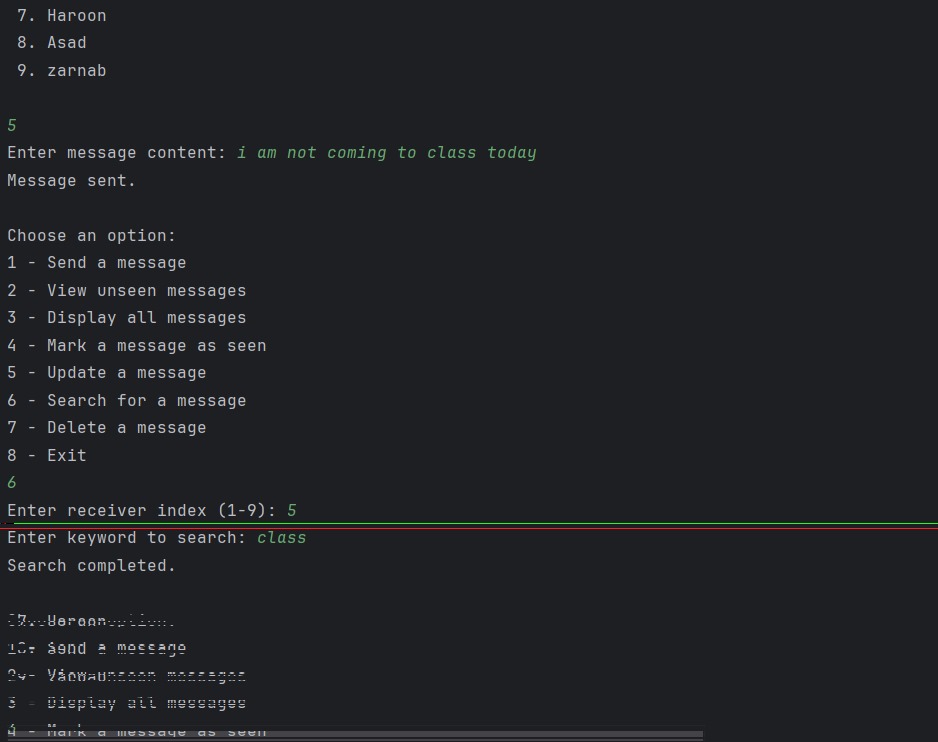
****

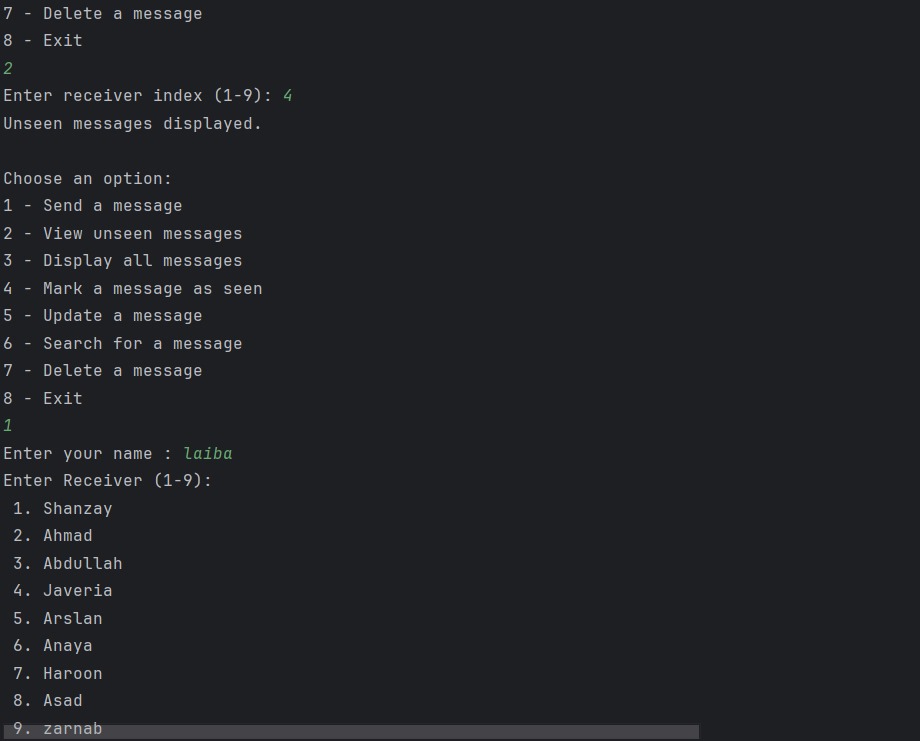
**Whole Output:**

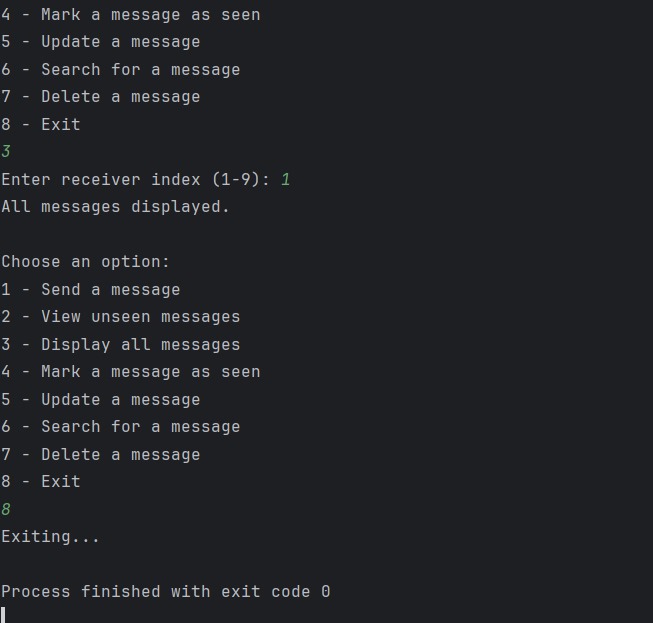
****

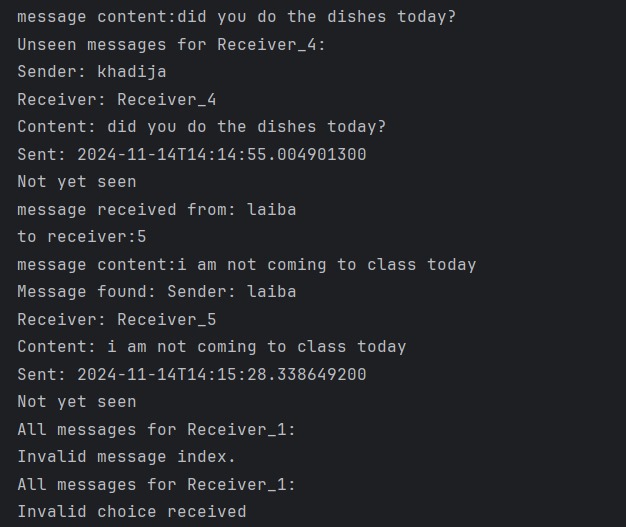
****

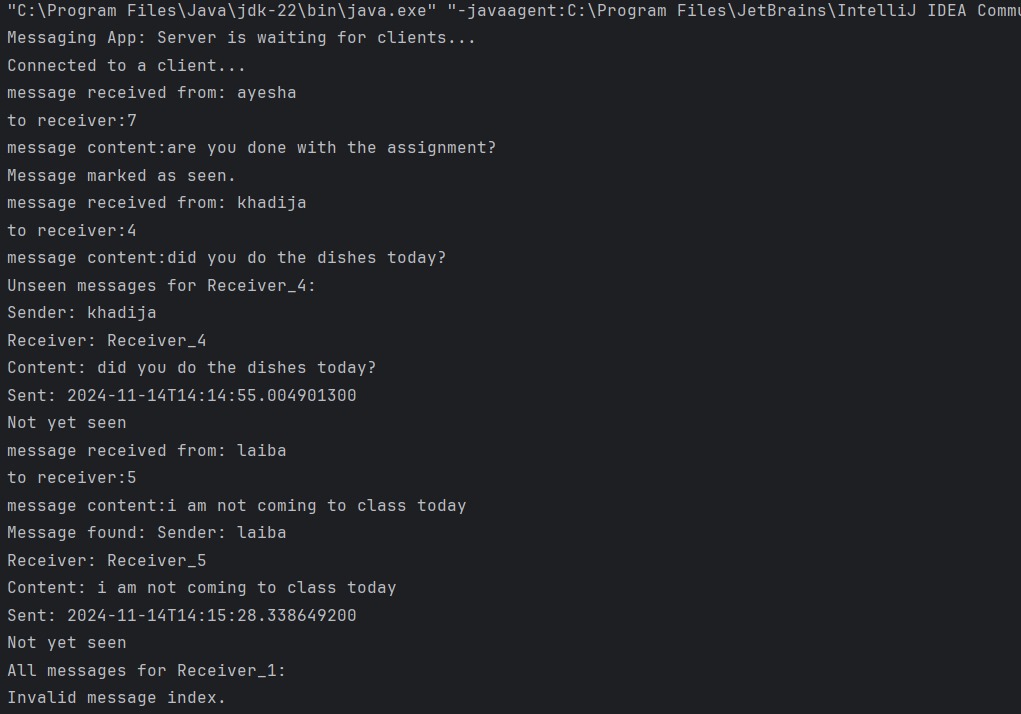
****

****

****

****

****

****