|  |
| --- |
|  |

|  |  |
| --- | --- |
| **Submitted by** | **Anas Ilyas, Hamza Amir** |
| **Subject** | **OOP** |
| **Assignment** | **Messaging app** |
| **Due Date** | **Nov 14,2024.** |

**Submitted to**

|  |  |
| --- | --- |
| **Moderator** | **Sir Shahid Bhatti** |

**Code Explanation**

**Overview**

The code implements a message history management system with the following key components:

**Messaging History:**

Handles saving, loading, and displaying the message history.

**User class:**

Represents a user with basic functionalities like adding contacts, blocking/unblocking, and sending/receiving messages.

**Premium User class:**

Extends User and adds editing and deleting message capabilities.

**Message class:**

Represents a message with metadata like content, timestamp, sender, receiver, and language.

**Test class:**

Demonstrates the usage of the system.

**Detailed Explanation:**

**Message History Class**

**- Save Message(Message):**

Saves a new message to the history file.

**- Load Message():**

Reads the message history from the file.

**- Display History():**

Prints the entire message history.

**- Save All (ArrayList<String>):**

Overwrites the history file with the provided messages.

**User Class**

**- Add Contact(String, String):**

Adds a new contact.

**- Block User(User):**

Blocks a user, removing them from contacts.

**- Unblock User(User):**

Unblocks a user, adding them back to contacts.

**- Send Request(User):**

Sends a message request to a user.

**- accept Request(User):**

Accepts a message request, adding the user to contacts.

**- Receive Request(User):**

Notifies the user of a received request.

**- Send Message(String, String):**

Sends a message to a contact, if not blocked.

**Premium User Class**

**- Edit Message(String, String):**

Edits a message in the history, if sent by the user.

**- Delete Message(String):**

Deletes a message from the history, if sent by the user.

**Message Class**

- Represents a message with content, timestamp, sender, receiver, and language.

- Provides methods to get the current time and detect the message language.

**Chat Server Class Overview (Short Version)**

The Chat sever class is a multi-client chat server that supports both **Base** and **Premium** users. It listens on port 8080 and handles client interactions via separate threads (ClientHandler). Each client can register as a Base or Premium user, with Premium users gaining access to additional features like editing and deleting messages.

**Key Features:**

* **Multi-client Support**: Handles multiple clients simultaneously through threads.
* **User Registration**: Clients choose between Base or Premium user types and set a username.
* **Actions**: Users can add contacts, send messages, and view message history. Premium users can also edit or delete messages.
* **Menu-Driven Interface**: Clients are presented with options to choose actions.
* **Message History**: Users can view past messages stored on the server.
* **Graceful Shutdown**: Clients can exit the chat at any time.

**Chat Client Class Overview**

The Chat Client class is a simple Java-based client for communicating with a server through sockets. It connects to a server running on the local machine (localhost) on port 8080. The program facilitates user interaction, sending and receiving messages .

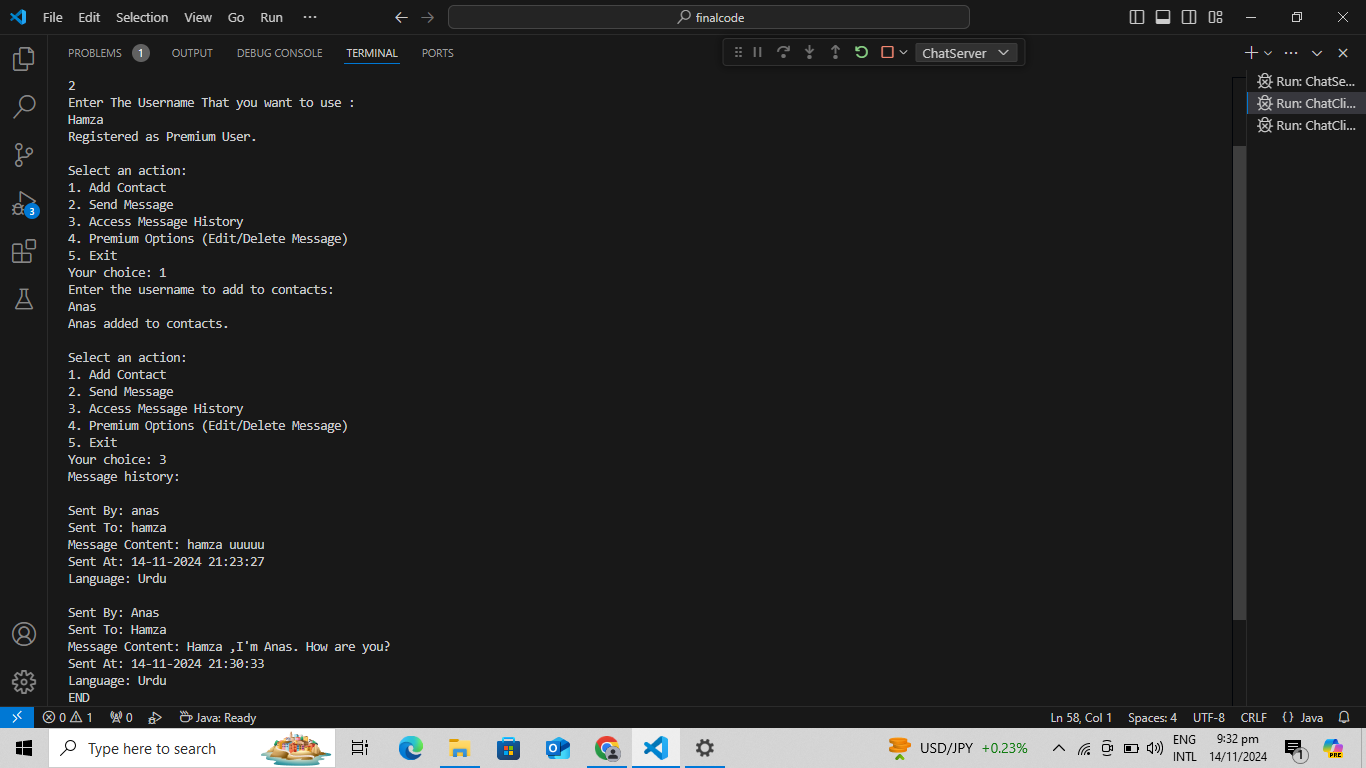
Upon connection, the client first reads and displays messages sent from the server, prompting the user for input. The user specifies their type and name, which is then sent to the server. After the initial interaction, the client enters a loop where it continuously listens for and displays server responses. It allows the user to make choices (e.g., selecting options in a menu) and sends those choices back to the server. .The program terminates if the server sends an "END" message, indicating that no further communication is required.

**Key Features:**

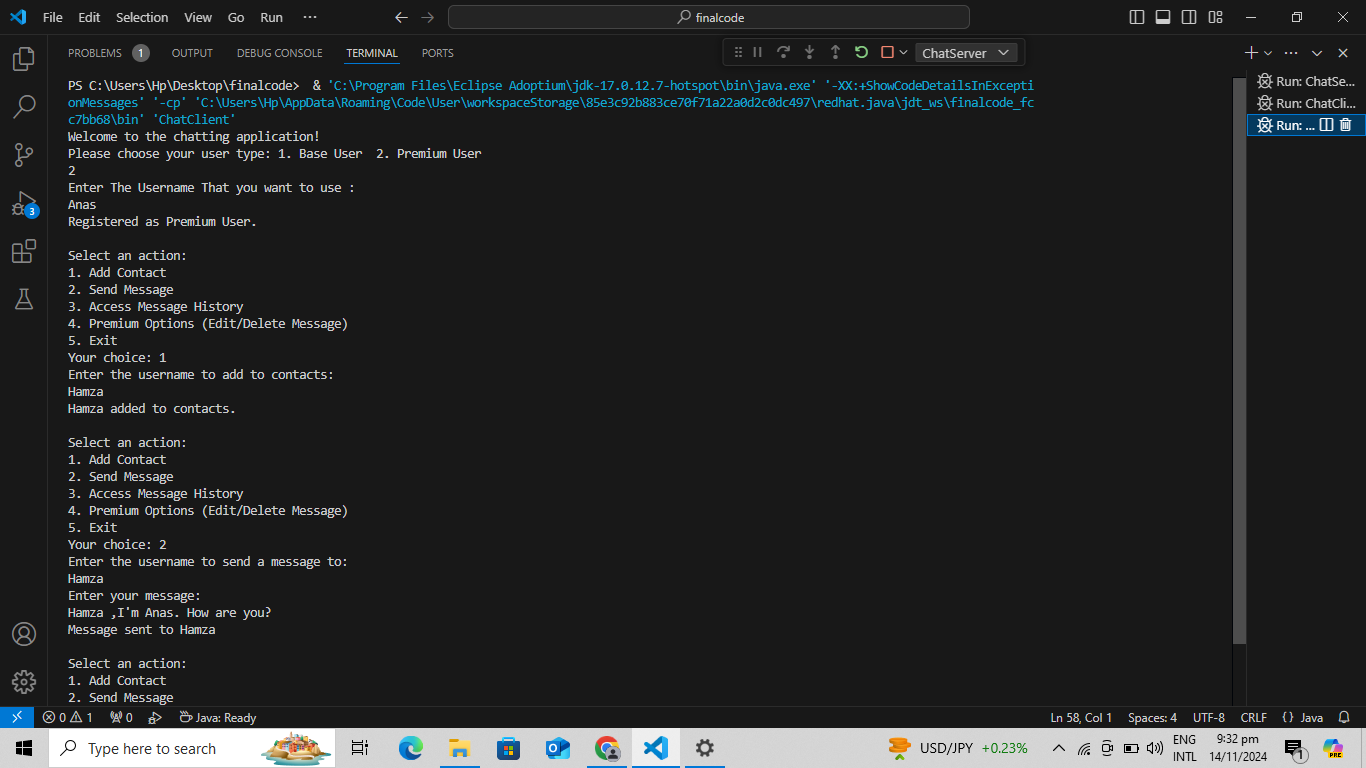
1. **Socket Communication**: The client uses a socket connection to send and receive data from the server.
2. **User Interaction:** It accepts input from the user for various prompts and sends the responses to the server.
3. **Looped Communication**: The client repeatedly listens for server messages, making choices and sending responses as directed.
4. **Error Handling**: Basic error handling is implemented to catch IOException.

In summary, this class implements a simple interactive client-server model where the client engages in a dialogue with the server, responding to prompts and sending choices or inputs as required.

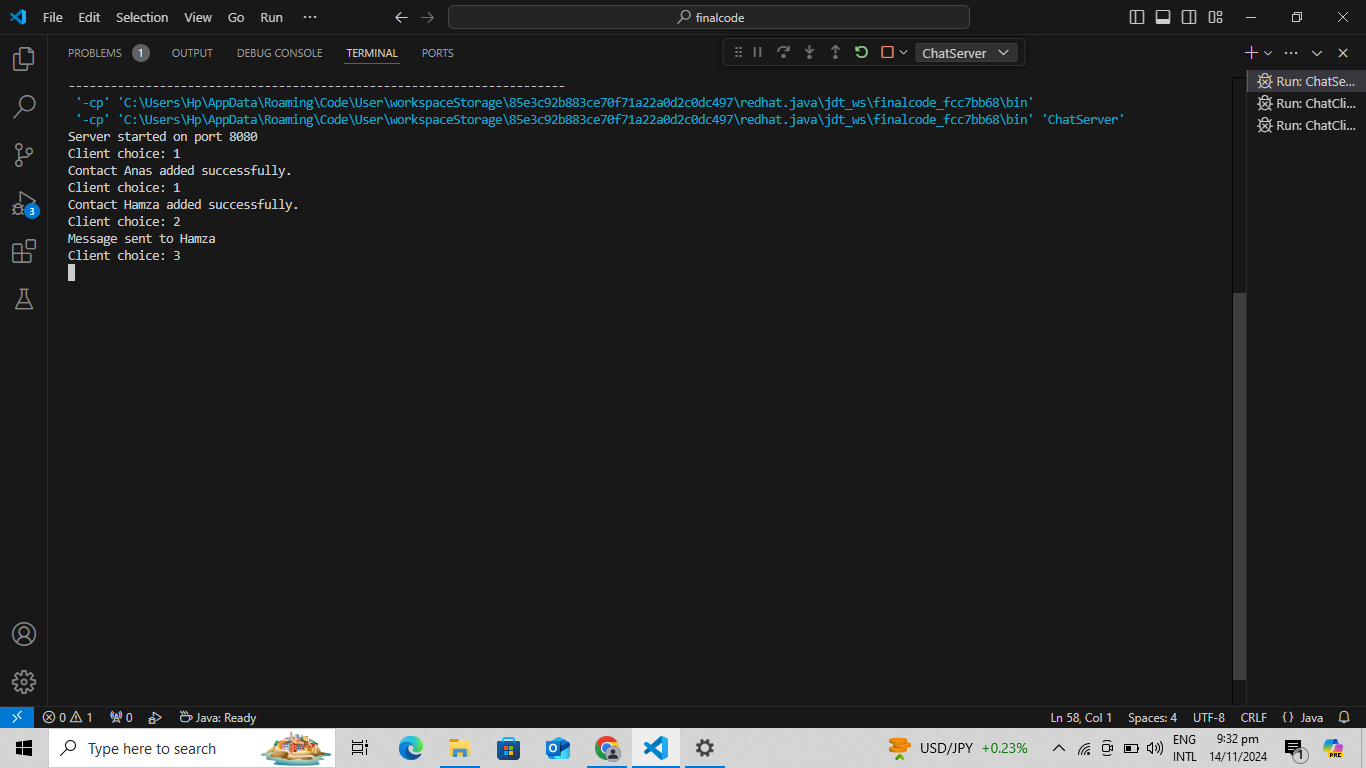
**Client 1**

****

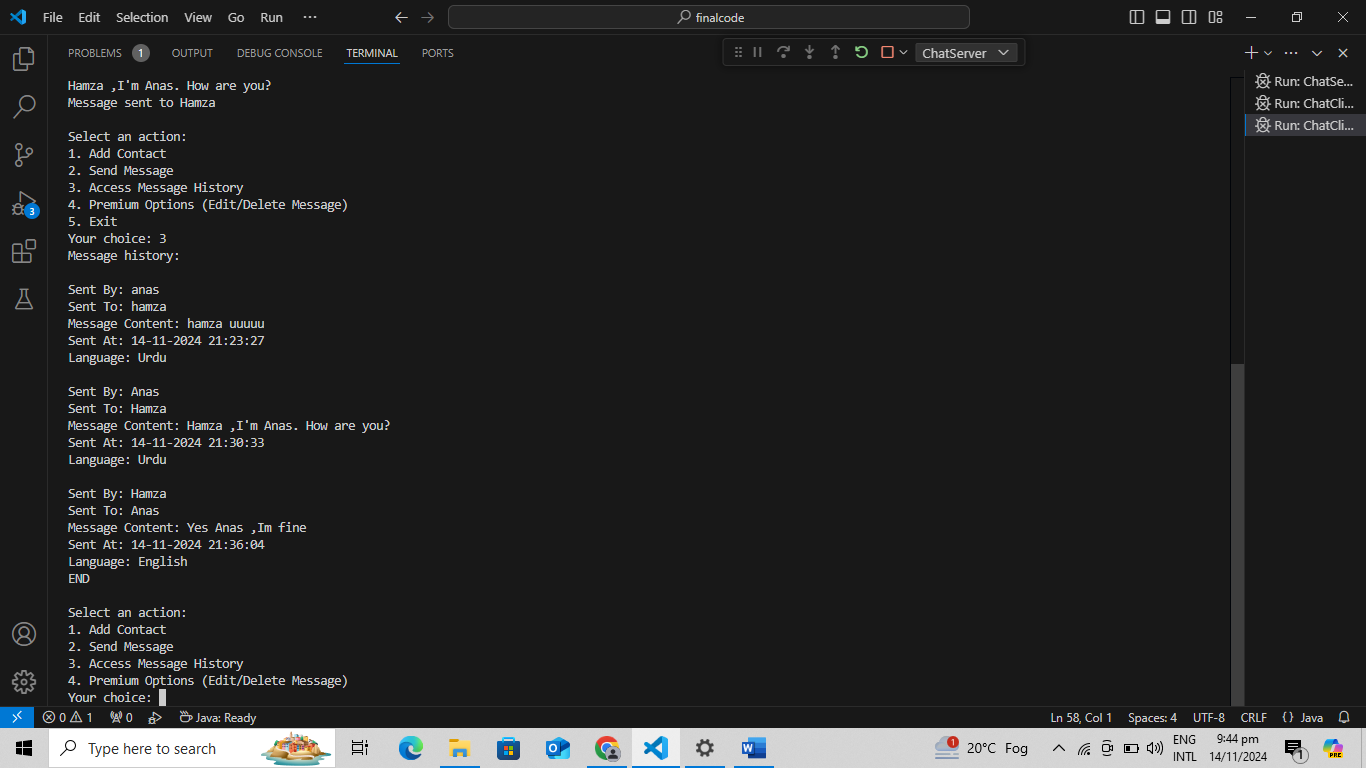
**Client 2:**



**Server:**

****

**Message History Between Two Users:**

****

**Explanation:**

Two users interact with each other by using the server. The server handles them by using ClientHandler as it is extending thread so anew thread is created every time a new client class is created and connects with server socket of the ChatServer class