COURSE INSTRUCTOR

Sir Shahid Bhatti

Socket programming documentation

COLLABORATION PROJECT

COLLABORATORS

HAFIZ MUHAMMAD AMMAR QASIM SP24-BSE-029-B

HAREEM MOHAL SP24-BSE-133-B

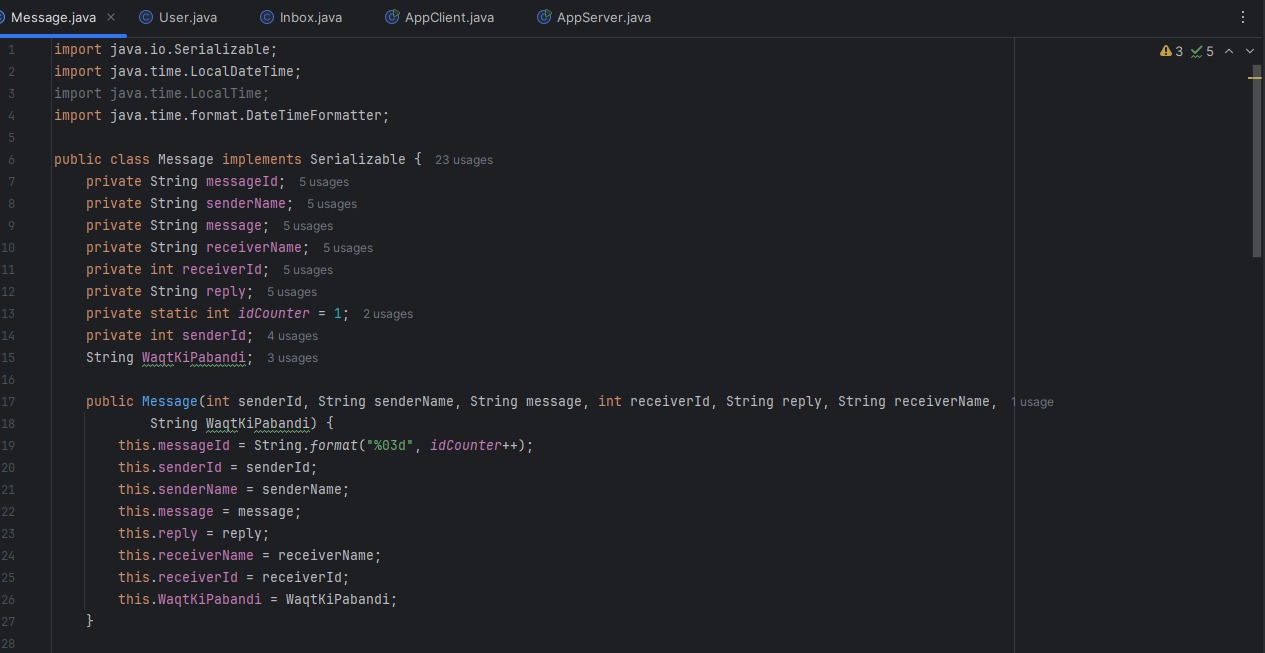
**Messaging System Documentation**

**Overview**

This program implements a simple client-server messaging application that allows users to send, receive, reply to, and manage messages. It consists of several classes, including Message, User, Inbox, and two main application classes: AppServer and AppClient. The AppServer listens for connections and communicates with the AppClient via sockets, allowing serialized Message objects to be transmitted between the client and server.

**Class Descriptions**

1. **Message Class**



* **Purpose**: Represents an individual message, encapsulating the sender and receiver information, message content, and timestamp. The class implements Serializable, allowing Message objects to be transmitted over a network.

**Attributes:**

* messageId (String): Unique identifier for each message, automatically incremented with idCounter.
* senderName (String): Name of the message sender.
* message (String): Content of the message.
* receiverName (String): Name of the message recipient.
* receiverId (int): Unique ID of the recipient.
* reply (String): Optional reply to the message.
* WaqtKiPabandi (String): Timestamp indicating when the message was created.
* senderId (int): Unique ID of the sender.

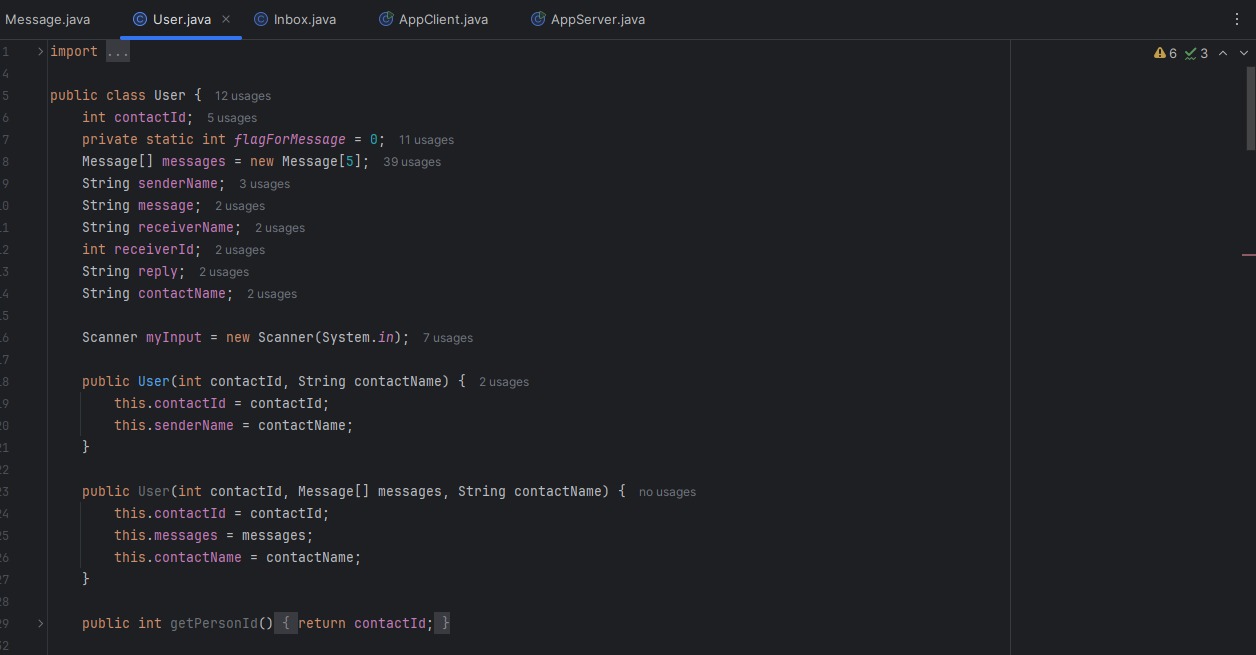
**Constructors:**

* Message(int senderId, String senderName, String message, int receiverId, String reply, String receiverName, String WaqtKiPabandi): Initializes a new Message with all provided details.
* Message(Message sentMsg): Copy constructor that creates a new Message by copying details from an existing message.

**Methods:**

* **Getters and Setters**: For accessing and modifying each attribute (e.g., getSenderName(), setReply()).
* toString(): Returns a string representation of the message details, formatted for easy reading.

**2. User Class**

****

* **Purpose**: Represents a user with a unique ID and an array of Message objects for managing sent and received messages.

**Attributes:**

* contactId (int): Unique identifier for the user.
* messages (Message[]): Array to store up to five Message objects.
* senderName, message, receiverName, receiverId, reply, contactName (String): Fields for holding user input during message creation and management.

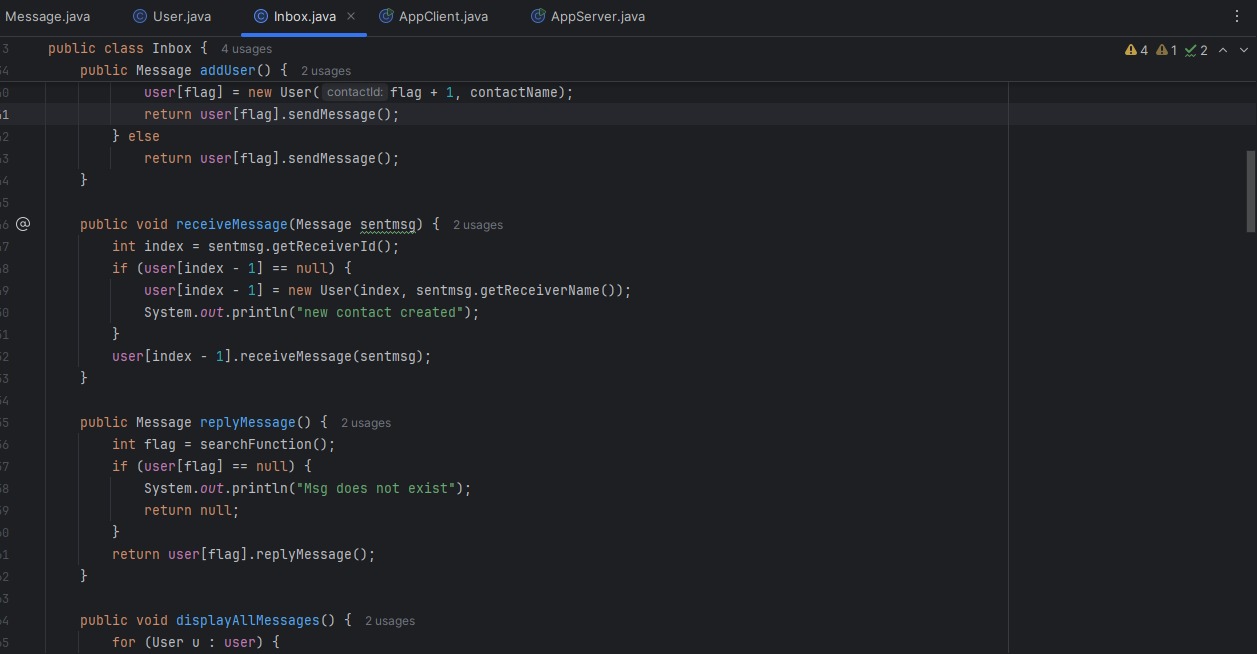
**Constructors:**

* User(int contactId, String contactName): Initializes a User with a contact ID and name.
* User(int contactId, Message[] messages, String contactName): Initializes a User with a contact ID, a set of messages, and a name.

**Methods:**

* **Messaging Functions**:
  + sendMessage(): Creates a new Message based on user input and adds it to the user's messages array.
  + receiveMessage(Message sentmsg): Receives a Message object and adds it to the user's messages array.
  + replyMessage(): Allows the user to reply to an existing message by matching message IDs.
* **Display and Search Functions**:
  + displayAllMessages(): Prints all messages in the user's messages array.
  + searchByContent(String msgContent): Searches for messages containing a specified content string.
  + byMsgId(): Searches for a message by its ID.
  + display(): Displays a specific message found by ID.
* **Delete Functions**:
  + deleteMessages(): Deletes a message based on its ID.

**3. Inbox Class**

****

* **Purpose**: Represents a collection of User objects, simulating an inbox for managing contacts and messages.

**Attributes:**

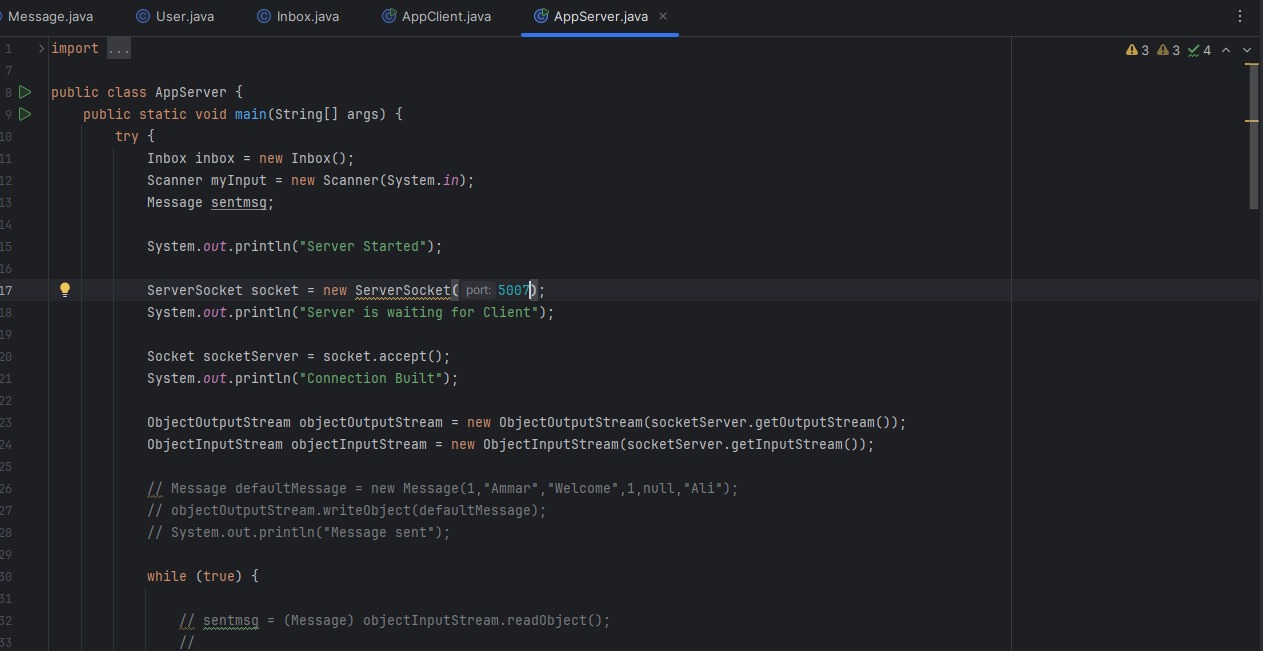
* personId (int): ID of the inbox owner.
* user (User[]): Array to store up to five User objects.

**Methods:**

* **User and Message Management**:
  + addUser(): Prompts for a new user or sends a message from an existing user.
  + receiveMessage(Message sentmsg): Receives a message and processes it for the appropriate user.
  + replyMessage(): Allows a user to reply to a message from an existing contact.
* **Display and Search Functions**:
  + displayAllMessages(): Displays messages for all contacts.
  + specificContactMsgs(): Displays messages for a specific contact by name or ID.
  + choiceForSearch(): Provides options to search for messages by name, ID, or message content.
* **Delete Functions**:
  + deleteMessage(): Prompts the user to delete a specific message or an entire chat.
  + deleteWholeChat(): Deletes all messages associated with a specific contact.
  + deleteSpecificMessage(): Deletes a specific message based on ID.
* **Helper Methods**:
  + byName(): Finds a user by name.
  + byContactId(): Finds a user by contact ID.
  + byContent(): Searches for messages containing specific content across all contacts.

**Application Classes**

**4. AppServer Class**

****

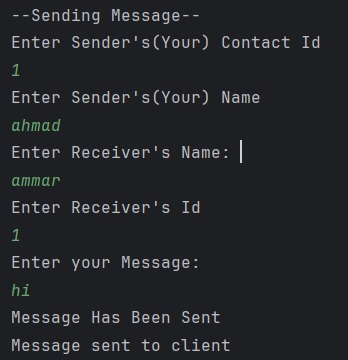
* **Purpose**: Acts as the server-side application, managing client connections, sending and receiving messages, and handling user operations via a menu-based interface.

**Workflow:**

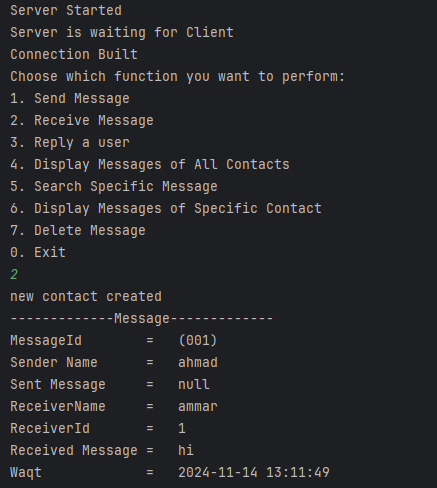
1. **Server Setup**: Starts a ServerSocket on port 5007, awaiting client connections.
2. **Connection Establishment**: Accepts a client connection, initializes input/output streams, and enters a main loop for user operations.
3. **Message Handling**:
   * Processes messages received from the client and provides server-side management options (e.g., send, receive, reply, delete messages).
4. **Main Menu Options**:
   * Provides a set of options to manage messages, such as sending messages, displaying messages, searching, and deleting messages.

**Menu Functions:**

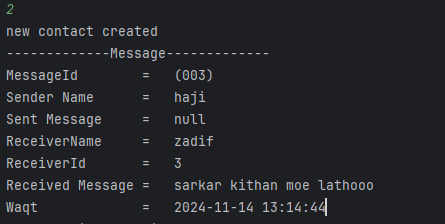
* **Option 1**: **Sends a new message** using inbox.addUser().

****

* **Option 2**: **Receives a message** from the client and adds it to the inbox.

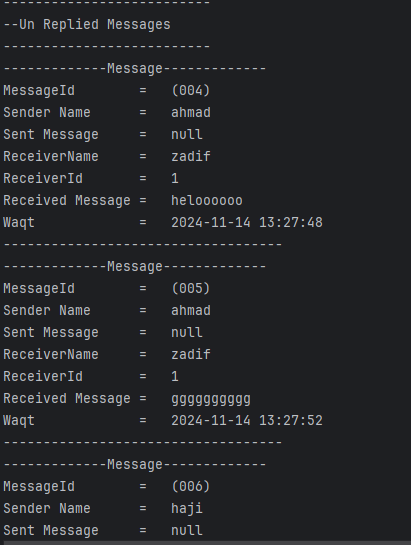




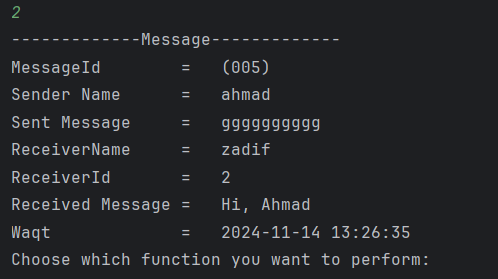


* **Option 3**: Sends a **reply message** to the client using **inbox.replyMessage().**

**Display Un- Replied Messages**

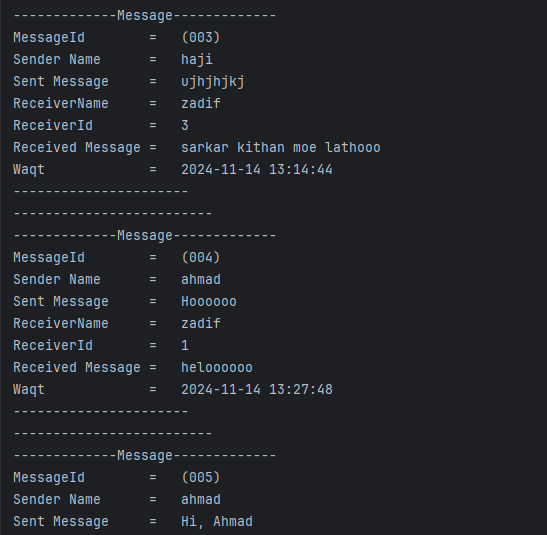


**Replied Message:**

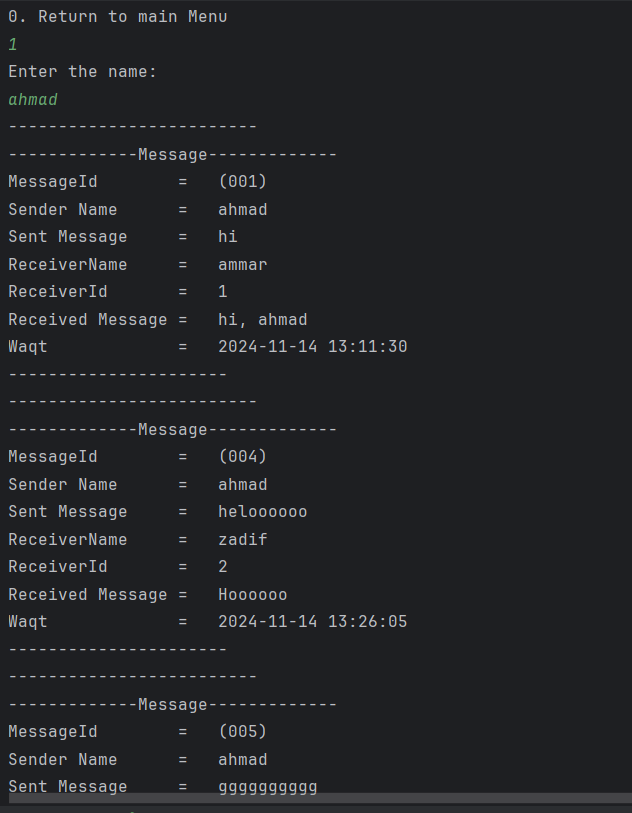
****

* **Option 4**: **Displays all messages** for all contacts.

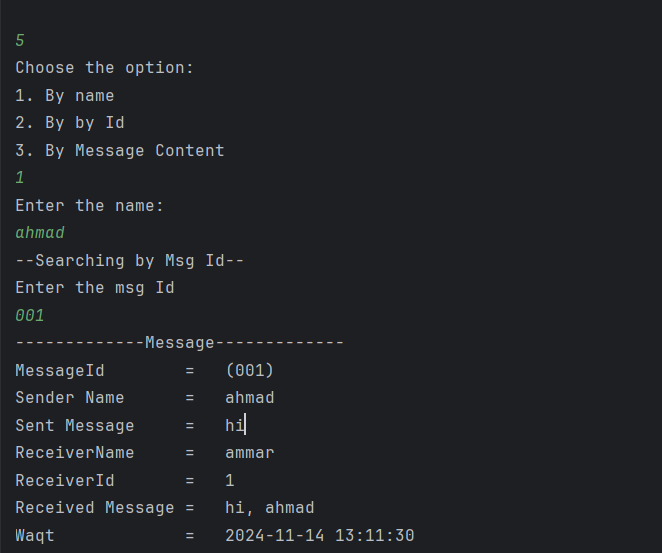
****

****

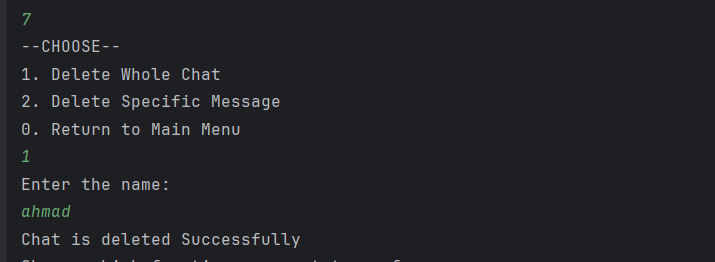
* **Option 5**: **Searches for messages** based on various criteria.

****

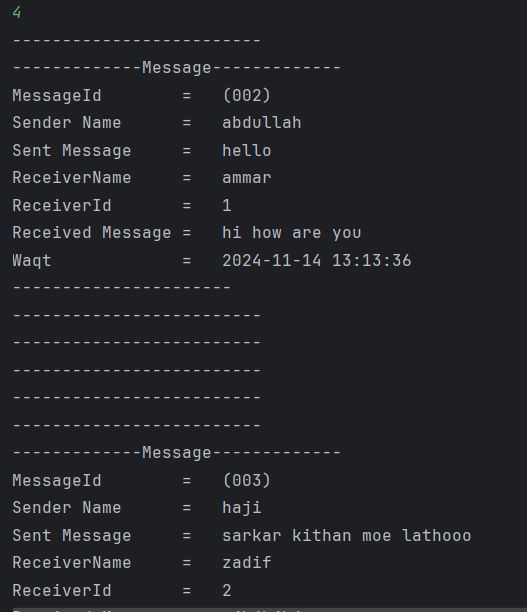
* **Option 6**: **Displays messages** for a specific contact.

****

* **Option 7**: **Deletes messages** or an entire chat.

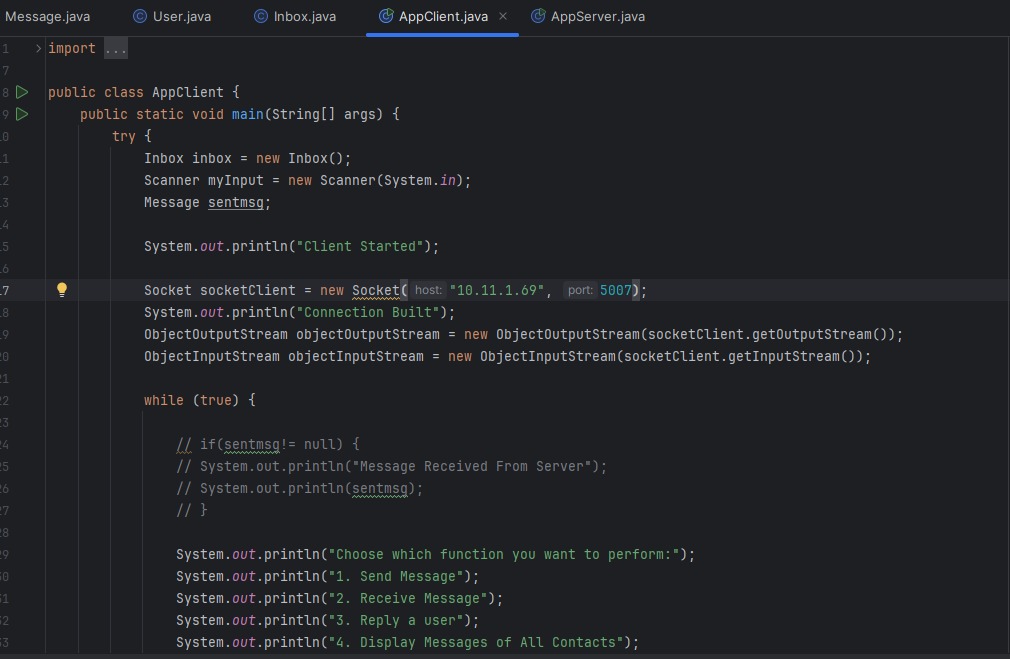
****

Display after deletion where chat is not shown now:

****

* **Option 0**: Closes the connection and exits the server application.

1. **AppClient Class**

****

* **Purpose**: Acts as the client-side application, connecting to the server to send, receive, and manage messages using a similar menu-based interface.

**Workflow:**

1. **Client Setup**: Connects to the server using Socket on localhost at port 5007.
2. **Stream Initialization**: Initializes input/output streams to exchange Message objects with the server.
3. **Message Handling**:
   * Similar to the server, provides options for the client user to manage messages by sending, receiving, and replying.
4. **Main Menu Options**:
   * Presents the same menu options as the server, allowing for bidirectional communication and message management.

**Menu Functions:**

* **Option 1**: Sends a new message using inbox.addUser().
* **Option 2**: Receives a message from the server and adds it to the inbox.
* **Option 3**: Sends a reply message to the server using inbox.replyMessage().
* **Option 4**: Displays all messages for all contacts.
* **Option 5**: Searches for messages based on name, ID, or content.
* **Option 6**: Displays messages for a specific contact.
* **Option 7**: Deletes messages or an entire chat.
* **Option 0**: Closes the client connection and exits the application.

THE END