**Code Description for Messaging Application**

# 1. Server Class

The Server class manages the server side of the messaging application. It handles client connections, manages message transmission, and allows the server to perform operations such as sending, displaying, searching, and deleting messages. This class operates as a console-based server that can interact with a connected client over a network.

## Attributes:

* serverSocket: Manages the server's socket connection, allowing it to listen on a specific port.
* clientSocket: Manages the socket connection with the connected client.
* out: PrintWriter used to send messages from the server to the client.
* in: BufferedReader used to receive messages from the client.
* messageList: A List<Message> that stores all messages sent and received for easy access and management.
* functionalities: An instance of the Functionalities class to enable message operations such as sending, displaying, searching, and deleting messages.

## Methods:

* startServer(int port): Initializes the server on the specified port, waits for a client to connect, and processes commands for different messaging functionalities.
* stopServer(): Safely closes all connections, including serverSocket, clientSocket, in, and out.
* main(String[] args): The entry point for running the server; it starts the server on a specified port (default is 5000).

# 2. Message Class

The Message class models individual messages within the system. It encapsulates essential information about each message, including the recipient, content, timestamp, status, and a unique identifier.

## Attributes:

* receiver: The intended recipient of the message.
* content: The actual text content of the message.
* timestamp: Records the time when the message was created, using LocalDateTime.
* status: A boolean indicating whether the message was 'Sent' (true) or 'Received' (false).
* messageId: A unique identifier for each message, generated based on a static counter.
* counter: A static integer that increments with each new message to ensure unique IDs.

## Methods:

* Message(String receiver, String content, boolean status): Constructor that initializes message attributes, including setting the timestamp and generating a unique message ID.
* toString(): Returns a formatted string representation of the message, including details such as receiver, message ID, status, and timestamp.

# 3. Functionalities Class

The Functionalities class encapsulates operations related to message handling. It provides methods to send, display, search, and delete messages, simplifying the main application logic in both the Server and Client classes.

## Attributes:

* scanner: A Scanner object for reading user input in console-based applications.

## Methods:

* sendMessage(String sender, String receiver, BufferedReader in, PrintWriter out, List<Message> messageList): Manages the process of sending and receiving messages in real time. It enables conversation flow and allows for special actions such as ending the conversation or deleting messages.
* displayMessages(List<Message>): Displays a formatted list of all messages stored in messageList, showing their details and order.
* searchMessages(List<Message>): Allows searching through the messageList by a user-provided keyword.
* deleteMessage(List<Message>, PrintWriter): Deletes a specific message by prompting the user to select a message and notifies the client about the deletion.
* deleteMessageById(List<Message>, String): Deletes a message based on its unique ID, used when a client initiates a deletion request.

# 4. Client Class

The Client class handles the client side of the messaging application, establishing a network connection to the server, managing message sending and receiving, and offering similar message management functionalities as provided on the server side.

## Attributes:

* socket: Manages the client’s connection to the server’s socket.
* out: PrintWriter used to send messages from the client to the server.
* in: BufferedReader used to receive messages from the server.
* messageList: A List<Message> that stores all messages sent and received by the client.
* functionalities: An instance of the Functionalities class to support message handling on the client side.

## Methods:

* startClient(String ip, int port): Establishes a connection to the server using the provided IP address and port, then initiates a command menu for messaging operations.
* stopClient(): Safely closes all client connections, including socket, in, and out.
* main(String[] args): The entry point for running the client application, connecting to a server at the specified IP address and port.