**Messaging Application Server Documentation**

**Project Overview**

This project implements a server-client messaging application using Java. The server application manages message exchanges, allowing connected clients to interact with the server to send, retrieve, delete, and search messages, as well as to view only seen messages. This project showcases client-server communication through socket programming, message storage and management, and server-side controls using a command-line menu.

**Project Structure**

The application consists of two main classes:

1. **Server**: This class initializes the server socket, accepts client connections, and provides a menu-based interface for managing messages. It interacts with the ServerMessageApp class, which performs specific message handling tasks.
2. **ServerMessageApp**: This class manages core message functionalities, such as sending, displaying, deleting, searching, and setting messages as seen or unseen. It serves as the main message management component for the server.

**Class Descriptions**

**1. Server Class**

The Server class is responsible for initializing the server, waiting for client connections, and displaying a command-line menu to interact with messages.

* **Attributes**:
  + ServerSocket serverSocket: Initializes the server socket that listens for incoming client connections.
  + ServerMessageApp messagingApp: The core message manager used to handle message-related functions.
* **Methods**:
  + **start()**: This method starts the server, waits for a client connection, and displays a menu with options for message management. The server keeps running until the user selects the "Exit" option.
  + **main()**: The main entry point of the application. It creates an instance of the Server class, initializes the server socket, and starts the application.
  + **Menu Options**:
    - **Start Chat**: Begins a chat session with the connected client.
    - **Display All Messages**: Displays all stored messages on the server and marks them as "seen."
    - **Delete Message by ID**: Deletes a specific message based on the provided ID.
    - **Find Message by Substring**: Searches for messages containing a specified substring (case-insensitive).
    - **Display Seen Messages**: Displays only the messages that have been marked as "seen."
    - **Exit**: Closes the server, stops the socket connection, and ends the application.

**2. ServerMessageApp Class**

The ServerMessageApp class performs all operations related to managing messages stored on the server.

* **Attributes**:
  + ArrayList<Message> messages: Stores all received and sent messages.
  + DataOutputStream outputStream: Sends data to the connected client.
  + boolean allMessagesSeen: Tracks whether all messages have been marked as "seen."
* **Methods**:
  + **startChat(Socket socket, Scanner scanner)**: Initiates a chat session with the client, handling message reading and writing between the client and server.
  + **displayAllMessages()**: Shows all stored messages on the server console and marks them as "seen."
  + **deleteMessageById(String id)**: Deletes a message based on the specified message ID.
  + **findMessageBySubstring(String substring)**: Searches for messages containing a specified substring, ignoring case.
  + **displaySeenMessages()**: Displays only the messages marked as "seen."
  + **setAllMessagesSeen(boolean seen)**: Updates the "seen" status of all messages.

**Execution Flow**

1. **Server Initialization**: The Server class is initialized, setting up a ServerSocket to listen for incoming connections on a specified port (15000). When the application starts, the start() method initiates the server and waits for a client to connect.
2. **Client Connection and Interaction**:
   * Once a client connects, the server outputs "Client connected" and provides a menu with options to manage messages.
   * The menu loop runs continuously, allowing the server to perform message operations until the user selects the "Exit" option.
   * Each menu option corresponds to a specific operation in the ServerMessageApp class, allowing the user to interact with stored messages.
3. **Menu Options and Message Management**:
   * **Option 1 - Start Chat**: The startChat() method starts a real-time chat session with the connected client. The server and client exchange messages until the server decides to exit the chat.
   * **Option 2 - Display All Messages**: Shows all messages stored on the server and marks each message as "seen."
   * **Option 3 - Delete Message by ID**: Deletes a message with a specific ID. The user is prompted to enter the ID of the message they want to delete.
   * **Option 4 - Find Message by Substring**: Searches for any messages containing a specified substring, ignoring case. This is useful for quickly locating messages based on keywords.
   * **Option 5 - Display Seen Messages**: Only shows messages marked as "seen." Messages become "seen" when they are displayed using displayAllMessages().
   * **Option 6 - Exit**: Closes the server by breaking out of the menu loop and stops the application.
4. **Return Statement in the Exit Case**:
   * In the "Exit" case (Option 6) of the menu, the return statement is used to immediately exit the start() method, ending the server’s operations.
   * This return statement effectively breaks out of the menu loop, as it stops further execution of the start() method, which results in closing the server socket and ending the server application.
5. **Closing the Server**:
   * When the user selects "Exit," the server outputs "Shutting down server," closes the server socket, and stops all operations. The application terminates at this point.

**Key Features**

1. **Message Storage and Management**: The server stores, displays, deletes, and searches messages, allowing for efficient server-side message management.
2. **Seen/Unseen Status**: The application keeps track of messages marked as "seen" or "unseen." Seen messages are displayed separately using the displaySeenMessages() method.
3. **Real-time Chat**: Allows the server and client to communicate in real-time through sockets, facilitating a live chat feature.
4. **Substring Search**: Provides a search functionality to locate messages based on a specific substring, ignoring case sensitivity.
5. **Menu-Based Control**: The server provides a user-friendly, command-line menu interface for managing messages and handling client communication.

**Summary**

This messaging server application demonstrates the use of Java sockets and multithreading to manage real-time server-client communication and message handling. The server offers various message management features, including storage, display, deletion, and search, while allowing for real-time chat with a connected client. The application is structured to run continuously, providing a stable environment for interacting with messages until the user explicitly closes the server. The return statement in the "Exit" option ensures a clean and orderly shutdown of the server by terminating the start() method and closing the server socket.