**Objective:**

Develop a TCP-based Client-Server service in Java that alternates between sending and receiving messages. The client initiates communication, and the server replies. The client sends a STOP message to terminate the connection, and the server confirms with a TERMINATE message.

**Key Components:**

1. **Client-Server Communication**:
   * **TCP Protocol**: Use sockets to handle communication.
   * **Alternating Messages**: Ensure the client starts communication and messages alternate between client and server.
2. **Message Structure**:
   * **Client to Server**: Includes action (add/remove) and event description.
   * **Server to Client**: Includes event lists or error messages.
3. **Exception Handling**:
   * **IncorrectActionException**: Define a custom exception for handling invalid actions.
4. **Threading**:
   * **Server**: Create a new thread for each client connection to handle concurrent clients.
5. **Event Management**:
   * **Dublin Events Calendar**: Store event descriptions in a memory-based data collection (like ArrayList).

**Functional Requirements:**

1. **Client**:
   * Initiates communication.
   * Sends actions (add, remove) with event details.
   * Sends STOP to terminate connection.
2. **Server**:
   * Listens for client connections.
   * Spawns a new thread for each client.
   * Handles messages from clients.
   * Adds or removes events based on client actions.
   * Sends event lists back to the client.
   * Responds to STOP with TERMINATE and closes connection.

**Example Usage:**

1. **Client Sends**: add; 2 November 2024; 6 pm; Fireworks Dublin City Centre
   * **Server Replies**: 2 November 2024; 12 pm, Food Market IFSC Square; 6 pm, Fireworks Dublin City Centre; 7.30 pm, Jersey Boys Bord Gais Energy Theatre
2. **Client Sends**: remove; 2 November 2024; 6 pm; Fireworks Dublin City Centre
   * **Server Replies**: 2 November 2024; 12 pm, Food Market IFSC Square; 7.30 pm, Jersey Boys Bord Gais Energy Theatre

**Steps to Implement:**

1. **Set Up Server**:
   * Initialize server socket.
   * Accept client connections.
   * Handle each connection in a new thread.
   * Process client messages (add/remove).
   * Implement IncorrectActionException.
2. **Set Up Client**:
   * Connect to the server socket.
   * Send messages from keyboard input.
   * Handle server responses.
3. **Event Storage**:
   * Use an ArrayList to store event descriptions.
   * Methods to add and remove events.
   * Method to get events by date.

**Code Skeletons:**

* **Server**:

Initialize server, handle connections, process messages

* **Client**:

// Connect to server, send messages, handle responses

**Testing and Debugging:**

* Ensure the server handles multiple clients.
* Test adding, removing, and listing events.
* Verify correct handling of invalid actions with IncorrectActionException.