Assignment 2: Distributed Shared Whiteboard

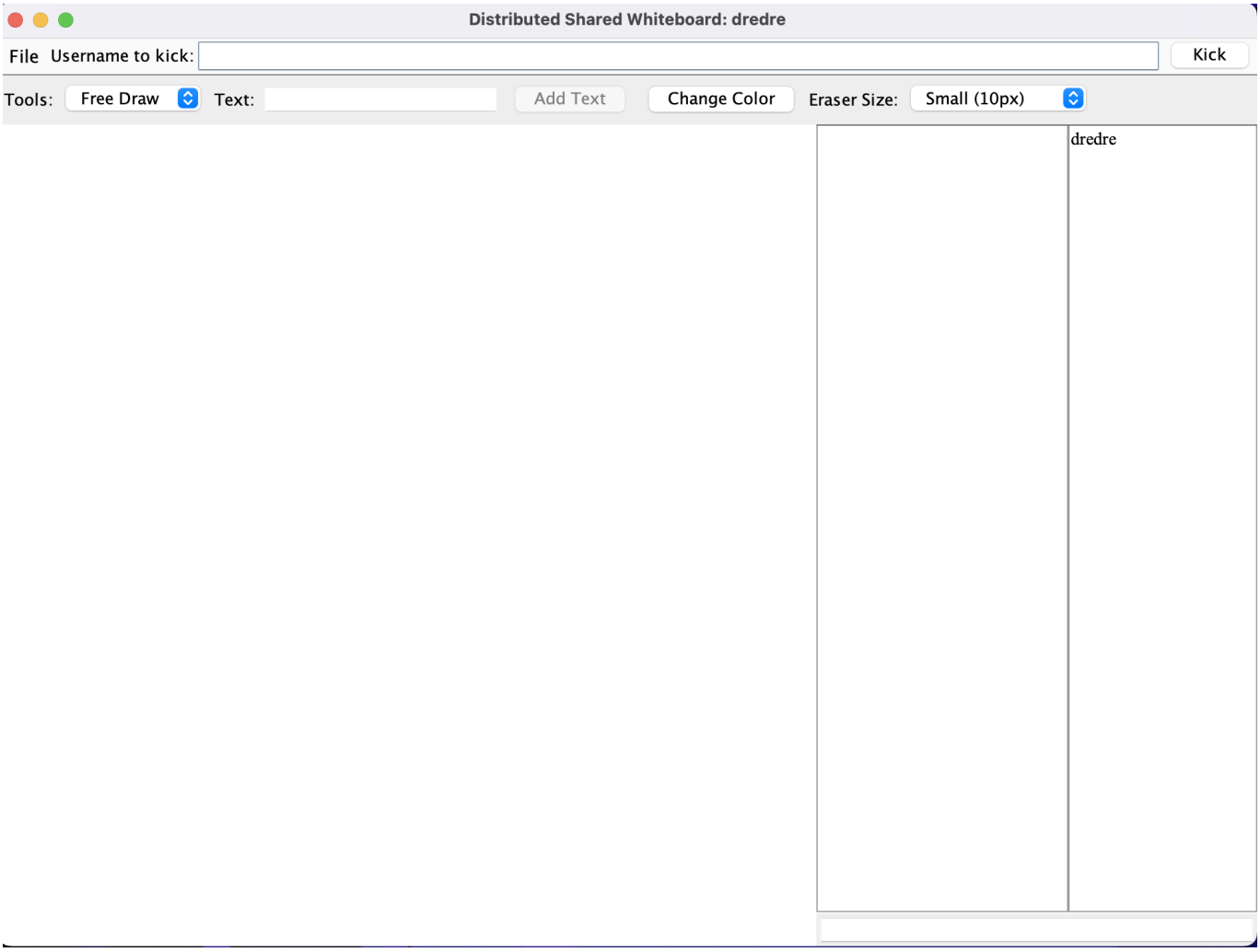
Student Name: **Zijun Zhang**

Student ID: **1160040**

1. System Architecture

Overview

The distributed shared whiteboard application enables multiple users to interact with a shared digital canvas in real-time through a client-server model. The server acts as the central coordinator, managing communications and synchronizing the state of the whiteboard across all connected clients.



Components

- **Client (WhiteboardClient.java)**:
    - Each user's machine runs the client component, which provides a graphical user interface for the whiteboard interaction. It captures user inputs, such as drawing commands and text entries, and sends these actions to the server as serialized objects.

    - The GUI is built using Java Swing, offering functionalities like drawing shapes, choosing colors, and typing text.

- **Server (WhiteboardServer.java)**:
    - The server component manages client connections, processes incoming commands, and maintains the whiteboard's state. It updates all clients in real-time with any changes to ensure consistency across the user views.
    - For each connected client, the server initiates a `ClientHandler` thread, which manages the communication with that specific client, allowing multiple users to interact with the system concurrently.

- **ClientHandler (ClientHandler.java)**:
    - This server-side component handles communication for each connected client. It processes received serialized commands, executes them, and broadcasts updates to other clients to synchronize the whiteboard view.
    - It handles various commands, including drawing actions, session management commands (e.g., join, leave), and board-clearing commands, ensuring actions are synchronized across clients.

- **Manager (WhiteboardManager.java)**:
    - The Manager is a specialized type of client with extended capabilities. In addition to the standard client functionalities, the Manager can create, open, save, and close the whiteboard. It also manages user sessions, including approving or rejecting new user connections and kicking out existing users by their names.
    - When a new client requests to join the whiteboard, they must receive approval from the Manager, who maintains overall control of the session. This role is crucial for managing the access and integrity of the collaborative environment.

- **CreateWhiteBoard (CreateWhiteBoard.java)**:
    - This class is the entry point for the whiteboard manager, responsible for initializing the `WhiteboardManager` and the `WhiteboardServer`.

- **JoinWhiteBoard (JoinWhiteBoard.java)**:
    - This class is used by regular clients to initiate the `WhiteboardClient`. It sets up the connection to the `WhiteboardServer` and integrates with the server's session management, waiting for approval from the `WhiteboardManager` before it can actively participate in the whiteboard session.

- **DrawingCanvas (DrawingCanvas.java)**:
    - This component handles the graphical rendering of all drawing actions on the client side. It uses various subclasses of the abstract `Shape` class to draw different shapes such as lines, rectangles, ovals, circles, freehand drawings, and text. The `Eraser` is another tool managed by `DrawingCanvas` to modify or remove existing drawings.

- **Shape Classes**:

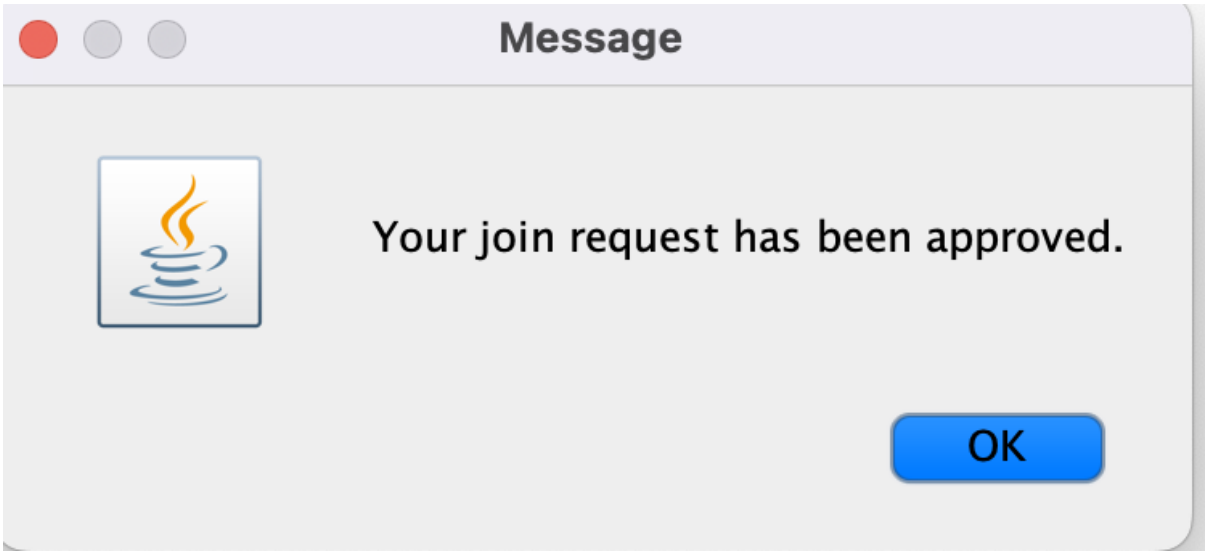
- `Shape` is an abstract class that serves as a base for specific shape classes like `Line`, `Rectangle`, `Oval`, `Circle`, `FreeDraw`, `Text`, and `Eraser`. Each class implements specific methods for rendering itself on the canvas, and instances of these classes are created and manipulated by the `DrawingCanvas` as users interact with the whiteboard.

```
abstract class Shape implements Serializable {
protected int startX, startY, endX, endY;
protected Color color;

public Shape(int startX, int startY, int endX, int endY, Color
color) {
    this.startX = startX;
    this.startY = startY;
    this.endX = endX;
    this.endY = endY;
    this.color = color;
}

public abstract void draw(Graphics g);
}
```

- **Command Classes (ClearCommand.java, etc.)**:
  - A series of command classes including `ClearCommand`, `OpenCommand`, `ServerQuitCommand`, and `UsernameTakenCommand` facilitate communication between the server and clients. These commands are used to handle various operational actions like clearing the board, opening a saved session, handling server shutdowns, and managing username conflicts.
- **JoinRequest and JoinResponse**:
  - These classes manage the initial interaction between a new client and the `WhiteboardManager`. A `JoinRequest` is sent by a new client to request joining the whiteboard session, and a `JoinResponse` is issued by the `WhiteboardManager` to accept or deny the request based on current session policies or user management decisions.



2. Communication Protocols and Message Formats

Choice of Technology

- The application uses TCP/IP sockets for network communication, ensuring reliable data transfer between the client and server. This choice supports the need for real-time synchronization and consistency across multiple clients.

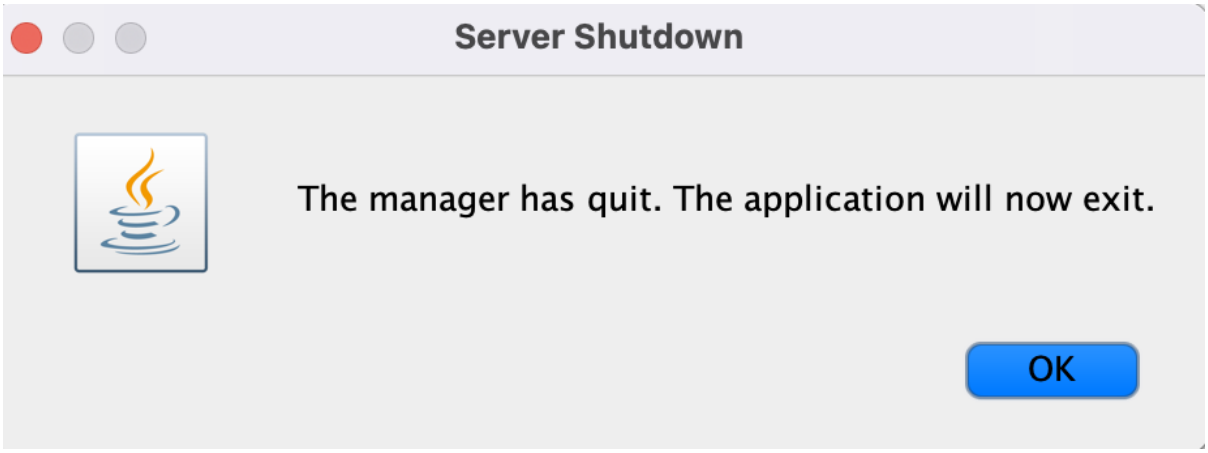
Protocols Used

- **TCP**: Given the interactive nature of the application and the necessity for reliable communication, TCP is used to ensure that all messages reach their destination intact and in order.

Message Types

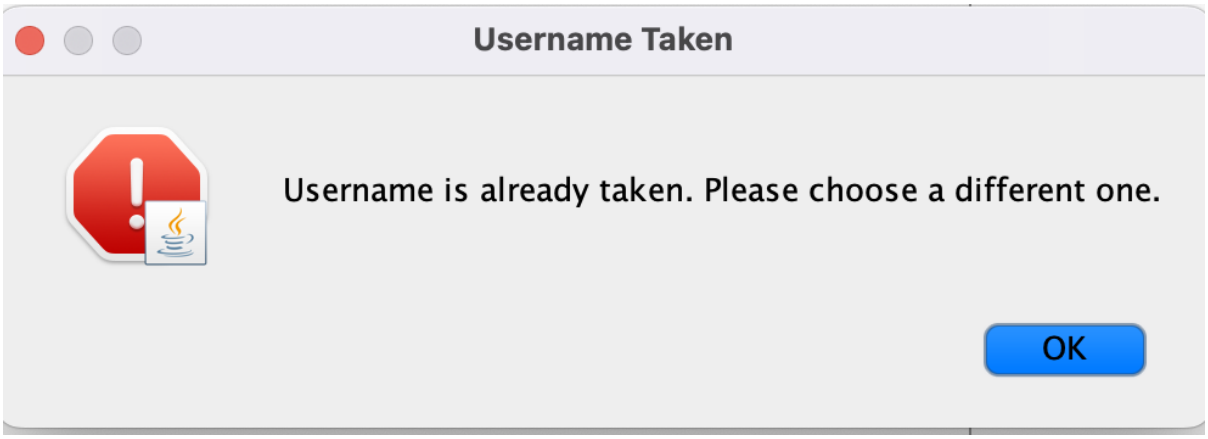
The application uses a variety of specialized message and command classes to handle different functionalities and interactions between the server and clients. These classes facilitate the communication of state changes, operational commands, and session management actions.

- **Drawing and State Management Commands**:
    - **ClearCommand**: Instructs the server and all clients to clear the whiteboard, resetting the drawing canvas to an empty state.
    - **OpenCommand**: Used by the manager to open a previously saved whiteboard session, which the server then broadcasts to all connected clients to ensure everyone views the same loaded state.
    - **ServerQuitCommand**: Sent by the manager to gracefully shut down the server and notify all clients to terminate their sessions, ensuring a coordinated closure of the application without data loss.



    - **UsernameTakenCommand**: Issued by the server when a new user attempts to join with a username that is already in use, prompting them to choose a different username to maintain

unique identifiers for each participant.



- **Session Management Commands**:
    - **JoinRequest**: Sent by a new client attempting to connect to the server, encapsulating information such as the requested username and possibly other credentials or session-specific data.
    - **JoinResponse**: A response from the manager or the server to the new client, indicating whether the join request has been approved or denied. This response ensures controlled access to the whiteboard session based on the manager's discretion or existing session policies.

These command classes are serialized into data packets before being sent over the network and are deserialized upon receipt by the receiving client or server. This serialization process ensures that complex command data is transmitted efficiently and reconstructed accurately at the destination, maintaining the integrity and functionality of the application's operations.

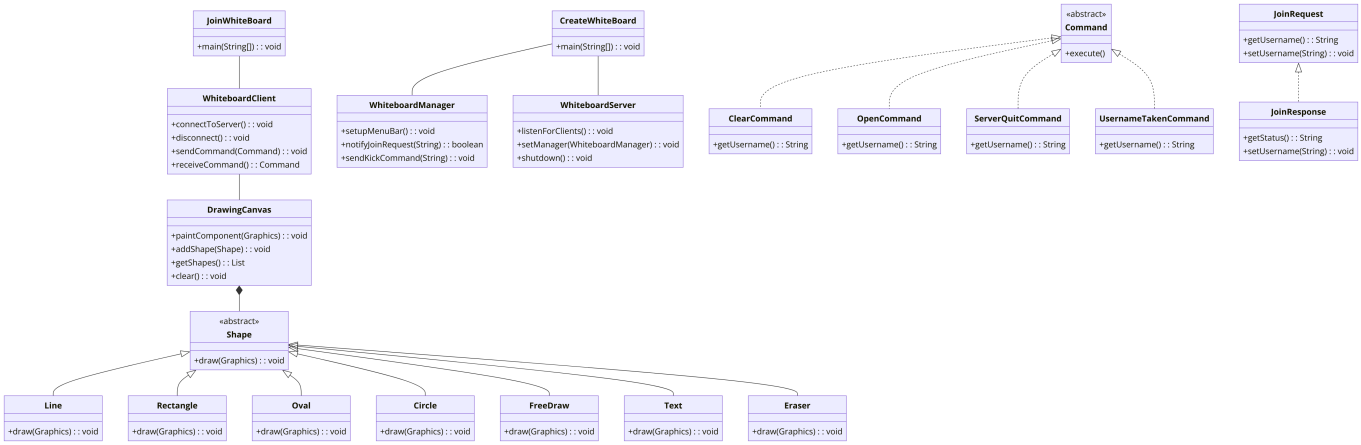
3. Design Diagrams

3.1 Class Diagram

The class diagram provides a visual representation of the application's structure, highlighting the relationship between various classes and their functionalities. This diagram is essential for understanding the object-oriented design of the system.

- **Class Overview**:
    - **CreateWhiteBoard**: Initializes the application by creating instances of `WhiteboardManager` and `WhiteboardServer`.
    - **JoinWhiteBoard**: Responsible for setting up a `WhiteboardClient` and facilitating its connection to the `WhiteboardServer`.
    - **WhiteboardClient**: Manages the client-side functionality, including sending and receiving commands, and interfacing with the `DrawingCanvas` for all drawing operations.
    - **WhiteboardManager**: Inherits from `WhiteboardClient` and adds administrative functions such as session management, opening and saving whiteboards, and managing user permissions.
    - **WhiteboardServer**: Listens for client connections and coordinates the communication between clients, ensuring the whiteboard's state is consistent across sessions.

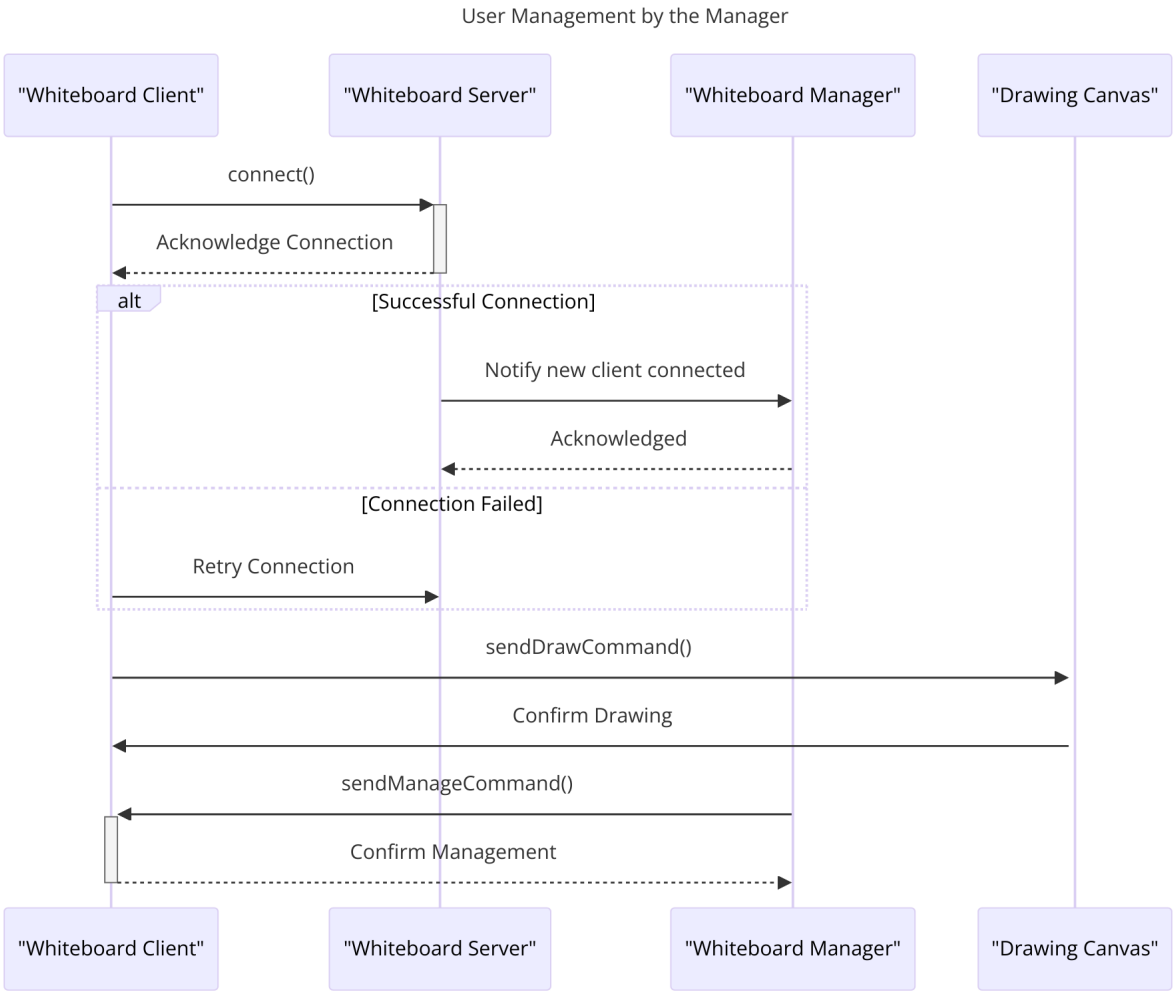
- **DrawingCanvas**: Handles all graphical operations on the client side, utilizing various shapes like `Line`, `Rectangle`, `Oval`, etc., which are derived from the abstract `Shape` class.
- **Command Classes** (e.g., `ClearCommand`, `OpenCommand`, etc.): These classes handle specific commands related to whiteboard management and client interactions.



3.2 Interaction Diagram

The interaction diagram illustrates how components of the system interact with each other to manage user connections and handle drawing commands. It is vital for understanding the flow of communication and the sequence of operations that occur during typical use cases.

- **Interaction Flow**:
  - The process begins with the `WhiteboardClient` connecting to the `WhiteboardServer`.
  - Upon a successful connection, the server acknowledges the client. If the connection fails, the client attempts to reconnect.
  - Once connected, the client can send drawing commands (`sendDrawCommand()`) which the server confirms, or management commands (`sendManageCommand()`) if the client is a manager.
  - The `WhiteboardManager` plays a crucial role in approving new client connections and managing existing ones. When a new client attempts to join, the manager is notified, and the new client is either accepted or rejected based on the manager's decision.
  - Drawing actions are processed by the `DrawingCanvas`, which updates the graphical display based on the commands received from the server.



4. Implementation Details

4.1 Client-Side Implementation

- **WhiteboardClient**:
  - The `WhiteboardClient` is responsible for handling the user interface and user interactions with the whiteboard. It includes methods to connect and disconnect from the server, send drawing commands, and receive updates.
  - **GUI Implementation**: The client GUI is built using Java Swing, providing a robust platform for rendering the canvas and other interface elements. It allows users to select drawing tools, choose colors, and manipulate shapes.
  - **DrawingCanvas**: This component extends `JPanel` and is used to handle drawing operations. It captures mouse events for drawing and uses the `Graphics` class to render shapes onto the panel. Shapes drawn by the user are stored in a list which can be updated or cleared.
- **Command Handling**:
  - Commands are sent to the server as serialized objects. The `sendCommand(Command)` method serializes the command objects and sends them through the socket connection to the server.
  - Upon receiving a command from the server, the `receiveCommand()` method deserializes the command object and determines the appropriate action to execute, such as updating the canvas or altering the user list.

4.2 Server-Side Implementation

- **WhiteboardServer**:
  - The server acts as the central hub for all client communications. It listens for incoming connections and spawns a new `ClientHandler` for each client.
  - **ClientHandler Threads**: Each client connection is managed by a dedicated thread, which handles all communications with that client, including receiving commands and broadcasting updates to other clients.
  - **Concurrency Management**: Synchronization mechanisms are employed to manage access to shared resources like the current state of the whiteboard. This ensures that all clients see a consistent view of the whiteboard.
- **Session Management**:
  - The server maintains a list of all active sessions and connected clients. It handles requests from new clients to join the session and communicates with the `WhiteboardManager` to get approval for each joining request.
  - Commands related to session management, such as opening a new whiteboard or saving the current state, are processed exclusively by the `WhiteboardManager`.

4.3 Communication Protocol

- **Message Serialization**:
  - Java's built-in serialization capabilities are utilized to encode and decode the command objects sent over the network. This approach simplifies the process of sending complex data structures across the network.
  - **Error Handling**: Robust error handling mechanisms are implemented to manage potential issues in network communication, such as lost connections or corrupted data packets.

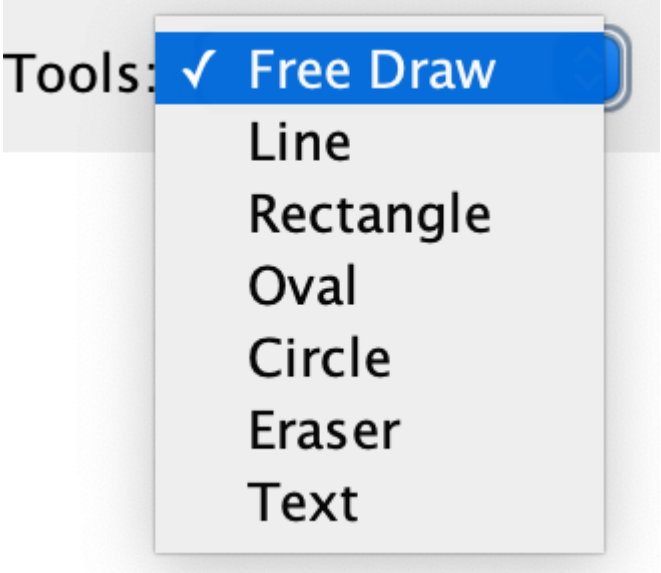
4.4 Manager-Specific Implementations

- **WhiteboardManager**:
  - Inherits all functionalities of `WhiteboardClient` and adds several administrative features. It can issue commands to open or save a whiteboard and has the authority to approve or reject client join requests.
  - **Kickout Functionality**: The manager can forcibly disconnect a client by sending a `KickCommand` to the server, which then closes the respective client's connection.
  - ```
    public void sendKickCommand(String username) {
        try {
            output.writeObject(new KickCommand(username));
            output.reset();
        } catch (IOException e) {
            System.err.println("Failed to send kick command: " +
    e.getMessage());
        }
    }
    ```
    ```

5. Innovations and Enhancements

5.1 Enhanced Drawing Features

- **Advanced Shape Dynamics**:
  - The application introduces an innovative approach to shape manipulation, enabling users to dynamically create, resize, and manipulate a variety of shapes including lines, rectangles, ovals, circles, and freehand drawings. Each shape is derived from an abstract `Shape` class, which standardizes the behavior of drawing operations and simplifies the addition of new shapes.



- **Interactive Text and Eraser Tools**:
  - Users can add text annotations directly onto the whiteboard, which is crucial for collaborative discussions and notes. The eraser tool has been enhanced to support multiple sizes, allowing for more precise control over content removal, catering to different user needs.

5.2 Real-Time Collaboration Enhancements

- **Immediate Update Broadcasting**:
  - A key innovation is the system's ability to broadcast updates to all connected clients instantaneously. This feature ensures that all users see the same changes in real-time, mirroring the collaborative experience of physically shared whiteboards.
- **Synchronous Drawing Sessions**:
  - The application supports synchronous drawing sessions where users can see others' drawing actions live. This feature not only enhances the collaborative experience but also helps in synchronizing team efforts on complex drawings or plans.

6. User Interaction and Management

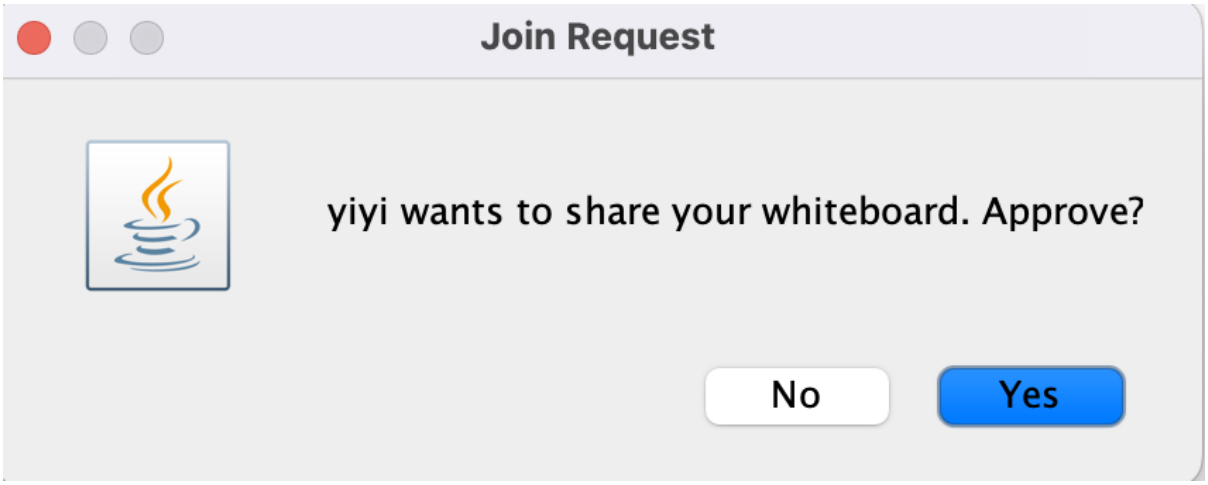
6.1 Operational Model

- **Session Initialization**:

  - Users initiate a session by running the `CreateWhiteBoard` or `JoinWhiteBoard` class, depending on whether they are starting a new whiteboard or joining an existing one. The manager starts the session by executing `CreateWhiteBoard`, which sets up the whiteboard and allows them to manage user connections and whiteboard sessions.

- **User Management**:

  - The `WhiteboardManager` has exclusive control over critical operations such as creating, saving, and closing the whiteboard. It also manages user permissions, approving or rejecting join requests from new users to ensure optimal collaboration and security.



6.2 User Experience

- **Interactive GUI**:

  - The GUI is designed to be user-friendly and intuitive, allowing users to easily select tools, change colors, and interact with the whiteboard. Tooltips and contextual help are provided to assist new users.

- **Real-Time Interaction**:

  - All users see updates made to the whiteboard in real-time. Changes made by one user are immediately visible to all others, mimicking the interaction of a physical whiteboard but with enhanced features.

- **Administrative Control**:

  - The manager can kick out users if necessary, which is crucial for maintaining order during collaborative sessions. This feature is implemented through the `sendKickCommand` method in the `WhiteboardManager`.

7. Conclusion

7.1 Summary of Achievements

- The project successfully implemented a multi-user distributed shared whiteboard application with features supporting real-time collaboration. Key technological choices include the use of Java Swing for the GUI, TCP/IP for network communication, and advanced concurrency management for handling multiple user interactions simultaneously.

7.2 Future Work

- **Enhancements**:
  - Future versions could include voice chat capabilities to enhance collaboration. Additionally, implementing a more robust security framework to encrypt all communication and adding support for more complex shapes and group layers would significantly benefit user experience.
- **Scalability**:
  - Scaling the server to support more simultaneous users and introducing more efficient data synchronization methods, like operational transformation, could be explored to enhance performance.