NetSketch

Valentyn Tokariuk

1. System Design

1.1 Architecture Overview

NetSketch is a client-server application designed to facilitate a collaborative whiteboard experience, enabling multiple users to interact on a shared canvas in real-time. The system architecture is split into two main components: the server and the clients. The server's role is to handle real-time data synchronization and manage network communication efficiently, while the clients are responsible for providing an intuitive user interface for drawing, chatting, and interacting with the canvas.

1.2 Components and Interactions

1. **Server**: The server component is central to the application, managing client connections, broadcasting updates, and maintaining the state of the shared canvas. Key responsibilities include:
    - Accepting client connections.
    - Managing drawing sequences.
    - Handling chat messages.
    - Synchronizing the background color across clients.
2. **Client**: Each client connects to the server to interact with the shared whiteboard. The client components are designed to be user-friendly, providing features for:
    - Drawing and modifying the canvas.
    - Sending and receiving chat messages.
    - Changing the background color of the canvas.
    - Undoing and redoing drawing actions.

1.3 Interaction Flow

The interaction flow between clients and the server can be summarized in the following steps:

1. **Client Login**: A user starts the client application, enters their username, and connects to the server. The server acknowledges the connection and sends the current state of the canvas.
2. **Drawing**: As users draw on their canvas, their actions are sent to the server. The server processes these actions and broadcasts them to all connected clients to ensure synchronization.
3. **Chatting**: Users can send chat messages to each other. Messages are sent to the server, which then broadcasts them to all clients.
4. **Undo/Redo**: Clients can undo or redo their drawing actions. These commands are sent to the server, which manages the undo/redo stacks and updates all clients.

5. **Background Color Change**: Users can change the background color of the canvas. This change is sent to the server and propagated to all clients, ensuring a consistent look across all devices.

1.4 Class Diagram

The class diagram of the system includes the following key classes and their interactions:

- **Server**: Manages client connections, drawing sequences, chat messages, and background color.
- **ThreadClient**: A thread that handles communication with individual clients.
- **Data**: A versatile class that encapsulates various types of data exchanged between the server and clients, including sequences, messages, actions, and background color changes.
- **UInterface**: The main client user interface that integrates all functionalities.
- **DrawingPanel**: A component within the client that handles drawing operations.
- **Sequence**: Represents a sequence of drawing actions, including points and brush attributes.
- **Point**: Represents individual points on the canvas.

---

2. Special Mechanisms

2.1 Real-time Synchronization

A core feature of NetSketch is its ability to synchronize data in real-time across multiple clients. This is achieved through:

- **Broadcasting**: The server broadcasts updates (drawing actions, chat messages, and background color changes) to all connected clients.
- **State Management**: The server maintains the state of the canvas and ensures that all clients have a consistent view by sending the complete state to new clients upon connection.

2.2 Concurrency Management

Concurrency management is critical in a multi-client environment to prevent race conditions and ensure data consistency. NetSketch employs:

- **Multi-threading**: Each client connection is managed by a separate thread (`ThreadClient`), allowing the server to handle multiple clients simultaneously without performance degradation.
- **Synchronized Access**: Critical sections of code, particularly those that modify shared resources (e.g., drawing sequences, chat messages), are protected to prevent concurrent modification issues.

---

3. Testing Approach

3.1 Test Cases

To ensure the reliability and robustness of NetSketch, a comprehensive set of test cases was designed and executed:

1. **Client Connection**:
   - *Objective*: Ensure that clients can successfully connect to the server.
   - *Expected Result*: Clients should connect without issues and receive the current state of the canvas.
   - *Result*: Passed.
2. **Drawing Synchronization**:
   - *Objective*: Verify that drawing actions by one client are accurately reflected on all other clients.
   - *Expected Result*: Drawings should appear in real-time across all connected clients.
   - *Result*: Passed.
3. **Chat Functionality**:
   - *Objective*: Ensure that chat messages are broadcasted to all clients.
   - *Expected Result*: Messages sent by one client should appear in the chat area of all other clients.
   - *Result*: Passed.
4. **Undo/Redo Functionality**:
   - *Objective*: Test the undo and redo operations.
   - *Expected Result*: Drawing actions should be undoable and redoable accurately.
   - *Result*: Passed.
5. **Background Color Change**:
   - *Objective*: Ensure that background color changes are propagated to all clients.
   - *Expected Result*: Changing the background color on one client should update the background color on all clients.
   - *Result*: Passed.

3.2 Results

All test cases were executed successfully, confirming the correct functionality and synchronization of the system. The application demonstrated stability and responsiveness under various scenarios.

---

4. Bugs and Issues

4.1 Encountered Problems

During development and testing, several issues were encountered and resolved:

1. **Ambiguity in Data Class**:

- *Description*: The `Data` class had multiple constructors, causing ambiguity errors during object creation.
- *Error Code*: `reference to Data is ambiguous`
- *Solution*: Added specific type checks and ensured unique constructors to resolve the ambiguity.

2. **Color Class Import**:
   - *Description*: The `Color` class from `java.awt` was not initially imported, leading to compilation errors.
   - *Error Code*: `cannot find symbol: class Color`
   - *Solution*: Imported `java.awt.Color` in the relevant classes.
3. **Client List Access**:
   - *Description*: The `getClients` method was missing in the `Server` class, causing issues when accessing the list of connected clients.
   - *Error Code*: `cannot find symbol: method getClients()`
   - *Solution*: Implemented the `getClients` method to return the list of client streams.
4. **Chat Update Method**:
   - *Description*: The `updateClientsChat` method was missing, leading to errors when attempting to broadcast chat messages.
   - *Error Code*: `cannot find symbol: method updateClientsChat()`
   - *Solution*: Implemented the `updateClientsChat` method to broadcast chat messages to all clients.
5. **Ambiguity in Constructor Calls**:
   - *Description*: Ambiguity between different constructor calls of the `Data` class when sending data.
   - *Error Codes*:
     - `Data.java:20: error: cannot find symbol`
     - `Data.java:53: error: cannot find symbol`
     - `Data.java:83: error: cannot find symbol`
   - *Solution*: Ensured that constructors were uniquely distinguishable and added more specific type handling.
6. **Synchronization Issues**:
   - *Description*: Initial synchronization delays were observed due to high network latency.
   - *Solution*: Optimized the data broadcasting mechanism to reduce latency and improve real-time performance.
7. **NullPointerException in Client**:
   - *Description*: Clients occasionally encountered a `NullPointerException` when trying to access the drawing sequences.
   - *Error Code*: `Exception in thread "main" java.lang.NullPointerException`
   - *Solution*: Added null checks and ensured that the server properly initializes the sequence list before sending it to clients.
8. **Socket Timeout**:
   - *Description*: Clients sometimes experienced socket timeouts when trying to connect to the server.
   - *Error Code*: `java.net.SocketTimeoutException: connect timed out`

    - *Solution*: Adjusted the server and client socket timeout settings to be more forgiving under varying network conditions.
9. **Inconsistent UI Updates**:
    - *Description*: The user interface sometimes failed to update consistently, especially under high load.
    - *Error Code*: No specific error code, observed as UI lag.
    - *Solution*: Refactored the UI update mechanism to ensure updates are processed on the Event Dispatch Thread (EDT) using `SwingUtilities.invokeLater`.

4.2 Potential Errors

Given the complexity of the system, several potential errors could arise, including:

1. **Connection Timeouts**:
    - *Description*: Clients might experience connection timeouts due to network instability or latency.
    - *Solution*: Implemented retry mechanisms and timeout handling to ensure robust connections.
2. **Data Loss**:
    - *Description*: Network interruptions could lead to data loss or inconsistency.
    - *Solution*: Implemented acknowledgment mechanisms to ensure data integrity.
3. **Concurrency Issues**:
    - *Description*: Without proper concurrency control, race conditions could lead to inconsistent states.
    - *Solution*: Used synchronized methods and thread-safe data structures to manage concurrent access.
4. **UI Performance**:
    - *Description*: The graphical user interface might become unresponsive under heavy load.
    - *Solution*: Optimized UI rendering and ensured that long-running operations were performed on background threads.

Conclusion

NetSketch successfully implements a collaborative whiteboard application, demonstrating advanced network programming and concurrency management principles. Despite initial challenges, the system achieved real-time synchronization, seamless user interaction, and robust performance. Future improvements could focus on enhancing the user interface, optimizing network performance, and adding more collaborative features to enrich the user experience. The development and testing process highlighted the importance of thorough testing and the need for robust error handling in distributed systems.