Distributed Shared Whiteboard Report

YuFan Zhang(Jeff)

Student ID:1196642

COMP90015 Distributed System

16/5/2024

Introduction

This project builds a distributed shared whiteboard system that allows multiple users to collaborate and edit content together on a shared canvas over a network. The system has two main user roles: the ***manager*** and ***normal users***.

Normal users can perform various drawing operations like creating shapes (***lines, circles, ovals, rectangles***), ***typing text anywhere*** on the canvas, ***choosing from 16 different colors***, and using an ***eraser tool to modify existing drawings***. Users ***can also communicate*** with each other through a text-based chat window in the application.

In addition to having all the drawing abilities of normal users, the manager takes a supervisory role. The manager can ***create a new whiteboard session***, ***open*** existing whiteboard files, ***save*** the current state, and ***close*** the entire application. The manager also has the authority to ***approve new user join*** requests and ***forcibly disconnect*** ("kick out") any user from the shared session.

The system utilizes ***Java Remote Method Invocation*** (RMI) for real-time collaboration and drawing synchronization across users. It ***employs Concurrent HashMap***, a thread-safe data structure, to handle concurrent access and manage the shared whiteboard state. RMI enables efficient communication between the server and clients, ensuring a seamless shared experience. This design addresses distributed state management and implements an effective protocol for real-time collaboration.

How to start

java -jar WhiteBoardServer.jar <port number>

e.g. java -jar WhiteBoardServer.jar 1234

java -jar WhiteBoardClient.jar <host address> <port number>

e.g. java -jar WhiteBoardClient.jar localhost 1234

System Architecture

1. Communication protocols

The system employs Java ***Remote Method Invocation*** (RMI) as the communication protocol, enabling real-time multi-user collaboration and drawing synchronization. Compared to socket-based applications, ***RMI greatly simplifies*** the development of distributed systems, allowing clients to seamlessly invoke methods on remote server objects, ***just like calling local methods.*** The ***server*** is ***responsible for maintaining the latest state of the whiteboard and broadcasting updates in real-time***. The ***server's Manager component uses the thread-safe ConcurrentHashMap to manage the list of connected clients***, improving concurrent performance. When a ***client starts***, it ***sends a registration request*** to ***the server via RMI***. If ***the client list is empty***, this ***client*** will ***be registered as the manager*** and ***open the whiteboard interface***. When a ***new user joins***, the server ***first attempts to register the user and notifies the manager to approve the join request***. The ***manager can choose to kick out a user***, and the ***server will correspondingly call the kick-out method on the respective client***. For the

manager's save, save as, and open operations, the server and clients ***exchange image data using byte arrays and PNG format conversions***, ***initiated by GUI events detected on the client side***.

2.Message formats

For the drawing message format, the system uses a structure the provided image “drawing Message” and example2. Whenever a drawing operation is performed, a message containing the following information is sent:

```
public Draw(String state, String name, String mode, Color color, Point pt, String text, int penSize, int eraserSize) throws RemoteException {
    this.state = state;
    this.clientName = name;
    this.mode = mode;
    this.color = color;
    this.point = pt;
    this.text = text;
    this.penSize = penSize;
    this.eraserSize = eraserSize;
}
```

Drawing Message

File format:

The system employs an efficient approach for transferring image data between the server and clients, leveraging the interconversion between byte arrays and PNG format. This approach is illustrated in below figures, where the text content is specified in the PNG format during the save operation.

```
// Get current message
public byte[] getCurrentImage() throws RemoteException, IOException;

// Administrator opens a new drawing board file
public void sendOpenedImage(byte[] rawImage) throws IOException;

// Open white board
public void openWhiteboard(byte[] imageData) throws RemoteException;
```

File format example

Chat Message：

The message format is implemented as a string. Additionally, the system adopts a serverless architecture design, where the server does not store or maintain chat records. When users leave, the chat records will be lost, and newly joined users cannot view previous records.

3.Class details:

The process ***begins*** with ***parameter handlin***g; the ***server*** requires a ***port number*** in order to start. If the provided parameters are insufficient or incorrect, the server will display usage instructions. Once ***successfully started***, ***it waits for client connections***.
For each new ***client***, they need to ***enter the server address and port number***, ***followed by inputting a valid name***. The client then ***sends a registration request to the server***. If it is the ***first user***, that user will be ***registered as the manager***, responsible for ***managing files and users***. Upon successful registration, the whiteboard will open. If ***not the first user***, the server

will ***send a request to the manager***, ***asking for approval to join***. Once a user has joined, they can participate in the online shared drawing.
On the server side, a ***ConcurrentHashMap is used to maintain each client***, allowing ***concurrent read and write operations*** without causing data inconsistency or thread safety issues.

Client Package

Canvas.java

Implements the client-side Canvas component of the shared whiteboard, comprising the following main functionalities:

- Client startup and registration
  - Entry point in the main method
  - User inputs server IP, port, and a valid username for registration
  - The first registered user becomes the manager
- Client interface functionality
  - Includes canvas, toolbar, user list, and chat window
  - Canvas class implements drawing capabilities
  - Manager has a "File" menu for file operations
- Server interaction
  - Communicates with the server via RMI
  - Receives and sends drawing instructions, updates user list, sends chat messages
  - Implements the CanvasClientInterface interface

Client.java

Implements the client-side logic of the shared whiteboard system, mainly covering the following aspects:

- Client startup and registration
  - Main entry point in the main method
  - User inputs server IP, port, and a valid username for registration
  - Determines the first registered user as the manager
- Client interface
  - Includes canvas area, toolbar, user list, and chat window
  - Canvas class implements drawing functionalities
  - Manager can perform file operations like create, open, save
- Server interaction
  - Communicates with the server via RMI
  - Receives and sends drawing instructions, updates user list, sends chat messages
  - Implements the CanvasClientInterface interface
- File operations
  - Manager can create, save/save as, and open whiteboard files

- Image data is transmitted as byte arrays

Draw.java

Implements the CanvasMsgInterface and is used to encapsulate and transmit drawing messages. It serves as a data carrier for drawing instructions between the client and the server, containing all the necessary information to execute specific drawing operations. The server can accurately recreate the corresponding drawing process on the whiteboard based on the contents of this message.

Remote Package

CanvasClientInterface.java

Is a remote interface that defines the communication protocol between the client and the server. It enables the client to respond to the server's instructions, update the local state, and interact with the user. It is a critical interface for implementing the distributed shared whiteboard system.

CanvasMsgInterface.java

Defines the data fields that a drawing message should contain, such as the drawer, state, color, mode, text, coordinates, pen size, and eraser size. Based on the received drawing message, the server can accurately recreate the corresponding drawing effect on the whiteboard.

CanvasServerInterface.java

Defines various methods provided by the server for managing client connections, handling drawing operations, and transmitting data. It is the core for implementing the server-side logic of the distributed shared whiteboard system.

Server Package

CanvasServer.java

Is the server-side implementation of the shared whiteboard system, playing a core role in control and coordination. It extends UnicastRemoteObject and implements the CanvasServerInterface interface, providing the following main functionalities:

- User Management
- Client Management
- Drawing Information Broadcast
- Image Data Transmission

Manager.java

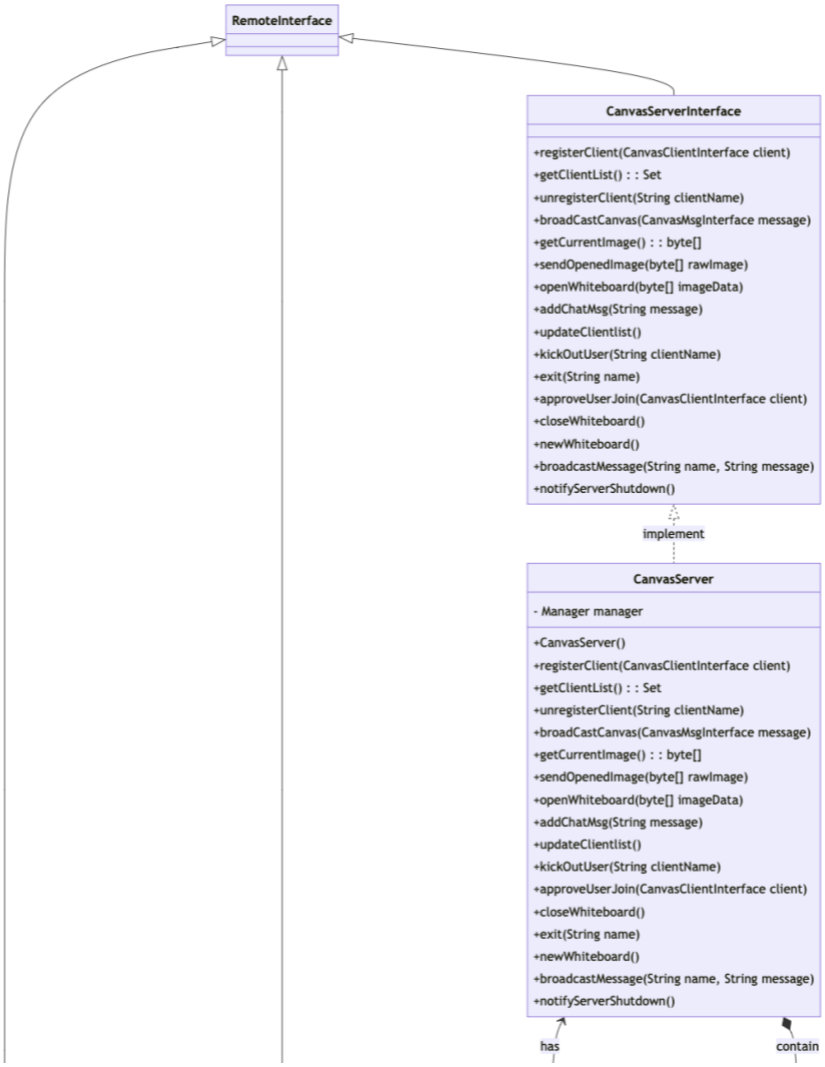
Is used to manage the clients connected to the shared whiteboard server. It creates a thread-safe Set collection called clientsList using ConcurrentHashMap to store the connected clients. It provides basic

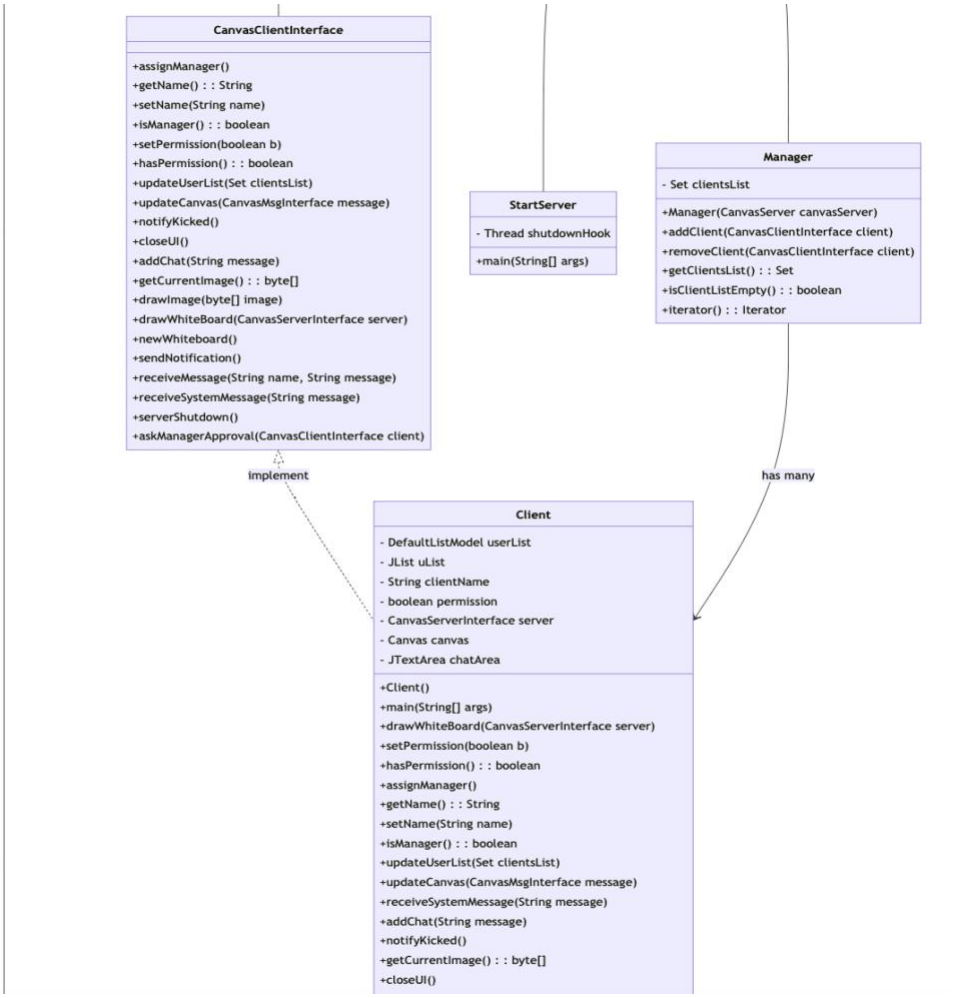
operations such as adding, removing, retrieving, and checking if the client list is empty.

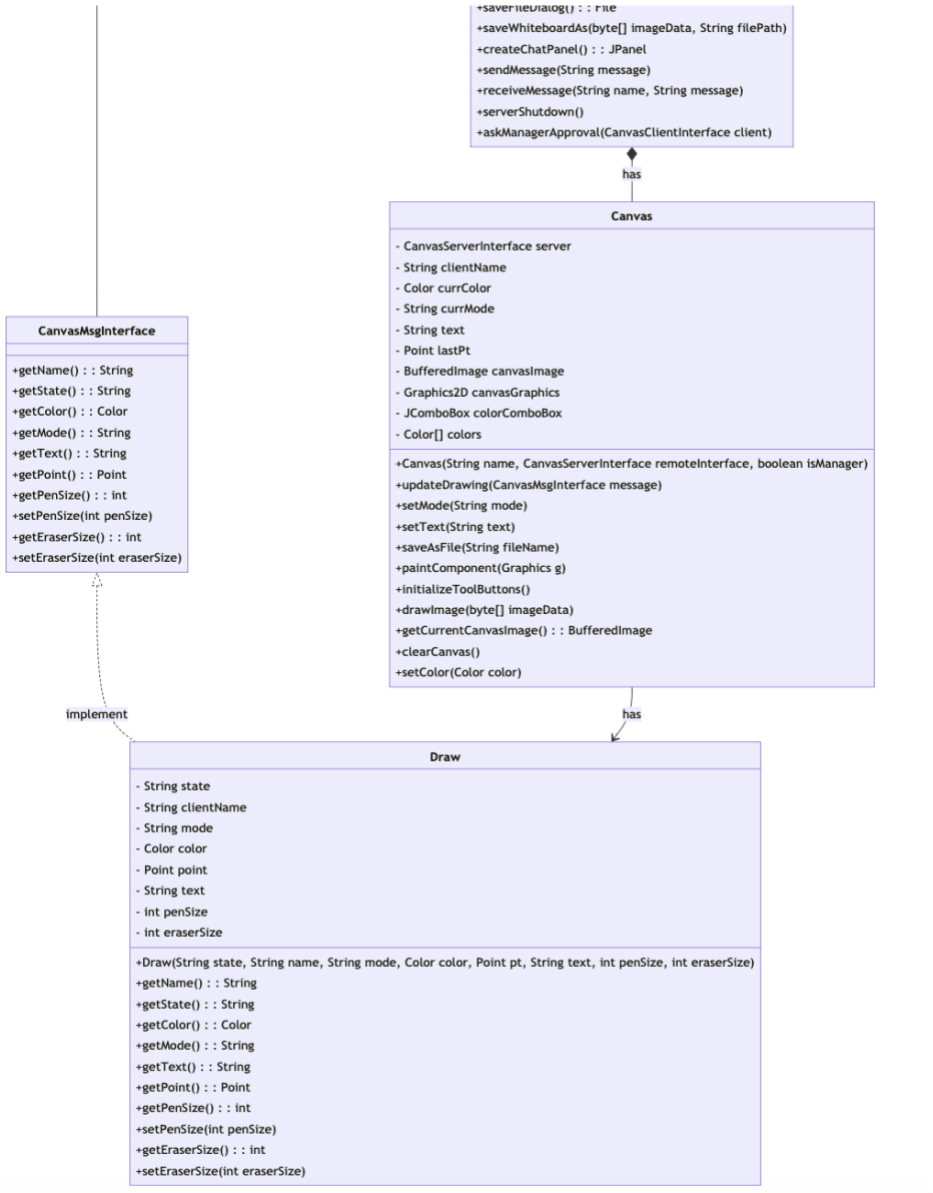
StartServer.java

Is the entry point for the server-side of the shared whiteboard system. Its main function is to create a CanvasServer instance, start the RMI registry, and bind the CanvasServer to the registry, allowing clients to discover and connect to the server. It also sets up the cleanup operation when the server shuts down to ensure that all clients receive the notification.

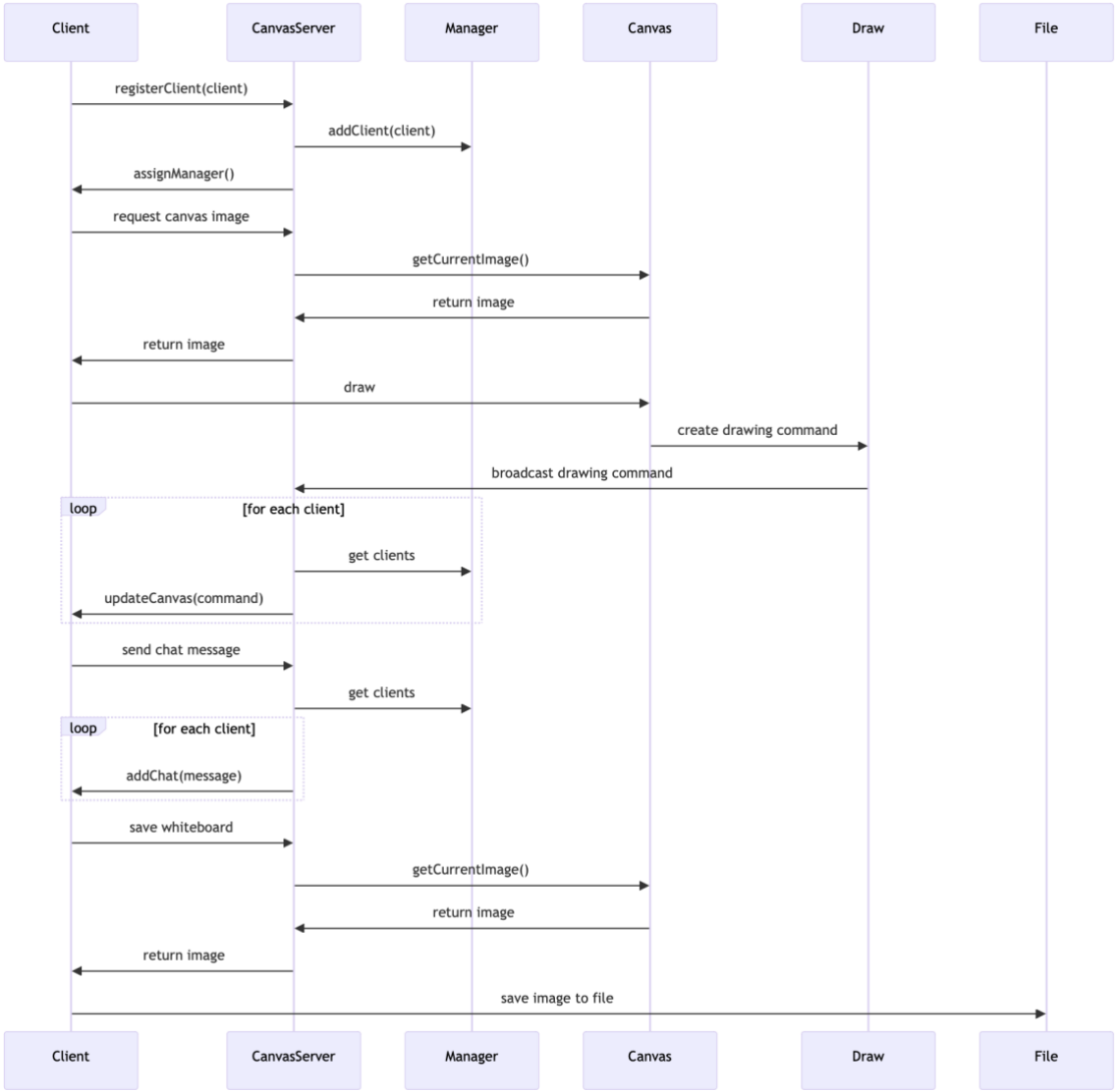
Overall design(diagrams)　（GPT 用plant UML去画）



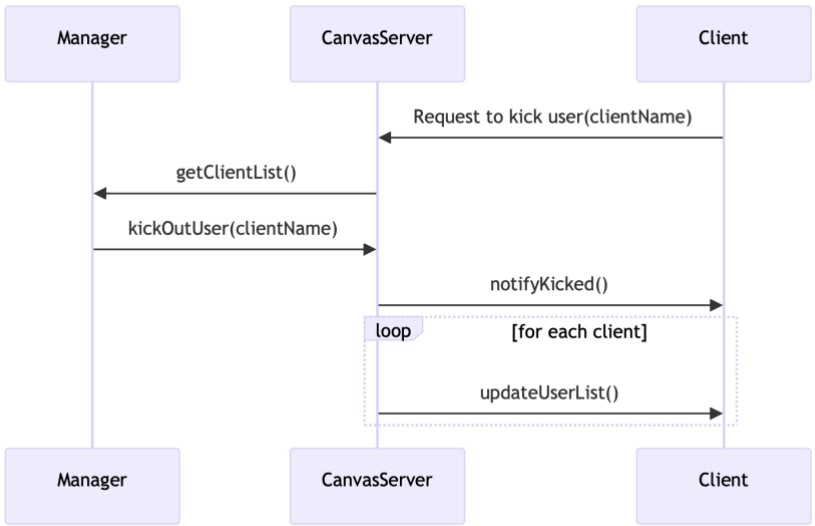

CanvasClientInterface
+assignManager()
+getName() : : String
+setName(String name)
+isManager() : : boolean
+setPermission(boolean b)
+hasPermission() : : boolean
+updateUserList(Set clientsList)
+updateCanvas(CanvasMsgInterface message)
+notifyKicked()
+closeUI()
+addChat(String message)
+getCurrentImage() : : byte[]
+drawImage(byte[] image)
+drawWhiteBoard(CanvasServerInterface server)
+newWhiteboard()
+sendNotification()
+receiveMessage(String name, String message)
+receiveSystemMessage(String message)
+serverShutdown()
+askManagerApproval(CanvasClientInterface client)
StartServer
- Thread shutdownHook
+main(String[] args)
Manager
- Set clientsList
+Manager(CanvasServer canvasServer)
+addClient(CanvasClientInterface client)
+removeClient(CanvasClientInterface client)
+getClientsList() : : Set
+isClientListEmpty() : : boolean
+iterator() : : Iterator
implement
has many
Client
- DefaultListModel userList
- JList uList
- String clientName
- boolean permission
- CanvasServerInterface server
- Canvas canvas
- JTextArea chatArea
+Client()
+main(String[] args)
+drawWhiteBoard(CanvasServerInterface server)
+setPermission(boolean b)
+hasPermission() : : boolean
+assignManager()
+getName() : : String
+setName(String name)
+isManager() : : boolean
+updateUserList(Set clientsList)
+updateCanvas(CanvasMsgInterface message)
+receiveSystemMessage(String message)
+addChat(String message)
+notifyKicked()
+getCurrentImage() : : byte[]
+closeUI()


+saveFileDialog() : : File
+saveWhiteboardAs(byte[] imageData, String filePath)
+createChatPanel() : : JPanel
+sendMessage(String message)
+receiveMessage(String name, String message)
+serverShutdown()
+askManagerApproval(CanvasClientInterface client)
has
Canvas
- CanvasServerInterface server
- String clientName
- Color currColor
- String currMode
- String text
- Point lastPt
- BufferedImage canvasImage
- Graphics2D canvasGraphics
- JComboBox colorComboBox
- Color[] colors
+Canvas(String name, CanvasServerInterface remoteInterface, boolean isManager)
+updateDrawing(CanvasMsgInterface message)
+setMode(String mode)
+setText(String text)
+saveAsFile(String fileName)
+paintComponent(Graphics g)
+initializeToolButtons()
+drawImage(byte[] imageData)
+getCurrentCanvasImage() : : BufferedImage
+clearCanvas()
+setColor(Color color)
CanvasMsgInterface
+getName() : : String
+getState() : : String
+getColor() : : Color
+getMode() : : String
+getText() : : String
+getPoint() : : Point
+getPenSize() : : int
+setPenSize(int penSize)
+getEraserSize() : : int
+setEraserSize(int eraserSize)
implement
has
Draw
- String state
- String clientName
- String mode
- Color color
- Point point
- String text
- int penSize
- int eraserSize
+Draw(String state, String name, String mode, Color color, Point pt, String text, int penSize, int eraserSize)
+getName() : : String
+getState() : : String
+getColor() : : Color
+getMode() : : String
+getText() : : String
+getPoint() : : Point
+getPenSize() : : int
+setPenSize(int penSize)
+getEraserSize() : : int
+setEraserSize(int eraserSize)

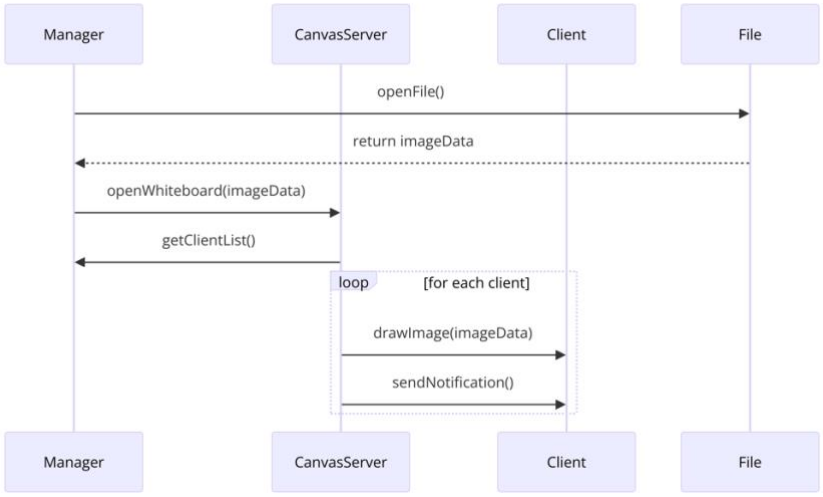
UML diagram



Sequence diagram of register, update Canvas and save image



Sequence diagram of kickout



Sequence diagram of manager open file

New innovations

- When creating a new whiteboard, there will be a prompt asking for confirmation to prevent accidental actions.
- When someone joins or leaves, the chat box will display system notifications.
- The chat box implements a serverless mechanism, which means previous chat records will not be saved.
- Users can choose the size of the drawing pen.
- Users can choose the size of the eraser.
- When the manager creates a new whiteboard, all users will receive a notification.
- When the manager opens an existing whiteboard, all users will receive a notification.
- If the server shuts down, all users will be notified before the application closes.
- Both the chat window and the user list window implement scrollable panels.

Conclusion

In conclusion, this report covers the aspects of system architecture, communication protocols, message formats, design diagrams, implementation details, and new innovations to elaborate on how the distributed shared whiteboard is implemented. This was an interesting assignment, and I learned a great deal through studying and researching RMI, as well as debugging RMI-related issues. Nevertheless, the process was enjoyable and led to significant learning.