ChatApp

Description:
This program implements a two-way network chat using JavaFX. It opens a window that can be used for a two-way network chat. The window can "listen" for a connection request on a port that is specified by the user. It can also request a connection to another ChatApp window on a specified computer and port. The window has an input box where the user can enter messages to be sent over the connection. A connection can be closed by clicking a button in the window or by closing the window.
Dependencies:

- JavaFX
- javafx.application.Application
- javafx.application.Platform
- javafx.stage.Stage
- javafx.scene.Scene
- javafx.stage.FileChooser
- javafx.scene.control.Button
- javafx.scene.control.Label
- javafx.scene.control.TextField
- javafx.scene.control.TextArea
- javafx.scene.control.Alert
- javafx.scene.layout.HBox
- javafx.scene.layout.VBox
- javafx.scene.layout.BorderPane
- javafx.event.ActionEvent
- javafx.geometry.Pos
- javafx.geometry.Insets
- java.io.*
- java.net.*

Classes:

1. ChatApp: The main class that extends javafx.application.Application and represents the GUI chat window. It contains methods to set up the GUI, handle events, and manage the network connection.
    - main(String[] args): The entry point of the program that launches the JavaFX application.
    - start(Stage stage): The method that sets up the GUI and event handling. It creates the UI components, configures their properties, and adds event listeners.
    - doAction(ActionEvent evt): The method that handles button clicks and other actions in the GUI. It determines the source of the event and performs the corresponding action.
    - doSave(): The method that saves the contents of the transcript area to a file selected by the user.
    - errorMessage(String message): A helper method to show an error alert with the specified message.

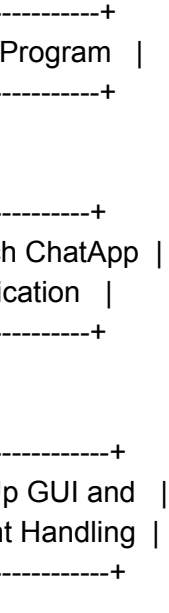
- postMessage(String message): A helper method to add a line of text to the transcript area.

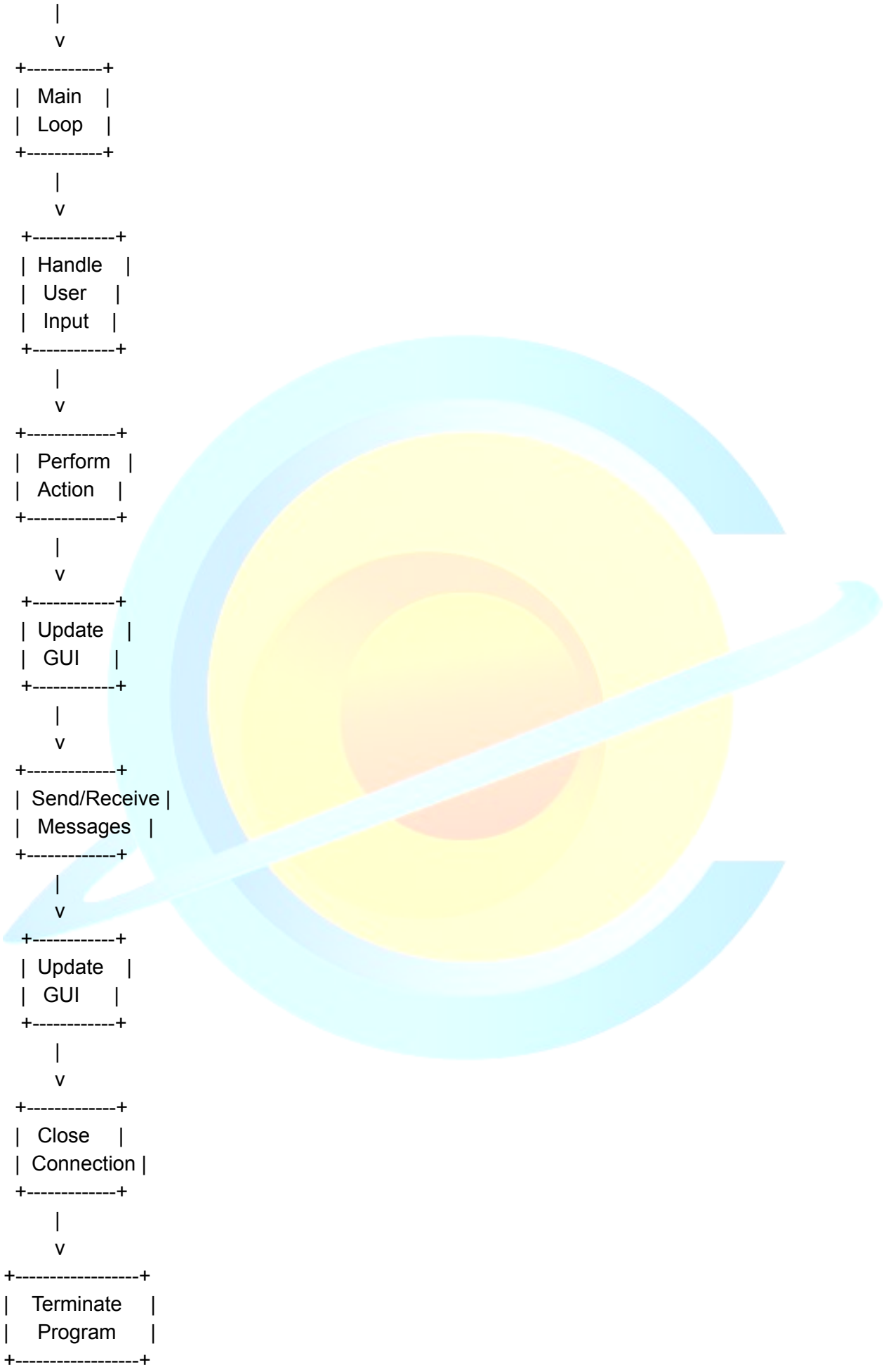
2. ConnectionHandler: A nested class that represents the thread that handles the network connection. It opens the connection, sends and receives messages, and manages the connection state.
    - ConnectionHandler(int port): The constructor for listening for a connection on the specified port.
    - ConnectionHandler(String remoteHost, int port): The constructor for opening a connection to the specified computer and port.
    - getConnectionState(): Returns the current state of the connection.
    - send(String message): Sends a message to the other side of the connection and posts the message to the transcript.
    - close(): Closes the connection.
    - received(String message): Called when a message is received from the other side of the connection. Posts the message to the transcript if the connection state is connected.
    - connectionOpened(): Called when the connection has been successfully opened. Enables the correct buttons and sets the connection state to connected.
    - connectionClosedFromOtherSide(): Called when the connection is closed from the other side. Posts a message to the transcript and sets the connection state to closed.
    - run(): The main method of the thread that handles the connection. It performs the necessary network operations and handles exceptions.

```
/**
 * Opens a window that can be used for a two-way network chat.
 * The window can "listen" for a connection request on a port
 * that is specified by the user. It can request a connection
 * to another ChatApp window on a specified computer and port.
 * The window has an input box where the user can enter
 * messages to be sent over the connection. A connection
 * can be closed by clicking a button in the window or by
 * closing the window. To test the program, several
 * copies of the program can be run on the same computer.
 */
public class ChatApp extends Application {
    ...
}

/**
 * Set up the GUI and event handling.
 */
public void start(Stage stage) {
    ...
}

/**
 * A little wrapper for showing an error alert.
 */
```

```
private void errorMessage(String message) {
    ...
}

/**
 * Defines responses to buttons.
 */
private void doAction(ActionEvent evt) {
    ...
}

/**
 * Save the contents of the transcript area to a file selected by the user.
 */
private void doSave() {
    ...
}

/**
 * Add a line of text to the transcript area.
 * @param message text to be added; a line feed is added at the end
 */
private void postMessage(String message) {
    ...
}

/**
 * Defines the thread that handles the connection.
 */
private class ConnectionHandler extends Thread {
    ...
}
```

Flowchart:




Main Loop
Handle User Input
Perform Action
Update GUI
Send/Receive Messages
Update GUI
Close Connection
Terminate Program

To test the above program, you can follow the steps below:

1. Compile the program: Compile the Java file containing the `ChatApp` class using the Java compiler. For example, if the file is named `ChatApp.java`, you can compile it by running the command `javac ChatApp.java` in the command prompt or terminal.

2. Run the program: After successful compilation, run the program by executing the command `java ChatApp` in the command prompt or terminal. This will start the GUI window for the network chat application.

3. Choose the desired options: In the GUI window, you have several options to choose from:
   - Listen on port: Enter a port number in the input box next to the "Listen on port:" button and click the button to listen for incoming connections on that port.
   - Connect to: Enter the remote host name and port number in the respective input boxes next to the "Connect to:" button and click the button to initiate a connection to the specified host and port.
   - Disconnect: Click the "Disconnect" button to close the current connection.
   - Clear Transcript: Click the "Clear Transcript" button to clear the message transcript area.
   - Save Transcript: Click the "Save Transcript" button to save the contents of the message transcript area to a file.
   - Quit: Click the "Quit" button to exit the application.

4. Interact with the chat: Depending on the options chosen, you can send and receive messages in the chat. If you have initiated a connection or received a connection request, you can enter messages in the "Your Message:" input box and click the "Send" button to send the message to the other side of the connection. Received messages will be displayed in the message transcript area.

5. Test different scenarios: You can test different scenarios by running multiple instances of the program on the same computer or different computers. For example, you can run one instance as a server (listening on a port) and another instance as a client (connecting to the server). You can then send messages between the server and client instances to test the communication.

6. Close the program: To close the program, click the "Quit" button or close the program window. If a connection exists, it will be closed automatically.

During testing, you can observe the messages in the transcript area to verify the communication and ensure that the program functions as expected.

Note: The program relies on JavaFX for the graphical user interface. Make sure you have JavaFX properly set up and configured to run the program.