Advanced Programming Report

Design Patterns

Design Pattern Name	Involved Classes/Methods	Justification
Observer Pattern	Client.java ChatGUI.java client.setOnMessageReceived(...) client.setOnMemberListUpdated(...)	• This pattern is used to enable **asynchronous event-driven updates** in the chat application. • The GUI listens for changes (new messages, user list updates, disconnections) without directly calling client logic. • This ensures **loose coupling**, **efficient updates**, and **scalability**—new events can be handled without modifying existing components.
Factory Pattern	MessageFactory.java MessageFactory.createPrivateMessage(...)	• Centralizes **message creation** to ensure a **consistent structure** for all message types (JOIN, LEAVE, PRIVATE, etc.). • Avoids redundant code by **encapsulating object creation logic** within the factory. • Makes the system **easier to maintain**—if message formatting changes, updates only need to be made in one place.

JUnit Testing

Test Name	Involved Classes/Methods	Description
testConnectAndReceiveJoin()	ClientTest.java Client.connect() (method under test) Client.setOnMessageReceived(Consumer<Message>) Client.isConnected() serverExecutor.submit(Runnable) (mock server logic) mockServerSocket.accept() (server socket handling) ObjectOutputStream.writeObject(Object) (sending messages) ObjectInputStream.readObject() (receiving messages)	This test verifies that a Client instance can successfully connect to a mock server, send a **JOIN** request, and receive the expected response. The mock server, running in a separate thread, accepts the client connection, reads the incoming message, and asserts that it is of type JOIN with the sender ID "testUser". The server then responds with a JOIN message from "server" containing the content "Welcome", which is sent back to the client. On the client side, a listener (setOnMessageReceived) checks the received message to ensure it matches the expected response. The test also ensures that the client connects within 3 seconds (assertTimeoutPreemptively), and assertTrue(client.isConnected()) confirms the connection status. The assertions validate that the client-server communication follows the expected handshake process.
testSetCoordinator()	MemberTest.java Member.isCoordinator() (checks if the member is a coordinator) Member.setCoordinator(boolean) (sets the coordinator status) createFakeMember(String, String, int, boolean) (creates a test member instance)	This test verifies that the setCoordinator method properly updates a Member instance's coordinator status. It starts by creating a test Member using createFakeMember, initializing it with the ID "u2", IP address "127.0.0.1", port 9000, and isCoordinator set to false. The test first asserts that the member is not a coordinator using assertFalse(member.isCoordinator()). Then, it calls setCoordinator(true), changing the coordinator status. A final assertion, assertTrue(member.isCoordinator()), ensures that the change is correctly applied, confirming that the method updates the state as expected.
testConstructorWithStringPayload()	MessageTest.java Message(MessageType, String, String) (constructor being tested) Message.getType() (retrieves the message type) Message.getSenderId() (retrieves the sender ID)	This test verifies that the Message constructor correctly initializes a message object when provided with a MessageType, sender ID, and content. It creates a Message instance of type BROADCAST with "alice" as the sender and "Hello World" as the content. The test then asserts that each field is correctly assigned using assertEquals, ensuring that the message type, sender ID, and content match the expected values. Finally, it confirms that getMemberList() returns null, validating that the

	Message.getContent() (retrieves the message content) Message.getMemberList() (retrieves the member list, expected to be null)	constructor correctly handles cases where no member list is provided.
testRemove Member()	ServerTest.java Server.addMember(String, Member) (adds a member to the server) Server.removeMember(String) (removes a member by ID) Server.getMember(String) (retrieves a member by ID) createFakeMember(String, boolean) (creates a test Member instance)	This test verifies that the removeMember method correctly removes a member from the server while keeping other members unaffected. It starts by creating two fake members, "a" and "b", both initially set as non-coordinators. These members are added to the server using addMember. The test then removes member "a" using removeMember("a"). Assertions confirm that "a" has been successfully removed (assertNull(server.getMember("a"))), while "b" remains in the server (assertNotNull(server.getMember("b"))). This ensures that removeMember functions as expected without affecting unrelated members.

Fault Tolerance

Fault Tolerance Feature	Involved Classes/Methods	Description
Duplicate ID Handling	Server.java ClientHandler.java	Prevents two users from joining with the same ID
Coordinator Re-Election (for unexpected disconnection, same for client)	Server.java CoordinatorElection.java	Uses either lowest ID or highest port strategy to elect new coordinator.

Usage of AI

AI Program	Classes and/or Methods	Contribution
Handling Last Member Disconnection Gracefully: ChatGPT https://chatgpt.com/	MainServer.java server.setOnLastMemberLeft (Runnable) Platform.runLater (Runnable) System.exit(0)	AI enhanced the implementation of setOnLastMemberLeft, ensuring that when the last client disconnects, a confirmation dialog appears before shutting down the server. It improved user experience by preventing unintended shutdowns. **40% est. AI contribution (**suggested structure, dialog flow, and safe thread handling)
Port Availability Check: ChatGPT https://chatgpt.com/	MainServer.java try (ServerSocket _ = new ServerSocket(port)) {} showAlert(String)	AI improved robustness by adding a port availability check, preventing crashes due to an unavailable port. The method ensures that the selected port is valid and not already in use. **35% est. AI contribution** (error handling, validation logic, and alert messaging)