Classes to be tested

Server, User, Supervisor, ChatRoom, Client

Operations to be Tested

1. Server

1. Constructor

* Test case: Invoking the constructor

1. All other functionality tested through its public interface, the Client

2. User

1. Constructor

* Test case: Invoking the constructor

1. Password changing (setting)

* Test case: Creating a User object with a default password and using the changePassword() method to change the User’s password attribute to the new password. This is checked using the getPassword() method.

1. Name setting

* Test case: Creating a User object with a default name and using the setName() method to change the User’s name attribute to the new name. This is checked using the getName() method.

1. User’s active ChatRoom setter

* Test case: Creating a User object with no default active Chatroom. The attribute is set with the setActiveChatRoom() method to alter the activeChatRoom attribute. This is checked using the getActiveChatRoom() method.

1. Name getter

* Test case: Creating a User object with a default name and checking the default value using the getName() method.

1. Password getter

* Test case: Creating a User object with a default password and checked the attribute using the getPassword() method.

3. Supervisor

1. Constructor

* Test case: Invoking the constructor and calling the parent class User’s constructor with the input

1. Password changing (setting)

* Test case: Creating a Supervisor object with one password and ensuring the changePassword() method changes the object’s password attribute to the new password. This is checked using the getPassword() method.

1. Name setting

* Test case: Creating a Supervisor object with one name and ensuring the setName() method changes the object’s password attribute to the new name. This is tested using the getName() method.

1. Supervisor’s ChatRoom name getter

* Test case: Creating a Supervisor object and setting its active ChatRoom using the getActiveChatRoom() method. If the attribute changes from null to getActiveChatRoom()’s input, we know the method is working.

1. Name getter

* Test case: Creating a Supervisor object and invoking the getName() method on the object.

1. Password getter

* Test case: Creating a Supervisor object and invoking the getPassword() method on the object.

4. ChatRoom

1. Adding Users/Supervisors

* Test case: Create a Chatroom that has a host and add a new user with the addUser() method. Then check to see if the amount of users becomes two with the help of the getChatUsersSize() method.

1. Room name getter

* Test case: Create a Chatroom with the help of a message holding the name of the Chatroom. Check to see if the name will appear with the help of getRoomName() method.

1. Chat is unlocked at initialization

* Test case: Create a Chatroom which chatLocked attribute is set to false by default. Check to see if chat is locked with the isLocked() method.

1. Locking the room

* Test case: Create a Chatroom and lock it with the setChatLock() method. Check if its locked with the help of the isLocked() method.

1. Unlocking the room

* Test case: Create a Chatroom and lock it with the setChatLock() method. Then unlock the Chatroom with the setChatUnlock() method. Check if it’s unlocked with the help of the isLocked() method.

1. Removing Users/Supervisors

* Test case: Create a Chatroom and add multiple users to the Chatroom. Then a user is removed from the Chatroom. The Chatroom’s activeUserCount attribute is compared to the expected number of active users.

1. Active user count getter

* Test case: Create a Chatroom and compare the activeUserCount attribute to one.

1. Incrementing the active user count

* Test case: Increment the activeUserCount attribute and compare this value to the expected value with the help of the getActiveUserCount() method.

1. Decrementing the active user count

- Test Case: Increment the activeUserCount attribute and then decrement it and then compare this value to the expected value with the help of the getActiveUserCount() method.

5. Client – Note that this test has to run simultaneously with the Server and the server must be reset every time the test is to be performed.

1. Getting socket after connecting to Server

* Test case: Once a client is connected to the Server, test that the port is ‘1234’.

1. Logging in

* Test case: Login with a valid username and password using the login() method, and check that the authentication variable is true.

1. Creating a chatroom

* Test case: Login and create a new room using createChatroom(). Test that the message received by the server is “VERIFIED”.

1. Joining a chatroom

* Test case: Login, create a new room, leave the room, and joins that same room with the joinChatroom() method. Test that the message received by the server has the status “VERIFIED”.

1. Leaving a chatroom

* Test case: Login, create a new room, and leave the room with the leaveChatroom() method. Test that the message received from the server has the status “VERIFIED”.

1. Changing a password (two parts)

* Test case: Two functions are used for this test. The first logs in and changes the password of the user with the changePass() method. The second function logs in as the same user with the changed password.

1. Displaying users

* Test case: Logs in and uses the displayUsers() method. The returned message from the server should be “VERIFIED”.

1. Locking chatroom

* Test case: Logs in, creates a chatroom, and locks the chatroom using the setChatLock() method. The returned message from the server should be “VERIFIED”.

1. Unlocking chatroom

* Test case: Logs in, creates a chatroom, locks the chatroom, and unlocks the chatroom using the setChatUnlock() method. The returned message from the server should be “VERIFIED”.

1. Displaying chatrooms

* Test case: Logs in and displays chatrooms using the displayChatRooms() method. The returned message from the server should be “VERIFIED”.

1. Sending a chat message

* Test case: Logs in, creates a chatroom, and sends a message using the deliverMessage() method. The returned message from the server should have a text of the username, inputted message, and sent receipt.

1. Creating a user (two parts)

* Test case: Two functions are used to create a user. A supervisor logs in and creates a user with the createUser() method. The second function logs in with the new username and password. We check if the login was successful by checking the authenticated variable.

1. Creating a supervisor (two parts)

* Test case: Two functions are used to create a supervisor. A supervisor logs in and creates a supervisor with the createSuper() method. The second function logs in with the new username and password. We check if the login was successful by checking the authenticated variable.

1. Deleting a user (two parts)

* Test case: Because we created a new user and supervisor above, we delete them. We log in as a supervisor, delete both users using the deleteUser() method. A second function is used to attempt to login using the usernames and passwords from tests XII and XIII. We check if the login failed by checking the authenticated variable.

1. Displaying chat users

* Test case: Login, create a new room, and display the chat users using the displayChatUsers() method. The returned message from the server should have a text of “CurrentUsers\n\*username\*\n”.

1. Logging out

* Test case: Login and logout using the logout() method. We can test this worked by checking if the client socket is closed.

1. Testing authentication when logging in

* Test case: Login with invalid credentials. Use the getAuthenticated() method to ensure that authenticated is false.

1. Setting current room

* Test case: Set the current room using the setCurrRoom() method. Use the getCurrRoom() function to test that the currentRoom variable was changed.

1. Getting current room

* Test case: Test the getCurrRoom() function because the constructor doesn’t initialize it, it should be null.

Instructions

1. Download or clone the repo from GitHub.

2. Import the parent folder into Eclipse as a Java Project

3. Open and then right-click the project in Eclipse’s Project Explorer and go to Properties > Java Build Path, select Libraries on the Java Build Path menu, and right-click on Classpath. Then, on the right side of that window, select “Add Library” and then select JUnit > JUnit 5. Click “Finish” in the pop-up window and then “Apply and Close”

4. Double-click the project in the Eclipse File Explorer so that the project directory expands. Double-click “src”. Then, double-click “(default package)” so that all the .java files are displayed in the File Explorer.

5. Right click “(default package)”, hover over the “Show In Local Terminal” option, and select your terminal of choice. Compile the Server.java file by inputting “javac Server.java” then run the server by inputting “java Server” (both without quotation marks).

6. Back in Eclipse, right click “AllTests.java” and then select Run As > JUnit Test. The tests will then run. NOTE: When retesting, restart the Server by terminating the current running Server with the key inputs: CTRL-C and reinputting “java Server” in your terminal. Repeat this step for all subsequent test runs.