Marvin Sevilla

GitHub: https://github.com/LoloMarty/2024-/tree/main/CS4800/Homework6

```java
public class DriverProgram {
    public static void main(String[] args) {
        ChatServer server = new ChatServer();

        User user1 = server.registerUser("Marvin");
        User user2 = server.registerUser("Sean");
        User user3 = server.registerUser("Nima");

        server.addBlockedUser(user2);

        user1.writeToChat("Hello guys!", new User[]{user2, user3});
        user2.writeToChat("Hi Marvin", new User[]{user1, user3});
        user3.writeToChat("Why are you guys not in class?", new User[]{user1, user2});

        user1.printPercievedChat();
        user2.printPercievedChat();
        user3.printPercievedChat();

        user1.requestUndoLastMessage();

        System.out.println();

        user1.printPercievedChat();
        user2.printPercievedChat();
        user3.printPercievedChat();
    }
}
```

```java
import java.util.LinkedList;

public class ChatServer {
    private static ChatServer instance;
    private static LinkedList<User> listOfUsers;
    private static LinkedList<User> listOfBlockedUsers;

    public ChatServer()
    {
        listOfUsers = new LinkedList<>();
        listOfBlockedUsers = new LinkedList<>();
    }

    public static ChatServer getChatServerInstance()
    {
        if (instance == null)
        {
            instance = new ChatServer();
        }

        return instance;
    }

    public void write(Message givenMessage)
    {
        if (!isSenderBlocked(givenMessage))
        {
            for(User user : givenMessage.getRecipiets())
            {
                user.updatePercievedChat(givenMessage);
            }
        }
    }

    public User registerUser(String givenUsername)
    {
        User newUser = new User(givenUsername);
        listOfUsers.add(newUser);

        return newUser;
    }

    public void unregisterUser(String givenUsername)
    {
        LinkedList<User> users = listOfUsers;
        User currentUser = users.pop();

        System.out.println();

        while(!(givenUsername.equals(currentUser.getUsername())))
        {
            users.addLast(currentUser);
            currentUser = users.pop();
```

```
        }

        currentUser = users.pop();
    }

    public void addBlockedUser(User userToBlock)
    {
        listOfBlockedUsers.add(userToBlock);
    }

    public boolean isSenderBlocked(Message messageToEvaluate)
    {
        boolean returnValue = false;

        for(User user : listOfBlockedUsers)
        {
            if (user.getUsername().equals(messageToEvaluate.getSender())) {
                returnValue = true;
                break;
            }
        }

        return returnValue;
    }

    public void sendUndoRequest(User userRequesting)
    {
        for(User user : listOfUsers)
        {
            user.processUndoLastMessage(userRequesting);
        }
    }
}
```

```java
public class Message {
    private final String sender;
    private final String text;
    private final User[] recipients;
    private final String timestamp;

    public Message(String givenUsername, String givenText, User[] givenRecipients, String givenTimestamp
    {
        this.sender = givenUsername;
        this.text = givenText;
        this.recipients = givenRecipients;
        this.timestamp = givenTimestamp;
    }

    public User[] getRecipiets()
    {
        return this.recipients;
    }

    public String getSender() {
        return sender;
    }

    public String getText() {
        return text;
    }

    public String getTimestamp()
    {return this.timestamp;}
}
```

```java
import java.util.LinkedList;

public class MessageMomento {
    private final Message storedMessage;

    public MessageMomento(Message givenMessage)
    {
        this.storedMessage = givenMessage;
    }

    public Message getChatVersion()
    {
        return this.storedMessage;
    }
}
```

```java
import java.util.Iterator;
import java.util.LinkedList;

public class SearchMessagesByUser implements Iterator<MessageMomento>{
    private LinkedList<MessageMomento> collection;

    public SearchMessagesByUser(LinkedList<MessageMomento> givenCollection)
    {
        this.collection = givenCollection;
    }

    @Override
    public boolean hasNext() {
        LinkedList<MessageMomento> copyList = null;

        for(MessageMomento message : this.collection)
        {
            copyList.addFirst(message);
        }

        copyList.pop();

        if (copyList.getFirst() != null)
        {
            return false;
        }else{
            return true;
        }
    }

    @Override
    public MessageMomento next() {
        MessageMomento returnItem = null;

        if (this.hasNext())
        {
            MessageMomento cycledItem = this.collection.pop();
            this.collection.addLast(cycledItem);
            returnItem = this.collection.getFirst();
        }

        return returnItem;
    }
}
```

```java
import java.util.Iterator;
import java.util.LinkedList;

public class User implements Iterable<MessageMomento>{
    private final ChatServer server;
    private final String username;
    private LinkedList<Message> percievedChat;
    private ChatHistory chatHistory;

    public User (String givenUsername)
    {
        this.username = givenUsername;
        this.server = ChatServer.getChatServerInstance();
        percievedChat = new LinkedList<>();
        this.chatHistory = new ChatHistory();
    }

    public void writeToChat(String givenMessage, User[] givenRecipients)
    {
        server.write(new Message(this.username, givenMessage, givenRecipients, "123"));
    }

    public String getUsername()
    {
        return this.username;
    }

    public void requestUndoLastMessage()
    {
        server.sendUndoRequest(this);
    }

    public void processUndoLastMessage(User userToUndoMessage)
    {
        for(MessageMomento momento : this.chatHistory.getWholeHistory())
        {
            if(momento.getChatVersion().getSender() == userToUndoMessage.getUsername())
            {
                this.chatHistory.getWholeHistory().remove(momento);
            }
        }

        for(Message message : this.percievedChat)
        {
            if(message.getSender() == userToUndoMessage.getUsername()) {
                this.percievedChat.remove(message);
            }
        }
    }

    public void updatePercievedChat(Message givenMessage)

    {
```

```
        this.percievedChat.addLast(givenMessage);
        chatHistory.addMessageToHistory(givenMessage);
    }

    public void printPercievedChat()
    {
        System.out.println(this.username + " sees: ");
        for (Message message : this.percievedChat)
        {
            System.out.println("Timestamp [" + message.getTimestamp() + "] " +
                    message.getSender() + ": " + message.getText());
        }
    }

    @Override
    public Iterator<MessageMomento> iterator() {
        return new SearchMessagesByUser(this.chatHistory.getWholeHistory());
    }

    class ChatHistory {
        LinkedList<MessageMomento> history;

        public ChatHistory()
        {
            this.history = new LinkedList<>();
        }

        public void addMessageToHistory(Message message)
        {
            history.addFirst(new MessageMomento(message));
        }

        public LinkedList<MessageMomento> getWholeHistory()
        {
            return this.history;
        }

        public MessageMomento getPriorHistory()
        {
            return history.pop();
        }

    }
}
```

```java
import org.junit.jupiter.api.Test;

import java.util.Iterator;
import java.util.LinkedList;

import static org.junit.jupiter.api.Assertions.*;

class UserTest {
    @Test
    void processUndoLastMessage() {
        User testUser = new User("testUser");

        Message message1 = new Message("sender1", "Hello", new User[]{testUser}, "123");
        Message message2 = new Message("sender1", "Hi", new User[]{testUser}, "456");

        testUser.updatePercievedChat(message1);
        testUser.updatePercievedChat(message2);

        int initialPerceivedChatSize = testUser.getPercievedChat().size();
        int initialChatHistorySize = testUser.getChatHistory().getWholeHistory().size();

        testUser.processUndoLastMessage(new User("sender1"));

        assertEquals(initialPerceivedChatSize - 1, testUser.getPercievedChat().size());

        assertEquals(initialChatHistorySize - 1, testUser.getChatHistory().getWholeHistory().size());
    }

    @Test
    void updatePercievedChat() {
        User testUser = new User("testUser");

        Message testMessage = new Message("sender1", "Hello", new User[]{testUser}, "123");

        testUser.updatePercievedChat(testMessage);

        LinkedList<Message> perceivedChat = testUser.getPercievedChat();
        assertNotNull(perceivedChat);
        assertEquals(1, perceivedChat.size());
        assertTrue(perceivedChat.contains(testMessage));

        LinkedList<MessageMomento> chatHistory = testUser.getChatHistory().getWholeHistory();
        assertNotNull(chatHistory);
        assertEquals(1, chatHistory.size());
        assertEquals(chatHistory.getFirst().getChatVersion(), testMessage);
    }

    @Test
    void iterator() {
        User testUser = new User("testUser");

        Message message1 = new Message("sender1", "Hello", new User[]{testUser}, "123");
        Message message2 = new Message("sender2", "Hi", new User[]{testUser}, "456");
```

```
54
55          testUser.updatePercievedChat(message1);
56          testUser.updatePercievedChat(message2);
57
58          Iterator<MessageMomento> iterator = testUser.iterator();
59
60          assertNotNull(iterator);
61
62          assertTrue(iterator.hasNext());
63          MessageMomento momento1 = iterator.next();
64          assertEquals(message2, momento1.getChatVersion());
65
66          assertTrue(iterator.hasNext());
67          MessageMomento momento2 = iterator.next();
68          assertEquals(message1, momento2.getChatVersion());
69
70          assertFalse(iterator.hasNext());
71      }
72  }
```

```java
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import java.util.List;


import static org.junit.jupiter.api.Assertions.*;

class ChatServerTest {

    @Test
    void getChatServerInstance() {
        ChatServer server1 = ChatServer.getChatServerInstance();
        ChatServer server2 = ChatServer.getChatServerInstance();

        assertEquals(server1, server2);
    }

    @Test
    void registerUser() {
        ChatServer chatServer = new ChatServer();

        User newUser = chatServer.registerUser("testUser");

        assertNotNull(newUser);

        assertEquals("testUser", newUser.getUsername());

        assertTrue(chatServer.getListOfUsers().contains(newUser));
    }

    @Test
    void addBlockedUser() {
        ChatServer chatServer = new ChatServer();
        User userToBlock = chatServer.registerUser("userToBlock");
        chatServer.addBlockedUser(userToBlock);
        assertTrue(chatServer.getListOfBlockedUsers().contains(userToBlock));
    }

    @Test
    void isSenderBlocked() {
        ChatServer chatServer = new ChatServer();
        User blockedUser = chatServer.registerUser("blockedUser");
        User senderUser = chatServer.registerUser("senderUser");

        chatServer.addBlockedUser(blockedUser);

        Message blockedMessage = new Message("blockedUser", "Hello", new User[]{senderUser}, "123");
        Message unblockedMessage = new Message("unblockedUser", "Hello", new User[]{senderUser}, "456");

        assertTrue(chatServer.isSenderBlocked(blockedMessage));
        assertFalse(chatServer.isSenderBlocked(unblockedMessage));
    }
```

```
    @Test
    void sendUndoRequest() {
        ChatServer chatServer = new ChatServer();
        User user1 = chatServer.registerUser("user1");
        User user2 = chatServer.registerUser("user2");
        User userRequesting = chatServer.registerUser("userRequesting");

        List<Message> messagesUser1 = new ArrayList<>();
        messagesUser1.add(new Message("user1", "Message 1", new User[]{user2}, "123"));
        messagesUser1.add(new Message("user1", "Message 2", new User[]{user2}, "124"));

        List<Message> messagesUser2 = new ArrayList<>();
        messagesUser2.add(new Message("user2", "Message 3", new User[]{user1}, "125"));
        messagesUser2.add(new Message("user2", "Message 4", new User[]{user1}, "126"));

        // Add some messages for each user
        for (Message message : messagesUser1) {
            user1.updatePercievedChat(message);
        }

        for (Message message : messagesUser2) {
            user2.updatePercievedChat(message);
        }

        // Send undo request for userRequesting
        chatServer.sendUndoRequest(userRequesting);

        // Verify that each user processed undo request for userRequesting
        assertFalse(user1.getPercievedChat().isEmpty());
        assertFalse(user2.getPercievedChat().isEmpty());
    }

}
```