Kevin Wong
Professor Davarpanah
CS4800
April 15, 2024

HW6

Github repo link:
https://github.com/kdub8/CS4800-HW6-Iterator-Mediator-Memento-Design-Patterns.git

Code snippets:

```
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class ChatApplication {
   public static void main(String[] args) throws InterruptedException {
       // Create users
       User user1 = new User("Alice");
       User user2 = new User("Bob");
       User user3 = new User("Deby");
       User user4 = new User("Kevin");

       // Register users with the chat server
       ChatServer.getInstance().registerUser(user1);
       ChatServer.getInstance().registerUser(user2);
       ChatServer.getInstance().registerUser(user3);
       ChatServer.getInstance().registerUser(user4);

       // Send messages between users

       // Block user2 by user1
       user1.blockUser(user2);
       user3.blockUser(user4);

       user1.sendMessage(user2, "1. Hello, Bob, it's me, Alice!");

       user1.sendMessage(user4, "2. UNDO THIS MESSAGE");

       // undo the last message from user1
       user1.undo();

       //bob sends to alice, but he got blocked by her
```

```
        user2.sendMessage(user1, "3. Hi, Alice, Bob here!");


        user3.sendMessage(user1, "4. What's going on, Alice! It's me, Deby!");
        user3.sendMessage(user4, "5. What's going on, Kevin! It's me, Deby!");
        user3.sendMessage(user4, "6. Do you mind if I go through your chat
history?");



        TimeUnit.SECONDS.sleep(1);
        user4.sendMessage(user3, "7. Hey Deby, it's me Kevin!"); // this message
is sent 2 seconds after
        user4.sendMessage(user3, "8. Hey Deby, you really should stop looking at
my chat history!");
        user4.sendMessage(user3, "9. I don't think it's okay for other users to
be able to see my history!");
        user4.sendMessage(user3, "10. Hey Deby, what kind of chat messaging app
allows other people to see my history using my username!");

        // Bob attempts to send a message after Alice blocked him
        user2.sendMessage(user1, "11. THIS MESSAGE SHOULD BE GETTING BLOCKED AND
SHOULD NOT BE SEEN BECAUSE ALICE BLOCKED BOB.");

        System.out.println();
        for (User user : ChatServer.getInstance().getUserList()) {
            String allCapsUserName = user.getUsername().toUpperCase();
            System.out.println("----------------------------------------");
            System.out.println("CHAT HISTORY FOR " + allCapsUserName + ":");

            for (Message message : user.getChatHistory().getMessageList()) {
                User[] recipients = message.getRecipients();
                String recipient = recipients[0].getUsername();
                System.out.println(message.getSender().getUsername() + " -> " +
recipient + ": " + message.getContent());
                System.out.println();
            }

        }

        System.out.println();

        System.out.println("-------------------------Viewing the chat history
for " + user3.getUsername()
                + "-------------------------");
        List<Message> msgList = user3.getChatHistory().getMessageList();

        for (Message msg : msgList) {
            System.out.println("Timestamp: " + msg.getTimestamp());
```

```
            System.out.println("Message content: " + msg.getContent());
            System.out.println();
        }

        System.out.println();

        System.out.println("----------------------Allowing User 3 to view the
chat history for " + user4.getUsername()
                + "-------------------------");
        List<Message> userHistory = user3.getChatHistoryForUser(user4);

        for (Message msg : userHistory) {
            System.out.println("Timestamp: " + msg.getTimestamp());
            System.out.println("Message content: " + msg.getContent());
            System.out.println();
        }
        ////////////////////////////////////////////////////////////////////////
        System.out.println("------------------------TESTING
ITERATORS------------------------");

        // Demonstrate iterator for User's chat history
        System.out.println("Iterating over User 1's chat history:");
        Iterator<Message> userIterator = user1.iterator(user1);
        while (userIterator.hasNext()) {
            Message message = userIterator.next();
            User[] recipients = message.getRecipients();
            String recipient = recipients[0].getUsername();
            System.out.println(message.getSender().getUsername() + " -> " +
                    recipient + ": " +
                    message.getContent());
        }

        // Demonstrate iterator for ChatHistory
        System.out.println("\nIterating over User 4's ChatHistory:");
        ChatHistory chatHistory = user4.getChatHistory();
        Iterator<Message> chatHistoryIterator = chatHistory.iterator(user4);
        while (chatHistoryIterator.hasNext()) {
            Message message = chatHistoryIterator.next();
            User[] recipients = message.getRecipients();
            String recipient = recipients[0].getUsername();
            System.out.println(message.getSender().getUsername() + " -> " +
                    recipient + ": " +
                    message.getContent());
        }


System.out.println("----------------------SearchMessagesByUser----------------
--------");
```

```
        // Search for messages for User 3
        //ChatHistory chatHistory3 = user3.getChatHistory();
        System.out.println("Before creating SearchMessagesByUser");
        SearchMessagesByUser searchMessagesByUser3 = new
SearchMessagesByUser(user3);
        System.out.println("After creating SearchMessagesByUser");
        Iterator<Message> user3Iterator = searchMessagesByUser3.iterator(user3);
        while (user3Iterator.hasNext()) {
            Message message = user3Iterator.next();
            User[] recipients = message.getRecipients();
            String recipient = recipients[0].getUsername();
            System.out.println(message.getSender().getUsername() + " -> " +
                    recipient + ": " +
                    message.getContent());
        }



    }
}
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class ChatHistory implements IterableByUser {
    private List<Message> messages;
    // Constructor to initialize the ChatHistory object
    public ChatHistory() {
        this.messages = new ArrayList<>();
    }
    // Method to add a message to the chat history
    public void addMessage(Message message) {
        messages.add(message);
    }
    // Method to remove a message from the chat history
    public void removeMessage(Message message) {
        messages.remove(message);
    }

    public Message getLastMessage() {
        return messages.isEmpty() ? null : messages.get(messages.size() - 1);
    }

    public List<Message> getMessageList() {
        return messages;
    }
```

```
    public void removeLastMessage() {
        if (!messages.isEmpty()) {
            messages.remove(messages.size() - 1);
        }
    }
    // Method to create an iterator for iterating over messages in the chat
history
    @Override
    public Iterator<Message> iterator(User userToSearchWith) {
        return new ChatHistoryIterator(this.messages, userToSearchWith);
    }
    // Iterator class for iterating over messages in the chat history
     public static class ChatHistoryIterator implements Iterator<Message> {
        private List<Message> messages;
        private int index;
        private User userToSearchWith;

        // Constructor to initialize the iterator
        public ChatHistoryIterator(List<Message> messages, User
userToSearchWith) {
            this.messages = messages;
            this.userToSearchWith = userToSearchWith;
            this.index = 0;
        }
        // Method to check if there are more messages for the specified user
        @Override
        public boolean hasNext() {
            // Implement hasNext() to check if there are more messages for the
specified user
            while (index < messages.size()) {
                Message message = messages.get(index);
                if (message.getSender().equals(userToSearchWith) ||
containsRecipient(message, userToSearchWith)) {
                    return true;
                }
                index++;
            }
            return false;
        }
        // Method to return the next message for the specified user
        @Override
        public Message next() {
            // Implement next() to return the next message for the specified
user
            while (index < messages.size()) {
                Message message = messages.get(index);
                index++;
```

```
                if (message.getSender().equals(userToSearchWith) ||
Arrays.asList(message.getRecipients()).contains(userToSearchWith)) {
                    return message;
                }
            }
            return null;
        }
        // Method to check if a message contains a specific recipient
        private boolean containsRecipient(Message message, User user) {
            for (User recipient : message.getRecipients()) {
                if (recipient.equals(user)) {
                    return true;
                }
            }
            return false;
        }
    }
}
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

// Mediator class for managing user interactions
public class ChatServer {
    private static ChatServer instance;
    private List<User> registeredUsers;
    private Map<User, List<User>> blockedUsers; // Map to store blocked users
    // Private constructor to prevent instantiation from outside
    private ChatServer() {
        this.registeredUsers = new ArrayList<>();
        this.blockedUsers = new HashMap<>();
    }

    public List<User> getUserList() {
        return this.registeredUsers;
    }
    // Get the singleton instance of ChatServer
    public static ChatServer getInstance() {
        if (instance == null) {
            instance = new ChatServer();
        }
        return instance;
    }

    public void registerUser(User user) {
        registeredUsers.add(user);
```

```
    }

    public void unregisterUser(User user) {
        registeredUsers.remove(user);
    }

    public void sendMessage(User sender, User recipient, String messageContent)
{
        if (registeredUsers.contains(sender) &&
registeredUsers.contains(recipient)) {
            recipient.receiveMessage(new Message(sender, new User[] { recipient
}, messageContent));
        }
    }
    // Block a user from communicating with another user
    public void blockUser(User blocker, User userToBlock) {
        if (!blockedUsers.containsKey(blocker)) {
            blockedUsers.put(blocker, new ArrayList<>());
        }
        blockedUsers.get(blocker).add(userToBlock);
    }

    // Unblock a user, allowing them to communicate again
    public void unblockUser(User unblocker, User userToUnblock) {
        if (blockedUsers.containsKey(unblocker)) {
            List<User> blockedList = blockedUsers.get(unblocker);
            blockedList.remove(userToUnblock);
        }
    }
    // Get a map of blocked users
    public Map<User, List<User>> getBlockedUsers() {
        return blockedUsers;
    }

}
```

```
import java.util.Iterator;
// Interface for classes that can be iterated over by a specific user
public interface IterableByUser {
    Iterator iterator(User userToSearchWith);
}
```

```
import java.util.Date;

public class Message {
```

```
    private User sender;
    private User[] recipients;
    private Date timestamp;
    private String content;
    private MessageMemento memento;

    public Message(User sender, User[] recipients, String content) {
        this.sender = sender;
        this.recipients = recipients;
        this.timestamp = new Date();
        this.content = content;
        this.memento = new MessageMemento(this);
    }

    public MessageMemento createMemento() {
        return new MessageMemento(this);
    }



    public User getSender() {
        return this.sender;
    }

    public void setSender(User sender) {
        this.sender = sender;
    }

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

    public void setRecipients(User[] recipients) {
        this.recipients = recipients;
    }

    public Date getTimestamp() {
        return this.timestamp;
    }

    public void setTimestamp(Date timestamp) {
        this.timestamp = timestamp;
    }

    public String getContent() {
        return this.content;
    }

    public void setContent(String content) {
```

```
        this.content = content;
    }


}
```

```
import java.util.Date;
// Memento class to store the state of a Message
public class MessageMemento {
    private Date timestamp;
    private String content;
    private Message message;
    private String state;
    // Constructor to create a memento from a Message object
    public MessageMemento(Message message) {
        this.timestamp = message.getTimestamp();
        this.content = message.getContent();
        this.message = message;
    }

    public MessageMemento(String state) {
        this.state = state;
    }

    public void restoreFromMemento(User sender, User recipient, Message message)
{
        Message lastMessage = sender.getChatHistory().getLastMessage();
        ChatHistory senderHistory = sender.getChatHistory();
        senderHistory.removeMessage(lastMessage);
        recipient.getChatHistory().removeMessage(message);
    }

    public Message getState() {
        return message;
    }

    public void setState(Message message) {
        this.message = message;
    }

    public String getContent() {
        return this.content;
    }

    public void print() {
        System.out.println("Timestamp: " + timestamp);
        System.out.println("Content: " + content);
```

```
    }

    public Date getTimestamp() {
        return timestamp;
    }

}
```

```
import java.util.Iterator;
import java.util.List;

public class SearchMessagesByUser implements IterableByUser {
    private List<Message> messages;
    private  User user;
    public SearchMessagesByUser(User user) {
        this.user = user;
        this.messages = user.getChatHistoryForUser(user);
    }


    @Override
    public Iterator<Message> iterator(User userToSearchWith) {
        return new
SearchMessagesByUserIterator(this.user.getChatHistory().getMessageList(),
userToSearchWith);
    }

    private static class SearchMessagesByUserIterator implements
Iterator<Message> {
        private List<Message> messages;
        private int index;
        private User userToSearchWith;
        public SearchMessagesByUserIterator(List<Message> messages, User
userToSearchWith) {
            this.messages = messages;
            this.userToSearchWith = userToSearchWith;
            this.index = 0;
        }

        @Override
        public boolean hasNext() {
            // Implement hasNext() to check if there are more messages for the
specified user
            while (index < messages.size()) {
                Message message = messages.get(index);
                if (message.getSender().equals(userToSearchWith) ) {
                    return true;
```

```
                }
                index++;
            }
            return false;
        }

        @Override
        public Message next() {
            // Implement next() to return the next message for the specified
user
            while (index < messages.size()) {
                Message message = messages.get(index);
                index++;
                if (message.getSender().equals(userToSearchWith) ) {
                    return message;
                }
            }
            return null;
        }
    }
}
```

```
// Importing JUnit assert methods for testing

// Importing JUnit test annotation
import org.junit.Test;

import java.util.*;

import static org.junit.Assert.*;

// Test class for the Chat Application
public class TestApplication {

    // Test method for checking the initialization of a User object
    @Test
    public void userNameTest() {
        User user1 = new User("NEW USER");
        String actualUserName = "NEW USER";
        // Verifying that the actual username matches the expected username
        assertEquals(actualUserName, user1.getUsername());
    }

    // Test method for checking if a user is registered
    @Test
    public void registeredUserTest() {
        User user2 = new User("USER 2");
```

```
        ChatServer.getInstance().registerUser(user2);
        List<User> actualRegisteredUsers = new ArrayList<>();
        actualRegisteredUsers = ChatServer.getInstance().getUserList();
        // Verifying that the actual registered users list contains the user
        assertEquals(actualRegisteredUsers,
ChatServer.getInstance().getUserList());
    }

    // Test method for checking if a user can be blocked
    @Test
    public void blockUserTest() {
        User user3 = new User("USER 3");
        User user4 = new User("USER 4");

        ChatServer.getInstance().registerUser(user3);
        ChatServer.getInstance().registerUser(user4);
        user3.blockUser(user4);
        Map<User, List<User>> blockedUsers = new HashMap<>();
        blockedUsers = ChatServer.getInstance().getBlockedUsers();
        assertEquals(blockedUsers, ChatServer.getInstance().getBlockedUsers());
        assertEquals(blockedUsers.containsKey(user3),
ChatServer.getInstance().getBlockedUsers().containsKey(user3));
        // Verifying that user4 is blocked by user3
        assertTrue(blockedUsers.get(user3).contains(user4));
        // Verifying that user4 cannot send messages to user3
        assertNull(blockedUsers.get(user4));
        // Attempting to send a message from user4 to user3 while blocked
        user4.sendMessage(user3, "Hey! PLEASE UNBLOCK ME!");
        // Verifying that the message is not received
        assertEquals(blockedUsers, ChatServer.getInstance().getBlockedUsers());

    }

    // Test method for checking if a user can send a message
    @Test
    public void sendMessageTest() {
        User user5 = new User("USER 5");
        User user6 = new User("USER 6");

        ChatServer.getInstance().registerUser(user5);
        ChatServer.getInstance().registerUser(user6);

        user5.sendMessage(user6, "What is up my brethren.");
        user6.sendMessage(user5, "Nothing much bro, hbu?");
        user5.sendMessage(user6, "Just vibing my dude.");
        // Verifying that the last message sent by user5 is correct
        String lastUser5Message =
user5.getChatHistory().getLastMessage().getContent();
        String ActualLastUser5Message = "Just vibing my dude.";
```

```
        assertEquals(ActualLastUser5Message, lastUser5Message);
    }

    // Test method for checking the details of a message
    @Test
    public void messageTest() {
        User user7 = new User("USER 7");
        User user8 = new User("USER 7");
        ChatServer.getInstance().registerUser(user7);
        ChatServer.getInstance().registerUser(user8);
        user7.sendMessage(user8, "What's goooood...");
        Date actualDate =
user7.getChatHistory().getLastMessage().getTimestamp();
        User actualRecipient = user8;
        User actualSender = user7;
        String actualContent = "What's goooood...";
        User[] recipients =
user7.getChatHistory().getLastMessage().getRecipients();
        // Verifying the details of the last message sent by user7
        assertEquals(user7.getChatHistory().getLastMessage().getTimestamp(),
actualDate);
        assertEquals(recipients[0], actualRecipient);
        assertEquals(user7.getChatHistory().getLastMessage().getSender(),
actualSender);
        assertEquals(user7.getChatHistory().getLastMessage().getContent(),
actualContent);
    }

    // Test method for checking if a message can be undone
    @Test
    public void undoTest() {
        User user7 = new User("USER 7");
        User user8 = new User("USER 7");
        ChatServer.getInstance().registerUser(user7);
        ChatServer.getInstance().registerUser(user8);
        user7.sendMessage(user8, "Hola muchachos!");
        user7.undo();
        int actualSize;
        actualSize = user7.getChatHistory().getMessageList().size();
        // Verifying that the message is undone
        assertEquals(actualSize, 0);
    }
}
```

```java
import java.util.Map;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

public class User implements IterableByUser {
   private String username;
   private ChatHistory chatHistory;
   private Map<User, Boolean> blockedUsers; // Map to store blocked users

   public User(String username) {
       this.username = username;
       this.chatHistory = new ChatHistory();
       this.blockedUsers = new HashMap<>();
   }

   public void sendMessage(User recipient, String messageContent) {
       if (!ChatServer.getInstance().getBlockedUsers().containsKey(this)) {
           Message message = new Message(this, new User[] { recipient },
messageContent);
           ChatServer.getInstance().sendMessage(this, recipient,
messageContent);
           chatHistory.addMessage(message);
       } else {
           System.out.println(recipient.getUsername() + " cannot receive any
messages from " + this.username
                   + " because " + this.username + " has been blocked by " +
recipient.getUsername() + ".");
       }

   }

   public void undo() {
       Message lastMessage = this.chatHistory.getLastMessage();
       MessageMemento memento = new MessageMemento(lastMessage);
       chatHistory.removeLastMessage();
       System.out.println("Message undone: " + memento.getContent());

   }

   public void blockUser(User userToBlock) {
       ChatServer.getInstance().blockUser(this, userToBlock);

   }

   public void unblockUser(User userToUnblock) {
       ChatServer.getInstance().unblockUser(this, userToUnblock);
```

```
    }

    public void receiveMessage(Message message) {
        if (!blockedUsers.containsKey(message.getSender())) {
            chatHistory.addMessage(message);
        }
    }

    public ChatHistory getChatHistory() {
        return this.chatHistory;
    }

    public List<Message> getChatHistoryForUser(User otherUser) {
        List<Message> userChatHistory = new ArrayList<>();
        for (Message message : chatHistory.getMessageList()) {
            if (message.getSender().equals(otherUser) ||
Arrays.asList(message.getRecipients()).contains(otherUser)) {
                userChatHistory.add(message);
            }
        }
        return userChatHistory;

    }

    public Iterator<Message> iterator(User userToSearchWith) {
        return this.chatHistory.iterator(userToSearchWith);
    }

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

Output from driver:

```
Alice cannot receive any messages from Bob because Bob has been blocked by Alice.
Deby cannot receive any messages from Kevin because Kevin has been blocked by Deby.
Deby cannot receive any messages from Kevin because Kevin has been blocked by Deby.
Deby cannot receive any messages from Kevin because Kevin has been blocked by Deby.
Deby cannot receive any messages from Kevin because Kevin has been blocked by Deby.
Alice cannot receive any messages from Bob because Bob has been blocked by Alice.

---------------------------------------
CHAT HISTORY FOR ALICE:
Alice -> Bob: 1. Hello, Bob, it's me, Alice!

Deby -> Alice: 4. What's going on, Alice! It's me, Deby!

---------------------------------------
CHAT HISTORY FOR BOB:
Alice -> Bob: 1. Hello, Bob, it's me, Alice!

---------------------------------------
CHAT HISTORY FOR DEBY:
Deby -> Alice: 4. What's going on, Alice! It's me, Deby!

Deby -> Kevin: 5. What's going on, Kevin! It's me, Deby!

Deby -> Kevin: 6. Do you mind if I go through your chat history?

---------------------------------------
CHAT HISTORY FOR KEVIN:
Deby -> Kevin: 5. What's going on, Kevin! It's me, Deby!

Deby -> Kevin: 6. Do you mind if I go through your chat history?


-------------------------Viewing the chat history for Deby-------------------------
Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 4. What's going on, Alice! It's me, Deby!

Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 5. What's going on, Kevin! It's me, Deby!

Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 6. Do you mind if I go through your chat history?
```

```
Deby -> Kevin: 5. What's going on, Kevin! It's me, Deby!

Deby -> Kevin: 6. Do you mind if I go through your chat history?



------------------------Viewing the chat history for Deby------------------------
Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 4. What's going on, Alice! It's me, Deby!

Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 5. What's going on, Kevin! It's me, Deby!

Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 6. Do you mind if I go through your chat history?



----------------------Allowing User 3 to view the chat history for Kevin------------------------
Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 5. What's going on, Kevin! It's me, Deby!

Timestamp: Mon Apr 15 23:10:18 PDT 2024
Message content: 6. Do you mind if I go through your chat history?

-----------------------TESTING ITERATORS-----------------------
Iterating over User 1's chat history:
Alice -> Bob: 1. Hello, Bob, it's me, Alice!
Deby -> Alice: 4. What's going on, Alice! It's me, Deby!

Iterating over User 4's ChatHistory:
Deby -> Kevin: 5. What's going on, Kevin! It's me, Deby!
Deby -> Kevin: 6. Do you mind if I go through your chat history?
----------------------SearchMessagesByUser------------------------
```

Output from passed junit test cases: