PROG5121 (POE)

Chat Messaging Application

Name: Lesego Ramoba

Student Number: ST10490519

Module: Programming5121

Lecture:

Sate: 14 April 2025

Contents

System Requirements & Specifications

The Chat Messaging Application is a Java console-based program that uses Object-Oriented Programming (OOP) principles to allow users to send, view, and mark messages as read. The application uses a ‘Message’ class to define each message object, a `Message Manager` class to store and manage messages, and a ‘Main’ class with a console menu. The application uses Arrays/Array Lists, Loops, Decision structures (if switch), and user input via a scanner. The application is designed to demonstrate understanding of basic OOP design, class interaction, message handling, and user interaction in Java. The application is ideal for demonstrating understanding of basic OOP design, class interaction, message handling, and user interaction in Java.

Class Design

Class Design

This Chat App consists of three main Java classes:

1. Message Class

Represents an individual message in the chat app.

Attributes:

- String payload: The message text
- String sender: The name of the person sending the message
- String receiver: The name of the person receiving the message
- Boolean isSent: Whether the message was sent
- Boolean isReceived: Whether the message was received
- Boolean isRead: Whether the message was read.

Methods

- markSent(): Marks the message as sent
- markReceived(): Marks the message as received
- markRead(): Marks the message as read
- toString(): Returns the message details as a formatted string

2. Message Manager Class

Handles all message-related operations.

Attributes:

- Array List<Message> messages: Stores all messages

Methods:

- send Message(String, String, String): Creates and sends a new message
- receive Messages(String): Views all messages for a receiver
- read Messages(String): Marks all messages for a receiver as read

3. Main Class

Acts as the user interface using the console.

Function:

- Displays a menu
- Accepts input
- Calls the relevant methods from Message Manager to send, view, or read messages

3. Pseudocode

```
Start Program
Repeat the following until user exits:
    Display Menu:
        1. Send Message
        2. View Messages
        3. Mark Messages as Read
        4. Exit
    Ask user to enter a choice
    If choice is 1:
        Ask for sender name
        Ask for receiver name
        Ask for message text
        Create a Message object
        Mark it as sent
        Store it in message list
    If choice is 2:
        Ask for user name (receiver)
        Find messages where receiver matches user
        Mark them as received
        Display them
    If choice is 3:
        Ask for user name (receiver)
        Find messages where receiver matches user
        Mark them all as read
    If choice is 4:
        Exit the program
End Program
```

4. Java Code

Message.java

```java
public class Message {
    private String payload;
    private String sender;
    private String receiver;
    private boolean isSent;
    private boolean isReceived;
    private boolean isRead;

    public Message(String payload, String sender, String receiver) {
        this.payload = payload;
        this.sender = sender;
        this.receiver = receiver;
        this.isSent = false;
        this.isReceived = false;
        this.isRead = false;
    }

    public String getPayload() { return payload; }
    public String getSender() { return sender; }
    public String getReceiver() { return receiver; }

    public boolean isSent() { return isSent; }
    public boolean isReceived() { return isReceived; }
    public boolean isRead() { return isRead; }

    public void markSent() { this.isSent = true; }
    public void markReceived() { this.isReceived = true; }
    public void markRead() { this.isRead = true; }

    @Override
    public String toString() {
        return "From: " + sender + " -> To: " + receiver + "\n" +
               "Message: " + payload + "\n" +
               "Sent: " + isSent + ", Received: " + isReceived + ", Read: " + isRead;
    }
}
```

MessageManager.java

```
 import java.util.ArrayList;

public class MessageManager {
    private ArrayList<Message> messages = new ArrayList<>();

    public void sendMessage(String text, String sender, String receiver) {
        if (text.isEmpty()) {
            System.out.println("Error: Message cannot be empty.");
            return;
        }
        Message msg = new Message(text, sender, receiver);
        msg.markSent();
        messages.add(msg);
        System.out.println("Message sent successfully!");
    }

    public void receiveMessages(String receiver) {
        boolean found = false;
        for (Message m : messages) {
            if (m.getReceiver().equalsIgnoreCase(receiver)) {
                m.markReceived();
                System.out.println("\n" + m);
                found = true;
            }
        }
        if (!found) {
            System.out.println("No new messages for " + receiver);
        }
    }

    public void readMessages(String receiver) {
        boolean hasUnread = false;
        for (Message m : messages) {
            if (m.getReceiver().equalsIgnoreCase(receiver) && !m.isRead()) {
                m.markRead();
                hasUnread = true;
            }
        }
        if (hasUnread) {
            System.out.println("All your messages have been marked as read.");
        } else {
            System.out.println("No unread messages to mark as read.");
        }
    }
}
```

Main.java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        MessageManager manager = new MessageManager();
        int choice;

        do {
            System.out.println("\n--- Chat App ---");
            System.out.println("1. Send Message");
            System.out.println("2. View Messages");
            System.out.println("3. Mark Messages as Read");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Sender Name: ");
                    String sender = scanner.nextLine();
                    System.out.print("Receiver Name: ");
                    String receiver = scanner.nextLine();
                    System.out.print("Message Text: ");
                    String text = scanner.nextLine();
                    manager.sendMessage(text, sender, receiver);
                    break;

                case 2:
                    System.out.print("Enter your name to view messages: ");
                    String viewer = scanner.nextLine();
                    manager.receiveMessages(viewer);
                    break;

                case 3:
                    System.out.print("Enter your name to mark messages as read: ");
                    String reader = scanner.nextLine();
                    manager.readMessages(reader);
                    break;

                case 4:
                    System.out.println("Exiting Chat App. Goodbye!");
                    break;

                default:
                    System.out.println("Invalid choice. Please enter 1-4.");
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

5. Test Plan Table

Test Case	**Description**	**Input**	**Expected Output**	**Actual Output**	**Result**
TC1	Send a message	Sender: Lesego, Receiver: John, Text: Hello	Message sent successfully	Same	Pass
TC2	View messages for user	Receiver: John	Display message + received = true	Same	Pass
TC3	Mark messages as read	Receiver: John	Message read = true	Same	Pass
TC4	Invalid menu option	Input: 6	Display “Invalid choice”	Same	Pass
TC5	View when no messages	Receiver: Unknown	Display “No new messages”	Same	Pass

Conclusion

I was able to comprehend the fundamental ideas of Java's Object-Oriented Programming (OOP) thanks to this chat messaging application. I gained knowledge about creating and implementing classes, managing data with Array Lists, and controlling program flow with loops and conditional logic. I also gained experience with console-based interface, status flags, and user input through the project. All things considered, it was a useful and fulfilling task that enhanced my programming and problem-solving abilities.

Sample Output

```
Chat App

1. Send Message

2. View Messages

3. Mark Messages as Read

4. Exit

Enter your choice: 1


Sender Name: Lesego
```

```
Receiver Name: John
Message Text: Hello John, are you done with the assignment?
Message sent successfully!

Chat App
1. Send Message
2. View Messages
3. Mark Messages as Read
4. Exit
Enter your choice: 2

Enter your name to view messages: John

From: Lesego -> To: John
Message: Hello John, are you done with the assignment?
Sent: true, Received: true, Read: false
```