

PROG5121
Assignment 3
ST10472114
Gareth De Jager

Link for personal GitHub

<https://github.com/GDJCodes/ST10472114-Assignment-3>

Link for GitHub Classroom

<https://github.com/IIEVCPMB/programming-1a-poe-GDJCodes>

Below is the code first for login and then for message class and after is me manually testing with the tests provided in POE:

Login Code:

```
package com.mycompany.login;
```

```
/*
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this  
license
```

```
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
```

```
*/
```

```
import java.io.FileNotFoundException;
```

```
import javax.swing.*;
```

```
/**
```

```
 *
```

```
 * @author garet
```

```
*/
```

```
public class Login {

    static String firstName;

    static String lastName;

    static String username;

    static String password;

    static String cellPhone;


    public static boolean checkUserName() {

        return username.contains("_") && username.length() <= 5;

    }


    public static boolean checkPasswordComplexity() {

        return password.length() > 8 && password.matches(".*[A-Z].*") &&
password.matches(".*\\d.*") && password.matches(".*[!@#$%^&*()_+=<>?].*");

    }


    public static boolean checkCellPhoneNumber() {

        return cellPhone.matches("\\+27\\d{9}");

    }


    public static String registerUser() {

        firstName = JOptionPane.showInputDialog("Please enter your first name: ");

        if(firstName == null){

            JOptionPane.showMessageDialog(null, "Cancelled");

            System.exit(0);

        }

        lastName = JOptionPane.showInputDialog("Please enter your last name: ");
```

```

if(lastName == null){

    JOptionPane.showMessageDialog(null, "Cancelled");

    System.exit(0);

}

while(true){

    username = JOptionPane.showInputDialog("Please enter your username: \n- Contains an
underscore. \n- No more than 5 characters long.");

    if(username == null)return "Registration cancelled";

    if(checkUserName()){

        JOptionPane.showMessageDialog(null, "Welcome " + firstName + " " + lastName + " it is
great to see you.");

    } else {

        JOptionPane.showMessageDialog(null, "Username is incorrectly formatted, please ensure
that your username contains an underscore and is no more than five characters in length.");

        continue;

    }

    password = JOptionPane.showInputDialog("Please enter a password: \n- At least eight
characters long. \n- Contain a capital letter. \n- Contain a number. \n- Contain a special
character.");

    if(password == null) return "Registraiton cancelled";

    if(checkPasswordComplexity()){

        JOptionPane.showMessageDialog(null, "Password successfully captured.");

    }else{

        JOptionPane.showMessageDialog(null, "Password is not correctly formatted, please ensure
that the password contains at least eight characters, a capital letter, a number, and a special
character.");

        continue;

    }

    cellPhone = JOptionPane.showInputDialog("Please enter a valid cell phone number starting
with the international country code: ");

    if(cellPhone == null) return "Registraiton cancelled";

    if(checkCellPhoneNumber()){

        JOptionPane.showMessageDialog(null, "Cell number successfully captured.");

```

```

    }else{

        JOptionPane.showMessageDialog(null, "Cell number is incorrectly formatted or does not
        contain an international code, please correct the number and try again.");

    }

    return "User has been registered successfully!";

}

}

public String getPassword(){

    return password;

}

public static void main(String[] args) {

    String registrationMessage = registerUser();

    JOptionPane.showMessageDialog(null, registrationMessage);

    if (registrationMessage.contains("successfully")) {

        boolean loginSuccess = false;

        while (!loginSuccess) {

            String loginUsername = JOptionPane.showInputDialog("Enter username to log in: ");

            String loginPassword = JOptionPane.showInputDialog("Enter password: ");

            loginSuccess = loginUser(loginUsername, loginPassword);

            JOptionPane.showMessageDialog(null, returnLoginStatus(loginSuccess));

            if(!loginSuccess){

                int retry = JOptionPane.showConfirmDialog(null, "Would you like to try again?", "login
                failed.", JOptionPane.YES_NO_OPTION);

                if (retry != JOptionPane.YES_OPTION)

                    break;

            }

        }

    }

```

```

    }

    Login login = new Login();

    Message chatApp = new Message();

    try{
        chatApp.run(login);
    }catch(FileNotFoundException e){
        JOptionPane.showMessageDialog(null, "Message not found: " + e.getMessage());
    }
}

public static boolean loginUser(String loginUsername, String loginPassword){
    return loginUsername.equals(username) && loginPassword.equals(password);
}

public static String returnLoginStatus(boolean loginSuccess) {
    if (loginSuccess) {
        return "A successful login.";
    } else {
        return "A failed login.";
    }
}
}

}

```

Message class code:

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */

package com.mycompany.login;

```

```
import java.awt.Dimension;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 *
 * @author garet
 */
public class Message {

    private ArrayList<String> draftMessages = new ArrayList<>();
    private ArrayList<String> disregardedMessages = new ArrayList<>();

    public void feedbackPrompt(){
        String filled = "\u2605";
        String empty = "\u2606";
```

```

String[] stars = {
    filled + empty + empty + empty + empty + " - Poor",
    filled + filled + empty + empty + empty + " - Fair",
    filled + filled + filled + empty + empty + " - Good",
    filled + filled + filled + filled + empty + " - Very Good",
    filled + filled + filled + filled + filled + " - Excellent"
};

String rating = (String) JOptionPane.showInputDialog(
    null,
    "How would you rate your experience",
    "Rate us!",
    JOptionPane.QUESTION_MESSAGE,
    null,
    stars,
    stars[2]
);

if(rating != null){
    JOptionPane.showMessageDialog(null, "Thank you for the feedback! \n You rated us: " +
rating);
}
}

public boolean checkMessageID(String id) {
    return id != null && id.matches("\\d{10}");
}

public static int checkRecipientCell(String number){
    if (number == null)
        return 0;

    if (number.startsWith("+27") && number.length() == 12 &&
number.substring(3).matches("\\d{9}")){
        return 1;
    }
}

```

```

    }return 0;
}

public void startChat(){
    JOptionPane.showMessageDialog(null, "Welcome to QuickChat");

}

public void run(Login login) throws FileNotFoundException {

    JOptionPane.showMessageDialog(null, "Login Successful. Welcome, " + Login.username +
"!");

    boolean running = true;
    while (running){
        String[] options = {
            "1 - Send Message",
            "2 - Show recently sent messages",
            "3 - Messages",
            "4 - Quit"
        };
        String choiceString = (String) JOptionPane.showInputDialog(null,
            "Choose an option:",
            "Main menu",
            JOptionPane.PLAIN_MESSAGE,
            null,
            options,
            options[0]
        );
        if (choiceString == null){
            feedbackPrompt();
            System.exit(0);
            continue;
        }
    }
}

```



```

int choice = Integer.parseInt(choiceString.substring(0, 1));
switch (choice){
    case 1 -> sendMessage(Login.username);
    case 2 -> JOptionPane.showMessageDialog(null, "Coming soon.");
    case 3 -> messagesMenu();
    case 4 -> {
        JOptionPane.showMessageDialog(null, "Session ended.");
        deleteMessagesOnExit();
        feedbackPrompt();
        running = false;
    }

    default -> JOptionPane.showMessageDialog(null, "Invalid option");
}

}

}

private List<String> loadMessagesFromJson(){
List<String> messages = new ArrayList<>();

try{
    File file = new File("messages.json");
    if(!file.exists()){
        file.createNewFile();
    }
    return messages;
}

try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
    String line;
    while ((line = reader.readLine()) != null){

```

```

    try{
        JSONObject obj = new JSONObject(line);
        String from = obj.optString("from");
        String to = obj.optString("to");
        String content = obj.optString("message");
        messages.add("From: " + from + " to " + to + ": " + content);
    }catch(JSONException je){
        System.out.println("Error reading! " + je.getMessage());
    }
} }

}catch(IOException e){
    JOptionPane.showMessageDialog(null, "Error reading file: " + e.getMessage());
}

return messages;
}

```

```

public void sendMessage(String username){

```

```

    String recipient;

```

```

    while(true){

```

```

        recipient = JOptionPane.showInputDialog("Enter recipients number: ");

```

```

        if (recipient == null) return;

```

```

        if (recipient.matches("\\+27\\d{9}")){

```

```

            break;

```

```

        }

```

```

    else{

```

```

        JOptionPane.showMessageDialog(null,"Recipient cannot be empty.");

```

```

        return;
    }
}

int messageCount = 0;

try{
    String countInput = JOptionPane.showInputDialog("How many messages would you like to send?");

    if (countInput == null)

        System.exit(0);

    messageCount = Integer.parseInt(countInput);

    if (messageCount <= 0) {
        JOptionPane.showMessageDialog(null, "Enter number.");
        return;
    }
}catch(NumberFormatException e){
    JOptionPane.showMessageDialog(null, "Invalid number");
    return;
}

for (int i = 1; i <= messageCount; i++){
    String message = JOptionPane.showInputDialog("Enter the message: ");
    if (message == null || message.trim().isEmpty()){
        disregardedMessages.add("Disregarded message!");
        return;
    }

    String messageID = checkMessageID();
    String messageHash = createMessageHash(messageID, i, message);

```

```

String fullMessage = "[Message ID: " + messageld + "]\nHash: " + messageHash +
    "\nFrom: " + username + " to " + recipient + ": " + message;
String[] options = {"Send", "Disregard", "Store for later" };
int choice = JOptionPane.showOptionDialog(
    null,
    "What would you like to do?" + fullMessage,
    "Confirm Message",
    JOptionPane.DEFAULT_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[0]
);
if(options == null){
    JOptionPane.showMessageDialog(null, "Cancelled!");
}

```

```

JSONObject messageObj = new JSONObject();
messageObj.put("Message ID", messageld);
messageObj.put("Message Hash", messageHash);
messageObj.put("from",username);
messageObj.put("to", recipient);
messageObj.put("message", message);

switch(choice){
    case 0 -> {
        try(FileWriter file = new FileWriter("messages.json", true)){
            file.write(messageObj.toString() + "\n");
            JOptionPane.showMessageDialog(null, "Message sent!");
        }catch(IOException e){

```

```

        JOptionPane.showMessageDialog(null, "Failed to send: " + e.getMessage());
    }
}

case 1 -> JOptionPane.showMessageDialog(null, "Message Discarded!");
case 2 -> {
    try(FileWriter file = new FileWriter("drafts.json", true)){
        file.write(messageObj.toString() + "\n");
        JOptionPane.showMessageDialog(null, "Message saved to drafts.");
    }catch(IOException e){
        JOptionPane.showConfirmDialog(null, "Failed to save draft: " + e.getMessage());
    }
}

default -> JOptionPane.showMessageDialog(null, "Invalid choice!");
}

}
}

```

```

public void messagesMenu() throws FileNotFoundException{
    boolean inMessagesMenu = true;

```

```

while (inMessagesMenu){
    String[] messageOptions = {
        "1 - Display senders and recipients",
        "2 - Display longest message",
        "3 - Message ID: ",
        "4 - Recipient number: ",
        "5 - Hash number",

```

```

        "6 - Display message report",
        "7 - More options",
        "8 - Back"
    };

    String msgChoiceString = (String)JOptionPane.showInputDialog(
        null,
        "Choose an option: ",
        "Back to main Menu",
        JOptionPane.PLAIN_MESSAGE,
        null,
        messageOptions,
        messageOptions[0]
    );

    if(msgChoiceString == null)
        break;

    int msgChoice = Integer.parseInt(msgChoiceString.substring(0, 1));

    switch (msgChoice) {
        case 1 -> displaySendersAndRecipients();
        case 2 -> displayLongestMessage();
        case 3 -> searchMessageID();
        case 4 -> searchRecipientNumber();
        case 5 -> searchHashNumber();
        case 6 -> displayMessageReport();
        case 7 -> moreOptionsMenu();
        case 8 -> {
            feedbackPrompt();
            deleteMessagesOnExit();
            inMessagesMenu = false;
        }
        default -> JOptionPane.showMessageDialog(null, "Invalid choice");
    }

```

```
    }  
    }  
}
```

```
public void displaySendersAndRecipients(){  
    List<String> messages = loadMessagesFromJson();  
  
    if(messages.isEmpty()){  
        JOptionPane.showMessageDialog(null, "No messages sent!");  
    }  
  
    StringBuilder summary = new StringBuilder("Senders and Recipients:\n\n");  
  
    for (String msg : messages){  
        String from = extractBetween(msg, "From: " to " ");  
        String to = extractBetween(msg,"to ", " :");  
        summary.append("From: ").append(from).append(" To: ").append(to).append("\n");  
    }  
  
    JTextArea textArea = new JTextArea(summary.toString());  
    textArea.setEditable(false);  
    JScrollPane scrollPane = new JScrollPane(textArea);  
    scrollPane.setPreferredSize(new Dimension(500, 300));  
  
    JOptionPane.showMessageDialog(null, scrollPane, "Sender and Recipient list: ",  
JOptionPane.INFORMATION_MESSAGE);  
}
```

```
private String extractBetween(String text, String start, String end){  
    int startIndex = text.indexOf(start);  
    int endIndex = text.indexOf(end, startIndex + start.length());
```

```

        if(startIndex == -1 || endIndex == -1)
            return "N/A";

        return text.substring(startIndex + start.length(), endIndex).trim();
    }

    private void displayLongestMessage(){
        List<String> messages = loadMessagesFromJson();

        if(messages.isEmpty()){
            JOptionPane.showMessageDialog(null, "No messages sent!");
        }

        String longestMessage = "";
        for (String message : messages){
            if(message.length() > longestMessage.length()){
                longestMessage = message;
            }
        }

        if (longestMessage.trim().isEmpty()){
            JOptionPane.showMessageDialog(null, "Longest message is empty!");
        }

        JTextArea textArea = new JTextArea("Longest message: \n\n" + longestMessage);
        textArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(textArea);
        scrollPane.setPreferredSize(new Dimension(500, 200));

        JOptionPane.showMessageDialog(null, scrollPane, "Longest Message",
        JOptionPane.INFORMATION_MESSAGE);
    }

```



```

private void searchMessageID(){
    List<String> messages = loadMessagesFromJson();

    if(messages.isEmpty()){
        JOptionPane.showMessageDialog(null, "No messages sent!");
    }

    String input = JOptionPane.showInputDialog("Enter message ID " + (messages.size() - 1) + ":");

    if (input == null) return;

    try {int id = Integer.parseInt(input);
        if (id < 0 || id >= messages.size()){
            JOptionPane.showMessageDialog(null, "Invalid message ID. Try again!");
            return;
        }

        String message = messages.get(id);
        String to = extractBetween(message, "to ", ":");
        String content = message.substring(message.indexOf(":") + 1).trim();

        JOptionPane.showMessageDialog(null,
            "Message ID: " + id + "\nRecipient: " + to + "\nMessage: " + content, "Message details",
            JOptionPane.INFORMATION_MESSAGE
        );
    }
    catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Please enter a valid message ID!");
    }
}

public void searchRecipientNumber(){
    List<String> messages = loadMessagesFromJson();

```

```

if(messages.isEmpty()){
    JOptionPane.showMessageDialog(null, "No messages sent!");
}

String number = JOptionPane.showInputDialog("Enter recipient number: ");

if(number == null || number.trim().isEmpty()) return;

StringBuilder result = new StringBuilder();

for (String msg : messages){
    String to = extractBetween(msg, "to ", ":");
    if(to.equalsIgnoreCase(number.trim())){
        result.append(msg).append("\n");
    }
}

if (result.length() == 0){
    JOptionPane.showMessageDialog(null, "No messages for: " + number);
}

JTextArea textArea = new JTextArea("Messages sent to: " + number + ":\n\n" +
result.toString());

textArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(textArea);
scrollPane.setPreferredSize(new Dimension(500, 300));

JOptionPane.showMessageDialog(null, scrollPane, "Search results",
JOptionPane.INFORMATION_MESSAGE);
}

private void searchHashNumber(){
    List<String> messages = loadMessagesFromJson();

```

```

if(messages.isEmpty()){
    JOptionPane.showMessageDialog(null, "No messages sent!");
}

String input = JOptionPane.showInputDialog("Enter message Hash: ");

if(input == null || input.trim().isEmpty()){
    try{
        int hashFind = Integer.parseInt(input);
        String messageFound = null;

        for (String msg : messages){
            if(msg.hashCode() == hashFind){
                messageFound = msg;
                break;
            }
        }
        if (messageFound == null){
            JOptionPane.showMessageDialog(null, "No message found using this Hash!");
        }

        int option = JOptionPane.showConfirmDialog(null, "Message: \n\n" + messageFound +
"\n\n Do you want to delete the message?", "Delete",
        JOptionPane.YES_NO_OPTION);
        if (option == JOptionPane.YES_OPTION){
            messages.remove(messageFound);
            JOptionPane.showMessageDialog(null, "Deleted");
        }
    }catch(NumberFormatException e){
        JOptionPane.showMessageDialog(null, "Invalid Hash entered, retry");
    }
}

```

```
}
```

```
}
```

```
public void displayMessageReport(){
```

```
    List<String> messages = loadMessagesFromJson();
```

```
    if(messages.isEmpty()){
```

```
        JOptionPane.showMessageDialog(null, "No messages sent!");
```

```
    }
```

```
    StringBuilder report = new StringBuilder("Full sent message report: \n\n");
```

```
    for (int i = 0; i<messages.size(); i++){
```

```
        String msg = messages.get(i);
```

```
        String from = extractBetween(msg, "From: ", " to ");
```

```
        String to = extractBetween(msg, "to ", " ");
```

```
        String content = msg.substring(msg.indexOf(":") + 1).trim();
```

```
        report.append("Message ID: ").append(i).append("\n");
```

```
        report.append("Hash: ").append(msg.hashCode()).append("\n");
```

```
        report.append("From: ").append(from).append("\n");
```

```
        report.append("To: ").append(to).append("\n");
```

```
        report.append("Message: ").append(content).append("\n");
```

```
        report.append("-----\n");
```

```
    }
```

```
    JTextArea textArea = new JTextArea(report.toString());
```

```
    textArea.setEditable(false);
```

```
    JScrollPane scrollPane = new JScrollPane(textArea);
```

```
    scrollPane.setPreferredSize(new Dimension(500, 400));
```

```
        JOptionPane.showMessageDialog(null, scrollPane, "Message report!",
JOptionPane.INFORMATION_MESSAGE);
    }
}
```

```
public void moreOptionsMenu() throws FileNotFoundException{
```

```
    boolean inMoreOptionsMenu = true;
```

```
    while(inMoreOptionsMenu){
```

```
        String [] moreOptions = {
```

```
            "1 - List sent messages",
```

```
            "2 - List all disregarded messages",
```

```
            "3 - List all stored messages",
```

```
            "4 - List all message Hashes",
```

```
            "5 - List all message ID's",
```

```
            "6 - Back"
```

```
        };
```

```
        String choiceStr = (String) JOptionPane.showInputDialog(
```

```
        null, "Choose option: ",
```

```
        "More Options",
```

```
        JOptionPane.INFORMATION_MESSAGE, null,
```

```
        moreOptions,
```

```
        moreOptions[0]
```

```
        );
```

```
        if (choiceStr == null)
```

```
            break;
```

```
        int choice = Integer.parseInt(choiceStr.substring(0, 1));
```

```
        switch (choice){
```

```

        case 1 -> listSentMessages();
        case 2 -> listDisregardedMessages();
        case 3 -> listStoredMessages();
        case 4 -> listMessageHashes();
        case 5 -> listMessageIDs();
        case 6 -> {
            feedbackPrompt();
            deleteMessagesOnExit();
            inMoreOptionsMenu = false;
        }
        default -> JOptionPane.showMessageDialog(null, "Invalid option,retry!");
    }
}

}

```

```

public void listSentMessages(){
    List<String> messages = loadMessagesFromJson();

    if(messages.isEmpty()){
        JOptionPane.showMessageDialog(null, "No messages sent!");
    }

    StringBuilder arrayDisplay = new StringBuilder("Sent messages: \n\n");

    for (int i = 0; i < messages.size(); i++){
        arrayDisplay.append("[").append(i).append("]").append(messages.get(i)).append("\n");
    }

    JTextArea textArea = new JTextArea(arrayDisplay.toString());
    textArea.setEditable(false);
}

```

```
JScrollPane scrollPane = new JScrollPane(textArea);  
scrollPane.setPreferredSize(new Dimension(500, 300));
```

```
OptionPane.showMessageDialog(null, scrollPane, "Sent messages",  
OptionPane.INFORMATION_MESSAGE);  
}
```

```
public void listDisregardedMessages(){  
    List<String> messages = loadMessagesFromJson();
```

```
    if(messages.isEmpty()){  
        JOptionPane.showMessageDialog(null, "No disregarded messages!");  
    }
```

```
    StringBuilder disregarded = new StringBuilder("Disregarded messages: \n\n");
```

```
    for (int i = 0; i < disregardedMessages.size(); i++){
```

```
        disregarded.append("[").append(i).append("]").append(disregardedMessages.get(i)).append("\n  
");  
    }
```

```
    JTextArea textArea = new JTextArea(disregarded.toString());  
    textArea.setEditable(false);  
    JScrollPane scrollPane = new JScrollPane(textArea);  
    scrollPane.setPreferredSize(new Dimension(500, 300));
```

```
    JOptionPane.showMessageDialog(null, scrollPane, "Disregarded messages.",  
OptionPane.INFORMATION_MESSAGE);  
}
```

```

public void listMessageHashes(){
    List<String> messages = loadMessagesFromJson();

    if(messages.isEmpty()){
        JOptionPane.showMessageDialog(null, "No messages!");
    }

    StringBuilder output = new StringBuilder("Message hashes: \n\n");

    for(int i = 0; i < messages.size(); i++){
        String message = messages.get(i);
        int hash = message.hashCode();
        output.append("[").append(i).append("]").append(hash).append("\n");
    }

    JTextArea textArea = new JTextArea(output.toString());
    textArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(textArea);
    scrollPane.setPreferredSize(new Dimension(500, 300));

    JOptionPane.showMessageDialog(null, scrollPane, "Message Hashes",
JOptionPane.INFORMATION_MESSAGE);
    }

public void listMessageIDs(){
    List<String> messages = loadMessagesFromJson();

```



```

if(messages.isEmpty()){
    JOptionPane.showMessageDialog(null, "No messages!");
}

StringBuilder output = new StringBuilder("Message ID's: \n\n");

for(int i = 0; i < messages.size(); i++){
    output.append("Message ID's: ").append(i).append("\n");
}

JTextArea textArea = new JTextArea(output.toString());
textArea.setEditable(false);
JScrollPane scrollPane = new JScrollPane(textArea);
scrollPane.setPreferredSize(new Dimension(500, 300));

JOptionPane.showMessageDialog(null,scrollPane, "Message ID's",
JOptionPane.INFORMATION_MESSAGE);
}

```

```

@SuppressWarnings("empty-statement")

```

```

private String checkMessageID(){
    Random rand = new Random();
    StringBuilder id = new StringBuilder();
    for (int i = 0; i < 10; i++){
        id.append(rand.nextInt(10));
    }
    return id.toString();
}

```

```

}

private String createMessageHash(String messageId, int messageNumber, String
messageContent){

    String firstTwoDigits = messageId.length() >= 2 ? messageId.substring(0, 2) : messageId;

    String[] words = messageContent.trim().split("\\s+");
    String firstWord = words.length > 0 ? words[0] : "";
    String lastWord = words.length > 1 ? words[words.length - 1] : firstWord;

    return firstTwoDigits + ":" + messageNumber + ":" + firstWord + lastWord;
}

private void printMessage(){

    List<String> messages = loadMessagesFromJson();

    if (messages.isEmpty()){
        JOptionPane.showMessageDialog(null, "No messages sent");
        return;
    }

    StringBuilder messageList = new StringBuilder("Sent messages:\n\n");

    for (String message : messages){
        messageList.append(message).append("\n\n");
    }

    JTextArea textArea = new JTextArea(messageList.toString());
    textArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(textArea);
    scrollPane.setPreferredSize(new Dimension(500, 400));

    JOptionPane.showMessageDialog(null, scrollPane, "Chat history",
JOptionPane.INFORMATION_MESSAGE);

```

```
}
```

```
public void listStoredMessages() throws FileNotFoundException{
```

```
    JSONArray messages = new JSONArray();
```

```
    try(BufferedReader reader = new BufferedReader(new FileReader("messages.json"))){
```

```
        String line;
```

```
        while((line = reader.readLine()) != null){
```

```
            JSONObject obj = new JSONObject(line);
```

```
            messages.put(obj);
```

```
        }
```

```
    }catch(FileNotFoundException e){
```

```
        JOptionPane.showMessageDialog(null, "JSON file not found!");
```

```
    }catch(IOException e){
```

```
        JOptionPane.showMessageDialog(null, "Error reading file!" + e.getMessage());
```

```
    }
```

```
    if(messages.isEmpty()){
```

```
        JOptionPane.showMessageDialog(null, "No stored messages!");
```

```
    }
```

```
    StringBuilder output = new StringBuilder("Stored Messages: \n\n");
```

```
    for(int i = 0; i<messages.length();i++){
```

```
        JSONObject msg = messages.getJSONObject(i);
```

```
        output.append("[").append(i).append("]");
```

```
        output.append("From: ").append(msg.optString("from")).append(" ");
```

```
        output.append("To: ").append(msg.optString("to")).append("\n");
```

```

        output.append("Message: ").append(msg.optString("message")).append("\n");
        output.append("-----\n");

    }

    JTextArea textArea = new JTextArea(output.toString());
    textArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(textArea);
    scrollPane.setPreferredSize(new Dimension(500, 300));

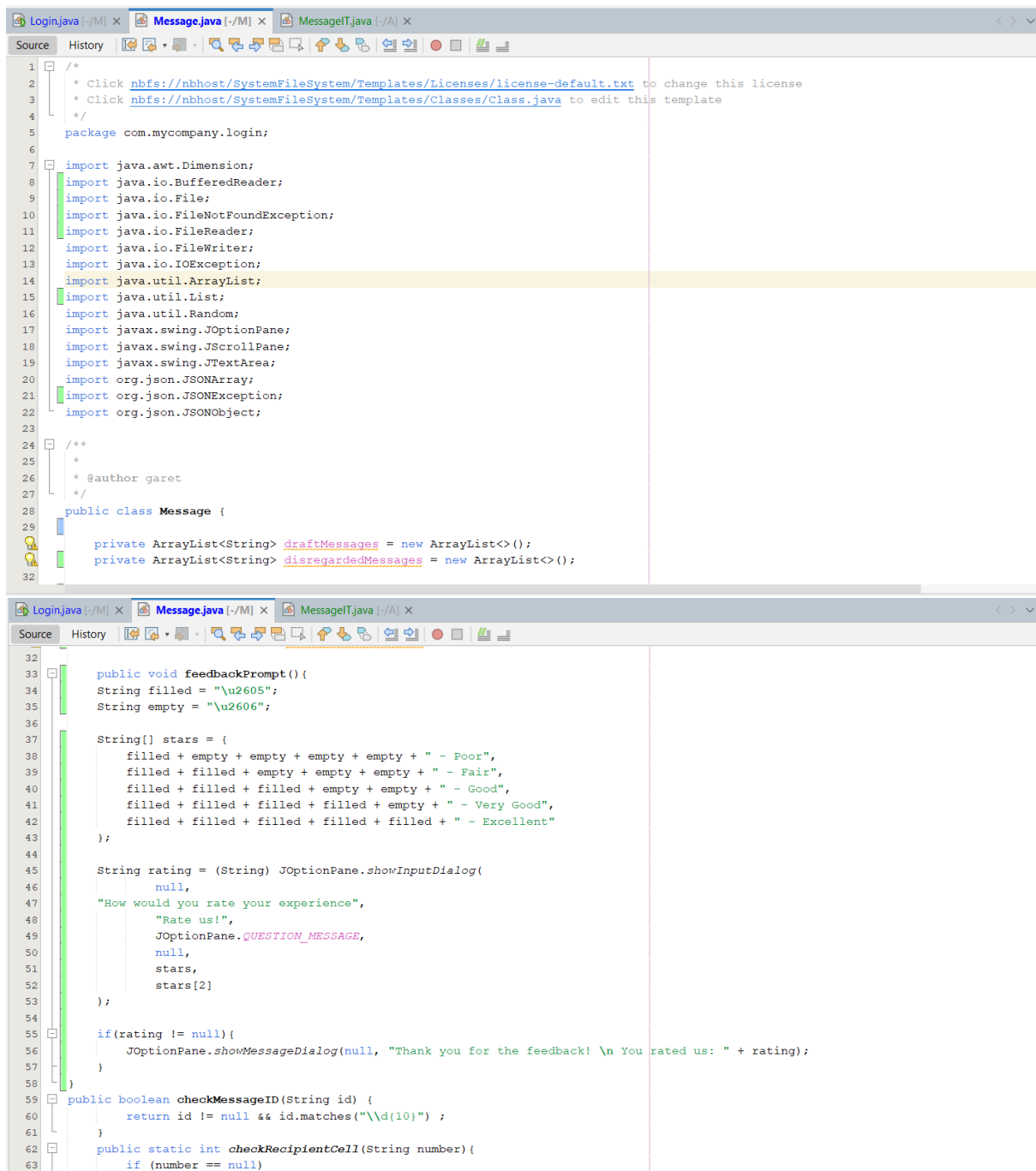
    JOptionPane.showMessageDialog(null, scrollPane, "Stored Messages",
JOptionPane.INFORMATION_MESSAGE);
    }

private void deleteMessagesOnExit(){
    File messagesFile = new File("messages.json");
    if(messagesFile.exists()){
        if(messagesFile.delete()){
            JOptionPane.showMessageDialog(null, "Messages deleted!");
        }else{
            JOptionPane.showMessageDialog(null, "Deletion failed!");
        }
    }
}

}

```

Screenshots of code:



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package com.mycompany.login;
6
7  import java.awt.Dimension;
8  import java.io.BufferedReader;
9  import java.io.File;
10 import java.io.FileNotFoundException;
11 import java.io.FileReader;
12 import java.io.FileWriter;
13 import java.io.IOException;
14 import java.util.ArrayList;
15 import java.util.List;
16 import java.util.Random;
17 import javax.swing.JOptionPane;
18 import javax.swing.JScrollPane;
19 import javax.swing.JTextArea;
20 import org.json.JSONArray;
21 import org.json.JSONException;
22 import org.json.JSONObject;
23
24 /**
25 *
26 * @author garet
27 */
28 public class Message {
29
30     private ArrayList<String> draftMessages = new ArrayList<>();
31     private ArrayList<String> disregardedMessages = new ArrayList<>();
32
33     public void feedbackPrompt() {
34         String filled = "\u2605";
35         String empty = "\u2606";
36
37         String[] stars = {
38             filled + empty + empty + empty + empty + " - Poor",
39             filled + filled + empty + empty + empty + " - Fair",
40             filled + filled + filled + empty + empty + " - Good",
41             filled + filled + filled + filled + empty + " - Very Good",
42             filled + filled + filled + filled + filled + " - Excellent"
43         };
44
45         String rating = (String) JOptionPane.showInputDialog(
46             null,
47             "How would you rate your experience",
48             "Rate us!",
49             JOptionPane.QUESTION_MESSAGE,
50             null,
51             stars,
52             stars[2]
53         );
54
55         if(rating != null){
56             JOptionPane.showMessageDialog(null, "Thank you for the feedback! \n You rated us: " + rating);
57         }
58     }
59     public boolean checkMessageID(String id) {
60         return id != null && id.matches("\\d{10}");
61     }
62     public static int checkRecipientCell(String number) {
63         if (number == null)
```

```
62 public static int checkRecipientCell(String number){
63     if (number == null)
64         return 0;
65     if (number.startsWith("+27") && number.length() == 12 && number.substring(3).matches("\\d{9}")){
66         return 1;
67     }return 0;
68 }
69 public void startChat(){
70     JOptionPane.showMessageDialog(null, "Welcome to QuickChat");
71 }
72 }
73 public void run(Login login) throws FileNotFoundException {
74
75     JOptionPane.showMessageDialog(null, "Login Successful. Welcome, " + Login.username + "!");
76     boolean running = true;
77     while (running){
78         String[] options = {
79             "1 - Send Message",
80             "2 - Show recently sent messages",
81             "3 - Messages",
82             "4 - Quit"
83         };
84         String choiceString = (String) JOptionPane.showInputDialog(null,
85             "Choose an option:",
86             "Main menu",
87             JOptionPane.PLAIN_MESSAGE,
88             null,
89             options,
90             options[0]
91         );
92         if (choiceString == null){
93             feedbackPrompt();
94
95             feedbackPrompt();
96             System.exit(0);
97             continue;
98         }
99
100         int choice = Integer.parseInt(choiceString.substring(0, 1));
101         switch (choice){
102             case 1 -> sendMessage(Login.username);
103             case 2 -> JOptionPane.showMessageDialog(null, "Coming soon.");
104             case 3 -> messagesMenu();
105             case 4 -> {
106                 JOptionPane.showMessageDialog(null, "Session ended.");
107                 deleteMessagesOnExit();
108                 feedbackPrompt();
109                 running = false;
110             }
111             default -> JOptionPane.showMessageDialog(null, "Invalid option");
112         }
113     }
114 }
115
116 private List<String> loadMessagesFromJson(){
117     List<String> messages = new ArrayList<>();
118
119     try{
120         File file = new File("messages.json");
121         if(!file.exists()){
122             file.createNewFile();
123             return messages;
124         }
125     }
```

```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History
125 try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
126     String line;
127     while ((line = reader.readLine()) != null) {
128
129         try {
130             JSONObject obj = new JSONObject(line);
131             String from = obj.optString("from");
132             String to = obj.optString("to");
133             String content = obj.optString("message");
134             messages.add("From: " + from + " to " + to + ": " + content);
135         } catch (JSONException je) {
136             System.out.println("Error reading! " + je.getMessage());
137         }
138     }
139 } catch (IOException e) {
140     JOptionPane.showMessageDialog(null, "Error reading file: " + e.getMessage());
141 }
142 return messages;
143 }
144
145
146 public void sendMessage(String username) {
147
148     String recipient;
149     while(true) {
150         recipient = JOptionPane.showInputDialog("Enter recipients number: ");
151
152         if (recipient == null) return;
153
154         if (recipient.matches("\\+27\\d{9}")) {
155             break;
156         }
157     }
158 }
```

```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History
157 else {
158     JOptionPane.showMessageDialog(null, "Recipient cannot be empty.");
159     return;
160 }
161
162 int messageCount = 0;
163 try {
164     String countInput = JOptionPane.showInputDialog("How many messages would you like to send?");
165     if (countInput == null)
166
167         System.exit(0);
168
169     messageCount = Integer.parseInt(countInput);
170
171     if (messageCount <= 0) {
172         JOptionPane.showMessageDialog(null, "Enter number.");
173         return;
174     }
175 } catch (NumberFormatException e) {
176     JOptionPane.showMessageDialog(null, "Invalid number");
177     return;
178 }
179
180 for (int i = 1; i <= messageCount; i++) {
181     String message = JOptionPane.showInputDialog("Enter the message: ");
182     if (message == null || message.trim().isEmpty()) {
183         disregardedMessages.add("Disregarded message!");
184         return;
185     }
186 }
187
188 String messageId = checkMessageID();
189 String messageHash = createMessageHash(messageId, i, message);
190 }
```

```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History
189 String fullMessage = "[Message ID: " + messageId + "]\nHash: " + messageHash +
190 "\nFrom: " + username + " to " + recipient + ": " + message;
191 String[] options = {"Send", "Disregard", "Store for later" };
192 int choice = JOptionPane.showOptionDialog(
193     null,
194     "What would you like to do?" + fullMessage,
195     "Confirm Message",
196     JOptionPane.DEFAULT_OPTION,
197     JOptionPane.QUESTION_MESSAGE,
198     null,
199     options,
200     options[0]
201 );
202 if(options == null){
203     JOptionPane.showMessageDialog(null, "Cancelled!");
204 }
205
206
207 JSONObject messageObj = new JSONObject();
208 messageObj.put("Message ID", messageId);
209 messageObj.put("Message Hash", messageHash);
210 messageObj.put("from", username);
211 messageObj.put("to", recipient);
212 messageObj.put("message", message);
213
214 switch(choice){
215     case 0 -> {
216         try(FileWriter file = new FileWriter("messages.json", true)){
217             file.write(messageObj.toString() + "\n");
218             JOptionPane.showMessageDialog(null, "Message sent!");
219         }catch(IOException e){
220             JOptionPane.showMessageDialog(null, "Failed to send: " + e.getMessage());
221         }
222     }
223     case 1 -> JOptionPane.showMessageDialog(null, "Message Discarded!");
224     case 2 -> {
225         try(FileWriter file = new FileWriter("drafts.json", true)){
226             file.write(messageObj.toString() + "\n");
227             JOptionPane.showMessageDialog(null, "Message saved to drafts.");
228         }catch(IOException e){
229             JOptionPane.showConfirmDialog(null, "Failed to save draft: " + e.getMessage());
230         }
231     }
232     default -> JOptionPane.showMessageDialog(null, "Invalid choice!");
233 }
234
235 }
236 }
237
238
239 public void messagesMenu() throws FileNotFoundException{
240     boolean inMessagesMenu = true;
241
242
243     while (inMessagesMenu){
244         String[] messageOptions = {
245             "1 - Display senders and recipients",
246             "2 - Display longest message",
247             "3 - Message ID: ",
248             "4 - Recipient number: ",
249             "5 - Hash number",
250             "6 - Display message report",
251             "7 - More options",
252         }
```



```

253         "8 - Back"
254     };
255     String msgChoiceString = (String)JOptionPane.showInputDialog(
256     null,
257         "Choose an option: ",
258         "Back to main Menu",
259         JOptionPane.PLAIN_MESSAGE,
260         null,
261         messageOptions,
262         messageOptions[0]
263     );
264     if(msgChoiceString == null)
265         break;
266     int msgChoice = Integer.parseInt(msgChoiceString.substring(0, 1));
267
268     switch (msgChoice) {
269         case 1 -> displaySendersAndRecipients();
270         case 2 -> displayLongestMessage();
271         case 3 -> searchMessageID();
272         case 4 -> searchRecipientNumber();
273         case 5 -> searchHashNumber();
274         case 6 -> displayMessageReport();
275         case 7 -> moreOptionsMenu();
276         case 8 -> {
277             feedbackPrompt();
278             deleteMessagesOnExit();
279             inMessagesMenu = false;
280         }
281         default -> JOptionPane.showMessageDialog(null, "Invalid choice");
282     }
283 }
284 }

```

```

285
286 public void displaySendersAndRecipients(){
287     List<String> messages = loadMessagesFromJson();
288
289     if(messages.isEmpty()){
290         JOptionPane.showMessageDialog(null, "No messages sent!");
291     }
292     StringBuilder summary = new StringBuilder("Senders and Recipients:\n\n");
293
294     for (String msg : messages){
295         String from = extractBetween(msg, "From: ", " to ");
296         String to = extractBetween(msg, "to ", ":");
297         summary.append("From: ").append(from).append(" To: ").append(to).append("\n");
298     }
299
300     JTextArea textArea = new JTextArea(summary.toString());
301     textArea.setEditable(false);
302     JScrollPane scrollPane = new JScrollPane(textArea);
303     scrollPane.setPreferredSize(new Dimension(500, 300));
304
305     JOptionPane.showMessageDialog(null, scrollPane, "Sender and Recipient list: ", JOptionPane.INFORMATION_MESSAGE);
306 }
307
308
309 private String extractBetween(String text, String start, String end){
310     int startIndex = text.indexOf(start);
311     int endIndex = text.indexOf(end, startIndex + start.length());
312     if(startIndex == -1 || endIndex == -1)
313         return "N/A";
314     return text.substring(startIndex + start.length(), endIndex).trim();
315 }
316

```

```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History

316
317 private void displayLongestMessage() {
318     List<String> messages = loadMessagesFromJson();
319
320     if(messages.isEmpty()){
321         JOptionPane.showMessageDialog(null, "No messages sent!");
322     }
323
324     String longestMessage = "";
325     for (String message : messages){
326         if(message.length() > longestMessage.length()){
327             longestMessage = message;
328         }
329     }
330     if (longestMessage.trim().isEmpty()){
331         JOptionPane.showMessageDialog(null, "Longest message is empty!");
332     }
333     JTextArea textArea = new JTextArea("Longest message: \n\n" + longestMessage);
334     textArea.setEditable(false);
335     JScrollPane scrollPane = new JScrollPane(textArea);
336     scrollPane.setPreferredSize(new Dimension(500, 200));
337
338     JOptionPane.showMessageDialog(null, scrollPane, "Longest Message", JOptionPane.INFORMATION_MESSAGE);
339 }
340
341
342 private void searchMessageID() {
343     List<String> messages = loadMessagesFromJson();
344
345     if(messages.isEmpty()){
346         JOptionPane.showMessageDialog(null, "No messages sent!");
347     }
348
349     String input = JOptionPane.showInputDialog("Enter message ID " + (messages.size() - 1) + ": ");
350
351     if (input == null) return;
352     try {int id = Integer.parseInt(input);
353         if (id < 0 || id >= messages.size()){
354             JOptionPane.showMessageDialog(null, "Invalid message ID. Try again!");
355             return;
356         }
357         String message = messages.get(id);
358         String to = extractBetween(message, "to ", ":");
359         String content = message.substring(message.indexOf(":") + 1).trim();
360
361         JOptionPane.showMessageDialog(null,
362             "Message ID: " + id + "\nRecipient: " + to + "\nMessage: " + content, "Message details",
363             JOptionPane.INFORMATION_MESSAGE);
364     }
365     catch (NumberFormatException e) {
366         JOptionPane.showMessageDialog(null, "Please enter a valid message ID!");
367     }
368 }
369
370 public void searchRecipientNumber() {
371     List<String> messages = loadMessagesFromJson();
372
373     if(messages.isEmpty()){
374         JOptionPane.showMessageDialog(null, "No messages sent!");
375     }
376     String number = JOptionPane.showInputDialog("Enter recipient number: ");
377
378     if(number == null || number.trim().isEmpty()) return;
379 }
```

```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History
380     StringBuilder result = new StringBuilder();
381
382     for (String msg : messages){
383         String to = extractBetween(msg, "to ", ":");
384         if(to.equalsIgnoreCase(number.trim())){
385             result.append(msg).append("\n");
386         }
387     }
388     if (result.length() == 0){
389         JOptionPane.showMessageDialog(null, "No messages for: " + number);
390     }
391
392     JTextArea textArea = new JTextArea("Messages sent to: " + number + ":\n\n" + result.toString());
393     textArea.setEditable(false);
394     JScrollPane scrollPane = new JScrollPane(textArea);
395     scrollPane.setPreferredSize(new Dimension(500, 300));
396
397
398     JOptionPane.showMessageDialog(null, scrollPane, "Search results", JOptionPane.INFORMATION_MESSAGE);
399 }
400 private void searchHashNumber(){
401     List<String> messages = loadMessagesFromJson();
402
403     if(messages.isEmpty()){
404         JOptionPane.showMessageDialog(null, "No messages sent!");
405     }
406
407     String input = JOptionPane.showInputDialog("Enter message Hash: ");
408
409     if(input == null || input.trim().isEmpty()){
410         try{
411             int hashFind = Integer.parseInt(input);
412
413             int hashFind = Integer.parseInt(input);
414             String messageFound = null;
415
416             for (String msg : messages){
417                 if(msg.hashCode() == hashFind){
418                     messageFound = msg;
419                     break;
420                 }
421             }
422             if (messageFound == null){
423                 JOptionPane.showMessageDialog(null, "No message found using this Hash!");
424             }
425             int option = JOptionPane.showConfirmDialog(null, "Message: \n\n" + messageFound + "\n\n Do you want to delete the mes
426                 JOptionPane.YES_NO_OPTION);
427             if (option == JOptionPane.YES_OPTION){
428                 messages.remove(messageFound);
429                 JOptionPane.showMessageDialog(null, "Deleted");
430             }
431             catch (NumberFormatException e){
432                 JOptionPane.showMessageDialog(null, "Invalid Hash entered, retry");
433             }
434         }
435     }
436     public void displayMessageReport(){
437         List<String> messages = loadMessagesFromJson();
438
439         if(messages.isEmpty()){
440             JOptionPane.showMessageDialog(null, "No messages sent!");
441         }
442         StringBuilder report = new StringBuilder("Full sent message report: \n\n");
```

```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History
443     for (int i = 0; i < messages.size(); i++) {
444         String msg = messages.get(i);
445         String from = extractBetween(msg, "From: ", " to ");
446         String to = extractBetween(msg, "to ", ":");
447         String content = msg.substring(msg.indexOf(":") + 1).trim();
448
449         report.append("Message ID: ").append(i).append("\n");
450         report.append("Hash: ").append(msg.hashCode()).append("\n");
451         report.append("From: ").append(from).append("\n");
452         report.append("To: ").append(to).append("\n");
453         report.append("Message: ").append(content).append("\n");
454         report.append("-----\n");
455     }
456     JTextArea textArea = new JTextArea(report.toString());
457     textArea.setEditable(false);
458     JScrollPane scrollPane = new JScrollPane(textArea);
459     scrollPane.setPreferredSize(new Dimension(500, 400));
460
461     JOptionPane.showMessageDialog(null, scrollPane, "Message report!", JOptionPane.INFORMATION_MESSAGE);
462 }
463
464 public void moreOptionsMenu() throws FileNotFoundException {
465     boolean inMoreOptionsMenu = true;
466
467     while(inMoreOptionsMenu) {
468         String [] moreOptions = {
469             "1 - List sent messages",
470             "2 - List all disregarded messages",
471             "3 - List all stored messages",
472             "4 - List all message Hashes",
473             "5 - List all message ID's",
474             "6 - Back"
475         };
476
477         String choiceStr = (String) JOptionPane.showInputDialog(
478             null, "Choose option: ",
479             "More Options",
480             JOptionPane.INFORMATION_MESSAGE, null,
481             moreOptions,
482             moreOptions[0]
483         );
484
485         if (choiceStr == null)
486             break;
487
488         int choice = Integer.parseInt(choiceStr.substring(0, 1));
489
490         switch (choice) {
491             case 1 -> listSentMessages();
492             case 2 -> listDisregardedMessages();
493             case 3 -> listStoredMessages();
494             case 4 -> listMessageHashes();
495             case 5 -> listMessageIDs();
496             case 6 -> {
497                 feedbackPrompt();
498                 deleteMessagesOnExit();
499                 inMoreOptionsMenu = false;
500             }
501             default -> JOptionPane.showMessageDialog(null, "Invalid option, retry!");
502         }
503     }
504 }
505
506 }
```

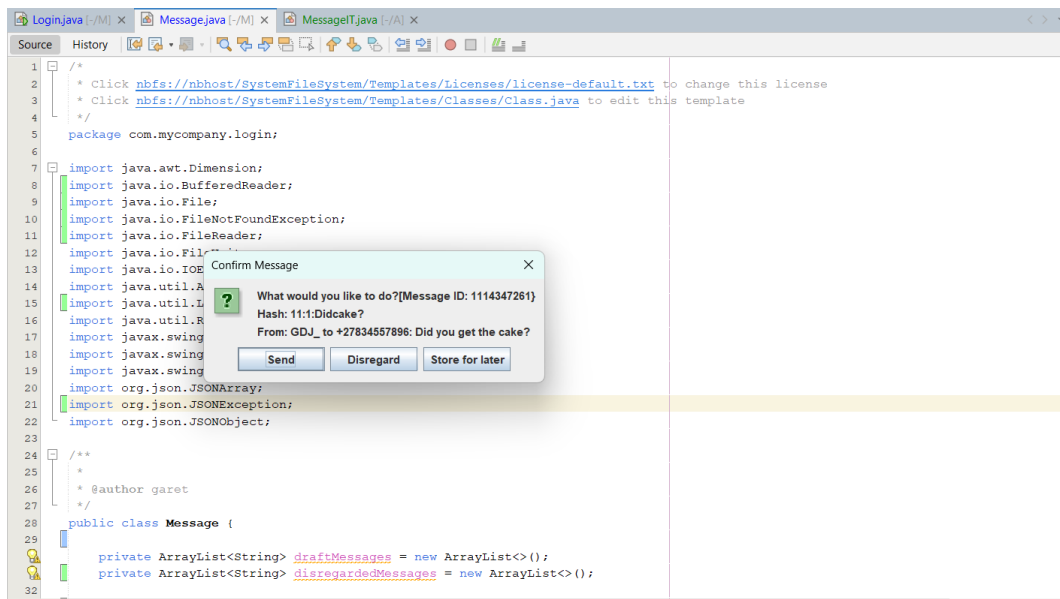
```
Login.java [-/M] x Message.java [-/M] x MessageT.java [-/A] x
Source History
507
508     public void listSentMessages() {
509         List<String> messages = loadMessagesFromJson();
510
511         if(messages.isEmpty()){
512             JOptionPane.showMessageDialog(null, "No messages sent!");
513         }
514
515         StringBuilder arrayDisplay = new StringBuilder("Sent messages: \n\n");
516
517         for (int i = 0; i < messages.size(); i++) {
518             arrayDisplay.append("[").append(i).append("]").append(messages.get(i)).append("\n");
519         }
520
521         JTextArea textArea = new JTextArea(arrayDisplay.toString());
522         textArea.setEditable(false);
523         JScrollPane scrollPane = new JScrollPane(textArea);
524         scrollPane.setPreferredSize(new Dimension(500, 300));
525
526
527         JOptionPane.showMessageDialog(null, scrollPane, "Sent messages", JOptionPane.INFORMATION_MESSAGE);
528     }
529
530     public void listDisregardedMessages() {
531         List<String> messages = loadMessagesFromJson();
532
533         if(messages.isEmpty()){
534             JOptionPane.showMessageDialog(null, "No disregarded messages!");
535         }
536
537         StringBuilder disregarded = new StringBuilder("Disregarded messages: \n\n");
538
539         for (int i = 0; i < disregardedMessages.size(); i++) {
540             disregarded.append("[").append(i).append("]").append(disregardedMessages.get(i)).append("\n");
541         }
542
543         JTextArea textArea = new JTextArea(disregarded.toString());
544         textArea.setEditable(false);
545         JScrollPane scrollPane = new JScrollPane(textArea);
546         scrollPane.setPreferredSize(new Dimension(500, 300));
547
548
549         JOptionPane.showMessageDialog(null, scrollPane, "Disregarded messages.", JOptionPane.INFORMATION_MESSAGE);
550     }
551
552
553
554     public void listMessageHashes() {
555         List<String> messages = loadMessagesFromJson();
556
557         if(messages.isEmpty()){
558             JOptionPane.showMessageDialog(null, "No messages!");
559         }
560
561         StringBuilder output = new StringBuilder("Message hashes: \n\n");
562
563         for(int i = 0; i < messages.size(); i++){
564             String message = messages.get(i);
565             int hash = message.hashCode();
566             output.append("[").append(i).append("]").append(hash).append("\n");
567         }
568
569         JTextArea textArea = new JTextArea(output.toString());
570         textArea.setEditable(false);
```

```
Login.java [-/M] x Message.java [-/M] x MessageIT.java [-/A] x
Source History
571 JScrollPane scrollPane = new JScrollPane(textArea);
572 scrollPane.setPreferredSize(new Dimension(500, 300));
573
574
575 JOptionPane.showMessageDialog(null, scrollPane, "Message Hashes", JOptionPane.INFORMATION_MESSAGE);
576 }
577
578 public void listMessageIDs() {
579     List<String> messages = loadMessagesFromJson();
580
581     if(messages.isEmpty()){
582         JOptionPane.showMessageDialog(null, "No messages!");
583     }
584
585     StringBuilder output = new StringBuilder("Message ID's: \n\n");
586
587     for(int i = 0; i < messages.size(); i++){
588         output.append("Message ID's: ").append(i).append("\n");
589     }
590     JTextArea textArea = new JTextArea(output.toString());
591     textArea.setEditable(false);
592     JScrollPane scrollPane = new JScrollPane(textArea);
593     scrollPane.setPreferredSize(new Dimension(500, 300));
594
595
596     JOptionPane.showMessageDialog(null,scrollPane, "Message ID's", JOptionPane.INFORMATION_MESSAGE);
597 }
598
599
600
601 @SuppressWarnings("empty-statement")
602
Login.java [-/M] x Message.java [-/M] x MessageIT.java [-/A] x
Source History
602
603
604 private String checkMessageID() {
605     Random rand = new Random();
606     StringBuilder id = new StringBuilder();
607     for (int i = 0; i < 10; i++) {
608         id.append(rand.nextInt(10));
609     }
610     return id.toString();
611 }
612 private String createMessageHash(String messageId, int messageNumber, String messageContent){
613     String firstTwoDigits = messageId.length() >= 2 ? messageId.substring(0, 2) : messageId;
614
615     String[] words = messageContent.trim().split("\\s+");
616     String firstWord = words.length > 0 ? words[0] : "";
617     String lastWord = words.length > 1 ? words[words.length - 1] : firstWord;
618
619     return firstTwoDigits + ":" + messageNumber + ":" + firstWord + lastWord;
620 }
621 private void printMessage() {
622     List<String> messages = loadMessagesFromJson();
623
624     if (messages.isEmpty()) {
625         JOptionPane.showMessageDialog(null, "No messages sent");
626         return;
627     }
628     StringBuilder messageList = new StringBuilder("Sent messages:\n\n");
629
630     for (String message : messages) {
631         messageList.append(message).append("\n\n");
632     }
633 }
```

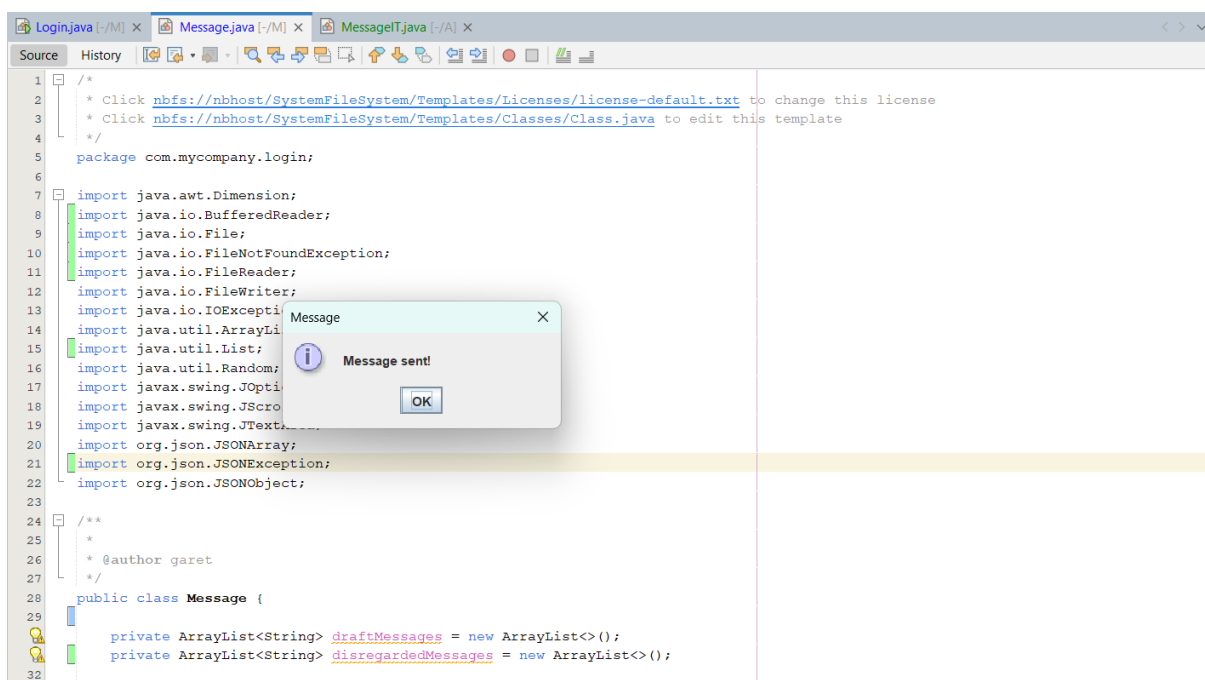
```
634      JTextArea textArea = new JTextArea(messageList.toString());
635      textArea.setEditable(false);
636      JScrollPane scrollPane = new JScrollPane(textArea);
637      scrollPane.setPreferredSize(new Dimension(500, 400));
638
639
640      JOptionPane.showMessageDialog(null, scrollPane, "Chat history", JOptionPane.INFORMATION_MESSAGE);
641  }
642
643
644
645  public void listStoredMessages() throws FileNotFoundException{
646
647      JSONArray messages = new JSONArray();
648
649      try(BufferedReader reader = new BufferedReader(new FileReader("messages.json"))){
650          String line;
651          while((line = reader.readLine()) != null){
652              JSONObject obj = new JSONObject(line);
653              messages.put(obj);
654          }
655      }catch(FileNotFoundException e){
656          JOptionPane.showMessageDialog(null, "JSON file not found!");
657      }catch(IOException e){
658          JOptionPane.showMessageDialog(null, "Error reading file!" + e.getMessage());
659      }
660
661      if(messages.isEmpty()){
662          JOptionPane.showMessageDialog(null, "No stored messages!");
663      }
664
665      StringBuilder output = new StringBuilder("Stored Messages: \n\n");
666
667      for(int i = 0; i<messages.length();i++){
668          JSONObject msg = messages.getJSONObject(i);
669          output.append("[").append(i).append("]");
670          output.append("From: ").append(msg.optString("from")).append(" ");
671          output.append("To: ").append(msg.optString("to")).append("\n");
672          output.append("Message: ").append(msg.optString("message")).append("\n");
673          output.append("-----\n");
674      }
675
676      JTextArea textArea = new JTextArea(output.toString());
677      textArea.setEditable(false);
678      JScrollPane scrollPane = new JScrollPane(textArea);
679      scrollPane.setPreferredSize(new Dimension(500, 300));
680
681
682      JOptionPane.showMessageDialog(null, scrollPane, "Stored Messages", JOptionPane.INFORMATION_MESSAGE);
683  }
684
685
686  private void deleteMessagesOnExit() {
687      File messagesFile = new File("messages.json");
688      if(messagesFile.exists()){
689          if(messagesFile.delete()){
690              JOptionPane.showMessageDialog(null, "Messages deleted!");
691          }else{
692              JOptionPane.showMessageDialog(null, "Deletion failed!");
693          }
694      }
695  }
696  }
```

Screenshots of testing:

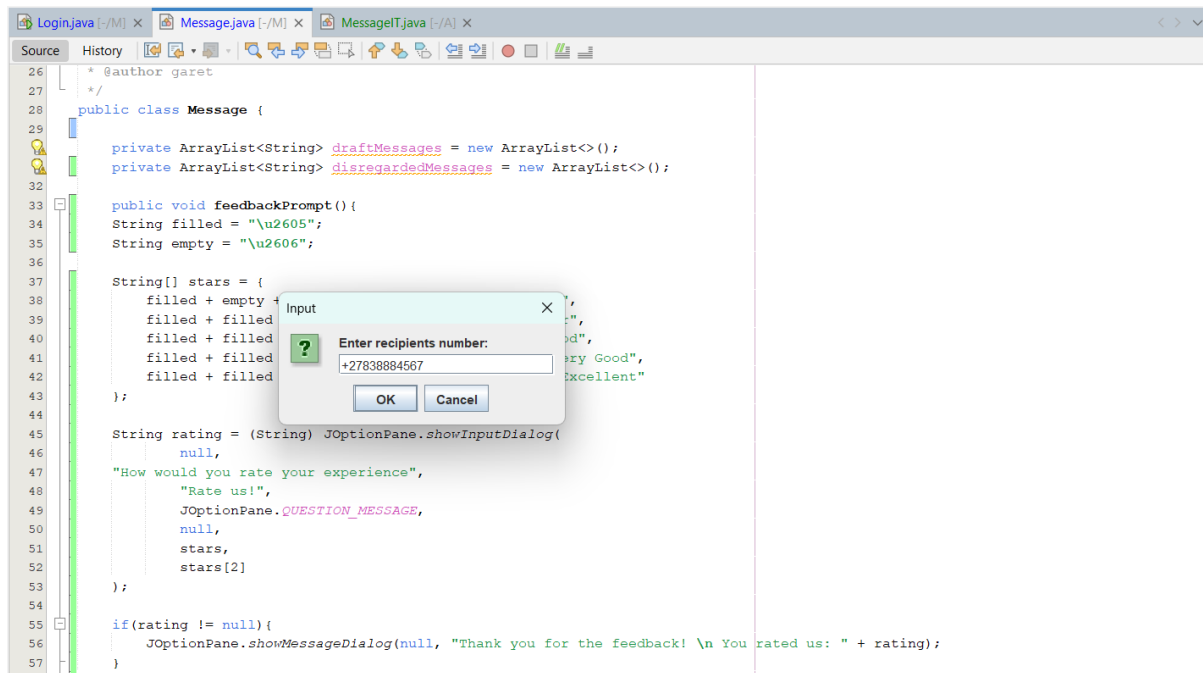
Test data 1_ confirmation of the message:



Test data 1_ message sent:



Test data 2_3_4_5_recipient number:

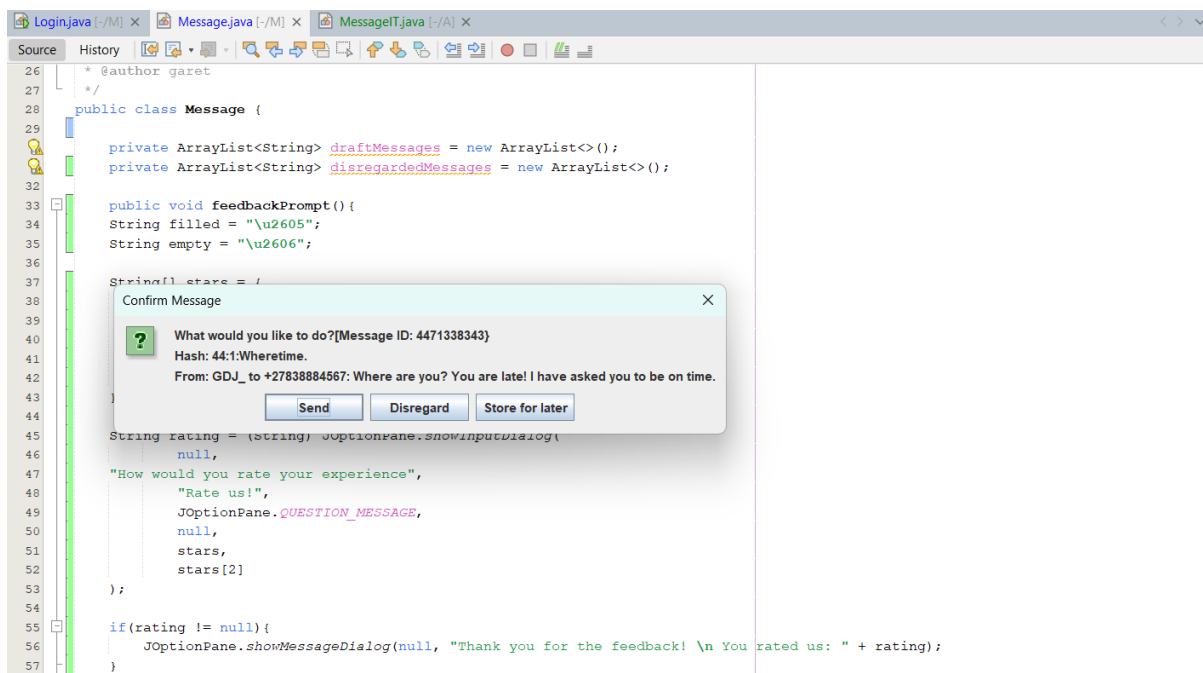


The screenshot shows an IDE with three tabs: Login.java, Message.java, and MessageIT.java. The Message.java file is open, showing the following code:

```
26  * @author garet
27  */
28  public class Message {
29
30      private ArrayList<String> draftMessages = new ArrayList<>();
31      private ArrayList<String> disregardedMessages = new ArrayList<>();
32
33      public void feedbackPrompt() {
34          String filled = "\u2605";
35          String empty = "\u2606";
36
37          String[] stars = {
38              filled + empty +
39              filled + filled +
40              filled + filled +
41              filled + filled +
42              filled + filled +
43          };
44
45          String rating = (String) JOptionPane.showInputDialog(
46              null,
47              "How would you rate your experience",
48              "Rate us!",
49              JOptionPane.QUESTION_MESSAGE,
50              null,
51              stars,
52              stars[2]
53          );
54
55          if (rating != null) {
56              JOptionPane.showMessageDialog(null, "Thank you for the feedback! \n You rated us: " + rating);
57          }
58      }
59  }
```

A dialog box titled "Input" is displayed over the code. It contains the text "Enter recipients number:" and a text field with the value "+27838884567". There are "OK" and "Cancel" buttons at the bottom.

Test data 2_message:

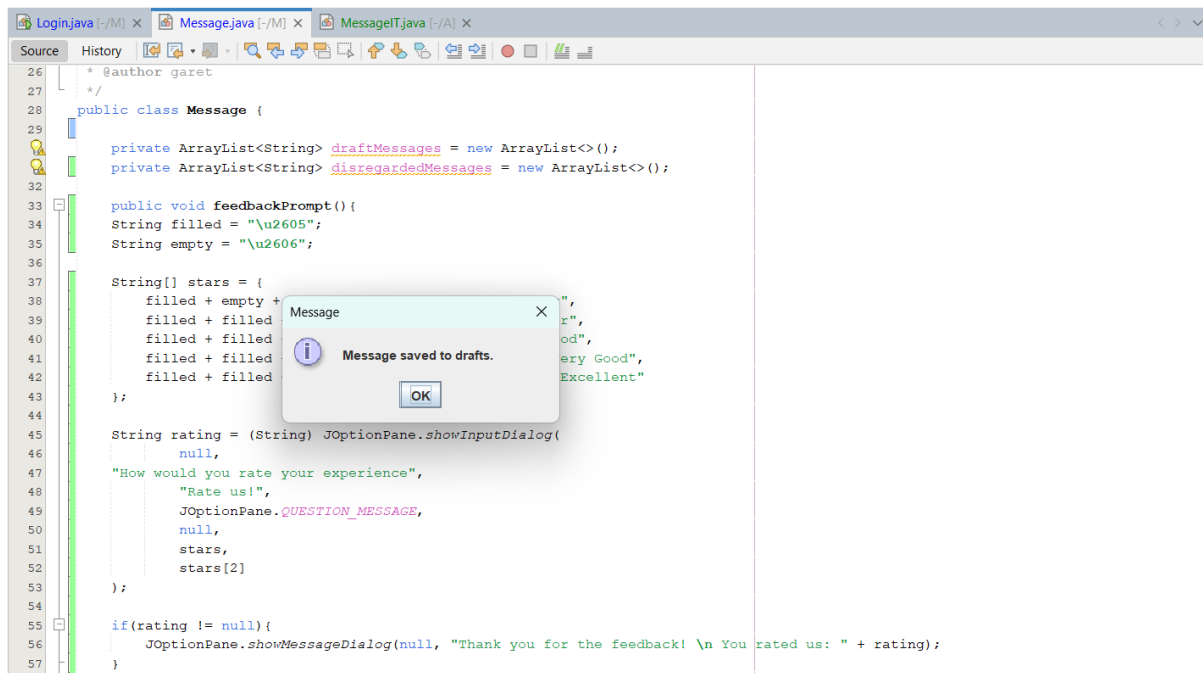


The screenshot shows the same IDE with the Message.java file open. The code is identical to the previous screenshot. A dialog box titled "Confirm Message" is displayed over the code. It contains the following text:

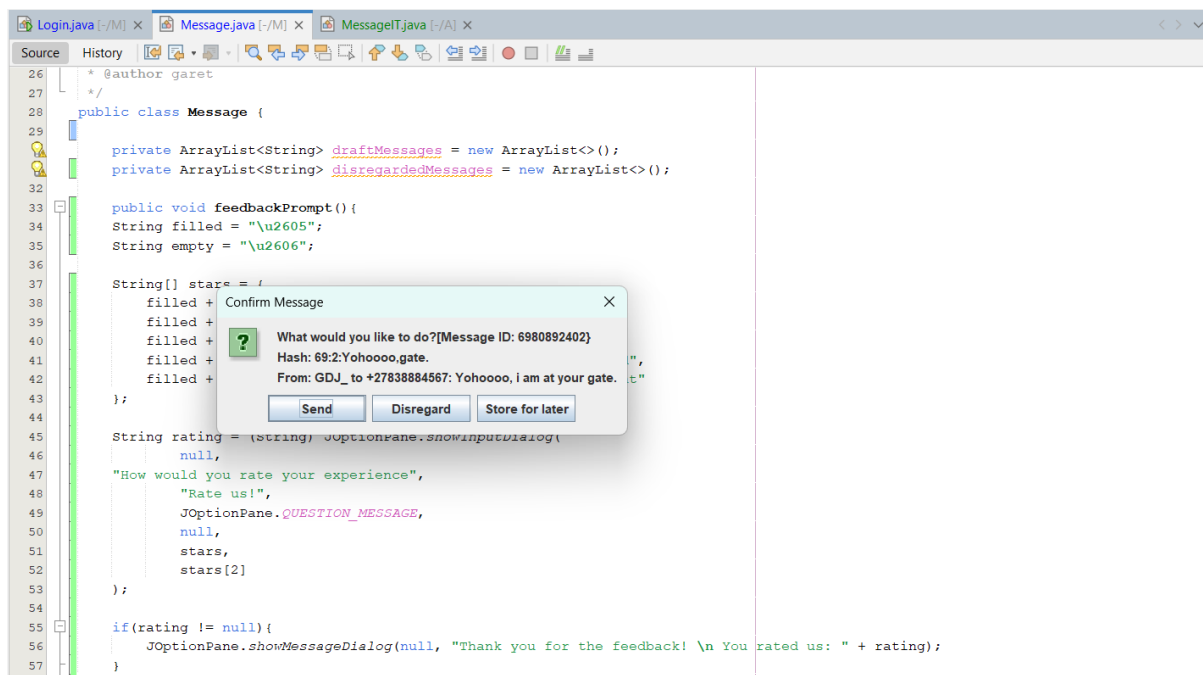
```
What would you like to do?[Message ID: 4471338343]
Hash: 44:1:Wherettime.
From: GDJ_to +27838884567: Where are you? You are late! I have asked you to be on time.
```

There are three buttons at the bottom: "Send", "Disregard", and "Store for later".

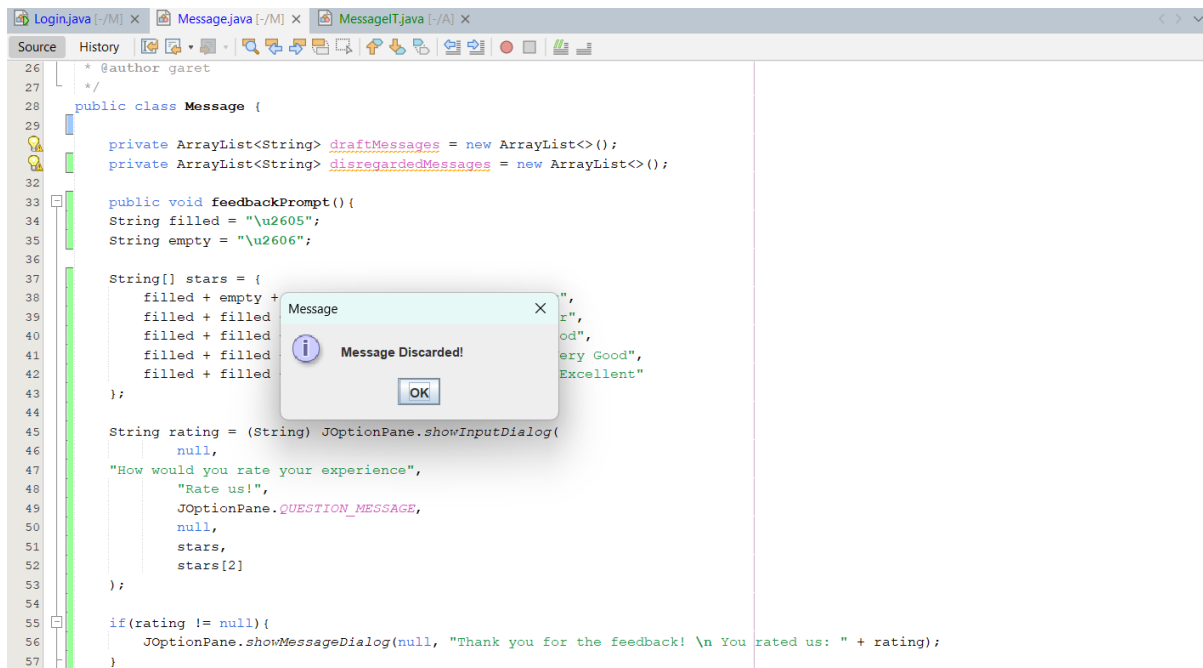
Test data 2_saved message:



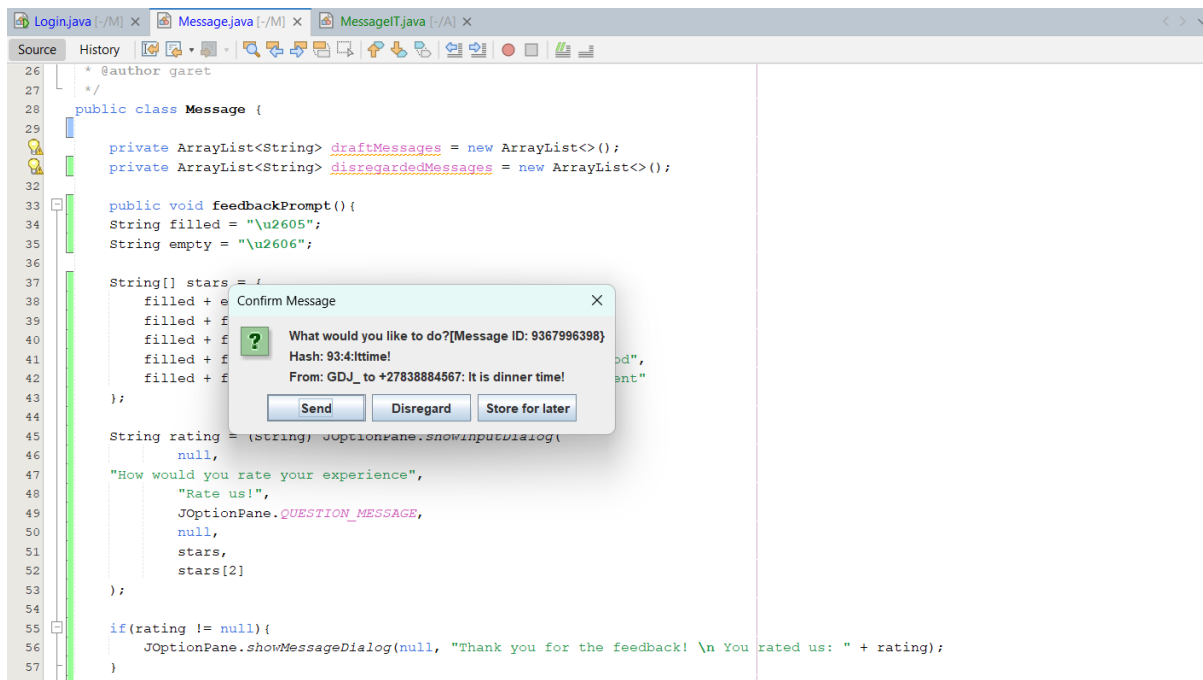
Test data 3_ message:



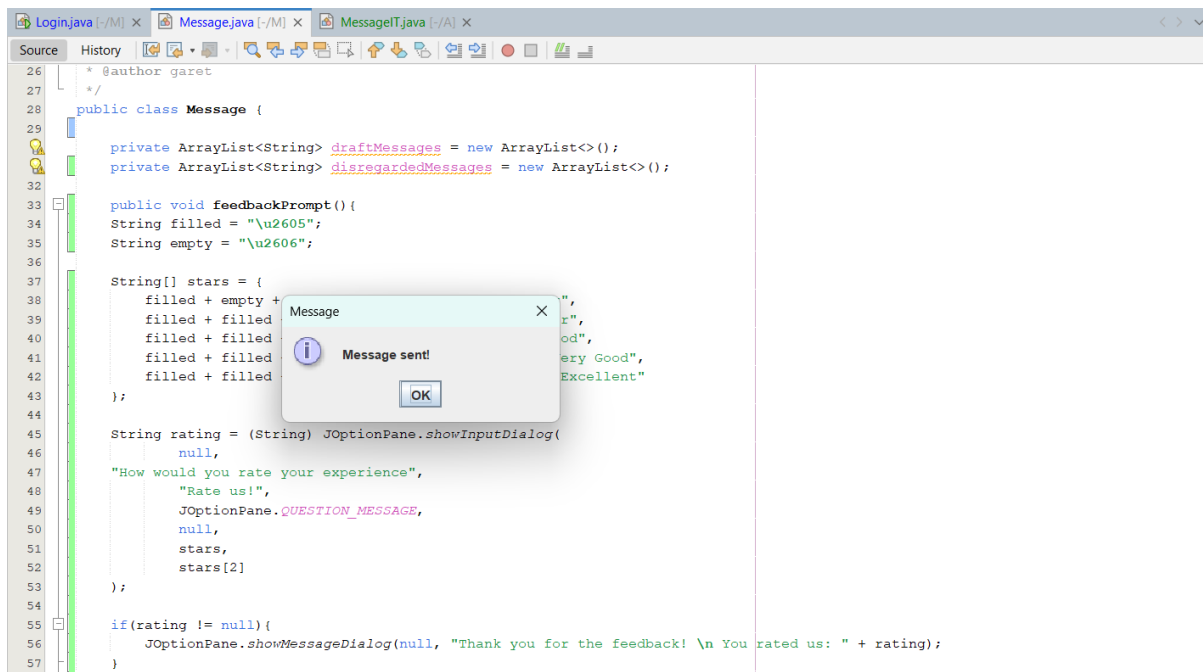
Test data 3_ message discarded:



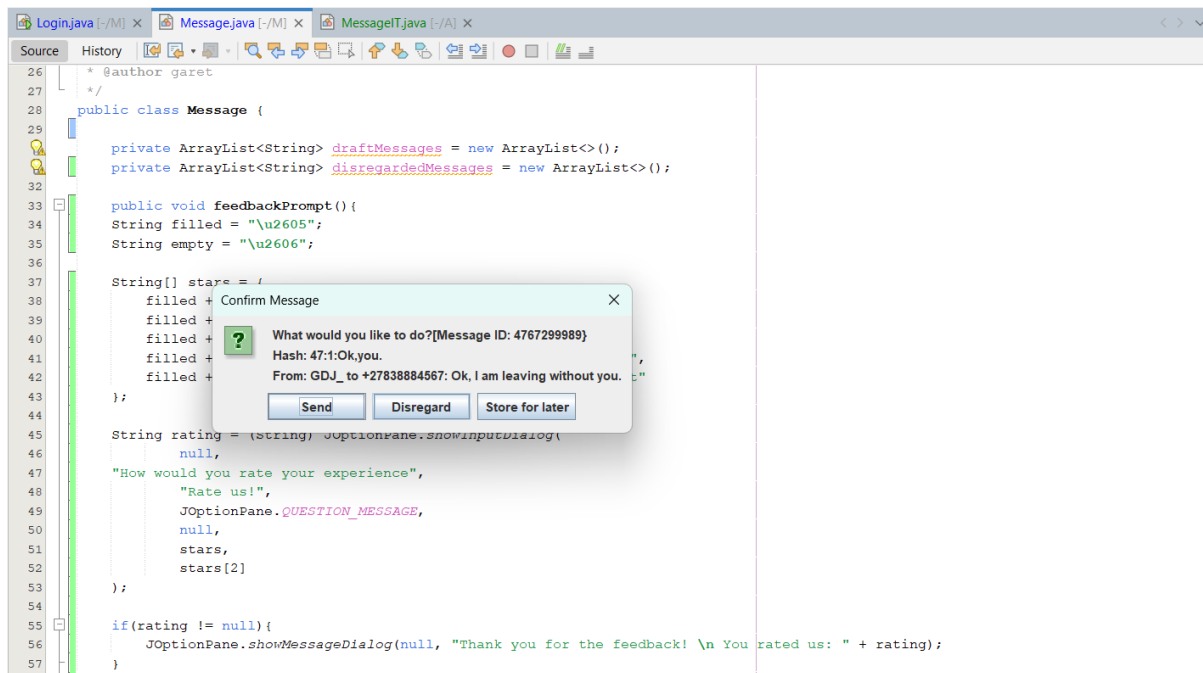
Test data 4_ message:



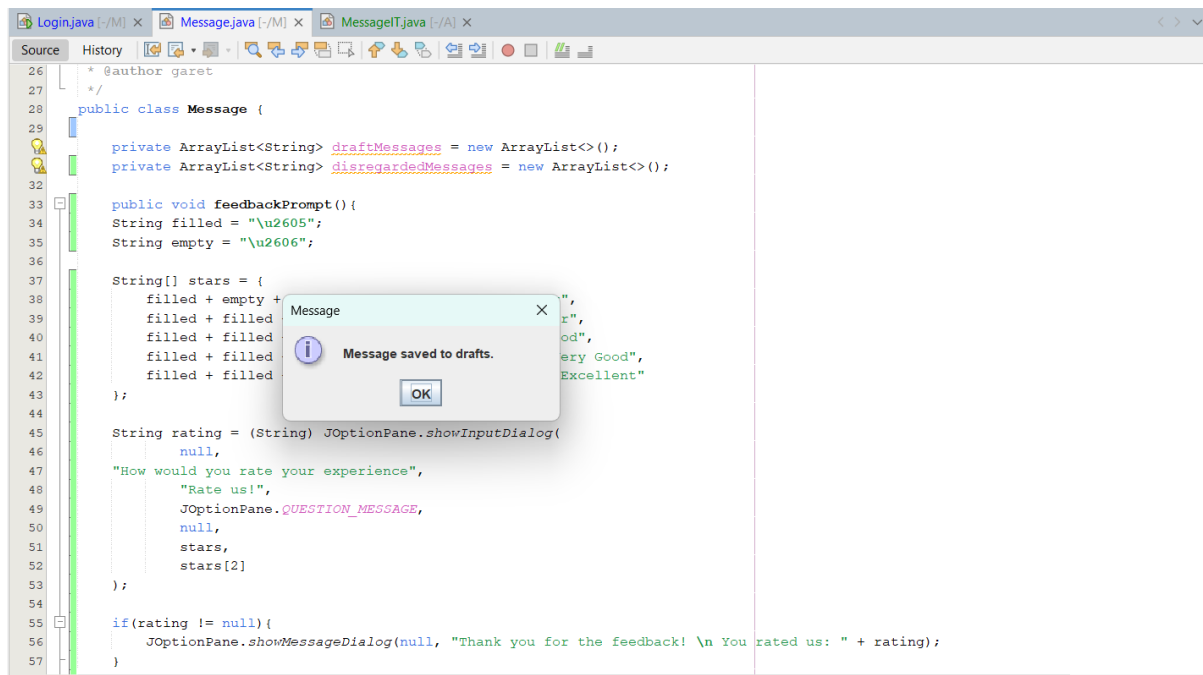
Test data 4_ message sent:



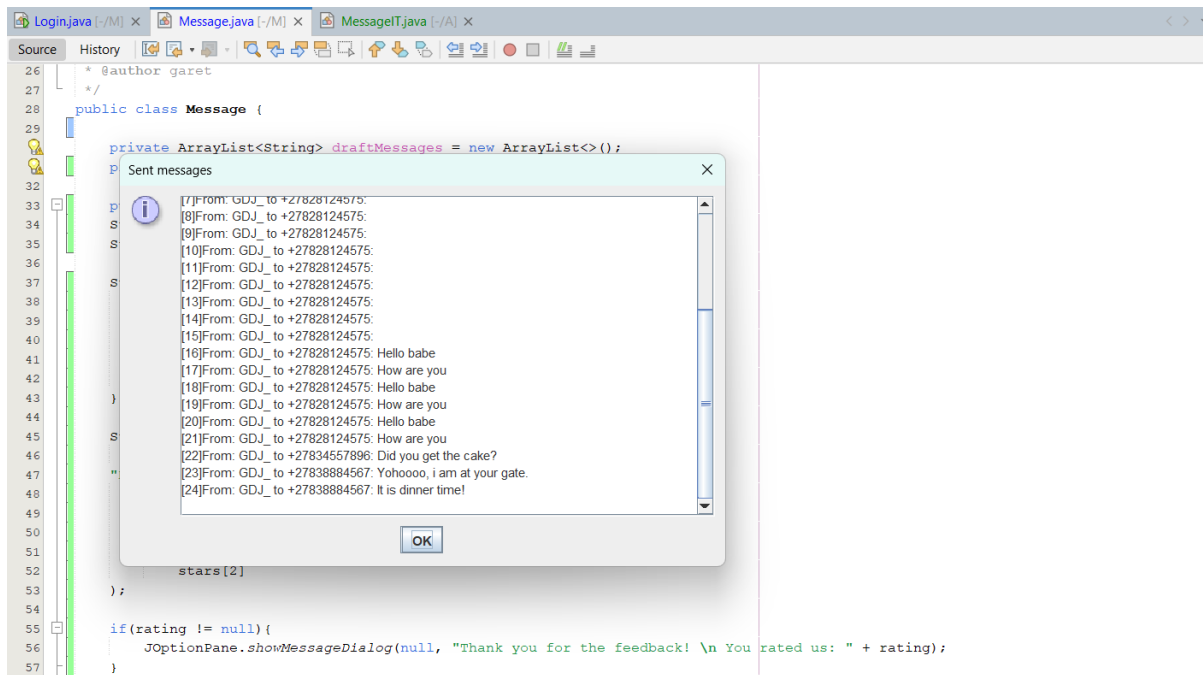
Test data 5_ message:



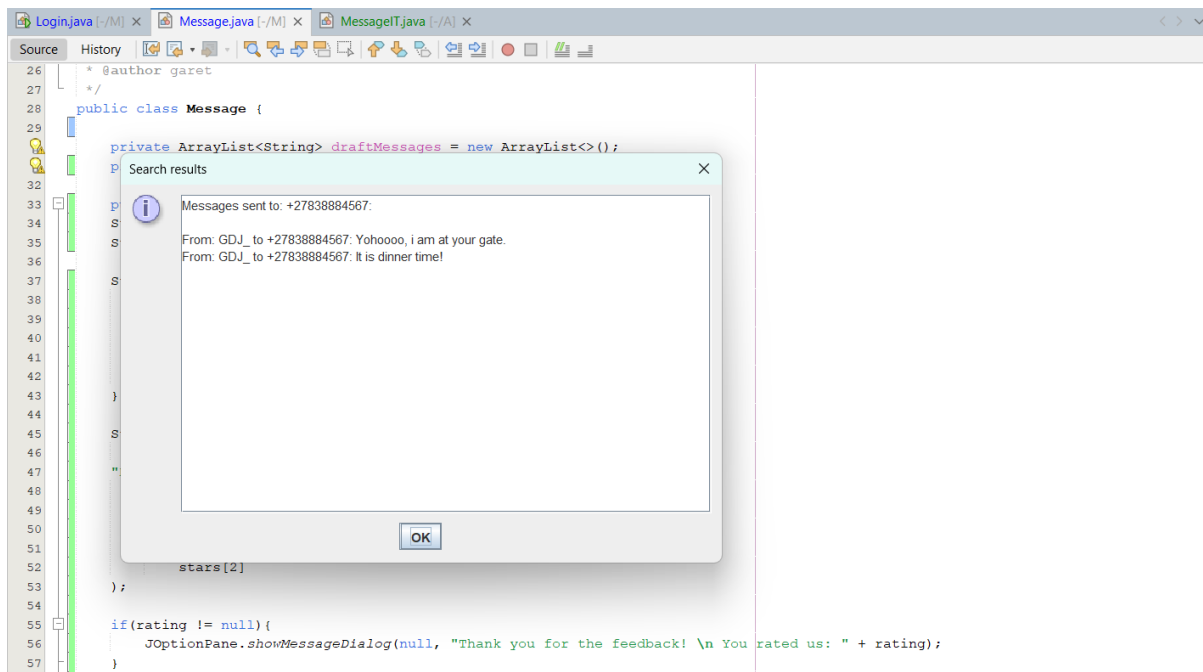
Test data 5_ saved message:



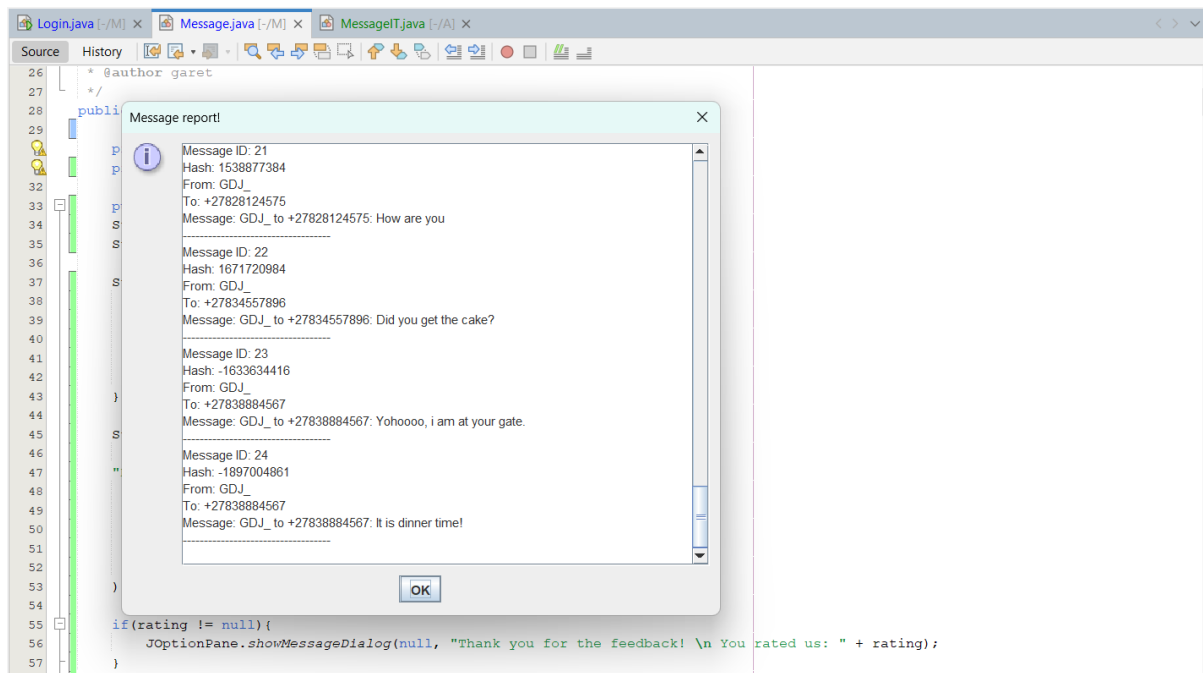
sent message page:



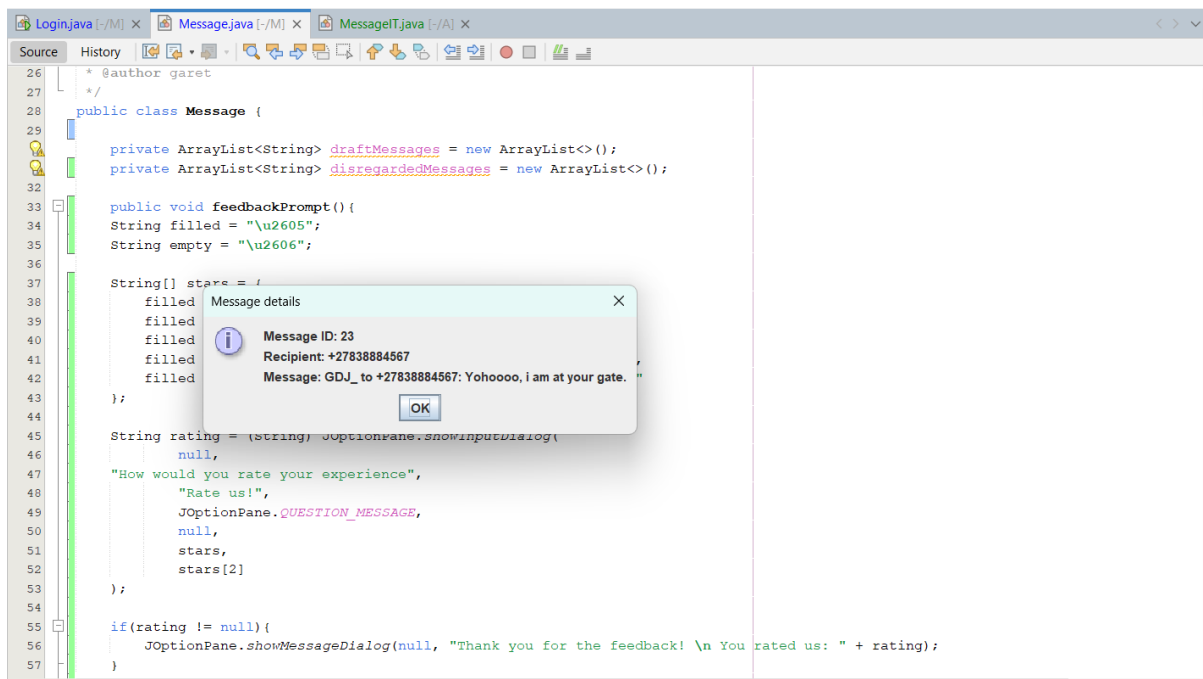
recipient search page:



message report page:



message ID page:



longest message page:

