OPP

ASSIGNMENT NO 1 (THEORY)

SP24-BSE-002

ABDUL MATEEN

SP24-BSE-B

This document provides a comprehensive overview of the **Messaging Application**, detailing its core components, functionalities, and usage. The application is designed to manage contacts and messages efficiently, allowing users to send, view, and manage messages seamlessly.

**Class Overview**

The application comprises two primary classes:

1. **Message Class**
2. **MessageApp Class**

**Message Class**

The Message class encapsulates the details of a single message, including its content, receiver, timestamp, status, and a unique identifier.

**MessageApp Class**

The MessageApp class serves as the main application controller, managing contacts and their associated messages. It provides functionalities to add, delete, display contacts, send messages, and check message statuses.

**Detailed Class Descriptions**

**Message Class**

**Attributes**

* **receiver** (String): The name of the message recipient.
* **content** (String): The textual content of the message.
* **timestamp** (LocalDateTime): The date and time when the message was created.
* **status** (boolean): Indicates whether the message has been sent (true) or not (false).
* **messageId** (String): A unique identifier for each message, auto-generated.
* **counter** (static int): A static counter to ensure each message has a unique ID.

**Constructor**

java

Copy code

public Message(String receiver, String content, boolean status)

* **Parameters**:
  + receiver: The recipient of the message.
  + content: The message content.
  + status: The sending status of the message.
* **Functionality**:
  + Initializes the receiver, content, and status attributes.
  + Sets the timestamp to the current date and time.
  + Generates a unique messageId by incrementing the static counter.

**Methods**

* **Getters and Setters**:
  + getReceiver(): Returns the receiver's name.
  + getContent(): Returns the message content.
  + getStatus(): Returns the message status.
  + setStatus(boolean status): Updates the message status.
  + getMessageId(): Returns the unique message ID.
  + getTimestamp(): Returns the timestamp of the message.
* **toString()**:
  + Overrides the default toString method to provide a formatted string representation of the message, including receiver, message ID, content, status, and timestamp.

**Example Usage**

java

Copy code

Message msg = new Message("Alice", "Hello, Alice!", true);

System.out.println(msg);

**Output:**

yaml

Copy code

Receiver: Alice

Receiver ID: 001,

Message: Hello, Alice!,

Status: Sent,

Time: 2024-04-27T15:30:45.123

**MessageApp Class**

**Attributes**

* **contactList** (ArrayList<String>): Stores the list of contact names.
* **messages** (ArrayList<ArrayList<Message>>): A 2D ArrayList where each sublist corresponds to a contact's messages.
* **scanner** (Scanner): Facilitates user input from the console.

**Constructor**

java

Copy code

public MessageApp()

* **Functionality**:
  + Initializes the contactList with predefined contacts: "Ahmad", "Ayan", and "Ali".
  + Initializes the messages 2D ArrayList, creating a sublist for each contact.
  + Sets up the scanner for user input.

**Methods**

1. **getContactIndex(String contact)**
   * **Purpose**: Retrieves the index of a contact in the contactList.
   * **Returns**: Index of the contact if found; otherwise, -1.
2. **sendMessage(String contact, String content, boolean status)**
   * **Purpose**: Sends a message to a specified contact.
   * **Parameters**:
     + contact: Recipient's name.
     + content: Message content.
     + status: Sending status (true for sent, false otherwise).
   * **Functionality**:
     + Validates the contact's existence.
     + Creates a new Message object and adds it to the corresponding sublist in messages.
     + Provides feedback on the operation's success or failure.
3. **displayContacts()**
   * **Purpose**: Displays the list of all contacts.
   * **Functionality**:
     + Prints each contact with an index number.
     + Handles the case when the contact list is empty.
4. **addContact()**
   * **Purpose**: Adds a new contact to the contactList.
   * **Functionality**:
     + Prompts the user for a new contact name.
     + Validates the input for non-emptiness and uniqueness.
     + Adds the new contact and initializes its message sublist.
5. **deleteContact()**
   * **Purpose**: Removes an existing contact from the contactList.
   * **Functionality**:
     + Prompts the user for the contact name to delete.
     + Confirms the deletion action.
     + Removes the contact and its associated messages upon confirmation.
6. **displayMessages(String contact)**
   * **Purpose**: Displays all messages associated with a specific contact.
   * **Parameters**:
     + contact: The contact whose messages are to be displayed.
   * **Functionality**:
     + Validates the contact's existence.
     + Iterates through the messages sublist and prints each message.
7. **checkMessageStatusByContact()**
   * **Purpose**: Checks and displays the status of all messages for a specific contact.
   * **Functionality**:
     + Prompts the user for a contact name.
     + Validates the contact's existence.
     + Iterates through the messages, displaying each message's content and status.
8. **main(String[] args)**
   * **Purpose**: Entry point of the application.
   * **Functionality**:
     + Initializes the MessageApp.
     + Presents a menu-driven interface for user interaction.
     + Handles user choices to perform various operations like displaying contacts, adding/deleting contacts, sending messages, displaying messages, checking message statuses, and exiting the application.

**Example Usage**

Upon running the application, the user is presented with a menu:

mathematica

Copy code

Here Messaging App

Menu:

1. Display Contacts

2. Add Contact

3. Delete Contact

4. Send Message

5. Display Messages for a Contact

6. Check Message Status

7. Exit

Enter your choice (1-7):

Selecting an option prompts further actions, such as adding a new contact or sending a message.

**User Interface Flow**

1. **Startup**:
   * The application initializes with a predefined contact list and corresponding message lists.
   * Displays the main menu to the user.
2. **Menu Options**:
   * **Display Contacts**: Lists all available contacts.
   * **Add Contact**: Allows the user to add a new contact after validating the input.
   * **Delete Contact**: Enables the user to remove an existing contact after confirmation.
   * **Send Message**: Facilitates sending a message to a chosen contact, capturing content and status.
   * **Display Messages for a Contact**: Shows all messages associated with a specific contact.
   * **Check Message Status**: Displays the sending status of all messages for a chosen contact.
   * **Exit**: Terminates the application gracefully.
3. **User Interaction**:
   * The application continuously prompts the user for actions until the "Exit" option is selected.
   * Input validation ensures robustness against invalid entries.

**Conclusion**

The **Messaging Application** provides a straightforward and efficient way to manage contacts and messages. By leveraging Java's ArrayList and Scanner for dynamic data handling and user input, the application ensures scalability and user-friendly interactions. The Message and MessageApp classes are designed with clear separation of concerns, facilitating maintenance and potential future enhancements.

**Appendix**

**Sample Code Snippets**

**Message Class**

java

Copy code

import java.time.LocalDateTime;

public class Message {

private String receiver;

private String content;

private LocalDateTime timestamp;

private boolean status;

private String messageId;

private static int counter = 0;

public Message(String receiver, String content, boolean status) {

this.receiver = receiver;

this.content = content;

this.status = status;

this.timestamp = LocalDateTime.now();

this.messageId = String.format("%03d", ++counter);

}

// Getters and Setters

@Override

public String toString() {

return "Receiver: " + receiver +

"\n Receiver ID: " + messageId +

", \nMessage: " + content +

", \nStatus: " + (status ? "Sent" : "Not Sent") +

", \nTime: " + timestamp;

}

}

**MessageApp Class**

java

Copy code

import java.util.ArrayList;

import java.util.Scanner;

public class MessageApp {

private ArrayList<String> contactList;

private ArrayList<ArrayList<Message>> messages;

private Scanner scanner;

public MessageApp() {

contactList = new ArrayList<>();

contactList.add("Ahmad");

contactList.add("Ayan");

contactList.add("Ali");

messages = new ArrayList<>();

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

messages.add(new ArrayList<>());

}

scanner = new Scanner(System.in);

}

// Other methods as described above

public static void main(String[] args) {

MessageApp app = new MessageApp();

// Menu-driven interface as described above

}

}