|  |  |
| --- | --- |
| **Name** | **M.Hamza** |
| **Class** | **BSE-B** |
| **Reg No** | **Sp24-BSE-074** |
| **Submitted to** | **Sir Shahid Bhatti** |

Java Assignment: Explanation of Classes

1. **Message Class:**

The Message class represents a single message that will be sent from one user to another. It contains all the necessary attributes to describe a message, such as the sender, receiver, the message text, and a unique message ID.

**Attributes:**

String sender: Hardcoded to "Hamza" for simplicity.

String receiver: Stores the name of the recipient of the message.

String id: A unique identifier for each message.

Boolean status: Tracks if the message has been read or not.

LocalDateTime time: Records the exact time the message was created.

String text: The content of the message.

**Constructor:**

The constructor initializes a new message by setting the receiver, text, and id. It also automatically captures the current time as the message creation time using LocalDateTime.now().

**Key Methods:**

setId(String id) and getId(): Set and retrieve the message's unique identifier.

setReceiver(String receiver) and getReceiver(): Set and get the name of the message's recipient.

setText(String text) and getText(): Set and retrieve the message content.

setStatus(Boolean status) and getStatus(): Update and check the status of the message (whether it's been read or not).

toString(): Provides a readable string representation of the message, displaying details such as the sender, receiver, ID, time, and the actual message text.

equals(Object obj): Compares two Message objects by checking if their id fields are the same. This method ensures that each message can be uniquely identified and compared.

This class provides the basic structure for sending, storing, and managing messages between contacts in the app.

**2. MessagingApp Class:**

The MessagingApp class acts as the main hub for managing contacts and messages. It contains arrays to store contacts and their corresponding messages and provides methods to perform operations like adding contacts, sending messages, and retrieving messages.

**Attributes:**

String[][] contact: A 2D array where each contact is stored as a record with three elements: contact ID, name, and phone number. It can store up to 100 contacts.

Message[][] m1: A 2D array to store messages for each contact. Each row corresponds to a contact, and each column stores messages sent to that contact.

int contactCount: Keeps track of how many contacts have been added to the app.

int[] chatCount: An array to store the number of messages sent to each contact.

Constructor:

Initializes the contact and message arrays, sets contactCount to 0, and prepares the application to store up to 100 contacts and 100 messages per contact.

Key Methods:

addContact(String name, String number): Adds a new contact to the contact array by storing the contact’s name, phone number, and generating a unique ID. The contactCount is incremented with each new contact.

findIndexOfContact(String name): Returns the index of a contact in the contact array based on the contact's name. If the contact is not found, it returns -1.

displayContact(): Loops through the contact array and prints the details (ID, name, number) of each saved contact.

addMessage(String name): Allows a user to send a message to a specified contact by prompting the user for message content and an ID. The message is stored in the m1 array at the position corresponding to the contact.

displayMessage(String name): Displays all the messages exchanged with a specific contact. If the contact has no messages, it informs the user.

searchMessage(String name, String id): Searches for a specific message by its ID for a given contact. If found, the message details are displayed.

deleteMessage(String name, String id): Deletes a message by setting its corresponding array entry to null, effectively removing the message.

The MessagingApp class provides the core functionality for managing contacts and messages, making it possible for users to perform basic messaging tasks like adding contacts, sending, and viewing messages.

**3. TestMessage Class:**

The TestMessage class serves as the entry point for the messaging application. It provides a simple console-based menu system that allows users to interact with the MessagingApp class. The menu presents different options for adding contacts, sending messages, viewing messages, searching for messages, and deleting messages.

**Main Method:**

A while loop runs continuously, offering the user a menu with various options:

Add Contact

Display Contacts

Send Message

Display Message

Search Message

Delete Message

Exit

After the user selects an option (by entering a number), the corresponding action is executed by calling the appropriate method from the MessagingApp instance.

Switch Statement:

The user's choice is handled using a switch statement:

Case 1: Adds a contact by calling addContact(). The program prompts the user for the contact’s name and phone number.

Case 2: Calls displayContact() to print all saved contacts.

Case 3: Sends a message by calling addMessage(). The user is asked to provide the recipient’s name, the message ID, and the message content.

Case 4: Displays all messages for a specific contact using displayMessage().

Case 5: Searches for a specific message by its ID using searchMessage().

Case 6: Deletes a message using deleteMessage().

Case 7: Exits the program by breaking out of the loop.

User Interaction:

After each action, the user is prompted to either continue performing actions or exit the program by entering a specific number (8 to continue, anything else to exit).

Program Flow:

The program flow is simple: the user selects an option, performs the corresponding action, and then decides whether to continue or exit. The loop ensures that the user can perform multiple actions in one session.

**Conclusion:**

This set of three classes (Message, MessagingApp, and TestMessage) forms a basic messaging system. The Message class represents individual messages, MessagingApp handles contact and message management, and TestMessage provides a user interface for interacting with the system. Together, they simulate the core functionality of a messaging app, such as adding contacts, sending and viewing messages, searching, and deleting messages.