**OOP ASSIGNMENT**

****

**Submitted by: Talha Manzoor.**

**Reg no: SP24-BSE-089.**

**Section: B.**

**Submitted to: Sir Shahid Bhatti.**

**Subject: OOP.**

**Assignment: Messaging App.**

**Overview of the Messaging Application**

**1. System Architecture and Design**

**Classes and Components**

* + - **User Class:** Handles user creation and management.
    - **Message Class:** Manages individual messages with unique identifiers.
    - **Conversation Class:** Facilitates conversations between users, storing and managing messages.
    - **Application Core:** Integrates all classes, providing a flawless user experience through a console interface.
* **Data Structures Used**
  + **Arrays and 2D Arrays:** Manage users and conversations systematically.
  + **Encapsulation:** Ensures data integrity and security within classes.
  + **Static Variables:** Generate unique message IDs across the application.
* **Design Principles Applied**
  + **Object-Oriented Programming (OOP):** Utilizing classes and objects.
  + **Encapsulation:** Protecting data through private access modifiers and public getters/setters.
  + **Constructor Overloading:** Providing multiple ways to instantiate objects.
  + **Method Overloading:** Enhancing functionality with methods of the same name but different parameters.

**2. Detailed Class Descriptions**

**a. User Class**

* **Purpose**
  + Represents a user within the messaging application.
* **Key Attributes**
  + username: Identifies the user uniquely.
* **Key Methods**
  + **Constructors:** Default, parameterized, and copy constructors.
  + **Getters and Setters:** Access and modify user attributes.
  + **Override Methods:** toString() for meaningful representation and equals() for comparison based on username.

**b. Message Class**

* **Purpose**
  + Represents an individual message exchanged between users.
* **Key Attributes**
  + messageId: Unique identifier for each message.
  + sender and receiver: Indicate the participants in the message exchange.
  + content: Stores the message text.
  + isRead and isSeen: Detect the status of the message.
* **Key Methods**
  + **Constructors:** Default, parameterized, and copy constructors.
  + **Getters and Setters:** Access and modify message attributes.
  + **Override Methods:** toString() for detailed message information and equals() for comparison based on messageId.

**c. Conversation Class**

* **Purpose**
  + Manages the collection of messages between two users.
* **Key Attributes**
  + user1 and user2: Participants in the conversation.
  + messages: Array to store Message objects.
  + messageCount: Detect the number of messages in the conversation.
* **Key Methods**
  + **Constructors:** Default, parameterized, and copy constructors.
  + **SendMessage():** Overloaded methods to send messages between users.
  + **ReadAllMessages():** Displays all messages in the conversation.
  + **ReadMessageById():** Displays a specific message based on its ID.
  + **DeleteMessageById():** Removes a message from the conversation.
  + **MarkAllAsSeen():** Updates the status of all messages to seen.
  + **Override Methods:** toString() for conversation details.

**d. MessagingApp Class**

* **Purpose**
  + Serves as the core of the application, handling user interactions and managing conversations.
* **Key Attributes**
  + users: Array to store User objects.
  + conversations: 2D array to manage Conversation objects between user pairs.
  + userCount: Tracks the number of users in the application.
* **Key Methods**
  + **AddUser():** Adds a new user to the application.
  + **FindUserIndex():** Retrieves the index of a user based on username.
  + **GetConversation():** Retrieves or creates a conversation between two users.
  + **Start():** Initiates the application and handles the main menu interactions.
  + **GetUserByUsername():** Helper method to bring a User object based on username.
  + **Main Method:** Entry point to run the application.

**3. Key Features and Functionalities**

**a. User Management**

* **Adding Users**
  + Ability to add new users with unique usernames.
  + Prevents duplicate usernames to maintain uniqueness.
* **User Validation**
  + Ensures that only existing users can participate in conversations.

**b. Messaging System**

* **Sending Messages**
  + Users can send messages to other users.
  + Each message is assigned a unique messageId.
* **Reading Messages**
  + Users can read all messages in a conversation.
  + Specific messages can be accessed using their messageId.
* **Deleting Messages**
  + Users can delete specific messages based on messageId.

**c. Conversation Management**

* **Handling Multiple Conversations**
  + Supports multiple conversations between different pairs of users.
  + Utilizes a 2D array to manage conversations systematically.
* **Message Tracking**
  + Tracks whether messages have been read or seen.
  + Updates statuses accordingly when messages are accessed.

**4. Enhancements and Optimizations**

**a. Input Validation and Error Handling**

* **Ensuring Validity**
  + Validates user inputs to prevent invalid operations.
  + Handles scenarios like exceeding maximum user or message limits gracefully.
* **User Feedback**
  + Provides informative messages to guide users during interactions.

**b. Data Integrity**

* **Encapsulation**
  + Maintains data integrity by keeping class attributes private and controlling access through methods.
* **Deep Copying**
  + Implements deep copy constructors to prevent unintended modifications of objects.

**5. User Interaction Flow**

**a. Main Menu Options**

1. **Add User**
   * Prompts for a new username and adds the user to the system.
2. **Send Message**
   * Collects sender and receiver usernames along with message content.
   * Sends the message if both users exist and are distinct.
3. **Read All Messages**
   * Displays all messages in a specific conversation between two users.
4. **Read Specific Message**
   * Allows users to view a particular message by entering its messageId.
5. **Delete Message**
   * Enables users to delete a message from a conversation using its messageId.
6. **Exit**
   * Terminates the application gracefully.

**b. Interaction Steps**

* **Adding Users**
  + Users enter unique usernames to participate in the messaging system.
* **Sending Messages**
  + Users specify the sender and receiver to establish the conversation context.
  + Messages are stored with unique IDs.
* **Managing Conversations**
  + Users can navigate through their conversations, reading and managing messages as needed.

**6. Testing and Validation**

**a. Functional Testing**

* **User Addition**
  + Tested adding users with unique and duplicate usernames.
* **Messaging Functionality**
  + Verified sending, reading, and deleting messages between users.
  + Ensured that message IDs are unique and correctly assigned.

**b. Edge Case Handling**

* **Invalid Inputs**
  + Tested the application's response to invalid menu choices and empty inputs.
* **Capacity Limits**
  + Confirmed that the application handles scenarios where maximum users or messages are reached.

**c. Data Integrity**

* **Deep Copy Verification**
  + Ensured that copy constructors create independent copies without shared references.
* **Status Tracking**
  + Verified that message statuses (isRead, isSeen) are updated correctly upon access.

**7. Conclusion**

* **Summary of Achievements**
  + Successfully developed a console-based messaging application with key functionalities.
  + Implemented valid class structures adhering to OOP principles.
  + Enhanced the application with systematic message tracking.
* **Key Learnings**
  + Gained practical experience in designing and implementing Java applications.
  + Learned to manage data effectively using arrays and encapsulation.
  + Enhanced problem-solving and debugging skills through iterative development.

Top of Form