Project Part - 3: Refactoring

- **Team:** Mridula Natrajan
  Hari Shreenivash Madras Vanamamalai
  Rajarshi Basak

- **Title:** ChatApp

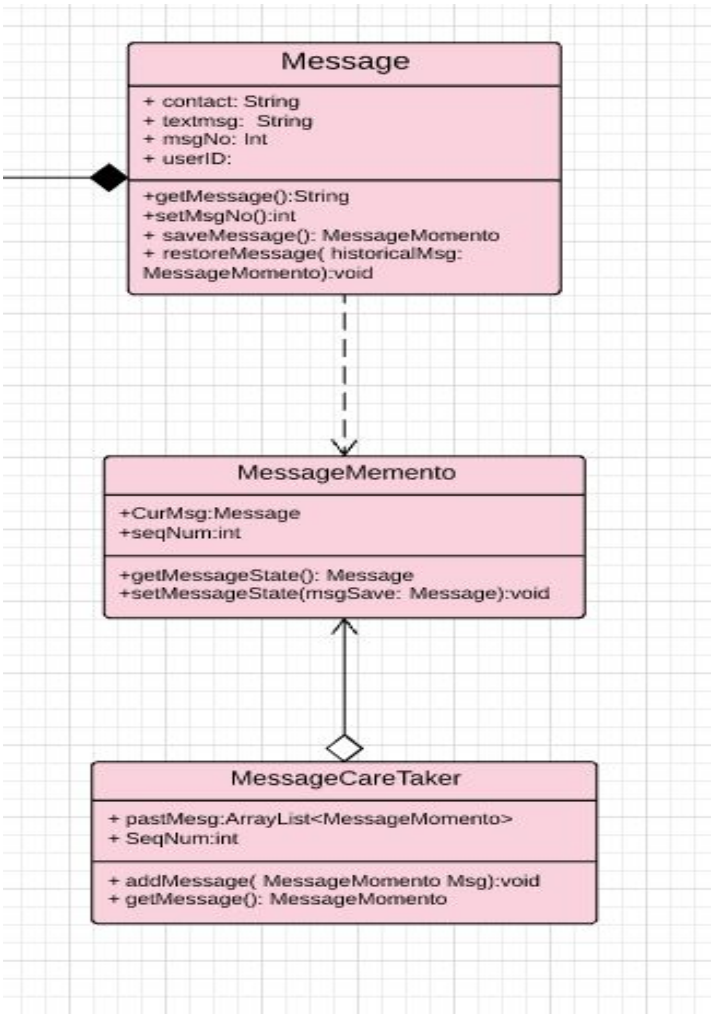
- **Project Summary:** A java-based messaging application which implements Object-oriented concepts and includes basic functionalities. The system is based on a client-server model in which multiple clients (contacts wishing to initiate a chat with other contacts) can request access through the server.

- **Refactoring:**

  Design Patterns:

  1) Memento:

  Implementing this design pattern helps in saving the chat history and restoring it when the user opens an existing chat with a contact.
  The Message class has been implemented as a originator class, this class is responsible for getting the text in a message and assign it a sequence number. The originator class(Message) then delegates the work of saving the state of the Message and restoring the state of Message to the MementoMessage Class. The MessageCareTaker class is reponsible for storing the stateof the messages to maintain a history of messages.



2) Observer:

When user 1 wants to send a message to user 2, the client (user 1) will send the message (with the recipient of the message attached to the message) to the server, and the server will ensure this message is delivered to the appropriate client (user 2).

On the other hand, suppose a group exists, consisting of five users. Suppose user 1 intends to send a message to the group. Hence, client 1 (user 1) will send the message (with the recipients of the message, i.e. all the group members) to the server, and the server will then write message on the walls of all the other group members.

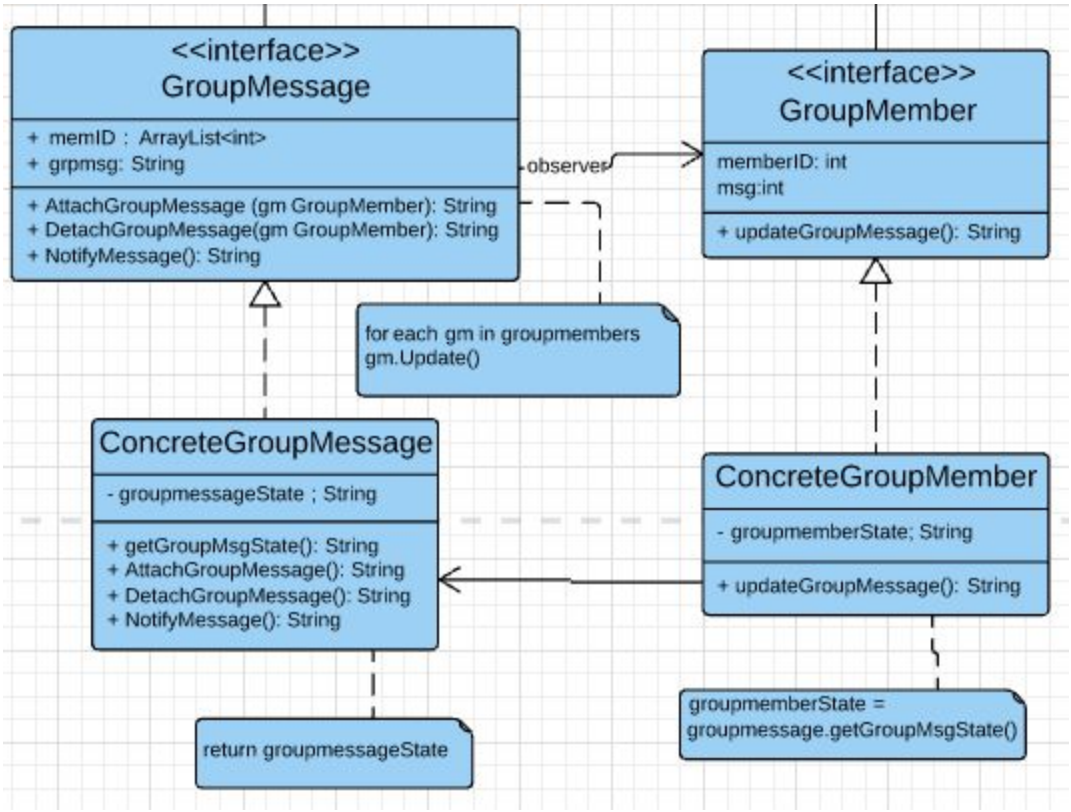
However, to ensure all the group members receive this message, this operation would consume

a lot of time since the server would have to write the message (that was intended to be delivered to the group) individually on all the clients which are group members.

To circumvent this, we employ the Observer pattern.

In our Observer Pattern, the Subject (the GroupMessage abstract class) knows its observers, who are the clients (members of the group), and provides an interface for attaching and detaching Observer (GroupMember) objects. The Observer (a client which is a member of the group) defines an updating interface for objects that should be notified of changes in the subject. The ConcreteSubject (the ConcreteGroupMessage class) stores our state of interest, which is the Group Message, to the ConcreteObserver objects, and sends a notification to its observers when its state changes. Finally, the ConcreteObserver, which maintains a reference to a ConcreteGroupMessage, stores state that should stay consistent with the subject's and also implements the Observer interface to ensure that its state is consistent with the state of the subject.

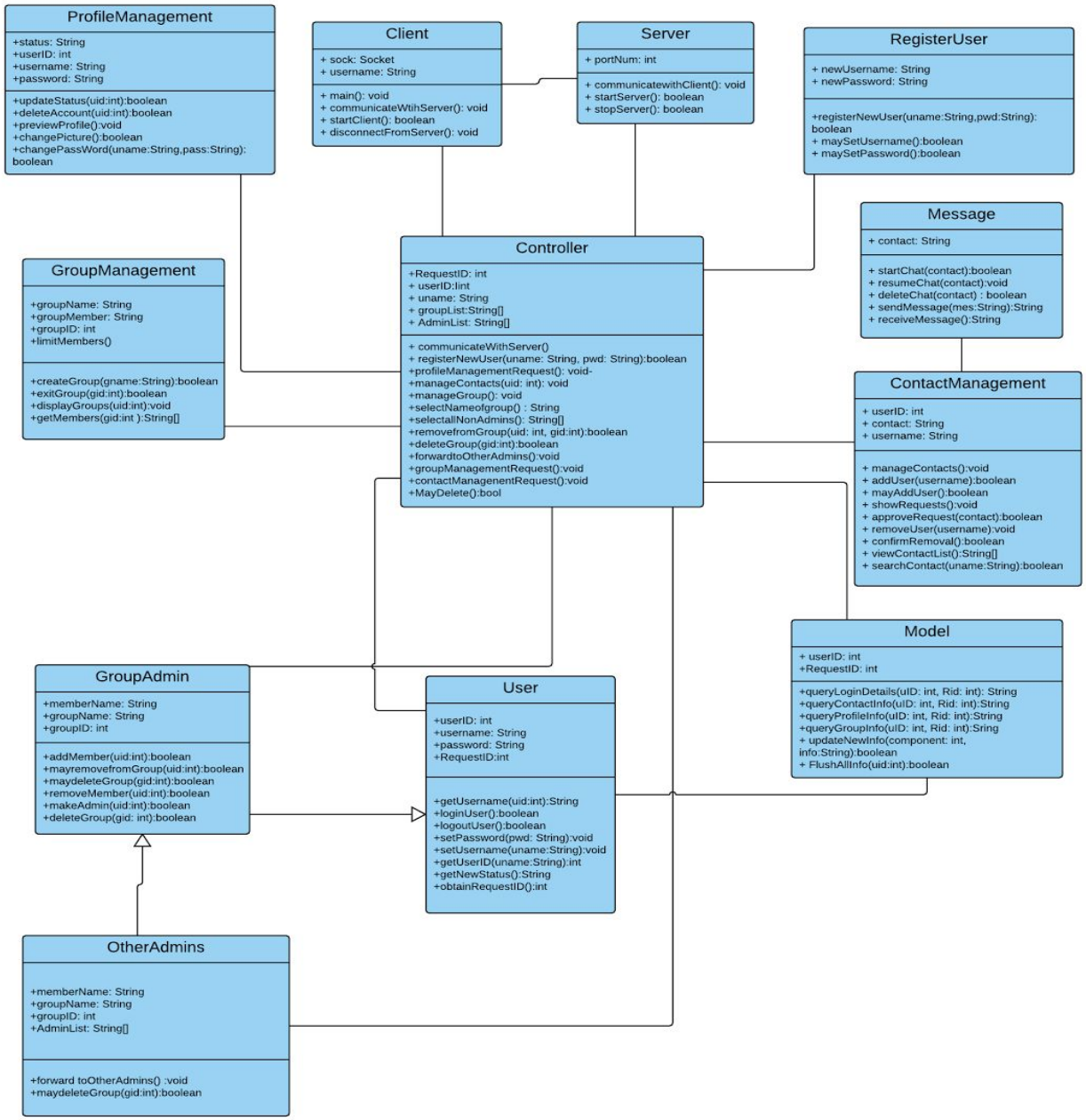
In short, each Group Member observes the state of the Group Message; whenever it changes (i.e. whenever there is a new group message), each group member is notified of that change. The member then updates that change.



Other Changes:

- We have included relationships among classes such as Association, Inheritance, Dependencies, Aggregation and Composition.

Existing class Diagram:



Refactored Class Diagram:

Memento - Pink classes
Observer - Blue classes
Existing classes from old class diagram - Orange Classes