# 1. Breve descrizione del progetto

**Tape** è un software di messaggistica istantanea che ti consente di inviare **messaggi ai tuoi contatti** tramite la creazione di conversazioni.

Le **conversazioni possono essere di gruppo o con un singolo contatto**.

**All'apertura** dell'applicazione bisogna inserire la **propria e-mail e password** per poter consultare le ultime conversazioni, oppure ci si può **registrare** come un nuovo utente

Si è deciso di implementare i pattern **Observer e Mediator** per il caso d'uso **UC1 Contattare un utente**

# 2. Parte del programma dove si implementano i pattern

## 2.1 Interfaccia Subject

```
1   package program.Domain;
2
    12 usages  1 implementation  PierfrancescoN
3   public interface Subject { //the subject interface
        1 usage  1 implementation  PierfrancescoN
4       void attach(Observer observer);
        1 usage  1 implementation  PierfrancescoN
5       void notifyANewMessage(Message message);
        1 usage  1 implementation  PierfrancescoN
6       void notifyCreation();
        1 usage  1 implementation  PierfrancescoN
7       void detach(User loggedUser);
8   }
9
```

## 2.2 Interfaccia Observer

```
1   package program.Domain;
2
3
    10 usages  1 implementation  PierfrancescoN
4   public interface Observer { //the observer interface
        1 usage  1 implementation  PierfrancescoN
5       void update(Subject subject, Object ob);
        1 usage  1 implementation  PierfrancescoN
6       void updateCreation(Subject subject);
7   }
```

## 2.3 Classe Conversation, implementa interfaccia Subject

Contatta il ChangeMaster per:

- effettuare le attach, detach e notify
- per ricevere i suoi messaggi (dati condivisi)

```java
package program.Domain;
import ...
48 usages  PierfrancescoN *
public class Conversation implements Subject{
    2 usages  PierfrancescoN
    public void sendAMessage(Message message) {
        ConversationDao conversationDao= DBManager.getInstance().getConversationDao();
        conversationDao.addAMessage(message);
        notifyANewMessage(message);
    }
    1 usage  PierfrancescoN
    public Set<Message> getAllMessages() { return ChangeMaster.getInstance().getAllMessagesForAConversation(this); }
    1 usage  PierfrancescoN
    @Override
    public void attach(Observer observer) { ChangeMaster.getInstance().register( subject: this, observer); }
    1 usage  PierfrancescoN
    @Override
    public void notifyANewMessage(Message message) { ChangeMaster.getInstance().notify( subject: this,message ); }

    1 usage  PierfrancescoN
    @Override
    public void notifyCreation() {
        ChangeMaster.getInstance().notifyCreation( subject: this);
    }
    1 usage  PierfrancescoN
    @Override
    public void detach(User loggedUser) {
        ChangeMaster.getInstance().unRegister( subject: this, loggedUser);
        removeParticipant(loggedUser.getUsername());
        ConversationDao conversationDao=DBManager.getInstance().getConversationDao();
        conversationDao.removeAParticipant(loggedUser.getUsername(), id);
    }
```

## 2.4 Classe User, implementa l'interfaccia Observer

- Contatta il ChangeMaster per ricevere le sue conversazioni (dati condivisi con altri utenti)
- Chiama l'attach di una Subject
- Effettua l'update della UI quando notificato

```java
package program.Domain;
import ...
 PierfrancescoN *
public class User implements Observer{
    1 usage   PierfrancescoN
    public Set<Conversation> getConversations() {
        Set<Conversation> conversations1=ChangeMaster.getInstance().getConversations( user: this); //gets the conversations that the user is part of
        for(Conversation conversation:conversations1){
                conversation.attach( observer: this); //the user attaches itself to the conversation
            }
        return conversations1;
    }
    1 usage   PierfrancescoN
    @Override
    public void update(Subject subject, Object ob) {  //receiving a new message
        if(subject instanceof Conversation && ob instanceof Message){
            controller.addMessage((Message) ob);
        }
    }
    1 usage   PierfrancescoN
    @Override
    public void updateCreation(Subject subject) { //the user is added to a conversation
        if(subject instanceof Conversation conversation){
            controller.addAConversation(conversation);
        }
    }
```

## 2.5 Classe ChangeMaster (Mediator)

- Contiene i dati condivisi e li passa ai client quando richiesti
- Contiene il mapping fra Subject e Observers
- Si occupa di notificare gli Observer e degli aggiornamenti dei dati
- Non viene creata la classe Mediatore astratto perché è presente un solo Mediatore concreto
- È una classe Singleton perché:
    - È presente una sola istanza e si vuole fornire un punto di accesso universale ed essa
    - Si vuole dare la possibilità in futuro di ridefinire le operazioni del ChangeMaster attraverso le sue sottoclassi

```
19 usages  PierfrancescoN
16  public class ChangeMaster { //singleton mediator class that manages the communication and the shared data
        6 usages
17      private final HashMap<Observer, ClientThread> clientThreadHashMap=new HashMap<>(); //the clientThread is the thread that is associated with the user
        7 usages
18      private final HashMap<Subject, Set<Observer>> subjectObserverHashMap=new HashMap<>(); //subject to observer mapping
        1 usage  PierfrancescoN
19      private ChangeMaster(){} //private constructor
        12 usages
20      private final HashMap<Conversation, Set<Message>> conversations=new HashMap<>(); //shared data
        3 usages
21      private final Set<User> users=new HashSet<>();
        1 usage
22      private static final ChangeMaster instance=new ChangeMaster(); //singleton instance
        PierfrancescoN
23      public static synchronized ChangeMaster getInstance() { return instance; } //returns the singleton instance
26
        1 usage  PierfrancescoN
27      public synchronized void  addAClient(ClientThread clientThread, User loggeduser) {...}
        1 usage  PierfrancescoN
30      public synchronized Set<Conversation> getConversations(User user) {...}
        2 usages  PierfrancescoN
38      public synchronized void register(Subject subject, Observer observer) {...}
        1 usage  PierfrancescoN
48      public synchronized User getUserWithUsernameAndPassword(String usernameText, String pass) {...}
        2 usages  PierfrancescoN
62      public synchronized Set<Message> getAllMessagesForAConversation(Conversation conversation) {...}
        1 usage  PierfrancescoN
70      public synchronized boolean getUserWithUsername(String username) {...}
        1 usage  PierfrancescoN
74      public synchronized void addAUser(User user) {...}
        1 usage  PierfrancescoN
79      public synchronized void notify(Subject subject, Object ob) {...}
        1 usage  PierfrancescoN
93      public synchronized void notifyCreation(Subject subject) {...}
101
        1 usage  PierfrancescoN
102     public synchronized void unRegister(Subject subject, Observer observer) {...}
        1 usage  PierfrancescoN
120     public synchronized void removeAObserver(Observer observer) {...} //removes the thread from the list of active threads (no more notifications will be sent)
123 }
```

# 3. Benefici che i pattern hanno portato al programma

- **Diminuzione delle copie di oggetti**: ogni utente quando effettua la Login senza il Mediator avrebbe dovuto accedere ai dati dal DB, effettuando delle copie locali di tutte le sue conversazioni con annessi messaggi.
- **Comunicazioni semplificate**: viene trasformata una relazione uno a molti (una Subject più Observer) in una relazione uno ad uno (una Subject un ChangeMaster).
- **Comunicazioni centralizzate**: le comunicazioni sono solo da o per il ChangeMaster.
- **Disaccoppia i Colleague (utenti)**: un colleague non deve conoscere i suoi colleagues per comunicare con loro.
- **Disaccoppia le Subject (conversazioni) dagli Observer (utenti)**: una Subject non conosce gli Observer a cui è collegata e viceversa.
- **È possibile aggiungere Observer in qualsiasi momento senza modificare le Subject o gli altri Observer**

# 4. Tecnologie utilizzate

- Database Postgres (il dump del DB è presente nella cartella /resources/tape/databaseDump)
- JDBC
- Librerie Java Swing e AWT
- Libreria Java Concurrent (per rappresentare più client)