UNIVERSITÀ DEGLI STUDI DI NAPOLI "PARTHENOPE"
FACOLTÀ DI SCIENZE DELL'IGNEGNERIA
E DELLA SALUTE
CORSO DI LAUREA IN INFORMATICA


# Sistema di gestione del servizio di microblogging Twitter

DOCENTE
Angelo Ciaramella

CANDIDATO
Martina Cimafonte

Matricola 0124001987

Anno Accademico 2022-2023

# Indice

**1 Introduzione** **1**

1.1 Requisiti . . . . . 1

1.2 Diagramma delle classi UML . . . . . 2

1.3 Descrizione Pattern utilizzati. . . . . 3

1.3.1 Mediator . . . . . .3-4

1.3.2 Command . . . . . .5-6

1.3.3 Strategy . . . . . 6-7

1.3.4 Singleton . . . . . 7

**2 Pratica** **7**

2.1 Parti rilevanti del codice sviluppato. . . . . 7-10

2.2 Gestione delle eccezioni. . . . . 10-12

2.3 Gestione dell'interfaccia grafica . . . . . .13-18

**3 Illustrazione funzionamento** **19**

3.1 immagini . . . . . 19-23

# Capitolo 1

## Introduzione

Si vuole sviluppare un sistema per la gestione di Twitter.
Twitter è un social network o per meglio dire, un servizio di micro-blogging, che consente di comunicare attraverso brevi messaggi che possono essere pubblicati sia da smartphone e tablet, oltre che da computer. È un servizio gratuito accessibile mediante registrazione alla piattaforma.

### 1.1 Requisiti

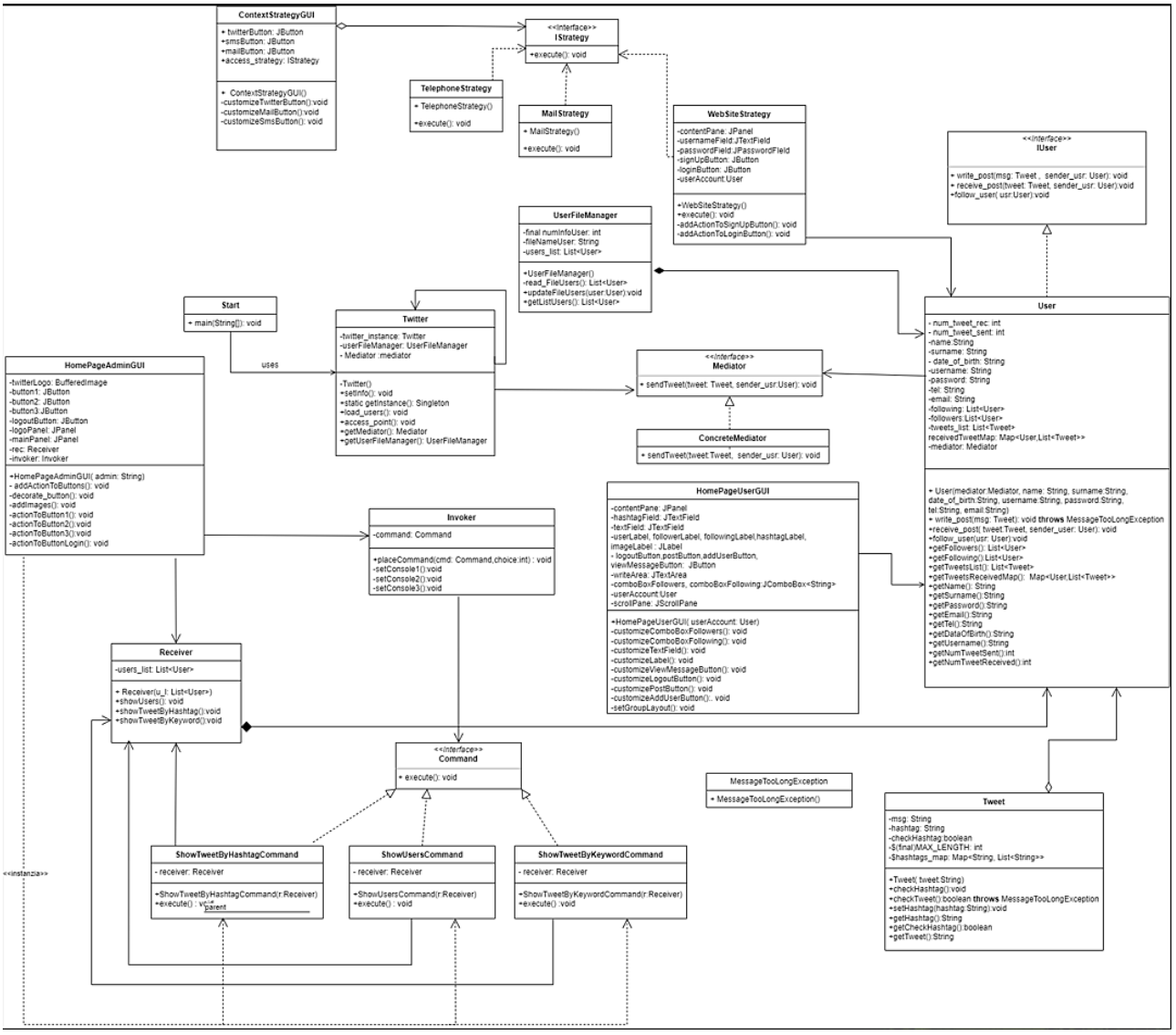
Il servizio di social networking fornisce agli utenti una pagina personale aggiornabile tramite messaggi di testo con lunghezza massima di 140 caratteri. Ogni utente può avere un certo numero di follower che ricevono i suoi messaggi pubblicati e può ricevere messaggi dagli utenti che segue.
Il sistema prevede l'accesso sia in modalità amministratore che in modalità utente.
L'amministratore può effettuare le seguenti operazioni:

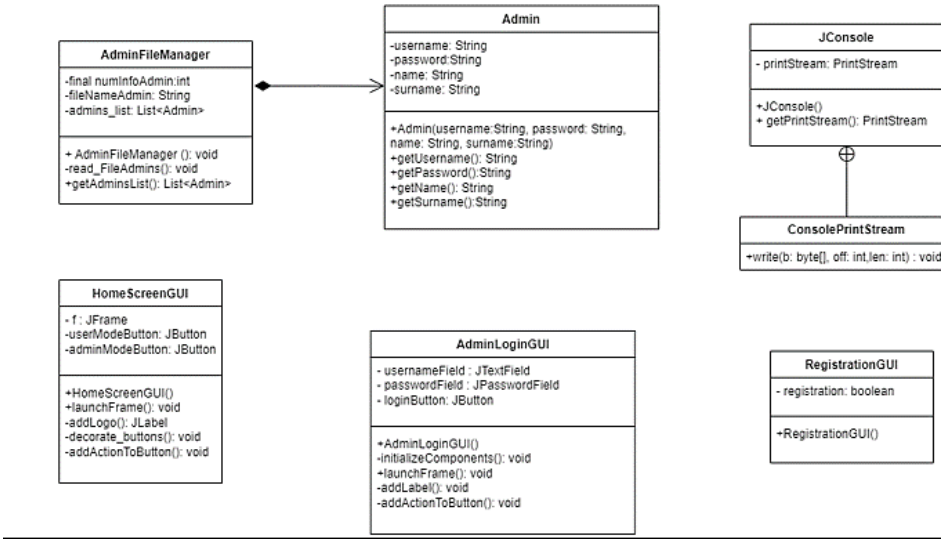
- mostrare l'elenco degli utenti in base al numero di messaggi ricevuti o inviati
- visualizzare i messaggi divisi in categorie in base agli hashtag
- data una parola, visualizzare tutti i messaggi dei diversi utenti che contengono quella parola.

L'utente può effettuare le seguenti operazioni:

- registrarsi al servizio
- aggiungere un follower
- scrivere un messaggio (eventualmente contenente un hashtag)

## 1.2 Diagramma delle classi UML



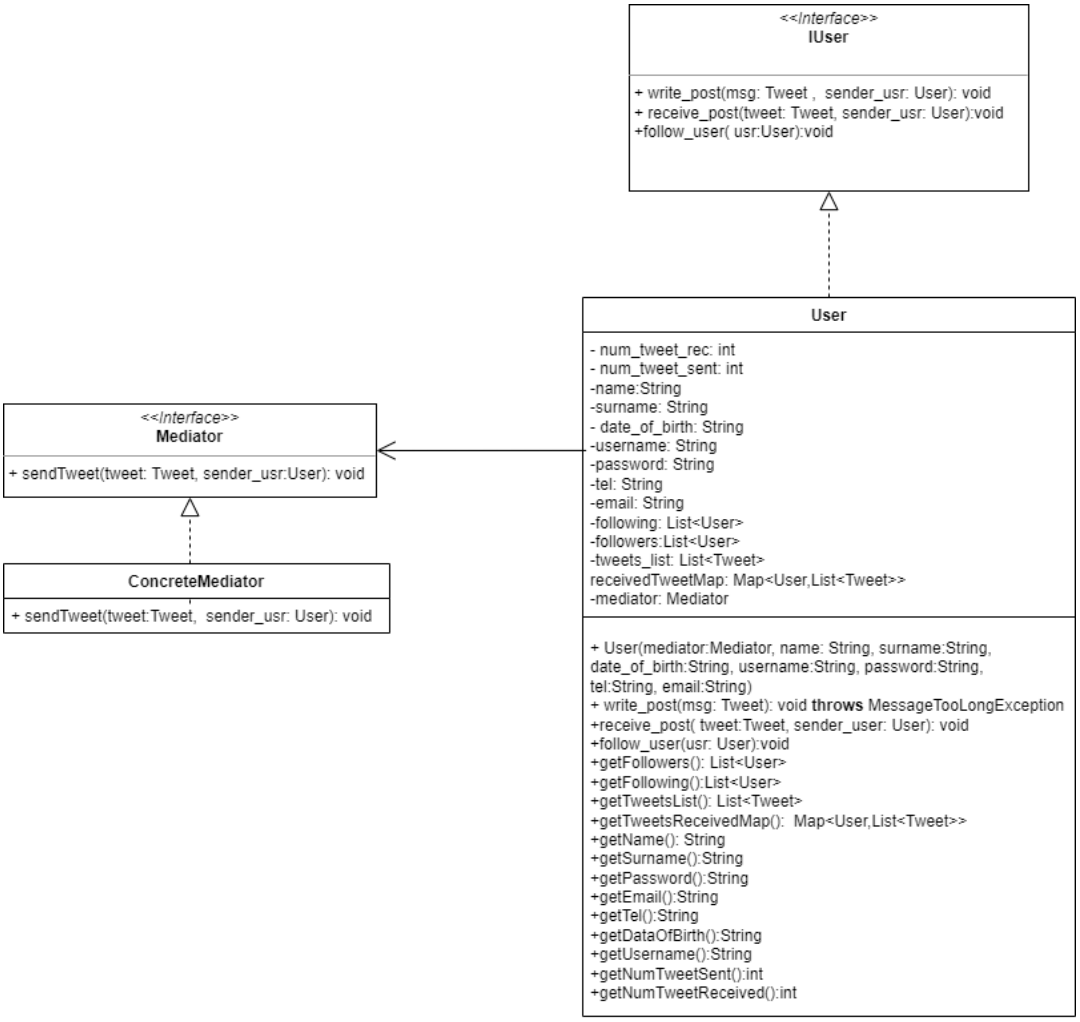



## 1.3 Descrizione Pattern utilizzati

In questa sessione vengono analizzati i pattern impiegati nella realizzazione dei requisiti del progetto.

### 1.3.1 Mediator

Il pattern Mediator è un pattern comportamentale che promuove la comunicazione indiretta tra oggetti, riducendo le dipendenze dirette tra di essi. In questo caso, l'obiettivo del pattern è di consentire agli utenti di scrivere e ricevere tweet senza dover conoscere direttamente gli utenti a cui devono inviare i tweet. Il Mediatore, dunque, agisce come intermediario che gestisce e coordina le interazioni tra gli utenti. Il pattern è stato implementato attraverso le classi Mediator, ConcreteMediator e l'interfaccia IUser insieme alla classe User.



Analizziamo le classi nel dettaglio:

- L'interfaccia Mediator definisce il metodo sendTweet(Tweet tweet, User sender_usr), che verrà implementato dalla classe ConcreteMediator. Questo metodo è responsabile dell'invio di un tweet da parte di un utente e dell'inoltro del tweet a tutti i follower di quell'utente.

- La classe ConcreteMediator implementa l'interfaccia Mediator. Nel suo metodo sendTweet, ottiene la lista dei follower dell'utente che ha scritto il tweet e inoltra il tweet a ciascuno di loro chiamando il metodo receive_post() su ciascun follower.

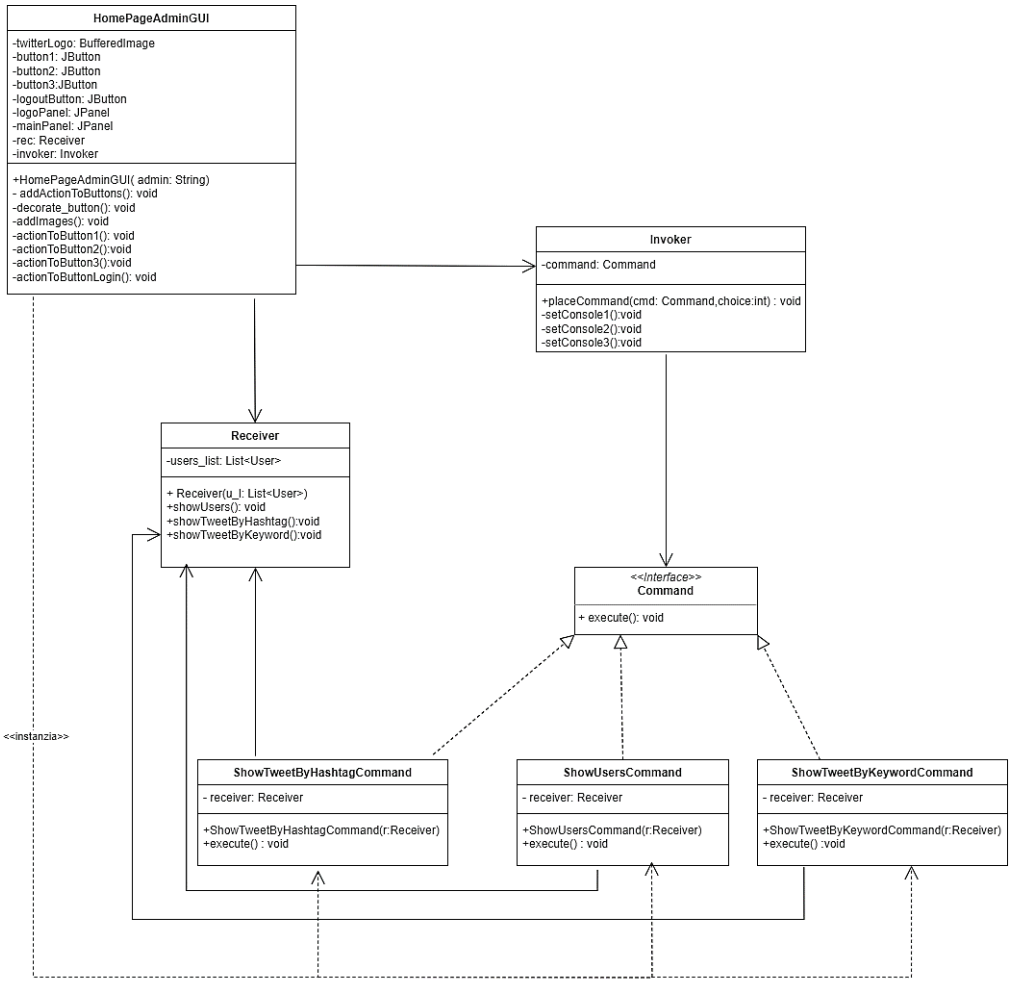
- L'interfaccia IUser definisce i metodi che gli utenti devono implementare. Questi metodi includono write_post(Tweet msg) per scrivere un tweet, receive_post(Tweet tweet, User sender_user) per ricevere un tweet da parte di un altro utente e follow_user(User usr) per seguire un altro utente.

- La classe User implementa l'interfaccia IUser e contiene la logica per scrivere e ricevere tweet, nonché per seguire altri utenti. Ogni utente mantiene un riferimento al Mediatore e utilizza il Mediatore per inviare i tweet ai follower.

L'uso del pattern Mediator in questo contesto aiuta a separare la logica di comunicazione tra gli utenti, consentendo loro di interagire attraverso il Mediatore senza dover avere una conoscenza diretta degli altri utenti. Ciò rende il sistema più flessibile e facilmente estendibile.

### 1.3.2 Command

Il pattern Command è stato implementato al fine di soddisfare opportunamente i comandi messi a disposizione dell'amministratore. Di seguito si illustra il diagramma:




Vediamo nel dettaglio le classi :

- Command : Questa è un'interfaccia che dichiara un metodo execute(), che rappresenta l'azione da eseguire. Le classi concrete che implementano questa interfaccia passano la richiesta al receiver invocando il metodo opportuno a soddisfare quel comando concreto.

- ShowTweetByHashtagCommand : Implementa l'interfaccia Command e rappresenta un comando specifico, ovvero la visualizzazione dei tweet categorizzati per hashtag. Ha un riferimento al Receiver (chi effettuerà l'azione) e implementa il metodo execute(), che chiama il metodo showTweetByHashtag() sul Receiver.

- ShowTweetByKeywordCommand : Implementa Command e rappresenta un altro comando specifico, ovvero la visualizzazione dei tweet contententi una parola chiave presa in input. Ha un riferimento al Receiver e una stringa di input per la parola chiave. Il metodo execute() chiama il metodo showTweetByKeyword() sul Receiver.

- ShowUsersCommand : Simile alle classi sopra, implementa Command e rappresenta un comando per visualizzare, per ogni utente, il numero di tweet ricevuti e inviati. Anch'essa ha un riferimento al Receiver e implementa execute() per chiamare il metodo showUsers() sul Receiver.

- Invoker : Questa classe agisce come invocatore dei comandi. Ha un campo privato Command che viene settato tramite il metodo placeCommand(). A seconda della scelta fornita, imposta un contesto grafico specifico e quindi chiama il metodo execute() sul comando corrente.

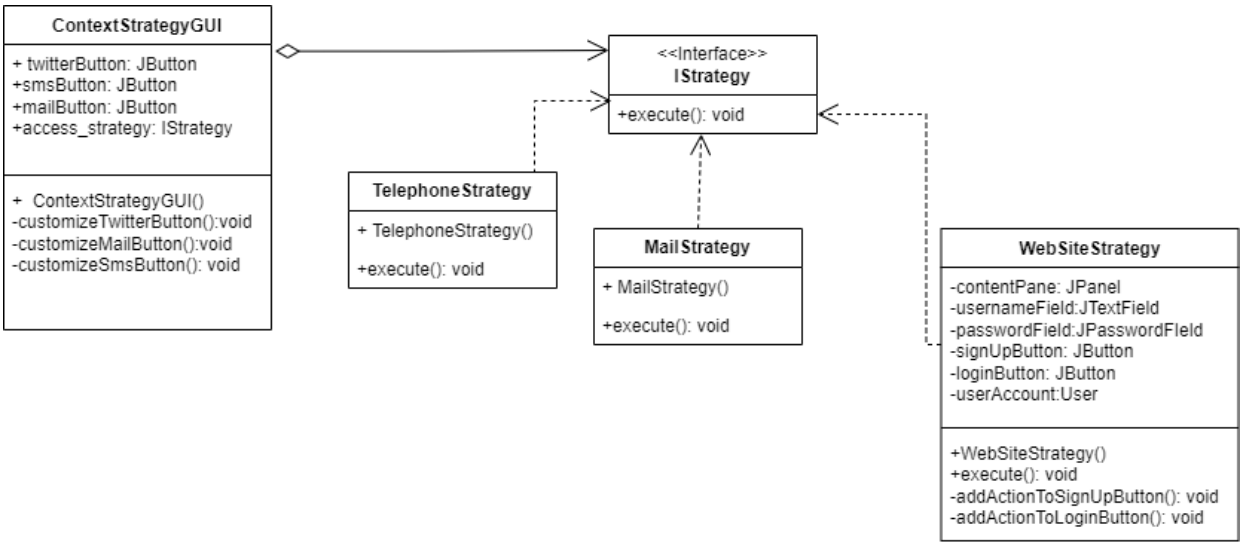
- Receiver : Questa classe rappresenta il destinatario dell'azione. Ha metodi quali showUsers(), showTweetByHashtag(), showTweetByKeyword() che effettuano l'azione reale. È responsabile dell'implementazione delle operazioni richieste dall'amministratore.

- HomePageAdminGUI : Questa classe rappresenta l'interfaccia utente (GUI) in cui l'utente può selezionare l'azione da eseguire. Quando l'utente seleziona un'opzione, viene creato un oggetto comando appropriato e passato all'invocatore (Invoker) per l'esecuzione.

In sostanza, il pattern Command viene utilizzato per incapsulare le richieste dell'admini come oggetti. Questo permette di parametrizzare oggetti con diverse richieste, ritardare l'esecuzione delle richieste, e supportare operazioni di annullamento. Inoltre, l'uso di un invocatore (Invoker) permette di disaccoppiare il mittente dell'azione dal ricevitore, fornendo un modo flessibile di gestire richieste diverse attraverso un'interfaccia comune.

### 1.3.3 Strategy

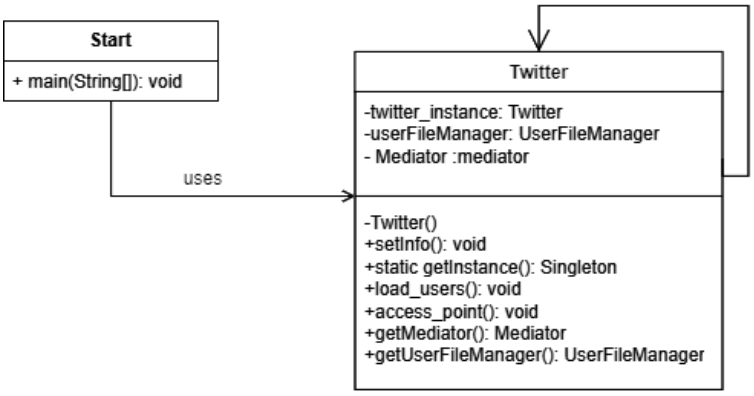
Il pattern Strategy è stato implementato nel codice attraverso le classi MailStrategy, TelephoneStrategy, e WebSiteStrategy, tutte implementano l'interfaccia IStrategy.
Questo pattern consente di definire un insieme di algoritmi (strategie) che possono essere scambiati tra loro senza influire sulla classe che li utilizza.



Ogni classe Strategy rappresenta un modo diverso di eseguire l'azione di postare un tweet. Ciascuna di queste classi implementa il metodo execute(), che è definito nell'interfaccia IStrategy. All'interno di questo metodo, vengono creati gli elementi dell'interfaccia grafica specifica per quella strategia, come campi di testo, pulsanti, etichette, ecc.
La classe ContextStrategyGUI rappresenta il contesto in cui il pattern Strategy viene utilizzato. Essa è responsabile per la creazione e la gestione dei pulsanti che corrispondono alle diverse strategie, ovvero l'accesso a Twitter, l'accesso alla posta e la scrittura di un SMS. Ogni volta che un pulsante viene cliccato, viene creata l'istanza appropriata di una delle classi Strategy e viene chiamato il metodo execute() corrispondente.
Questa implementazione permette di aggiungere nuove strategie senza dover modificare la classe ContextStrategyGUI. Inoltre, ogni strategia è separata e incapsulata, il che facilita il mantenimento, l'estendibilità e la riutilizzabilità del codice.

### 1.3.4 Singleton

Singleton è stato implementato attraverso la classe Twitter utilizzando un approccio di *Eager initialization*. Il pattern Singleton garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a tale istanza.




La classe Twitter contiene twitter_instance che è dichiarato private, static e final. Questo campo è inizializzato solo una volta, all'avvio dell'applicazione. Questo assicura che vi sia una sola istanza di Twitter nell'intera esecuzione del programma.
Il costruttore della classe Twitter è dichiarato private, impedendo così la creazione di nuove istanze al di fuori della classe stessa.
Il metodo statico getInstance() restituisce l'istanza unica di Twitter che è stata creata.
Il metodo load_users() e altri metodi all'interno della classe Twitter possono essere invocati sull'istanza singleton ottenuta tramite getInstance(), garantendo che tutte le operazioni siano eseguite sulla stessa istanza di Twitter.
Si noti che l'approccio di inizializzazione statica del singleton è thread-safe e garantisce che l'istanza venga creata una sola volta, indipendentemente da quante volte venga chiamato il metodo getInstance

# Capitolo 2

## Pratica

Successivamente vengono proposte alcune sessioni per poter illustrare porzioni di codice che mettono in risalto le logiche applicate nella realizzazione del progetto

### 2.1 Parti rilevanti del codice sviluppato

Per quanto riguarda le operazioni dell'amministratore queste vengono implementate mediante il pattern Command, in particolare analizziamo la classe Receiver che si preoccupa dell'effettiva implementazione delle richieste degli amministratori. In sintesi, la classe Receiver ha la responsabilità di gestire e mostrare informazioni relative agli utenti e ai tweet, in base alle diverse operazioni che possono essere richieste dall'amministratore.

Vediamo nel dettaglio cosa fa ciascun metodo della classe:

```
public void showUsers()
{
     for ( User u: Twitter.getInstance().getUserFileManager().getListUsers())
        {
         System.out.println();
         System.out.println(" - Username: "+u.getUsername());
         System.out.println("Ha ricevuto "+u.getNumTweetReceived()+ " tweet ed ha mandato "+ u.getNumTweetSent()+" tweet");
         System.out.println();
        }

}
```

showUsers() si occupa di mostrare l'elenco degli utenti in base al numero di messaggi ricevuti o inviati. Il metodo cicla attraverso la lista degli utenti e stampa informazioni relative a ciascun utente.
Vengono stampati, per ogni utente, username con numero di tweet ricevuti e il numero di tweet inviati.
Le informazioni vengono prese dalla lista degli utenti tramite metodi getter quali getUsername(), getNumTweetReceived() e getNumTweetSent().

```java
public void showTweetByHashtag()
{
     System.out.println();
    for (Map.Entry<String, List<String>> entry :Tweet.hashtags_map.entrySet()) {
        String chiave = entry.getKey();
        List<String> valori = entry.getValue();

        System.out.println("Tweet contenenti  " + chiave+ " : ");

        for (String valore : valori) {
            System.out.println("- " +chiave+" "+ valore);
        }
        System.out.println(); // Riga vuota tra le chiavi
        System.out.println();
    }

}
```

Questo metodo permette di mostrare all'amministratore i messaggi categorizzati per hashtag. Cicla attraverso la mappa di hashtag definita nella classe Tweet (variabile statica) e per ogni hashtag, stampa i tweet associati.
La mappa di hashtag è definita con il modificatore statico nella classe Tweet, in quanto l'obbiettivo è far sì che che tutti i tweet condividono la stessa mappa di hashtag da aggiornare all'evenienza.

```java
    public void showTweetByKeyword(String keyword)
    {  boolean keyCheck=false;
          System.out.println();
           for( User u: users_list)
           {
               for( Tweet t: u.getTweetsList())
               {
                   if (t.getTweet().contains(keyword))
                   {   keyCheck=true;
                       System.out.println(" "+u.getUsername()+" : "+t);
                       System.out.println();
                   }
               }

           }

           if(!keyCheck)
           {
               System.out.println(" la parola  "+keyword +" non è presente in nessun tweet!");
           }
    }
}
```

Questo metodo cerca all'interno dei tweet di ogni utente se c'è una corrispondenza con la parola chiave specificata in input.
Se viene trovata una corrispondenza, il tweet viene stampato insieme al nome dell'utente che l'ha pubblicato.
Se non vengono trovate corrispondenze, viene stampato un messaggio che indica che la parola chiave non è presente in nessun tweet.

Per quanto riguarda gli utenti, abbiamo visto come, attraverso il pattern mediator, incapsuliamo le relazioni di notifica dei tweet.
La classe ConcreteMediator infatti fa da mediatore tra un utente e i suoi follower, notificandoli  ogni qual volta un tweet viene postato . In sintesi,  gestisce la distribuzione dei tweet ai follower dell'utente che ha pubblicato il tweet.

```java
public void sendTweet(Tweet tweet, User sender_usr) {
    Iterator <User> u_iterator= sender_usr.getFollowers().iterator();
    while (u_iterator.hasNext())
    {
         u_iterator.next().receive_post(tweet, sender_usr);
    }
```

Questo metodo è l'implementazione del metodo definito nell'interfaccia Mediator. Riceve come argomenti un oggetto Tweet che rappresenta il messaggio pubblicato da un utente e l'oggetto User che rappresenta l'utente che ha pubblicato il tweet.
Itera attraverso la lista di follower dell'utente che ha pubblicato il tweet (sender_usr). Per ogni follower, chiama il metodo receive_post(tweet, sender_usr) dell'oggetto follower :

```
public void receive_post(Tweet tweet, User sender_user) {
    // aggiorna contatore che tiene traccia del numero di tweet ricevuti
    num_tweet_received++;
    //aggiorna la mappa che tiene traccia, per ogni utente seguito, la lista contenente
    // i messaggi postati
    receivedTweetMap.computeIfAbsent(sender_user, k -> new ArrayList<>()).add(tweet);
}
```

Questo metodo è implementato nella classe User per gestire l'aggiornamento dei tweet ricevuti dagli utenti seguiti. Dunque, il metodo receive_post() implementa la parte del pattern Mediator che consente a un utente di ricevere un tweet da un altro utente.
In altre parole, quando un utente riceve un tweet, il mediatore agisce come intermediario per consegnare il tweet al destinatario corretto attraverso il metodo receive_post che aggiorna il contatore di tweet ricevuti dell'utente corrente (num_tweet_received++) e aggiunge il tweet ricevuto alla mappa receivedTweetMap per tener traccia dei tweet ricevuti. Nella mappa, la chiave è l'utente mittente (sender_user), e il valore è una lista dei tweet inviati da quel mittente. Se l'utente mittente non esiste ancora nella mappa, viene creato un nuovo elemento della mappa con la chiave dell'utente mittente e una lista contenente il tweet.
In questo modo, l'utente ricevente tiene traccia dei tweet che ha ricevuto da ogni mittente attraverso la mappa receivedTweetMap.
Questo approccio centralizzato aiuta a ridurre le dipendenze dirette tra gli utenti e consente al Mediatore di coordinare le interazioni tra di essi in modo più ordinato e scalabile.

Nei requisiti del progetto è anche richiesto di consentire all'utente di aggiungere follower. Quest'operazione è resa possibile all'interno della classe User con il metodo follow_user(User user) :

```
public void follow_user(User usr) {

    following.add(usr); //aggiorno la lista degli utenti seguiti
    usr.getFollowers().add(this); // aggiorno la lista follower dell'utente che ho iniziato a seguire

}
```

Quando un utente decide di seguire un altro utente, chiama il metodo follow_user(usr) passando l'utente che vuole seguire come parametro. Ecco come avviene il processo:

- following.add(usr): Aggiunge l'utente usr alla lista dei seguiti dall'utente corrente. Questo significa che l'utente corrente inizierà a seguire l'utente usr.

- usr.getFollowers().add(this): Aggiunge l'utente corrente alla lista dei follower dell'utente usr. Questo significa che l'utente corrente diventerà un follower dell'utente usr.

In sostanza, questo processo consente di stabilire una relazione di "seguito" tra due utenti. L'utente corrente diventa un follower dell'utente specificato come parametro (usr), e contemporaneamente l'utente specificato (usr) viene aggiunto alla lista dei seguiti dell'utente corrente. In questo modo, la classe User gestisce l'aggiunta di follower e il mantenimento delle relazioni tra gli utenti.

## 2.2 Gestione delle eccezioni

Come sappiamo la pagina personale dell'utente è aggiornabile tramite messaggi di testo con lunghezza massima consentita di 140 caratteri. Tale vincolo si è scelto di gestirlo mediante un eccezione personalizzata.

```
public class MessageTooLongException extends Exception {

    private static final long serialVersionUID = 1L;

    public MessageTooLongException()
    {
        super("Il messaggio può avere una lunghezza massima di 140 caratteri. ");
    }


}
```

La classe MessageTooLongException è una classe di eccezione personalizzata che estende la classe Exception. Questa classe è stata creata per gestire situazioni in cui un messaggio supera la lunghezza massima consentita di 140 caratteri, tipica dei messaggi su piattaforme di social network come Twitter. All'interno della classe, è definito un costruttore che chiama il costruttore della classe madre (Exception) con un messaggio di errore predefinito, che indica che il messaggio può avere al massimo 140 caratteri. Questo messaggio verrà visualizzato quando l'eccezione viene sollevata. In sostanza, questa classe viene utilizzata per identificare e segnalare specificamente quando un messaggio supera la lunghezza massima consentita, consentendo al codice che cattura questa eccezione, presente nella classe HomePaseUserGUI, di gestire la situazione in modo adeguato informando l'utente sulle relative restrizione di lunghezza massima dei messaggi da postare .

```
postButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String newMessage = writeArea.getText();
        //String newHashtag= hashtagField.getText();

        if ( newMessage.length()>0)
        {  Tweet newTweet=new Tweet(newMessage);
         if(hashtagField.getText().length()>0)
           newTweet.setHashtag(hashtagField.getText());
            try {
                userAccount.write_post(newTweet);
                JOptionPane.showMessageDialog(null, ""+" tweet pubblicato !  ");
            }
            catch (MessageTooLongException ex) {
                // Mostriamo una finestra di dialogo con il messaggio di errore
             JOptionPane.showMessageDialog(HomePageUserGUI.this, "Si è verificata un'eccezione !\n " +
                ex.getMessage(), "Errore", JOptionPane.ERROR_MESSAGE);
            }

        }

        else
        {
             JOptionPane.showMessageDialog(HomePageUserGUI.this, " Non è stato possibile pubblicare il tuo tweet."
                    +" \n Assicurati di aver inserito il messaggio", "Errore", JOptionPane.ERROR_MESSAGE);
        }
```

Il codice fornito fa parte della classe HomePaseUserGUI e fa riferimento all'azione di pubblicazione di un nuovo tweet in una GUI. Ecco cosa fa il codice:

- postButton.addActionListener(...): Questo codice sta aggiungendo un listener all'oggetto postButton. Questo listener risponderà all'evento di clic sul pulsante "Post" (pulsante per pubblicare un nuovo tweet).

- public void actionPerformed(ActionEvent e) { ... }: Tutto il codice all'interno di questo metodo sarà eseguito in risposta all'evento di clic sul pulsante "post" .

Riguardo l'eccezione gestita analizziamo la porzione di codice seguente:

if (newMessage.length() > 0) { ... }

Questo controllo verifica se il messaggio inserito ha una lunghezza maggiore di zero, quindi se l'utente ha effettivamente inserito del testo. Se è stato inserito un testo nell'area di testo, verrà creato un nuovo oggetto Tweet con quel testo inserendo eventualmente l'hashtag qualora l'utente l'avesse definito . Dentro il blocco dell'if, c'è un blocco try-catch che gestisce l'eventuale eccezione MessageTooLongException. Questo blocco verrà eseguito quando si tenta di pubblicare il tweet usando il metodo write_post(newTweet) del userAccount. Se viene lanciata un'eccezione MessageTooLongException durante la pubblicazione del tweet, verrà visualizzata una finestra di dialogo di errore con il messaggio dell'eccezione. In sintesi, il codice gestisce l'azione di pubblicazione di un nuovo tweet nell'interfaccia utente grafica, controllando la lunghezza del messaggio e gestendo l'eccezione personalizzata quale MessageTooLongException che può essere lanciate durante la pubblicazione del tweet. Si osserva che all'esterno del blocco try-catch, ci sono anche controlli aggiuntivi nel caso in cui il messaggio inserito fosse vuoto o non valido. In tal caso, verrà visualizzata una finestra di dialogo di errore.

Le operazioni di lettura o scrittura di file possono generare eccezioni di tipo IOException se si verificano problemi durante l'accesso ai file. Infatti, nelle classi AdminFileManager e UserFileManager le parti che coinvolgono l'apertura, la lettura e la scrittura dei file stanno gestendo eccezioni di tipo IOException. Si ricorda che IOException è una classe di eccezioni generale associata a errori di input/output. Vediamo nel dettaglio:

```
private List<User> read_FileUsers()
  {
    // try-with-resource chiusura automatica del file
  try (BufferedReader reader = new BufferedReader(new FileReader(fileNameUser))) {
      String line;
      while ((line = reader.readLine()) != null) {
          String[] parts = line.split("\\s+");
          if (parts.length >= numInfoUser) {
              User usr=new User(Twitter.getInstance().getMediator(),parts[0],parts[1],parts[2],parts[3],parts[4],parts[5],parts[6]);
              users_list.add(usr);
          }
      }
  } catch (IOException e) {
      System.err.println("Errore nella lettura del file: " +fileNameUser+" "+ e.getMessage());
  }

  return users_list;
  }
```

Qui, viene utilizzata un'eccezione IOException nel blocco try con risorse (try with resources). Questo è un costrutto che consente di aprire risorse come file all'interno del blocco try e assicura che le risorse vengano chiuse correttamente alla fine del blocco, indipendentemente dal fatto che si verifichino eccezioni o meno. L'eccezione

IOException catturata viene utilizzata per gestire eventuali errori di lettura dal file.

```
public void updateFileUsers(User user)
{
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileNameUser,true))) {
       writer.write("\n"+user.getName()+" "+user.getSurname()+" "+user.getDataOfBirth()+
                    " "+user.getUsername()+" "+user.getPassword()+ " "+user.getTel()+" "+
                     user.getEmail());

       users_list.add(user);

    } catch (IOException e) {
        System.err.println("Errore nella scrittura sul file: " +fileNameUser+" "+ e.getMessage()); }
}
```

Anche nel metodo updateFileUsers(User user) viene utilizzata un'eccezione IOException all'interno del blocco try con risorse. Questa volta viene utilizzata per gestire eventuali errori durante la scrittura su un file, quando si scrive l'informazione di un nuovo utente nel file fileNameUser. In entrambi i casi, l'eccezione IOException è catturata e il messaggio dell'errore viene visualizzato in output, indicando quale file ha causato l'errore e fornendo ulteriori dettagli sull'errore stesso. Questo aiuta a identificare e risolvere problemi legati all'accesso ai file.  Si osserva che il codice fa riferimento alla classe UserFileManager ma il discorso è analogo per AdminFileManager.
Anche nella classe HomePageAdminGUI viene gestita un eccezione di tipo IOException quando si tenta di caricare l'immagine del logo Twitter. L'eccezione viene catturata e gestita all'interno del blocco try-catch.
Ecco dove avviene la gestione dell'eccezione:

```
private void addImages()
{
    // Carica l'immagine del logo Twitter
    try {
          FileInputStream fis = new FileInputStream("Image/twitter_icon.png");
          twitterLogo = ImageIO.read(fis);
          fis.close();
      } catch (IOException ex) { ex.printStackTrace();}

      // Crea un pannello per l'immagine del logo Twitter
       logoPanel = new JPanel() {
      private static final long serialVersionUID = 1L;

          @Override
          protected void paintComponent(Graphics g) {

              super.paintComponent(g);
              if (twitterLogo != null) {
                  g.drawImage(twitterLogo, 0, 0, getWidth(), getHeight(), this);
              }
              g.setColor(new Color(29,161,242));
          }
      };

      logoPanel.setBackground(new Color(14,153,226));
}
```

In questo frammento di codice, si tenta di aprire il file "Image/twitter_icon.png" utilizzando un oggetto FileInputStream, quindi si tenta di leggere l'immagine utilizzando ImageIO.read(fis). Entrambi questi passaggi potrebbero generare un'eccezione di tipo IOException se si verificano problemi nell'apertura o nella lettura del file. Se l'eccezione viene catturata, il suo tracciato verrà stampato utilizzando ex.printStackTrace() per aiutare nella diagnosi dell'errore. In questo modo, l'applicazione non si interrompe improvvisamente se si verifica un problema nell'apertura o nella lettura dell'immagine, ma invece gestisce l'eccezione e ne mostra i dettagli.

## 2.3 Gestione dell'interfaccia grafica

La registrazione al sito viene implementata nella classe RegistrationUserGUI, la quale crea una finestra di registrazione per un utente attraverso un'interfaccia grafica.
Ecco come il codice gestisce il processo di registrazione dell'utente:

1. Inizializzazione e creazione dell'interfaccia grafica

```
public class RegistrationUserGUI {
    private  boolean registration;
    public RegistrationUserGUI() {
        registration = false;
        JFrame frame = new JFrame("Registrati a Twitter !");
        frame.setIconImage(Toolkit.getDefaultToolkit().getImage("C:\\Users\\Marti\\OneDrive\\Desktop\\Twitter\\Image\\twitter_icon.png"));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel(new GridLayout(8, 2, 10, 10)); // 8 righe, 2 colonne
        panel.setBackground(new Color(14, 153, 226));
        panel.setForeground(Color.white);

        JLabel nameLabel = new JLabel("Nome:");
        nameLabel.setForeground(Color.white);
        nameLabel.setFont(new Font("Century Gothic", Font.BOLD, 12));
        JTextField nameField = new JTextField(20);

        JLabel surnameLabel = new JLabel("Cognome:");
        surnameLabel.setForeground(Color.white);
        surnameLabel.setFont(new Font("Century Gothic", Font.BOLD, 12));
        JTextField surnameField = new JTextField(20);
        JLabel birthDateLabel = new JLabel("Data di Nascita:");
        birthDateLabel.setForeground(Color.white);
        birthDateLabel.setFont(new Font("Century Gothic", Font.BOLD, 12));
        JTextField birthDateField = new JTextField(20);
        birthDateField.addFocusListener(new FocusAdapter() {
            @Override
            public void focusGained(FocusEvent e) {birthDateField.setText(""); } });

        JLabel phoneLabel = new JLabel("Telefono:");
        phoneLabel.setForeground(Color.white);
        phoneLabel.setFont(new Font("Century Gothic", Font.BOLD, 12));
        JTextField phoneField = new JTextField(20);

        JLabel emailLabel = new JLabel("Email:");
        emailLabel.setForeground(Color.white);
        emailLabel.setFont(new Font("Century Gothic", Font.BOLD, 12));
        JTextField emailField = new JTextField(20);

        JLabel usernameLabel = new JLabel("Username:");
        usernameLabel.setForeground(Color.white);
        JTextField usernameField = new JTextField(20);

        JLabel passwordLabel = new JLabel("Password:");
        passwordLabel.setForeground(Color.white);
        passwordLabel.setFont(new Font("Century Gothic", Font.BOLD, 12));
        JPasswordField passwordField = new JPasswordField(20);
```

Viene creato un oggetto JFrame che rappresenta la finestra di registrazione.
Vengono creati diversi elementi grafici come etichette, campi di testo, campi password e un pulsante di registrazione. Infine viene impostato il comportamento di chiusura della finestra quando viene premuta la "X" in alto a destra.

2. Gestione dell'azione del pulsante di registrazione (registerButton):

```
registerButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // gestisci la registrazione dell'utente con i dati inseriti nei campi
        String name = nameField.getText();
        String surname = surnameField.getText();
        String birthDate = birthDateField.getText();
        String phone = phoneField.getText();
        String email = emailField.getText();
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());
        // controlliamo che username sia diverso da altri
        boolean check =true;
          for( User user: Twitter.getInstance().getUserFileManager().getListUsers())
          {   if(username.equals(user.getUsername()))
            { check=false;
              JOptionPane.showMessageDialog(frame, "Esiste già un account con username : "+username, "Errore", JOptionPane.ERROR_MESSAGE);

            }

          }
          if(check)
          {        // verifica che i campi siano compilati
              if ( nameField.getText().length()>0 && usernameField.getText().length()>0
                      && surnameField.getText().length()>0 && birthDateField.getText().length()>0
                      && phoneField.getText().length()>0 && emailField.getText().length()>0
                      && passwordField.getPassword().length>0 )
              {  registration=true; }

              else
              {
                  JOptionPane.showMessageDialog(frame, " Non hai riempito tutti i campi !", "Errore", JOptionPane.ERROR_MESSAGE);
              }


              if(registration)
              {   // azioni di registrazione necessarie
                  User newUser= new User(Twitter.getInstance().getMediator(),name,surname,birthDate,phone,email,username,password);
                  // aggiorna la lista e il file che tengono  traccia degli utenti twitter
                  Twitter.getInstance().getUserFileManager().updateFileUsers(newUser);
                  // Esempio di messaggio di conferma
                  JOptionPane.showMessageDialog(frame, "Registrazione completata!");
                  frame.dispose();
                  new ContextStrategyGUI();
              }
          }
      }
    });
```

Quando l'utente preme il pulsante di registrazione, l'azione associata viene eseguita. Vengono recuperati i dati inseriti dall'utente nei campi di testo e nei campi password. In particolare, viene verificato se l'username inserito è già utilizzato da un altro utente. Se sì, viene mostrato un messaggio di errore e il processo di registrazione si interrompe. Chiaramente si effettua un controllo per garantire che tutti i campi siano stati compilati correttamente. Se tutti i campi sono stati compilati correttamente e l'username non è già in uso, il flag registration viene impostato su true. Se il flag registration è true, viene creato un nuovo oggetto User con i dati inseriti dall'utente e viene mostrato un messaggio di conferma di registrazione completata. L'oggetto User viene quindi aggiunto alla lista di utenti di Twitter tramite l'istruzione :

**Twitter.getInstance().getUserFileManager().updateFileUsers(newUser);**

Dopodiché, la finestra di registrazione viene chiusa e viene creata un'istanza di ContextStrategyGUI, che rappresenta l'interfaccia principale dell'applicazione per l'utente registrato.

Per quanto riguarda la creazione dell'interfaccia grafica:

```
        panel.add(nameLabel);
        panel.add(nameField);
        panel.add(surnameLabel);
        panel.add(surnameField);
        panel.add(birthDateLabel);
        panel.add(birthDateField);
        panel.add(phoneLabel);
        panel.add(phoneField);
        panel.add(emailLabel);
        panel.add(emailField);
        panel.add(usernameLabel);
        panel.add(usernameField);
        panel.add(passwordLabel);
        panel.add(passwordField);
        panel.add(new JLabel()); // Spazio vuoto
        panel.add(registerButton);

        frame.add(panel);
        frame.pack();
        frame.setLocationRelativeTo(null); // Centra la finestra
        frame.setVisible(true);
    }

    
}
```

Vengono creati e posizionati i vari elementi grafici (etichette, campi di testo, campi password, pulsante) all'interno del pannello. Il pannello viene aggiunto alla finestra (frame) e viene chiamato il metodo pack() per adattare le dimensioni della finestra in base al contenuto. Infine la finestra viene centrata sullo schermo e resa visibile.
In sintesi, Registration crea un'interfaccia grafica di registrazione per gli utenti. Quando l'utente compila tutti i campi richiesti e preme il pulsante di registrazione, il codice verifica le informazioni inserite, gestisce la registrazione dell'utente, e in caso di successo mostra un messaggio di conferma e chiude la finestra di registrazione, passando poi all'interfaccia principale dell'applicazione.

In generale, l'interfaccia grafica è stata implementata utilizzando il framework Java Swing. Sono state create diverse classi, ciascuna rappresentante una finestra grafica specifica, e all'interno di queste classi sono stati aggiunti i componenti grafici come bottoni, campi di testo, etichette e aree di testo. Di seguito una descrizione delle principali classi coinvolte nell'implementazione dell'interfaccia grafica.

- HomePageUserGUI: Questa classe rappresenta la finestra della home page dell'utente loggato. È derivata dalla classe JFrame ed è stata organizzata utilizzando il layout manager GroupLayout, che consente di posizionare e allineare

i componenti in modo flessibile all'interno della finestra.

- ContextStrategyGUI: Questa classe rappresenta la finestra di accesso iniziale, dove l'utente può scegliere tra diverse strategie di accesso ( tramite Twitter, posta elettronica o SMS). È anch'essa derivata dalla classe JFrame e contiene bottoni per selezionare le diverse strategie. Ciascun pulsante è associato a un'azione specifica, come l'apertura del browser per accedere a Twitter, l'apertura di una finestra per l'accesso tramite posta elettronica o SMS.

- MailStrategy: Questa classe rappresenta la finestra per l'accesso tramite posta elettronica. Contiene campi di testo per l'indirizzo email, l'hashtag e il testo del tweet, oltre a bottoni per inviare il tweet o tornare alla schermata di accesso iniziale. Quando l'utente invia il tweet, viene verificato se l'indirizzo email è associato a un account Twitter e il tweet viene inviato tramite il metodo write_post dell'oggetto User. La classe TelephoneStrategy funziona in modo analogo con l'unica differenza che rappresenta l'accesso tramite sms e dunque la verifica avviene sul numero telefonico.

- WebSiteStrategy: Rappresenta la finestra per accedere a Twitter. Contiene campi di testo per l'inserimento dell'username e della password. I pulsanti "Iscriviti" e "Accedi" sono decorati e hanno azioni associate per gestire l'accesso o la registrazione. Il metodo execute() imposta i componenti grafici nella finestra.

- HomeScreenGUI: Crea una finestra (JFrame) di benvenuto definendo due pulsanti: "Modalità Utente" e "Modalità Amministratore". I pulsanti sono decorati per dimensioni e stile con il metodo decorate_buttons() mentre le azioni ai pulsanti, per gestire il passaggio alle diverse modalità vengono definite tramite addActionToButton() .

- AdminLoginGUI: Crea una finestra (JFrame) per la pagina di accesso dell'amministratore. Contiene campi di testo per l'inserimento dell'username e della password. Il pulsante "Accedi" viene decorato e ha un'azione associata che verifica le credenziali dell'amministratore. Il metodo launchFrame() imposta la disposizione dei componenti nella finestra.

- HomePageAdminGUI: Crea una finestra che rappresenta la home page dell'amministratore. Contiene diversi pulsanti che consentono di selezionare

diverse opzioni. Il metodo addActionToButtons() aggiunge azioni ai pulsanti per eseguire diverse operazioni. Ricordiamo che si utilizza il pattern Command per implementare un modello di comando per le operazioni selezionate.

Nel complesso, ciascuna classe utilizza metodi Swing per creare e posizionare gli elementi grafici, nonché per gestire le azioni degli utenti. Il codice si basa su componenti Swing come JFrame, JButton, JTextField, JLabel, JPasswordField, JPanel, ecc. Dunque, le classi sfruttano le funzionalità offerte da Swing per creare interfacce utente interattive e visivamente accattivanti e gli oggetti delle classi vengono creati e visualizzati nelle rispettive finestre utilizzando i metodi setVisible(true).

Infine osserviamo l'uso della classe JConsole, sottoclasse di JTextArea, progettata per catturare l'output del flusso System.out e visualizzarlo in una finestra di testo all'interno di un'interfaccia grafica Swing. In questo modo viene fornito un modo per indirizzare l'output standard ( visualizzato sulla console della riga di comando) alla finestra di testo della GUI.

```
public class JConsole extends JTextArea {

    //Prende in input una stringa(quella del System.out), e tale stringa la trasforma in un qualcosa che sia adattabile a JConsole

    private static final long serialVersionUID = 1L;
    private PrintStream printStream;

      public JConsole() {
        printStream = new PrintStream(new ConsolePrintStream());
      }

      public PrintStream getPrintStream() {
        return printStream;
      }

      /** ConsolePrintStream è una classe interna modellata per poterla collegare direttamente ad un qualsiasi PrintStream
       *  oltre al System.out. Dunque è possibile collagarla ad un qualsiasi output di dati.
       */
      private class ConsolePrintStream extends ByteArrayOutputStream {
         // ridefinizione del metodo write della superclasse ByteArrayOutputStream.
        public synchronized void write(byte[] b, int off, int len) {
          setCaretPosition(getDocument().getLength());
          String str = new String(b);
         //ogni flusso di byte scritto sullo stream lo aggiungiamo alla JTextArea tramite il metodo append().
          append(str.substring(off, len));
        }
      }
```

È un esempio di come l'output del sistema può essere intercettato e reindirizzato per essere visualizzato in un'interfaccia utente. Ha un costruttore che crea un nuovo oggetto PrintStream utilizzando una nuova istanza di ConsolePrintStream. Fornisce un metodo getPrintStream() che restituisce il PrintStream creato. La classe contiene una classe interna ConsolePrintStream che estende ByteArrayOutputStream per catturare l'output e reindirizzarlo verso la finestra di testo e sovrascrive il metodo write(byte[] b, int off, int len) di ByteArrayOutputStream per aggiungere il testo all'area di testo (JTextArea).
In sostanza, JConsole offre un modo per catturare e visualizzare l'output in una finestra di testo all'interno dell'interfaccia utente, consentendo di fornire feedback visivo delle operazioni eseguite nell'applicazione.

All'interno della classe Invoker viene utilizzata JConsole.

```
public class Invoker  {
   private  Command command;

/**
public void placeCommand(Command cmd, int choice)
{
    switch (choice) {
        case 1:
            this.command=cmd;
            setConsole1();
            break;

        case 2:
            this.command=cmd;
            setConsole2();
            break;

        case 3:
            this.command=cmd;
            setConsole3();
            break;
    }

    command.execute();

}
```

Il metodo placeCommand(Command cmd, int choice) crea un'istanza di JConsole e reindirizza l'output standard (System.out e System.err) verso l'istanza di JConsole.
A seconda del valore del parametro choice, viene chiamato uno dei metodi setConsole1(), setConsole2() o setConsole3(), ognuno dei quali crea una finestra con un titolo e un bordo appropriati per il tipo di output desiderato. Ad esempio, setConsole1() è definita nel seguente modo:

```
private void setConsole1()
{
     //Interfaccia grafica
     JConsole console = new JConsole();
     console.setEditable(false);
     //Collegamento del System.out alla JConsole
     System.setOut(console.getPrintStream());  //metodi che  ci permettono di dirottare il System.out sull'output implementato
     System.setErr(console.getPrintStream());

    JFrame frame = new JFrame("Twitter Console ");
    TitledBorder t= new TitledBorder("Elenco utenti in base ai messaggi ricevuti e inviati");
    t.setTitleColor(Color.white);
    console.setBorder(t);
    frame.getContentPane().add(new JScrollPane(console));
    frame.setSize(500,400);
    frame.setVisible(true);
    frame.setLocationRelativeTo(null);
    console.setBackground(new Color(14,153,226));
    console.setFont(new Font("Century Gothic", Font.BOLD,12));
    console.setForeground(Color.white);
}
```

Riassumendo possiamo dire che l'output generato dalle operazioni eseguite tramite il modello Command verrà visualizzato all'interno della finestra di testo creata da JConsole.

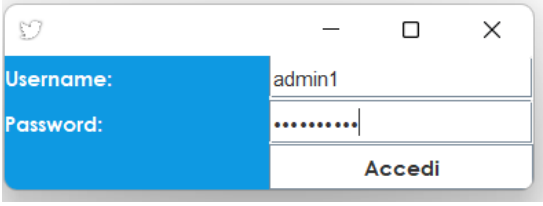
# Capitolo 3

## Illustrazione funzionamento

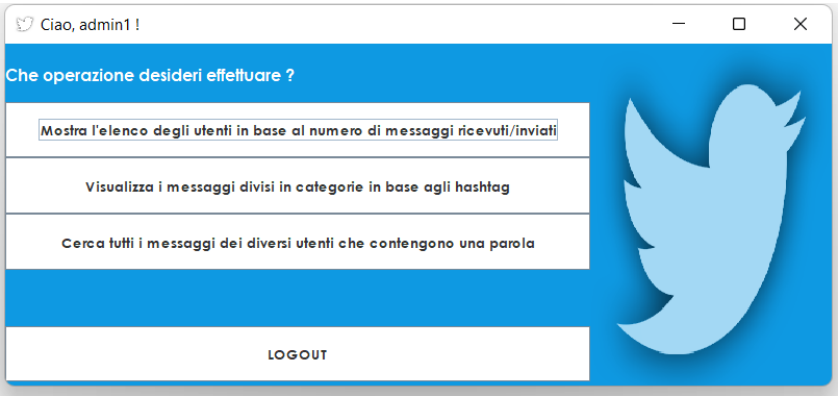
Al fine di verificare il progetto sono stati effettuati numerosi test sulle varie interfacce grafiche implementate. Di seguito vengono riportate le immagini.

### 3.1 Immagini


All'avvio del programma è reso possibile scegliere la modalità di accesso alla piattaforma Twitter.

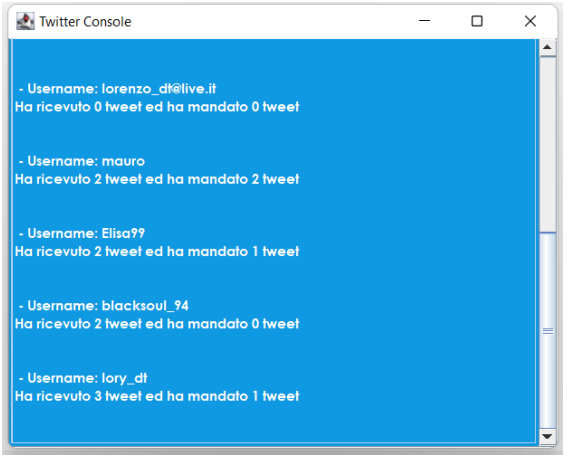



Scegliendo la modalità amministratore verrà aperta una finestra di login per inserire le credenziali d'accesso. Cliccando sul pulsante "Accedi" verranno effettuati opportuni controlli per verificare l'esistenza o meno dell'amministratore che tenta di loggarsi al sito. Qualora l'autenticazione andasse a buon fine viene mostrata la seguente home page degli:

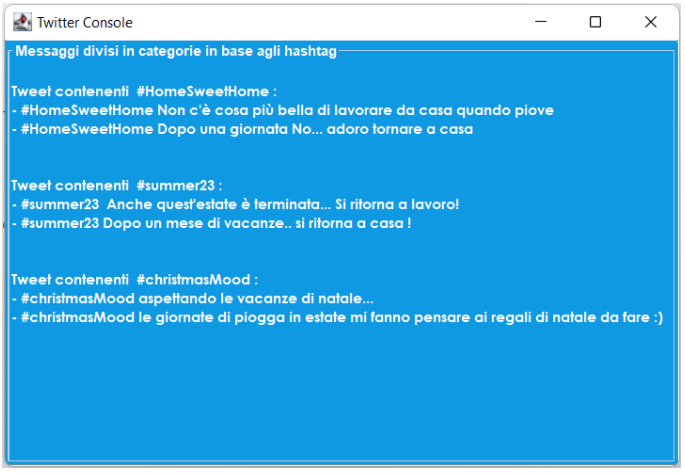


L’amministratore ha la possibilità di selezionare una delle 3 possibili operazioni consentite e, inoltre, può disconnettersi dal social cliccando il pulsante “LOGOUT” in qualsiasi momento . Di seguito riportiamo le immagini relative alle azioni associate ai tre pulsanti:

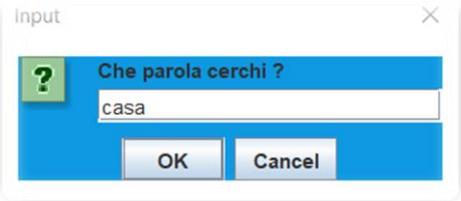
- Azione associata al pulsante
  “Mostra l’elenco degli utenti in base al numero di messaggi ricevuti/inviati” :



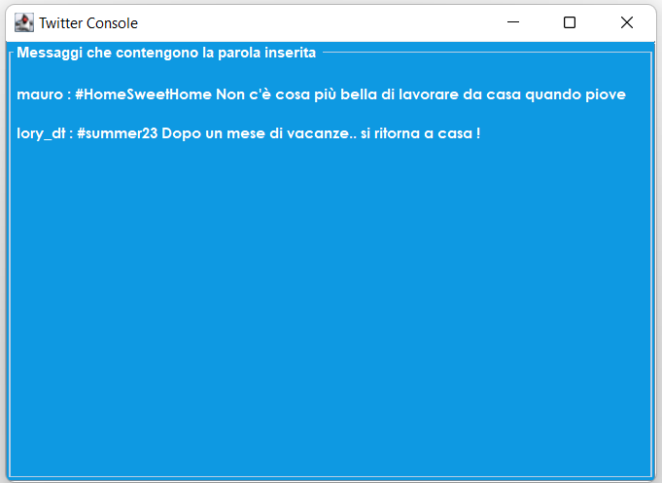

- Azione associata al pulsante “ Visualizza i messaggi divisi in categorie in base agli hashtag” :



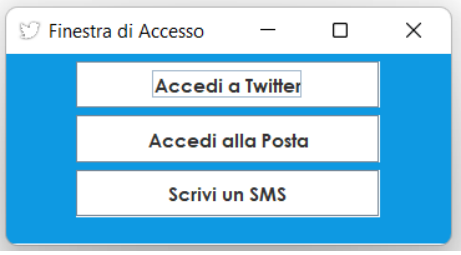

- Azione associata al pulsante
  “Cerca tutti i messaggi dei diversi utenti che contengono una parola”.



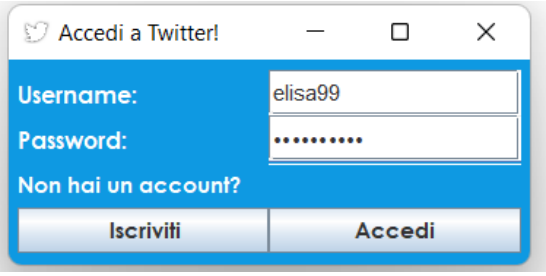
Supponiamo di voler cercare la parola “casa” nei messaggi scritti dai diversi utenti




Cliccando sul pulsante di logout ci si disconnette dal sito e viene riportata la prima schermata raffigurata. Supponiamo stavolta  di scegliere la modalità di accesso Utente, verrà presentata la seguente schermata grafica :

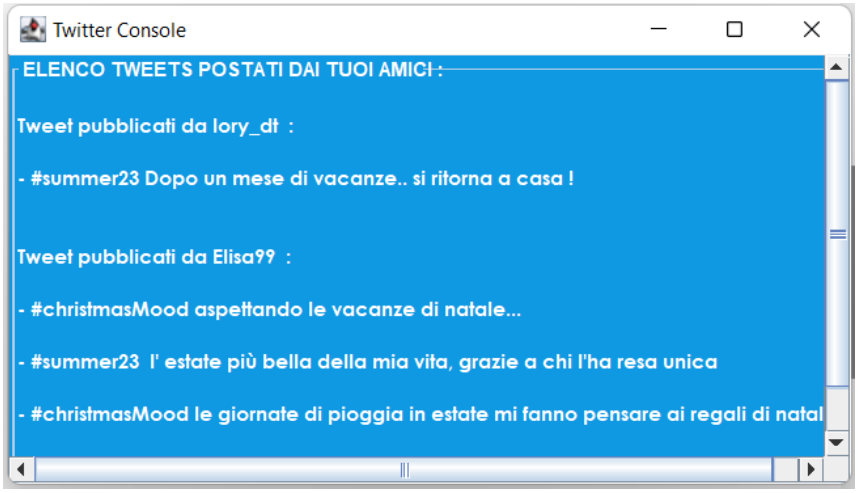



L’utente ha la possibilità di decidere in che modalità accedere alla piattaforma per poter postare i propri tweet. Cliccando sul pulsante “Accedi a twitter” verrà data la possibilità di collegarsi direttamente al sito, infatti a seguito di tale scelta l’output è il seguente:




Anche nella modalità utente si verificherà l’autenticazione delle credenziali inserite. In caso affermativo si accede alla home page dell’utente.


Come vediamo, la pagina personale dell'utente permette di scrivere tweet, aggiungere follower e controllare i tweet postati dai propri amici .

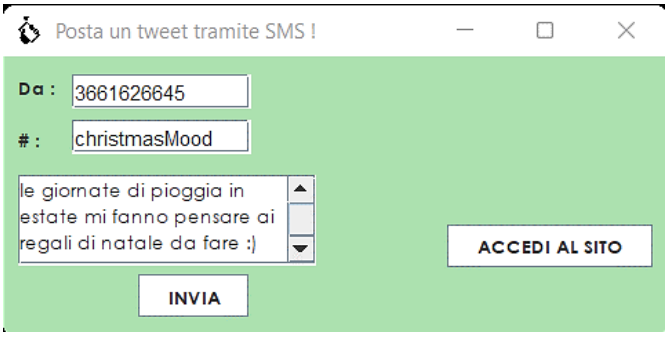



L'utente ha anche la possibilità di vedere la lista dei follower e la lista dei seguiti.
Chiaramente è data la possibilità di disconnettersi, in qualsiasi momento, dalla piattaforma mediante il pulsante "LOGOUT".

Vediamo le altre due modalità di accesso alla piattaforma per aggiornare la propria pagina personale :

- Pubblicazione di un messaggio mediante posta elettronica


- Pubblicazione di un messaggio mediante SMS




Se non si è ancora registrati al sito basta cliccare sul pulsante “Iscriviti” nella finestra di accesso degli utenti e verrà eseguita la seguente schermata in cui l’utente potrà compilare i campi necessari all’iscrizione alla piattaforma.