**CS5800 – Advanced Software Engineering - Final Project**

**Group Names:**

1. **Abdullah Irfan Siddiqui**
2. **Amir Mohideen Basheer Khan**

**Strategy & Observer:**

Github: [Dubs2310/CS5800-Final (github.com)](https://github.com/Dubs2310/CS5800-Final)

Strategy & Observer Class UML Diagram:

A picture containing text, diagram, line, screenshot

Description automatically generated

Strategy & Observer Code:

OpenMeteo.java:

import java.beans.PropertyChangeListener;  
  
public class OpenMeteo extends WeatherObservable implements WeatherService {  
 @Override  
 public void addPropertyChangeListener(PropertyChangeListener listener) {  
 super.addPropertyChangeListener(listener);  
 }  
  
 @Override  
 public void sendWeatherNotification() {  
 int counter = super.getNotificationCounter();  
 super.setLatestNotification(new WeatherNotification("Notification (" + counter + ") received from OpenMeteo"));  
 }  
}

User.java:

import java.util.List;  
  
public class User {  
 private WeatherNotificationFeed feed = new WeatherNotificationFeed();  
 private List<WeatherService> services;  
  
 public User(List<WeatherService> services) {  
 this.services = services;  
 services.forEach(service -> service.addPropertyChangeListener(feed));  
 }  
  
 public WeatherNotificationFeed getFeed() {  
 return feed;  
 }  
  
 public void addWeatherService(WeatherService service) {  
 services.add(service);  
 }  
}

WeatherApi.java:

import java.beans.PropertyChangeListener;  
  
public class WeatherApi extends WeatherObservable implements WeatherService {  
 @Override  
 public void addPropertyChangeListener(PropertyChangeListener listener) {  
 super.addPropertyChangeListener(listener);  
 }  
  
 @Override  
 public void sendWeatherNotification() {  
 int counter = super.getNotificationCounter();  
 super.setLatestNotification(new WeatherNotification("Notification (" + counter + ") received from WeatherApi"));  
 }  
}

WeatherGov.java:

import java.beans.PropertyChangeListener;  
  
public class WeatherGov extends WeatherObservable implements WeatherService {  
 @Override  
 public void addPropertyChangeListener(PropertyChangeListener listener) {  
 super.addPropertyChangeListener(listener);  
 }  
  
 @Override  
 public void sendWeatherNotification() {  
 int counter = super.getNotificationCounter();  
 super.setLatestNotification(new WeatherNotification("Notification (" + counter + ") received from WeatherGov"));  
 }  
}

WeatherNotification.java:

public class WeatherNotification {  
 private String message;  
  
 public WeatherNotification(String message) {  
 this.message = message;  
 }  
  
 public String getMessage() {  
 return message;  
 }  
  
 @Override  
 public String toString() {  
 return message;  
 }  
}

WeatherNotificationFeed.java:

import java.beans.PropertyChangeEvent;  
import java.beans.PropertyChangeListener;  
import java.util.ArrayList;  
import java.util.List;  
  
public class WeatherNotificationFeed implements PropertyChangeListener {  
 private List<WeatherNotification> notifications = new ArrayList<>();  
  
 public List<WeatherNotification> getNotifications() {  
 return notifications;  
 }  
  
 @Override  
 public void propertyChange(PropertyChangeEvent evt) {  
 notifications.add((WeatherNotification) evt.getNewValue());  
 }  
}

WeatherObservable.java:

import java.beans.PropertyChangeListener;  
import java.beans.PropertyChangeSupport;  
  
public abstract class WeatherObservable {  
 private PropertyChangeSupport support = new PropertyChangeSupport(this);  
 private WeatherNotification latestNotification = null;  
 private int notificationCounter = 1;  
  
 public int getNotificationCounter() {  
 return notificationCounter++;  
 }  
  
 public void addPropertyChangeListener(PropertyChangeListener listener) {  
 support.addPropertyChangeListener(listener);  
 }  
  
 public void setLatestNotification(WeatherNotification latestNotification) {  
 support.firePropertyChange("latestNotification", this.latestNotification, latestNotification);  
 this.latestNotification = latestNotification;  
 }  
}

WeatherService.java:

import java.beans.PropertyChangeListener;  
  
public interface WeatherService {  
 void addPropertyChangeListener(PropertyChangeListener listener);  
 void sendWeatherNotification();  
}

Main.java:

import java.util.Arrays;  
  
public class Main {  
 public static void main(String[] args) {  
 WeatherService api = new WeatherApi();  
 WeatherService gov = new WeatherGov();  
 WeatherService openMeteo = new OpenMeteo();  
  
 User user1 = new User(Arrays.*asList*(api, gov, openMeteo));  
 User user2 = new User(Arrays.asList(api, gov));  
 User user3 = new User(Arrays.asList(openMeteo));  
  
 api.sendWeatherNotification();  
 api.sendWeatherNotification();  
 gov.sendWeatherNotification();  
 openMeteo.sendWeatherNotification();  
 openMeteo.sendWeatherNotification();  
  
 System.out.println("User 1's Feed:");  
 user1.getFeed().getNotifications().forEach(System.out::println);  
  
 System.out.println("\nUser 2's Feed:");  
 user2.getFeed().getNotifications().forEach(System.out::println);  
  
 System.out.println("\nUser 3's Feed:");  
 user3.getFeed().getNotifications().forEach(System.out::println);  
 }  
}

Output:

A screenshot of a computer screen

Description automatically generated with medium confidence

**Command & Template:**

Github: [Dubs2310/CS5800-Final (github.com)](https://github.com/Dubs2310/CS5800-Final)

Command & Template UML Diagram (version 2):

Shifted construction of request handlers from main to class called DigitalAssistantApp

Main is instead responsible for creation of instance of DigitalAssistantApp and appropriate method calls.

A picture containing text, diagram, line, plan

Description automatically generated

Command & Template Code:

DigitalAssistantApp.java:

public class DigitalAssistantApp {  
 private RequestHandlerTemplate phoneCall = new PhoneCallRequestHandler();  
 private RequestHandlerTemplate sendEmail = new SendEmailRequestHandler();  
 private RequestHandlerTemplate playMusic = new PlayMusicRequestHandler();  
 private RequestHandlerTemplate setReminder = new SetReminderRequestHandler();  
  
 public void makePhoneCall(String phoneNumber) {  
 phoneCall.processRequest(phoneNumber);  
 System.*out*.println();  
 }  
  
 public void sendEmail(String email) {  
 sendEmail.processRequest(email);  
 System.*out*.println();  
 }  
  
 public void playMusic(String music) {  
 playMusic.processRequest("Jazz");  
 System.*out*.println();  
 }  
  
 public void setReminder(String time) {  
 setReminder.processRequest("6:00 am");  
 System.*out*.println();  
 }  
}

EmailSenderReceiver.java:

public class EmailSenderReceiver {  
 public void sendEmail(String input) {  
 System.*out*.println("Sending an email to " + input);  
 }  
}

MusicPlayerReceiver.java:

public class MusicPlayerReceiver {  
 public void playMusic(String input) {  
 System.*out*.println("Playing some " + input + " music");  
 }  
}

PhoneCallRequest.java:

public class PhoneCallRequest implements Request {  
 private PhoneDialerReceiver receiver;  
  
 public PhoneCallRequest(PhoneDialerReceiver receiver) {  
 this.receiver = receiver;  
 }  
  
 @Override  
 public void execute(String input) {  
 receiver.makePhoneCall(input);  
 }  
}

PhoneCallRequestHandler.java:

public class PhoneCallRequestHandler extends RequestHandlerTemplate {  
 private PhoneDialerReceiver receiver;  
  
 public PhoneCallRequestHandler() {  
 this.receiver = new PhoneDialerReceiver();  
 }  
  
 @Override  
 public boolean validateInput(String input) {  
 System.*out*.println("Validating the following phone number: " + input);  
 return true;  
 }  
  
 @Override  
 public void logRequest(String input) {  
 System.*out*.println("Logging the phone call to: " + input);  
 }  
  
 @Override  
 public void notifyUser(String input) {  
 System.*out*.println("Notifying the user that a phone call was made to: " + input);  
 }  
  
 @Override  
 public void processRequest(String input) {  
 super.processRequest(new PhoneCallRequest(receiver), input);  
 }  
}

PhoneDialerReceiver.java:

public class PhoneDialerReceiver {  
 public void makePhoneCall(String input) {  
 System.*out*.println("Making a phone call to " + input);  
 }  
}

PlayMusicRequest.java:

public class PlayMusicRequest implements Request {  
 private MusicPlayerReceiver receiver;  
  
 public PlayMusicRequest(MusicPlayerReceiver receiver) {  
 this.receiver = receiver;  
 }  
  
 @Override  
 public void execute(String input) {  
 receiver.playMusic(input);  
 }  
}

PlayMusicRequestHandler.java:

public class PlayMusicRequestHandler extends RequestHandlerTemplate {  
 private MusicPlayerReceiver receiver;  
  
 public PlayMusicRequestHandler() {  
 this.receiver = new MusicPlayerReceiver();  
 }  
  
 @Override  
 public boolean validateInput(String input) {  
 System.*out*.println("Validating the following music selection: " + input);  
 return true;  
 }  
  
 @Override  
 public void logRequest(String input) {  
 System.*out*.println("Logging the request to play the following music: " + input);  
 }  
  
 @Override  
 public void notifyUser(String input) {  
 System.*out*.println("Notifying the user that following music was played: " + input);  
 }  
  
 @Override  
 public void processRequest(String input) {  
 super.processRequest(new PlayMusicRequest(receiver), input);  
 }  
}

ReminderSetterReciever.java:

public class ReminderSetterReceiver {  
 public void setReminder(String input) {  
 System.*out*.println("Setting a reminder for " + input);  
 }  
}

Request.java:

public interface Request {  
 void execute(String input);  
}

RequestHandlerTemplate.java:

public abstract class RequestHandlerTemplate {  
 public abstract boolean validateInput(String input);  
 public abstract void logRequest(String input);  
 public abstract void notifyUser(String input);  
 public abstract void processRequest(String input);  
  
 public void processRequest(Request request, String input) {  
 validateInput(input);  
 logRequest(input);  
 request.execute(input);  
 notifyUser(input);  
 }  
}

SendEmailRequest.java:

public class SendEmailRequest implements Request {  
 private EmailSenderReceiver receiver;  
  
 public SendEmailRequest(EmailSenderReceiver receiver) {  
 this.receiver = receiver;  
 }  
  
 @Override  
 public void execute(String input) {  
 receiver.sendEmail(input);  
 }  
}

SendEmailRequestHandler.java:

public class SendEmailRequestHandler extends RequestHandlerTemplate {  
 private EmailSenderReceiver receiver;  
  
 public SendEmailRequestHandler() {  
 this.receiver = new EmailSenderReceiver();  
 }  
  
 @Override  
 public boolean validateInput(String input) {  
 System.*out*.println("Validating the email sent to: " + input);  
 return true;  
 }  
  
 @Override  
 public void logRequest(String input) {  
 System.*out*.println("Logging the email sent to: " + input);  
 }  
  
 @Override  
 public void notifyUser(String input) {  
 System.*out*.println("Notifying the user that an email was sent to: " + input);  
 }  
  
 @Override  
 public void processRequest(String input) {  
 super.processRequest(new SendEmailRequest(receiver), input);  
 }  
}

SetReminderRequest.java:

public class SetReminderRequest implements Request {  
 private ReminderSetterReceiver receiver;  
  
 public SetReminderRequest(ReminderSetterReceiver receiver) {  
 this.receiver = receiver;  
 }  
  
 @Override  
 public void execute(String input) {  
 receiver.setReminder(input);  
 }  
}

SetReminderRequestHandler.java

public class SetReminderRequestHandler extends RequestHandlerTemplate {  
 private ReminderSetterReceiver receiver;  
  
 public SetReminderRequestHandler() {  
 this.receiver = new ReminderSetterReceiver();  
 }  
  
 @Override  
 public boolean validateInput(String input) {  
 System.*out*.println("Validating the reminder set for: " + input);  
 return true;  
 }  
  
 @Override  
 public void logRequest(String input) {  
 System.*out*.println("Logging the reminder set at: " + input);  
 }  
  
 @Override  
 public void notifyUser(String input) {  
 System.*out*.println("Notifying the user that a reminder was set for: " + input);  
 }  
  
 @Override  
 public void processRequest(String input) {  
 super.processRequest(new SetReminderRequest(receiver), input);  
 }  
}

Main.java:

public class Main {  
 public static void main(String[] args) {  
 DigitalAssistantApp assistant = new DigitalAssistantApp();  
 assistant.makePhoneCall("943-324-6456");  
 assistant.sendEmail("johndoe@gmail.com");  
 assistant.playMusic("Jazz");  
 assistant.setReminder("6:00 am");  
 }  
}

Output:

A screenshot of a computer

Description automatically generated with medium confidence

**Twitter:**

**Github:** [Dubs2310/CS5800-Final (github.com)](https://github.com/Dubs2310/CS5800-Final)

Twitter UML Diagram:

A picture containing text, diagram, plan, technical drawing

Description automatically generated

Twitter Code:

AuthStrategy.java:

public interface AuthStrategy {

    void authenticate(User user);

}

BasicStrategy.java:

public class BasicAuth implements AuthStrategy {

    @Override

    public void authenticate(User user) {

        System.out.println(user.getUsername() + " authenticated with basic strategy!");

    }

}

Client.java:

public class Client {

    private String loggedInUsername = null;

    private Server server = Server.getInstance();

    private TweetBuilder builder = null;

    public void authenticate(Credentials credentials, AuthStrategy strategy) {

        loggedInUsername = server.authenticate(credentials, strategy);

    }

    public void signOut() {

        loggedInUsername = null;

        builder = null;

    }

    public void startNewTweet() {

        if (loggedInUsername == null) return;

        builder = new TweetBuilder(loggedInUsername);

    }

    public void setContent(String content) {

        if (loggedInUsername == null) return;

        if (builder == null) return;

        builder.setContent(content);

    }

    public void attachImage(String imageName) {

        if (loggedInUsername == null) return;

        if (builder == null) return;

        builder.attachImage(imageName);

    }

    public void attachVideo(String videoName) {

        if (loggedInUsername == null) return;

        if (builder == null) return;

        builder.attachVideo(videoName);

    }

    public void attachAudio(String audioName) {

        if (loggedInUsername == null) return;

        if (builder == null) return;

        builder.attachAudio(audioName);

    }

    public void attachFile(String fileName) {

        if (loggedInUsername == null) return;

        if (builder == null) return;

        builder.attachFile(fileName);

    }

    public void postTweet() {

        if (loggedInUsername == null) return;

        if (builder == null) return;

        Tweet tweetToPost = builder.build();

        server.postTweet(tweetToPost);

        builder = null;

    }

    public void followUserByUsername(String username) {

        if (loggedInUsername == null) return;

        server.followUserByUsername(loggedInUsername, username);

    }

    public void followHashTagByName(String hashTagName) {

        if (loggedInUsername == null) return;

        server.followHashTagByName(loggedInUsername, hashTagName);

    }

}

Credentials.java:

public class Credentials {

    private String username;

    private String password;

    public Credentials(String username, String password) {

        this.username = username;

        this.password = password;

    }

    public String getUsername() {

        return username;

    }

    public String getPassword() {

        return password;

    }

}

Database.java:

import java.util.ArrayList;

import java.util.List;

public class Database {

    private List<User> users = new ArrayList<>();

    private List<HashTag> hashTags = new ArrayList<>();

    private static Database instance = null;

    private Database() {}

    public static Database getInstance() {

        if (instance == null)

            instance = new Database();

        return instance;

    }

    public void addUser(User user) {

        users.add(user);

    }

    public void addHashTag(HashTag hashTag) {

        hashTags.add(hashTag);

    }

    public User getUserByCredentials(Credentials credentials) {

        for (User user: users)

            if (user.matchCredentials(credentials))

                return user;

        System.out.println("Incorrect Username or Password!");

        return null;

    }

    public User getUserByUsername(String senderUsername) {

        for (User user: users)

            if (user.getUsername().equals(senderUsername))

                return user;

        return null;

    }

    public HashTag getOrCreateHashTagByName(String hashTagName) {

        for (HashTag hashTag: hashTags)

            if (hashTag.getName().equals(hashTagName))

                return hashTag;

        HashTag hashTag = new HashTag(hashTagName);

        addHashTag(hashTag);

        return hashTag;

    }

}

Feed.java:

import java.beans.PropertyChangeEvent;

import java.beans.PropertyChangeListener;

import java.util.ArrayList;

import java.util.List;

public class Feed implements PropertyChangeListener {

    private List<Tweet> tweets = new ArrayList<>();

    public List<Tweet> getTweets() {

        return tweets;

    }

    @Override

    public void propertyChange(PropertyChangeEvent evt) {

        tweets.add((Tweet) evt.getNewValue());

    }

}

Hashtag.java:

public class HashTag extends TweetObservable {

    private String name;

    public HashTag(String name) {

        this.name = name;

    }

    public String getName() {

        return name;

    }

}

Media.java:

public class Media {

    private String mediaName;

    private MediaType media;

    public Media(String mediaName, MediaType media) {

        this.mediaName = mediaName;

        this.media = media;

    }

}

MediaType.java:

public enum MediaType {

    IMAGE,

    VIDEO,

    AUDIO,

    FILE

}

OAuth.java:

public class OAuth implements AuthStrategy {

    @Override

    public void authenticate(User user) {

        System.out.println(user.getUsername() + " authenticated with OAuth strategy!");

    }

}

Server.java:

import java.util.HashMap;

import java.util.List;

public class Server {

    private Database database = Database.getInstance();

    private static Server instance = null;

    private Server() {}

    public static Server getInstance() {

        if (instance == null)

            instance = new Server();

        return instance;

    }

    public String authenticate(Credentials credentials, AuthStrategy strategy) {

        User user = database.getUserByCredentials(credentials);

        if (user == null || !user.authenticate(strategy)) return null;

        return user.getUsername();

    }

    public void postTweet(Tweet tweetToPost) {

        String senderUsername = tweetToPost.getSenderUsername();

        User user = database.getUserByUsername(senderUsername);

        user.setLatestTweet(tweetToPost);

        user.addPost(tweetToPost);

        List<String> hashTagNames = tweetToPost.getHashTagNames();

        for (String hashTagName: hashTagNames) {

            HashTag hashTag = database.getOrCreateHashTagByName(hashTagName);

            hashTag.setLatestTweet(tweetToPost);

        }

    }

    public void followUserByUsername(String loggedInUsername, String username) {

        if (loggedInUsername.equals(username)) return;

        User loggedInUser = database.getUserByUsername(loggedInUsername);

        User userToFollow = database.getUserByUsername(username);

        loggedInUser.follow(userToFollow);

    }

    public void followHashTagByName(String loggedInUsername, String hashTagName) {

        User loggedInUser = database.getUserByUsername(loggedInUsername);

        HashTag hashTag = database.getOrCreateHashTagByName(hashTagName);

        loggedInUser.follow(hashTag);

    }

}

Tweet.java:

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class Tweet {

    private String senderUsername;

    private String timestamp;

    private String content;

    private List<Media> attachments;

    public Tweet(String senderUsername, String content, List<Media> attachments) {

        this.senderUsername = senderUsername;

        this.timestamp = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date());

        this.content = content;

        this.attachments = attachments;

    }

    public String getSenderUsername() {

        return senderUsername;

    }

    public List<String> getHashTagNames() {

        Pattern pattern = Pattern.compile("(?:\\s|^)(?:#(?!(?:\\d+|\\w+?\_|\_\\w\*?)(?:\\s|$)))(\\w+)(?=\\s|$)");

        Matcher matcher = pattern.matcher(content);

        List<String> hashTagNames = new ArrayList<>();

        while (matcher.find())

            hashTagNames.add(matcher.group().trim());

        return hashTagNames;

    }

    @Override

    public String toString() {

        return "Tweet by \"" + senderUsername + "\":\n" +

                    "\tPosted at " + timestamp + '\n' +

                    "\tContent: " + content + '\n';

    }

}

TweetBuilder.java:

import java.util.ArrayList;

import java.util.List;

public class TweetBuilder {

    private String senderUsername;

    private String content = "";

    private List<Media> attachments = new ArrayList<>();

    public TweetBuilder(String senderUsername) {

        this.senderUsername = senderUsername;

    }

    public TweetBuilder setContent(String content) {

        this.content = content;

        return this;

    }

    public TweetBuilder attachImage(String imageName) {

        attachments.add(new Media(imageName, MediaType.IMAGE));

        return this;

    }

    public TweetBuilder attachVideo(String videoName) {

        attachments.add(new Media(videoName, MediaType.VIDEO));

        return this;

    }

    public TweetBuilder attachAudio(String audioName) {

        attachments.add(new Media(audioName, MediaType.AUDIO));

        return this;

    }

    public TweetBuilder attachFile(String fileName) {

        attachments.add(new Media(fileName, MediaType.FILE));

        return this;

    }

    public Tweet build() {

        return new Tweet(

                senderUsername,

                content,

                attachments

        );

    }

}

TweetObservable.java:

import java.beans.PropertyChangeListener;

import java.beans.PropertyChangeSupport;

public abstract class TweetObservable {

    private PropertyChangeSupport support = new PropertyChangeSupport(this);

    private Tweet latestTweet = null;

    public void addPropertyChangeListener(PropertyChangeListener listener) {

        support.addPropertyChangeListener(listener);

    }

    public void setLatestTweet(Tweet latestTweet) {

        support.firePropertyChange("latestTweet", this.latestTweet, latestTweet);

        this.latestTweet = latestTweet;

    }

}

TwoFactorAuth.java:

public class TwoFactorAuth implements AuthStrategy {

    @Override

    public void authenticate(User user) {

        System.out.println(user.getUsername() + " authenticated with 2FA strategy!");

    }

}

User.java:

import java.util.ArrayList;

import java.util.List;

public class User extends TweetObservable {

    private String username;

    private String password;

    private List<AuthStrategy> enabledStrategies;

    private List<Tweet> posts = new ArrayList<>();

    private List<TweetObservable> observables = new ArrayList<>();

    private Feed feed = new Feed();

    public User(String username, String password, List<AuthStrategy> enabledStrategies) {

        this.username = username;

        this.password = password;

        this.enabledStrategies = enabledStrategies;

    }

    public String getUsername() { return username; }

    public List<Tweet> getPosts() { return posts; }

    public Feed getFeed() { return feed; }

    public boolean matchCredentials(Credentials credentials) {

        return credentials.getUsername().equals(username) && credentials.getPassword().equals(password);

    }

    public boolean authenticate(AuthStrategy strategy) {

        if (enabledStrategies.contains(strategy)) {

            strategy.authenticate(this);

            return true;

        }

        System.out.println("User \"" + username + "\" has not enabled the following strategy: " + strategy.getClass().getSimpleName());

        return false;

    }

    public void addPost(Tweet post) {

        posts.add(post);

    }

    public void follow(TweetObservable observable) {

        if (observables.contains(observable)) return;

        observable.addPropertyChangeListener(feed);

        observables.add(observable);

    }

}

Main.java:

import java.util.Arrays;

import java.util.List;

public class Main {

    public static void main(String[] args) {

        AuthStrategy basic = new BasicAuth();

        AuthStrategy oauth = new OAuth();

        AuthStrategy \_2FA = new TwoFactorAuth();

        Database database = Database.getInstance();

        database.addUser(new User("abdullahi", "passwordOne", Arrays.asList(basic, oauth, \_2FA)));

        database.addUser(new User("amirm", "passwordTwo", List.of(basic)));

        database.addUser(new User("jahinm", "passwordThree", List.of(oauth)));

        database.addUser(new User("yousufm", "passwordFour", List.of(\_2FA)));

        Client client = new Client();

        client.authenticate(new Credentials("amirm", "passwordTwo"), oauth);

        client.authenticate(new Credentials("abdullahi", ""), basic);

        client.authenticate(new Credentials("abdullahi", "passwordOne"), \_2FA);

        Client client2 = new Client();

        client2.authenticate(new Credentials("amirm", "passwordTwo"), basic);

        client2.followUserByUsername("abdullahi");

        Client client3 = new Client();

        client3.authenticate(new Credentials("jahinm", "passwordThree"), oauth);

        client3.followHashTagByName("#CSE");

        client.startNewTweet();

        client.setContent("Testing... #CSE #SWE #DESIGN\_PATTERNS");

        client.attachImage("image.png");

        client.attachVideo("video.mp4");

        client.attachAudio("audio.mp3");

        client.attachFile("file.txt");

        client.postTweet();

        client.startNewTweet();

        client.setContent("Testing2... #SWE #DESIGN\_PATTERNS");

        client.attachImage("image.png");

        client.attachVideo("video.mp4");

        client.attachAudio("audio.mp3");

        client.attachFile("file.txt");

        client.postTweet();

        System.out.println();

        System.out.println("\"abdullahi\" Posts:");

        database.getUserByUsername("abdullahi").getPosts().forEach(System.out::println);

        System.out.println("\"abdullahi\" Feed:");

        database.getUserByUsername("abdullahi").getFeed().getTweets().forEach(System.out::println);

        System.out.println("\"amirm\" Posts:");

        database.getUserByUsername("amirm").getPosts().forEach(System.out::println);

        System.out.println("\"amirm\" Feed:");

        database.getUserByUsername("amirm").getFeed().getTweets().forEach(System.out::println);

        System.out.println("\"jahinm\" Posts:");

        database.getUserByUsername("jahinm").getPosts().forEach(System.out::println);

        System.out.println("\"jahinm\" Feed:");

        database.getUserByUsername("jahinm").getFeed().getTweets().forEach(System.out::println);

    }

}

Output:

A screenshot of a computer

Description automatically generated