Assignment 10%

Part 1:

You are responsible for designing an application for notifying the followers of Twitter account, about the new tweets. A followers can follow multiple accounts. A followers can unfollow an account.

ou are responsible for designing an API for notifying the Followers of Twitter accounts, about new tweets. A Follower can follow multiple accounts.

A Follower can unfollow an account. Your solution should enable easy extension to future Follower user interfaces. Make sure to include UML

notes wherever it is necessary

ou are responsible for designing an API for notifying the Followers of Twitter accounts, about new tweets. A Follower can follow multiple accounts.

A Follower can unfollow an account. Your solution should enable easy extension to future Follower user interfaces. Make sure to include UML

notes wherever it is necessary

ou are responsible for designing an API for notifying the Followers of Twitter accounts, about new tweets. A Follower can follow multiple accounts.

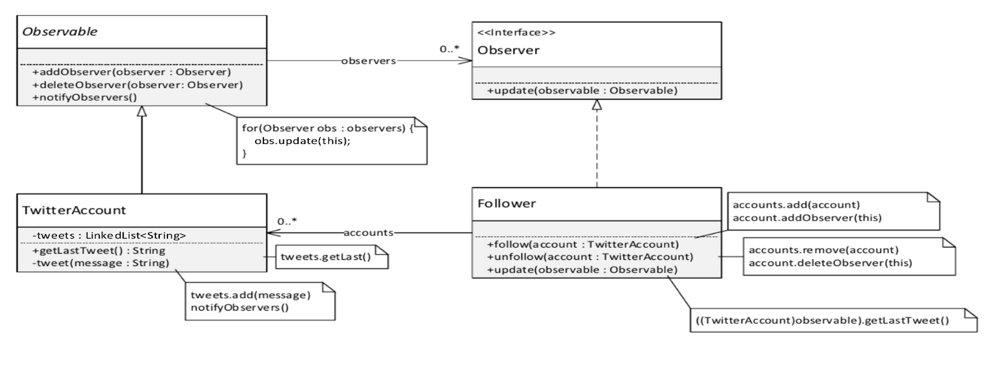
A Follower can unfollow an account. Your solution should enable easy extension to future Follower user interfaces. Make sure to include UML

notes wherever it is necessary

You are responsible for designing an API for notifying the Followers of Twitter accounts, about new tweets. A Follower can follow multiple accounts.

A Follower can unfollow an account. Your solution should enable easy extension to future Follower user interfaces. Make sure to include UML

notes wherever it is necessary.

****

public static void main(String[] args) {  
 Observable account1 = new TwitterAccount();  
 Observable account2 = new TwitterAccount();  
 Observable account3 = new TwitterAccount();  
  
 Observer follower1 = new Follower();  
 Observer follower2 = new Follower();  
 Observer follower3 = new Follower();  
  
 ((Follower) follower1).follow((TwitterAccount) account1);  
 ((Follower) follower2).follow((TwitterAccount) account2);  
 ((Follower) follower3).follow((TwitterAccount) account3);  
  
 account1.tweet("account1's first tweet");  
 account2.tweet("account2's first tweet");  
 account3.tweet("account3's first tweet");  
  
 ((Follower) follower1).follow((TwitterAccount) account2);  
 account2.tweet("I am account02 Follow me!!!");  
   
 ((Follower) follower1).unfollow((TwitterAccount) account2);  
 account2.tweet("I am account02 Why unfollow me???");  
}

**Text

Description automatically generated**

Now account2 has 2 followers. When account02 tweets a message

follower1, follwer2 updated message.

Follower1 unfollows account2.

**Part 2 :**

Implementation of Creational Design Patterns: Singleton Pattern.

The Singleton Pattern proposes solutions for classes that demand to be instantiated only once

throughout the whole application time and gives a universal access to that object. Think where you can

apply singleton design pattern on your selected game scenario.

For e.g., you have selected candy crush.

The application records your current level, your achieved score, the candies you collected etc. which

means one universal history per user.

Think of any such game and apply Singleton Design pattern. Make a class diagram first and then

implement it.

Hint: A Game History class can be made which demands to be instantiated only once, so a private

constructor must be used. But we will need an access to this unique instance, that’s why a static

function (get Instance ()) is created to allow us to use the user game history.

|  |
| --- |
| I’d select World Of Warcraft.  - This game records my character info such as Level, Items history, PvP history, Skills, Dungeons history, etc. When I log in to my character, I can control it and do anything that I want. And on the website of World Of Warcraft, somebody can see my character’s info and the history of what I did.  - In this game, has instance dungeons. It could be played by 5 characters. When it was made, only selected 5 characters can play in it and recorded everything. |