TWITTERLITE REPORT

Our project is based on the Model-View-Controller architecture. These are the classes related to each MVC part:

- **Controller**
  - AdminController
  - UserPanelController
  - TestDriver
- **View**
  - AdminControlPanel
  - UserPanel
- **Model**
  - Tweet
  - User
  - UserGroup
  - UserComponent
  - UserComponentFactory
  - UserDatabase
  - UserDBMS
  - PositiveWordList
  - Subject
  - Observer

We are using several Design Patterns in our implementation of TwitterLite.

The **Observer** pattern, Every user is a Subject and an Observer at the same time. User notifies all the followers when a user tweets. And the User's update method is called when any of the followings has a new post.

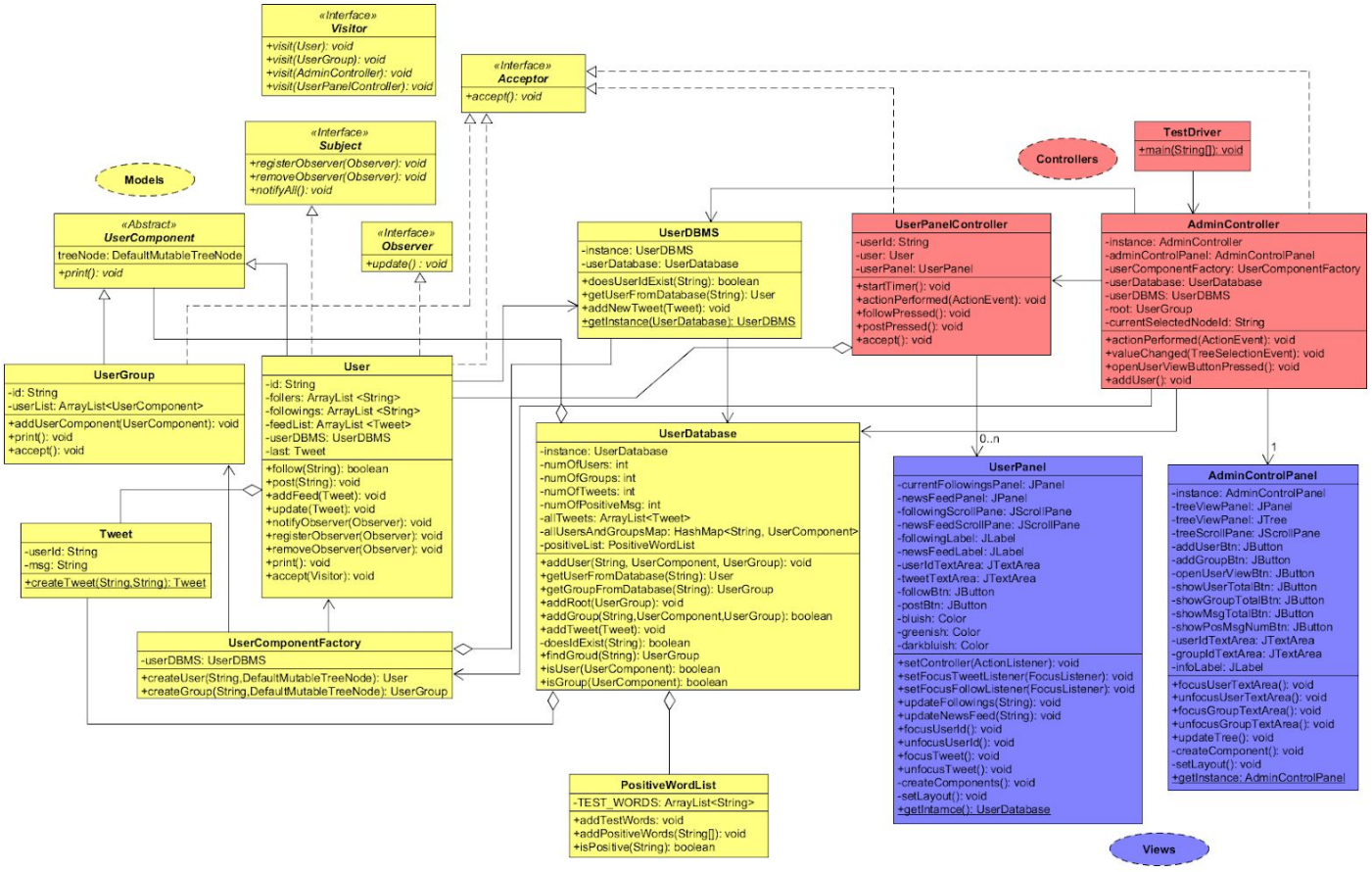
The **Factory** pattern, used by the class *UserComponentFactory* to create generic *UserComponents* of type User or *UserGroup*. Factory has a UserDBMS instance, so that the factory can attach the *UserDBMS* instance to every User to be created.

The **Iterator** and **Composite** patterns, we use Composite pattern by Abstracting the common things between *User* and *UserGroup* classes with the *UserComponent* abstract class. The iterator pattern is included in the print methods inside *User* and *UserGroup* classes.

The **Singleton** pattern, used by *AdminControlPanel* and *AdminController* to create single instances of these classes so be used only once, to differentiate from *UserPanel* that can be created multiple times.

The **Visitor** pattern, we implemented the visitor for *User*, *UserGroup*, *AdminController* and *UserControllerPanel*, but we do not have any concrete Visitor implementation or usage.

MVC architecture is they key to this Twitter application. We strived for loose coupling as much as possible. Therefore, neither any model or any view knows controllers. For example, User knows neither *UserPanelController* or *UserPanel*, as well *UserPanel* knows neither User or *UserPanelController*. This architecture made developing the application much much easier. We can modify a class without needing modifying much about other classes.



Participants

TestDriver - This class has a main method. Please, run this class first. This class just instantiates a AdminController.

AdminController - This class is the only class that is instantiated from the driver. This AdminController instantiates *AdminControlPanel*. AdminController accounts for a creation of users and groups by using the UserFactory. The created users and groups are sent to the *UserDatabase* and stored.
AdminController implements ActionListener and TreeSelectionListener. The *AdminController* methods are triggered by what *AdminControlPanel* gets from the frame. And *AdminController* updates the information to the *AdminControlPanel*.
AdminController also creates UserPanel which is a individual user's Twitter screen.
AdminController is a Singleton.

AdminControlPanel - *AdminControlPanel* is a JPanel. This is created when *AdminController* is created. This panel has JTree, JButtons, JTextAreas and JLabels for a administrator to add a new user, to add a new group, to show the number of users, groups messages and positive percentage. *AdminControlPanel* is a Singleton.

UserPanelController - UserPanelController implements ActionListener and FocusListener. This class is instantiated when a *AdminController* opens an individual user view. *UserPanelController* has a corresponding User and *UserPanel* as its instance variables. This class accounts for following and tweeting.

UserPanel - *UserPanel* is a JPanel. It accounts for a user view. It accepts an user id to follow, a tweet, and shows newsfeed.

UserComponent - This class is a parent abstract class of *User* and *UserGroup* class. Each *UserComponent* has a tree node for itself. This class is useful for the composite pattern.

User - *User* is a *UserComponent*, and it is both a *Subject* and a *Observer*. User models a single user who is capable of following and tweeting. As a *Subject*, when a user posts a new tweet, it notifies all the followers. As a *Observer*, the new following's tweet is stored in the newsfeed. Each user has a reference to the *UserDBMS*.

UserComponentFactory - This class produce an instance of a User or a Group. The creation method accepts an id and the DefaultMutableTreeNode which has the id in it.

UserDatabase - This class models a database of the all information of this Twitter application. All Users and UserGroups are stored in a HashMap, and all the Tweet instances are stored in a ArrayList. This class keeps the track of the number of Users, Groups, Tweet and the positive messages. You can ask to retrieve information from this *UserDatabase*.

UserDBMS - This class allows Users to query the UserDatabase. User class does not have a direct reference to UserDatabase, but UserDBMS.

Tweet - This class models a Tweet. Tweet has a user ID of its creator and the massage as a String. Tweet can only be created by calling static createTweet method by sending a user id and a massage as parameters.

PositiveWordList - PositiveWordList is an ArrayList. This PositiveWordList is held by UserDatabase. This has boolean isPosive method to check if a given String contains a positive word or not.

Subject - Subject is an interface having notifyObservers, registerObserver, and removeObserver. User extends this Subject.

Observer - Observer is an interface having a update method which takes Tweet as a parameter. User extends this Observer.