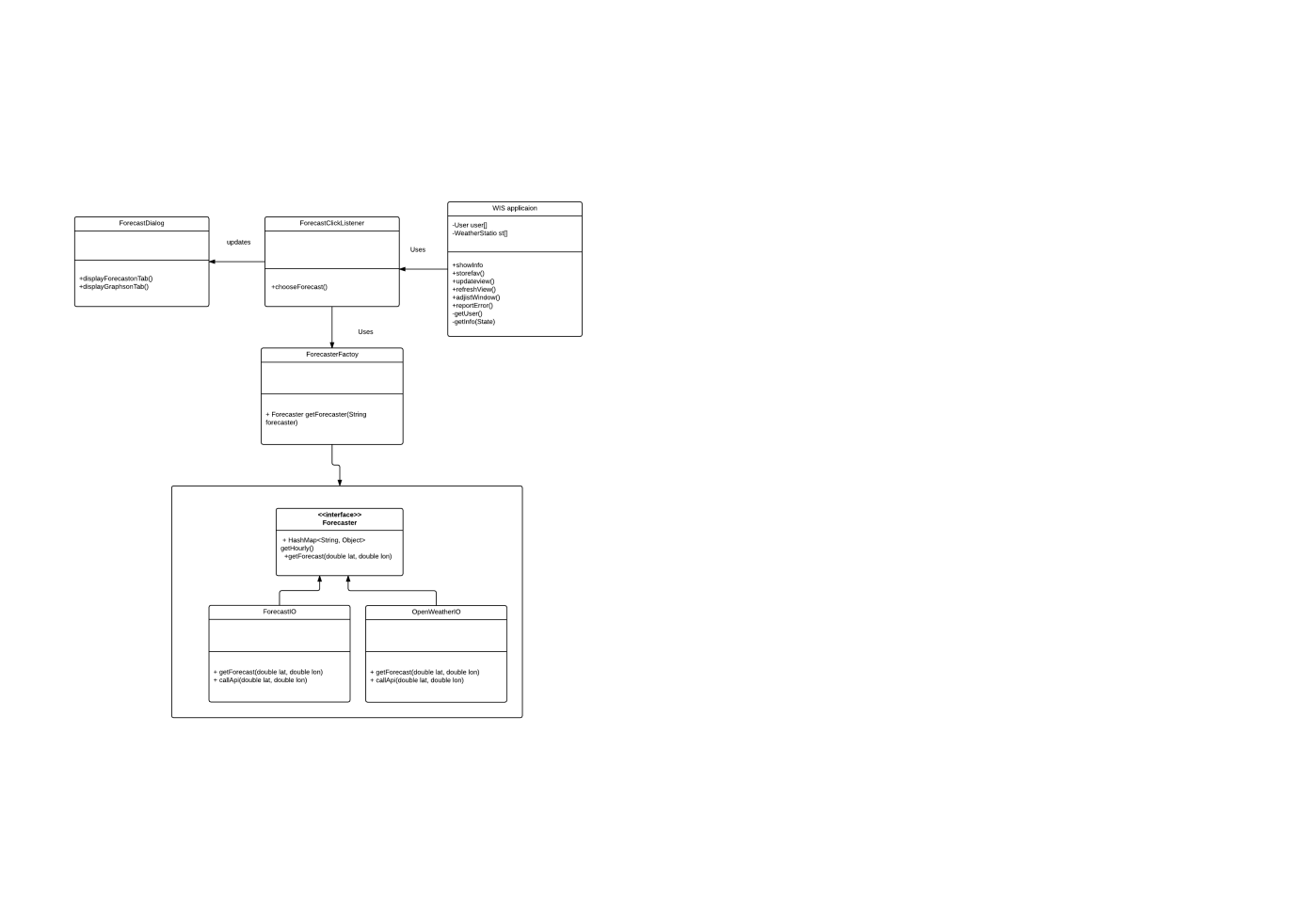
**Assignment 2 Design Implementations**

* **MVC Pattern**

**UML diagram**



**Implementation**

Model view controller in java is used when a component such as view is displayed to the user. The user acts on one of the view components requesting/providing some information triggering a controller which causes the view to update with the appropriate information.

The model is a plain java object (POJO) component. The view inherits from one of the container classes. The controller is basically a listener which listens to events that occur on the view.

This paradigm has been used in several places for assignment 1 already. The MVC patterns for assignment 2 are applicable for the following conditions:

1. Forecast Listener🡪 ForecastIO/Openweather (ForcasterFactory)🡪ForecastDialog

(Controller) Fetches from (Model) updates (View)

The abstract factory implementation allows a generic forecast object to be created. As a generic forecast object can be created. The choice of data from forecastIO or OpenWeather can be made at runtime. A a generic object implies implmentations that are common between forecastIO and openweather.

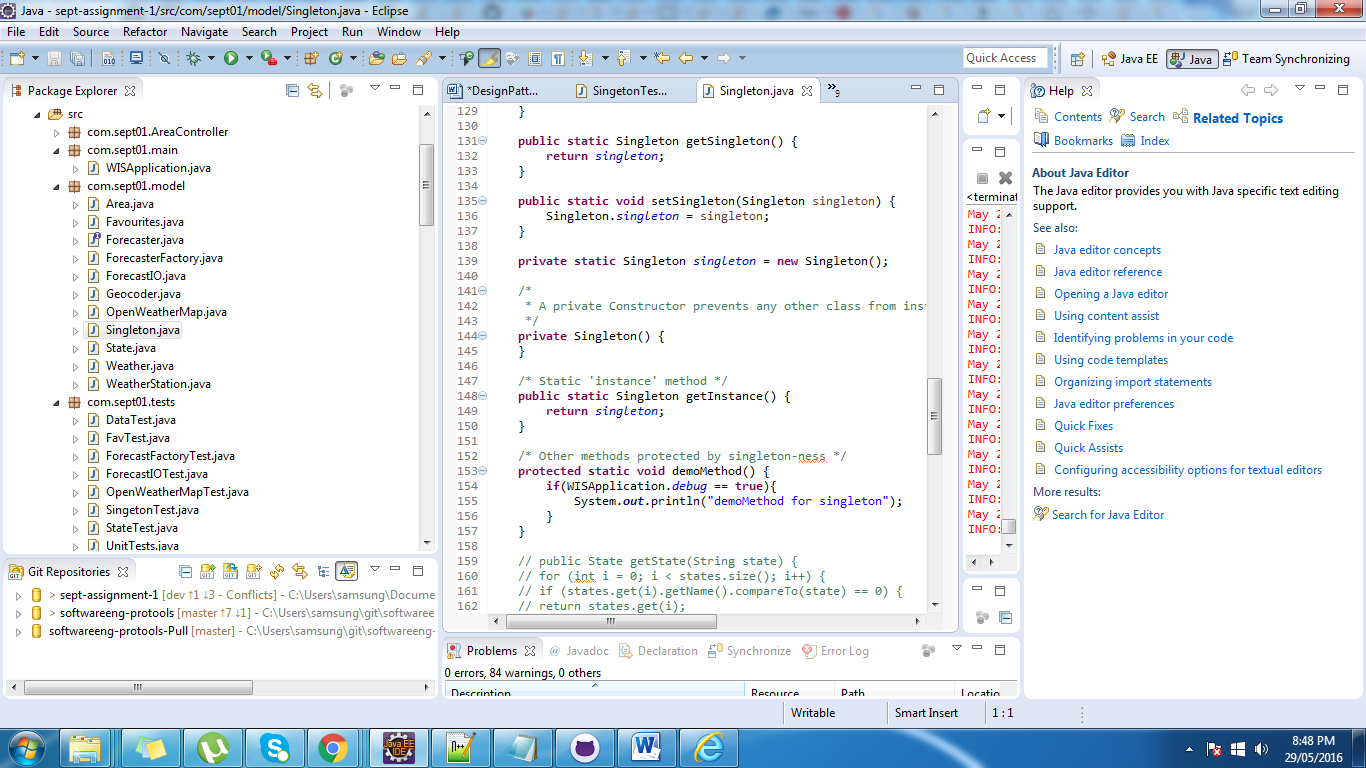
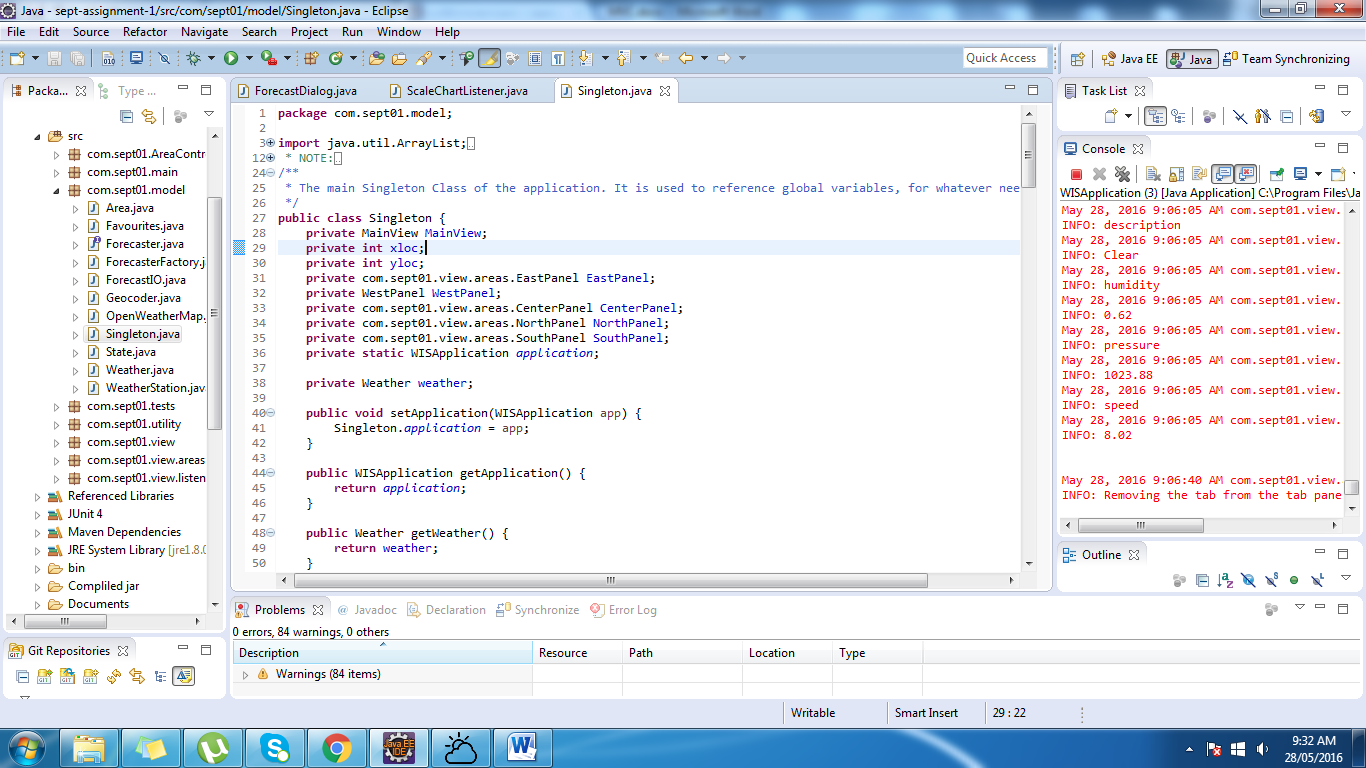
The **FORECASTCLICKLISTENER.JAVA** class allows the user to choose between forecastIO and openweather.Depending upon the choice, there is corresponding call to the appropriate api of the users choice. This is then casted to a generic forecast object. The listener then fetches the data from the model and provides it to the view. The model in this case being **FORECASTFACTORY.JAVA** provides the necessary data, this data is then displayed as a view in the Forecast Dialog (**FORECASTDIALOG.JAVA**). The display can be either values or graphical data.

1. GraphSelector🡪 ForecastIO/Openweather (Factory)🡪ForecastDialog

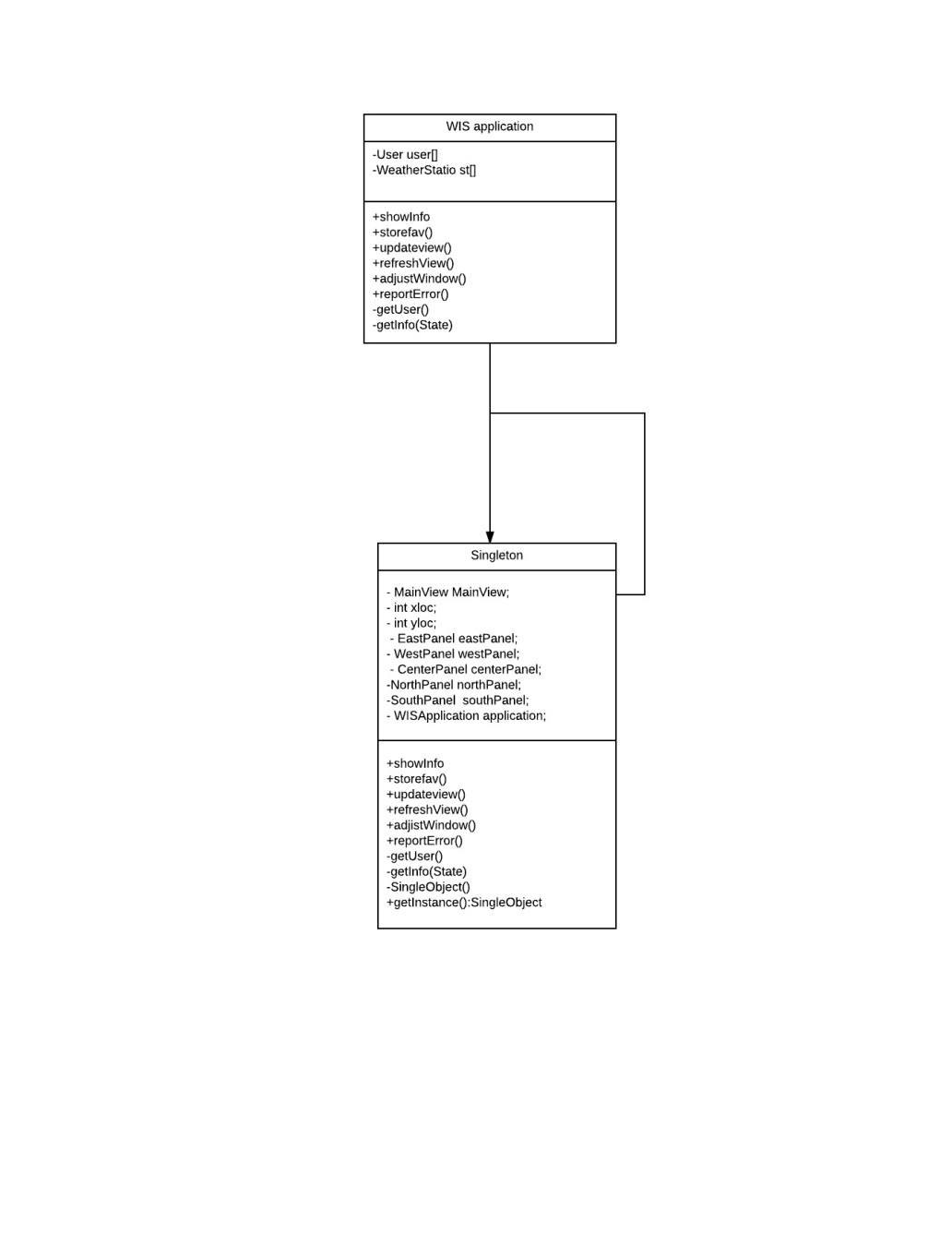
An implementation explanation similar to the above follows for the **GRAPHSELECTOR**.**JAVA** class. The graph selector class fetches the data from the model (abstract factory). This can be for ForecastIO or Openweather and then displays the data from the model in the form of graph in forecast dialog. The forecast dialog acts as a view.

* **Singleton:**

**Snippet**



**UML diagram**



Returns

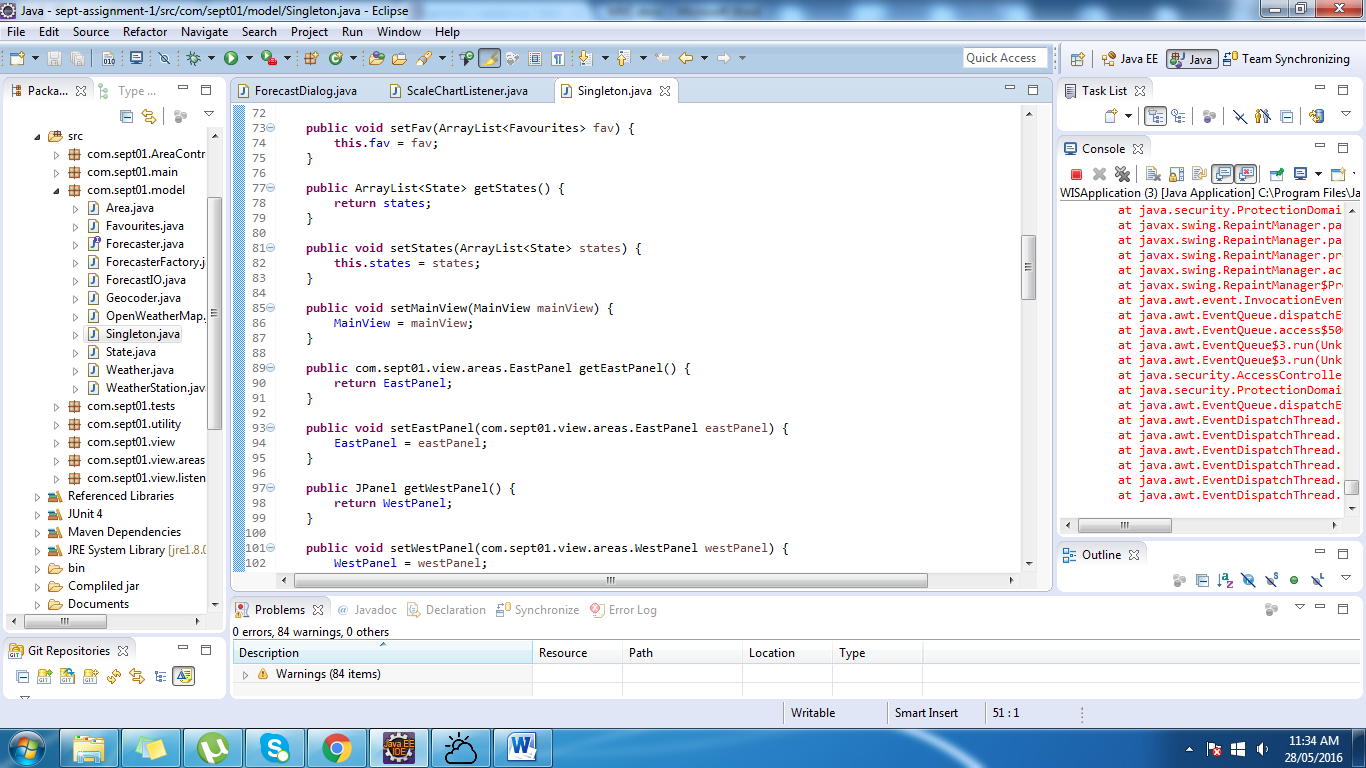
Asks

Singleton class is used in the assignment implementation to ensure that only one instance of an application exists. Therefore any changes to that single instance is reflected throughout the programme. The singleton instance indirectly has been assigned to store the following :

1. Instance of the WIS application (entire application WISAPPLICATION.java). Within the application instances of data that persist are the following:
   1. ForecastData
   2. OpenWeather Data
   3. Historical Data
   4. Favouriates
   5. Screen Preferences

The above stated data must persist throughout the entire run of the program. As we create new instances of class we don’t need to make calls to make multiple calls to services, since data persists in the main instance of the application. Creation of singleton allows flexibility in retrieving data as opposed to passing every instance of the application to another class.

The following view components are also stored in singkleton

1. Instances of the Panels (View)(since every panel is controlled individually ) independent of each other
   1. East Panel
   2. West Panel
   3. NorthPanel
   4. SouthPanel

Since every view in the main view is independent of each other. One can be modified without the knowledge of the other. This can be achieved my retrieving and modifying single instances of panels throughout the application.

As we exit out of the program to remember the favoriates we store the data to a text file and load it back to our singleton instance. Similar is the case for screen preferences.

* **Factory Pattern:**

Factory pattern is one of most used design pattern in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

We created a Forecast Interface and concrete classes such as ForecastIO and OpenWeatherIO implemented the Forecaster Interface. A factory class ForecastorFactory is defined as a next step.

The ForecastorFactory class allows us to create an instance of either ForecastIO or OpenWeatherIO without exposing the creation logic to the client and refer to newly created object using a common interface(Forecaster).

**UML Diagram**

