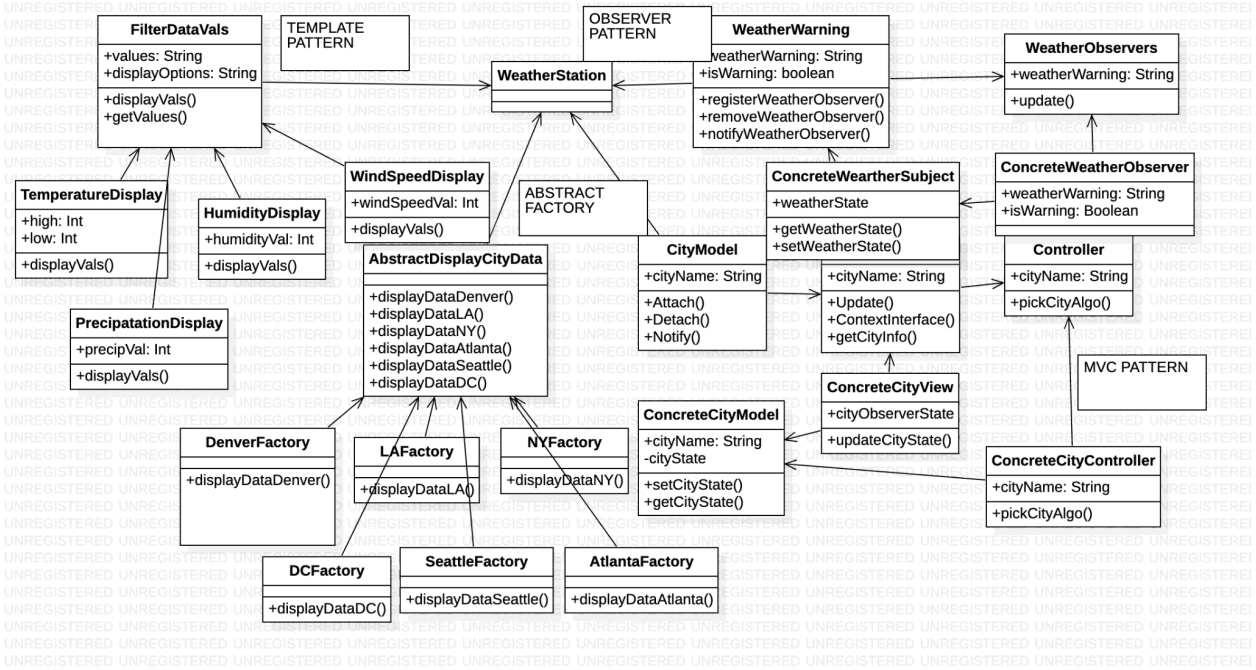
Project Title: Weather Station Simulator

Team Members:
Sarthak Shukla(sash9861@colorado.edu)
Shane O'hagan (shoh1127@colorado.edu)

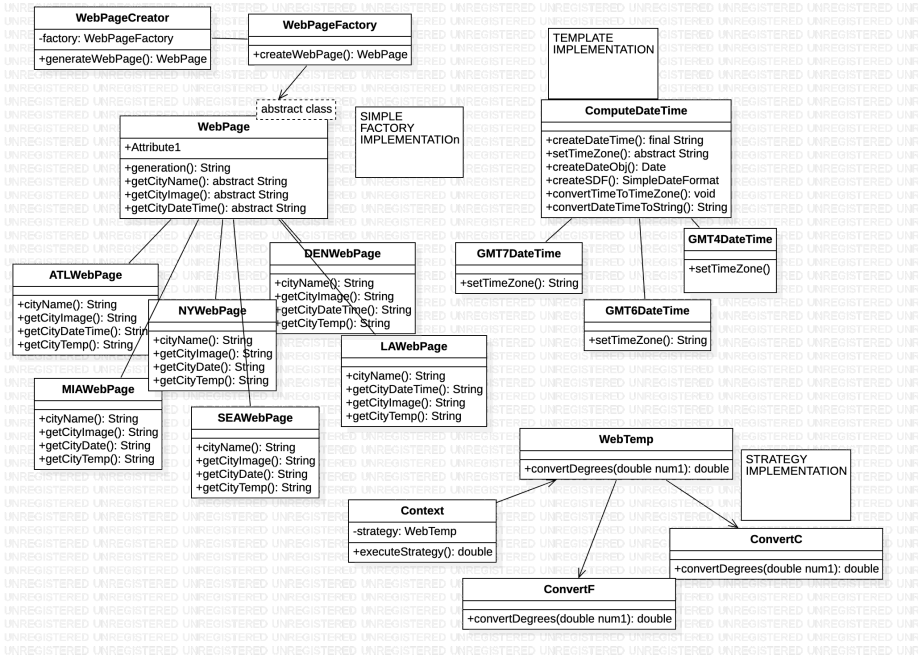
Final State of System:

In our project 6 report we mentioned that we would use a database to query information. This did not work well for our application and we decided to use Open Weather API which allowed us to access temperature, with a high and a low, humidity, windspeed, and dew point. We also decided to not have past, present and future weather information due to the difficulty of getting access to that specific information. Our API did not allow us access to this information unless paid for so we decided to not implement this. Due to time constraints we were unable to host our website to a server to generate users. We additionally changed the uses of our design patterns. We decided to forego the idea of using the MVC to forward to the appropriate web page since we decided that we want to have only one web page which serves as a template and then is loaded with data. We also slightly modified the usage of the factory pattern and are using a simple factory instead of an abstract factory now. The simple factory helps us manage the template web page such that, when a city is clicked on the home page, it transfers you to the template webpage and loads the data on this template webpage using the factory for that city. Instead of using the template pattern to display filtering of data, we are using the template pattern to modify the date time object of each city, since they are in different time zones, so that each city's factory will return the date time object in their respective time zone (using template to manage time zones). We are not using the observer pattern for severe weather warnings, this was our original plan, however it took too long to figure out how to work with the API, and so we were unable to manage this pattern. Finally, we are using the strategy pattern on the template webpage for each city. The strategy pattern allows us to convert the temperature from the data which is in kelvin, and turn it into celsius and fahrenheit units. This is then displayed on a modal. In general, what we have implemented is the full web UI of the software, we are able to return some of the basic data from the API for each city, there is only one web template webpage for all six cities and the data is loaded through simple factory, the factory receives the date time object in the correct timezone for each city based on usage of the template pattern, and finally we are able to view the temperature in different units using the strategy pattern.

Initial Class Diagram:



Final Class Diagram:



What's Changed in the UML Class Diagram: So basically since we switched around what patterns we want to implement and how we want to implement it the class diagrams are very different now. Instead of doing an abstract factory, we are now doing a simple factory to load the web pages. We are not using the same implementations as outlined for Observer, Template, and MVC. Instead we are now using a template pattern to compute the date time object for each timezone for each city. Additionally, we are now using the strategy pattern to convert the weather data for temperature from Kelvin to Fahrenheit and Celsius.

Third-Party Code v.s. Original Code Statement:

Our original code includes:

- All of the files in the package for template pattern (called template_datetime), we only used the lecture slides as reference
- All of the files in the package for the simple factory pattern (called webpage_package), we only used the lecture slides as reference
- Some of the code in the index.jsp and TemplateCity.jsp files is our own code, while a lot of it is pasted from the BootStrap documentation
- The styles.css file is original code

Third-Party Code:

- The whole of the project setup was generated from Eclipse IDE
  - We used this JavaTPoint tutorial to learn how to build the web app
  - This tutorial basically initialized everything server-side for the project
  - We used Apache TomCat server documentation to help us configure some set up for the server
- We used this tutorial on strategy pattern to implement the temperature display (this includes the files in the package named decorator_temp)
- We use this OpenWeather API to send us data
- We used this youtube video to learn how to work with API calls in java

Statement on Overall OOAD Process:

We found that overall, the design patterns we learned may seem very simple, however they are extremely useful in organizing our code as well as extremely applicable to real life coding. Originally, the projects that we did in class were all just terminal software, and so we never really understood that these patterns were applicable to more than just simulations. After implementing these patterns in an actual web application, we have learned so much more about the patterns that we learnt in class. One of the biggest positive elements in the design process was learning more about factories. Originally, our software had 6 JSP pages, one for each city. This was a lot of html, which was essentially the same code on each webpage, but different data. We learned about the usage of simple factory and were able to have only one JSP page which gets loaded with data for each based on their factories. This really helped us consolidate and

organize our code. Another experience that we had was somewhat negative, but also a good learning experience. We had a lot of trouble trying to implement the decorator pattern into our code, trying to follow the lecture slides but also outside documentation. There was a lot of time spent trying to implement this pattern, however we could just not get it to work with our system. Through trial and error we learned that the decorator pattern was just not meant for the usage we wanted it to work for, therefore we ended up switching to the strategy pattern and had an easy time implementing that into our system. Lastly, our experience in working with the template pattern was a very positive learning experience. To ensure good organization in our code, we added the template pattern to compute the temperature conversion to different degrees. This was especially nice because we did not need to inject java code into our JSP files, but instead had it computed server side and then displayed for the user. Therefore the client did not need to see these unnecessary calculations.