Overview

For this practical we were required to use the OpenWeatherMap API to access and manipulate weather data published at openweathermap.org. In order to access the data we had to first register for a free API key, then construct a URL using data input from the user and our own API key.
We were given a Java class file which is used to create a representational state transfer communication in Java. Here the URL is used to read the response from the server and post the data from the server at the specified URL, to the Java client. Finally, we were required to parse JSON data for the current weather and a 5 day forecast, which returned by the server and display the required information in a readable format.

Design

In order to add more functionality to my program I have decided to use two more APIs. These include the Google maps API and the DB-IP API. The reason I have used the google maps API is to allow a user to search for a location by postcode. If they choose to do so, the google maps API is used to geocode and get the name of location for that postcode, for example if the user where to input KY169XW the google maps API will translate this to St Andrews, which can then be used to get the weather for that location. Moreover, the Google Maps API also allows me to get images from google maps, of the input locations. This meant I could show a map of the search for location in the GUI.
The next API I've used is the DB-IP API. This API allows me to get the location of the machine the program is being run by getting machines local IP address. This means the user can now also get the weather for their current location.

For this practical I have decided to create a graphical user which the user is able to interact with to display the forecast results. In order to create the GUI, I have created a separate class which is almost solely used to display swing objects and receive user input, while processing as little data as possible. In this class I have created various swing object in which allow the user to input data and let returned data be displayed on the GUI.
To keep the GUI user friendly and as uncluttered as possible I have used one Jframe which contains multiple "JPanels" for different purposes. In order to switch panels easily I have decided to use a "CardLayout" as well as "TabbedPane". This means I have avoid creating multiple forms. The first panel is designed to allow the user to search for a location, this is the form which takes in data from the user. Upon inputting valid data, a new panel is displayed which contains the "TabbedPane" layout that holds two further panels.
One panel is used to display the current whether for the input location and the second is used to show a 5-day weather forecast. The reason I have used the "TabbedPane" as opposed to another "CardLayout" is because it allows the user to easily select a view, whilst maintaining a professional look and feel.
When the program is run a new instance of the "GUI" class is initiated on the AWT thread, which allows a user to modify the GUI from other threads. Upon initialization, the GUI class reads in a small CSV file in, which is used to populate a combo box with various city names. The purpose of this was to allow a user to simple select a city from a dropdown box as well as giving them the option to manually type a location. All the components are then added to the GUI and the GUI is displayed.
To avoid having a cluttered interface I have deiced to not add buttons for all of the search boxes, but instead have used "ActionListners" which allow the program to detect key presses. Thus if decides they want to type a location they can simply type the location and press enter.
In order to display the current status of the weather at the user's specified location I have used multiple "JLabels". Some labels are used to simply display text where as other are used to display icons. The two main final components of the GUI are the 3 "RadioButtons"

which allow the user to select which temperature unit they wish to have the temperature displayed in. These radio buttons are grouped together, so that only one button can be selected at any given time. Again the radio buttons have action listens assigned to them which allow them call method in different classes. A Search button, which is displayed on the "CurrentDay" panel and the "5Day" panel allows the user to switch cards and returns to the search panel, so they can enter a new location.
Finally, I have decided to add a "Weather" object as an attribute of the GUI class. This allows me to hand around "Weather" objects which hold the relevant data to populate the interface with. Next in order for other classes to gain access and update the interface the GUI object created, is set as a static attribute in the "updateGUI" class and the "weatherInfo" class.

In order to get the JSON data from the various servers, I have created a class called "weatherInfo". This class contains several constants which hold the URL's for the different the different API's as well as my API keys. This class also contains several methods which all create a JSON parser and construct a URL containing an API key and a given parameter(s). Once the URL is constructed the method "makeRESTCall" in the "Connection" class is called. Here an HttpURLConnection object is created with a URL as a parameter, which allows data to be read and posted between the server at the specified URL, and the Java client. If the URL is for the Open Weather API the parameters in the URL can be a location input by the user, a location found using the Google Maps API or the user's current location found using the DB-IP API. Once the required parameters are found and validated an instance of the JSONWeatherParser class is initiated and the JSON response for both the current weather and the 5-day forecast, from the server is given to the JSONWeatherParser. The JSONWeatherParser class then creates new "Weather" objects, which contain the required information from the server responses.
If alternatively, the URL is for the DB-IP API the parameters in the URL will be the IP address of the user local machine. Once the server responds with the JSON data an instance of the JSONLocationParser class is initiated and the JSON response, from the server is handed in. The JSONLocationParser class then creates new "Location" object, which contain the required information from the server responses. The user's current location is then used by the Open Weather API to get the weather for their current location.

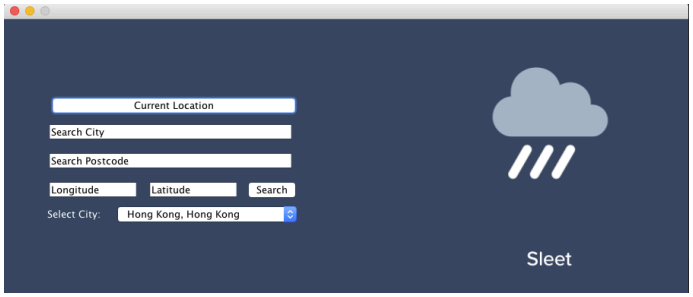
The final class I've used is the "UpdateGUI" class. This class holds the current GUI and thus is able to access all of the swing items on the interface. This class also takes in weather objects and applies the inform from the weather objects to the correct objects in the GUI. In this class I have also decided to create a method which contains a switch stament which takes in the icon identifier for the weather object and translates it to a picture. This meant I could map the Open Weather Map icon identifiers to my pictures of my own choice.

Extensions

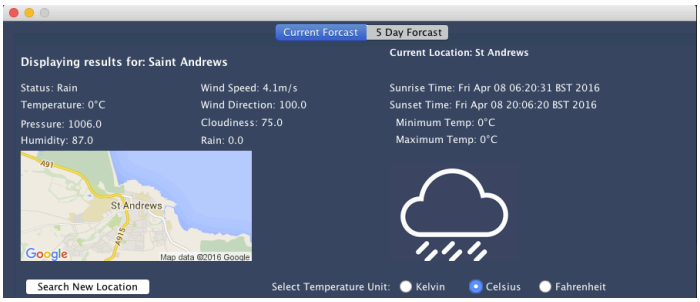
I've implemented the following extensions:

- Graphical User Interface
- Getting the users current location
- Allowing the user to search for location by postcode
- Displaying the input location on google maps

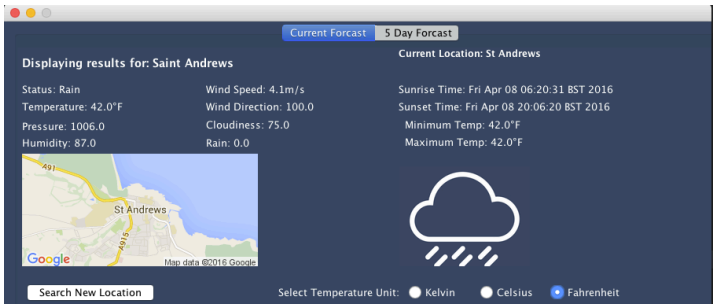
Examples



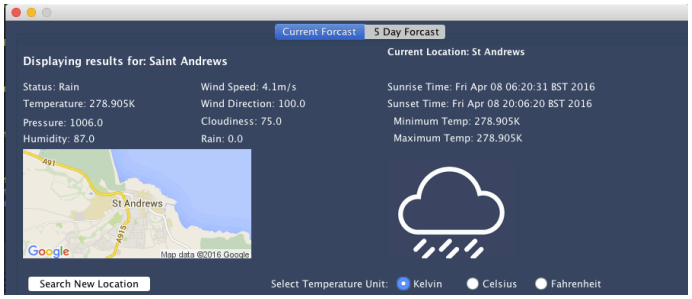
This is the initial screen where the user can enter a location various different ways



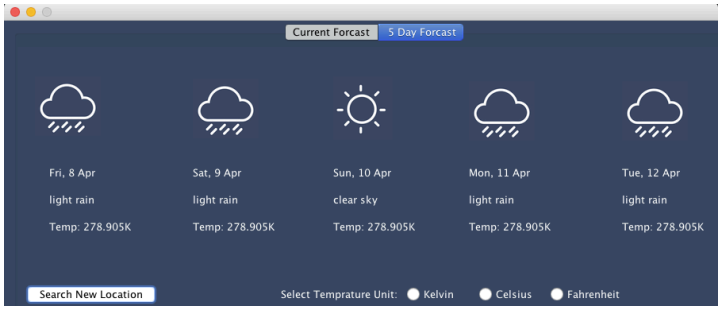
Once a location is entered this screen shows he current weather for the input location, I have changed the temperature to Celsius and the form correctly displays the temp in degrees C.



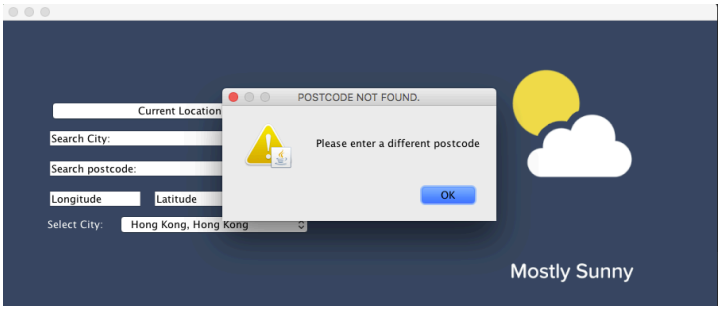
Here the temperature is set to be shown in Fahrenheit and the form correctly displays the temp in degrees F.



Here the temperature is set to be shown in Kelvin and the form correctly displays the temp in kelvin.



This is the 5-day weather forecast



This is how the program outputs error messages to the user

Testing

Test Description	Expected Output	Actual Output	Comment
Enter valid city	The program will display the correct weather for the input location, the google maps icon will also display the searched for location.	The program displayed the correct weather for the input location, the google maps icon also displayed the searched for location.	Worked as expected
Enter Valid Postcode	The program will display the correct weather for the input location, the google maps icon will also display the searched for location.	The program displayed the correct weather for the input location, the google maps icon also displayed the searched for location.	Worked as expected
Enter valid longitude and valid latitude	The program will display the correct weather for the input location, the google maps icon will also display the searched for location.	The program displayed the correct weather for the input location, the google maps icon also displayed the searched for location.	Worked as expected
Select a city	The program will display the correct weather for the input location, the google maps icon will also display the searched for location.	The program displayed the correct weather for the input location, the google maps icon also displayed the searched for location.	Worked as expected
Select Current Location	The program will display the correct weather for the user's current location, the google maps icon will also display the searched for location.	The program displayed the correct weather for the user's current location, the google maps icon also displayed the searched for location.	Worked as expected

Enter invalid city	The program will attempt to auto correct if it can not find a similar city to that of the input city the program will ask the user to reenter a valid city	One some attempts the OpenWeatherApp API managed to auto correct the input if it couldn't an error message was displayed	Worked as expected
Enter invalid postcode	The program will ask the user to input a valid postcode	The program asked the user to input a valid postcode	Worked as expected
Enter invalid latitude but valid longitude	The program will ask the user to correct the input coordinates	The program asked the user to correct the input coordinates	Worked as expected
Enter invalid longitude but valid latitude	The program will ask the user to correct the input coordinates	The program asked the user to correct the input coordinates	Worked as expected
Enter invalid longitude and invalid longitude	The program will ask the user to correct the input coordinates	The program asked the user to correct the input coordinates	Worked as expected
Change temp to kelvin	The program will update the temperatures values displayed on the GUI to correct values	The program updated the temperatures values displayed on the GUI to correct values	Worked as expected
Change temp to degrees C	The program will update the temperatures values displayed on the GUI to correct values	The program updated the temperatures values displayed on the GUI to correct values	Worked as expected
Change temp to degrees F	The program will update the temperatures values displayed on the GUI to correct values	The program updated the temperatures values displayed on the GUI to correct values	Worked as expected
Use VPN then check use current location	The program will show the results for the location where the VPN is set to	The program dispayed the results for the location where the VPN is set to	Worked as expected

Evaluation

After testing my program, I pleased to say it works as expected and is able to gracefully display weather forecast results. Moreover, I am also pleased the GUI I created is clean and user friendly and the icons give make the interface look more simplistic but professional. The google maps API and the DB-IP API add a lot of helpful functionality to the program too.

Conclusion

In conclusion I'm satisfied with end program, it is able to carry out the required tasks with ease and is able to handle erroneous data without crashing. The added extensions massively increase the overall functionality.