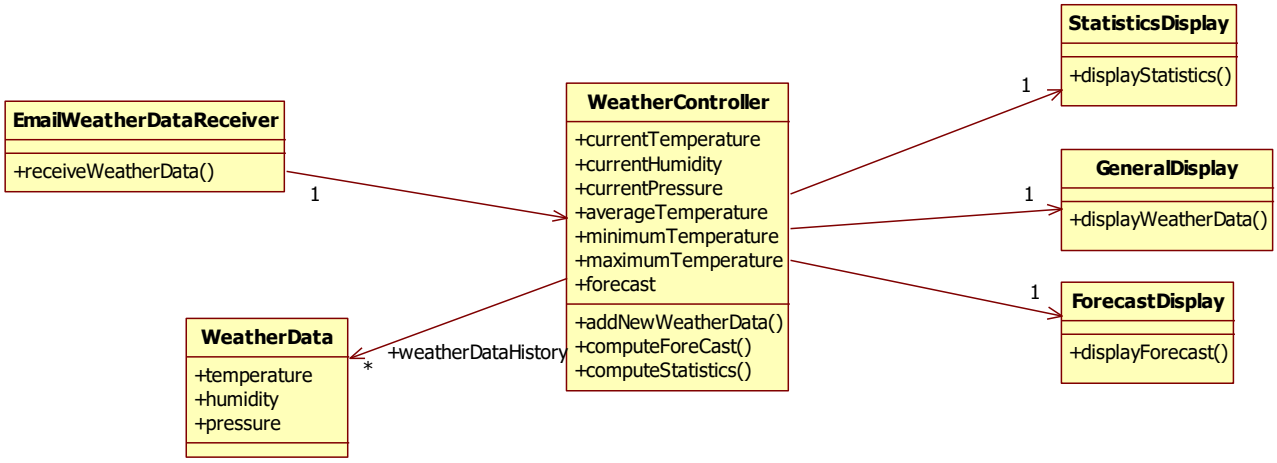
**Lab 11**

**Part 1**:

Suppose you have to design a weather monitoring application. The application works as follows:

- A piece of hardware receives weather data from sensors that measure the temperature, the humidity and the pressure. Anytime when the temperature or the humidity or the pressure changes, an email containing the new temperature, humidity and pressure is sent to our weather monitoring application.
- The weather monitoring application contains 3 different displays:
  - The GeneralDisplay that shows the current temperature, humidity and pressure
  - The StatisticsDisplay that shows the weather statistics: average temperature, minimum temperature and maximum temperature
  - The ForecastDisplay that shows the weather forecast like "cloudy and rain" or "sunny and warm"
- When the weather monitoring application receives the email with the new weatherdata, it calculates the new weather statistics and the new weather forecast, and then updates the 3 displays.

We design the weather monitoring application as follows:




Now we want to design a more flexible and extendable weather monitoring framework with the following requirements:

1. Our current application can only receive new weather data by email. The framework should support also different ways to receive weather data, like receiving new weather data by webservice call, by SMS, by remote method call, etc.

2. Right now, the WeatherController computes the forecast based on an algorithm that works well in North America. In South America they use a different algorithm to compute the weather forecast. The framework should support both the North America and the South America algorithm, but it should also be possible to extend the framework such that it uses a different algorithm to compute the forecast.
3. The current application supports 3 different displays: The GeneralDisplay, the StatisticsDisplay and the ForecastDisplay. The framework should support 2 extra displays:
    a. A ForecastHistoryDisplay that shows all forecasts from the last year. This display shows the date, time and forecast of all the different forecasts it computed the last year.

        Example: 03/12/2011 11:05 "sunny and warm"
        03/13/2011 05:45 "cloudy"

        Make sure your class diagram supports the ability to have the history of forecasts available. The weather station application has no database, so all data remains in memory.

    b. A StatisticsHistoryDisplay that shows the all weather statistics from the last year. This display shows the date and statistics of all the different statistics it computed the last year.

        Example: 03/12/2011 avg temp= 77 min temp = 69max temp = 82
        03/13/2011 avg temp= 74 min temp = 72max temp = 76

        Make sure your class diagram supports the ability to have the history of statistics available. The weather station application has no database, so all data remains in memory.

4. It should be very easy to add new types of displays when you use the framework, without changing the WeatherController class.

Draw the **UML class diagram** of weather monitoring framework. Show clearly the necessary objects, attributes, methods and associations.

**Part 2:**

Suppose you need to design a simple framework that allow us to race with a car.
We have the following requirements for the **car racing framework**:
1. The car has a certain speed, and the framework should support to increase and decrease the speed of the car. Because this is a simple framework, the car can only go faster or go slower. You cannot go left or go right.

2. The User Interface that uses this framework has a button to go faster and a button to go slower. The framework supports the following build in functionality:

- When the speed is below 70 miles per hour, then the speed will be increased or decreased with 1 mile per hour, depending on the button you click.
- When the speed is more than 70 miles per hour, then the speed will be increased or decreased with 3 mile per hour, depending on the button you click.

It should be easy to change this build in behaviour

3. It should be possible to see the list of all actions we have done on the car so that we can replay all actions we have done so far.

4. It should be easy to add other actions to the car, like go left, go right, jump, etc.

5. The framework will log all changes to the car speed in a logfile on the filesystem

6. It should be easy to add other listener classes that need to do something with the car speed when it changes.

With this framework we need to design a race application that uses this framework. This application has the following requirements:
1. The application also supports the functionality to jump with the car. So the User Interface has an extra button that allows us to make a jump with the car.

2. The application works with the following speed functionality:

- When the speed is below 40 miles per hour, then the speed will be increased or decreased with 1 mile per hour, depending on the button you click.
- When the speed is between 40 and 80 miles per hour, then the speed will be increased or decreased with 2 mile per hour, depending on the button you click.
- When the speed is more than 80 miles per hour, then the speed will be increased or decreased with 3 mile per hour, depending on the button you click.

3. The application will log all changes to the car speed in the database.

**a. Draw the class diagram of your design.**
Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc.) to communicate the important parts of your design.
So this class diagram should show the design of the framework, and the design of the application that uses the framework. **In the class diagram, show clearly which classes are within the framework, and which classes are outside the framework.**
Make sure that your design follows the design principles we studied in this course.

b. **Draw the sequence diagram that shows how your design works**. Make sure you add all necessary UML elements to communicate the important parts of your design. The

sequence diagram should show the following scenario: The precondition is that the car goes with a speed of 30 miles per hour
1. The user clicks the increment button.
2. The user clicks the jump button.

**c. Implement your solution.** Write the framework classes in a separate package
Write the application using the framework in another package.
The window frames are just normal Java classes that use System.out.println(). You do not need to use a Java Swing UI.
The Logger class does not really write to a log file, the log() method just uses a System.out.println().
The DAO class does not really go to the database. It just simulates that it goes to the database. It keeps the speed of the car in memory.