Lab4: Observer Pattern

Bài 1: Hệ thống theo dõi thời tiết

Xây dựng một trạm theo dõi thời tiết có

- Trạm thời tiết - thiết bị phần cứng mà thu thập dữ liệu từ các cảm biến khác nhau (độ ẩm, nhiệt độ, áp suất).
- **WeatherData** đối tượng tương tác với các phần cứng trạm thời tiết.

Cần phải cài đặt ba thành phần hiển thị sử dụng **WeatherData** và phải được cập nhật mỗi lần WeatherData có số đo mới

- Một **màn hình điều kiện hiện tại** hiển thị: nhiệt độ, độ ẩm, sự thay đổi áp suất.
- Một **màn hình số liệu thống kê thời tiết** hiển thị nhiệt độ trung bình, nhiệt độ nhỏ nhất, nhiệt độ lớn nhất.
- Và một **màn hình hiển thị dự báo**

Hệ thống phải có thể mở rộng: có thể tạo ra các thành phần hiển thị mới và có thể thêm hoặc loại bỏ càng nhiều thành phần hiển thị như mong muốn của ứng dụng

1. Tạo project ***WeatherStation1***, cài đặt tất cả các lớp và phương thức sau::
   – *Lớp* ***WeatherData*** có 3 thuộc tính: nhiệt độ (temperature), độ ẩm (humidity), áp suất (pressure). Cài đặt các phương thức truy xuất (getter methods) cho các thuộc tính này. Lớp ***WeatherData*** cũng có phương thức **meansurementsChange()** mà được gọi mỗi khi có dữ liệu thời thiết mới.

```
public class WeatherData {
   private CurrentConditionsDisplay currentConditionsDisplay;
   private StatisticsDisplay statisticsDisplay;
   private ForecastDisplay forecastDisplay;
   private float temperature;
   private float humidity;
   private float pressure;

   public float getTemperature() {
      return temperature;
   }

   public float getHumidity() {
      return humidity;
   }

   public float getPressure() {
      return pressure;
   }

   public void measurementsChanged() {
      float temp = getTemperature();
      float humidity = getHumidity();
      float pressure = getPressure();
      currentConditionsDisplay.update(temp, humidity, pressure);
      statisticsDisplay.update(temp, humidity, pressure);
      forecastDisplay.update(temp, humidity, pressure);
   }

   public void setMeasurements(float temperature, float humidity, float pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      this.pressure = pressure;
```

```
        measurementsChanged();
    }

    public void setCurrentConditionsDisplay(
            CurrentConditionsDisplay currentConditionsDisplay) {
        this.currentConditionsDisplay = currentConditionsDisplay;
    }

    public void setStatisticsDisplay(StatisticsDisplay statisticsDisplay) {
        this.statisticsDisplay = statisticsDisplay;
    }

    public void setForecastDisplay(ForecastDisplay forecastDisplay) {
        this.forecastDisplay = forecastDisplay;
    }
}
```

– Lớp ***CurrentConditionsDisplay*** hiển thị các số đo thời tiết hiện thời.

```
public class CurrentConditionsDisplay {
    private float temperature;
    private float humidity;
    private WeatherData weatherData;

    public CurrentConditionsDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
        weatherData.setConditionDisplay(this);
    }

    public void update(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        display();
    }

    public void display() {
        System.out.println("Current conditions: " + temperature
                + "F degrees and " + humidity + "% humidity");
    }
}
```

– Lớp ***StatisticsDisplay*** hiển thị giá trị trung bình, cực tiểu, cực đại của mỗi số đo thời tiết.

```
public class StatisticsDisplay {
    private float maxTemp = 0.0f;
    private float minTemp = 200;
    private float tempSum = 0.0f;
    private int numReadings;
    private WeatherData weatherData;

    public StatisticsDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
        weatherData.setStatisticsDisplay(this);
    }

    public void update(float temp, float humidity, float pressure) {
        tempSum += temp;
        numReadings++;
        if (temp > maxTemp) maxTemp = temp;
        if (temp < minTemp) minTemp = temp;
        display();
```

```
    }

    public void display() {
        System.out.println("Avg/Max/Min temperature = "
                + (tempSum / numReadings)
                + "/" + maxTemp + "/" + minTemp);
    }
}
```

– Lớp ***ForeCastDisplay*** hiển thị dự báo thời tiết đơn giản.

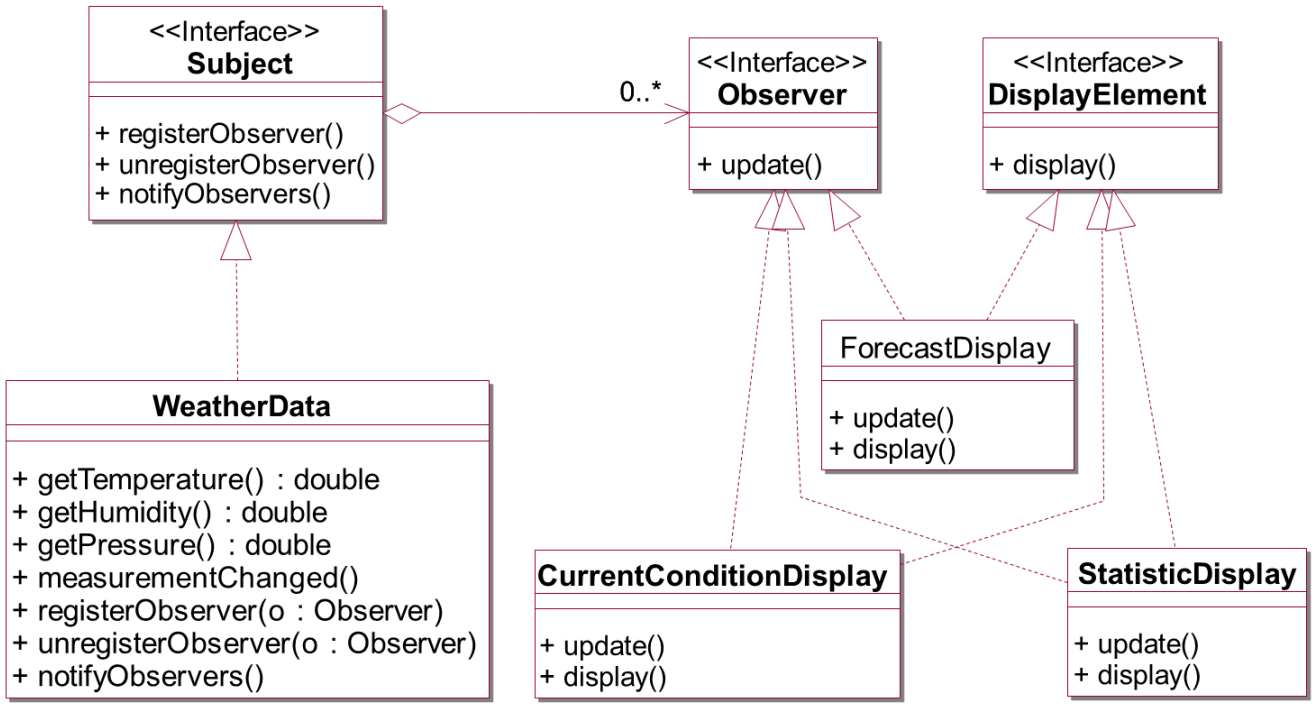
```
public class ForecastDisplay {
    private float currentPressure = 29.92f;
    private float lastPressure;
    private WeatherData weatherData;

    public ForecastDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
        weatherData.setForecastDisplay(this);
    }

    public void update(float temp, float humidity, float pressure) {
        lastPressure = currentPressure;
        currentPressure = pressure;

        display();
    }

    public void display() {
        System.out.print("Forecast: ");
        if (currentPressure > lastPressure) {
            System.out.println("Improving weather on the way!");
        } else if (currentPressure == lastPressure) {
            System.out.println("More of the same");
        } else if (currentPressure < lastPressure) {
            System.out.println("Watch out for cooler, rainy weather");
        }
    }
}
```

– ***WeatherStation*** test class:

```
public class WeatherStation {
    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
        CurrentConditionsDisplay currentDisplay =
            new CurrentConditionsDisplay(weatherData);
        StatisticsDisplay statisticsDisplay =
            new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

        weatherData.setMeasurements(80, 65, 30.4f);
        System.out.println("------------------------");
        weatherData.setMeasurements(82, 70, 29.2f);
        System.out.println("------------------------");
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```

– *Nhận xét:* Hệ thống có vẻ cài đặt đơn giản. Tuy nhiên,
  - Bạn không có cách nào để thêm hoặc loại bỏ các thành phần hiển thị khác mà không làm thay đổi chương trình.
  - Tất cả các thành phần hiển thị nên thực hiện một interface chung bởi vì tất cả đều có một phương thức update() nhận các tham số temperature, humidity, và pressure.

2. Tạo project ***WeatherStation2***; cài đặt hệ thống theo dõi thời tiết dùng mẫu Observer.



Cài đặt code và bổ sung các phần còn thiếu (// TODO)

– Tạo interface ***Subject*** để đăng ký, hủy và thông báo các observers. Cả hai phương thức registerObserver() và removeObserver() lấy một đối tượng Observer như là tham số được đăng ký hoặc hủy. Bất cứ khi nào trạng thái của Subject thay đổi, phương thức notifyObserver() được gọi để thông báo cho các observers.

```
package headfirst.observer.weather;
public interface Subject {
   public void registerObserver(Observer o);
   public void removeObserver(Observer o);
   public void notifyObservers();
}
```

– Lớp ***WeatherData*** bây giờ cài đặt interface ***Subject***. Nó có một danh sách ArrayList được tạo trong constructor để lưu giữ các observer.

```
package headfirst.observer.weather;
import java.util.ArrayList;
public class WeatherData implements Subject {
   private ArrayList<Observer> observers;
   private float temperature;
   private float humidity;
   private float pressure;

   public WeatherData() {
      observers = new ArrayList<Observer>();
   }
```

```
    public void registerObserver(Observer o) {
        // TODO
    }

    public void removeObserver(Observer o) {
        // TODO
    }

    public void notifyObservers() {
        // TODO
    }

    public void measurementsChanged() {
        // TODO
    }

    public void setMeasurements(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }

    // other WeatherData methods here

    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }
}
```

– Định nghĩa một interface chung để thao tác khi đến lúc cập nhật các observers. Có nghĩa là chúng ta sẽ tạo một interface mới là ***Observer*** có phương thức **update()**.

```
package headfirst.observer.weather;
public interface Observer {
    public void update(float temp, float humidity, float pressure);
}
```

– Cũng định nghĩa một interface ***DisplayElement*** cho tất cả các thành phần hiển thị cài đặt. Nó có một phương thức duy nhất là **display()**.

```
package headfirst.observer.weather;
public interface DisplayElement {
    public void display();
}
```

– Các lớp ***CurrentCoditionsDisplay***, ***StatisticsDisplay***, ***ForecastDisplay*** cài đặt interface ***Observer*** và ***DisplayElement*** vì thế chúng phải cài đặt phương thức update() và display(). Phương thức constructor của các lớp này được truyền đối tượng ***WeatherData*** (***Subject***), mà có thể dùng để đăng ký thành phần hiển thị này như là một observer.

– ***CurrentConditionsDisplay*** hiển thị số đo hiện thời của đối tượng ***WeatherData***.

```
package headfirst.observer.weather;

public class CurrentConditionsDisplay implements Observer, DisplayElement {
   private float temperature;
   private float humidity;
   private Subject weatherData;

   public CurrentConditionsDisplay(Subject weatherData) {
      this.weatherData = weatherData;
      // TODO: register this to observe weatherData
   }

   public void update(float temperature, float humidity, float pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      display();
   }

   public void display() {
      System.out.println("Current conditions: " + temperature
            + "F degrees and " + humidity + "% humidity");
   }
}
```

– ***ForecastDisplay*** shows the weather forecast based on the barometer.

```
package headfirst.observer.weather;

import java.util.*;

public class ForecastDisplay implements Observer, DisplayElement {
   private float currentPressure = 29.92f;
   private float lastPressure;
   private WeatherData weatherData;

   public ForecastDisplay(WeatherData weatherData) {
      this.weatherData = weatherData;
      // TODO: register this to observe weatherData
   }

   public void update(float temp, float humidity, float pressure) {
      lastPressure = currentPressure;
      currentPressure = pressure;
      display();
   }

   public void display() {
      System.out.print("Forecast: ");
      if (currentPressure > lastPressure) {
         System.out.println("Improving weather on the way!");
      } else if (currentPressure == lastPressure) {
         System.out.println("More of the same");
      } else if (currentPressure < lastPressure) {
         System.out.println("Watch out for cooler, rainy weather");
      }
   }
}
```

– ***StatisticsDisplay*** displays the average, minimum, and maximum measurements.

```java
package headfirst.observer.weather;

public class StatisticsDisplay implements Observer, DisplayElement {
   private float maxTemp = 0.0f;
   private float minTemp = 200;
   private float tempSum = 0.0f;
   private int numReadings;
   private WeatherData weatherData;

   public StatisticsDisplay(WeatherData weatherData) {
      this.weatherData = weatherData;
      // TODO: register this to observe weatherData
   }

   public void update(float temp, float humidity, float pressure) {
      tempSum += temp;
      numReadings++;

      if (temp > maxTemp) {
         maxTemp = temp;
      }

      if (temp < minTemp) {
         minTemp = temp;
      }
      display();
   }

   public void display() {
      System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)
            + "/" + maxTemp + "/" + minTemp);
   }
}
```

– ***WeatherStation*** test class:

```java
package headfirst.observer.weather;
public class WeatherStation {
   public static void main(String[] args) {
      WeatherData weatherData = new WeatherData();
      CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(
            weatherData);
      StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
      ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);

      weatherData.setMeasurements(80, 65, 30.4f);
      System.out.println("------------------------");
      weatherData.setMeasurements(82, 70, 29.2f);
      System.out.println("------------------------");
      weatherData.setMeasurements(78, 90, 29.2f);
   }
}
```

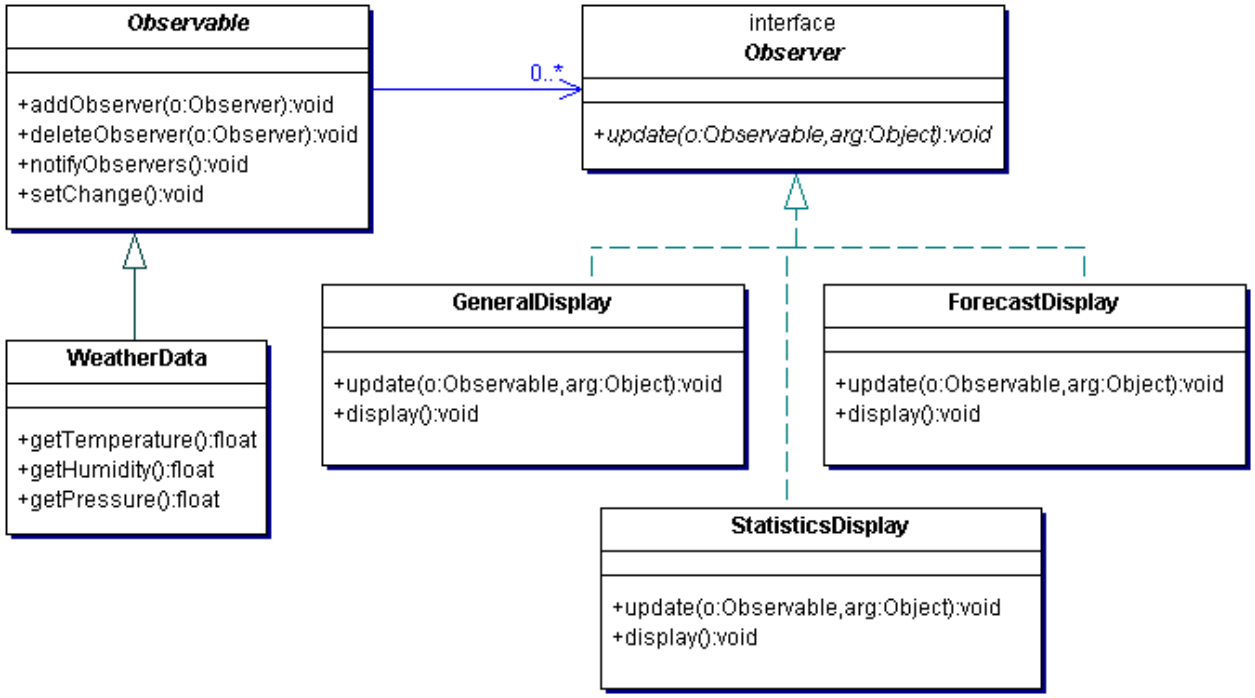
– **Thêm màn hình hiển thị `HeatIndexDisplay`** và test

```java
package headfirst.observer.weather;

public class HeatIndexDisplay implements Observer, DisplayElement {
   float heatIndex = 0.0f;
   private WeatherData weatherData;
```

```
    public HeatIndexDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    public void update(float t, float rh, float pressure) {
        heatIndex = computeHeatIndex(t, rh);
        display();
    }

    private float computeHeatIndex(float t, float rh) {
        float index = (float) ((16.923 + (0.185212 * t) + (5.37941 * rh)
                - (0.100254 * t * rh) + (0.00941695 * (t * t))
                + (0.00728898 * (rh * rh)) + (0.000345372 * (t * t * rh))
                - (0.000814971 * (t * rh * rh))
                + (0.0000102102 * (t * t * rh * rh)) - (0.000038646 * (t * t * t))
                + (0.0000291583 * (rh * rh * rh))
                + (0.00000142721 * (t * t * t * rh))
                + (0.000000197483 * (t * rh * rh * rh))
                - (0.0000000218429 * (t * t * t * rh * rh)) + 0.000000000843296 * (t
                * t * rh * rh * rh)) - (0.0000000000481975 * (t * t * t * rh * rh * rh)));
        return index;
    }

    public void display() {
        System.out.println("Heat index is " + heatIndex);
    }
}
```

```
package headfirst.observer.weather;

public class WeatherStationHeatIndex {
    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
        CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(
                weatherData);
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);
        HeatIndexDisplay heatIndexDisplay = new HeatIndexDisplay(weatherData);

        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```

3. Create another project called ***WeatherStation3*** to solve your problem with the Java's build-in Observer pattern.
   - Java provides you the **java.util.Observable** class and the **java.util.Observer** interface to build your project based on the **Observer** Pattern. The **Observable** class keeps track of all your observers and notifies them for you.



– The **WeatherData** class extends the **Observable** class. You don't need to override the **registerObserver()**, **removeObserver()**, and **notifyObserver()** methods anymore. You inherit that behavior form the super class, **Observable**.

```
import java.util.Observable;
import java.util.Observer;

public class WeatherData extends Observable {
   private float temperature;
   private float humidity;
   private float pressure;

   public WeatherData() {
   }

   public void measurementsChanged() {
      setChanged();
      notifyObservers();
   }

   public void setMeasurements(float temperature,
      float humidity, float pressure) {
      this.temperature = temperature;
      this.humidity = humidity;
      this.pressure = pressure;
      measurementsChanged();
   }

   public float getTemperature() {
      return temperature;
   }

   public float getHumidity() {
      return humidity;
   }
```

```
    public float getPressure() {
        return pressure;
    }
}
```

- You can left out the **Display** interface but all your display elements still implement it too.
- The update() method of all display elements must change to take both an Observable and the optional data argument.
    - **CurrentConditionsDisplay**

```
import java.util.Observable;
import java.util.Observer;

public class CurrentConditionsDisplay implements Observer, DisplayElement {
    Observable observable;
    private float temperature;
    private float humidity;

    public CurrentConditionsDisplay(Observable observable) {
        this.observable = observable;
        observable.addObserver(this);
    }

    public void update(Observable obs, Object arg) {
        if (obs instanceof WeatherData) {
            WeatherData weatherData = (WeatherData) obs;
            this.temperature = weatherData.getTemperature();
            this.humidity = weatherData.getHumidity();
            display();
        }
    }

    public void display() {
        System.out.println("Current conditions: " + temperature
                + "F degrees and " + humidity + "% humidity");
    }
}
```

    - **StatisticsDisplay**

```
import java.util.Observable;
import java.util.Observer;

public class StatisticsDisplay implements Observer, DisplayElement {
    private float maxTemp = 0.0f;
    private float minTemp = 200;
    private float tempSum = 0.0f;
    private int numReadings;

    public StatisticsDisplay(Observable observable) {
        observable.addObserver(this);
    }

    public void update(Observable observable, Object arg) {
        if (observable instanceof WeatherData) {
            WeatherData weatherData = (WeatherData) observable;
            float temp = weatherData.getTemperature();
            tempSum += temp;
```

```
            numReadings++;

            if (temp > maxTemp) {
               maxTemp = temp;
            }

            if (temp < minTemp) {
               minTemp = temp;
            }

            display();
         }
      }

      public void display() {
         System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)
               + "/" + maxTemp + "/" + minTemp);
      }
}
```

- **ForecastDisplay**

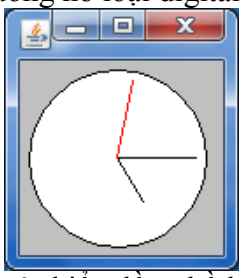
```
import java.util.Observable;
import java.util.Observer;

public class ForecastDisplay implements Observer, DisplayElement {
   private float currentPressure = 29.92f;
   private float lastPressure;

   public ForecastDisplay(Observable observable) {
      observable.addObserver(this);
   }

   public void update(Observable observable, Object arg) {
      if (observable instanceof WeatherData) {
         WeatherData weatherData = (WeatherData) observable;
         lastPressure = currentPressure;
         currentPressure = weatherData.getPressure();
         display();
      }
   }

   public void display() {
      System.out.print("Forecast: ");
      if (currentPressure > lastPressure) {
         System.out.println("Improving weather on the way!");
      } else if (currentPressure == lastPressure) {
         System.out.println("More of the same");
      } else if (currentPressure < lastPressure) {
         System.out.println("Watch out for cooler, rainy weather");
      }
   }
}
```

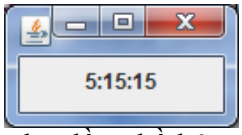
– ***WeatherStation*** test class is similar

Note:

What are the differences of your Observer pattern implementation and Java build-in one? Which one is refer to use?

Bài 2:
Viết chương trình hiển thị 2 đồng hồ loại digital và analog cho thời gian hệ thống.





Thiết kế sao cho có thể thêm các hiển đồng hồ khác như đồng hồ kép, đồng hồ cho một thành phố nào đó.

Bài 3: