

Question 2 (20 pts)

Answer True or False.

- a) Always code to an implementation, not an interface.
- b) Classes should be open for modification but closed for extension.
- c) The Decorator pattern attaches additional responsibilities to an object dynamically.
- d) Organize projects around functionality, not job functions.
- e) The Observer pattern defines a one-to-many relationship between a set of objects.

Question 3 (20 pts)

A Use Case Diagram is a diagram that shows a set of use cases and actors and their relationships. You apply use case diagrams to visualize the behavior of a system so that users can comprehend how to use that element, and so that developers can implement that element.

Create a Use Case Diagram to model the behavior of a cellular phone. Include actors, use cases, and dependencies in your diagram.

Question 4 (20 pts)

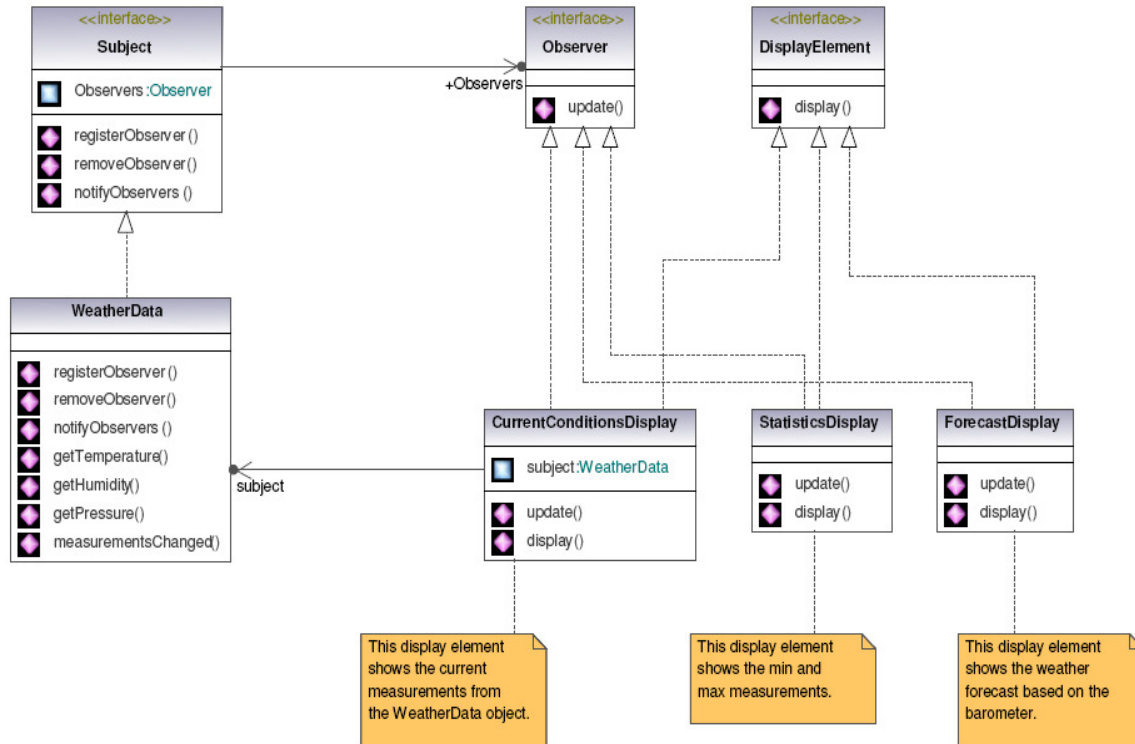
A residential home builder has moved into the Bozeman area. The competition is tough, as there are other builders in the area that also produce high quality homes. The new builder's company is named *Bobcat Homes*, and they have decided to only offer 3 basic models. Buyers however have the option to customize their home by choosing higher quality flooring, lighting, landscaping, and/or appliances. The value of a home is dependent on which of the base models is chosen, and on which upgrades a customer uses to customize their home.

a) Draw a **UML Class Diagram** of your solution. **(15 pts)**

b) In Java, write the code to buy a fully landscaped home with upgraded appliances and lighting. The buyer does not care which of the 3 basic models is chosen. **(5 pts)**

Question 5 (20 pts)

Consider the following UML Class diagram:



And consider the Java code for the UML Class diagram:

```
public interface Subject {
    public void registerObserver(Observer o);
    public void removeObserver(Observer o);
    public void notifyObservers();
}

public interface Observer {
    public void update(float temp, float humidity, float pressure);
}

public interface DisplayElement {
    public void display();
}
```

```
public class WeatherData implements Subject {
    private ArrayList observers;
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() {
        observers = new ArrayList();
    }

    public void registerObserver(Observer o) {
        observers.add(o);
    }

    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }

    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer = (Observer)observers.get(i);
            observer.update(temperature, humidity, pressure);
        }
    }

    public void measurementsChanged() {
        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;
    }
}
```

```

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

    public CurrentConditionsDisplay(Subject weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(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");
    }
}

public class WeatherStation {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();

        CurrentConditionsDisplay currentDisplay =
            new CurrentConditionsDisplay(weatherData);

        weatherData.setMeasurements(80, 65, 30.4f); // ← For part (c)
    }
}

```

a) What is the output when the main method is run. (5 pts)

b) In the picture provided for this question, add any missing multiplicities to the UML Class Diagram. (2 pts)

- c) Draw the **UML Sequence Diagram** starting at the point in the main function where the `setMeasurements()` API is called. This is indicated by an arrow (←) (13 pts).