Observer Pattern

Shin-Jie Lee (李信杰)
Associate Professor
Computer and Network Center
Department of CSIE
National Cheng Kung University

Design Aspect of Observer

Number of objects that depend on another object; how the dependent objects stay up to date

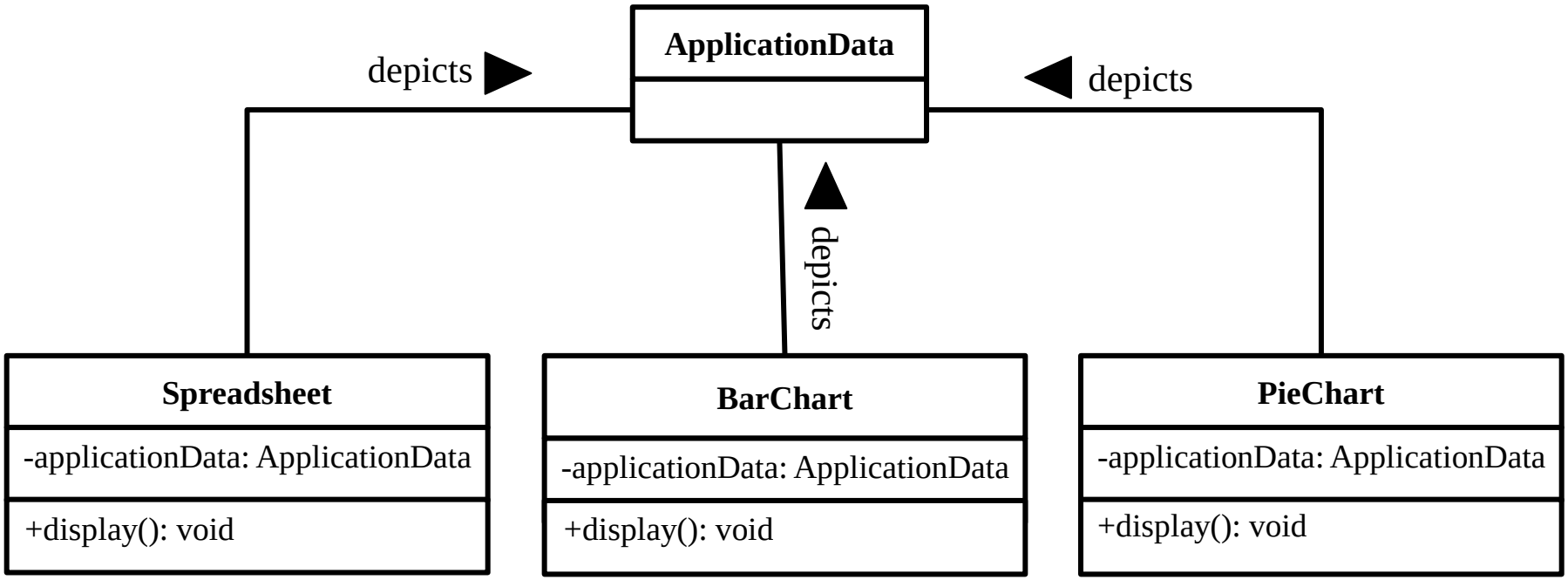
Spreadsheet Application (Observer)

Outline

- Requirements Statement
- Initial Design and Its Problems
- Design Process
- Refactored Design after Design Process
- Recurrent Problems
- Intent
- Observer Pattern Structure
- More Examples

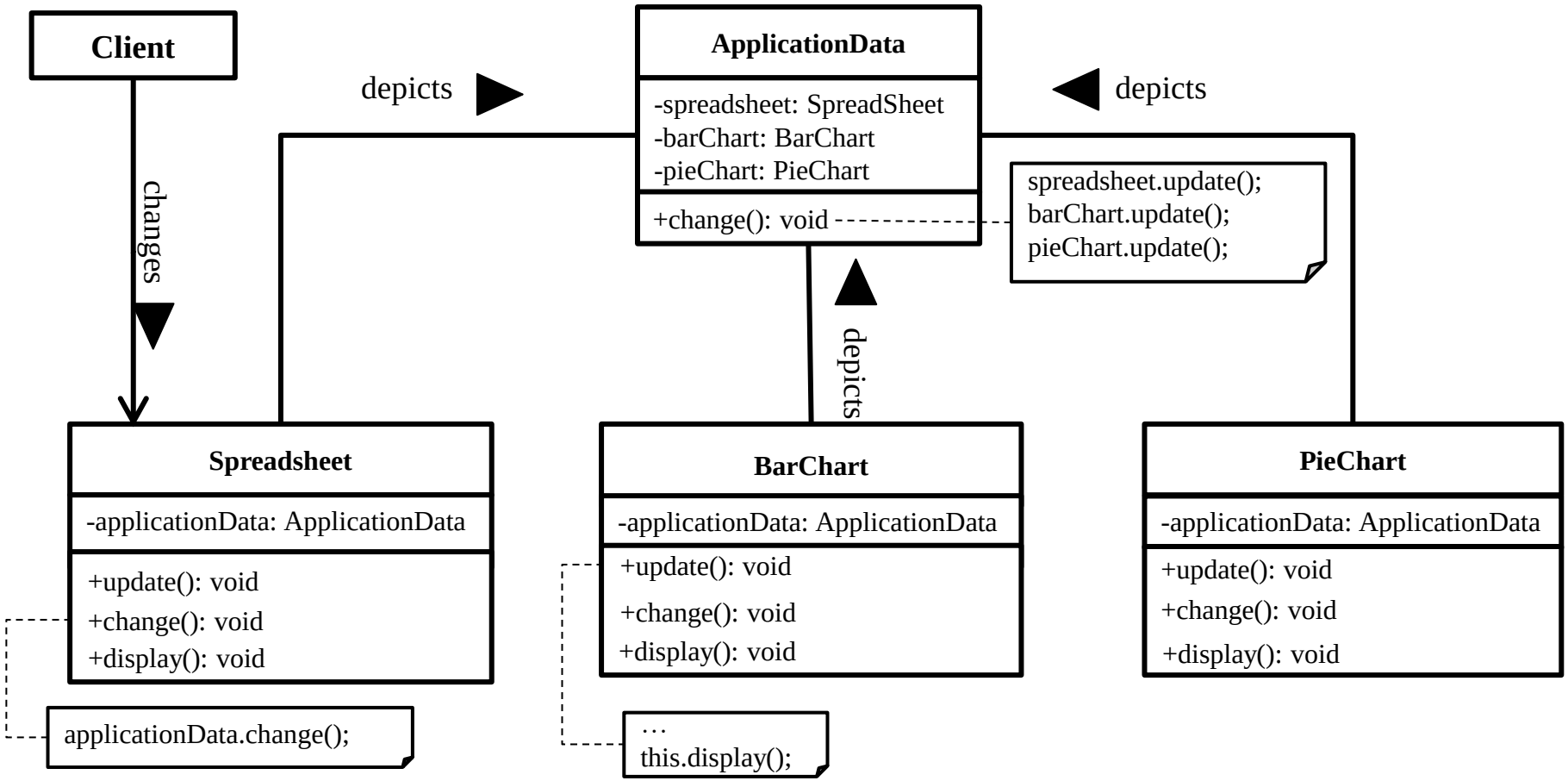
Requirements Statement1

- In a spreadsheet application,
  - A spreadsheet object, bar chart object, and pie chart object can depict information in the same application data object by using different presentations.

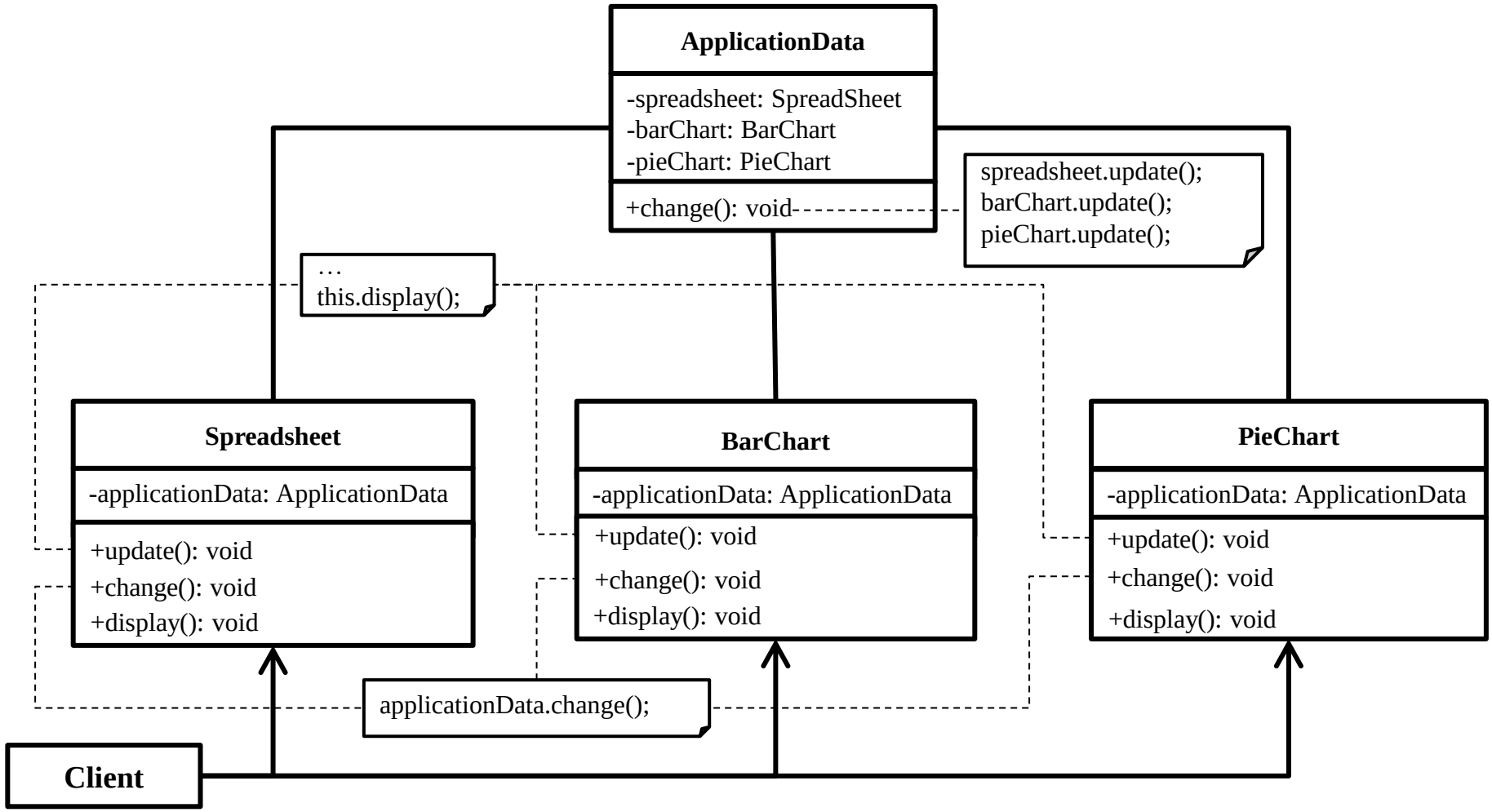


Requirements Statement2

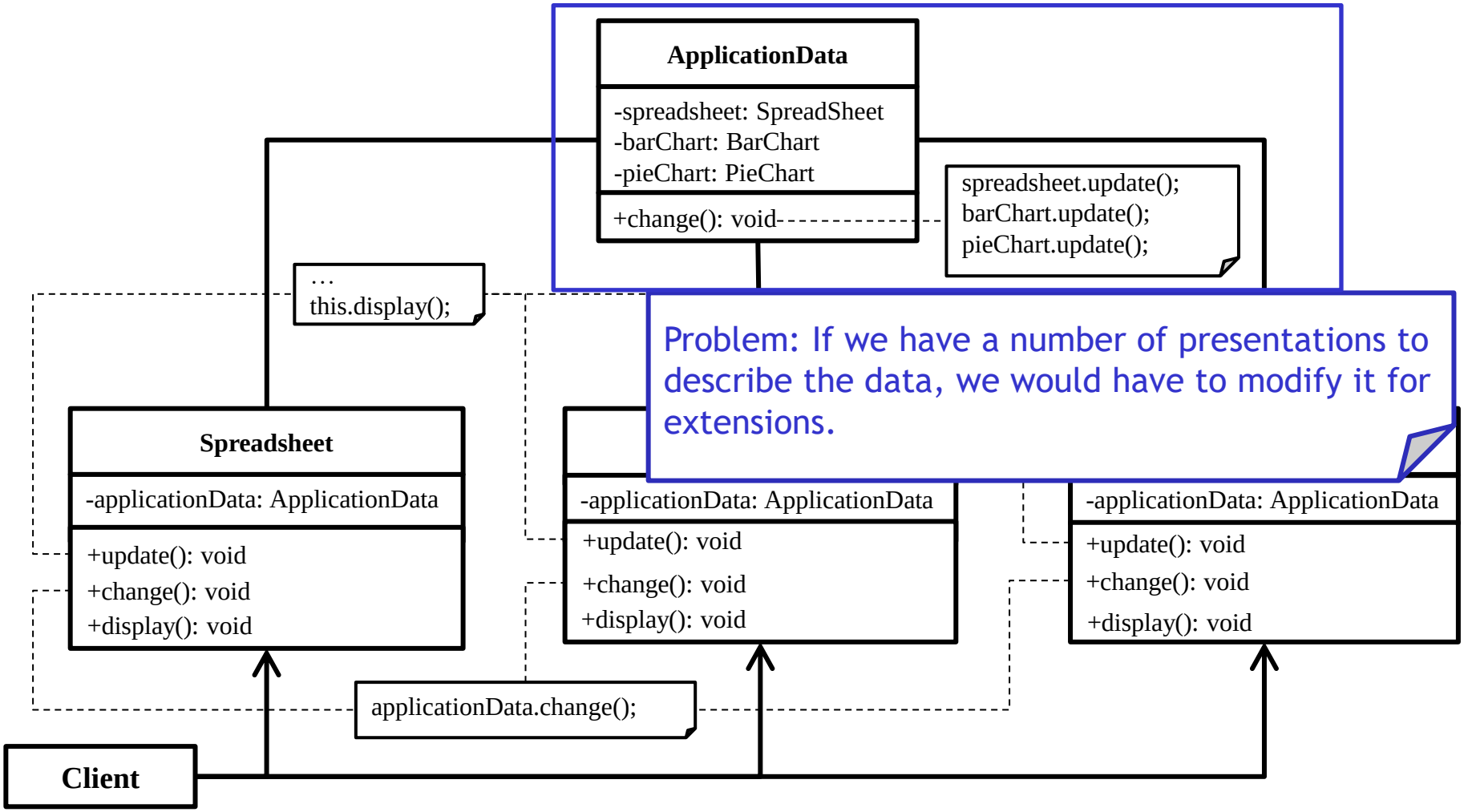
- When the user changes the information in the spreadsheet, the bar chart reflects the changes immediately, and vice versa.



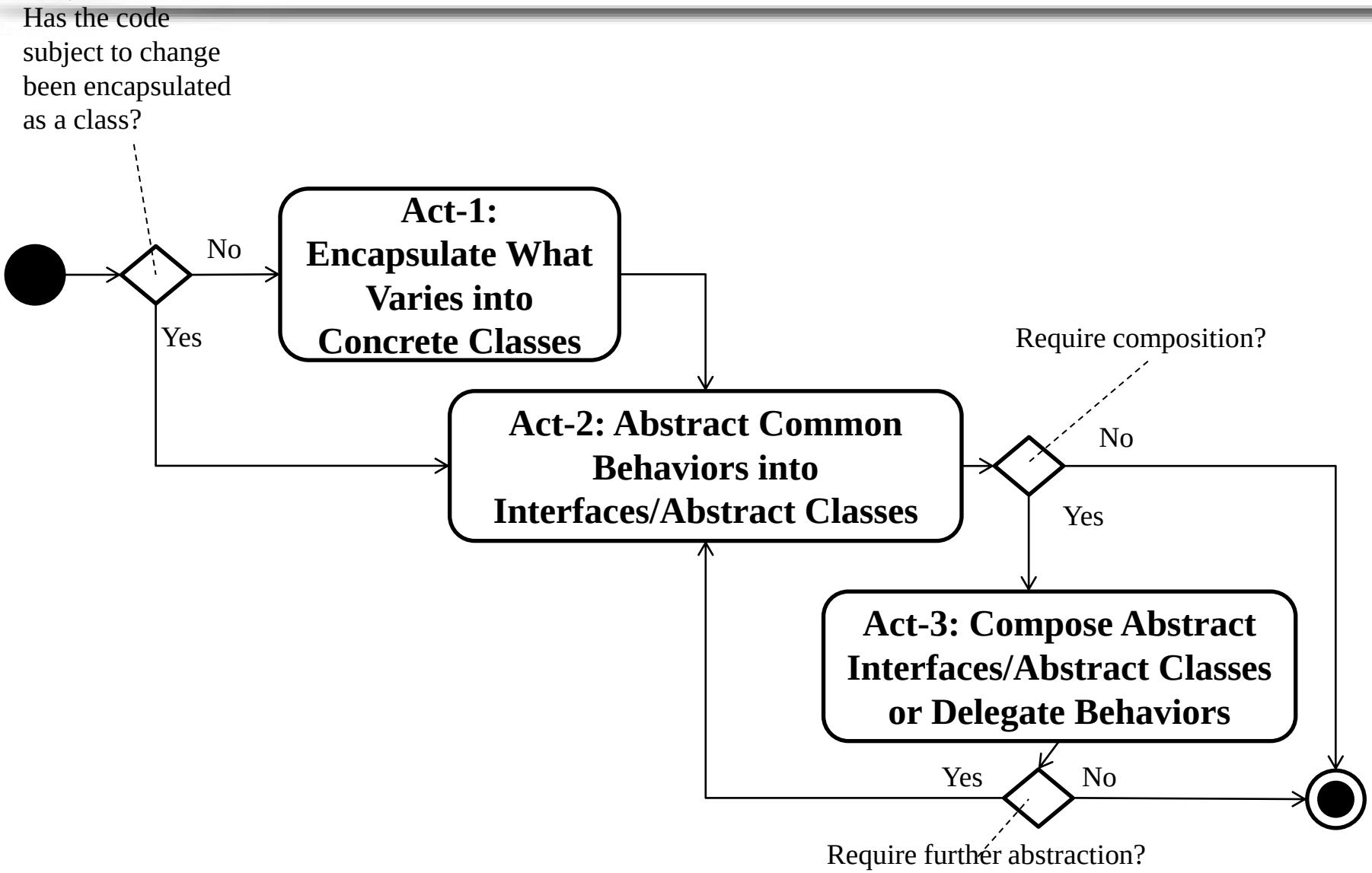
Initial Design



Problems with Initial Design

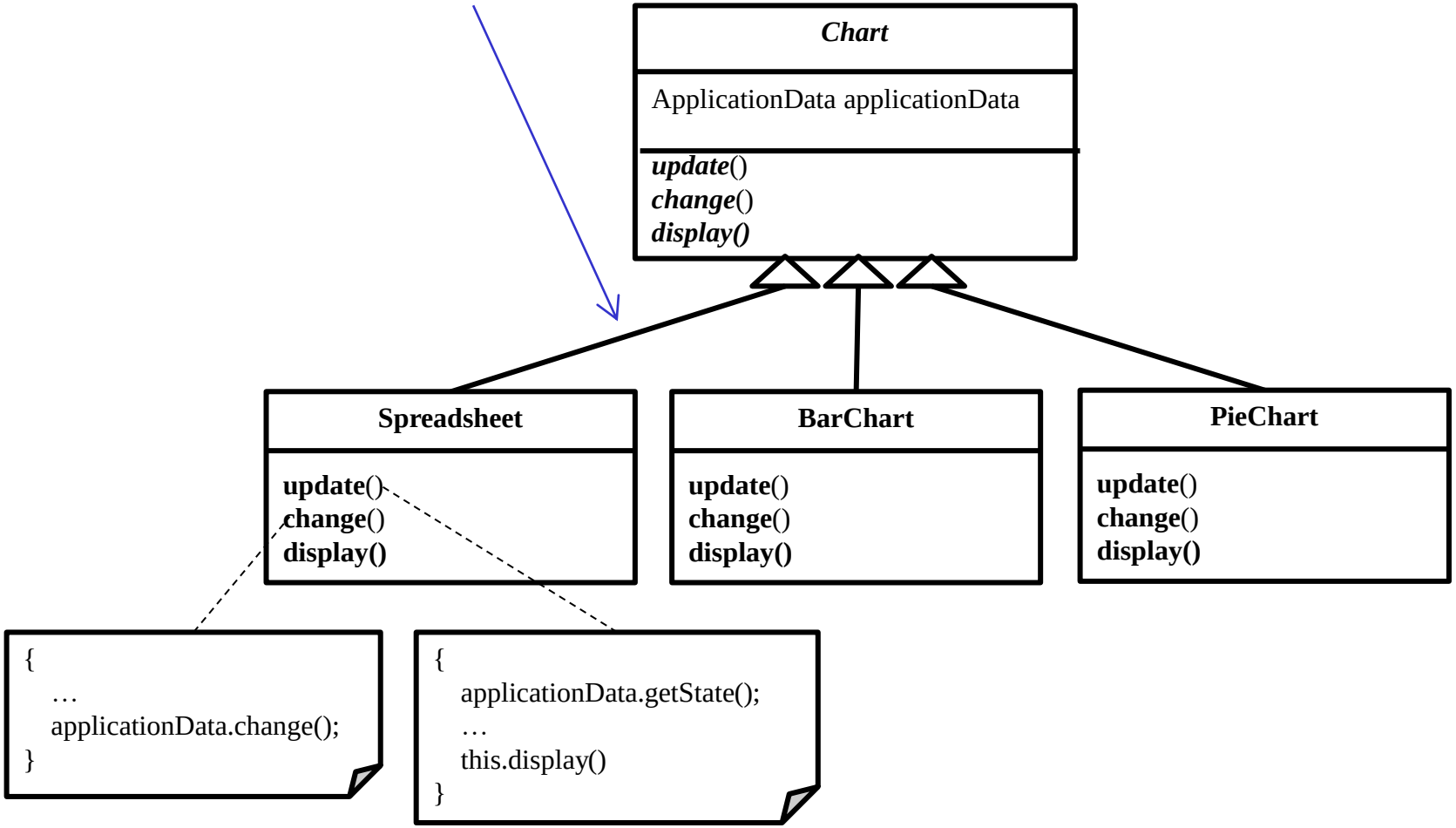


Design Process for Change



Act-2: Abstract Common Behaviors

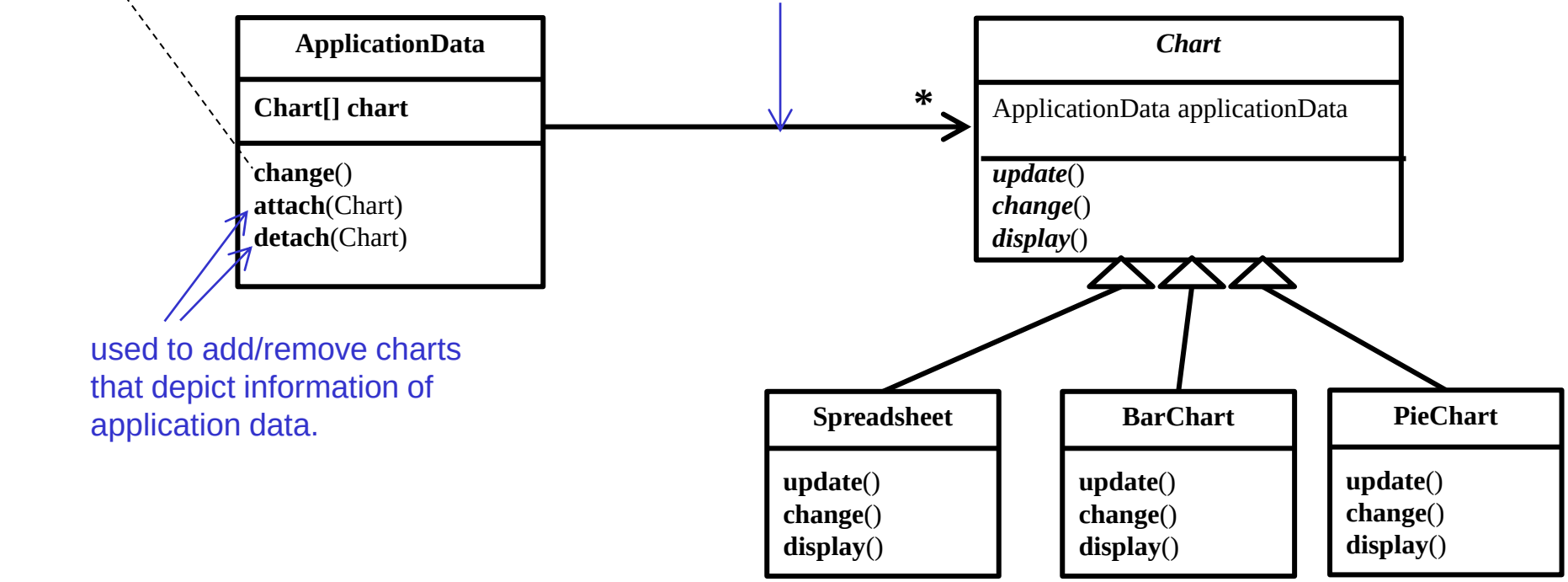
Act-2.2: Abstract common behaviors with a same signature into abstract class through inheritance



Act-3: Compose Abstract Behaviors

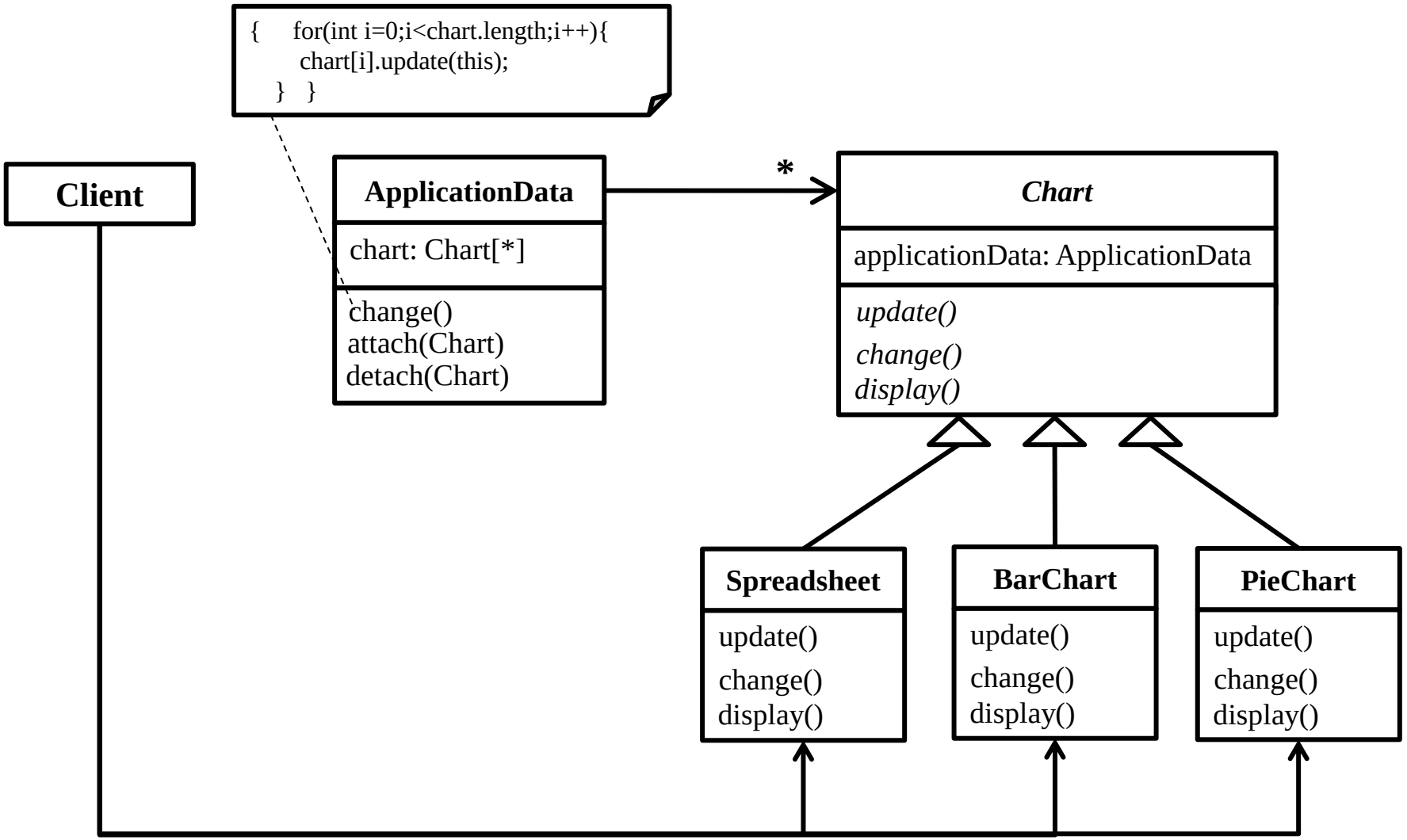
```
{    for(int i=0;i<chart.length;i++){
      chart[i].update(this);
   }  }
```

Act-3.1: Compose behaviors of an interface or an abstract class



used to add/remove charts that depict information of application data.

Refactored Design



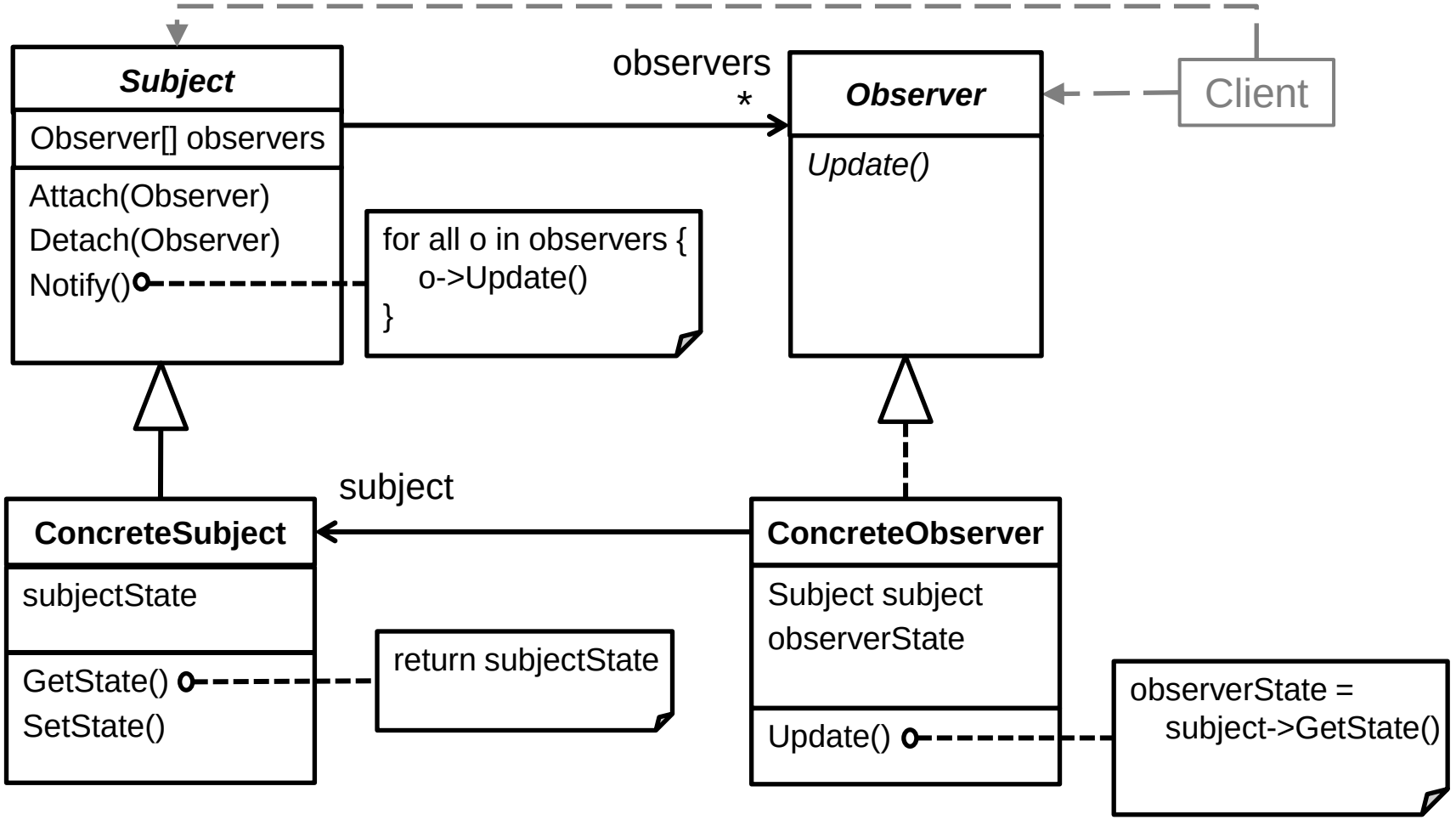
Recurrent Problem

- The code will be modified if a new display/presentation is going to be added.
  - Many graphical user interface toolkits separate the presentational aspects of the user interface from the underlying application data.
  - Classes defining application data and presentations can be reused independently.

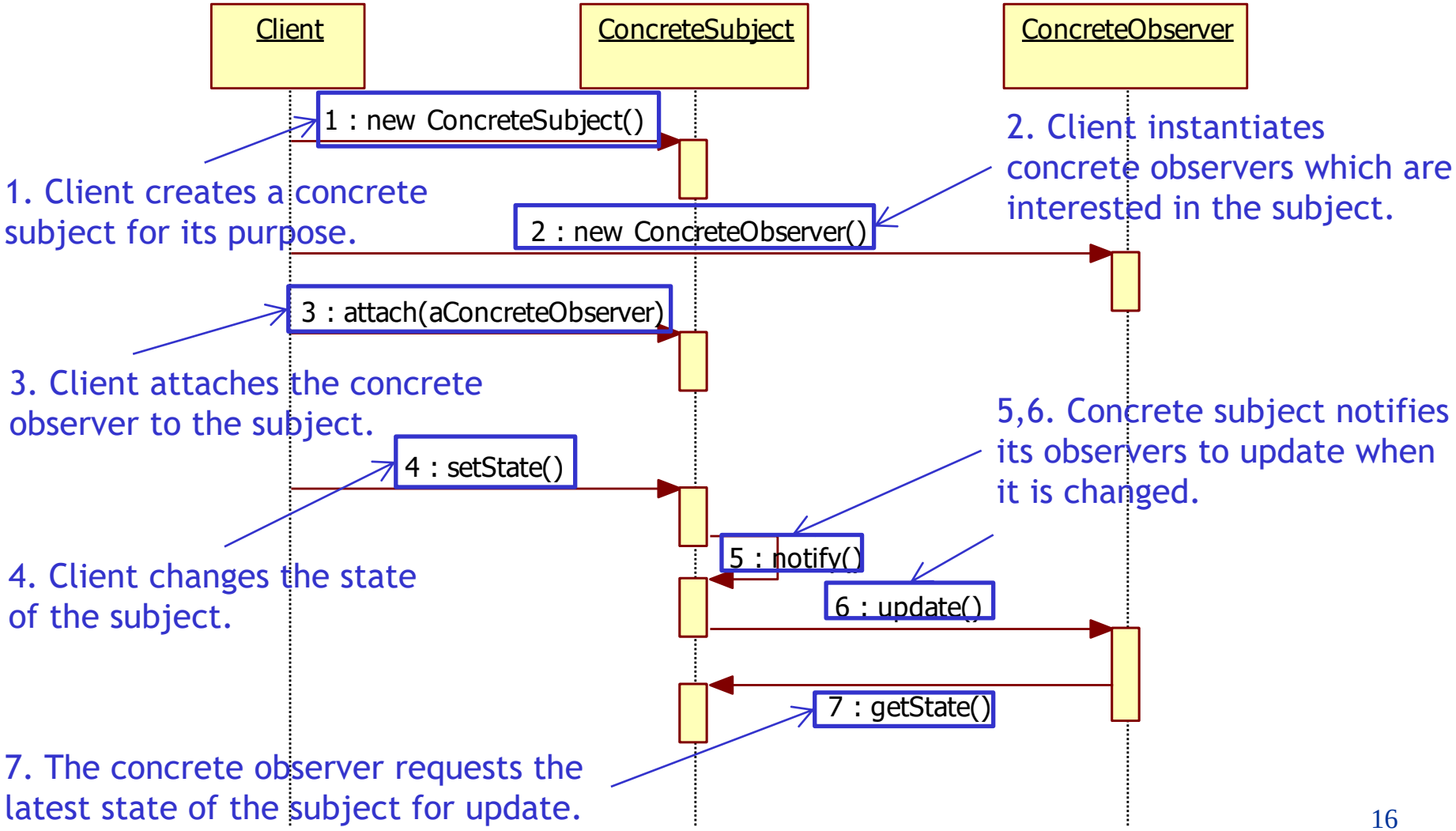
Observer Pattern

- Intent
  - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

Observer Pattern Structure1



Observer Pattern Structure4



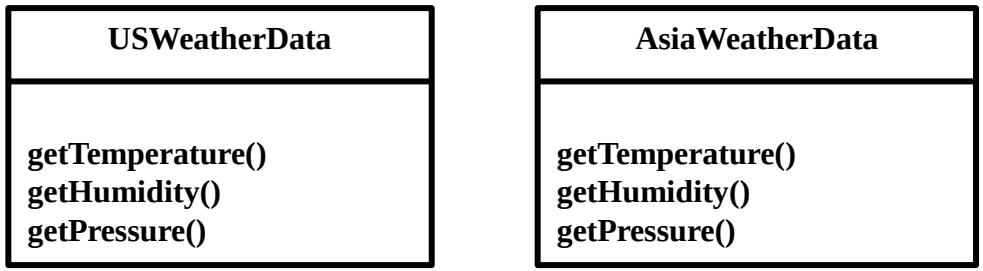
Observer Pattern Structure4

	Instantiation	Use	Termination
Client	Other class except classes in observer pattern	Other class except classes in the observer pattern	Other class except classes in the observer pattern
Subject	X	Client uses this abstract class to attach and detach ConcreteObserver	X
Concrete Subject	The client class or other class except classes in the observer pattern	ConcreteObserver or other classes use this class to set/get state of ConcreteSubject	Classes who hold the reference of ConcreteSubject
Observer	X	Subject uses this interface to notify ConcreteObserver to update the state of ConcreteSubject through polymorphism	X
Concrete Observer	The client class or other class except classes in the observer pattern	Subject notifies this class which is attached by client to update the state of ConcreteSubject through polymorphism	Classes who hold the reference of ConcreteObserver

Weather Broadcast (Observer)

Requirements Statement$_1$

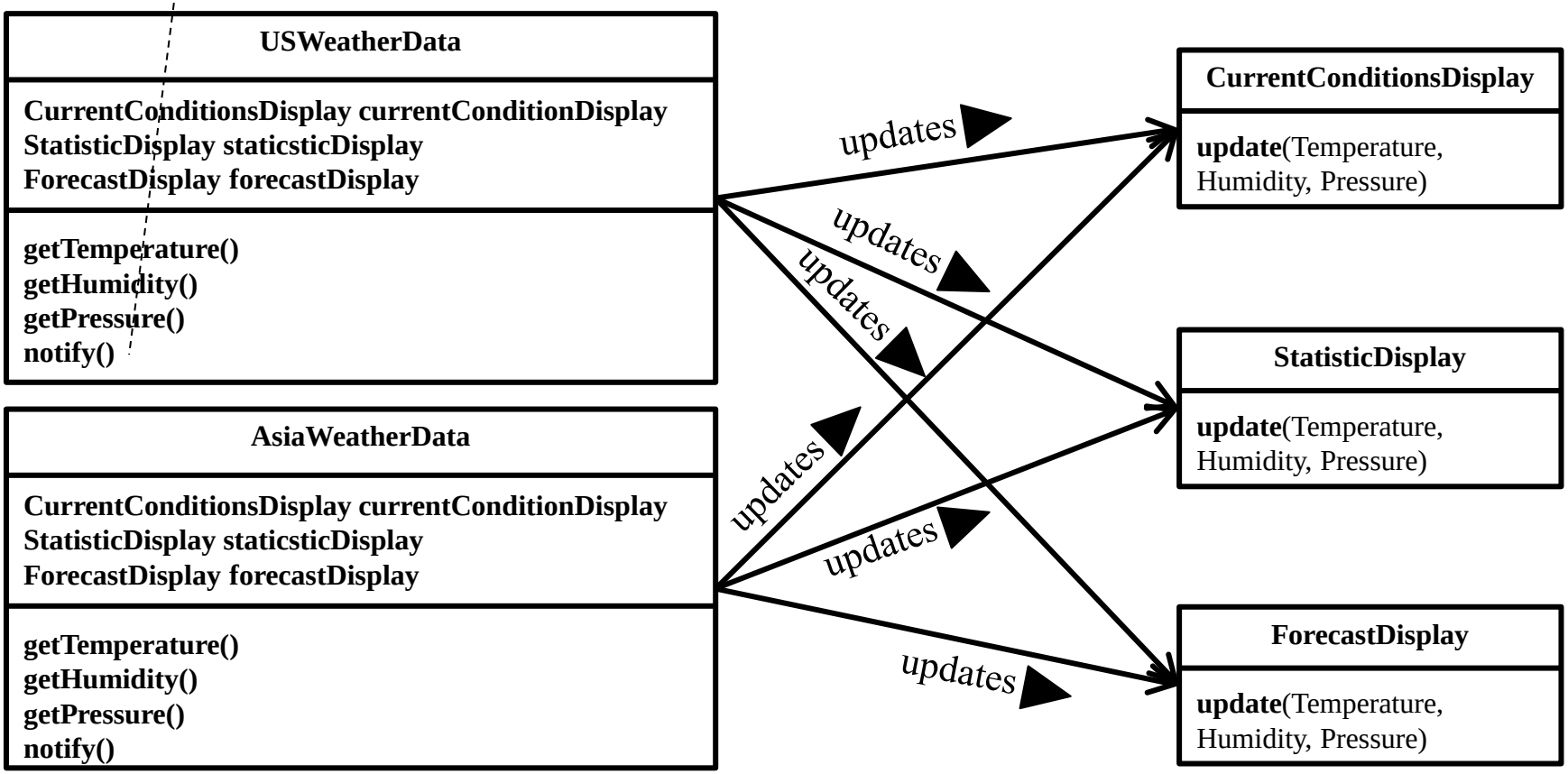
- Weather Monitoring Station System
  - The system will be based on WeatherData object, which tracks current weather conditions (temperature, humidity, and barometric pressure) in a specific area (e.g. U.S. or Asia).



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

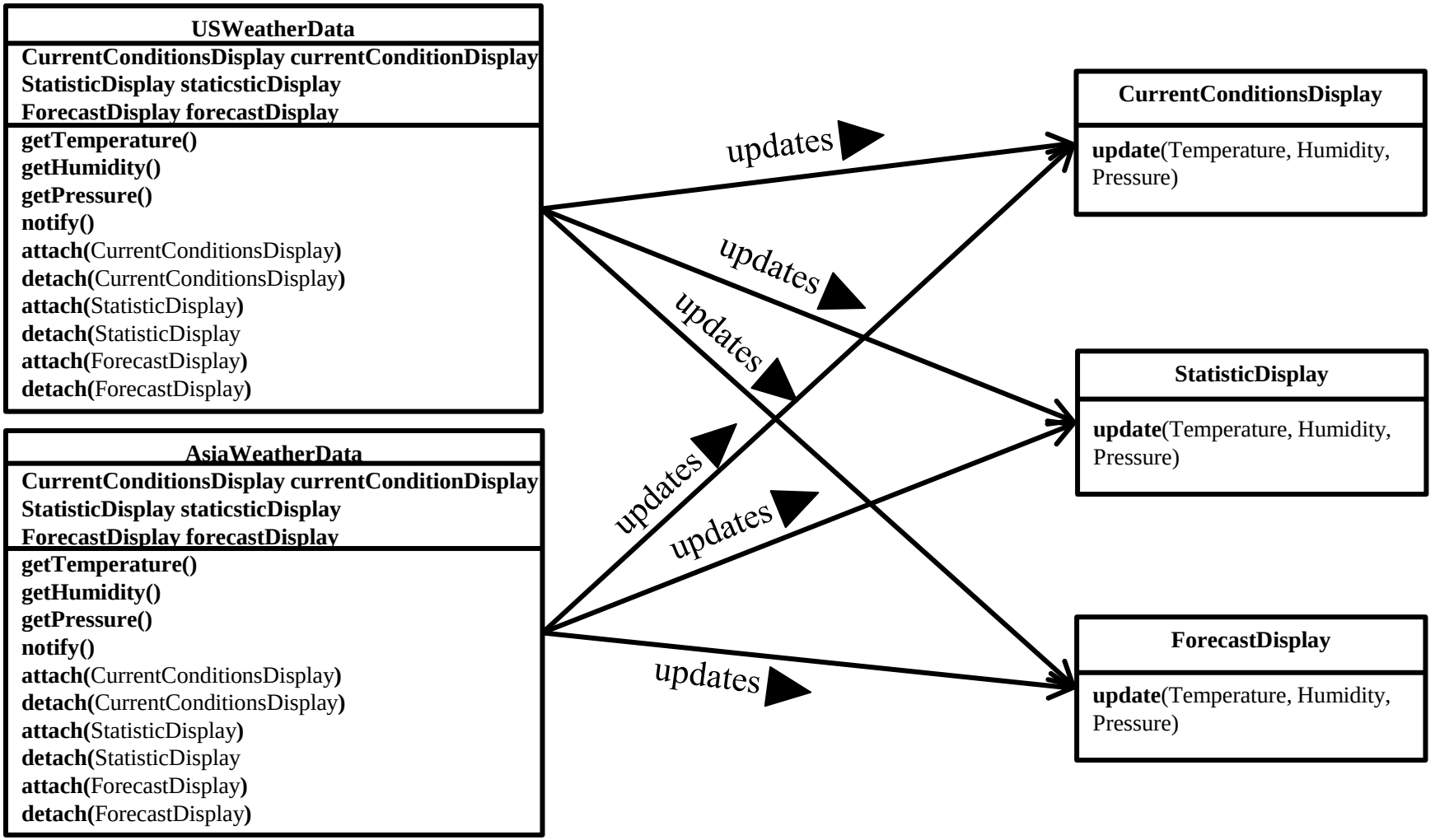
atement$_2$

- The system initially provides three display elements: current conditions, weather statistics and a simple forecast, all updated in real time as the WeatherData object acquires the most recent measurements.



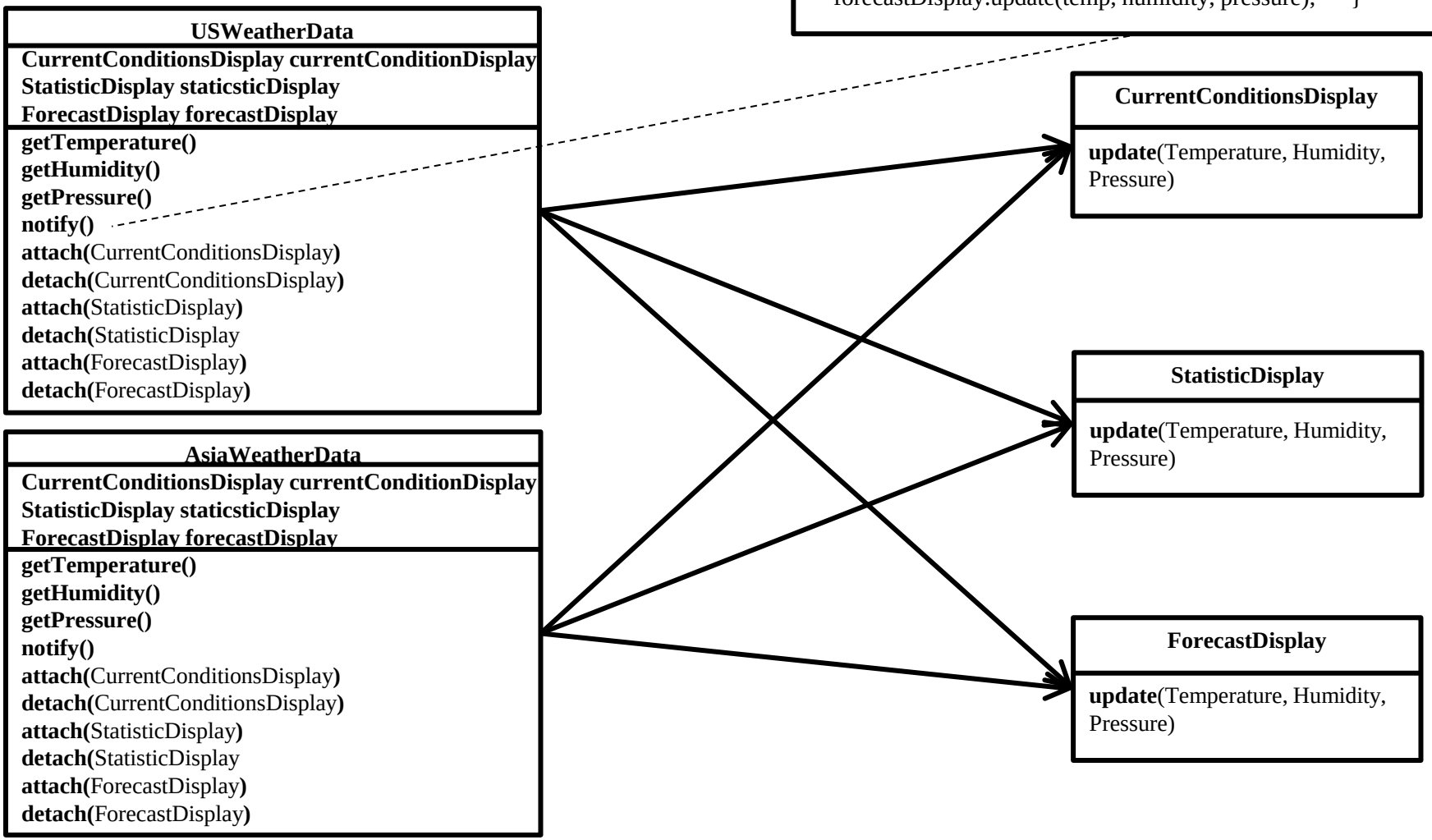
Requirements Statement$_3$

- The system should supply an API so that other developers can write their own weather displays and plug them right in.



Initial Design - Class Diagram

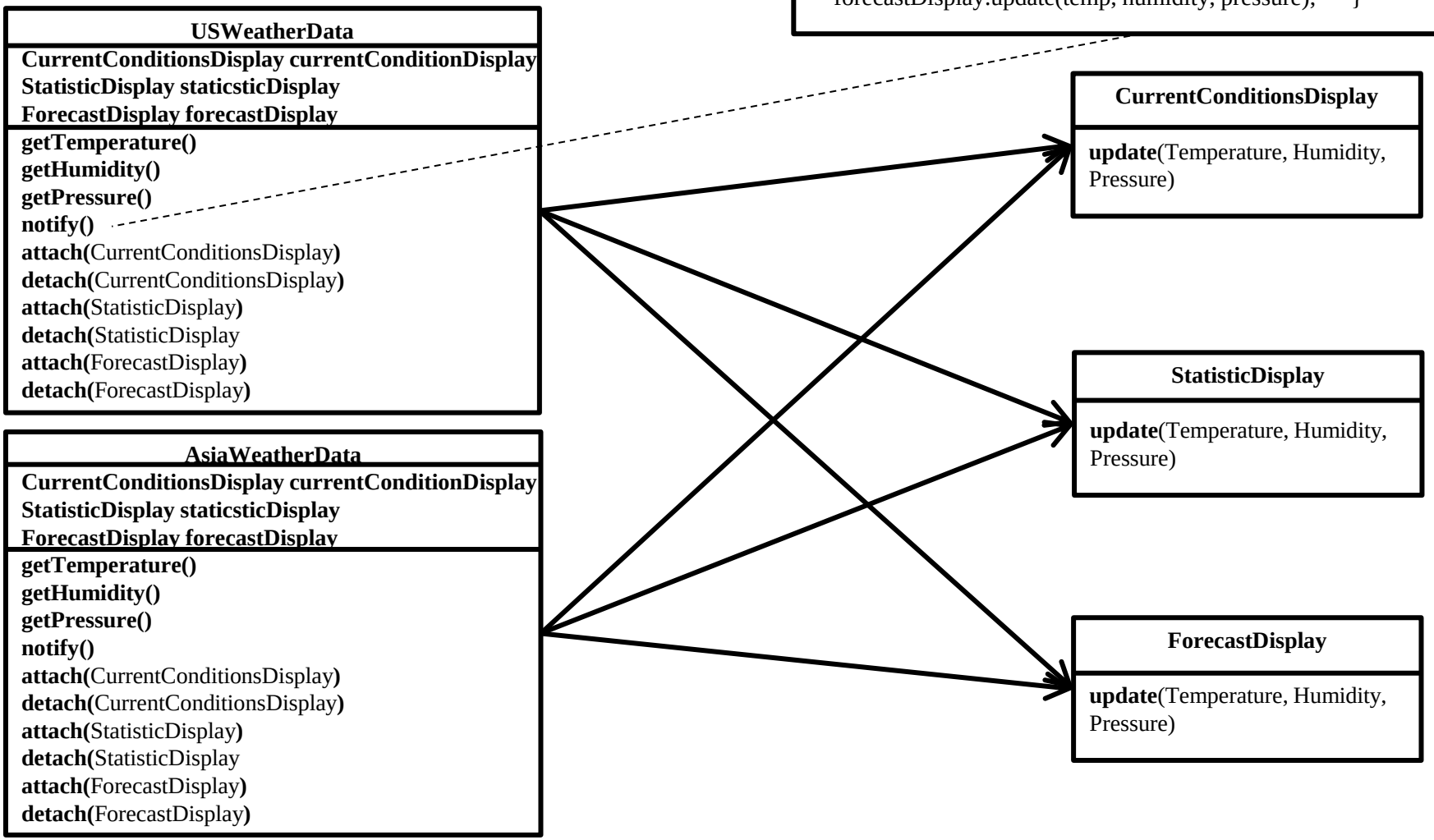
```
{    float temp = getTemperature();
   float humidity = getHumidity();
   float pressure = getPressure();
   currentConditionsDisplay.update(temp,humidity, pressure);
   statisticsDisplay.update(temp, humidity, pressure);
   forecastDisplay.update(temp, humidity, pressure);      }
```



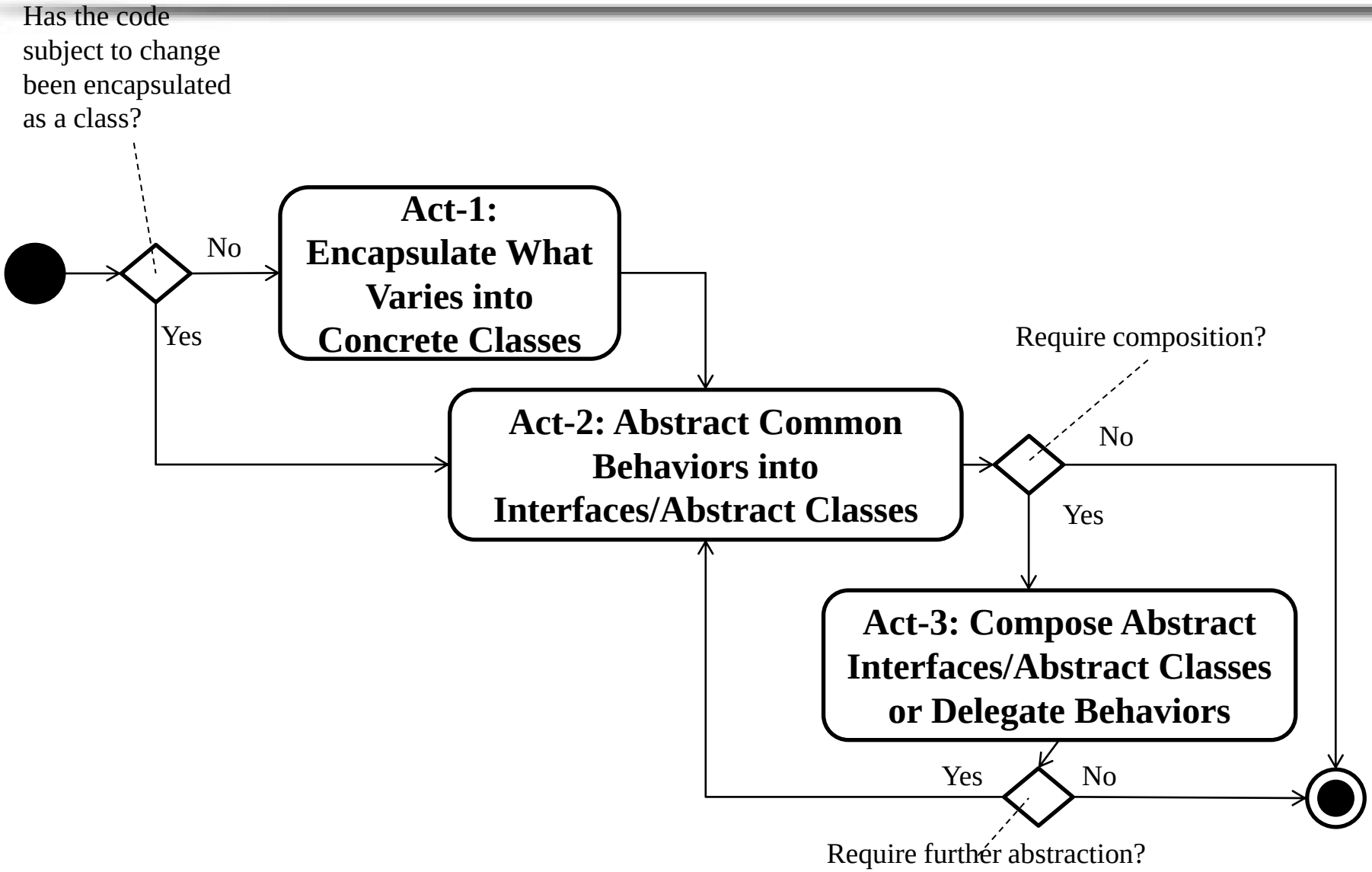
Problems with Initial Design

Problem: The code will be modified if a new display is going to be added.

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

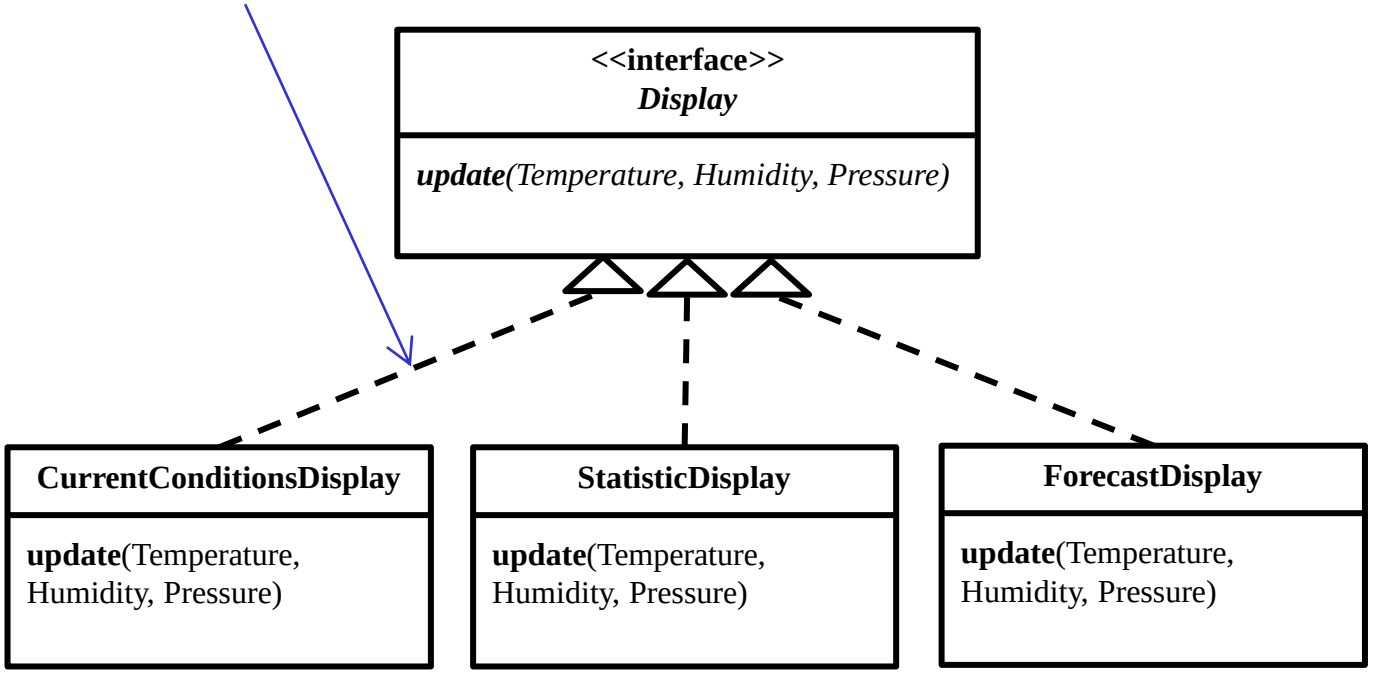


Design Process for Change



Act-2: Abstract Common Behaviors

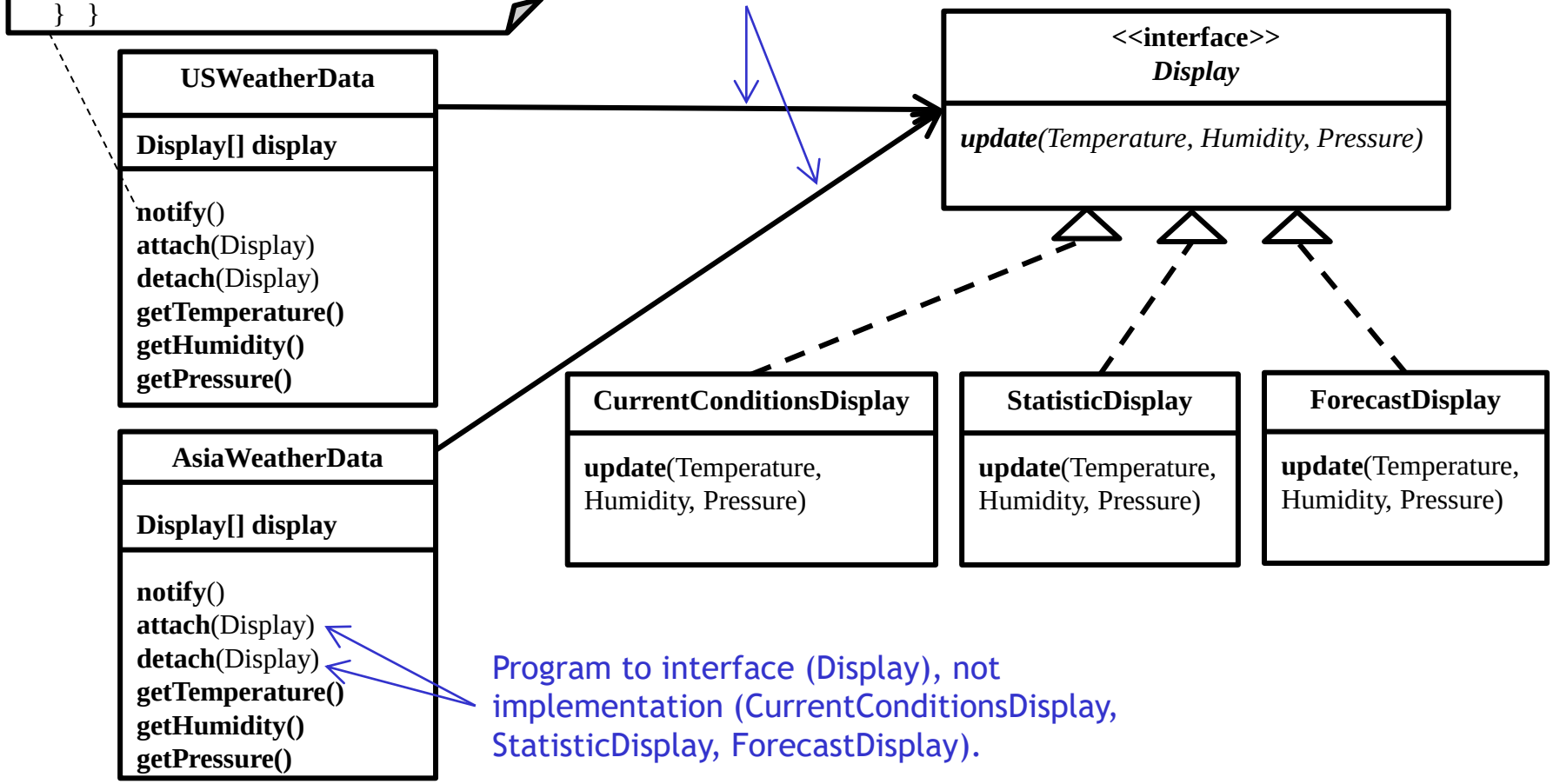
Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism



Act-3: Compose Abstract Behaviors

```
{    for(int i=0;i<observer.length;i++){
       observer[i].update(getTemperature(),
                          getHumidity(),
                          getPressure());
  }  }
```

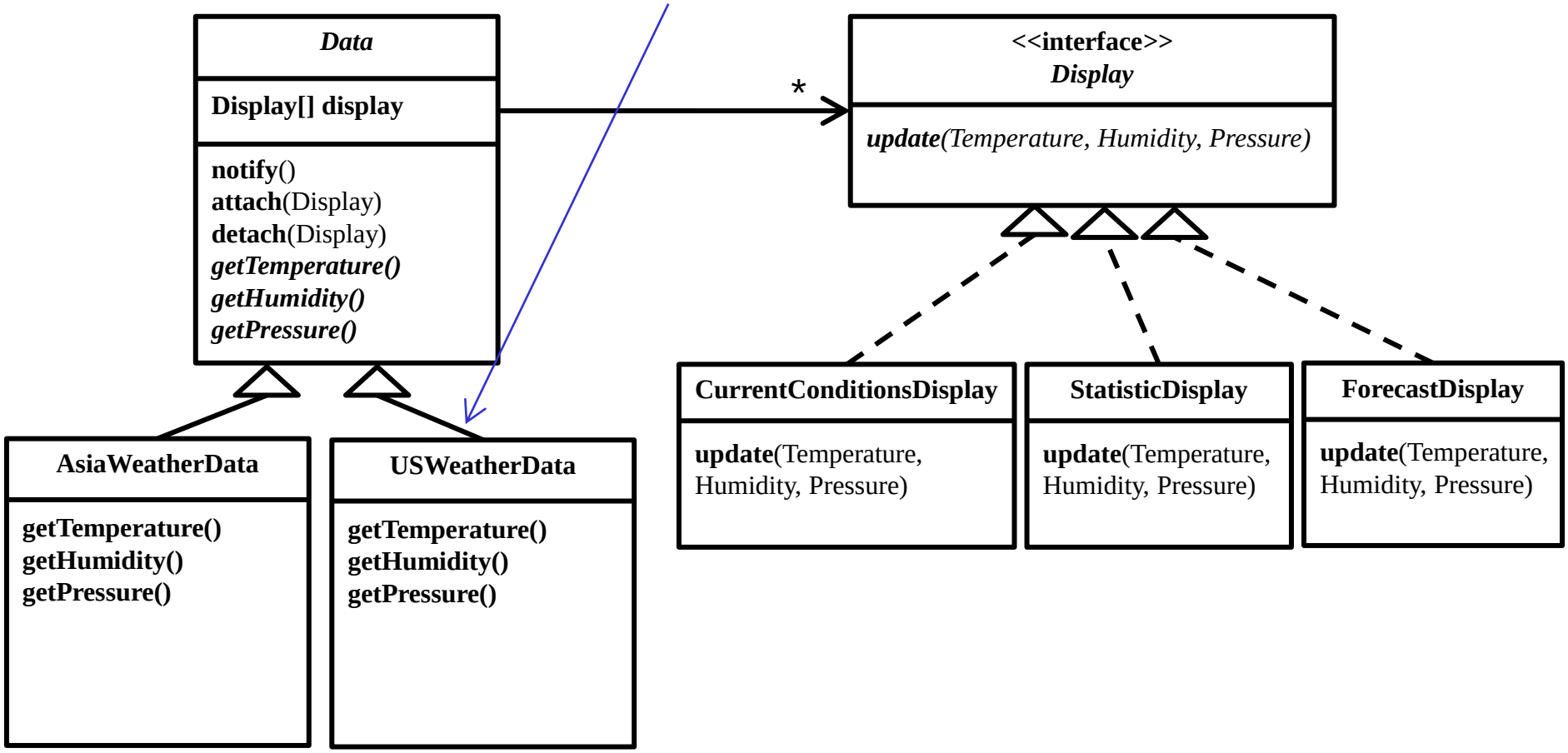
Act-3.1: Compose behaviors of an interface or an abstract class



Program to interface (Display), not implementation (CurrentConditionsDisplay, StatisticDisplay, ForecastDisplay).

Act-2: Abstract Common Behaviors

Act-2.2: Abstract common behaviors with a same signature into abstract class through inheritance



Refactored Design after Design Process