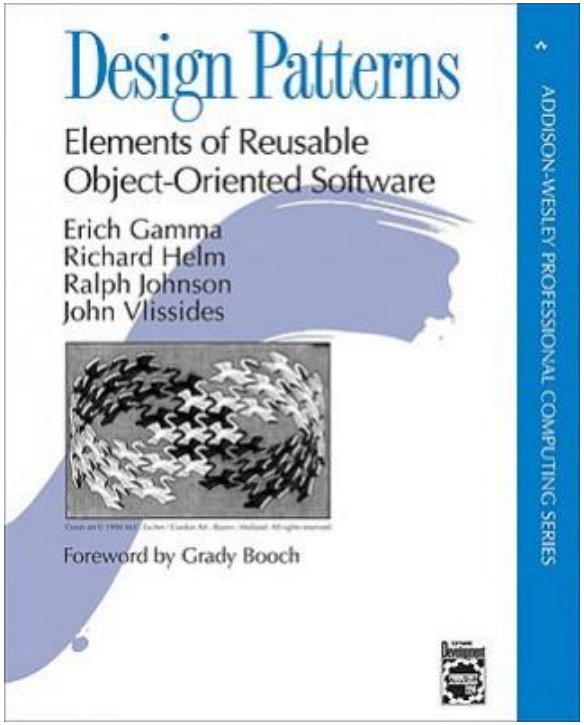
The Observer Design Pattern

- [GoF] Erich Gama, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software" – chapter 5
- Eric Freeman, Elisabeth Robson, "Head First Design Patterns", 2nd Ed, 2020 – chapter 2



Observer - Motivation

- **Observer – Motivation:**
- Need to maintain consistency between related cooperating objects, but without their classes tightly coupled
- Example: graphical user interface classes <-> application data

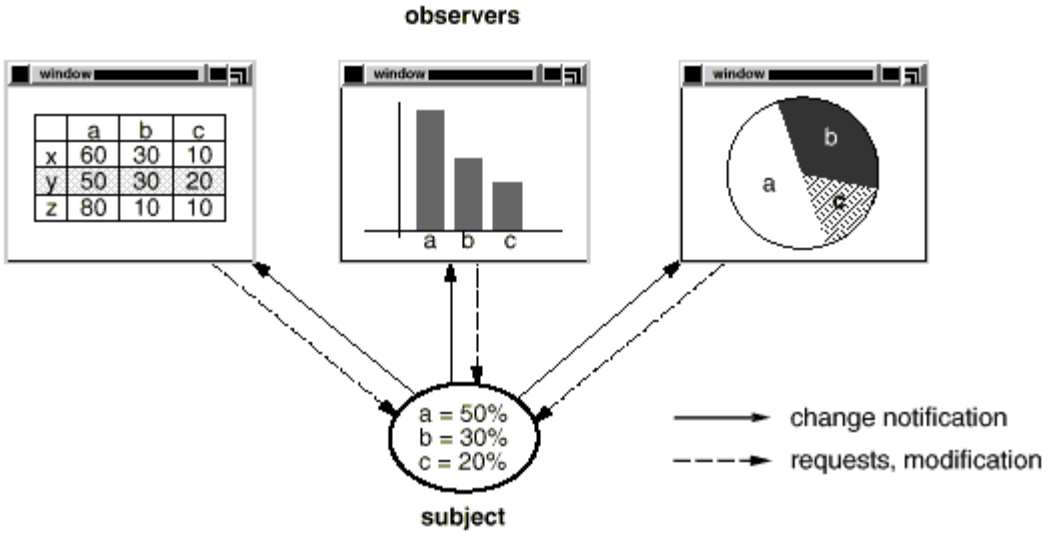


Figure from [GoF]

Observer - Intent

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

- Example: the different graphical views are dependent on the data object and therefore should be notified of any changes in its state

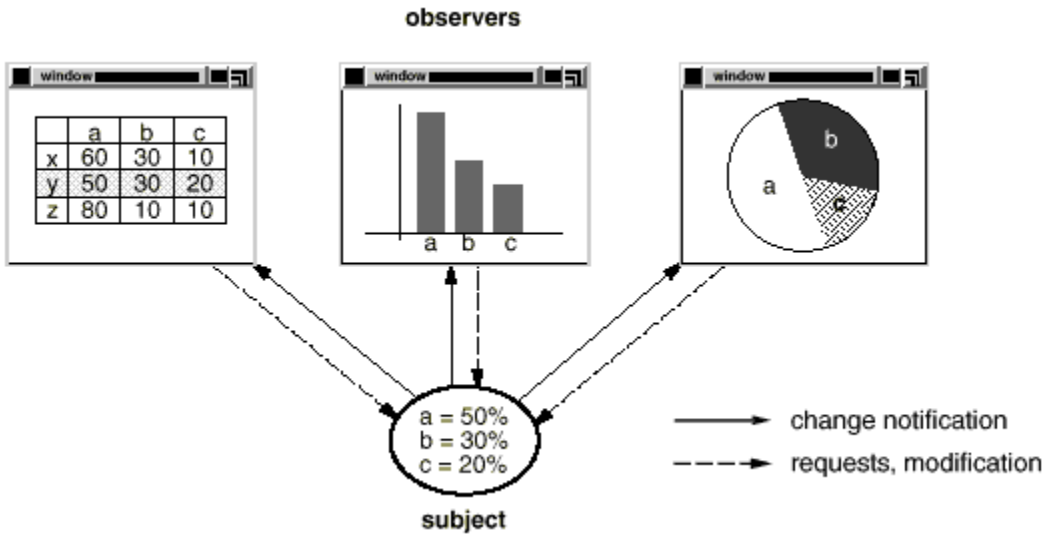
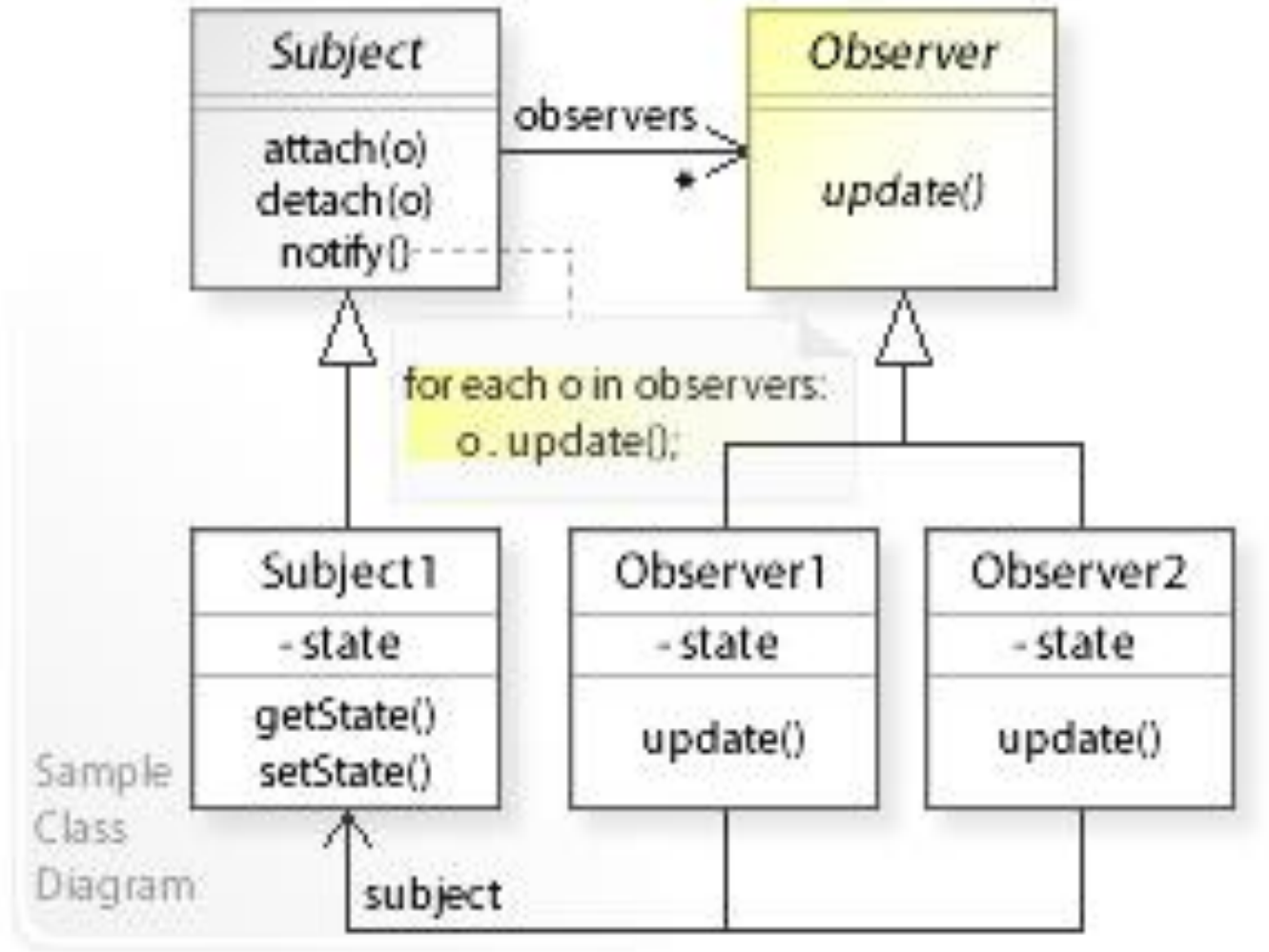


Figure from [GoF]

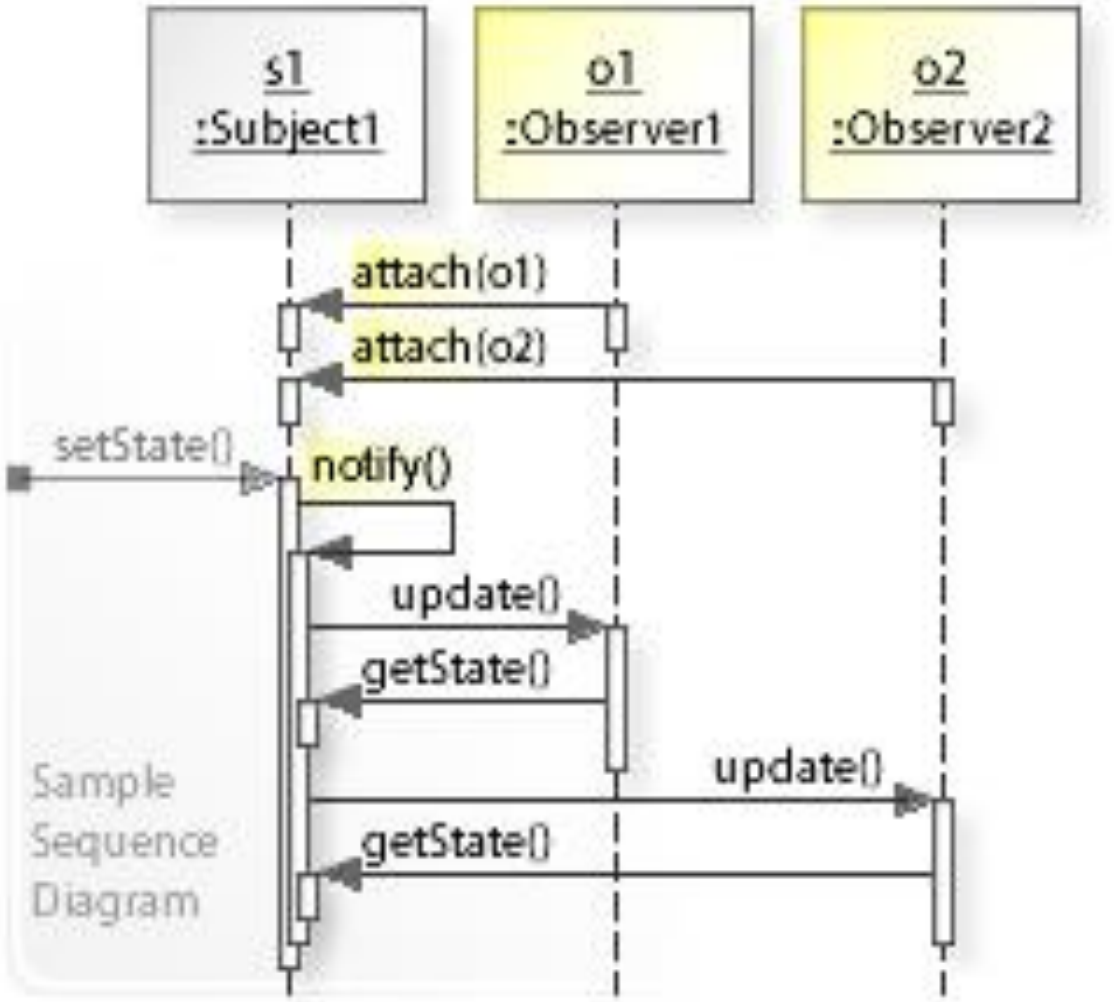
Observer – Participants

- **Subject:**
  - knows its observers.
  - any number of Observer objects may observe a subject.
  - provides an interface for attaching and detaching Observer objects.
- **Observer:**
  - defines an updating interface for objects that should be notified of changes in a subject.
- **ConcreteSubject:**
  - stores state of interest to ConcreteObserver objects.
  - sends a notification to its observers when its state changes.
- **ConcreteObserver:**
  - maintains a reference to a ConcreteSubject object.
  - stores state that should stay consistent with the subject's.
  - implements the Observer updating interface to keep its state consistent with the subject's.

Observer - Structure



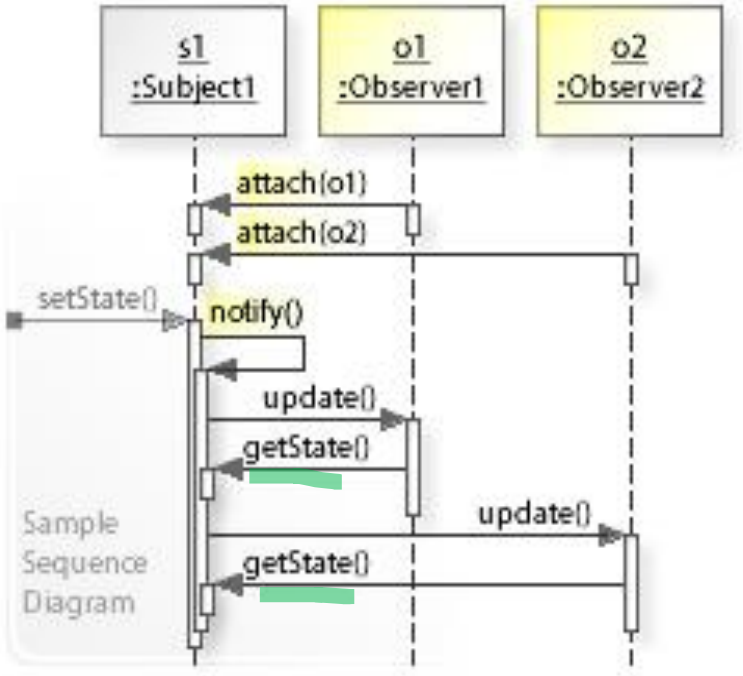
Observer - Behaviour



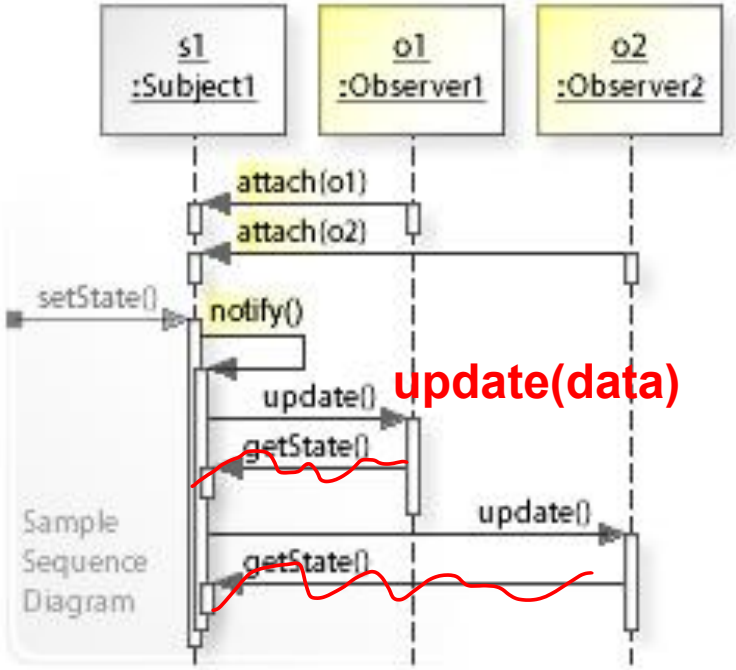
Observer – Implementation

- The Subject calls the update() method on all the Observers, in order to communicate that there was a change in the Subject's state.
- How does the Observer get the data that was changed?
  - 2 approaches:
    1. The **PULL** approach: the subject only sends the notification that a change happened, then the concrete observer explicitly pulls the data from the concrete subject:
       - The concrete observer gets all the data from the concrete subject by calling getter methods on the concrete subject.
       - The concrete observer must have/must get a reference to the concrete subject
       - The update() method in the Observer interface has as argument the Subject object
       - The diagrams from [GoF] depict the PULL approach
    2. The **PUSH** approach: the subject sends all the possible data together with the notification:
       - the update() method in the Observer interface has as arguments the data transmitted from the subject to the observer.
       - The concrete observer does not need to call getter methods on the concrete subject.
       - The concrete observer does not need a reference to the concrete subject while updating
       - It is easier to implement

Observer – PULL vs PUSH



In the **PULL** variant, the update() method of a ConcreteObserver calls a getState() on a ConcreteSubject. It needs to receive a reference to the ConcreteSubject



In the **PUSH** variant, the update() method of a ConcreteObserver does not need to know a ConcreteSubject. It gets all the state data from the ConcreteSubject as argument of the update() method.

Observer - Example

- A temperature sensor gives the current value of the measured temperature.
- The measured temperature values are needed by different displays:
  - A NumericDisplay
  - A TextDisplay
  - An AverageDisplay
- Every time the sensor data changes, all the attached displays must update
- Displays can be dynamically attached and detached to a sensor
- New types of displays can be invented
- Displays must be reusable together with another type of sensor (pressure sensor)

- Solution: Observer pattern:
  - ConcreteSubject = TemperatureSensor
  - ConcreteObservers = NumericDisplay, TextDisplay, AverageDisplay

- Implementation decision: Choose the PUSH approach for passing temperature values
  - Concrete observers do not need a reference to concrete subject
  - The same observer can be attached to several subjects simultaneously

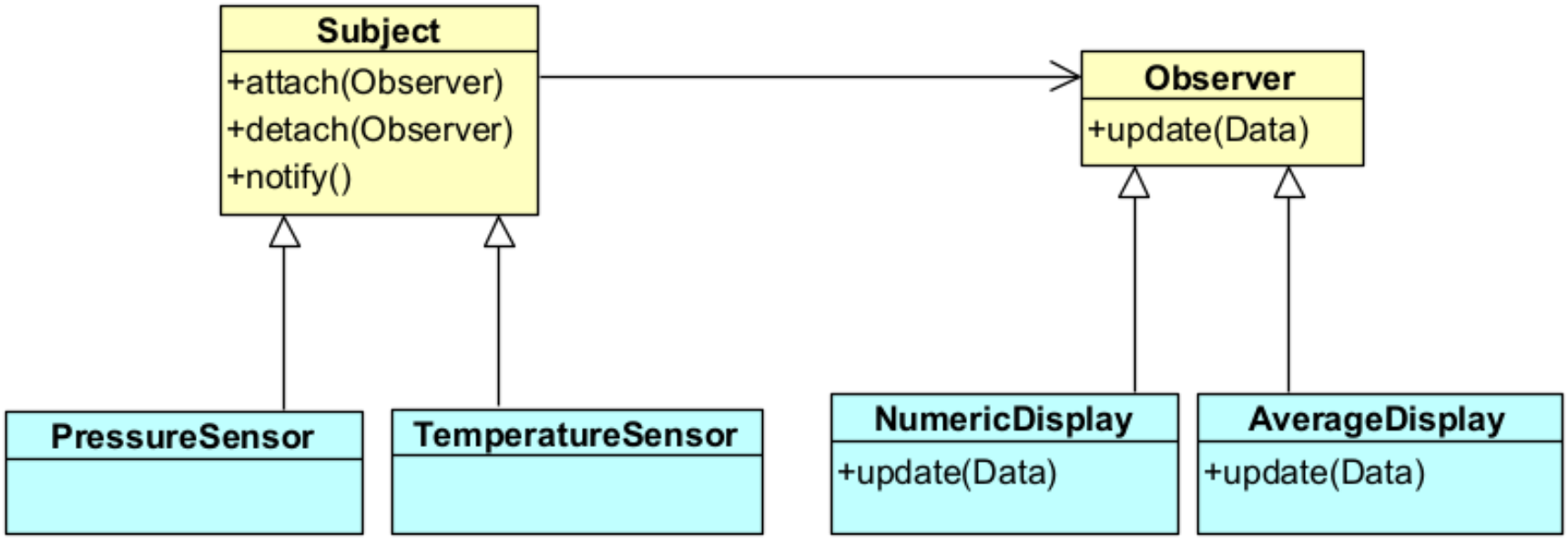
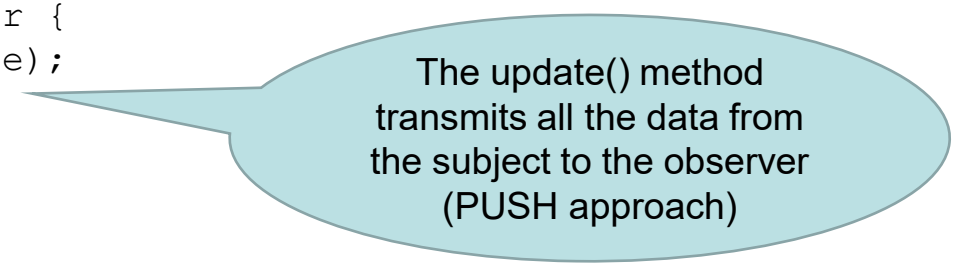


Diagram made with [Visual Paradigm]

```
public interface Observer {
    void update(int value);
}
```




```
public interface Subject {
    public void attach(Observer o);
    public void detach(Observer o);
    public void notifyObservers();
}
```

Concrete Subject

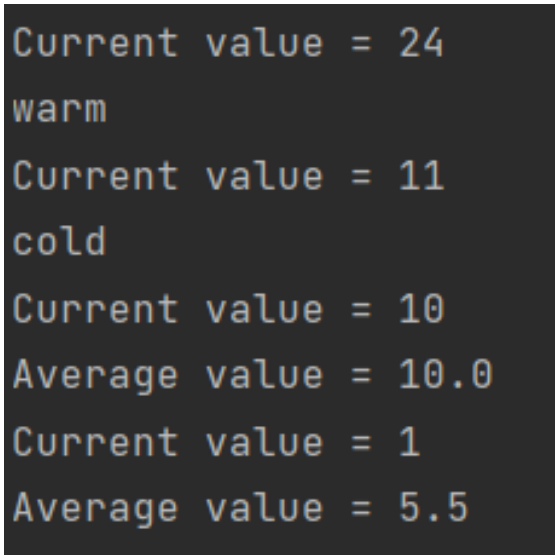
```
public class TemperatureSensor implements Subject{
    private ArrayList<Observer>observers;
    private int tempState;

    public TemperatureSensor(int initialTemp){
        tempState=initialTemp;
        observers=new ArrayList<Observer>();
    }


    public void setTemp(int newTemp){
        tempState=newTemp;
        notifyObservers();
    }

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

    public  void detach(Observer o) {
        observers.remove(o);
    }

    public void notifyObservers(){
        for (Observer o:observers)
            o.update(tempState);
    }

}
```

Concrete Observer

```
public class NumericDisplay implements Observer{

    private int value;

    public NumericDisplay(){
    }

    public void update(int value) {
        this.value=value;
        display();
    }

    public void display(){
        System.out.println("Current value = "+value);
    }
}
```

All the data is automatically pushed to the observer

```
public class MainDriver {
    public static void main(String[] args) {
        TemperatureSensor s=new TemperatureSensor(7);

        NumericDisplay o1=new NumericDisplay();
        TextDisplay o2 = new TextDisplay();
        AverageDisplay o3=new AverageDisplay();

        s.attach(o1);
        s.attach(o2);

        s.setTemp(24);
        s.setTemp(11);

        s.detach(o2);
        s.attach(o3);

        s.setTemp(10);
        s.setTemp(1);
    }
}
```



Observer – Consequences

- Advantages:
- Abstract coupling between subject and observers: the subject only knows that it has a list of abstract observers, defined by the simple Observer interface
- You can reuse the subject without reusing its observers
- You can reuse the observers together with an other subject
- You can attach many observers to a subject. The subject does not care how many observers there are (notify = broadcast type of communication)

- Disadvantages:
- The concrete observers must know the concrete subject they are observing
- If a concrete observer follows many concrete subject instances it must attach to each one of these
- Example of bad scenario: A display must receive data from any temperature sensor; there are many sensors; sensors may appear and disappear.