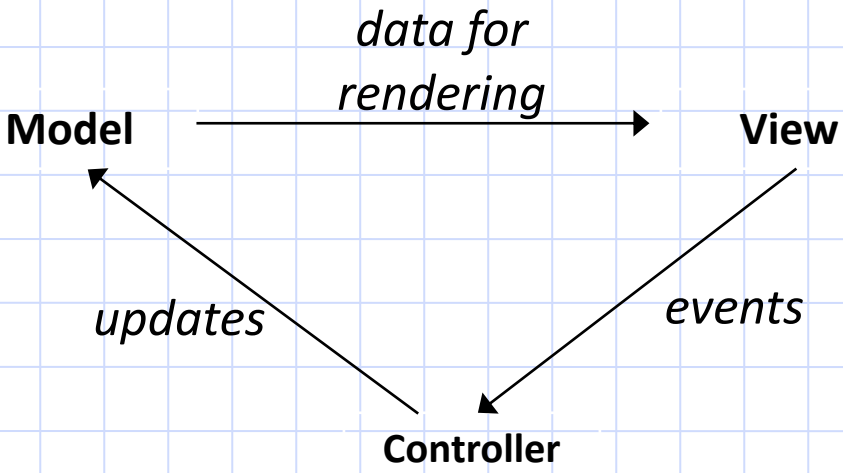
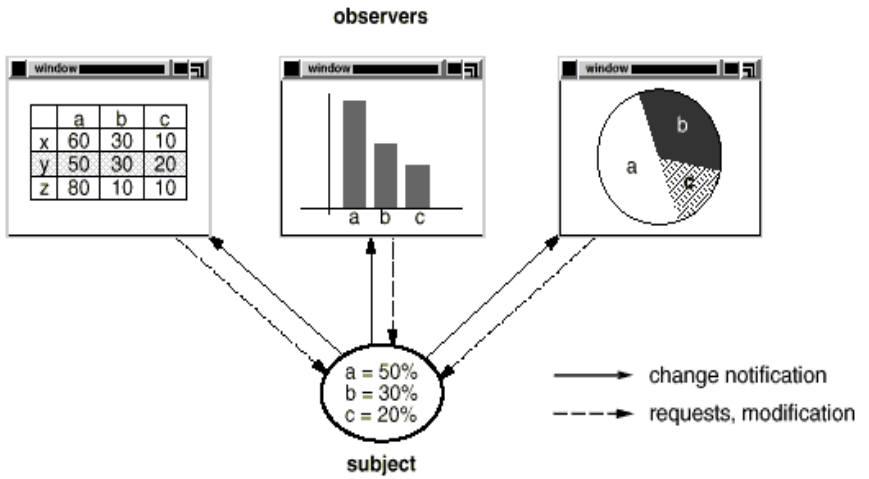
Design Pattern: Observer

objects whose state can be watched

Model-View-Controller

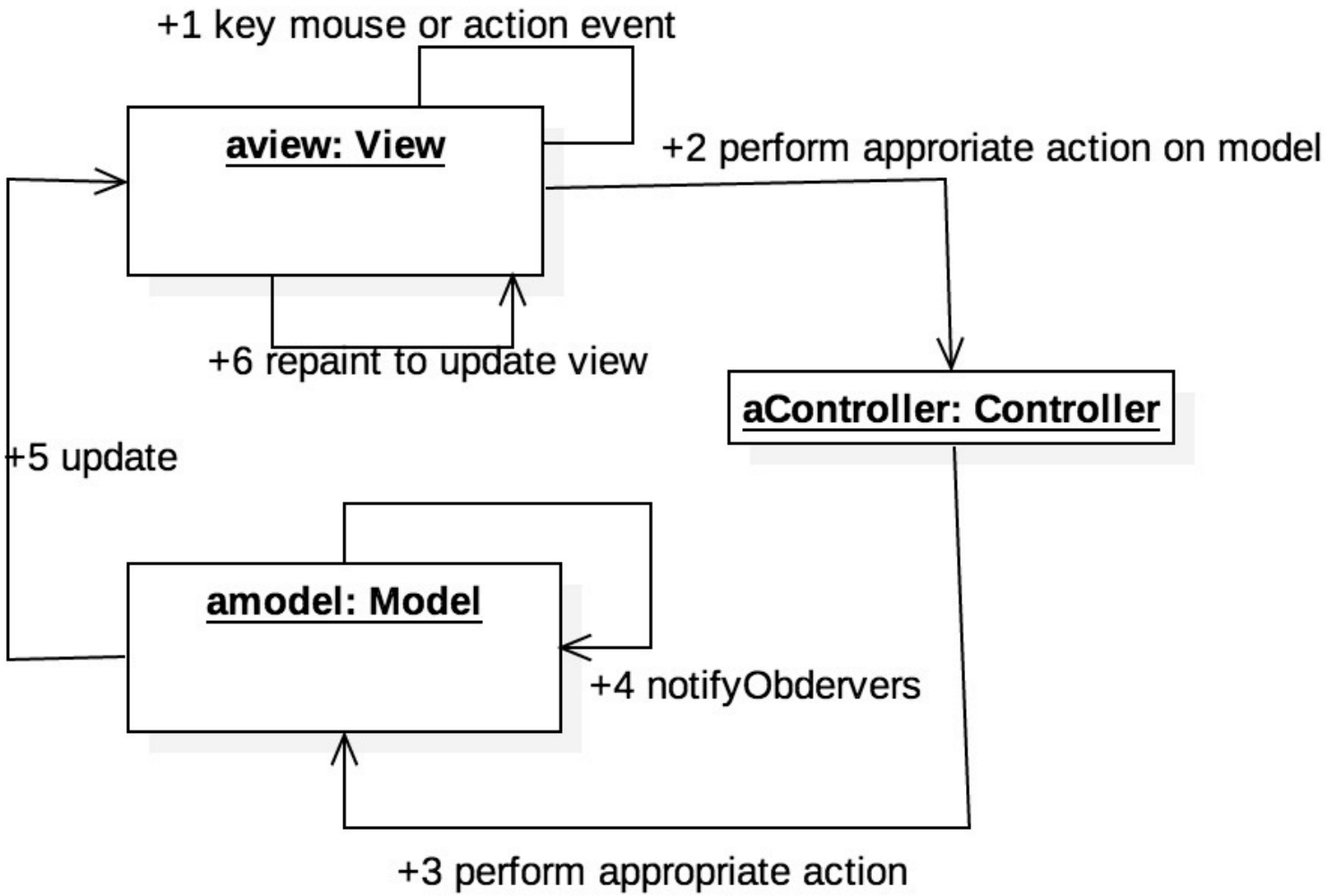
- **model-view-controller (MVC)**: common design paradigm for graphical systems





https://www.gofpatterns.com/design-patterns/module6/tradeoffs-implementing-observerPattern.php

Observer Pattern Object Diagram



MVC Pattren

- **model**: classes in your system that are related to the internal representation of the state of the system
- **view**: classes in your system that display the state of the model to the user
- **controller**: classes that connect model and view

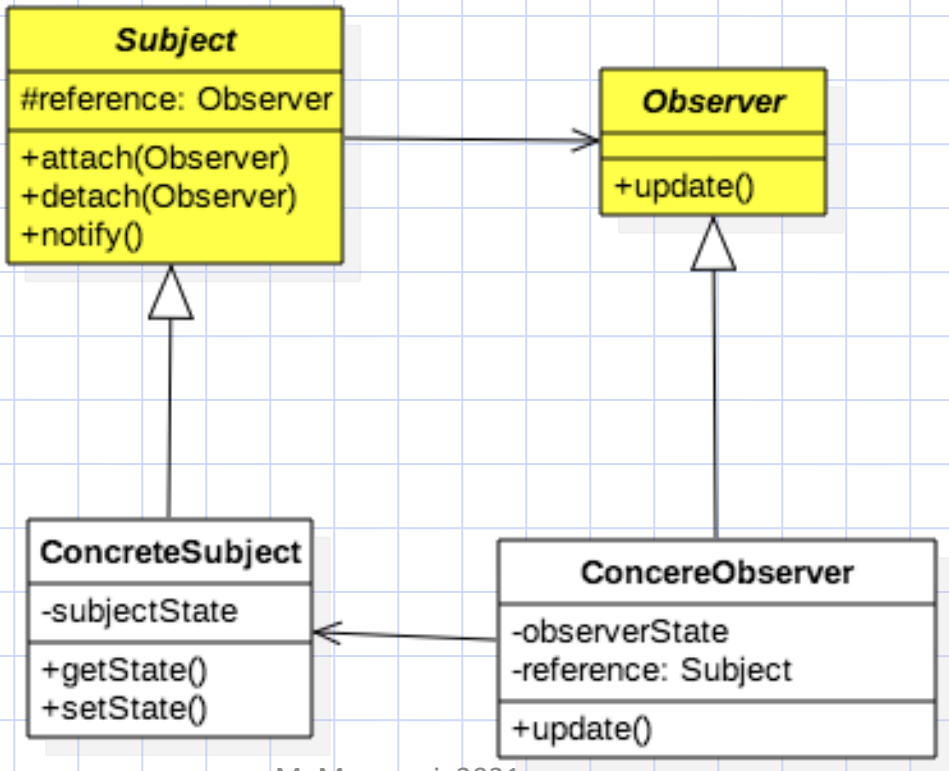
Observer pattern

- **observer**: an object that "watches" the state of another object and takes action when the state changes in some way

- **observable object**: an object that allows observers to examine it (often the observable object notifies the observers when it changes)

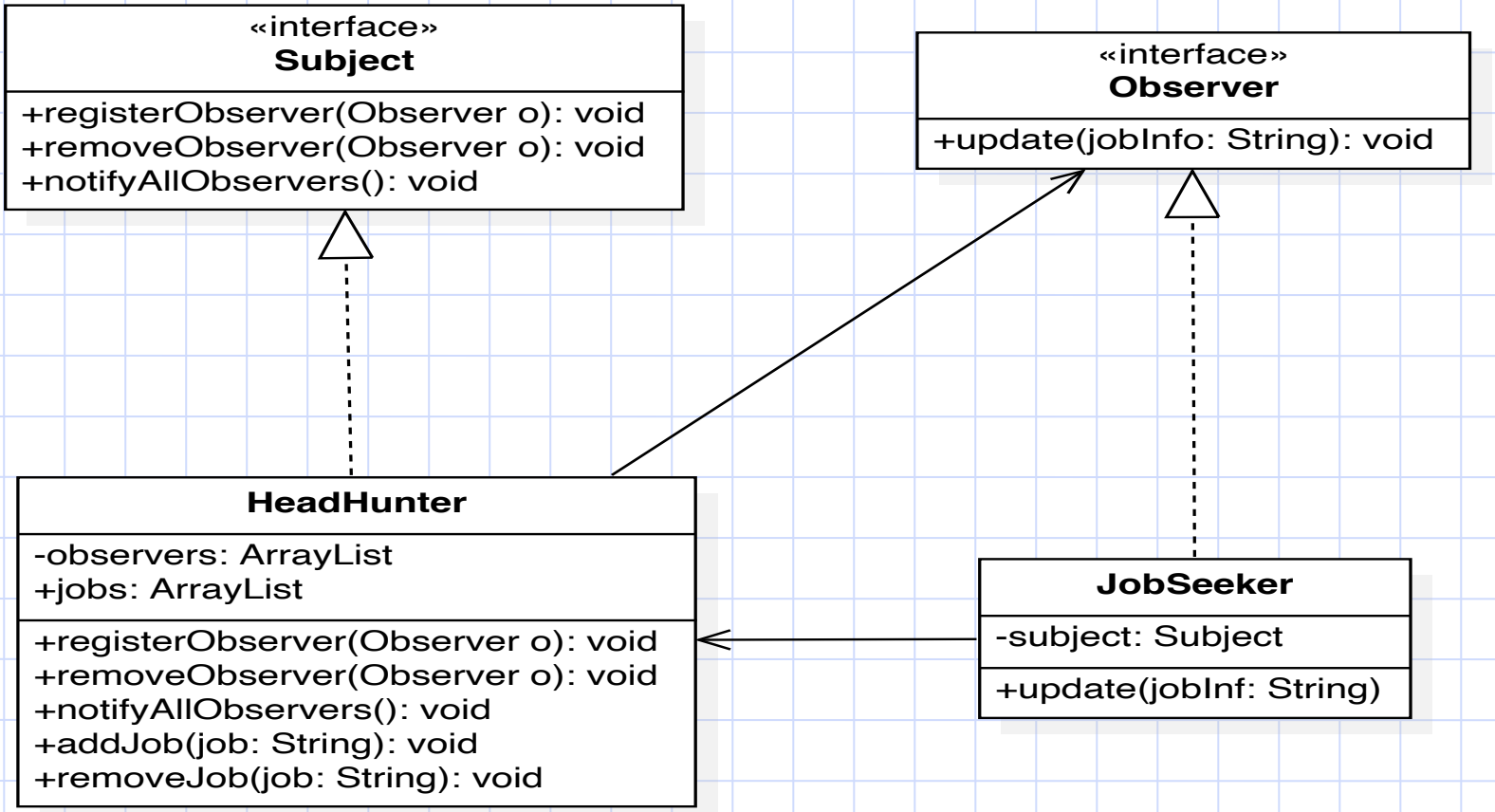
Observer Pattern Model

- The Observer pattern is one of the **behavioural** patterns.
- It's again used to form relationships between objects at runtime.
- This diagram shows Observer Pattern in C++ format using abstract classes instead of interfaces for



Other Applications of Observer Pattern

- Using observer pattern is not limited to GUI presentation; it can be used for any notification system. Here is an example:



Implementation Steps

1. Create an Observer Interface with a n update method.
2. Create either an interface or abstract class for Subject that contains methods to add or remove an observer object.
3. Create a class that implements Subject
4. Create one or more class that that implements Observer:

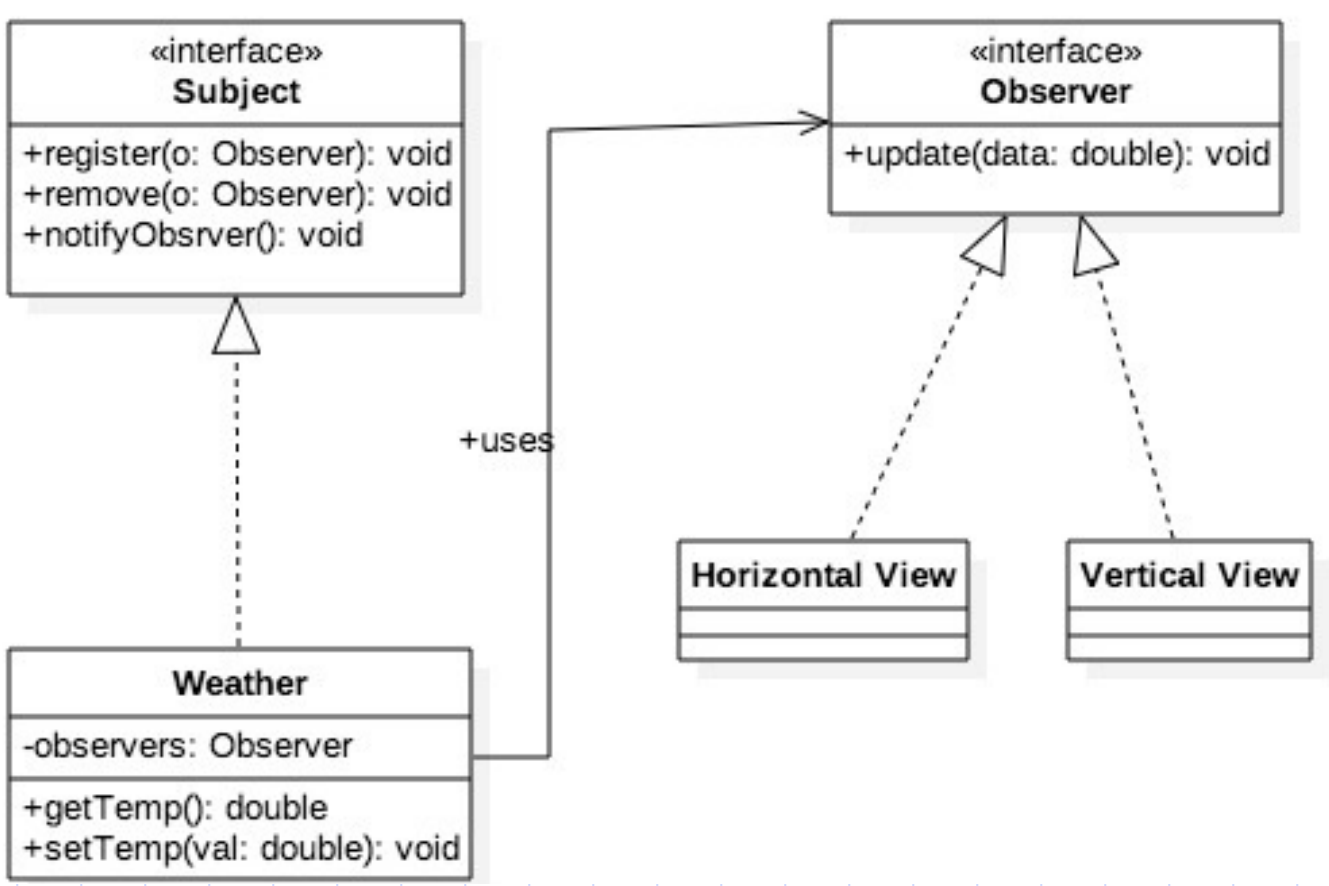
Observer Pattern Example

A Class Exercise

- Let's implement a small Java program that uses observer pattern to allow weather data to be observed in different format/different views.
- We need the following Java classes and interfaces:
  - An interface called **Observe**r with an abstract method called update
  - An interface called Subject with three abstract methods called: register, remove, and notifyObserver.
  - And, three classes
    - Class Weather that manages the data
    - Class HorizontalView that displays the data in horizontal format
    - Class VertaicalView that displays the data in vertical format

A Class Exercise

- Here is the UML model for this simple application.



Note: the Java implementation will be discussed during the lecture

Benefits of Observer Pattern

- Supports loose coupling between objects that interact with each other.
  - abstract coupling between subject and observer;
- Allows sending data to other objects without any change to the Subject or Observer classes. Observers need only to register with the Subject
- dynamic relationship between subject and observer:
  - Relationship can be established at runtime
  - Observers can be added/removed at anytime
  - Observers can be extended and reused individually
- Automatic Broadcast:
  - notification is broadcasted automatically to all interested objects that subscribed to it.
- Any Disadvantages?