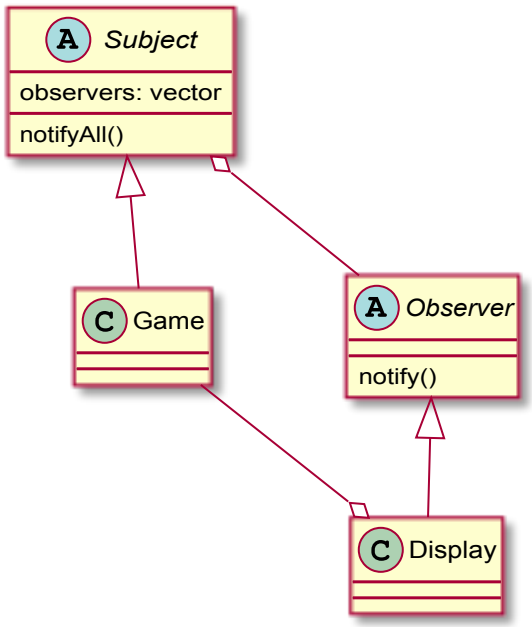
# Observer Pattern

Based on the publish/subscribe model of programming.

The purpose is to decrease coupling and allows abstraction away from the observer and the subject while allowing communication.

This is a “1 to many” dependency.




The idea behind this is to allow the Game to notify the display without needing to know how display is implemented.

- We give it a single access point called notifyAll() that calls notify() on all of the observers.
- The display may need information about the game, so it carries a pointer in case it needs access (preferably limiting the amount of info that is provided).

## Observer.h

```
1
2  #ifndef _OBSERVER_H_
3  #define _OBSERVER_H_
4
5  class Observer {
6   public:
7    virtual void notify() = 0;
8    virtual ~Observer();
9  };
10
11 #endif
```

## Observer.cc

```
#include "observer.h"

Observer::~Observer() {}
```

### Subject.h

```
#ifndef _SUBJECT_H_
#define _SUBJECT_H_
#include <vector>
#include "observer.h"

class Subject {
  std::vector<Observer*> observers;

 public:
  Subject();
  void attach(Observer *o);
  void detach(Observer *o);
  void notifyObservers();
  virtual ~Subject()=0;
};

#endif
```

### Subject.cc

```
#include "subject.h"

Subject::Subject() {}
Subject::~Subject() {}

void Subject::attach(Observer *o) {
  observers.emplace_back(o);
}

void Subject::detach(Observer *o) {
  for (auto it = observers.begin(); it != observers.end(); ++it) {
    if (*it == o) {
      observers.erase(it);
      break;
    }
  }
}

void Subject::notifyObservers() {
  for (auto ob : observers) ob->notify();
}

```