The Observer Pattern

Intermediate Application Development

Otago Polytechnic
Dunedin, New Zealand
Kaiako: Tom Clark

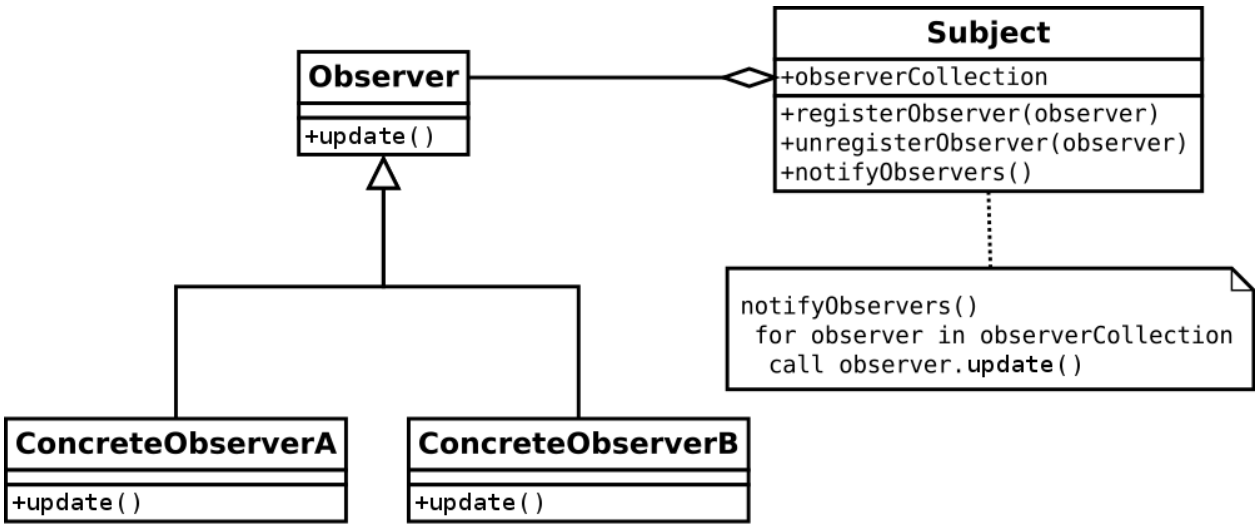
Introduction

"Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically."
(*GoF*)
In this pattern we have one object, the *Subject* that notifies other objects, called *Observers* whenever its state changes.

Examples

Suppose we have an object that represents some data, and multiple UI elements that need to update when that data changes. In this case the data object is the Subject and the UI elements are observers.

Structural Diagram



Subject Class

```
class Subject:

    def __init__(self):
        self._observers = set()

    def register(self, observer):
        self._observers.add(observer)

    def unregister(self, observer):
        self._observers.discard(observer)

    def notify(self):
        for o in self._observers:
            o.update(self)
```

Observer Class

```
class Observer:
    def update(subject):
        pass
```

Programming Activity

1. Pull the course materials repo.
2. Create a new branch, `13-practical` in your practicals repo.
3. Add a subdirectory, `13-practical` and copy `11-practical.ipynb` from the class materials into it.
4. Open a shell, cd to this directory, and run `jupyter notebook` to open the notebook. Complete the first two questions.
5. We will discuss results in 20ish minutes.

A Pythonic Refinement

We can remove the requirement that Observers implement a method named exactly "`update()`".

```
class Subject:
    def __init__(self):
        self._observers = {} # dictionary

    def register(self, observer, update_method):
        self._observers[observer] = update_method

    def unregister(self, observer):
        try:
            del(self_observers[observer})
        except KeyError:
            pass

    def notify(self):
        for update in self._observers.values():
            update{self)
```

Publish-Subscribe Variation

It's also possible to take the decoupling a step further, with what is sometimes called the Publish-Subscribe pattern.