Glossary of Terms
&
Observer Pattern Supplemental

DRY
Don't Repeat Yourself

- Repetition = redundancy.
- Extra work keeping everything in sync.
- You'll probably miss one of the repetitions.
- Source of subtle hard-to-find bugs.

Cohesion

- The degree to which a component addresses a single, well-defined issue.
- Component can be:
  - A method
  - A class
  - A subsystem of classes
- We want systems with ***high cohesion***.

Separation of Concerns

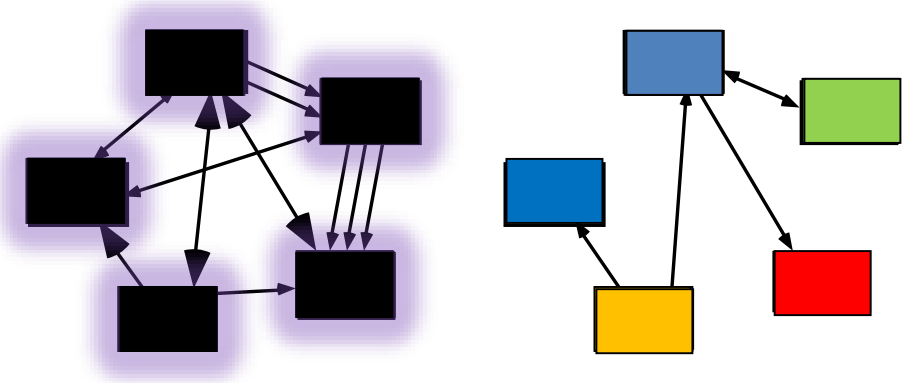
- Each component addresses a single, separate design concern.
- A concern is a well-define aspect of the problem being solved.
- SOC tends towards highly-cohesive components.
- *Do One Thing Well.*

Class Responsibilities

- What do the objects in a class ***know***?
  - Internal state
  - Instance variables
- What can the objects in a class ***do***?
  - Visible behavior.
  - Public methods
- Do nothing significant = data class - bad karma.
  - A class must *pull its weight* in the overall design.
- Know nothing significant = algorithm class - may be ok.

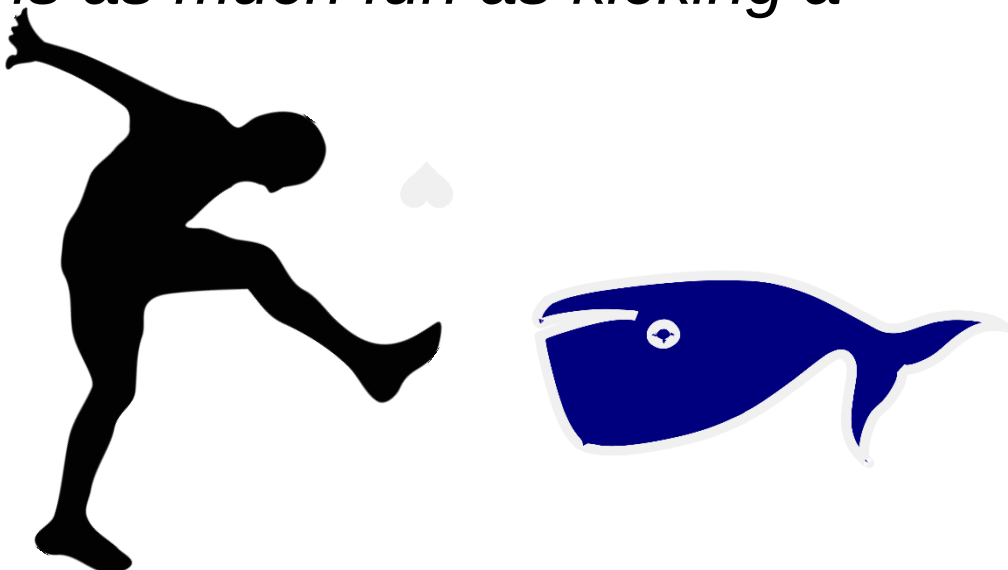
Coupling

- Coupling is the manner and degree of interdependence between software modules.
- Not just a *count* of the number of dependent classes.
- Also need the *degree* of dependence:
  - How easy is it to update this component?
  - How easy is it to replace a dependent component?
- In general, the less the interdependence - the ***lower the coupling*** - the better.



Technical Debt (Code Debt)

- Work that needs to be done before a particular job can be considered properly complete.
- Debt not repaid **keeps on accumulating interest**, making it hard to implement changes later on.
- Debt is not necessarily bad as long **as you control it**.
- Uncontrolled financial debt leads to bankruptcy.
- Uncontrolled technical debt leads to systems that can't adapt (and may need to be scrapped.
- *Working under heavy code debt is as much fun as kicking a dead whale along the beach.*



Refactoring

- Restructuring existing code without changing its external behavior.
- To improve nonfunctional attributes such as readability, maintainability, and design integrity.
- One technique to help keep technical debt under control.
- Little at a time refactoring vs. "big bang" refactoring.
- Refactor a bit, enhance a bit.

How did we refactor WS?

Design Pattern

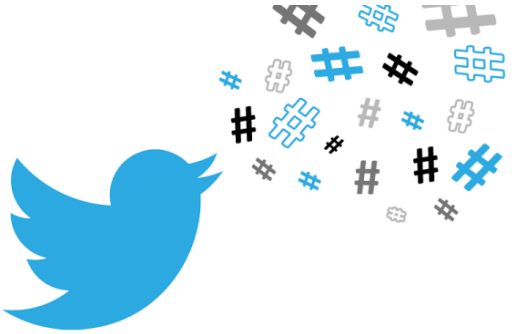
- The re-usable form of a general solution to a common design problem.
- Christopher Alexander in *The Timeless Way of Building*:

> Each ***pattern*** describes a ***problem*** that occurs ***over and over*** again in our environment, and then describes the ***core of the solution*** to that problem, in such a way that you ***can use this solution a million times over, without ever doing it the same way twice***.

- Software design patterns are the essence of empirically proven solutions to common design problems.

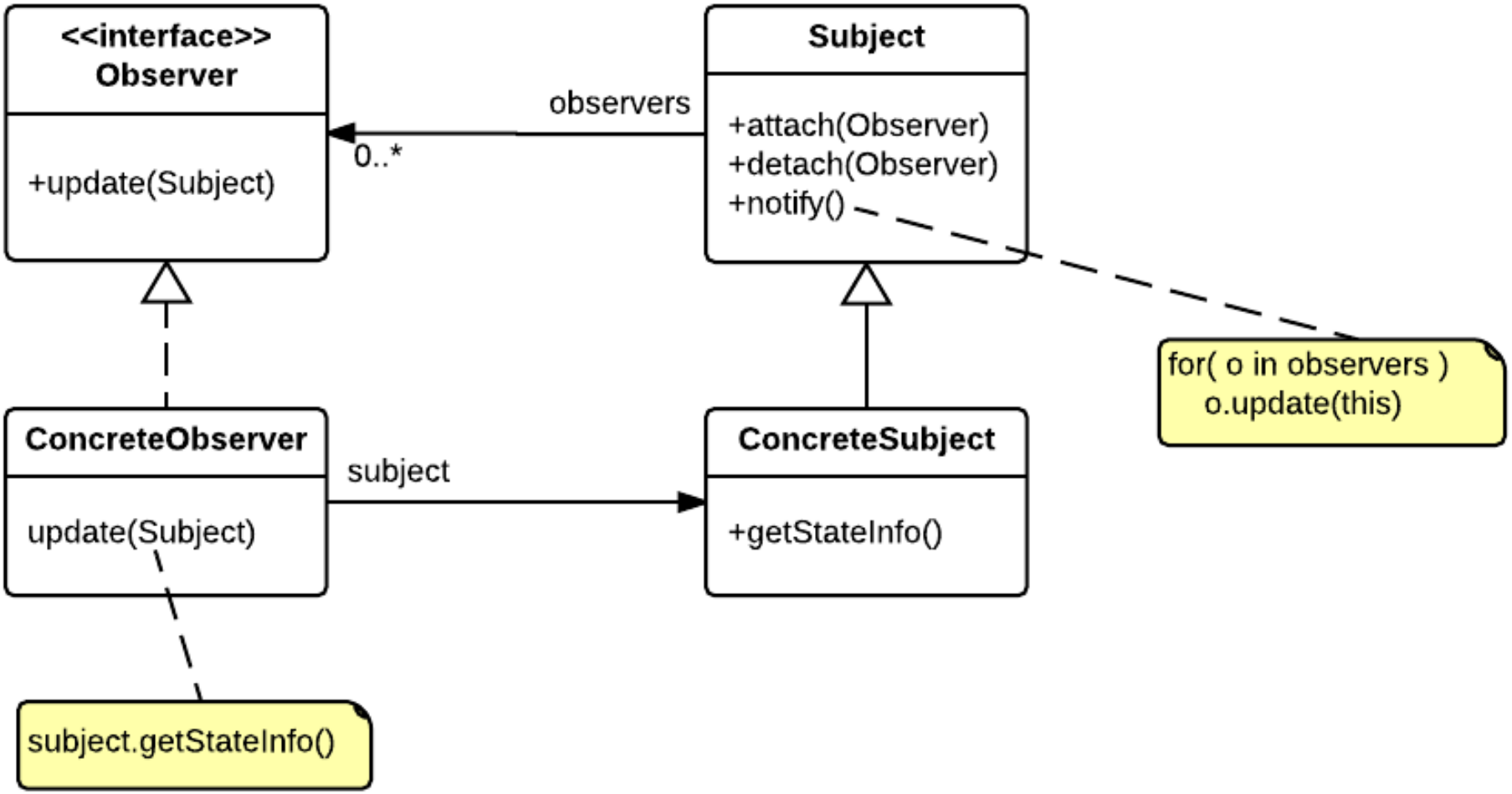
Observer Pattern

- Define a one-to-many dependency between objects so that when the *one object (A)* changes state, *all its dependents (B) are notified and updated* automatically.
- Don't want to maintain the consistency by tightly coupling (A) to the dependents (B).
- Those interested, **Observers**, *register* interest in changes to the **Subject** object.
- When the **Subject** changes, it *notifies* all registered **Observers** by calling a generic *update()* method.
- Particular names - *notify*, *update*, *Observer*, *Subject,* etc. - are not fixed; they just indicate roles and responsibilities!

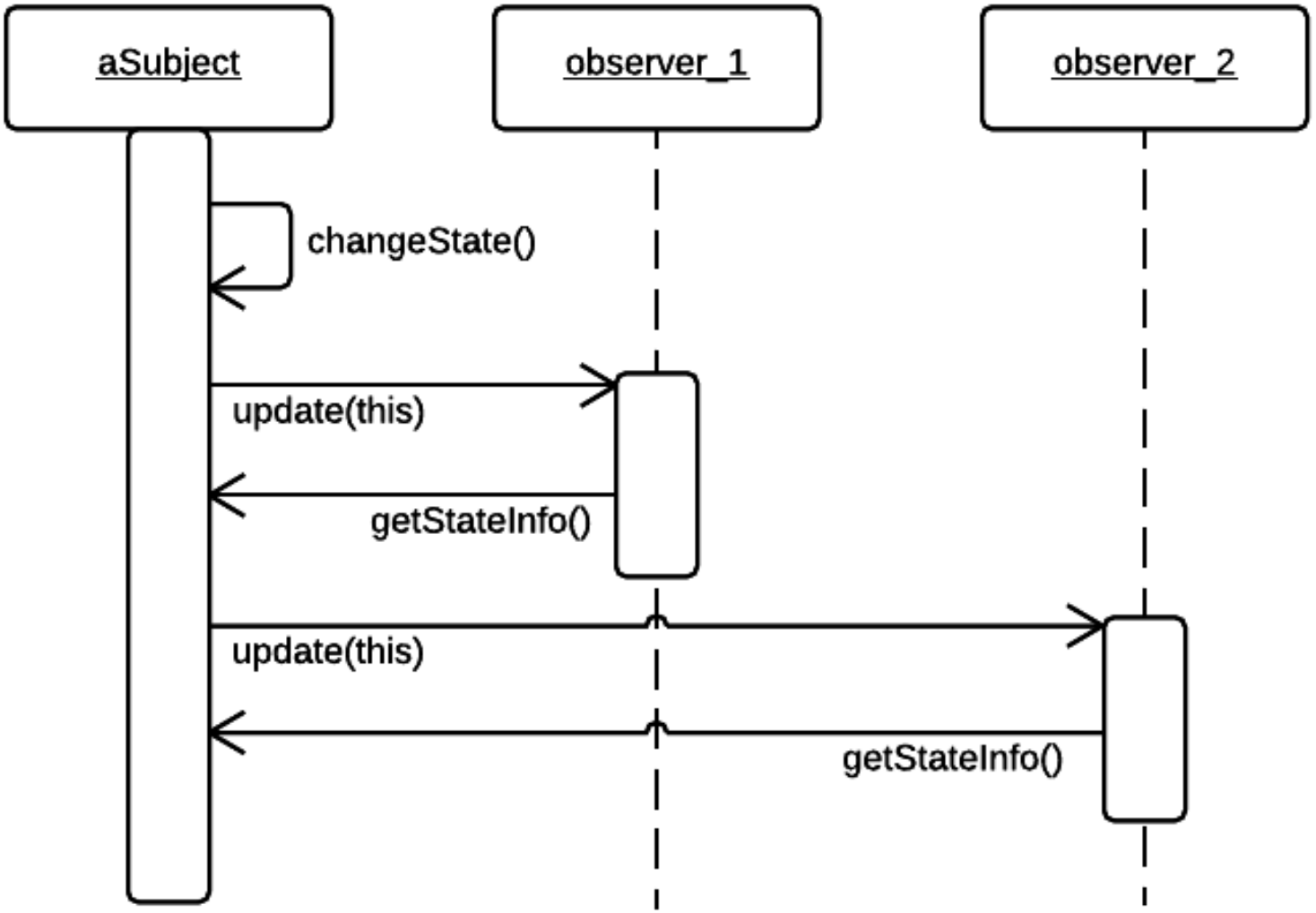


OR

Observer Class Diagram



Observer Sequence Diagram



Observer in Java

- Interface **`java.util.Observer`** [implementable Observer]

```
public interface Observer {
        public void update(Observable obs, Object
obj) ;
}
```

- Class **`java.util.Observable`** [extendable Subject]

```
public class Observable {
        public void addObserver(Observer obs) ;
        public void deleteObserver(Observer obs) ;
        public void setChanged() ;
        public void clearChanged() ;
        public boolean hasChanged() ;
        public void notifyObservers() ;
        public void notifyObservers(Object arg) ;
}
```