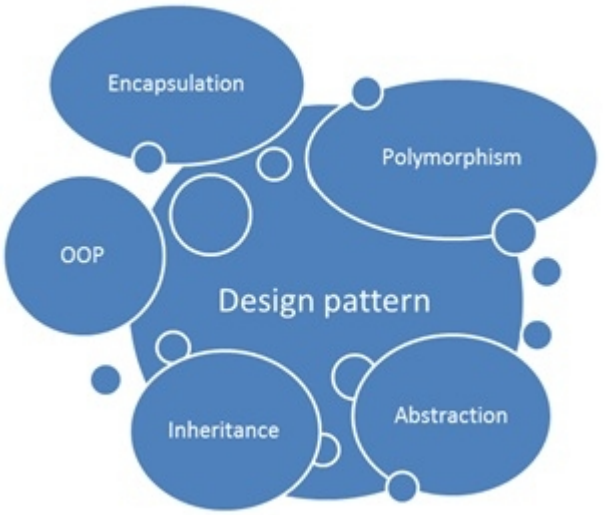
Design Pattern



Sayem Mohammad Siam

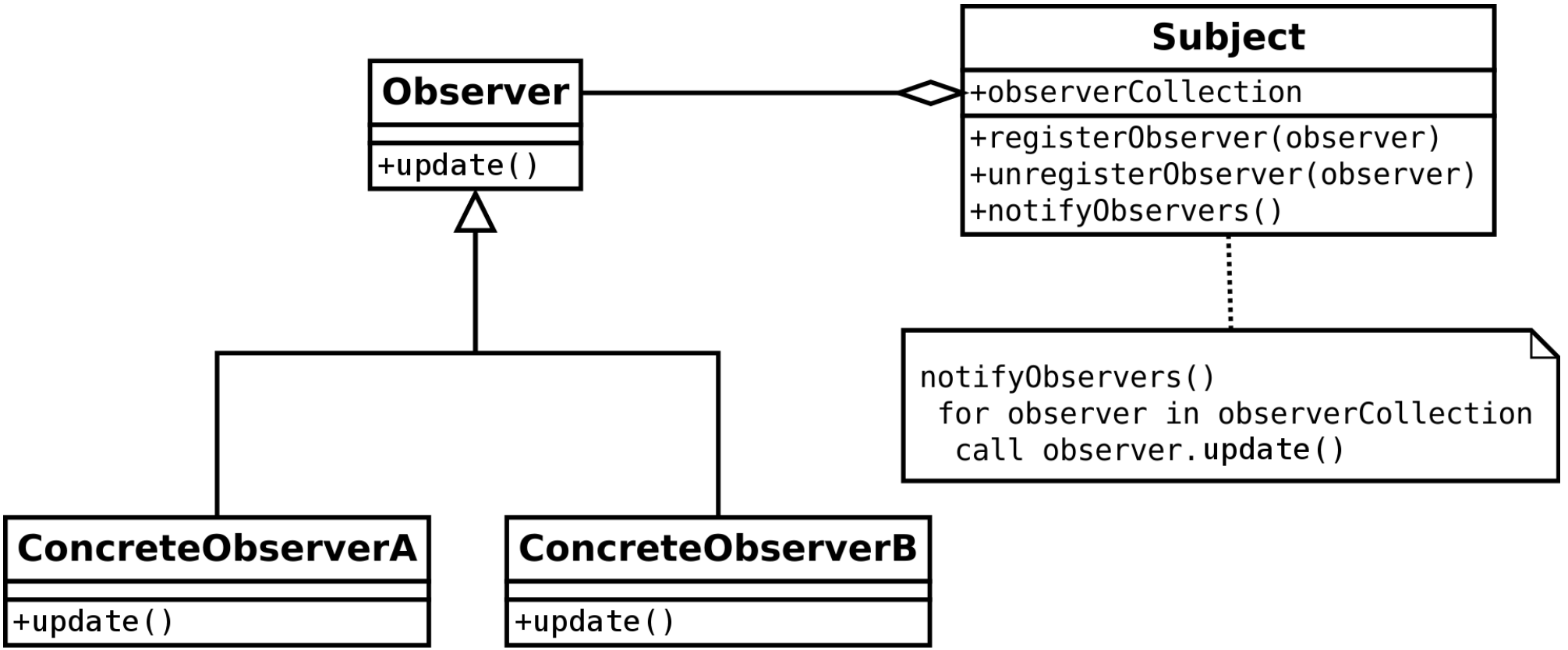
Why design pattern?

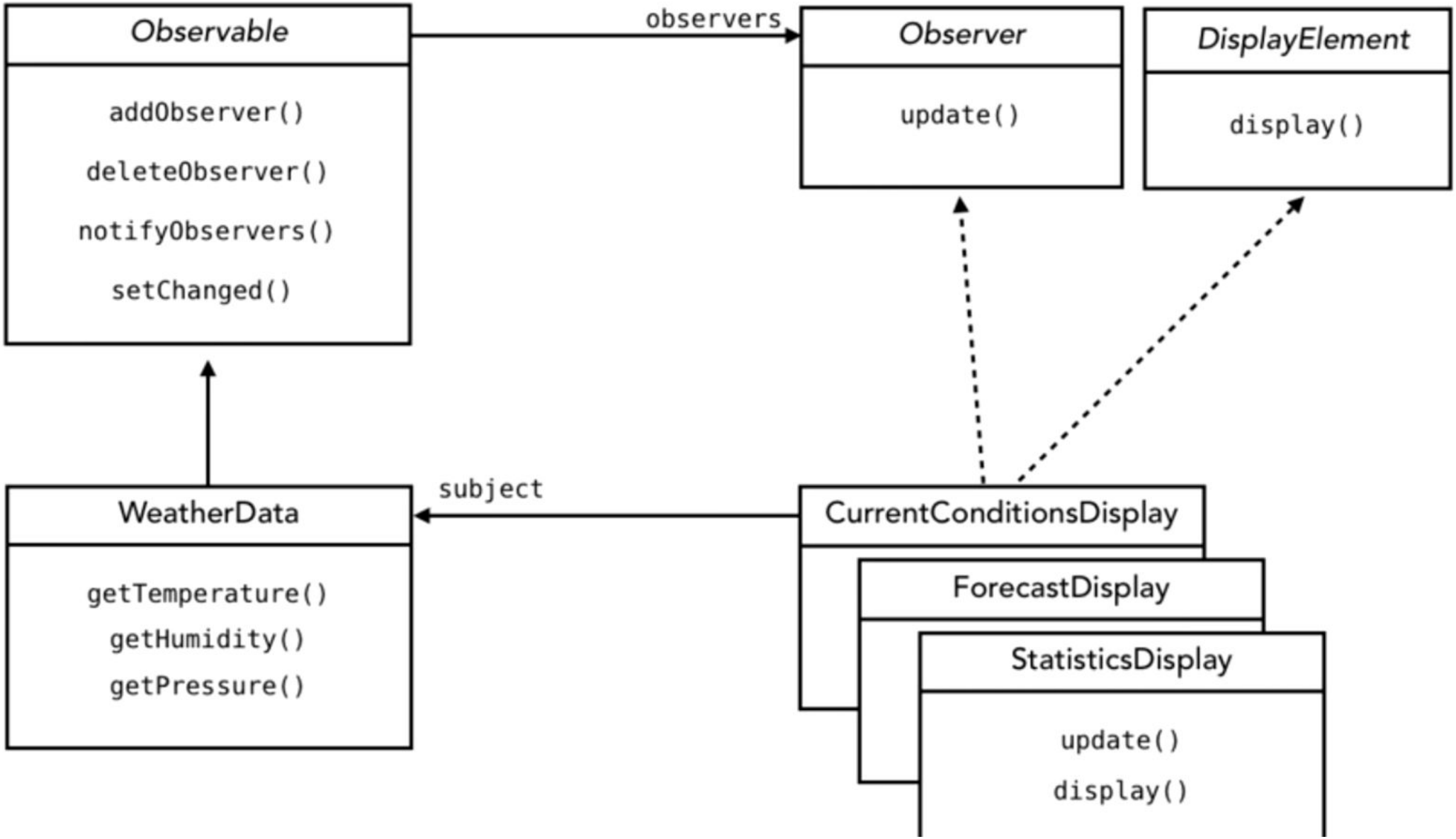
We can use those also!

Observer Pattern

- It is mainly used to implement distributed event handling systems, in "event driven" software.
- A one-to-many dependency between objects should be defined without making the objects tightly coupled.
- It should be ensured that when one object changes state an open-ended number of dependent objects are updated automatically.
- It should be possible that one object can notify an open-ended number of other objects.

Observer Pattern




Observable
addObserver()
deleteObserver()
notifyObservers()
setChanged()
observers
Observer
update()
DisplayElement
display()
WeatherData
getTemperature()
getHumidity()
getPressure()
subject
CurrentConditionsDisplay
ForecastDisplay
StatisticsDisplay
update()
display()

Observer Pattern

- The Observer design pattern is one of the twenty-three well-known "Gang of Four" design patterns
- Describe how to solve recurring design problems to design flexible and reusable object-oriented software
- Objects that are easier to implement, change, test, and reuse.

Design Principles

1. DRY (Don't repeat yourself)

- Don't write duplicate code, instead use Abstraction to abstract common things in one place
- If you have a block of code in more than two places consider making it a separate method
- If you use a hard-coded value more than one time make them public final constant

DRY

- "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system".

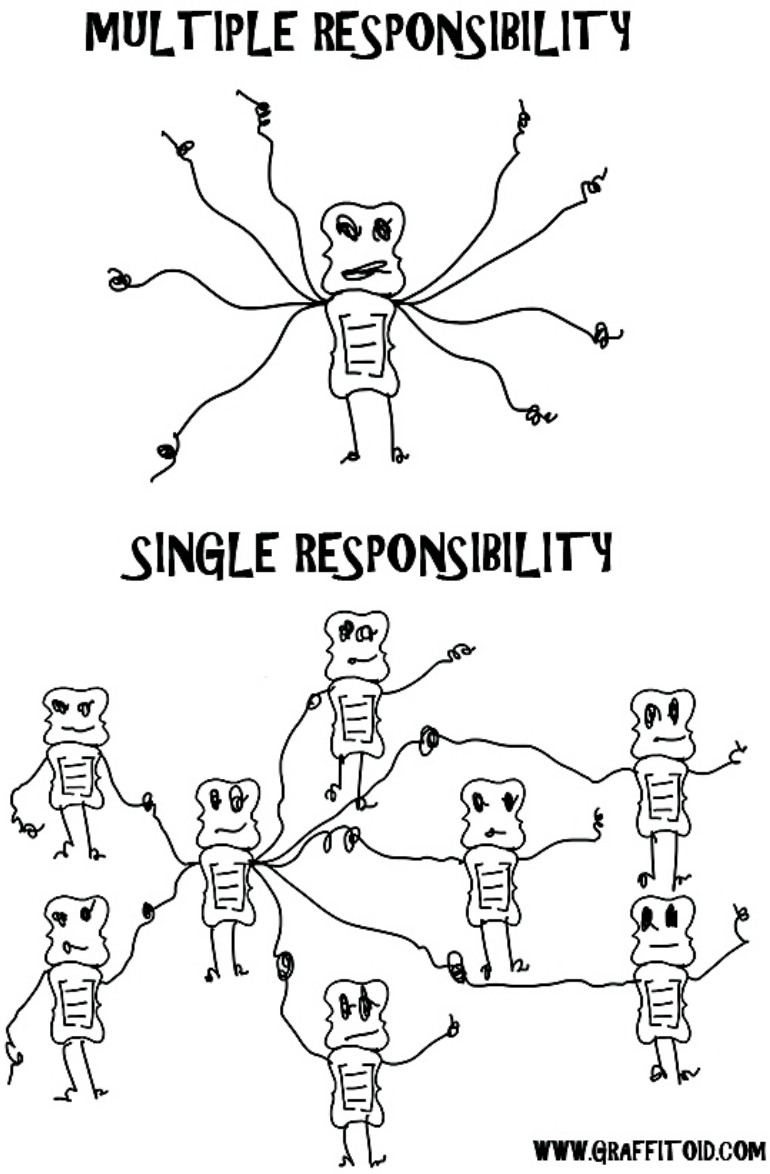
2. Encapsulate What Changes

- The benefit of this OOP Design principle is that It's easy to test and maintain proper encapsulated code

3. Open Closed Design Principle

- Classes, methods or functions should be Open for extension (new functionality) and Closed for modification.

4. Single Responsibility Principle (SRP)



5. Dependency Injection or Inversion principle

- The beauty of this design principle is that any class which is injected by DI framework is easy to test with the mock object
- Because object creation code is centralized in the framework and client code is not littered with that.

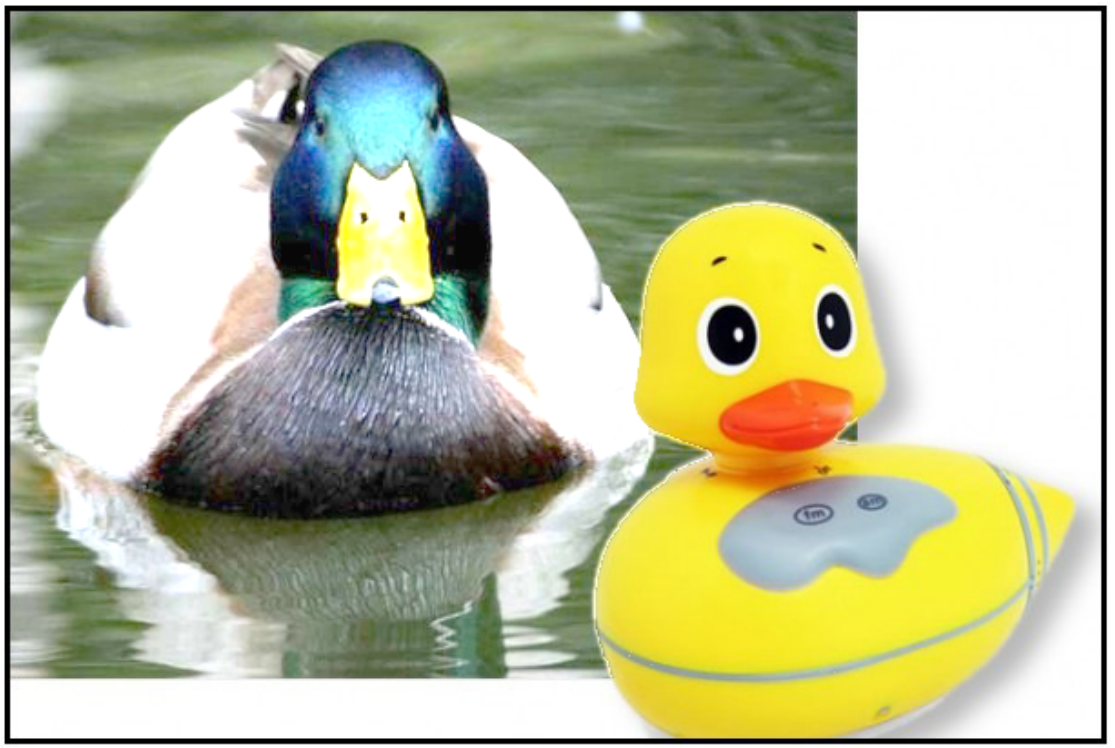
6. Favor Composition over Inheritance

- Always favor composition over inheritance, if possible.
- Flexibility to replace with better implementation any time.

7. Liskov Substitution Principle (LSP)

- According to the Liskov Substitution Principle, "*Subtypes must be substitutable for supertype i.e. methods or functions which uses superclass type must be able to work with the object of subclass without any issue*".
- LSP is closely related to Single responsibility principle and Interface Segregation Principle. If a class has more functionality than subclass might not support some of the functionality and does violate LSP.
- subclass and superclass are related to each other by Inheritance which provides IS-A property

Liskov Substitution Principle



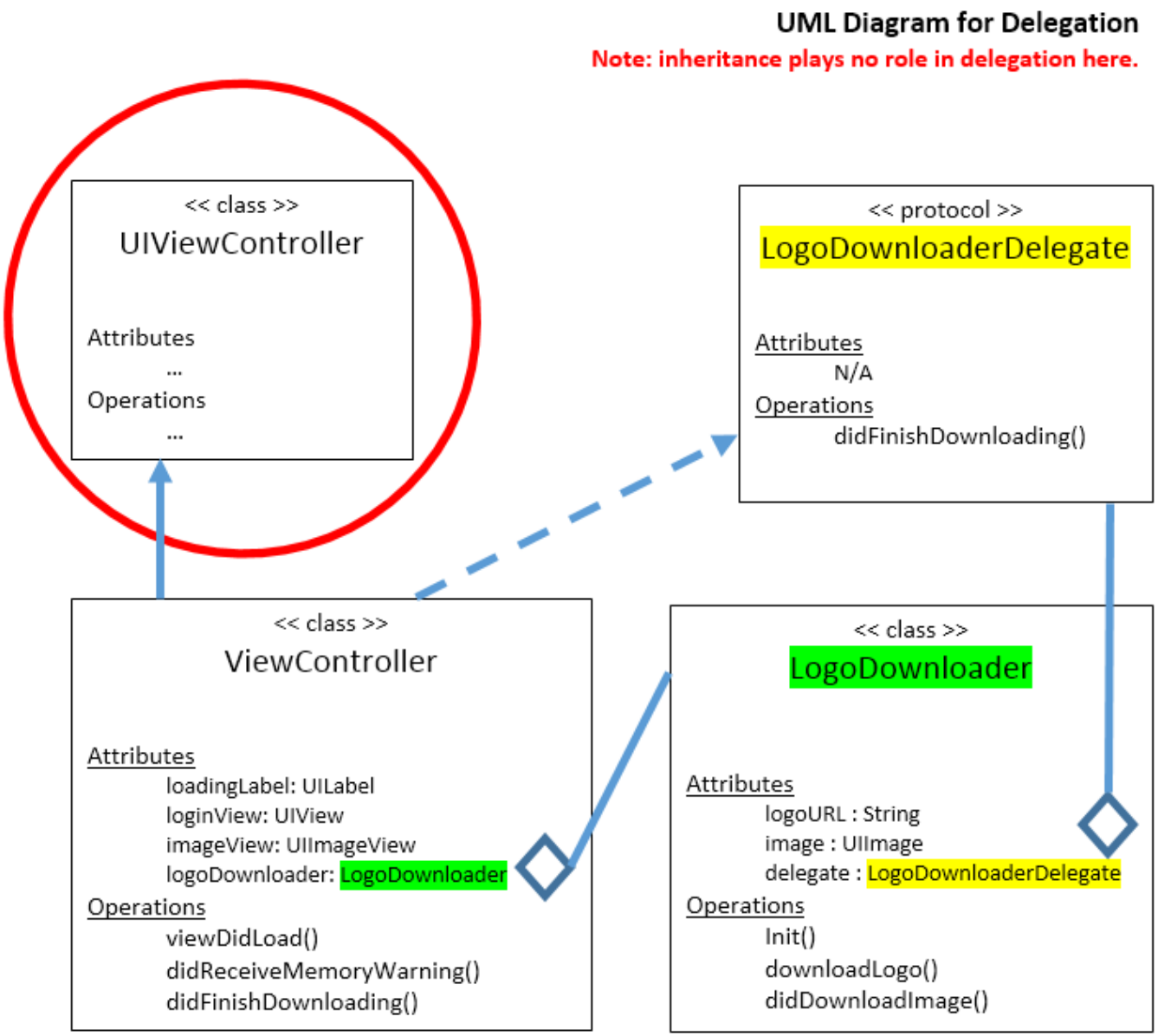
LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

8. Interface Segregation Principle (ISP)

- a client should not implement an interface if it doesn't use that.

9. Delegation principles



Grady Booch's Defination of delegation

- Delegation is a way to make composition as powerful for reuse as inheritance.
- In delegation, two objects are involved in handling a request: a receiving object delegates operations to its delegate.
- This is analogous to subclasses deferring requests to parent classes. But with inheritance, an inherited operation can always refer to the receiving object through the this member variable in C++ and self in Smalltalk.
- To achieve the same effect with delegation, the receiver passes itself to the delegate to let the delegated operation refer to the receiver.

Grady Booch (born February 27, 1955) is an American software engineer, best known for developing the Unified Modeling Language (UML) with Ivar Jacobson and James Rumbaugh. He is recognized internationally for his innovative work in software architecture, software engineering, and collaborative development environments.

9. Programming for Interface not implementation

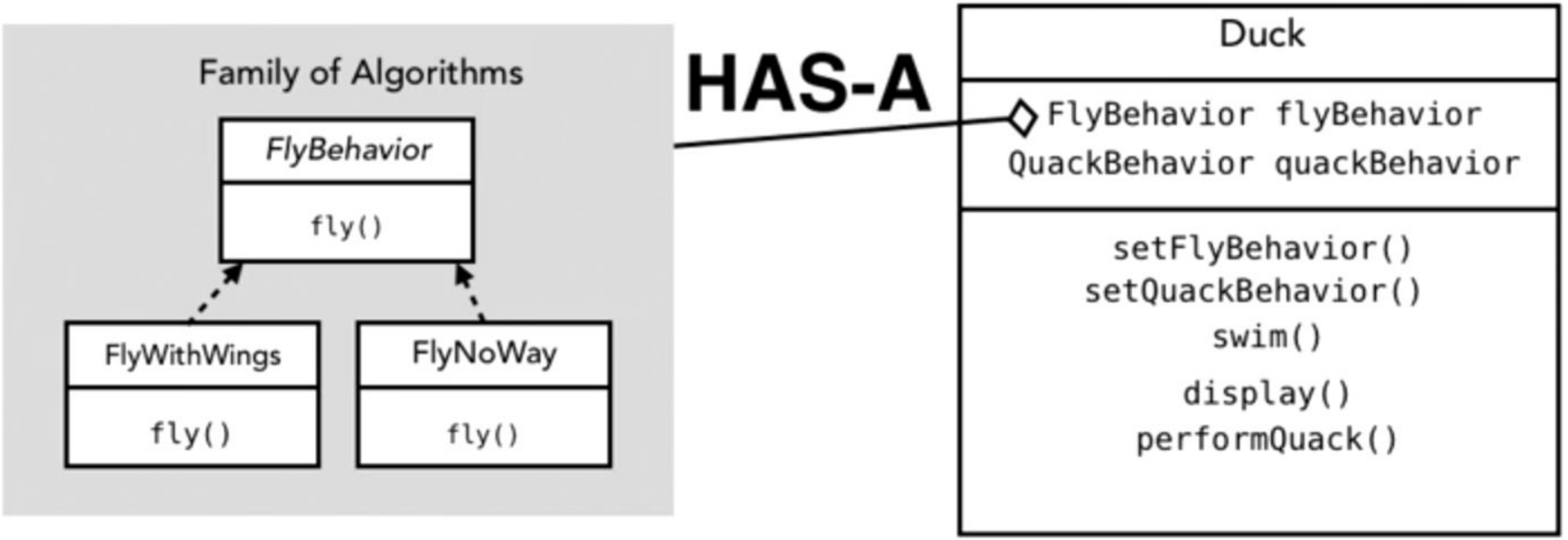
- Client program is not worried about implementation and the interface signature determines what all operations can be done. This can be used to change the behavior of a program at run-time.
- It also helps you to write far better programs from the maintenance point of view.

SOLID

- SRP – Single Responsibility Principle
- OCP – Open/Closed Principle
- LSP – Liskov Substitution Principle
- ISP – Interface Segregation Principle
- DIP – Dependency Inversion Principle

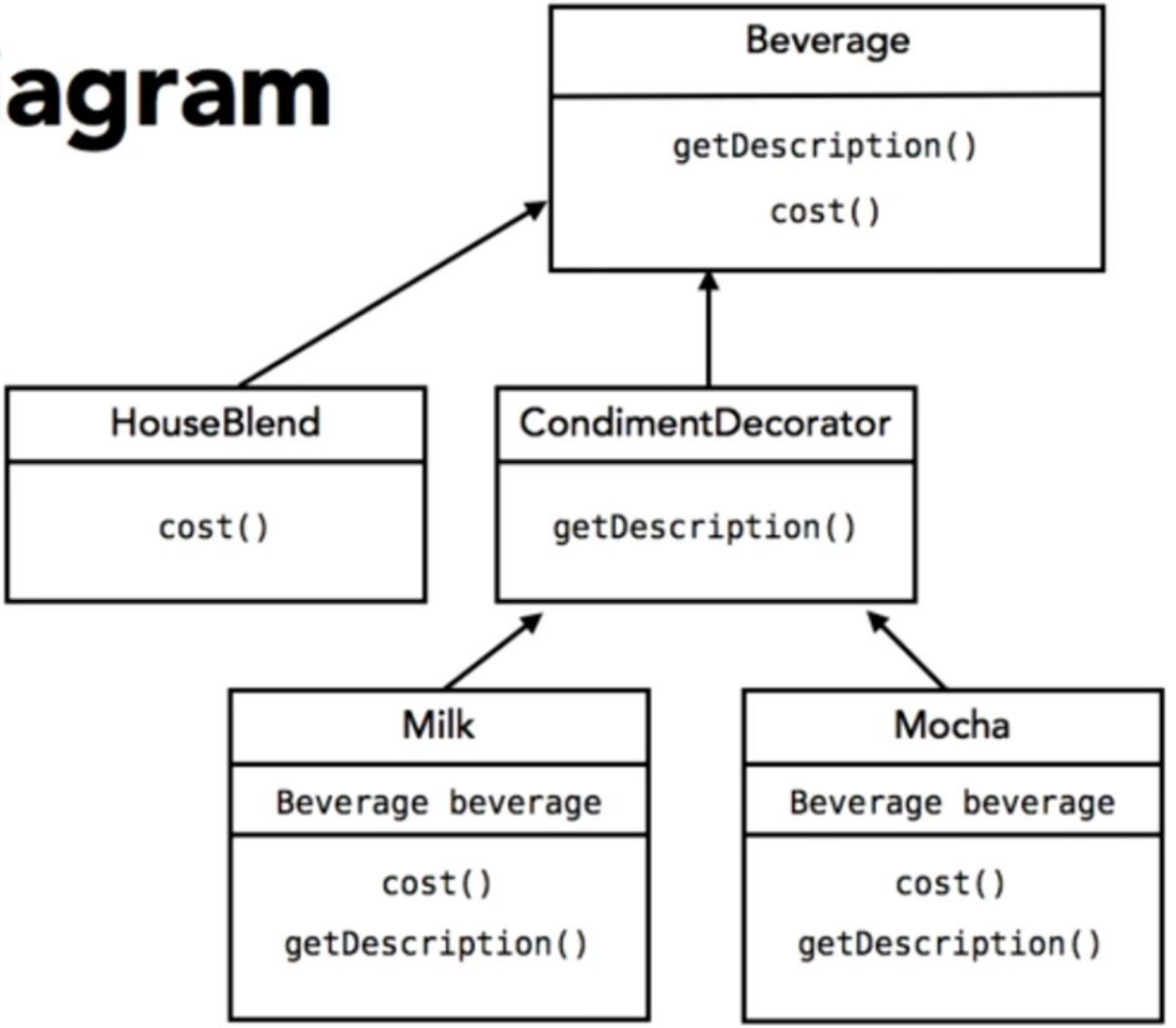
Some more design pattern!

Strategy Pattern



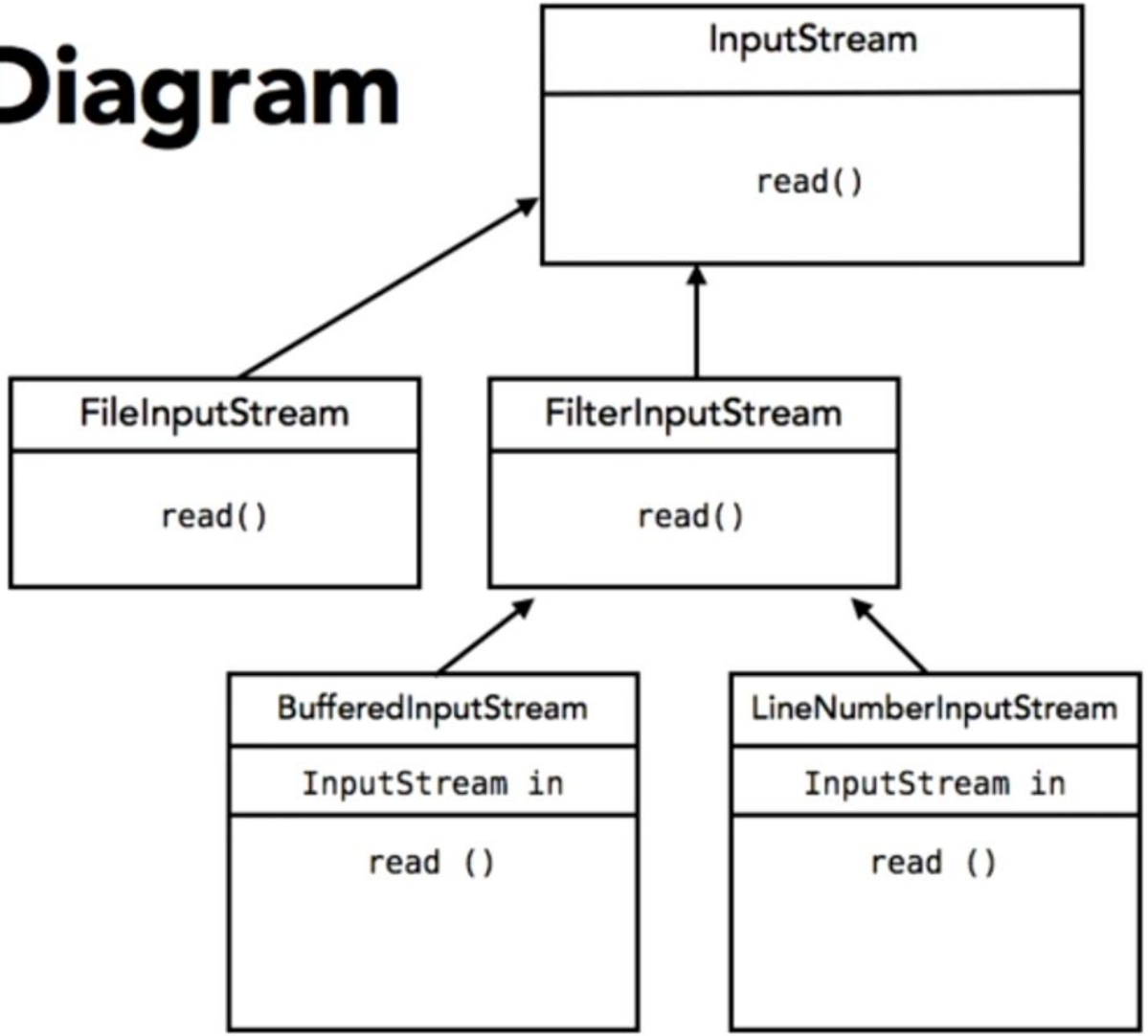
Decorator Pattern

Coffee Class Diagram

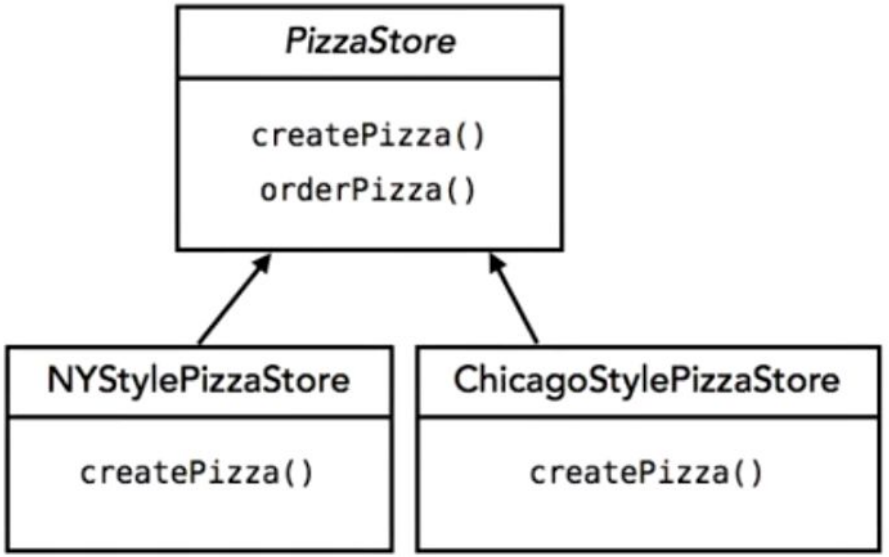


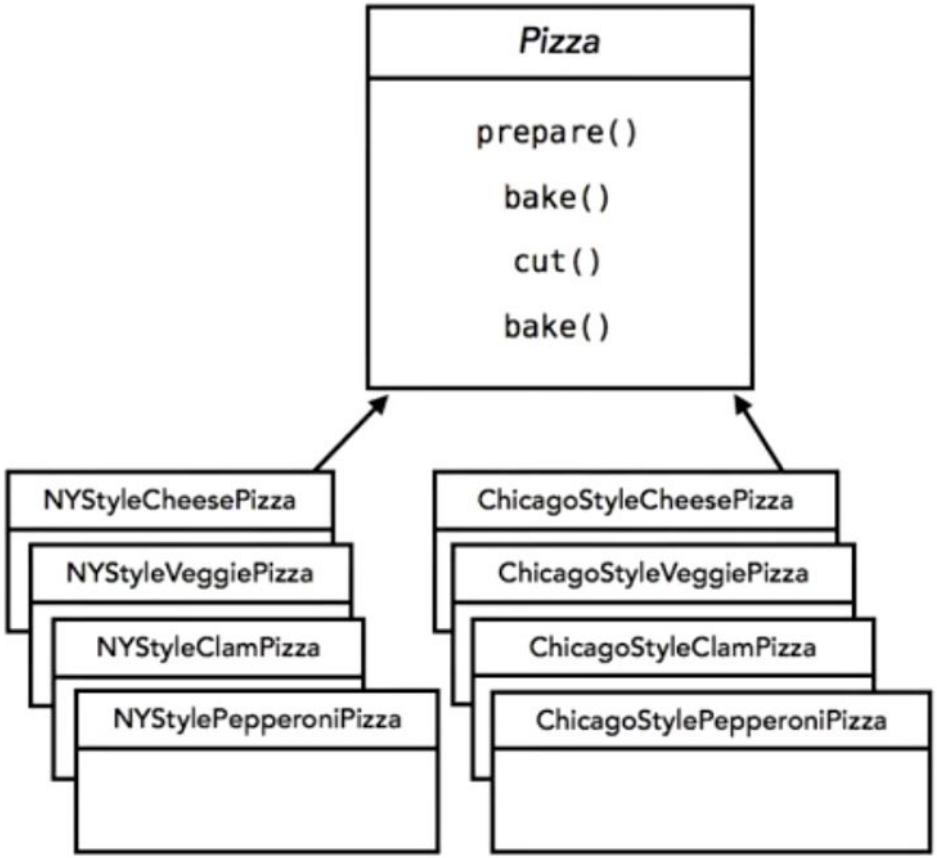
Decorator Pattern

java.io.* Class Diagram

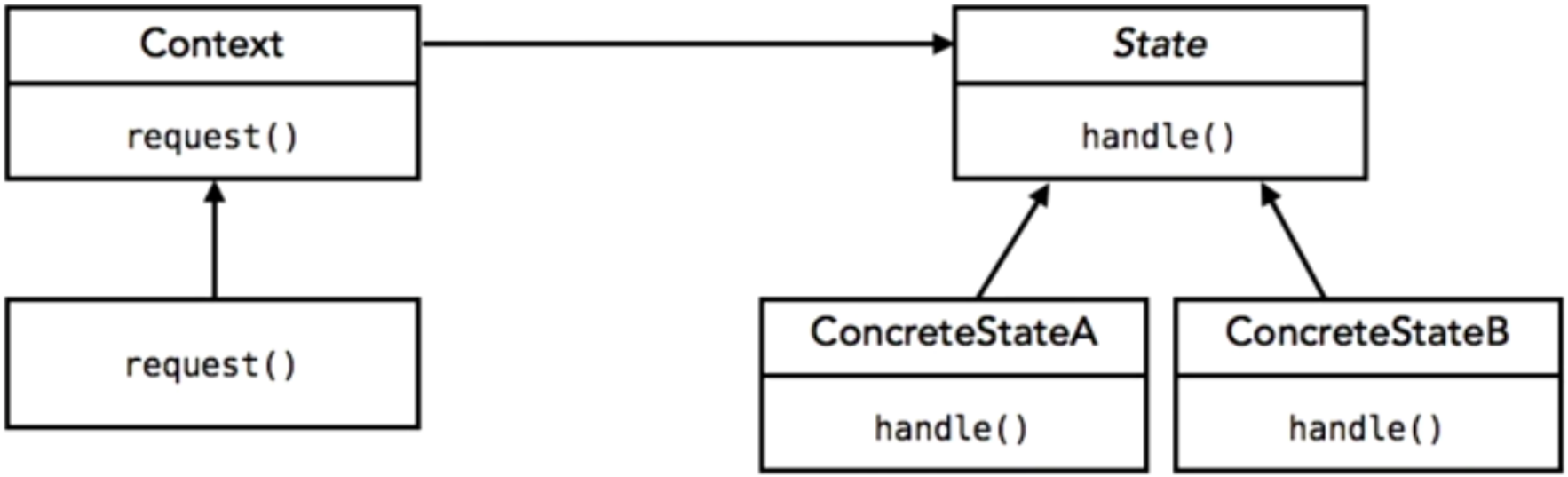


Factory Pattern

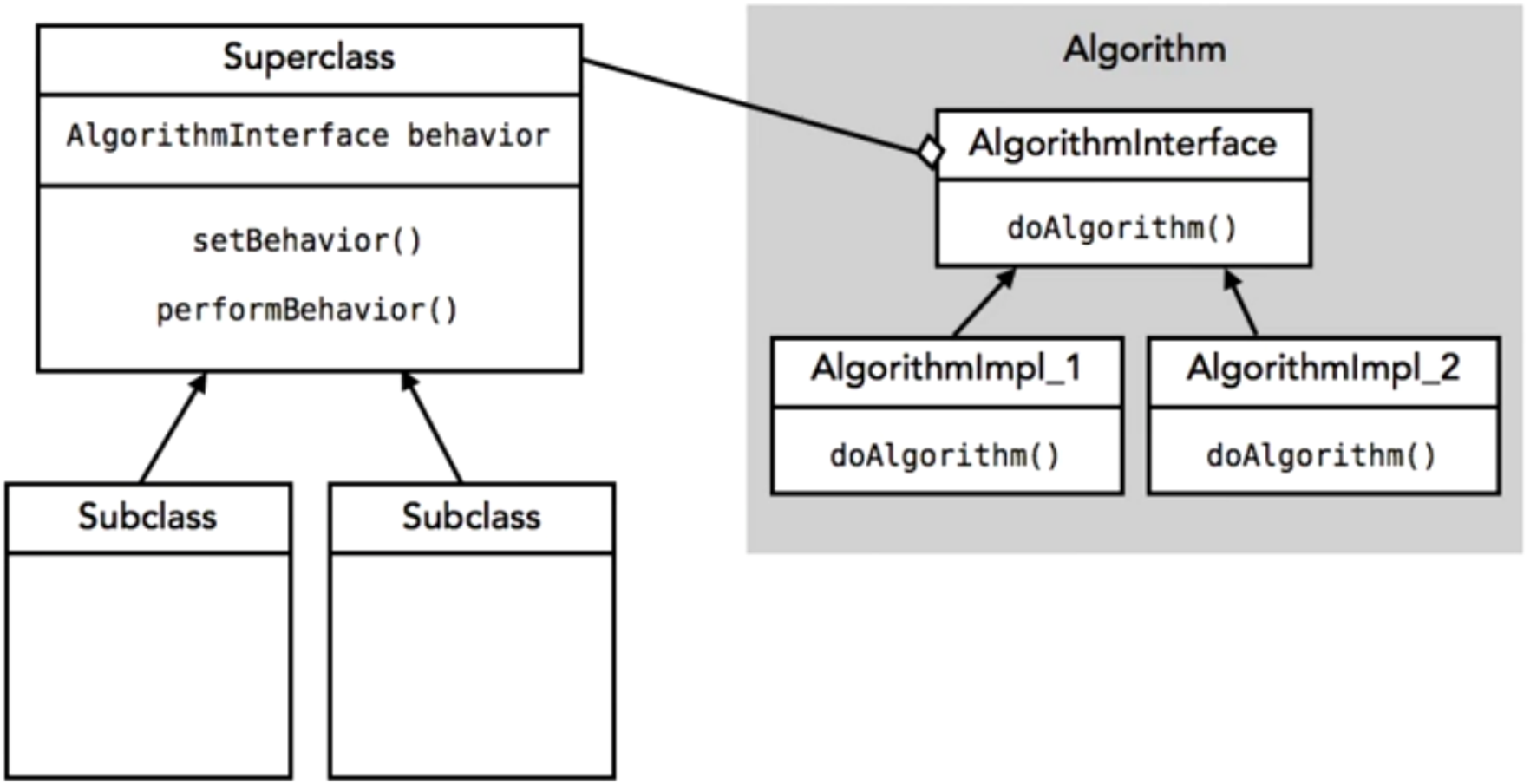




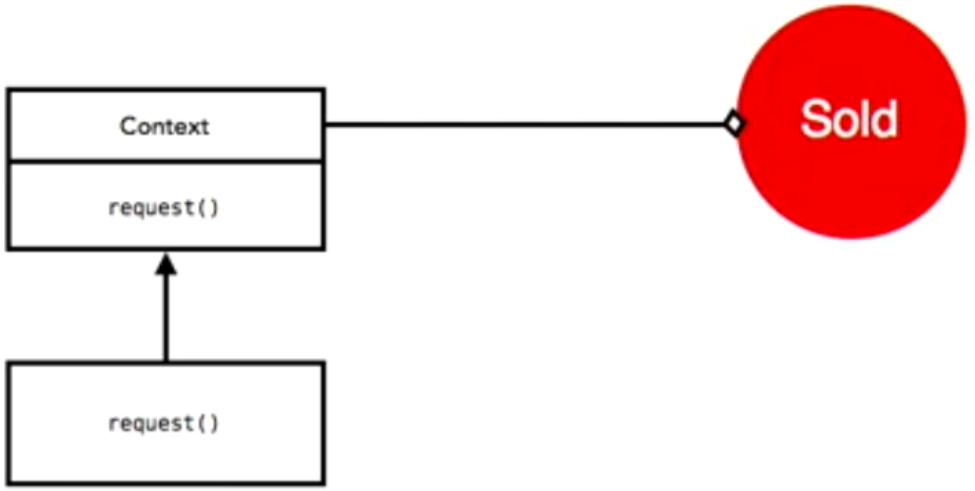
State Pattern

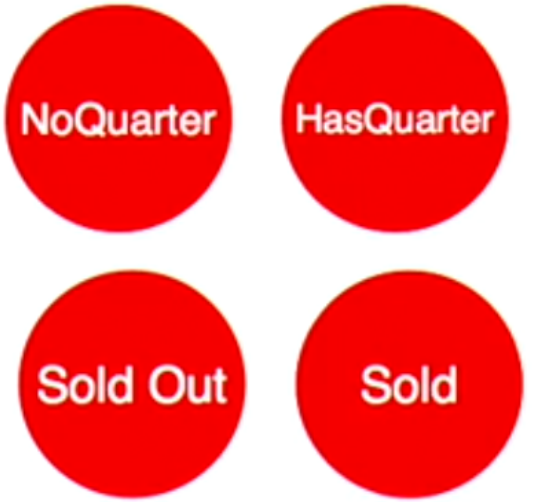


State Pattern



State Pattern





Thank You