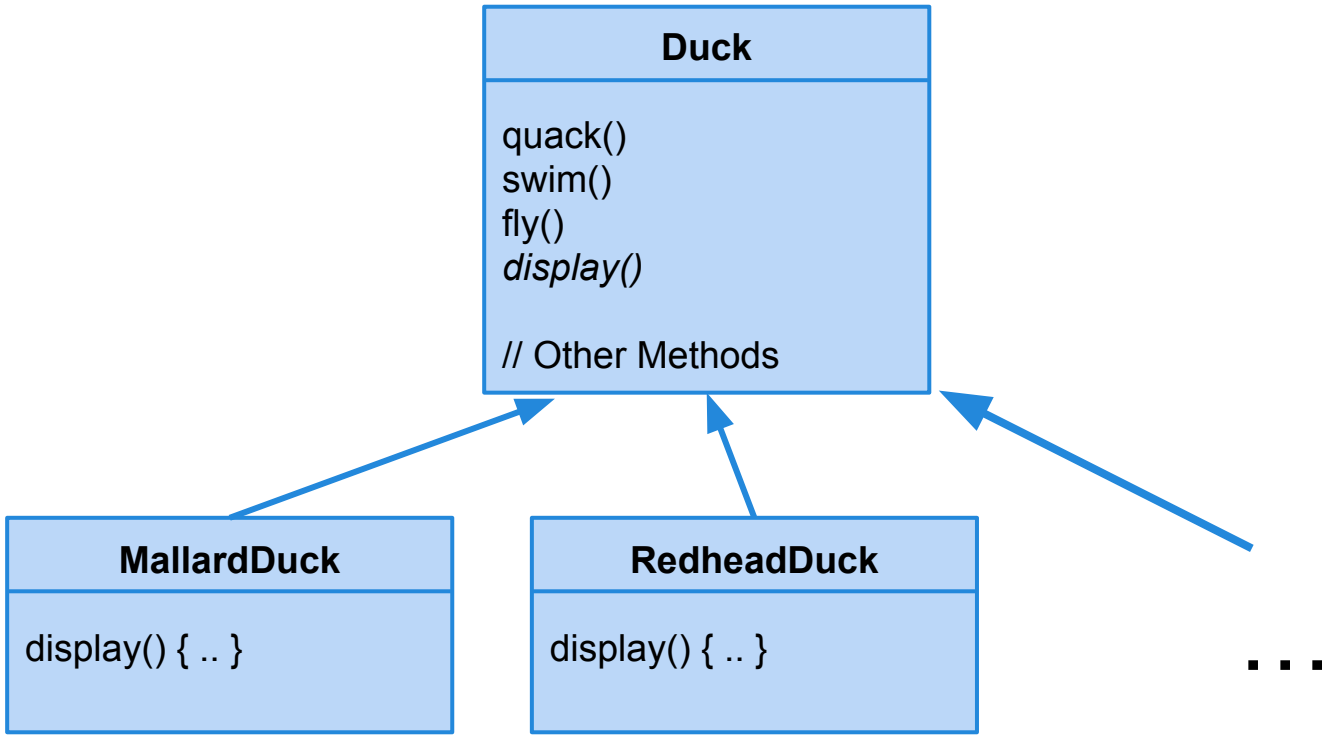
Design Patterns

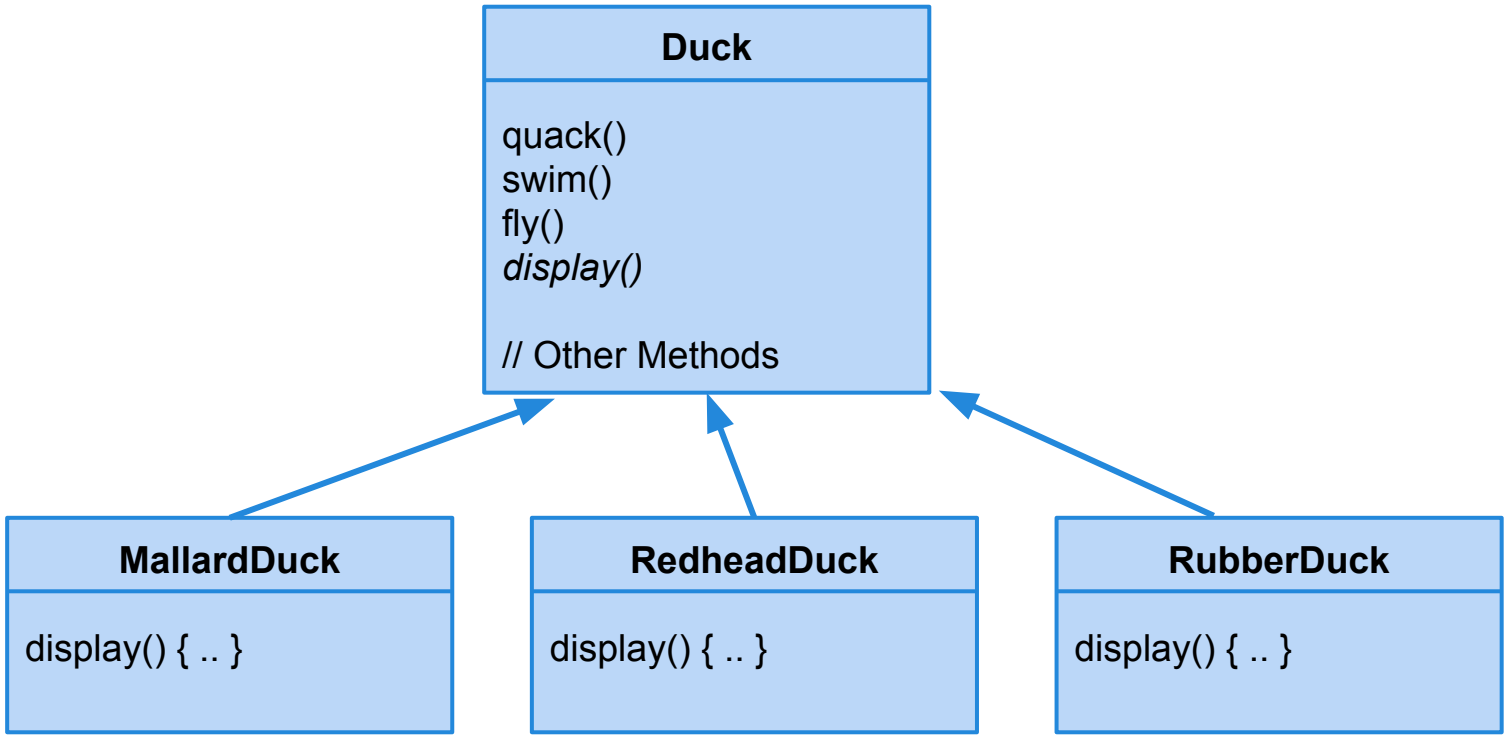
CSCE 740 - Lecture 17 - 10/20/2016

(Partially adapted from Head First Design Patterns by Freeman, Bates, Sierra, and Robson)

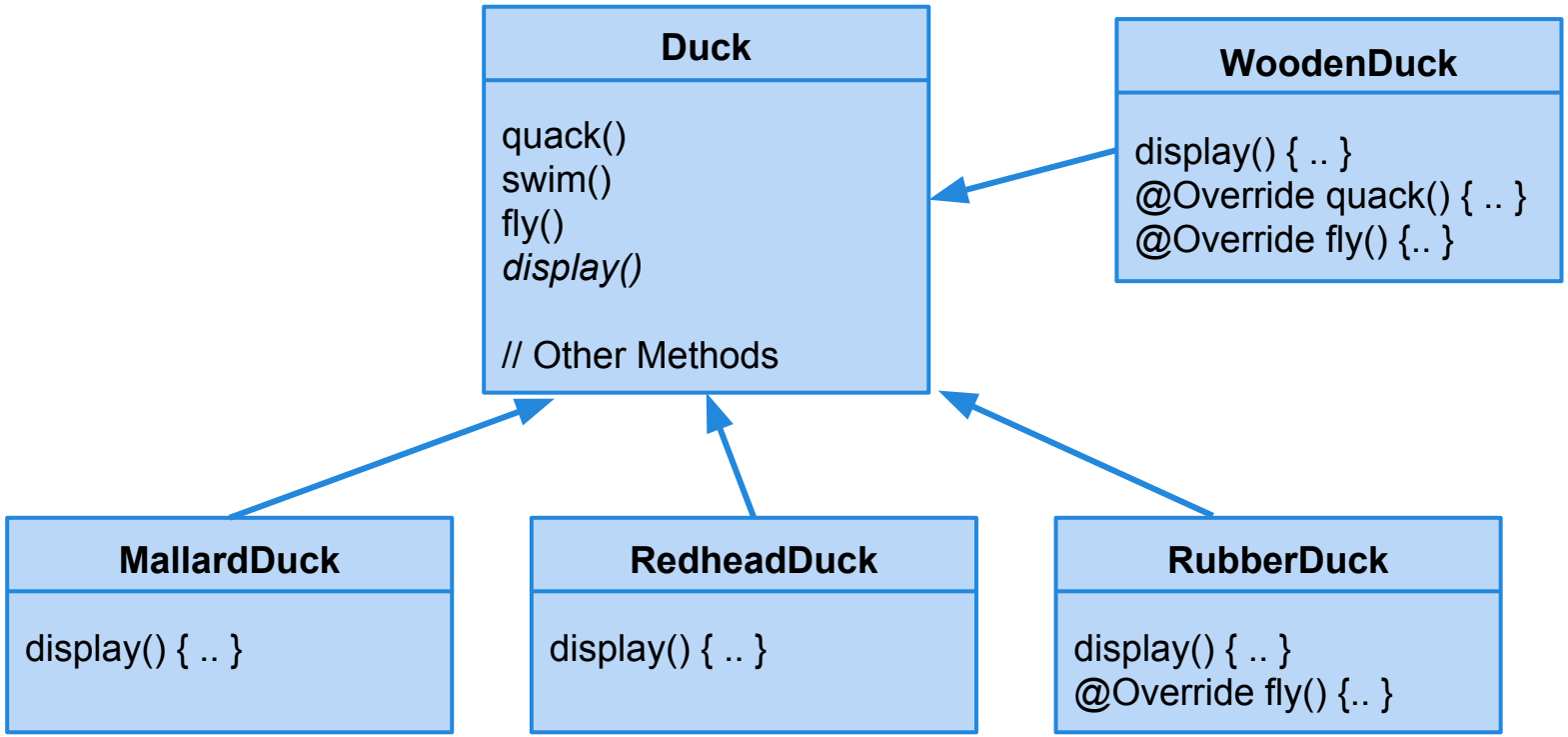
OO Design Exercise:
Building a Better Duck



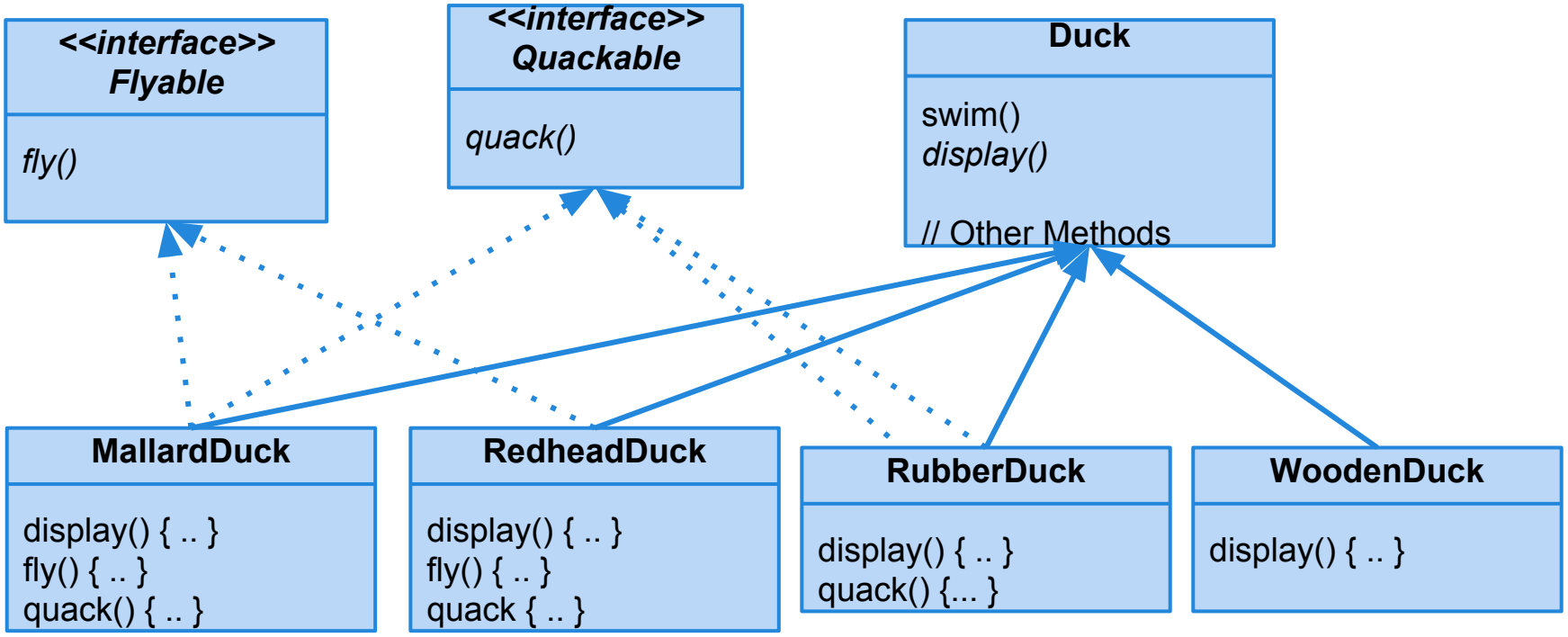
Adding new ducks



Why not override?



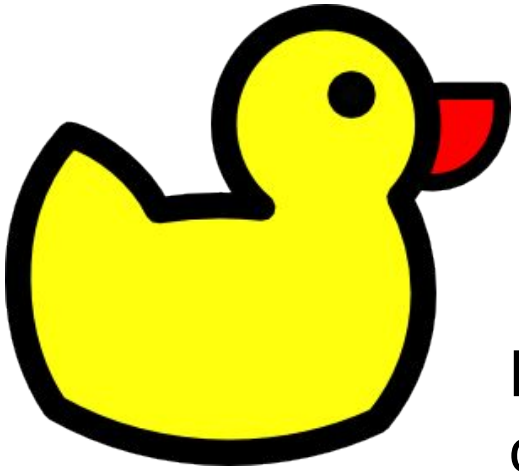
Why not interfaces?



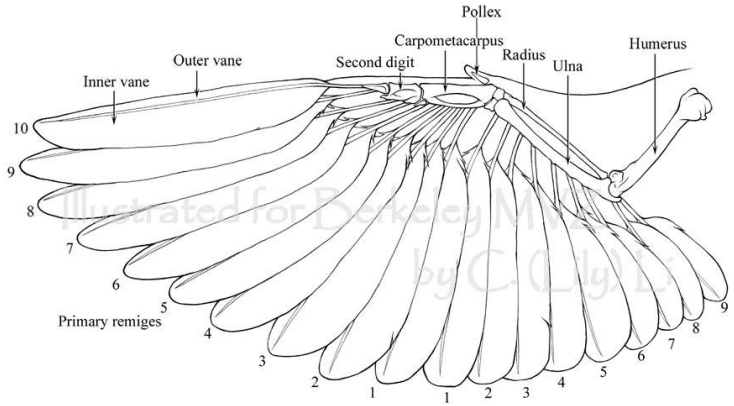
How do we fix this mess?

Apply good OO design principles!

Step 1: Identify the aspects that vary and encapsulate them.



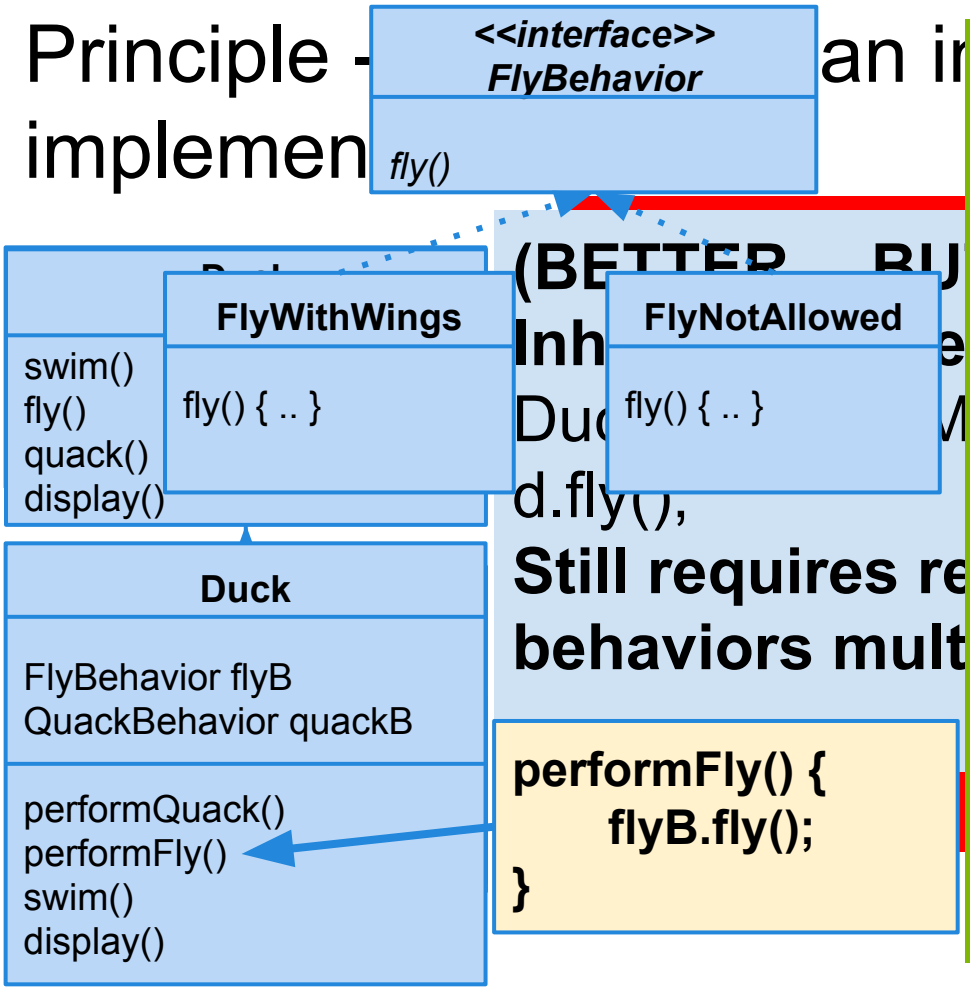
Duck class



Flying behaviors

Step 2:
Implement behaviors as classes

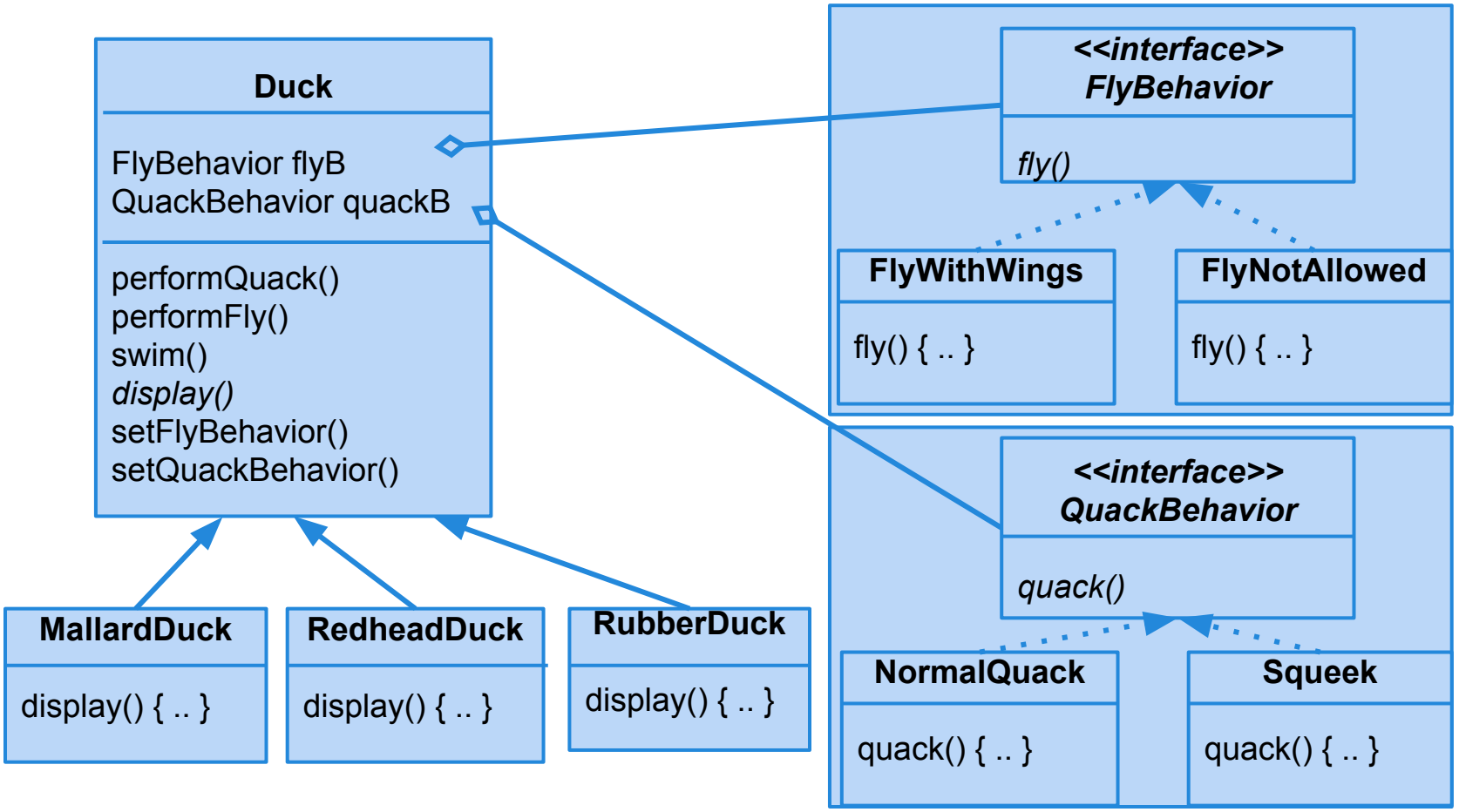
Principle - an in
implemen



(GOOD)
Programming to an interface:
Duck d = new MallardDuck()
d.performFly();
Flying/Quacking behavior called in the same way for all ducks. Only implement specific version of behavior once.

HAS-A can be better than IS-A

Principle: Favor composition over inheritance.



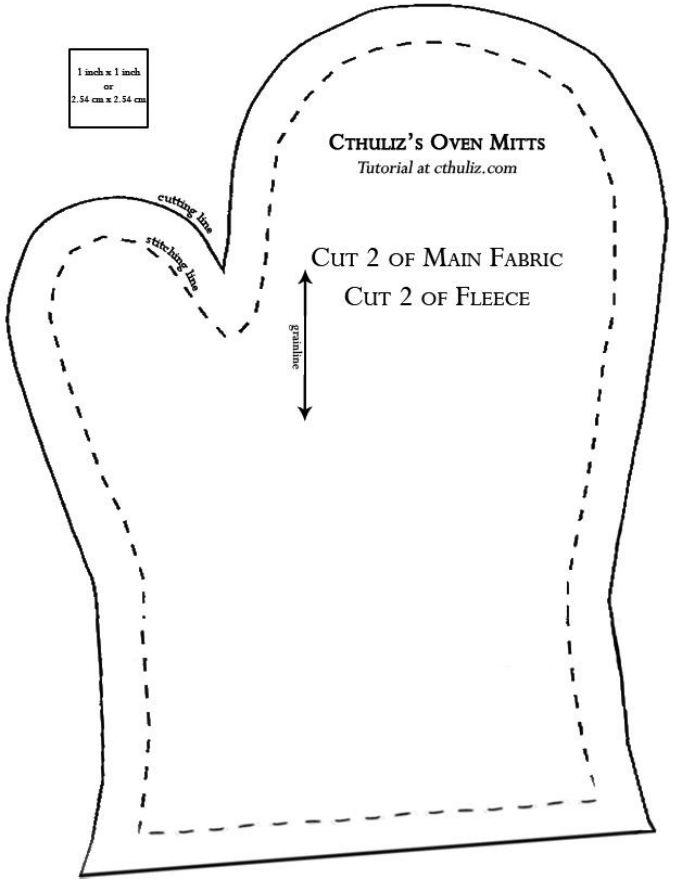
Challenge - Duck Call

A duck call is a device that hunters use to mimic the sound of a duck. How would you implement a duck call in this framework?

Enter... Design patterns

Don't just describe *classes*, describe ***problems***.

Patterns prescribe design guidelines for common problem types.



Guidelines, not solutions

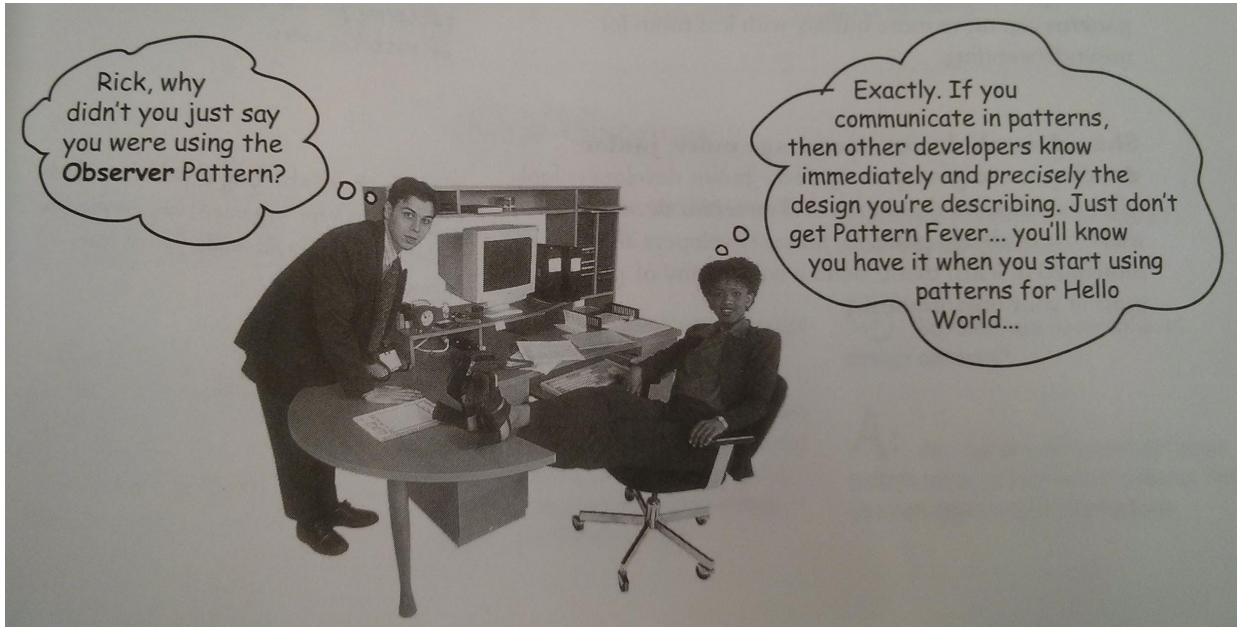
"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over, without ever doing it the same way twice."

- Christopher Alexander

Categories of design patterns

1. Behavioral
   Describe how objects interact.
2. Creational
   Decouple a client from objects it instantiates.
3. Structural
   Clean organization into subsystems.

Why use design patterns?

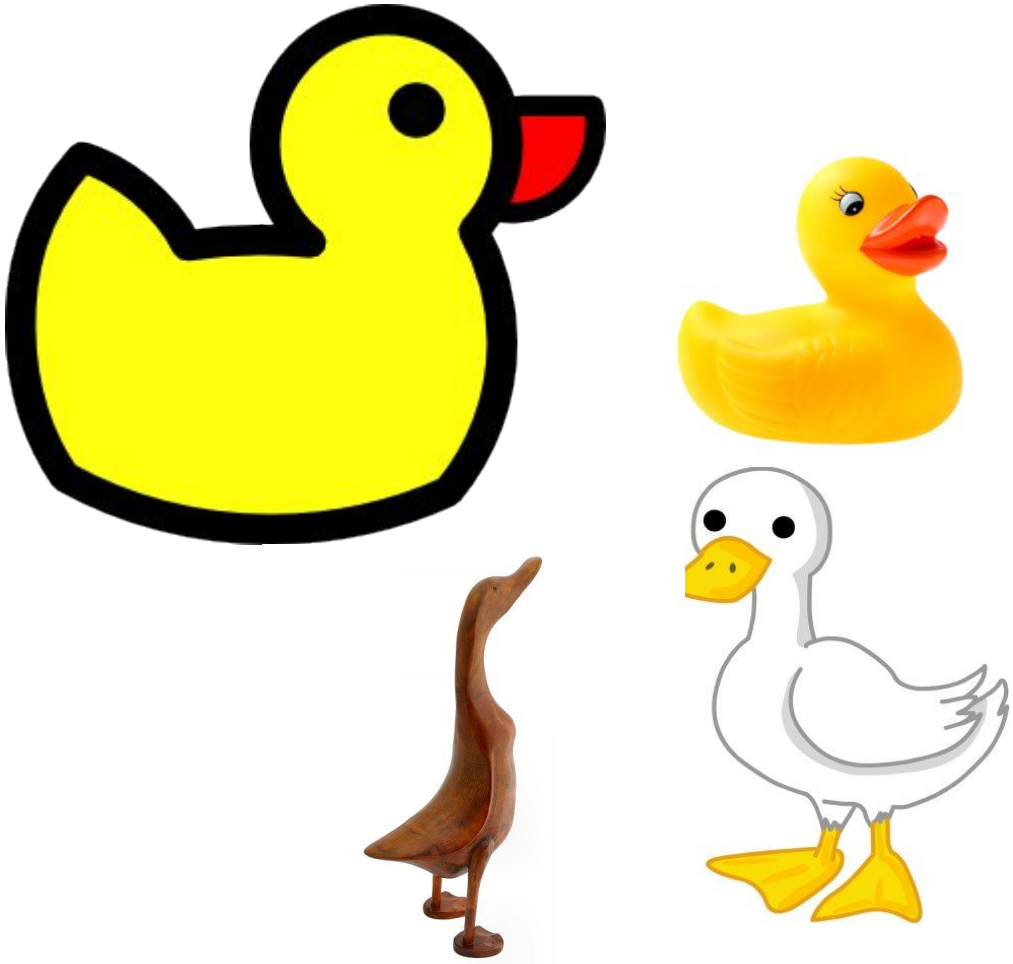


1. Good examples of OO principles.
2. Faster design phase.
3. Evidence that system will support change.
4. Offers shared vocabulary between designers.

You already applied one pattern

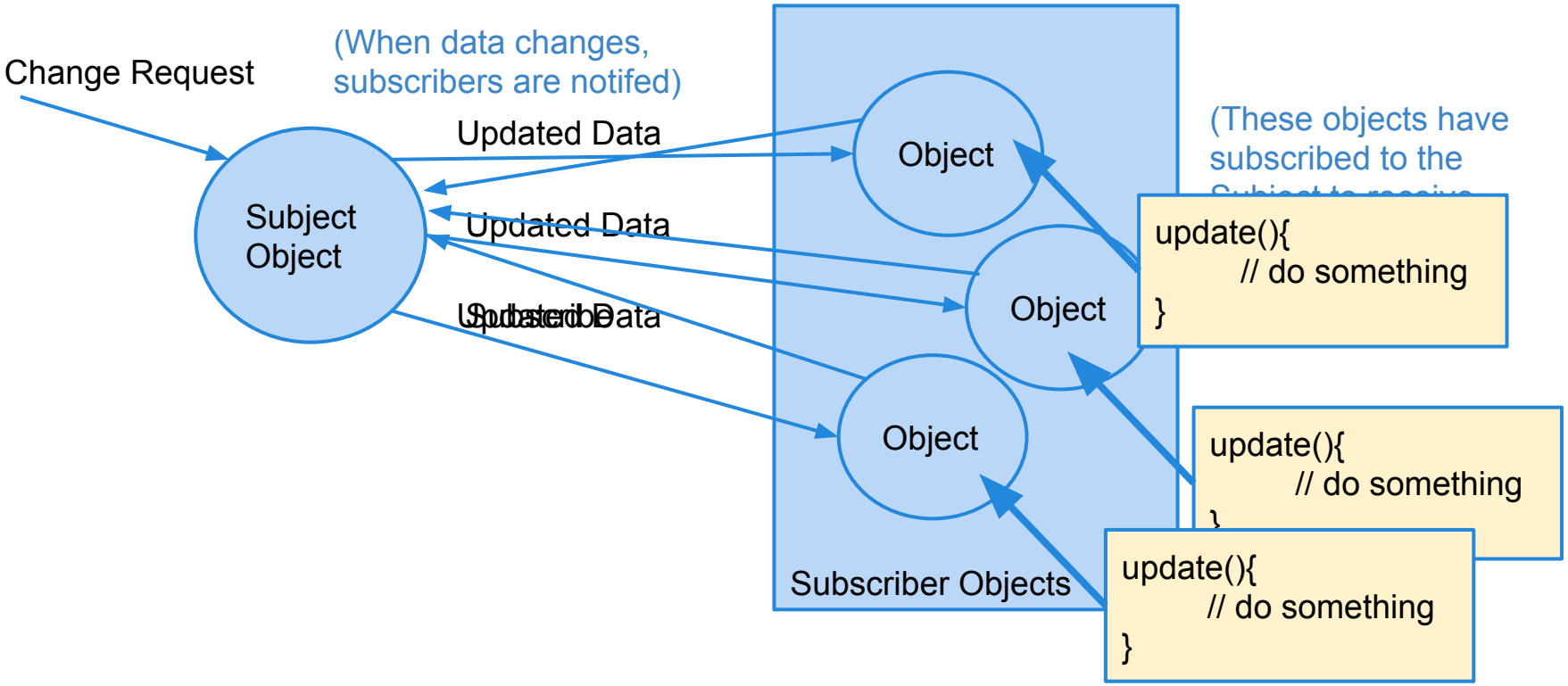
Strategy Pattern

Defines a family of algorithms, encapsulates them, makes them interchangeable.

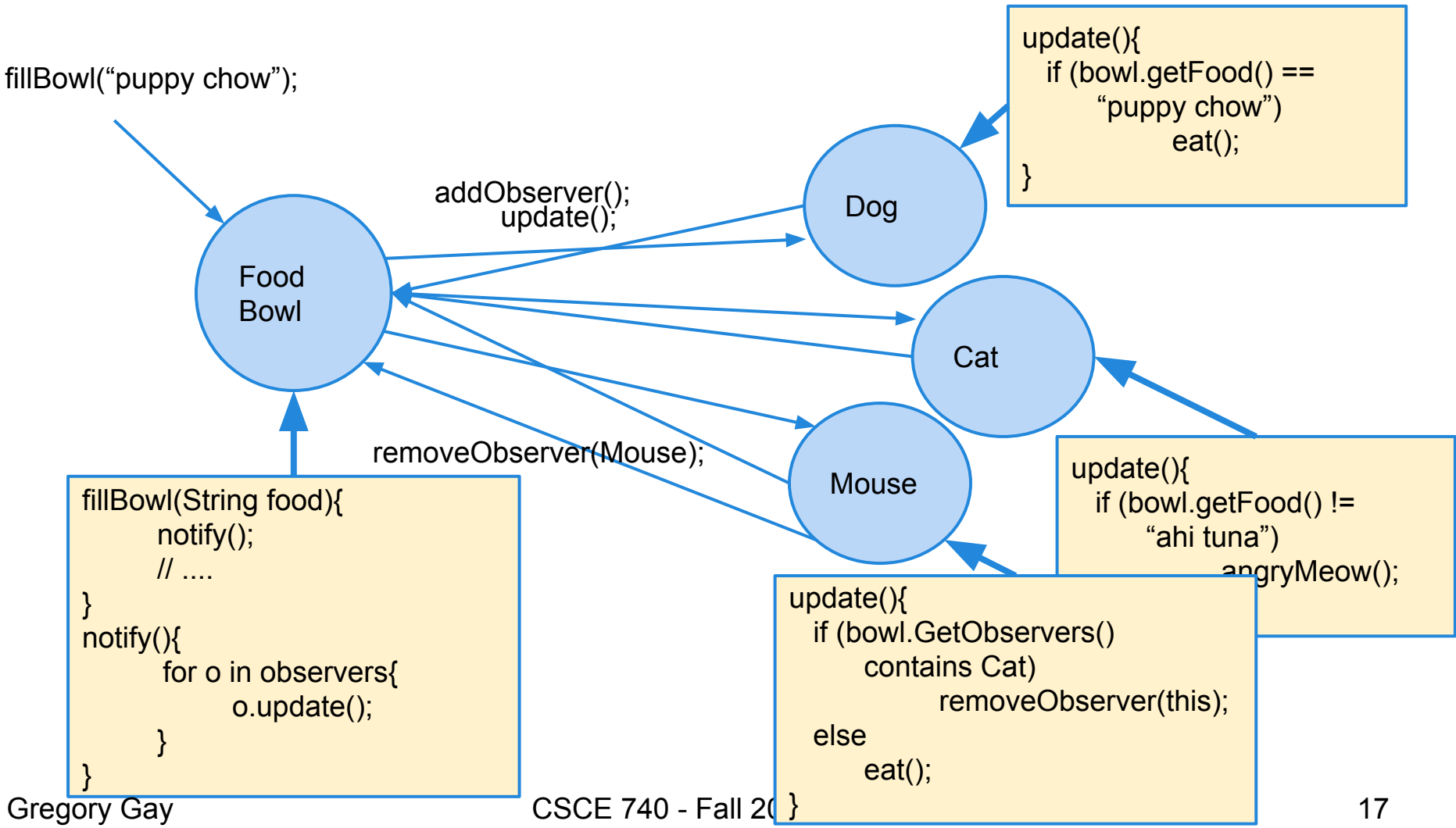


Observer Pattern - Motivation

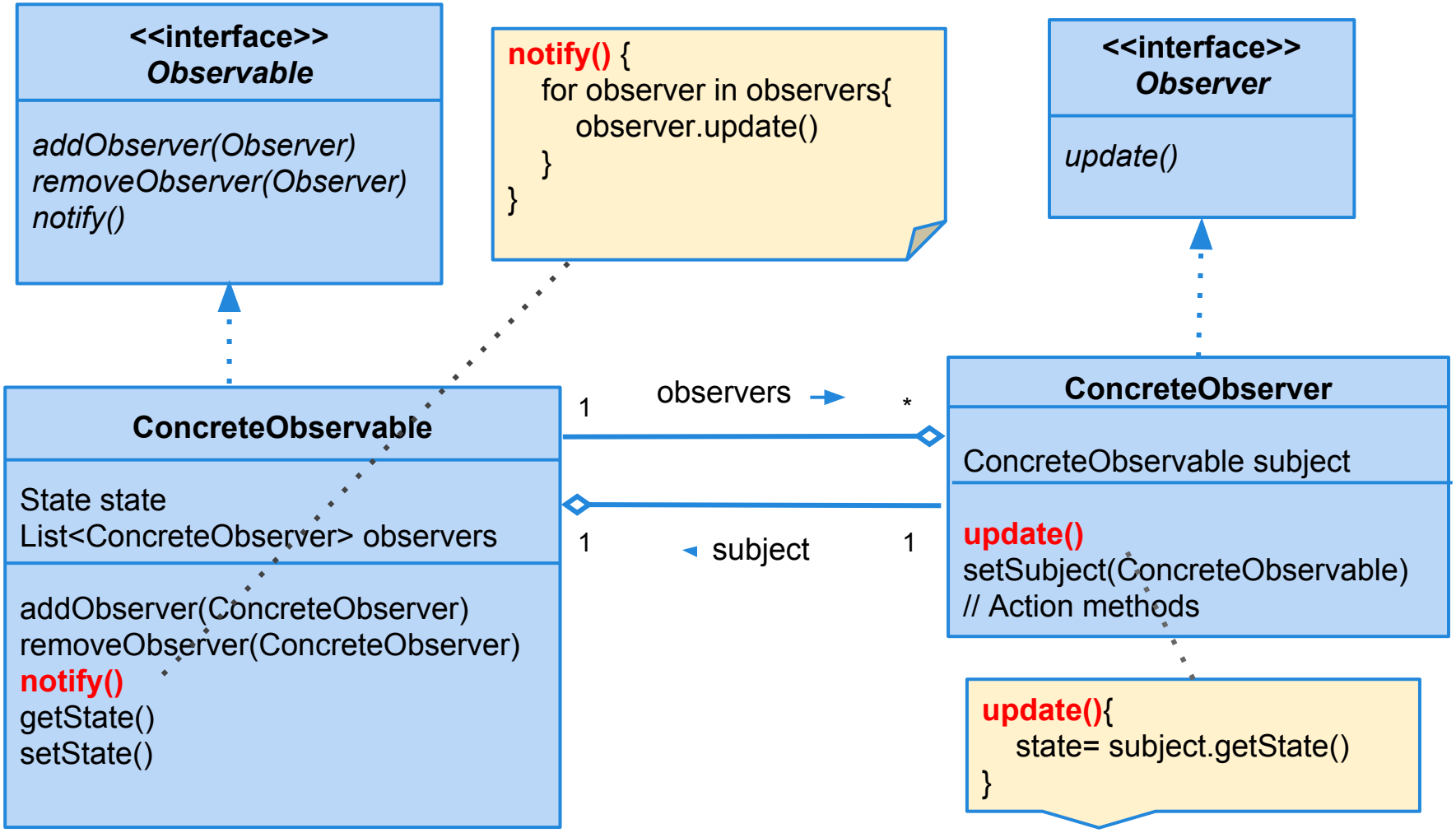
Observer Pattern - Definition



Observer Pattern Example
Pet Feeding



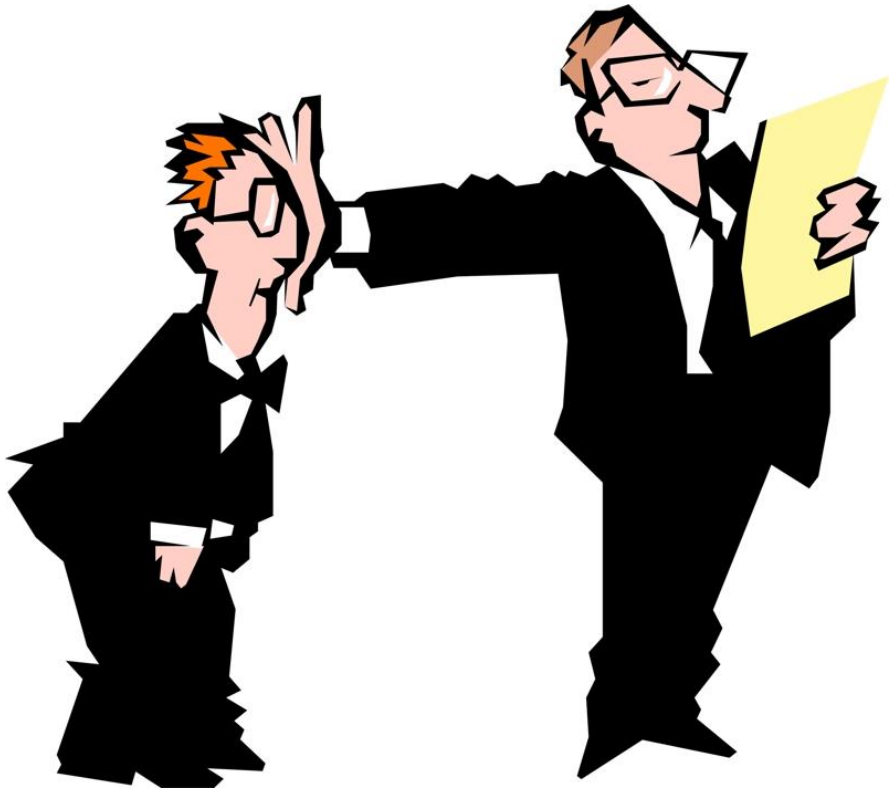
Observer Pattern - In Practice



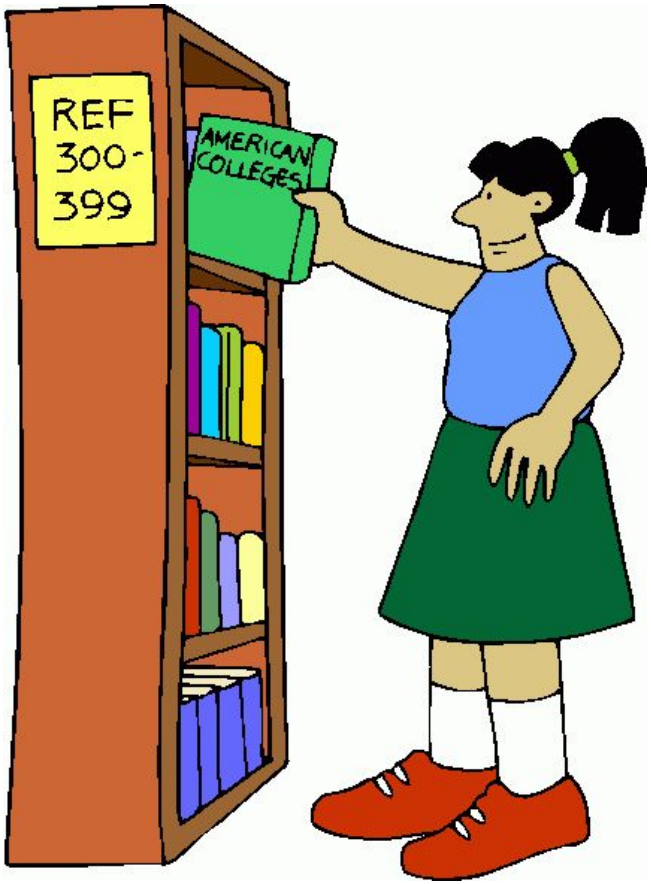
Benefits of Observer Pattern

When objects are loosely coupled, they can interact while lacking knowledge of each other.

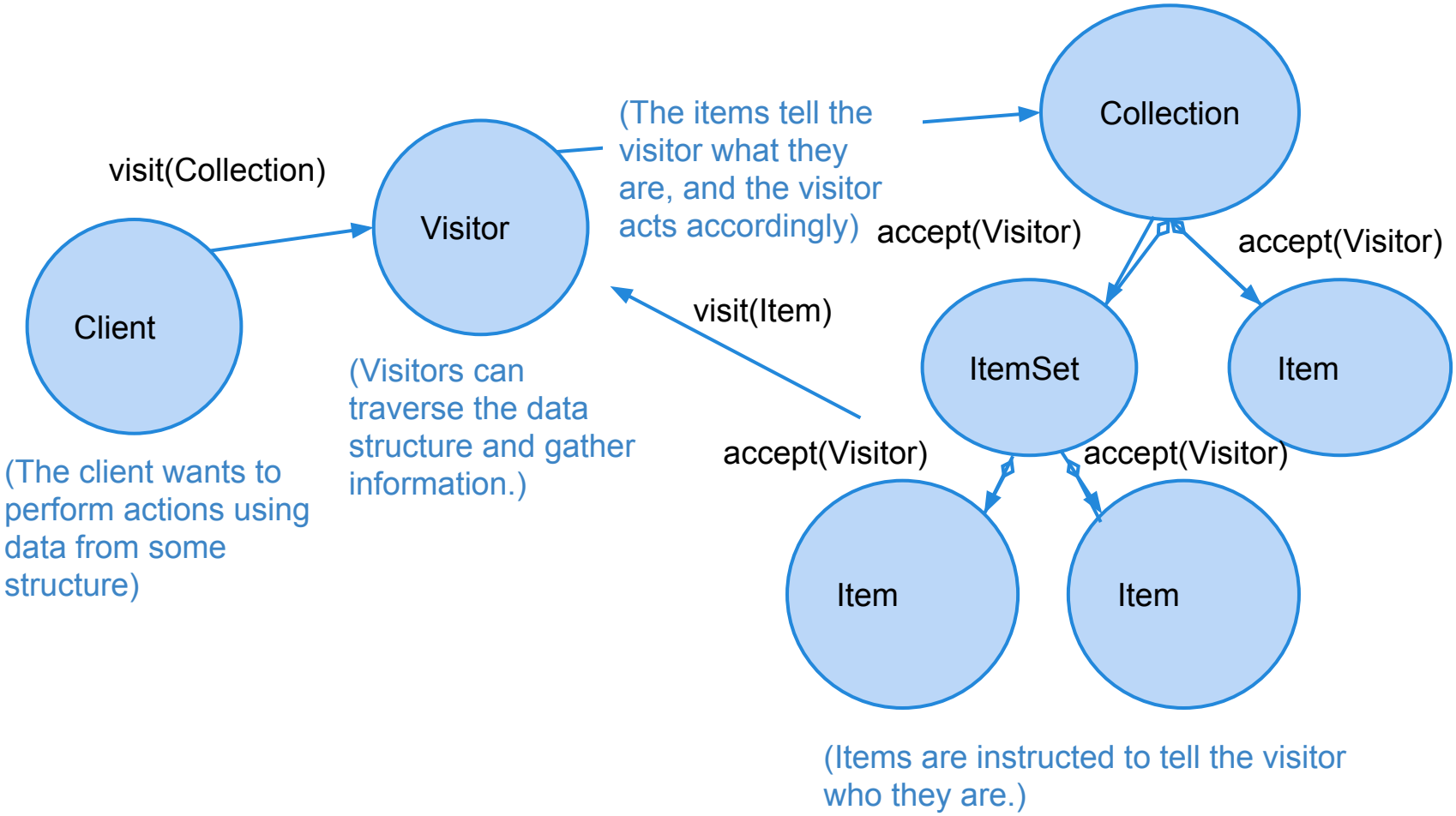
1. Can add new observers at any time.
2. Never need to modify subject.
3. Easy code reuse.
4. Easy change.



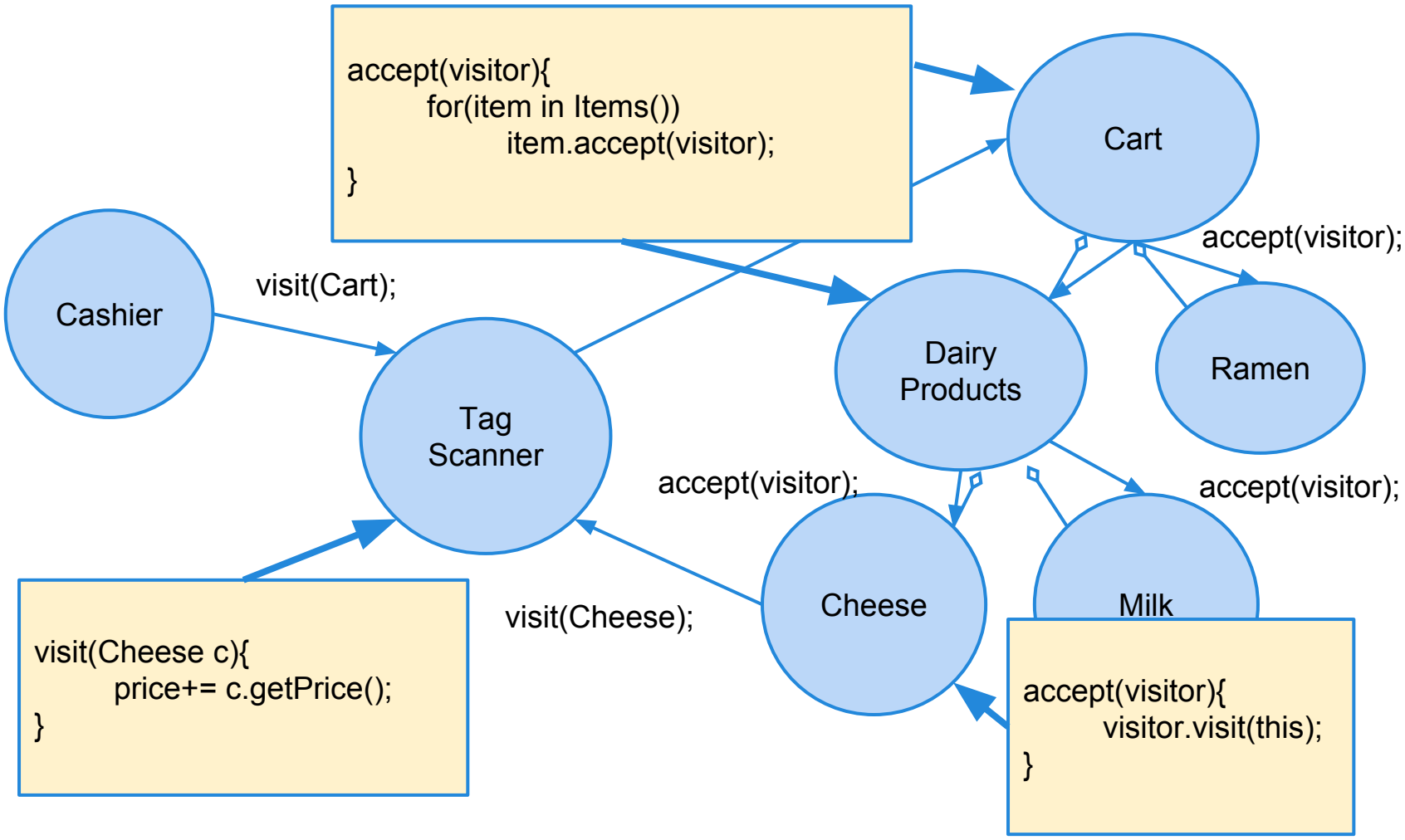
Visitor Pattern - Motivation



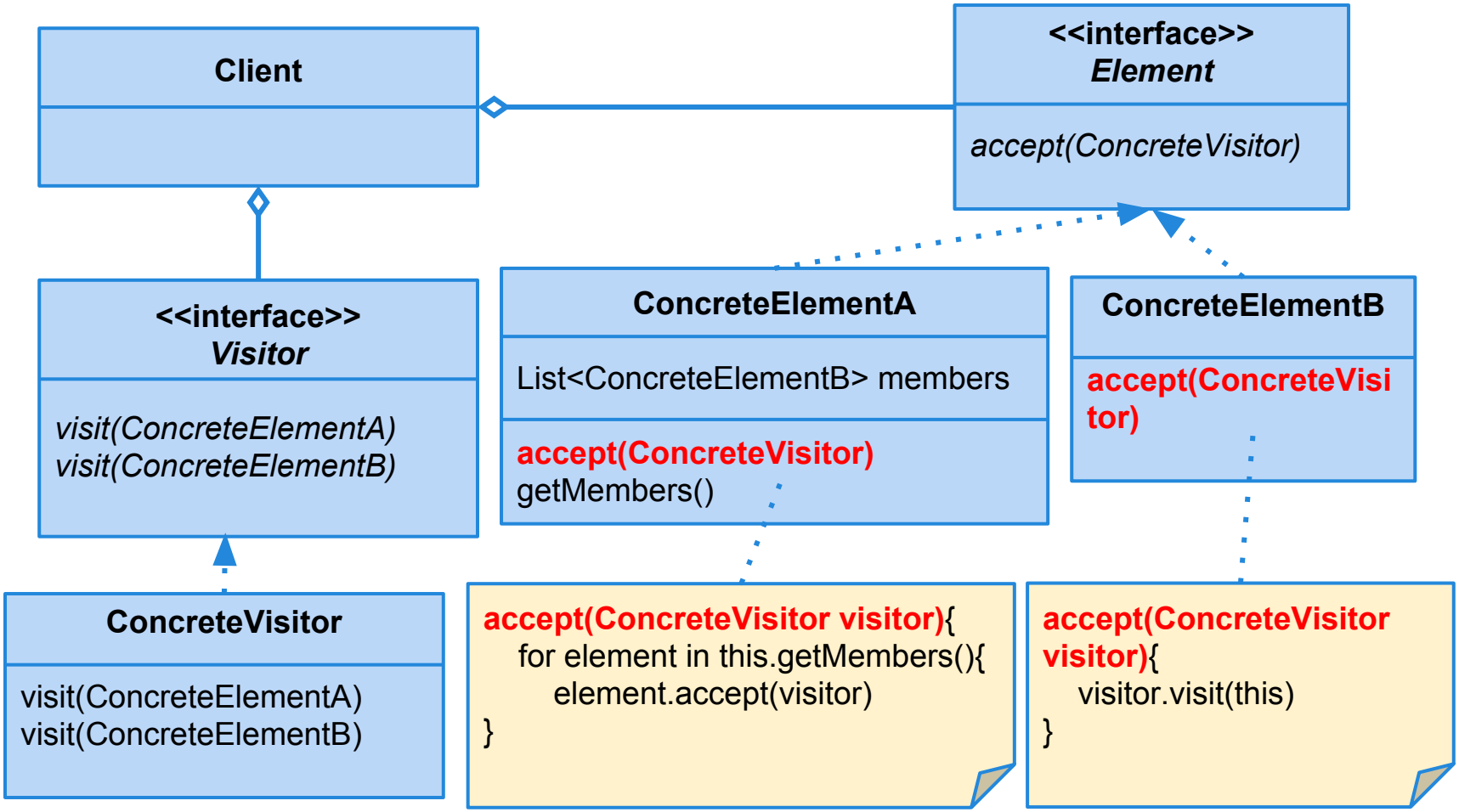
Visitor Pattern - Definition



Visitor Pattern Example
Grocery Checkout



Visitor Pattern - In Practice



Benefits of Visitor Pattern

1. Can add operations to a collection without changing the collection structure.
2. Thus, adding new functionality and operations is easy.
3. Operation code is centralized.

Activity

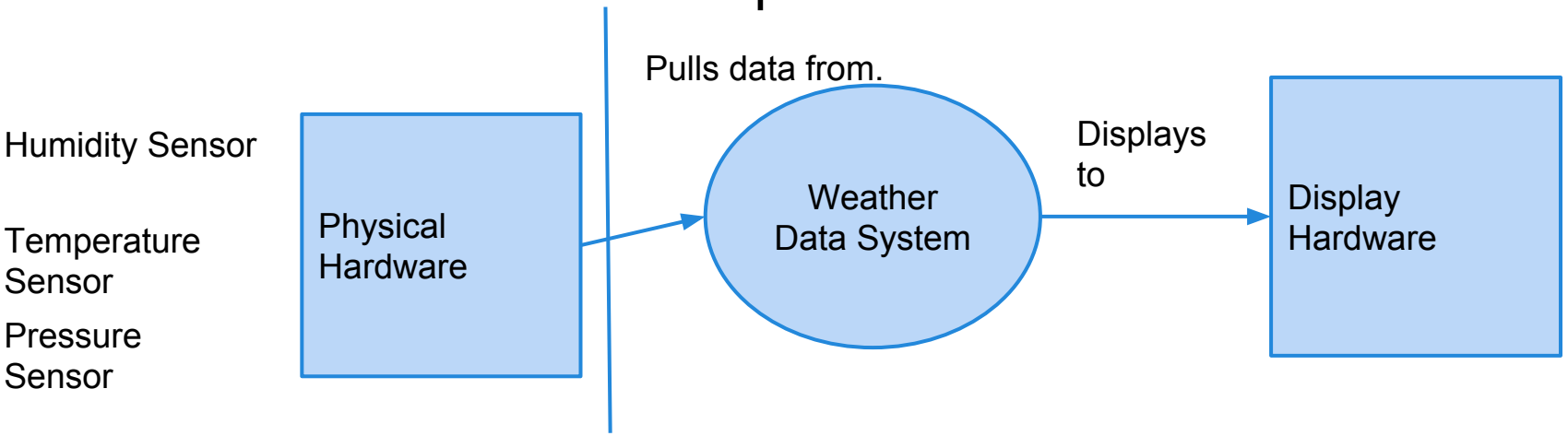
Building a weather monitoring application.

Generates three displays: current conditions, weather statistics, simple forecast.

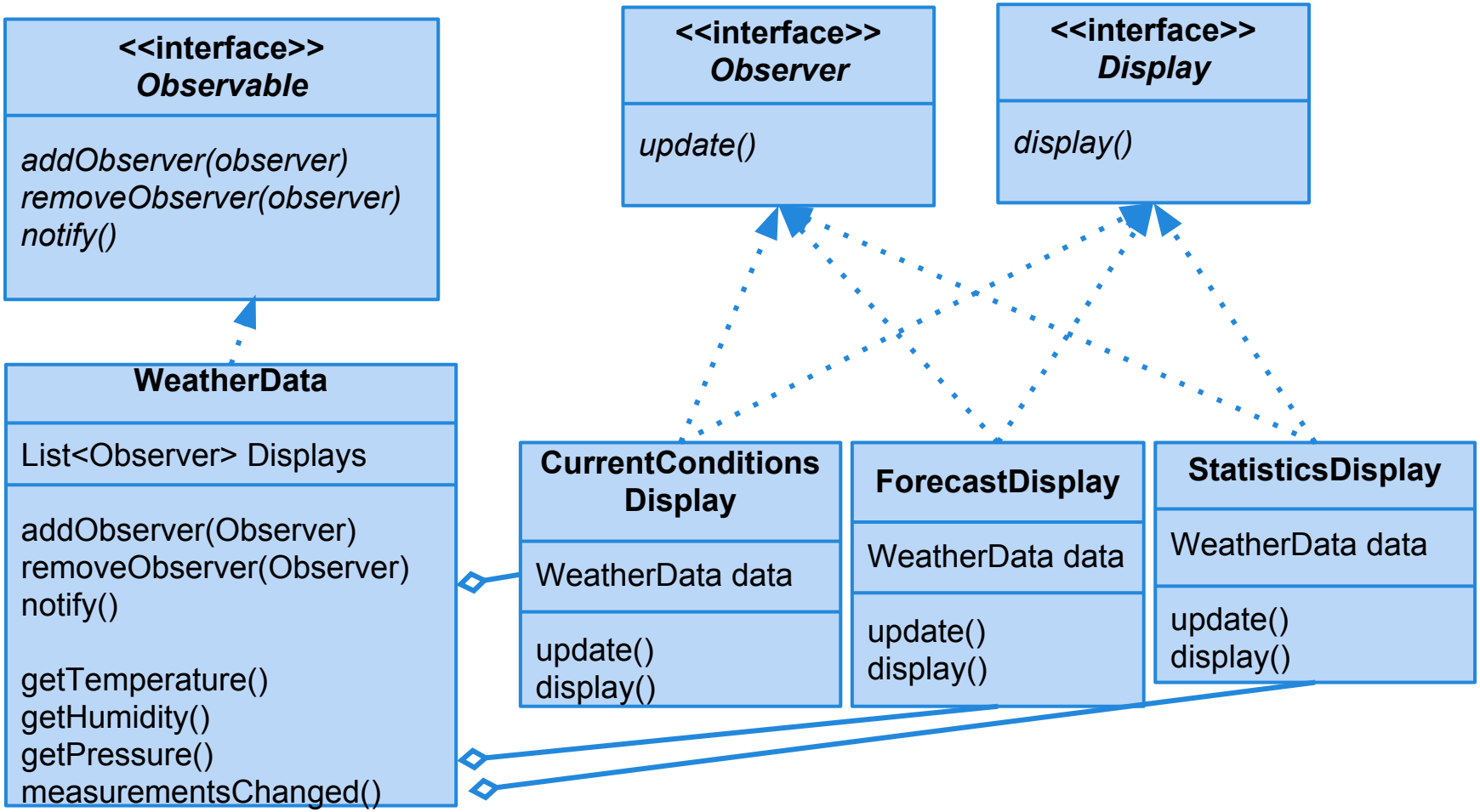
Design system using either visitor or observer pattern.

Provided:

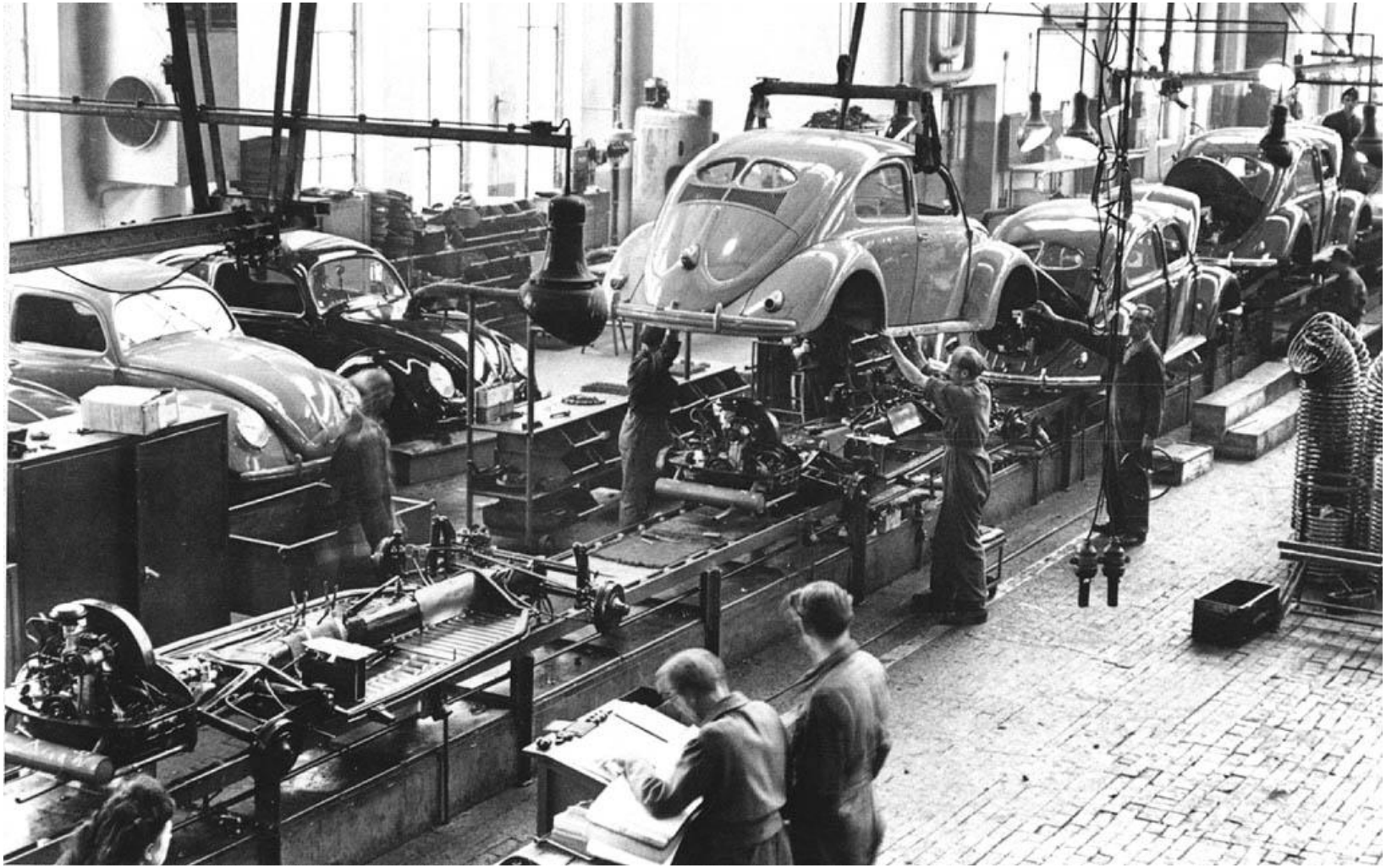
To Implement:



Activity Solution - Observer Pattern



Factory Pattern - Motivation



Factory Pattern - Motivation

```
Pizza orderPizza(){
    Pizza pizza = new Pizza();

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```

Factory Pattern - Motivation

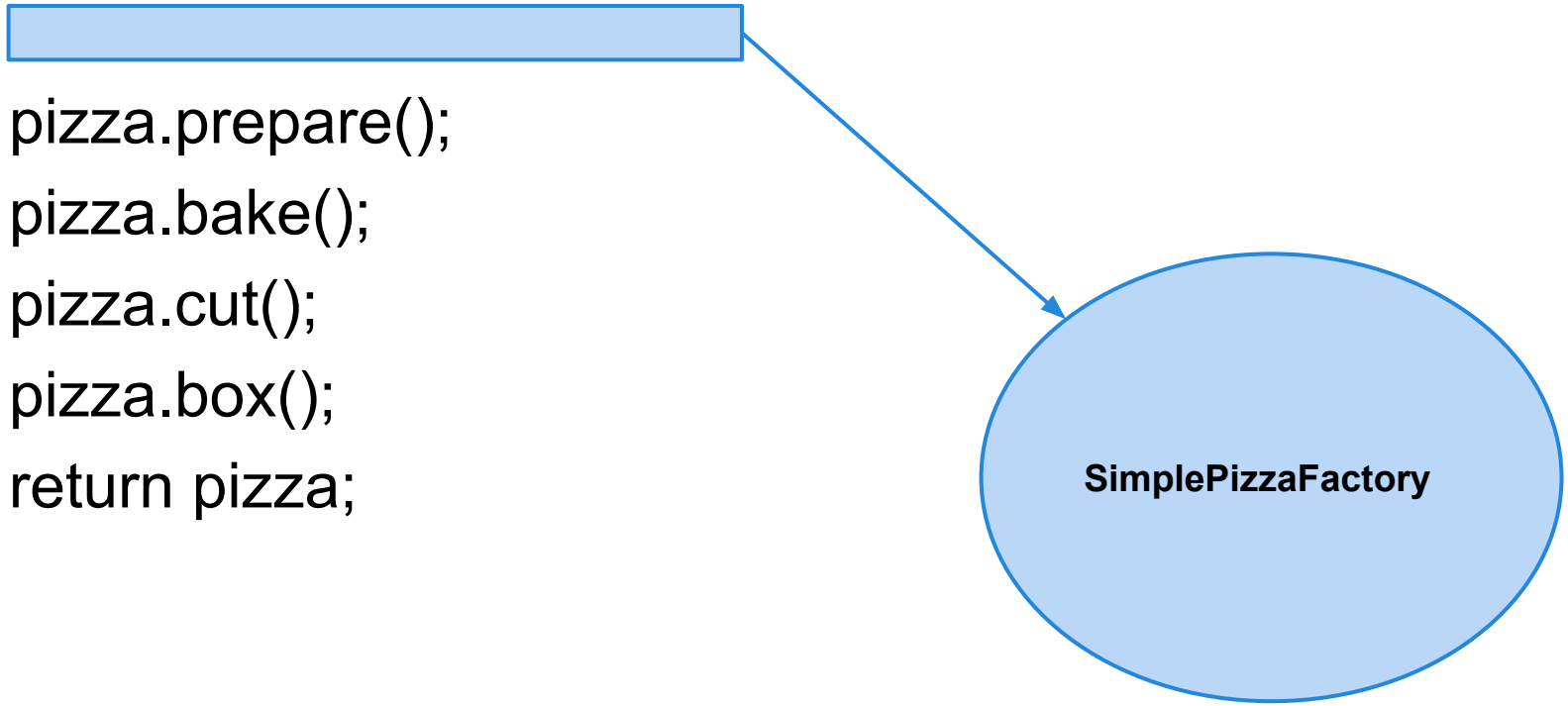
```
Pizza orderPizza(String type){
    Pizza pizza;
    if (type.equals("cheese")){
        pizza = new CheesePizza();
    else if(type.equals("pepperoni")){
        pizza = new PepperoniPizza();
    }
    // Prep methods
}
```

Factory Pattern - Motivation

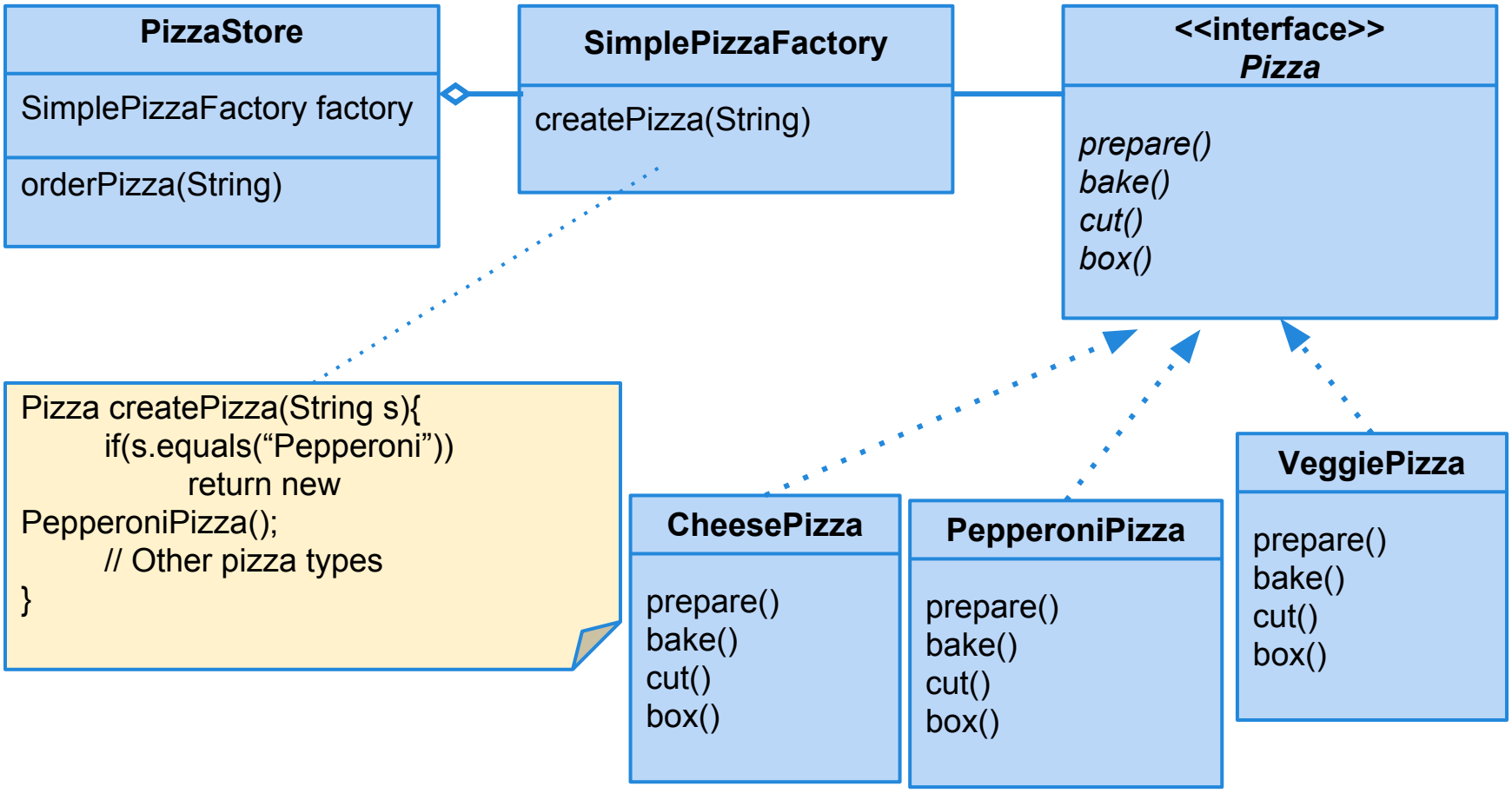
```
Pizza orderPizza(String type){
    Pizza pizza;
    if (type.equals("cheese")){
        pizza = new CheesePizza();
    else if(type.equals("pepperoni")){
        pizza = new PepperoniPizza();
    } else if(type.equals("veggie")){
        pizza = new VeggiePizza();
    }
    // Prep methods
}
```

Factory Pattern - Motivation

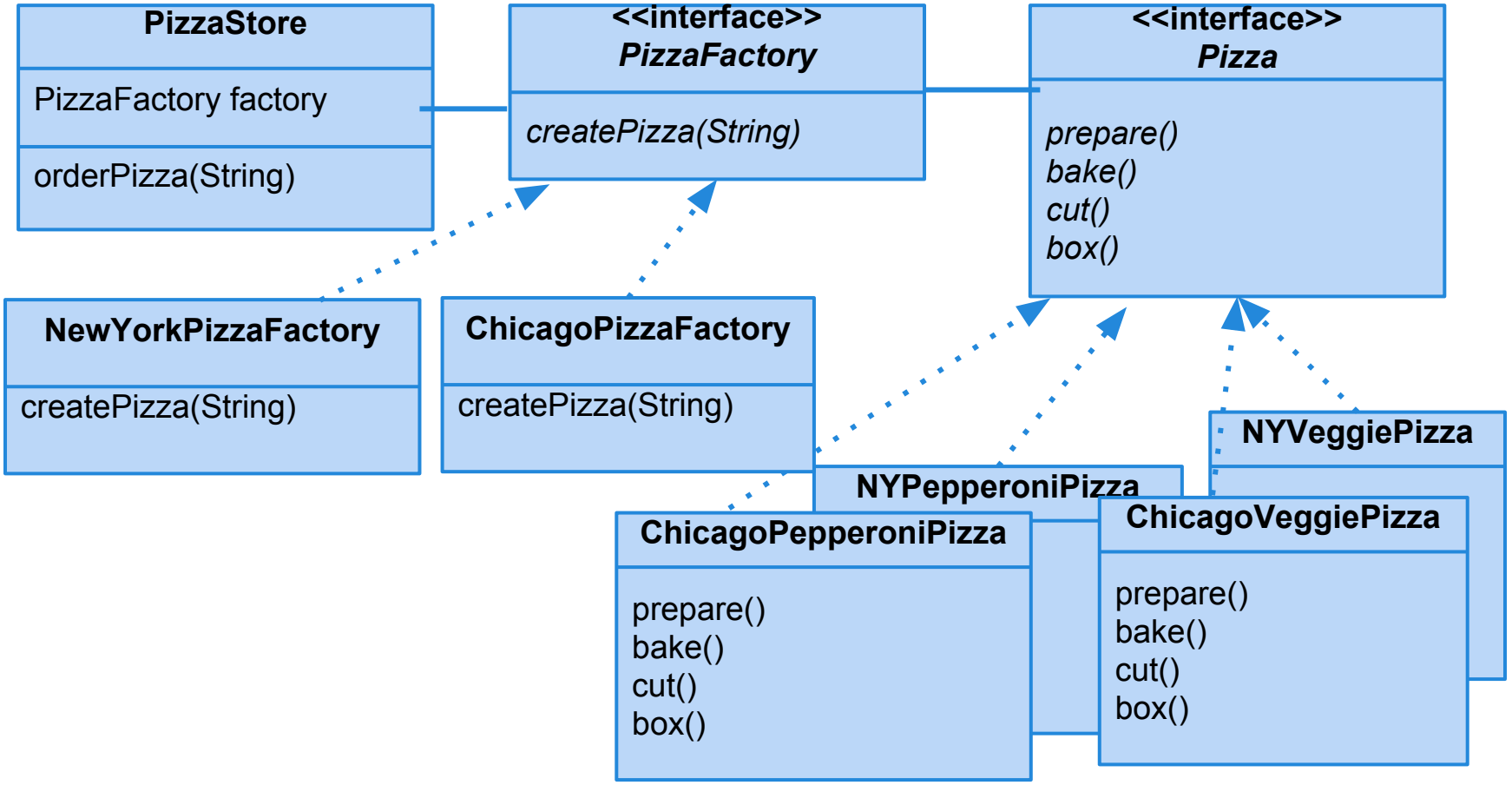
```
Pizza orderPizza(String type){
    Pizza pizza;

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```



The Simple Factory



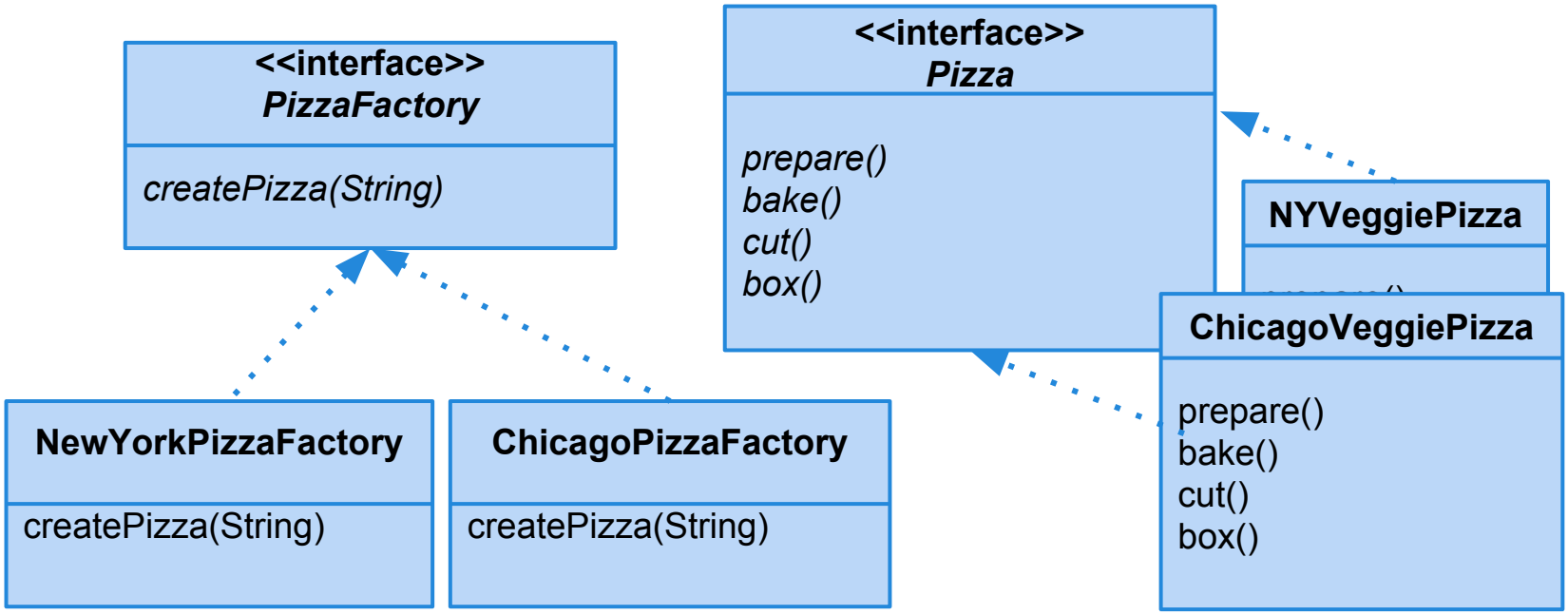
Franchising the Factory



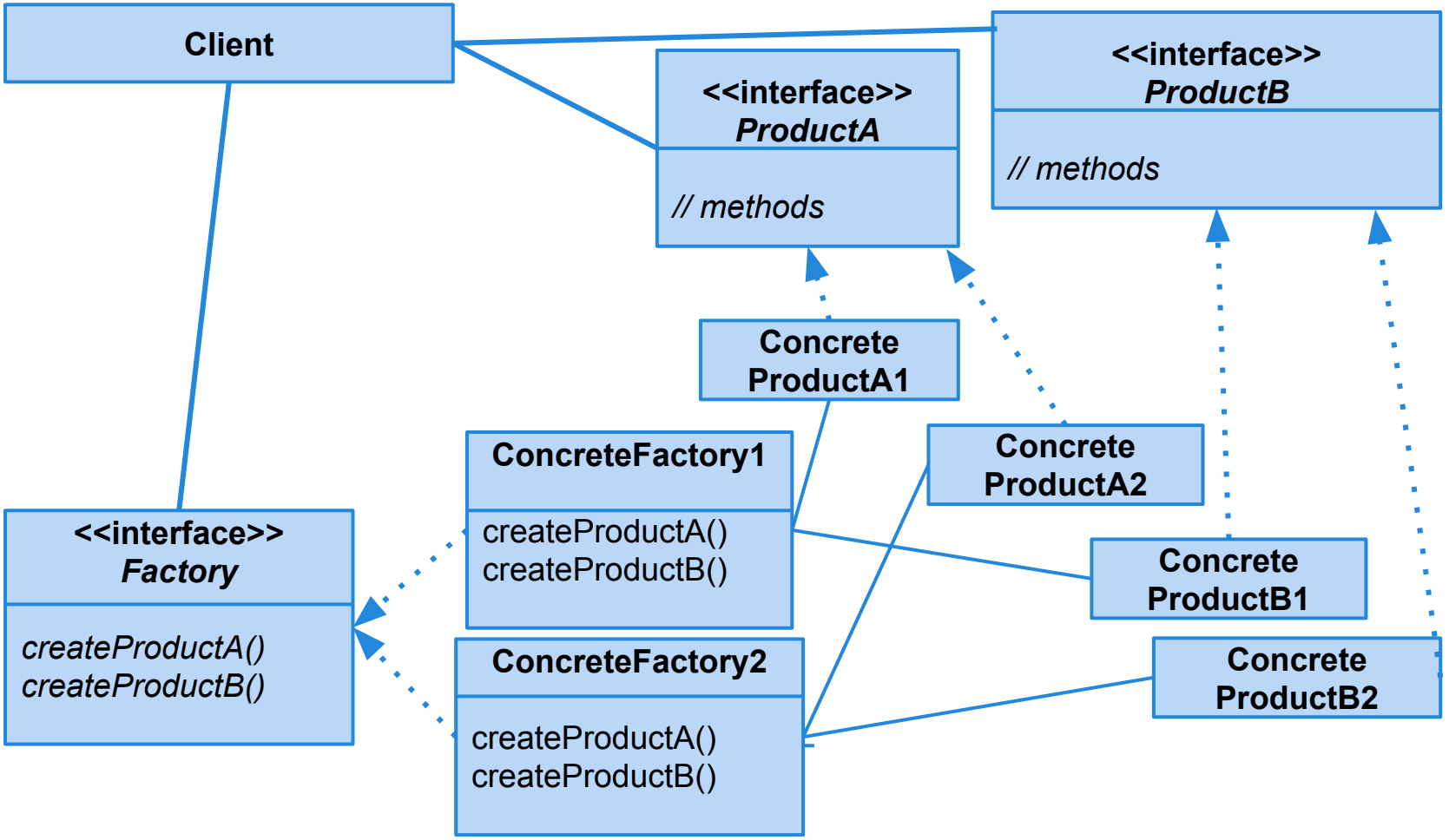
Factory Pattern - Definition

Defines an interface for creating an object, but lets subclasses decide which object to instantiate.

Allows reasoning about **creators** and **products**.

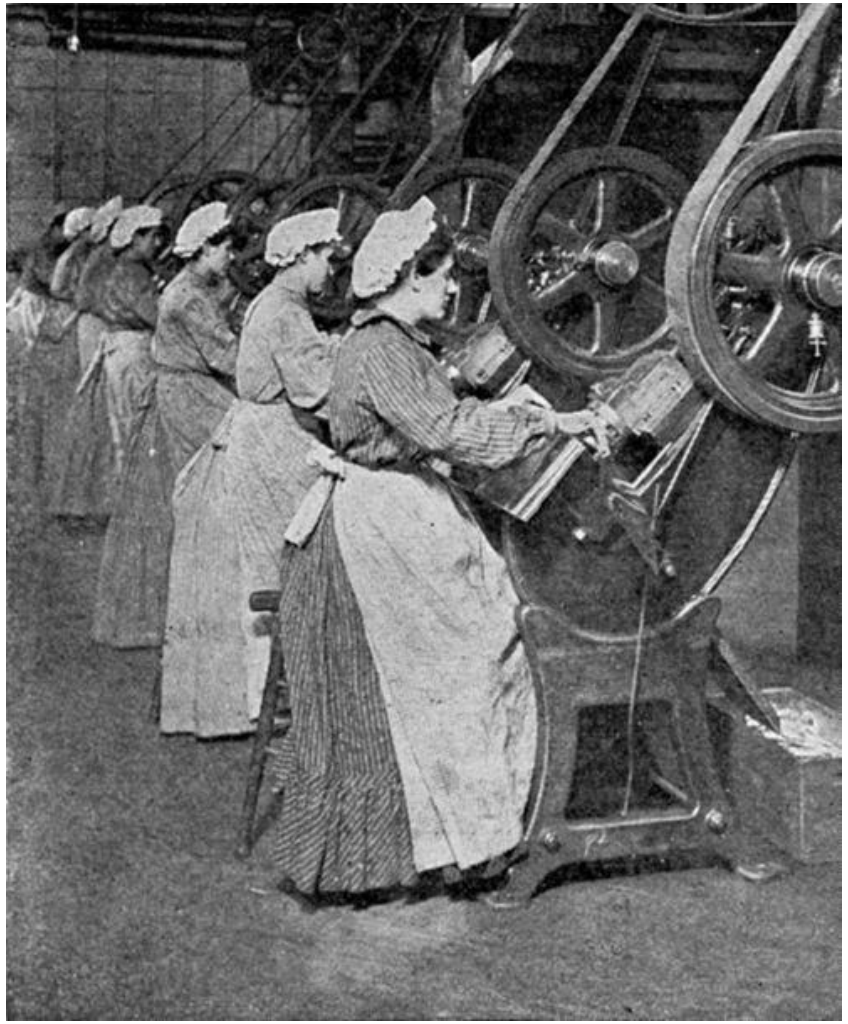


Factory Pattern - In Practice



Benefits of Factory Pattern

1. Loose coupling.
2. Creation code is centralized.
3. Easy to add new classes.
4. Lowered class dependency (can depend on abstractions, not concrete classes).



Why not use a design pattern?

What are the drawbacks to using patterns?

- Potentially over-engineered solution.
- Increased system complexity.
- Design inefficiency.

How can we avoid these pitfalls?

Resources

Web:

- oodesign.com
- c2.com/cgi/wiki?PatternIndex

Book:

- Head First Design Patterns, by Eric Freeman, Bert Bates, Kathy Sierra, and Elisabeth Robson.
- Design Patterns: Elements of Reusable Object Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (Gang of Four)

We Have Learned

When in doubt:

1. Reason about the problem, then the objects.
2. Patterns provide templates for OO design.

Patterns come in many flavors.

Think about patterns and MEAT (hint, hint).

Next Time

- Design Patterns, round 2
- Homework
  - Questions on class diagrams?