COM S 362
Object-Oriented Analysis & Design

Abstract Factory and Strategy Patterns

Reading

Alexander Shvets. Dive Into Design Patterns, 2020.

- Abstract Factory
- Strategy

Behavioral Patterns

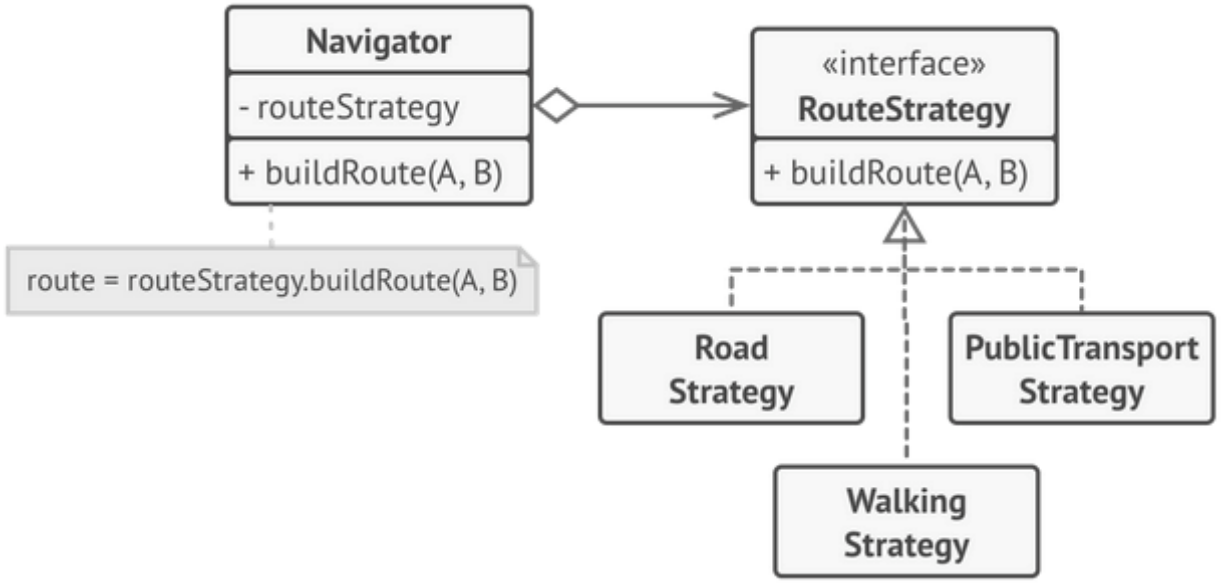
- Chain of Responsibility
  - Pass requests along a chain of handlers.
- Command
  - Turn a request (method calls) into an object.
- Iterator
  - Traverse elements of a collection.
- Mediator
  - Reduce dependencies between objects.
- Memento
  - Save and restore the state of an object.
- Observer
  - Publish and subscribe to events.
- State
  - State based behavior.
- Strategy
  - Define a family of algorithms, each in their own class
- Visitor
  - separate algorithms from the objects on which they operate

Strategy Pattern: Overview

- Intent
  - Define a family of algorithms, encapsulate each one, and make them interchangeable
  - Strategy lets the algorithms vary independently from clients that use it

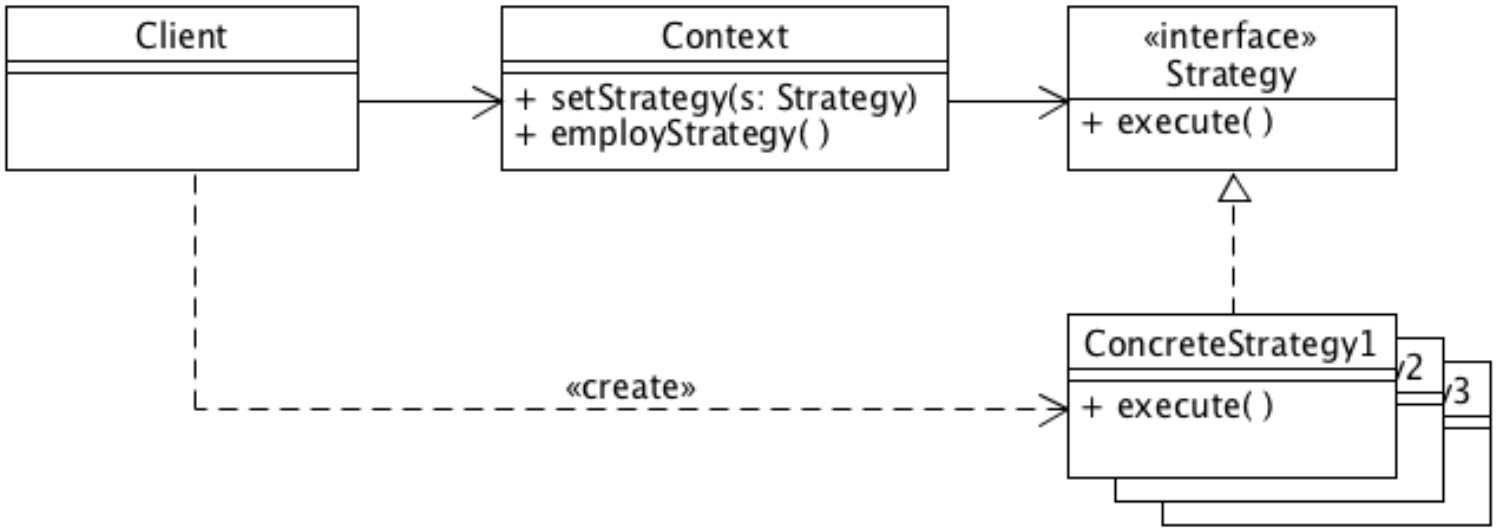
Strategy Pattern: Problem/Solution

```
class Navigator {
    Route buildRoute(Location a, Location b, TrasportMethod method) {
        Route route = new Route();
        if (method == WALK) {
            // ...
        } else if (method == DRIVE) {
            // ...
        } else if (method == BOAT) {
            // ...
        } else if (method == PUBLIC_TANSIT) {
            // ...
        }
        return route;
    }
}
```



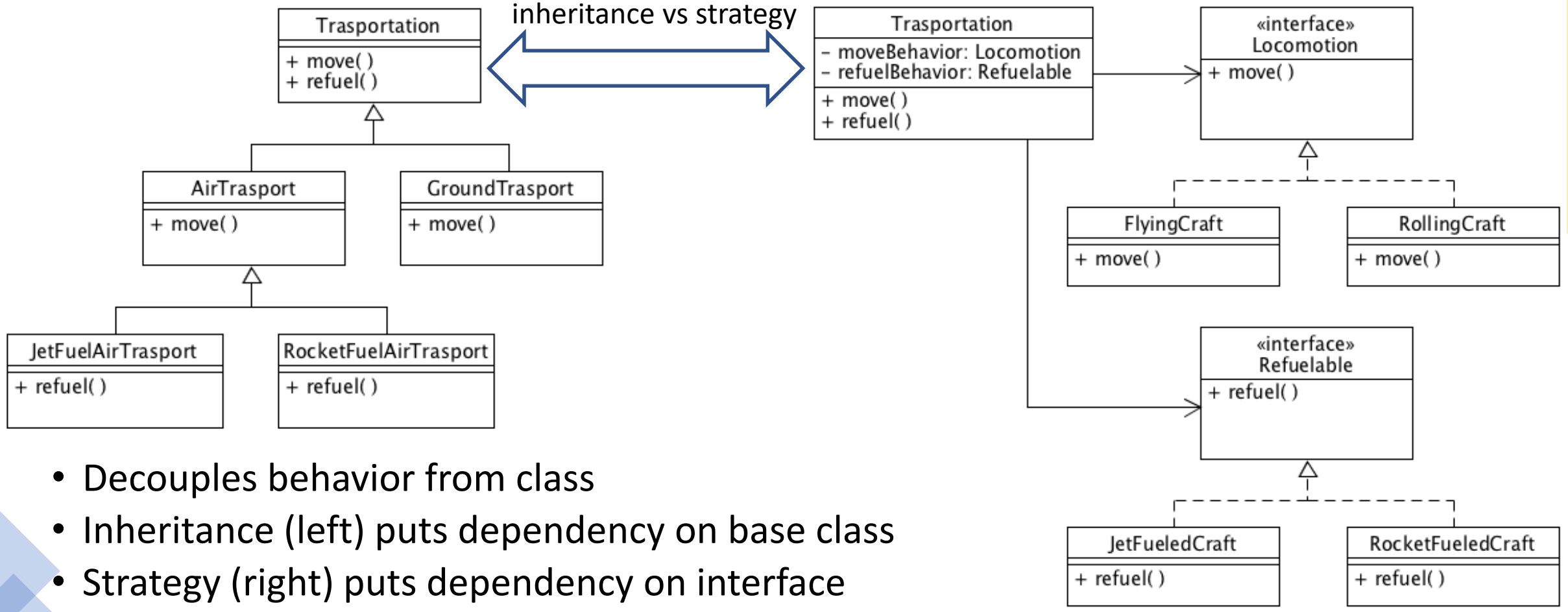
- Problem: You have created a class that finds driving routes between destinations
- Then you receive a requirement to build walking, boating, public transit, etc. routes
- The methods are becoming bloated with else-if
- Solution: Put each variation (strategy) for building a route into a separate class that implements a common interface, navigator can use any object that implements the interface to get the route

Strategy Pattern: Structure



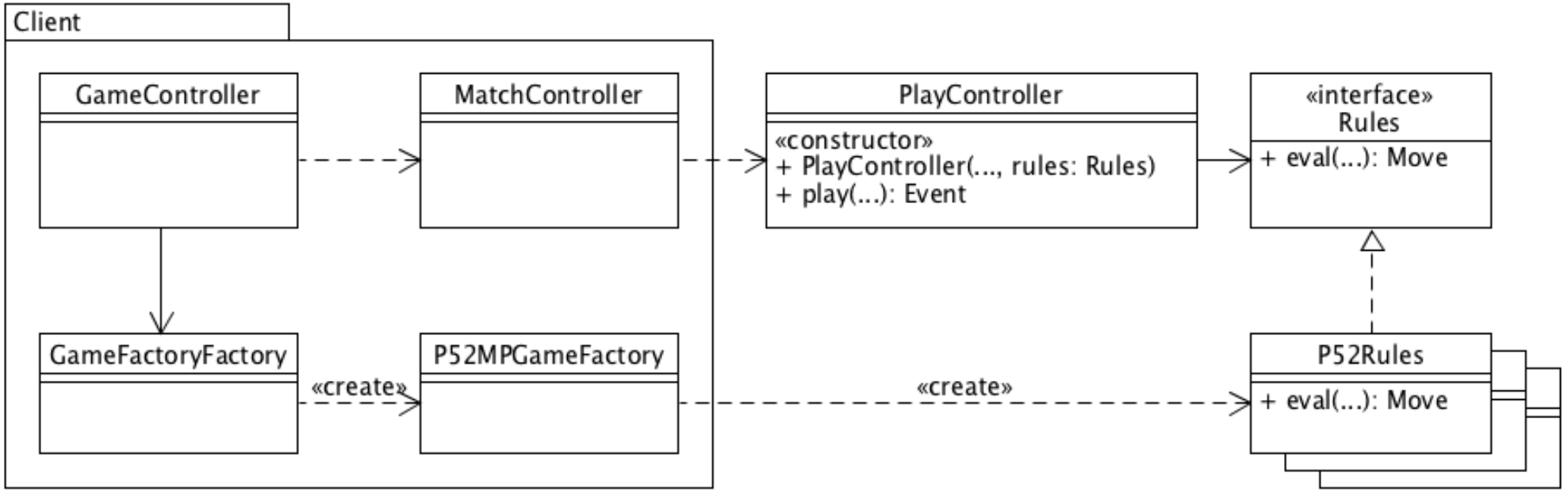
- Strategy: interface common to all supported algorithms
- ConcreteStrategy: implements the algorithm
- Context: maintains a reference to one of ConcreteStrategy and communicates with it via the strategy interface
- Client: creates a ConcreteStrategy and sets it in the Context

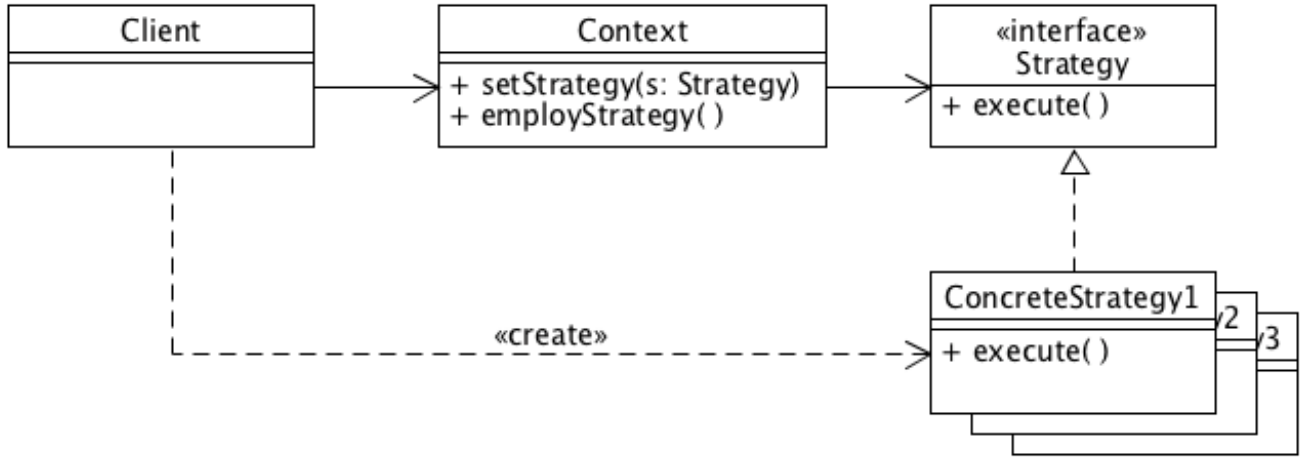
Alternative to Inheritance



- Decouples behavior from class
- Inheritance (left) puts dependency on base class
- Strategy (right) puts dependency on interface
- Multiple independent dimensions
- Compare adding a rocket fueled car

Strategy Pattern in Cards362





Discussion

- Advantages
  - Use different variants of an algorithm and be able to switch algorithm during runtime
  - Replace inheritance, good where there are many dimensions of variation
- Disadvantages
  - When there are only few variations that don’t need to change at runtime, inheritance is a simpler solution
  - Strategy pattern is the OO replacement for Lambda expressions in functional programming, now that Java has Lambda expressions there is a less bloated approach available

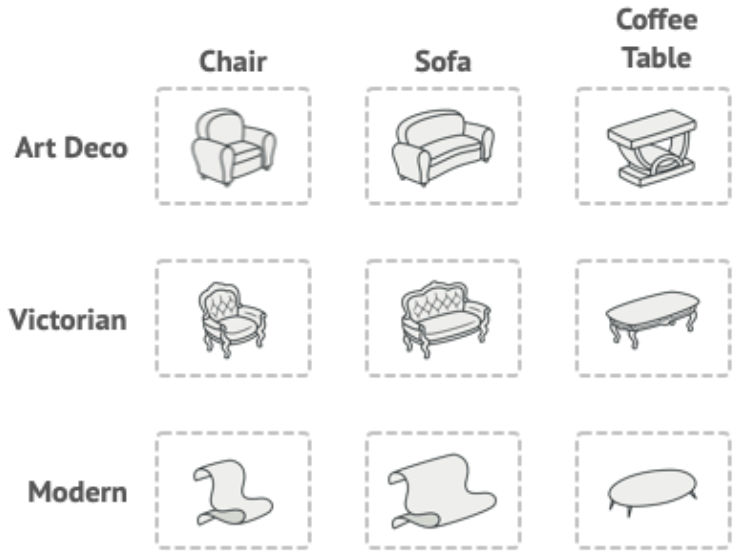
Creational Patterns Overview

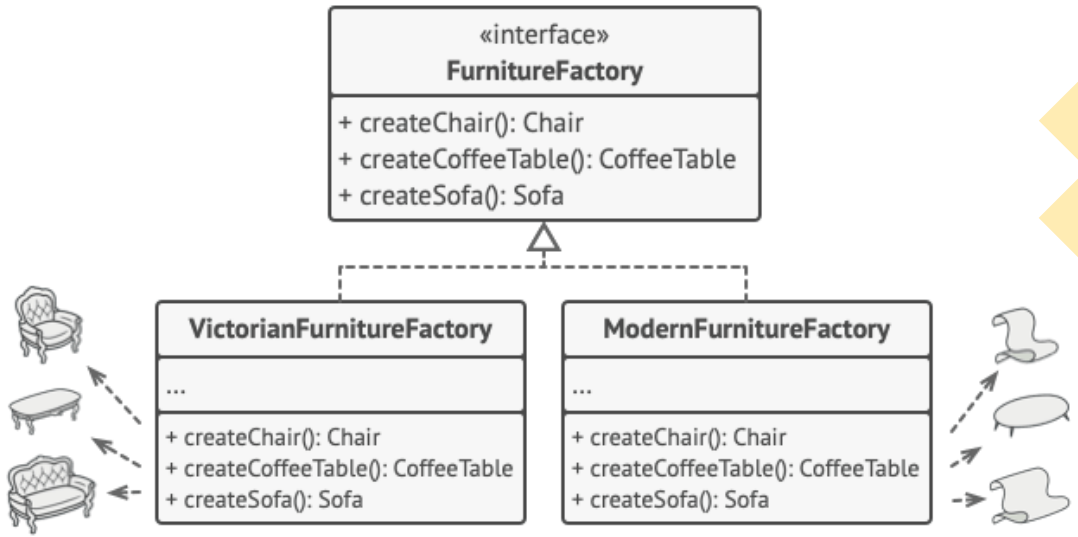
- Factory Method
  - Creates an object in a manner determined by a subclass
- Abstract Factory
  - Creates an instance of several families of classes
- Builder
  - Separates object construction from its representation
- Prototype
  - A fully initialized instance to be copied and cloned
- Singleton
  - A class of which only a single instance can exist

Abstract Factory: Overview

- Intent
  - produce families of related objects without specifying their concrete classes

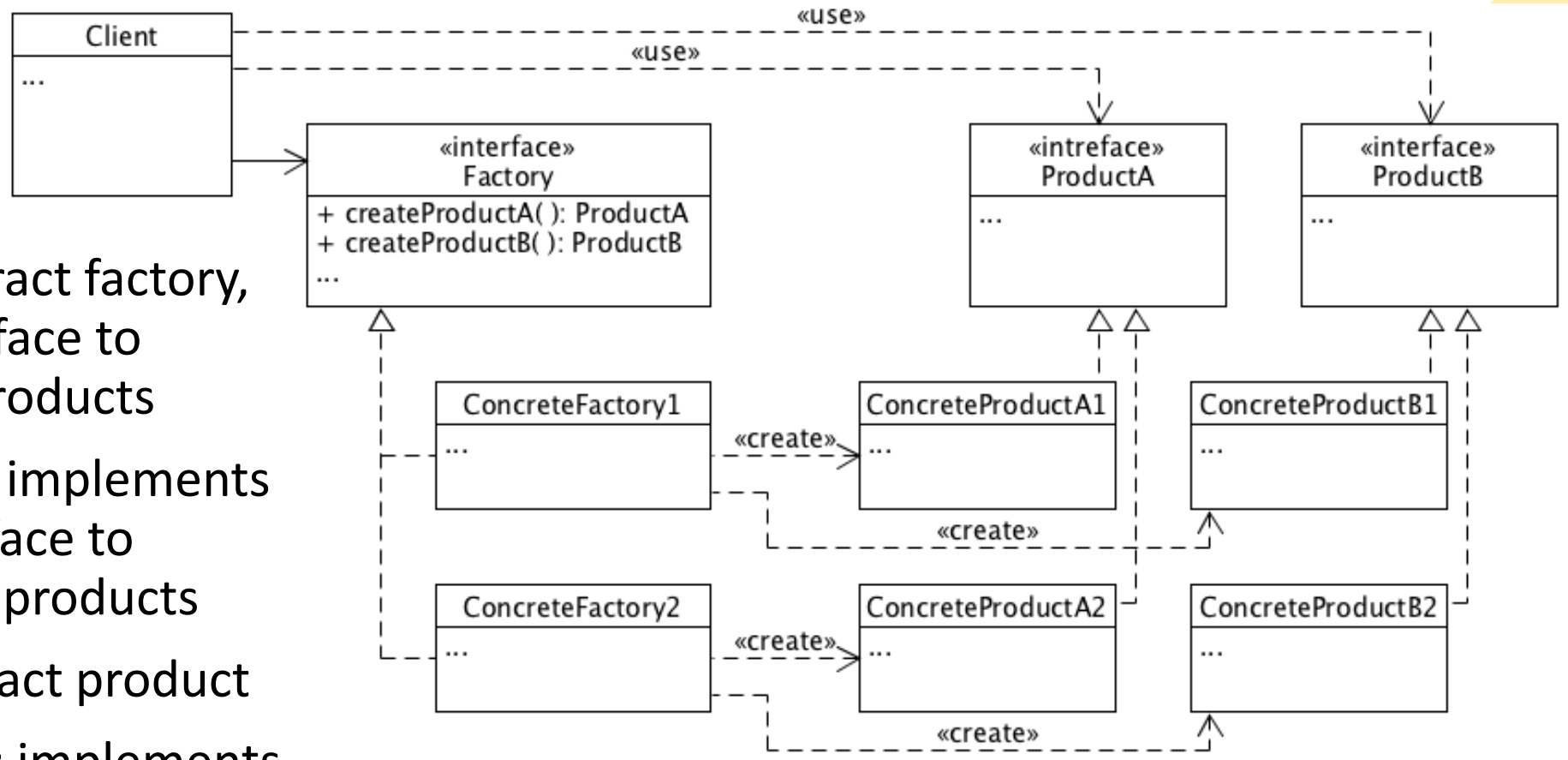
Abstract Factory: Problem/Solution





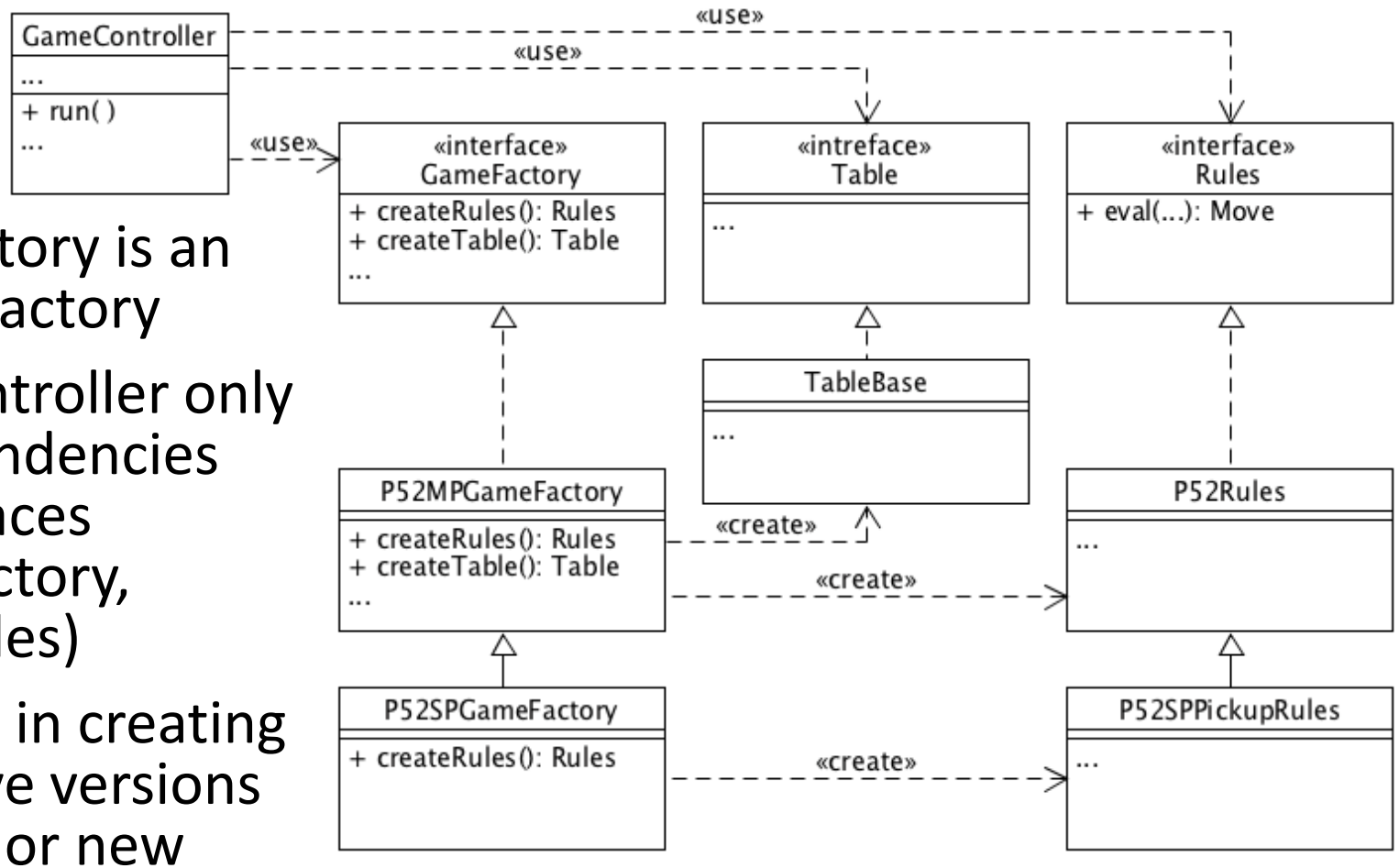
- Problem: You are creating an interior decorating application, you want the Layout class to be able to support different themes (e.g., art deco, Victorian, modern)
- It doesn't seem like a good idea to couple Layout with specific themes
- How to create and use new furniture?
- Solution: Create different factories that produce different families of furniture, the Layout class should only depend on a factory interface and furniture interfaces

Abstract Factory: Structure



- Factory: the abstract factory, provides an interface to create abstract products
- ConcreteFactory: implements the Factory interface to provide concrete products
- Product: an abstract product
- ConcreteProduct: implements the Product interface

Abstract Factory in Cards362



- GameFactory is an abstract factory
- GameController only has dependencies on interfaces (GameFactory, Table, Rules)
- Flexibility in creating alternative versions of games or new games

Discussion

- Advantages
  - Removes coupling between client and the creation of objects
  - Each concrete factory provides a family of consistent/compatible objects
  - Easy to exchange product families
- Disadvantages
  - Introduces many new classes and interfaces to the code, use only when needing to deal with families of objects