COM S 362
Object-Oriented Analysis & Design

Abstract Factory and Strategy Patterns

Reading

Alexander Shvets. Dive Into Design Patterns, 2020.

- Singleton
- Facade
- Decorator

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

Singleton Pattern

Singleton object is static class variable

Private constructor prevents external construction

Create the singleton on demand

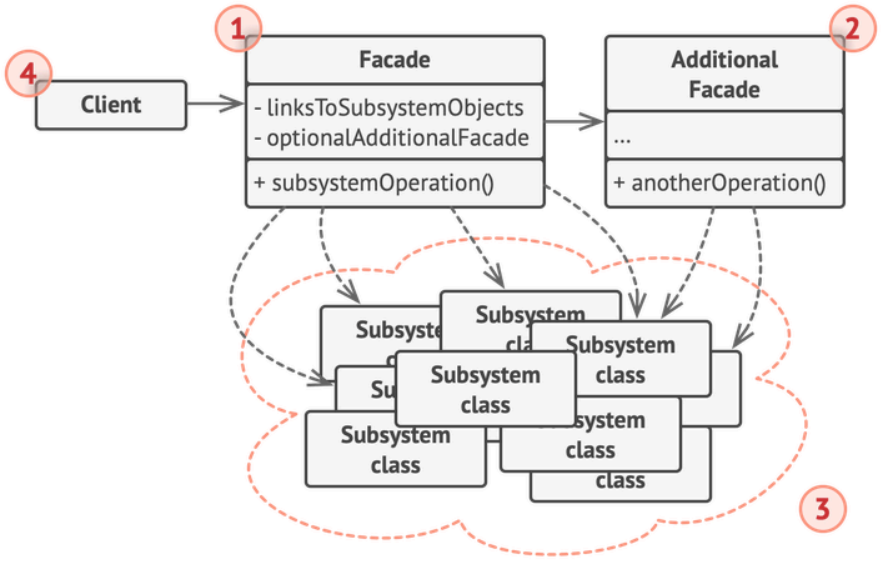
```
public class RemoteTableGateway {

    RemoteEndpoint remote = null;
    Map<String, RemoteEndpoint> epIndex = new HashMap<String, RemoteEndpoint>();

    private static RemoteTableGateway instance = null;

    private RemoteTableGateway(){

    }

    public static synchronized RemoteTableGateway getInstance() {
        if (instance == null){
            instance = new RemoteTableGateway();
            System.out.println("Creating Gateway "+instance.toString());
        }
        return instance;
    }
```

- Ensure that a class has only one instance, while providing a global access point to the instance
- Purpose is to control access to some shared resource (e.g., gateway to remote connections in Cards362)
- Provides global access thorough a static class method

Structural Patterns Overview

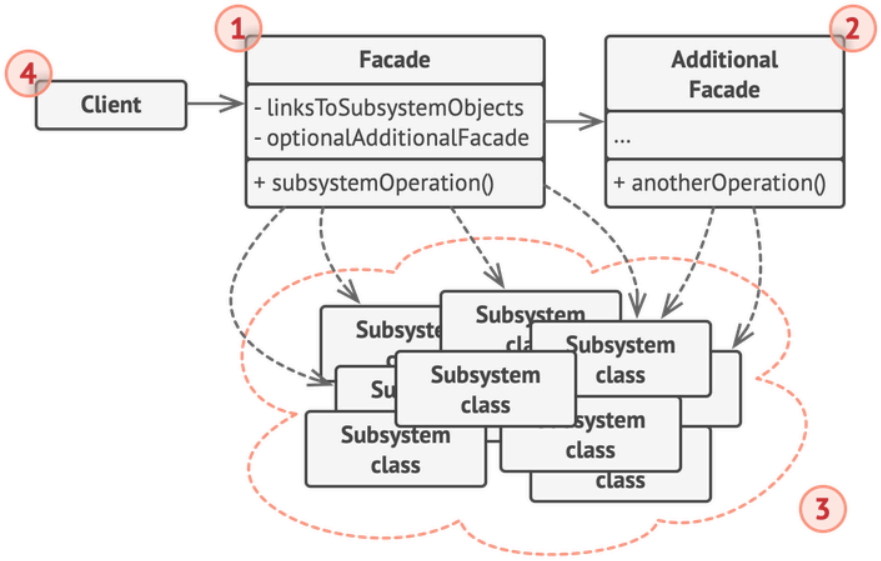
- Adapter
  - Allows objects with incompatible interfaces to collaborate
- Bridge
  - Develop abstraction and implementation separately
- Composite
  - Work with trees of objects as individual objects
- Decorator
  - Attach behaviors to objects by placing them in wrapper objects
- Facade
  - Provide a simplified interface to a library or framework
- Flyweight
  - Fit more objects into limited RAM by sharing common parts
- Proxy
  - Provide a substitute or placeholder for another object

Facade Pattern: Problem/Solution



- Problem: You need to interface with a library or complex set of classes that you don't want your code to be dependent on (coupled)
- Solution: Create a class that provides a simple interface to the complex system

Facade Pattern: Structure



- Façade should only expose methods relevant to the client
- Details of the subsystems are abstracted away
- Facades can be broken up to provide different interfaces or abstractions to different clients

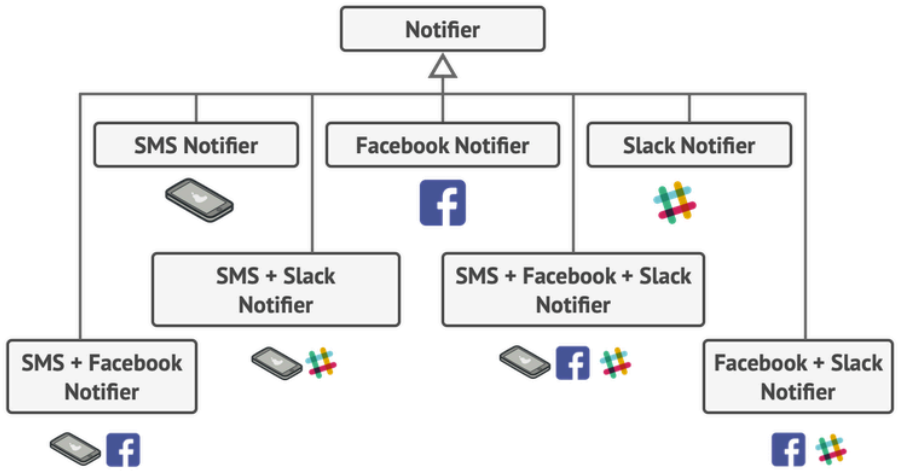
Structural Patterns Overview

- Adapter
  - Allows objects with incompatible interfaces to collaborate
- Bridge
  - Develop abstraction and implementation separately
- Composite
  - Work with trees of objects as individual objects
- Decorator
  - Attach behaviors to objects by placing them in wrapper objects
- Facade
  - Provide a simplified interface to a library or framework
- Flyweight
  - Fit more objects into limited RAM by sharing common parts
- Proxy
  - Provide a substitute or placeholder for another object

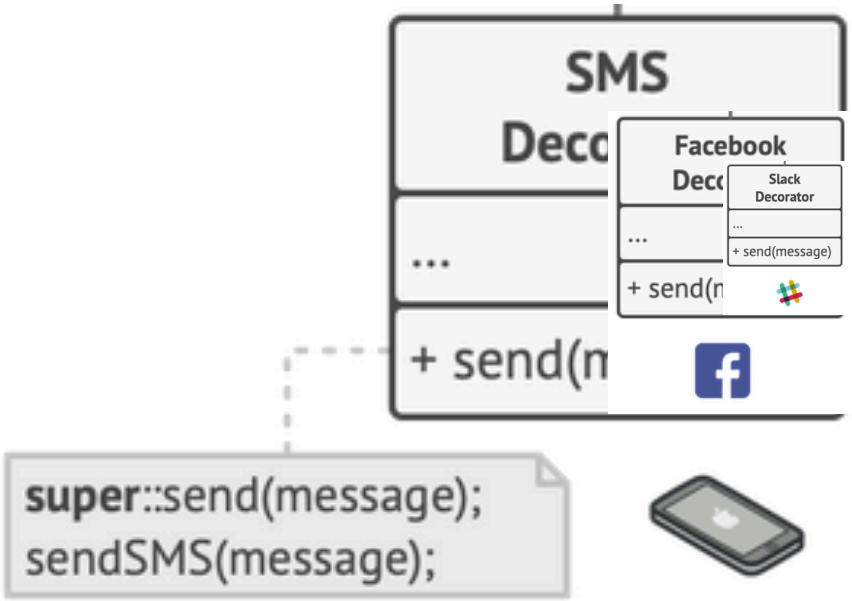
Decorator Pattern: Overview

- Intent
  - Attach behaviors to objects by wrapping them in other objects

Decorator Pattern: Problem/Solution



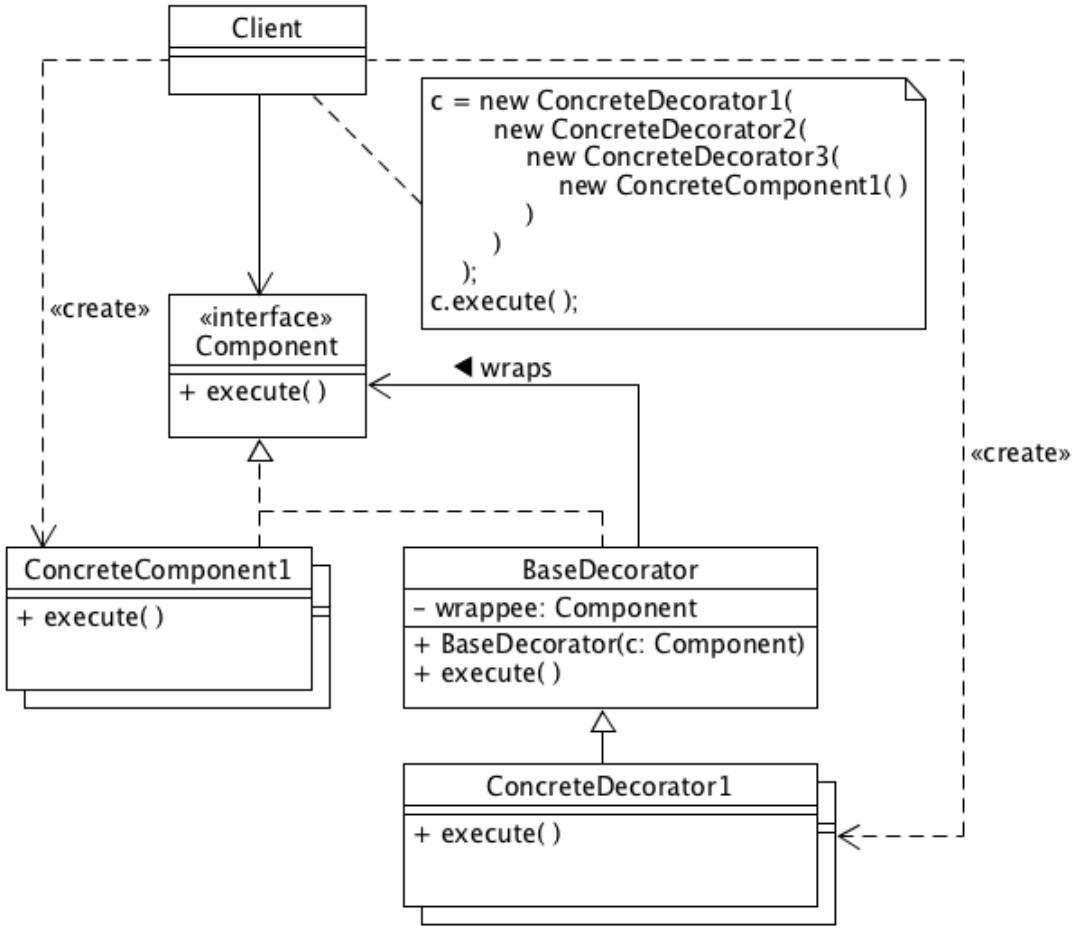
Combinatorial explosion of subclasses.



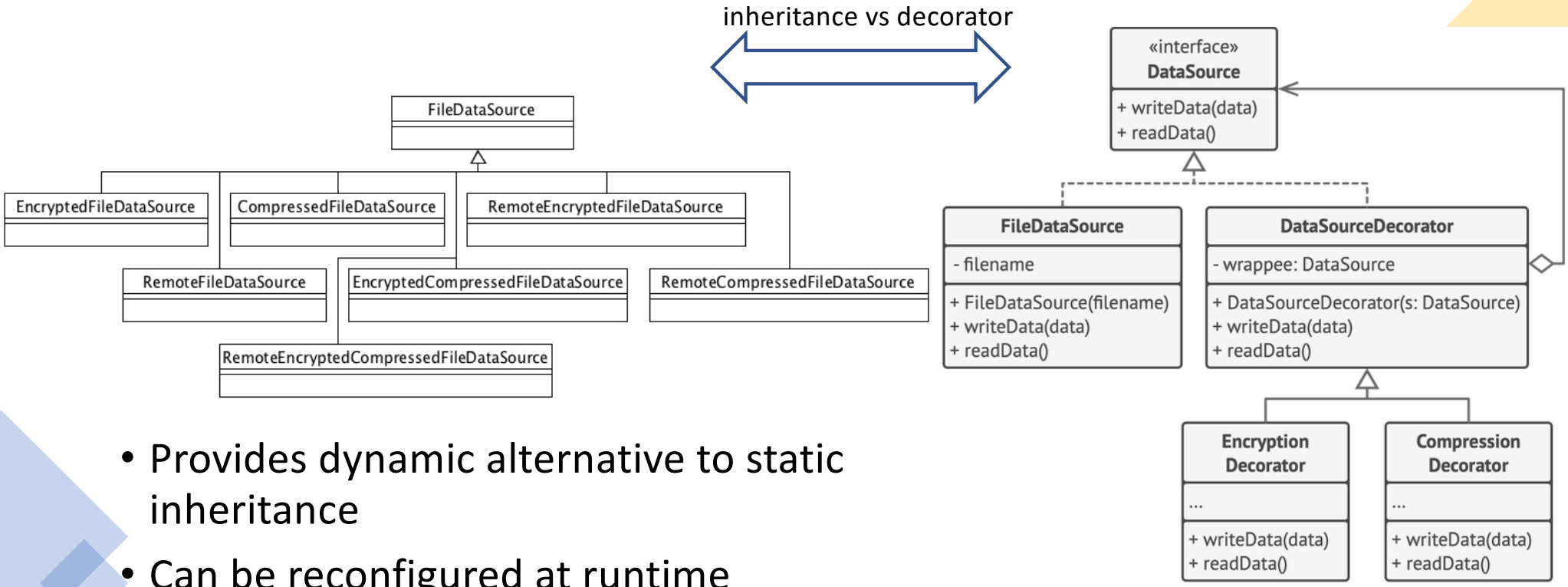
- Problem: You want to create subclasses that take on different combinations of behaviors
- Solution: Put each variation (decorator) into a separate class that wrappers an object of a common superclass, an overridden method call the super class version before performing its own behavior

Decorator Pattern: Structure

- ConcreteComponent: inner most object being wrapped
- Component: the interface seen by the client
- BaseDecorator: an abstract class that defines the common features of all decorators
- ConcreteDecorator: the decorators that can wrap other decorators and a ConcreteComponent



Alternative to Inheritance



- Provides dynamic alternative to static inheritance
- Can be reconfigured at runtime
- Can take on multiple combinations