Chapter 8
Design Patterns (2)

Jean Privat

IT069IU — Object-Oriented Analysis and Design
2015

Overview

Overview

Adapter

Context/problem

- How to resolve incompatible interfaces or provide a stable interface to similar components with different interfaces?

Examples

- Make two components compatible
- Replace a component with another that does not have the same interface

Adapter

Solution

- Convert the original interface of a component into another interface through an intermediate adapter object.

Participants

- Adaptee: provide operations with an unwanted API
- Client: require operations with a wanted API
- Adapter: interface reefing the wanted API
- ConcreteAdapter: wrap the Adaptee and implements the Adapter

Adapter

GRASP

- Use interfaces and polymorphism to add a level of indirection to varying APIs in other components.
- Protected Variations, Low Coupling, Polymorphism, Indirection helps us to view through the myriad details and see the essential alphabet of design techniques being applied.

Tips

- Naming Convention has the advantage of easily communicating to others reading the code or diagrams what design patterns are being used.

Overview

Decorator

Context/Problem

- Add responsibilities to objects instead of a whole class
- Combine responsibilities in a more flexible way than inheritance

Examples

- GUI components
- Java Reader Hierarchy

Decorator

Solution

- Wrappers that encapsulate and extend objects

Participants

- Component: interface of things to decorate
- ConcreteComponent: specific objects to decorate
- Decorator: encapsulate a component and forward operations to it
- ConcreteDecorator: add responsibilities to components

Decorator

GRASP

- Pure Fabrication: each set of responsibilities in its own class
- Polymorphism: common interface for components and specific behavior for decorators
- Indirection: neither the client not the decorators really know the real objects

Overview

Strategy

Context/Problem

- How to design for varying, but related, algorithms or policies?
- How to design with the ability to change these algorithms or policies?

Example

- Application of specific business rules (like sale promotion)

Strategy

Solution

- Define each algorithm/policy/strategy in a separate class, with a common interface

Participants

- Strategy: common interface for all algorithms
- ConcteteStrategy: specific implementation of an algorithm
- Context: the object to which algorithm are applied

Strategy

Collaboration

- A strategy object is attached to a context object: the object to which it applies the algorithm.

Creation

- Who should create the strategy?
- A straight forward approach is to apply the Singleton pattern