Chapter 9
Design Patterns (3)

Jean Privat

IT069IU — Object-Oriented Analysis and Design
2015

Overview

Overview

State

Context/Problem

- How to have the behavior of an object change with its state?
- How to *change class* but keeping the identity
- Doing this avoiding conditional statements

Example

- Adding new capabilities to existing objects (promotion)
- Having a different behavior according to the state of the object (created, open, closed, destroyed)

State

Solution

- Encapsulate the part of the state and its behavior in a sub-object
- The main object delegates to its sub-objects

Participants

- Context: the main object that has a state that dictate its behavior
- State: the abstract state that can act on a context
- ConcreteState: specific behavior for a given state

Overview

Facade

Context/Problem

- A common, unified interface to a disparate set of implementations or interfaces- such as within a subsystem-is required.
- There may be undesirable coupling to many things in the subsystem, or the implementation of the subsystem may change. What to do?

Example

- A session-controller class that interface the UI and the domain layer

Facade

Solution

- Define a single point of contact to the subsystem: a facade object that wraps the subsystem.
- This facade object presents a single unified interface and is responsible for collaboration with the subsystem components.

Participants

- Client: communicate with the facade
- Facade: hide the complex system to the client

Facade

Discussion

- A Facade is a front-end object that is the single point of entry for the services of a subsystem; the implementation and other components of the subsystem are private and can't be seen by external components.
- Facade provides Protected Variations from changes in the implementation of a subsystem.
- The subsystem hidden by the facade object could contain dozens or hundreds of classes of objects, or even a non-object-oriented solution, yet as a client to the subsystem one can see only its one public access point.

Overview

Observer

Context/Problem

- Different kinds of objects are interested in the state changes or events of a publisher object, and want to react in their own unique way when the publisher generates an event.
- Moreover, the publisher wants to maintain low coupling to the subscribers.

Example

- Update on the domain objects should be reflected on various views in a user interface

Observer

Solution

- Objects send notifications of event to interested objects
- Objects register themselves anonymously to notification of event

Participants

- Subject: interface that accepts observers can notify them
- ConcreteSubject: implement Subject
- Observer: interface that can receive update
- ConcreteObserver: objects that observe and are updated

Observer

Discussion

- Observer is not only for connecting UIs and Model Objects but also used for GUI widget event handling in both Java technology and Microsoft's .Net.
- One publisher can have many subscribers for an event.
- Observer provides a way to loosely coupled objects in terms of communication.
- Publishers know about subscribers only through an interface, and subscribers can register dynamically with the publisher.

Overview

Proxy

Context/Problem

- Control the access to an object

Example

- Virtual Proxy: Access to complex objects with lazy initialization
- Remote Proxy: Access to object outside the address space
- Protector Proxy: Control the access to the object
- Smart Proxy: Make additional operation on access

Proxy

Solution

- Create a proxy object that encapsulate a real object

Participants

- Subject: common interface that offers common services
- RealSubject: the real object, not accessed directly by the client
- Proxy: encapsulate the real subject and forward operation