Facade Pattern

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

Outcomes

After today's lecture you will be able to:

- Understand the use of the Facade Design Pattern
- Use and implement the Facade Pattern

Inspiration

“Just because you don’t know a technology, doesn’t mean you won’t be called upon to work with it.” – Mike Bongiovanni

Yet Another Adapter: Facade Pattern

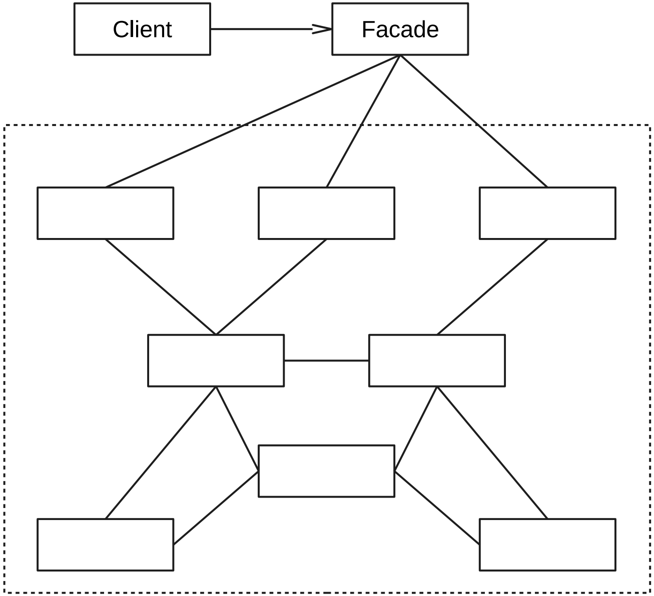
- There is another way in which an adapter can be used between a client an an adaptee: to simplify the interface of the adaptee(s)
- Imagine a library of classes with a complex interface and/or complex interrelationships
  - Book's Example: Home Theater System
    - Amplifier, DVDPlayer, Projector, CDPlayer, Tuner, Screen, PopcornPopper, and TheatreLights
    - Each with its own interface and interclass dependencies
  - Imagine steps for "watch movie"
    - turn on popper, make popcorn, dim lights, screen down, projector on, set projector to DVD, amplifier on, set amplifier to DVD, DVD on, etc.
  - Now imagine resetting everything after the movie is done, or configuring the system to play a CD, or play a video game, etc.

Facade Pattern: Definition

- The Facade Pattern provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
  - We place high level methods like “watch movie”, “reset system”, “play CD” in a facade object and encode all of the steps for each high level service in the facade.
  - Client code is simplified and the client’s dependencies are greatly reduced
    - A facade not only simplifies an interface, it decouples a client from a subsystem of components
- Relationship to Adapter Pattern?
  - Both facades and adapters may wrap multiple classes, but a facade’s intent is to simplify, while an adapter’s is to convert between interfaces

Idaho State University Computer Science

Facade Pattern: Structure



Demonstration

ROAR

Idaho State University
Computer Science

New Design Principle

- The facade pattern demonstrates a new design principle
- Principle of Least Knowledge: “Talk only to your immediate friends”
  - reminds you to create loosely coupled systems of cohesive objects
  - also known as “The Law of Demeter”
- We want to reduce an object’s class dependencies to the bare minimum
- How many classes is the code coupled to?

```java
public float getTemp() {
  return station.getThermometer().getTemperature();
}
```

ROAR

Principle of Least Knowledge: Heuristics

- In order to implement the principle of least knowledge, follow these guidelines
- For any object within any method of that object
  - you may invoke methods that belong to
    - the object itself
    - objects passed in as a parameter to the method
    - any object the method creates or instantiates
    - any object that is stored as an instance variable of the host object
- The code on the previous slide violates these guidelines because we invoke the method `getTemperature()` on a “friend of a friend”
  - Change code to `return station.getTemperature()` to follow guidelines
  - Requires adding “wrapper” method to station class

“Legal” Method Invocations

```java
public class Car {

  private Engine engine;

  public Car() {}

  public void start(Key key) {
    Door doors = new Doors();
    boolean authorized = key.turns(); // obj passed as param

    if (authorized) {
      engine.start(); // component method
      updateDashboardDisplay(); // local method
      doors.lock(); // object created by method
    }
  }

  public void updateDashboardDisplay() { }
}
```

Wrapping Up

- Facade is a variant of the adapter pattern in which the purpose is to (greatly) simplify the adaptee’s interface
- Facade demonstrates the use of a new design principle, the Principle of Least Knowledge, also known as the Law of Demeter
  - often phrased “talk only to your friends”
  - focus is on reducing coupling between classes

Are there any questions?