УНИВЕРСИТЕТ ИТМО

Архитектура программных систем

Лабораторная работа № 2

Выполнили:

Хуан Сыюань P33101

Преподаватель:

Перл Иван Андреевич

Санкт-Петербург, 2021

Задание

Из списка шаблонов проектирования GoF и GRASP выбрать 3-4 шаблона и для каждого из них придумать 2-3 сценария, для решения которых могу применены выбранные шаблоны.

Сделать предположение о возможных ограничениях, к которым можем привести использование шаблона в каждом описанном случае. Обязательно выбрать шаблоны из обоих списков.

Adapter, GOF

Adapter is a structural design pattern that allows objects with incompatible interfaces to collaborate. It allows the interface of an existing class to be used as another interface. It is often used to make existing classes work with others without modifying their source code.

Weak point:

Cannot override methods in Adaptee . Cannot reuse Adapter with subclasses of Target . Adapter and Adaptee are different objects. The overall complexity of the code increases because you need to introduce a set of new interfaces and classes. Sometimes it's simpler just to change the service class so that it matches the rest of your code.

Scenarios:

1. We're creating a stock market monitoring app. The app downloads the stock data from multiple sources in XML format and then displays charts and diagrams for the users.Sometimes we decide to improve the app by integrating a smart 3rd-party analytics library. But the analytics library only works with data in JSON format. We could change the library to work with XML. However, this might break some existing code that relies on the library. We can create an adapter. This is a special object that converts the interface of one object so that another object can understand it.
2. When you travel from the China to Russia for the first time, you may get a surprise when trying to charge your laptop. The power plug and sockets standards are different in different countries. That's why your US plug won't fit a German socket. The problem can be solved by using a power plug adapter that has the American-style socket and the European-style plug.

Builder, GOF

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.

Weak point:

Although we can construct objects step-by-step, defer construction steps or run steps recursively but the overall complexity of the code increases since the pattern requires creating multiple new classes.

Scenarios:

1. We have a complex object that requires laborious, step-by-step initialization of many fields and nested objects. Such initialization code is usually buried inside a monstrous constructor with lots of parameters. Or they are scattered all over the client code. Like we want to create a House object. To build a simple house, we should construct four walls and a floor, install a door, fit a pair of windows, and build a roof. But it can't work if we want a bigger house, with a backyard and other goodies (like a heating system, plumbing, and electrical wiring). The Builder pattern let us extract the object construction code out of its own class and move it to separate objects called builders. The pattern divides object construction into steps (buildWalls, buildDoor, etc.). A series of these steps are performed on a builder object to create an object. The important thing to remember is that you do not have to call all of the steps. Only the steps required to produce a specific configuration of an object can be called.
2. When the process of creating a new object should not depend on what parts this object consists of and how these parts are related to each other. For example, a class must acquire additional functionality over the course of its life. The builder can add "modules" with new functionality to an existing object. In some languages this is solved using prototypal inheritance or extension functions:

```
fun main() {

   fun String.praiseVT() = this.plus("Programming forever!!!")

   " now all lines can be like this: ".praiseVT() // now all lines can be like this: Programming forever

}
```

Factory Method, GoF

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

Weak point:

The code may become more complicated since you need to introduce a lot of new subclasses to implement the pattern. The best case scenario is when you're introducing the pattern into an existing hierarchy of creator classes.

Scenarios:

1. We're creating a logistics management application. The first version can only handle transportation by trucks and the bulk of code lives inside the Truck class. At present, most of your code is coupled to the Truck class. Adding Ships into the app would require making changes to the entire codebase. Moreover, if later we decide to add another type of transportation to the app, you will probably need to make all of these changes again.The code will be riddled with conditionals that switch the app's behavior depending on the class of transportation objects. We can replace direct object construction calls (using the new operator) with calls to a special factory method.
2. If we want to produce Xiaomi and iPhone, we should create an interface(AbtractFactory). And create other two classes to implement it. Here are codes.

```
public interface AbstractFactory {
    Phone makePhone();
}
public class XiaoMiFactory implements AbstractFactory{
    @Override
    public Phone makePhone() {
        return new MiPhone();
    }
}
public class AppleFactory implements AbstractFactory {
    @Override
    public Phone makePhone() {
        return new IPhone();
    }
}
public class Demo {
    public static void main(String[] arg) {
        AbstractFactory miFactory = new XiaoMiFactory();
        AbstractFactory appleFactory = new AppleFactory();
        miFactory.makePhone();            // make xiaomi phone!
        appleFactory.makePhone();        // make iphone!
    }
}
```

Pure fabrication, GRASP

A pure fabrication is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived (when a solution presented by the information expert pattern does not). This kind of class is called a "service" in domain-driven design.

Weak point:

It is difficult to fit into the topic area. Because of the need for highly abstract code, it is difficult for beginners to implement such codes. (In process-oriented languages, there is no such problem).

Scenarios:

1. When we do not want to violate High Cohesion and Low Coupling but solutions offered by Expert (for example) are not appropriate. We should find out the object

which should have the responsibility. Object - oriented designs are sometimes characterized by implementing as software classes representations of concepts in the real - world problem domain to lower the representational gap; for example a Sale and Customer class. However, sometimes assigning responsibilities only to domain layer software classes leads to problems in terms of poor cohesion or coupling, or low reuse potential. Assign a highly cohesive set of responsibilities to an artificial or convenience class that does not represent a problem domain concept - something made up, to support high cohesion, low coupling, and reuse.

2. If we want to implement the Data Transfer Object pattern used in the backend, we should transform "raw" data from the client into a usable model. Pure fabrication is the best solution.

Conclusion

Design patterns are very critical basic knowledge. It is different from the algorithm, it can shape the prototype of the program. Often such code will have a clearer structure, and use simple and clear code to achieve very complex functions. I also learned more about the difference between GoF and GRASP. GRASP is the premise of the GoF design pattern. The GoF design pattern is an object-oriented design pattern that meets the requirements of the GRASP pattern principle. GRASP is a route planning scheme, and GoF is a concrete and implementable route formulated according to the plan.