Design Pattern Presentation

CodeTrip

March 8, 2018

CS 446 - Team CodeTrip
Jiajun Zhou
20522165
j86zhou@edu.uwaterloo.ca

Tianrui Ma
20550222
t33ma@edu.uwaterloo.ca

Jiahua Zhu
20474723
j66zhu@edu.uwaterloo.ca

Haolin Wu
20572042
h97wu@edu.uwaterloo.ca

Purpose and motivation

The Observer pattern addresses the following problems:

- Require one-to-many relationship between objects which are loose coupling.
- As one object changes state, other dependent objects need to be automatically updated.

Tightly coupled objects are difficult in implementing, modifying, testing, and reusing. In the case of building up a one-to-many dependency between objects, if we allow one object to directly update the dependent objects, we would end up with a tightly coupled structure since the update methods may vary from one to another depending on the type of the object. Thus, we need to find another way to solve this problem.

Intended use case

The Observer pattern defines Subject and Observer in a way that whenever the critical field is updated, the Subject will be notifying all registered Observers the updates.

Use the Observer pattern in any of the following situations:

- When an abstraction has two aspects, one dependent on the other. Encapsulating these aspects in separate objects lets you vary and reuse them independently.
- When a change to one object requires changing others, and you don't know how many objects need to be changed.
- When an object should be able to notify other objects without making assumptions about who these objects are.

Vocabulary

Subject:

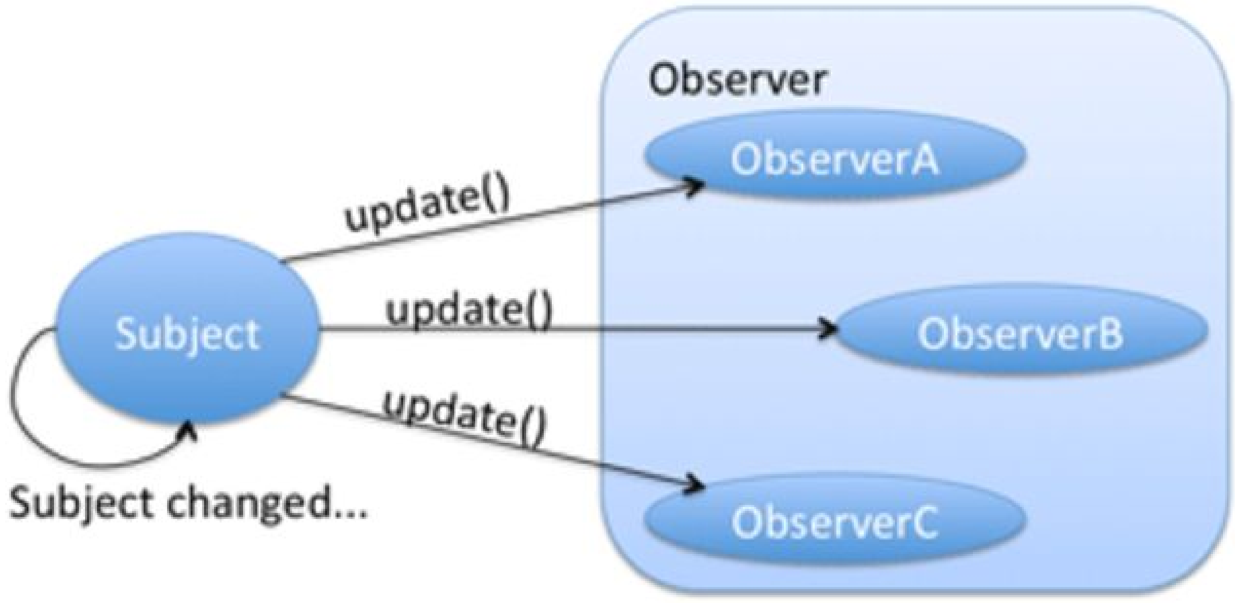
Subject have knowledges about it’s observers and provide the interface of register and unregister it’s observers.

Observer:

Defined an updating interface for subject to use when observer need to be notified.

Specific structure and runtime behaviour

In the Observer architecture, the subject does not update the state of dependent objects directly. By implementing the “update()” interface, the subject is independent of how the state of dependent objects is been updated. As an observer first register with the subject, it will be store in the register list in the subject. As the state of subject been updated, it will notify observers in the register list and the state of observers will be updated.



Positive and negative consequences

Positive consequences:

1. **Related classes are loosely coupled.** All a subject knows is that it has a list of observers, each conforming to the simple interface of the abstract Observer class. The subject doesn't know the concrete class of any observer.

2. **Allow broadcast communication.** The subject doesn't care how many interested objects exist; The notification is broadcast automatically to all interested objects that observes it.

3. **Adapt changes quickly.** Observer pattern gives you the freedom to add and remove observers at any time. It's up to the observer to handle or ignore a notification.

Negative consequences:

1. **Unexpected expensive updates.** A simple update operation on the subject may cause a cascade of updates to observers and their dependent objects.
2. **Add unnecessary complexity.** Observer pattern may add unnecessary complexity to the code because observers have no knowledge of each other's presence.
3. **Undependable notifications.** The order of Observer notifications is undependable in some language’s built-in library, they use set instead of looping through an array.
4. **May cause memory leak.** The leak happens when an observer fails to unsubscribe from the subject when it no longer needs to listen.

Improve and degrade NFPs

Improved NFPs:

- **Dependability**: Making classes between each other low coupling
- **Scalability**: Easy to scale as the number of observers increase
- **Security**: Each class will be able to apply encapsulation, thus improves security.
- **Usability**: Observer pattern makes code easier to learn and use.
- **Efficiency**: Better efficiency when the subject is very complex

Degrade NFPs:

- **Complexity**: Makes code more complex as the subject because more complex

Presentation Script

Purpose and motivation

Jasmine: As software developers, we have probably all encountered situations where some objects need to be informed frequently of changes in state of some other objects. For example, followers of a twitter account should receive notifications whenever a new tweet from that account is published.

Because we are experts in software design and architecture, we know that these objects should be as loosely coupled as possible to reduce dependency. We would also like to be able to freely add or remove an observer object at runtime.

This is where the Observer pattern comes in handy. The observer pattern is a one-to-many design pattern where one object called “the subject” is followed by many observers.

The Subject maintains a list of observers and it typically has register and unregister methods to allow observers add/remove themselves from the list. When a change in state occurs in the Subject, it may call its notify method to broadcast this change to all registered observers.

Since observers could be of arbitrary types, they have the freedom to handle this update in whatever way they want.

Now consider the real life example of a stock market, there are millions of stockholders wishing to obtain first hand price updates on a certain stock. How do they find out about this information without the use of observer pattern?

Stock market example (without observer pattern)

Jay: I’m the stock exchange market and have the information about the apple stock which is now at 5 dollar per share.

Anthony: I am an stockholder of the Apple company.What is the stock price for the Apple now?

Tianrui Ma: I am also a stockholder of Apple Company, and I need to know every update of the price from the stock exchange. What is the stock price for the Apple now?

Jay: It’s now at 5 dollar per share

Jay: It’s still at 5 dollar per share

Jasmine: Now, let's see how would stock market operate using the observer pattern

Stock market example (with observer pattern)

Jay: I’m the Subject which is a stock exchange market. I will be give out information about the apple stock.

Anthony: I am an observer of the stock market and I would like to get a notification when the stock price of the Apple changes because I hold 3000 share of the Apple stock. Please put me on your register list.

Jay: Now I will put Observer1 onto my observer list and notify him when the stock price is changed. Oh, the stock price for apple just raised to 10 dollar. I need to notify my observer. (Face to observe 1)The apple stock is now 5 dollar per share.

Anthony: Oh the stock price raised! Come on! Keep going!

Tianrui Ma: I am an observer of the stock market and I just bought in 3000 share of stock from Apple. Now please put me on your register list and let me know any update of the price.

Jay: Ok, I will now add Observer 2 onto my observer list and notify them when the stock price is changed. Oh, the stock price for apple just dropped to 1 dollar. I need to notify all my observer. Hey, Observer 1, the apple stock is now worth 1 dollar per share.

Anthony: What! I lost all my money! Remove me from your register list. I don’t want to see any further notifications. I quit!

Jay: And I will remove observer 1 from my list. Hey, Observer 2, the apple stock is now worth 1 dollar per share.

Tianrui Ma: That is bad but I think I’m gonna wait for a little bit longer and see.

Jay: Now, the stock price for apple just rise from 50 dollar. I need to notify my observer and I check my list and find out I only need to notify Observer 2. Hey, Observer 2, the apple stock is now worth 1 dollar per share.

Tianrui Ma: Yeah

Jay: Now, anthony and catrina will talk about the positive and negative consequences of using observer pattern.

Positive and negative consequences

Anthony: Because the subject does not directly communicate to the observers, therefore we do not have to define a specific update method for each observer and it is loosely coupled between related objects. When the subject notify observers, it doesn't care how many interested objects exist; The notification is broadcast automatically to all interested objects that observes it.

Catrina: Since observer are stateless, we often need several of them to simulate a state machine. Without careful implementation, this could cause many kinds of problems such as infinite loop and memory leak problem that are hard to trace down.
Also the order of Observer notification is undependable in some built-in library. This may cause unpredictable behavior.

Catrina: Now Jay will talk about the NFP about observer pattern.

Improve and degrade NFPs

Jay: Observer Pattern improve code's Scalability because it's gives you the freedom to add and remove observers at any time, and subject would not be affected by the addition of observers. However, Observer Pattern also would increase code's complexity, as events and interactions among subject become more complex, observers will becomes increasingly difficult to implement.