ISEP
Algorithmics and Advanced programming
March 2020

Tutorial Course 3

Design patterns

I. Strategy Pattern

Goal

Assume that we have an online application which provides multiple ways of authentication to its users (Basic, LDAP, OpenID, OAuth …). Our goal is to implement the logic which will allow our program to easily switch the authentication strategy

---

The authentication itself is not important here. We will assume that the site allows 2 different authentication method : Basic and OAuth.

Intuitive algorithm

a. Create 2 concrete classes BasicAuthentication and OAuthAuthentication which all just have :
   i. A public void authenticate() method that only prints the authentication method name.

just one method

b. Now create an AuthProgram class which will orchestrate our authentication process.
   i. It receives a String "authenticationStrategy" as a constructor parameter, and sets its strategy (using an encapsulated property).
   ii. It has a public void authenticate(String username, String password) method which invoke the correct authentication strategy.
c. Test your implementation.
d. How do we add an authentication strategy to the current algorithm? Do it with a DigestAuthentication class. Modify the AuthProgram accordingly.
   i. Does this respect the Opened/Closed principle of SOLID principles?

2. **Strategy pattern**

Let's now modify our first implementation in order to use the strategy pattern.

a. Create an interface called IAuthenticationStrategy with an public void authenticate() method.
b. Make all your current strategies implement this interface.
c. Refactor your AuthProgram class in order to use the IAuthenticationStrategy interface.
d. Test your implementation.
e. How would you add an authentication strategy to the current program? Do it with an OpenIDAuthentication strategy.
   i. Does this respect the Opened/closed principle of SOLID principles?
f. What are the advantages of the strategy pattern you just used versus the previous intuitive approach?

the advantage of using strategy pattern is the interface that they have in comon
easy to implement and modify but not caused a sideeffect on program since following the SOLID principles.

II. Observer pattern

Goal

We want to represent a data centralization system to which many media/newspaper subscribe. Every time this system updates its data, all the subscribers get notified with the update.

We also want to be able to add as many subscribers as we want in the simplest manner.
The specification of such a system are.

The publisher should :
- Provide a way for the subscribers to subscribe/unsubscribe
- Be able to notify its subscribers

The Subscriber (observer) should:
- Be able to be updated with every new message coming from the Publisher

---

For this example, we will keep the main subject (which updates the data) as simple as possible. What we want to modelize here is just the update of its state and the notification sent to the subscribers.

1. How would you create such a system? Describe the main classes and their interactions in order to achieve such a goal, through a UML diagram (or any simple schema you find suitable) without coding it.
2. Let's now do it using the observer pattern.
    a. Create an interface IPublisher with the following methods:
        i. public void attach(Observer o); ➔ method to subscribe to the subject
        ii. public void detach(Observer o); ➔ method to unsubscribe to the subject
        iii. public void notifyUpdate(Message m); ➔ method to notify subscribers with the update
    b. Create a concrete "MessagePublisher"
        i. Add a list of encapsulated observers to this class as an ArrayList
        ii. Make it implement the IPublisher interface.
    c. Create an IObserver interface with the method:
        - public void update(Message message);
    d. Create Two concrete Observers with the names of any newspaper you want.
        - Make them implement the IObserver interface and simply display a message stating that they received the updated message, with its content and their names.
    e. Try and add as many new subscribers you want.
    f. Test your code.