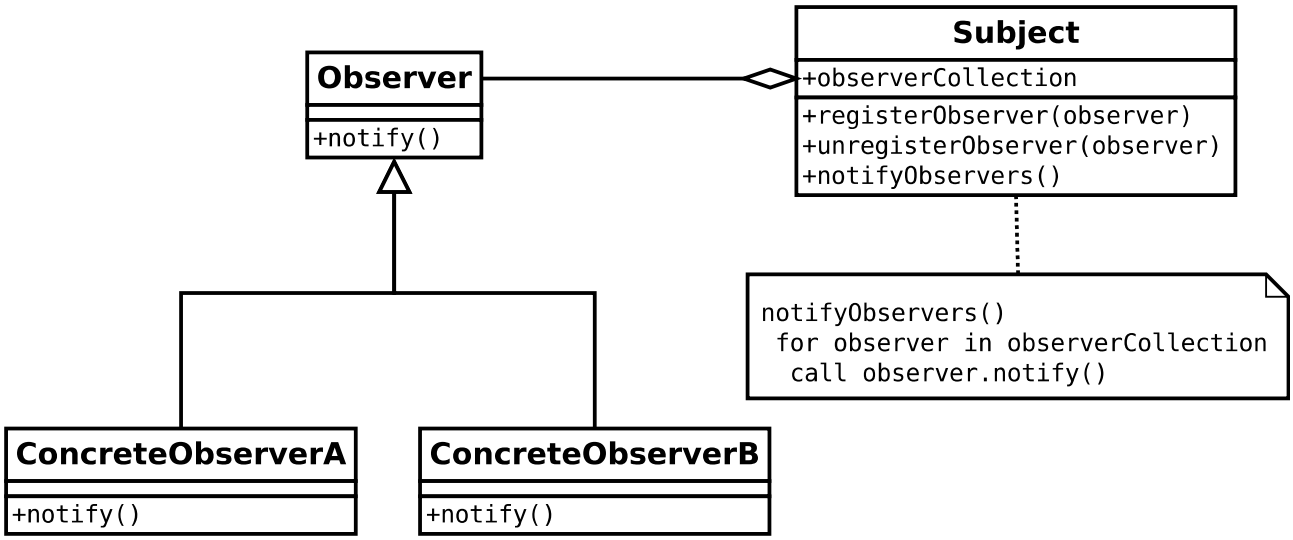
Implementacja wzorca projektowego obserwator w różnych językach programowania

Robert Cebula

Table of Contents

1. Opis wzorca

Wzorzec obserwatora jest wzorcem w którym jeden obiekt zwany tematem (ang. Subject) informuje inne zainteresowane nim obiekty zwane obserwatorami (ang. Observer) o zmianie stanu pewnego innego obiektu.



Rys.1 Schemat wzorca w przypadku obiektowego języka programowania (źródło: wikipedia.org)

2. Java

W javie do implementacji wzorca obserwatora można posłużyć się klasami z biblioteki standardowej: java.util.Observer i java.util.Observable. Żeby stworzyć temat dziedziczymy po klasie Observable. Posiada ona między innymi dwie metody: setChanged() i notifyObservers(). Pierwsza oznacza obiekt jako zmieniony, a druga informuje wszystkich obserwatorów o zmianie. Następnie dziedziczymy po klasie Observer, która posiada jedną metodę update() wywoływaną w momencie informowania obserwatora przez temat (metoda notifyObservers).

```
import java.util.Observable;
import java.util.Scanner;

class EventSource extends Observable implements Runnable {
    public void run() {
        while (true) {
            String response = new Scanner(System.in).next();
            setChanged();
            notifyObservers(response);
        }
    }
}
```

```
import java.util.Observable;
import static java.lang.System.out;

class MyApp {
    public static void main(String[] args) {
        out.println("Enter Text >");
        EventSource eventSource = new EventSource();

        eventSource.addObserver( (Observable obj, Object arg) -> {
            out.println("\nReceived response: " + arg);
        });

        new Thread(eventSource).start();
    }
}
```

Rys.2 Przykład implementacji w javie (źródło: wikipedia.org). Zamiast dziedziczenia po klasie Observer została użyta funkcja lambda

3. C#

W języku C# możemy użyć podobnego rozwiązania jak w javie (zasadniczo dla każdego języka obiektowego można tak zrobić), ale poza tym możemy zastąpić klasę Observable i Observator zdarzeniami. Zdarzenie w języku C# (ang. Event) to coś podobnego do wskaźnika do funkcji z języka C/C++, czyli obiekt przechowujący referencję do funkcji. Typy przyjmowanych i zwracanego argumentu określa delegacja (ang. Delegate). Najpierw tworzymy delegację, która opisuje nam sygnaturę przyjmowanych funkcji. Następnie tworzymy zdarzenie, które będzie naszym tematem. Dodajemy do zdarzenia funkcję będące obserwatorami obserwatorów. Od teraz w momencie wywołania zdarzenia, wywołane zostaną wszystkie przypisane do niego funkcje.

```
1  namespace SimpleEvent
2  {
3      public delegate void someEventDelegate(string param);
4      public class EventTest
5      {
6          private event someEventDelegate someEvent;
7
8          public void addObserver(someEventDelegate observer)
9          {
10             someEvent += observer;
11         }
12
13         public void removeObserver(someEventDelegate observer)
14         {
15             someEvent -= observer;
16         }
17
18         public void notifyObservers(string str)
19         {
20             if (someEvent != null)
21             {
22                 someEvent(str);
23             }
24         }
25     }
26
27     public class MainClass
28     {
29         public static void Main()
30         {
31             EventTest et = new EventTest();
32
33             et.addObserver(delegate (string param)
34             {
35                 System.Console.WriteLine("metoda 1: " + param);
36             });
37
38             et.addObserver(delegate (string param)
39             {
40                 System.Console.WriteLine("metoda 2: " + param);
41             });
42
43             et.notifyObservers("hello");
44         }
45     }
46 }
```

Rys.3 Przykład implementacji w języku C# (źródło: własne)

4. Python

Następny przykład będzie implementacją w języku python z wykorzystaniem kilku cech tego języka: dekoratorów, wykorzystaniu faktu, że każda funkcja jest obiektem, więc możemy do niej przypisać zmienne oraz przekazywanie do funkcji listy wartości(*args) oraz słownika kluczy-wartości(**kwargs). Implementacja ta zaoferuje funkcjonalność bardzo podobną do funkcjonalności w języku BIO, tzn. dla każdej funkcji dla której użyjemy dekoratora @Event będzie możliwe dodanie i usunięcie funkcji obserwatora. Użycie *args i **kwargs umożliwi nam wykorzystanie dekoratora @Event z funkcjami o dowolnej liczbie parametrów. Dekorator @Event tworzy funkcję, która przy wywołaniu wywołuje funkcję pierwotną, a następnie wywołuje wszystkich obserwatorów z tymi samymi parametrami. Dodatkowo dodaje do tej funkcji listę obserwatorów.

```python
def attach_to_event(subject, observer):
    subject.observers.append(observer)

def detach_from_event(subject, observer):
    subject.observers.remove(observer)

def event(func):
    def inner(*args, **kwargs):
        func(*args, **kwargs)
        for observer in inner.observers:
            observer(*args, **kwargs)

    inner.observers = []
    return inner

@event
def some_event(data, data2):
    print("some_event " + str(data) + str(data2))

def foo(data, data2):
    print("foo " + str(data) + str(data2))

def foo2(data, data2):
    print("foo2 " + str(data))

attach_to_event(some_event, foo)
attach_to_event(some_event, foo2)
some_event("test", " wiadomosc")
detach_from_event(some_event, foo)
some_event("test", " wiadomosc")
```

Rys.4 Przykład implementacji w pythonie, która daje bardzo podobną funkcjonalność jak ta w języku BIO (źródło: własne)

5. BIO

Poniżej przedstawiona będzie implementacja wzorca obserwatora w języku BIO. Język BIO charakteryzuje się tym, że każdej funkcji możemy przypisać inną funkcję będącą obserwatorem (także funkcją wbudowanym jak PRINT). Żeby to osiągnąć wywołujemy funkcję ATTACH_TO_EVENT, której podajemy temat i obserwatora. Ilość parametrów przyjmowanych przez funkcję obserwatora musi być taka sama jak przyjmowana przez funkcję tematu.

```
1  def SOME_EVENT(input)
2  end
3
4  def FOO(param)
5     PRINT(param)
6  end
7
8  def onSTART()
9     ATTACH_TO_EVENT(SOME_EVENT, FOO)
10    SOME_EVENT("test") % zostanie wypisane na ekran "test"
11 end
```

Rys.5 Przykład implementacji w języku BIO (źródło: własne)