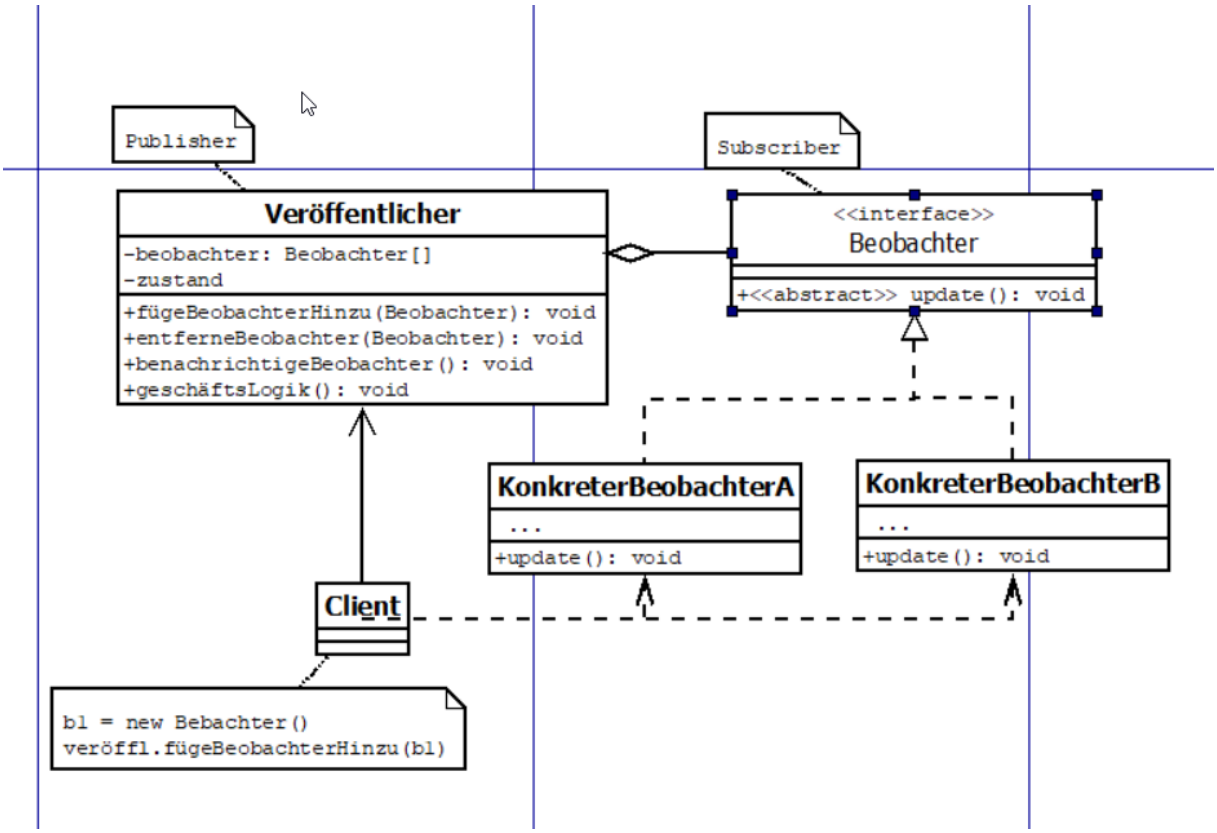
GoF Entwurfsmuster

Observer Pattern / Beobachter-Muster

Kategorie:

Verhaltensmuster



Einsatzzweck

Das Problem erläutert an einem kleinen Beispiel:

Stellen Sie sich vor es gibt zwei Objekte eine Boutique und eine Kundin,
die gerne das neueste Designerstück ergattern möchte, was
bald erhältlich sein soll.

Es gibt nun zwei Strategien, wie die Kundin an ihr begehrtes Stück kommen könnte:

1. Sie geht täglich in die Boutique und schaut, ob das Produkt nun erhältlich ist. -> Nachteil, die meisten ihrer Wege sind überflüssig.
2. Die Boutique versendet immer Unmengen von Emails an alle ihre Kunden, sobald ein neues Produkt erhältlich ist. Viele Kunden könnten das aber auch als Spam auffassen.

Wir haben also folgenden Konflikt: Entweder verschwendet die Kundin ihre Zeit, um die Produktverfügbarkeit zu prüfen oder die Boutique verschwendet Ressourcen, um die

falschen Kunden zu benachrichtigen und zu verärgern. Lösen könnten wir diesen Konflikt mit Hilfe des Beobachter-Musters.

Informationen

- Das Beobachter-Muster ist ein Verhaltensmuster, ist auch bekannt als Event-Subscriber oder Listener-Muster.
- Ist ein Verhaltensmuster, wir definieren ein Abonnement Mechanismus, um mehrere Objekte zu benachrichtigen, wenn mit dem beobachteten Objekt besondere Ereignisse geschehen.
- Das Objekt, dass die interessanten Zustände hat benachrichtigt andere Objekte über die Änderungen seines Zustands. Darum nennen wir dieses Objekt auch Publisher (Veröffentlicher). Die anderen Objekte, die diese Änderungen interessiert verfolgen, werden Subscribers (Abonnenten) genannt.
- Das Observer Pattern verlangt, dass ein Abonnement Mechanismus in der Veröffentlicher Klasse implementiert wird, so dass die interessierten Objekte diesen Strom an Events abonnieren oder deabonnieren können.

Implementierung

- Dieses Muster ist leichter zu implementieren, als man zuerst glauben mag.
- Der Mechanismus besteht aus:
  1. einem Array Attribut, um eine Liste von Referenzen zu den Abonnenten zu speichern.
  2. Verschiedene öffentliche Methoden, um die Abonnenten der Liste hinzuzufügen oder um diese wieder entfernen zu können.

- Wann immer nun ein wichtiges Ereignis mit dem Veröffentlicher Objekt geschieht, wird dieses an seine Abonnenten weitergereicht und ruft die speziellen Benachrichtigungsmethoden auf die Objekte auf.
- Echte Anwendungen haben häufig dutzende von verschiedenen Abonnementen Klassen, die Interesse an den Events derselben Veröffentlicher Klasse haben. Du möchtest den Veröffentlicher nicht mit all diesen Klassen eng verkuppeln, sondern
- dem Veröffentlicher sollte es egal sein, wer ihn abonniert. Darum ist es wichtig, dass alle Abonnement Klassen dasselbe Interface implementieren und dass der Veröffentlicher ausschließlich über dieses Interface mit ihnen kommuniziert.
- Dadurch erreichen wir eine sehr lose Kopplung. Das Interface deklariert die Benachrichtigungsmethode mit einigen Parametern, die der Veröffentlicher verwenden kann, um kontextuelle Daten gemeinsam mit seinen Benachrichtigungen weiterzugeben.

- Analogie in der echten Welt: Abodienste sorgen dafür, dass Kunden nicht mehr in ein Geschäft gehen müssen, um zu sehen, ob das Objekt der Begierde vorrätig ist. Stattdessen versendet der Veröffentlicher Hinweise an die Abonnenten. Der Veröffentlicher hat eine Liste vor mit all seinen Abonnenten und weiß an welchen Magazinen sie jeweils Interesse haben. Die Abonnenten können jederzeit das Abo abbestellen und sich aus der Liste entfernen lassen.

Struktur und Implementierungsschritte des Observer Muster

1. Eine Publisher Klasse leitet interessante Ereignisse weiter an die abonierenden Objekte. Diese Ereignisse treten auf, wenn der Publisher seinen Zustand ändert oder bestimmtes Verhalten ausführt. Publisher haben einen Abonnement Mechanismus, der es erlaubt neue Abonnenten seiner Liste hinzuzufügen und bestehende Abonnenten wieder aus seiner Liste zu löschen.

2. Wenn ein neues Ereignis auftritt, wird die Publisher Klasse über seine Liste der Abonnenten iterieren und die Methoden, die imAbonnement Interface deklariert wurden auf jedes Abonnenten Objekt aufrufen.
3. Das Abonnenten Interface deklariert das Benachrichtigungsinterface. In den meisten Fällen besteht dies aus einer einzigen Update Methode. Die Methode kann über verschiedene Parameter verfügen, wodurch es dem Publisher ermöglicht wird, Ereignisdetails mitzuliefern.
4. Abonnenten führen bestimmte Aktionen aus, sobald der Publisher das Ereignis feuert. Alle Abonnenten Klassen müssen dasselbe Interface implementieren. Dadurch ist der Publisher nicht mit konkreten Klassen verkuppelt.
5. Die Abonnenten brauchen in der Regel einige kontextbezogene Informationen, um mit dem Update korrekt umgehen zu können. Aus diesem Grund werden Publisher Klassen häufig Kontextdaten als Argumente übergeben mittels der Benachrichtigungsmethode. Der Publisher kann sichauch selbst als Argument übergeben, das ermöglicht Abonnenten alle benötigten Daten direkt abzugreifen.
6. Der Client kreiert Publisher und Abonnenten Objekte und registriert die Abonnenten für die Publisher Updates.