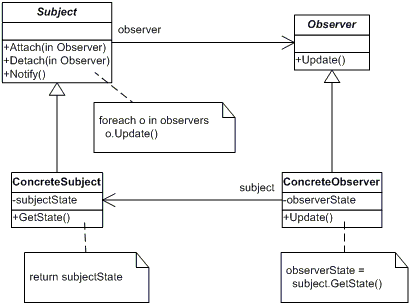
Design Patterns / Шаблони за дизайн

Наблюдател (Observer) - Дефинира зависимост "един към много" между обектите, така че ако един обект промени състоянието си, всички зависими от него обекти да бъдат известени и обновени автоматично.

Шаблонът за дизайн Наблюдател е още един, който попада в категорията на "шаблон на поведение(bahavioral pattern)". Съдейки по името му, можем да кажем, че наблюдател е нещо, коeто се търси при(следене) друг обект(и). Наблюдател шаблон е известен да се основава на "Холивуд Принципа", който казва-"Не ни се обаждайте, ние ще ви се обадим." Pub-Sub (Publisher-Subscriber) е още един популярен псевдоним даден модел Observer.

Въз основа на "принципа на Холивуд", ние може да направим предположение, че в шаблона на наблюдател, има специален Холивуд обект, от който всички други общи предмети са заинтересовани. В реални условия в схемата на наблюдател - "Има N номера на наблюдатели (обекти), които са заинтересовани в специален обект (предмет). Обяснявайки една стъпка по-нататък-има различни обекти (наречени наблюдатели), които се интересуват от нещата, които се случват на специален предмет. Така те се регистрират(или абонират) към обекта, който следят. Наблюдателите са заинтересовани към проявата на дадено събитие(това събитие обикновено се случва в границата на обект обект), когато това събитие се случи(от обект) наблюдателите са уведомени(те са се абонирали за случващото се събитие).



*// Наблюдател*

**using** System;

**using** System.Collections.Generic;

**namespace** Observer

{

*/// <summary>*

*/// Observer Design Pattern - главна програма*

*/// </summary>*

**class** MainApp

{

*/// <summary>*

*/// стартиращ метод на програмата*

*/// </summary>*

**static** **void** Main ()

{

*// конфигуриране*

ConcreteSubject objConcreteSubject = new ConcreteSubject ();

objConcreteSubject.Attach ( new ConcreteObserver ( objConcreteSubject, "X" ) );

objConcreteSubject.Attach ( new ConcreteObserver ( objConcreteSubject, "Y" ) );

objConcreteSubject.Attach ( new ConcreteObserver ( objConcreteSubject, "Z" ) );

*// промени темата и съобщи за промяната на наблюдателите*

objConcreteSubject.SubjectState = "ABC";

objConcreteSubject.Notify ();

*// промени темата и съобщи за промяната на наблюдателите*

objConcreteSubject.SubjectState = "DEF";

objConcreteSubject.Notify ();

Console.ReadKey ();

}

}

*/// <summary>*

*/// Абстрактен клас на темата*

*/// </summary>*

**abstract** **class** Subject

{

**private** List<Observer> \_LstObservers = new List<Observer> ();

**public** **void** Attach ( Observer objObserver )

{

\_LstObservers.**Add** ( objObserver );

}

**public** **void** Detach ( Observer objObserver )

{

\_LstObservers.**Remove** ( objObserver );

}

**public** **void** Notify ()

{

**foreach** (Observer objObserver **in** \_LstObservers)

{

objObserver.Update ();

}

}

}

*/// <summary>*

*/// Конкретния клас на темата*

*/// </summary>*

**class** ConcreteSubject : Subject

{

**private** **string** \_subjectState;

*// Gets or sets subject state*

**public** **string** SubjectState

{

**get** { **return** \_subjectState; }

**set** { \_subjectState = **value**; }

}

}

*/// <summary>*

*/// абстрактния клас на Наблюдателя*

*/// </summary>*

**abstract** **class** Observer

{

**public** **abstract** **void** Update ();

}

*/// <summary>*

*/// Конкретния наблюдател*

*/// </summary>*

**class** ConcreteObserver : Observer

{

**private** **string** \_name;

**private** **string** \_LstObserverstate;

**private** ConcreteSubject \_subject;

*// Constructor*

**public** ConcreteObserver (

ConcreteSubject subject, **string** name )

{

**this**.\_subject = subject;

**this**.\_name = name;

}

**public** **override** **void** Update ()

{

\_LstObserverstate = \_subject.SubjectState;

Console.WriteLine ( "Observer {0}'s new state is {1}",

\_name, \_LstObserverstate );

}

*// Сложи или предай тема*

**public** ConcreteSubject Subject

{

**get** { **return** \_subject; }

**set** { \_subject = **value**; }

}

}

}