Паттерн Mediator

Название

Посредник

Также известен как

-

Классификация

По цели: поведенческий
По применимости: к объектам

Частота использования

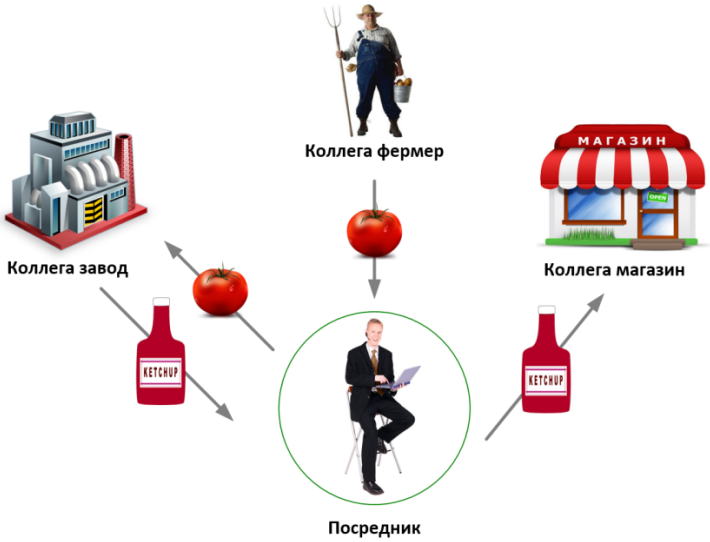
Ниже средней - 1 **2** 3 4 5

Назначение

Паттерн Mediator – предоставляет объект-посредник, скрывающий способ взаимодействия множества других объектов-коллег. Объект-посредник обеспечивает слабую связанность системы, избавляя объектов-коллег от необходимости явно ссылаться друг на друга, позволяя тем самым независимо изменять взаимодействия между объектами-коллегами.

Введение

Предлагается рассмотреть пример работы человека-посредника в объективной реальности с точки зрения его полезности, а не с точки зрения извлечения им прибыли. На рисунке ниже показана роль посредника в процессе производства кетчупа. Посредник покупает свежие помидоры у фермера, отправляет эти помидоры на консервный завод для переработки в кетчуп, а полученный кетчуп оптом продает в магазин розничной торговли.



Достоинство такого подхода заключается в том, что фермеру не требуется знать о заводе по переработке помидоров и в какой продукт будет «превращен» выращенный им помидор. Заводу не требуется заботиться о сбыте своей продукции так как посредник сам позаботится о поиске подходящего магазина розничной торговли для того чтобы товар донести до покупателя. Каждый занимается своим делом.

Предлагается выразить «*программно*» пример с фермером, фабрикой, магазином и посредником.

```
class Program
{
    static void Main()
    {
        var mediator = new ConcreteMediator();
        var farmer = new Farmer(mediator);
        var cannery = new Cannery(mediator);
        var shop = new Shop(mediator);

        mediator.Farmer = farmer;
        mediator.Cannery = cannery;
        mediator.Shop = shop;

        farmer.GrowTomato();
    }
}
```

```
abstract class Mediator
{
    public abstract void Send(string msg,
                      Colleague colleague);
}
```

```
abstract class Colleague
{
    protected Mediator mediator;

    public Colleague(Mediator mediator)
    {
        this.mediator = mediator;
    }
}
```

```
class Farmer : Colleague
{
    public Farmer(Mediator mediator)
        : base(mediator)
    {
    }

    public void GrowTomato()
    {
        string tomato = "Tomato";
        Console.WriteLine(this.GetType().Name
            + " raised " + tomato);
        mediator.Send(tomato, this);
    }
}
```

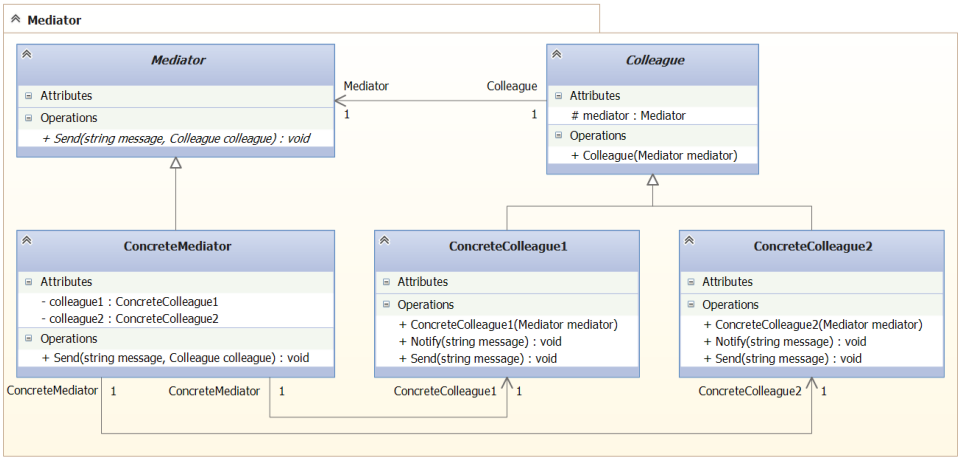
```
class ConcreteMediator : Mediator
{
    public Farmer Farmer { get; set; }
    public Cannery Cannery { get; set; }
    public Shop Shop { get; set; }

    public override void Send(string message,
                      Colleague colleague)
    {
        if (colleague == Farmer)
        {
            Cannery.MakeKetchup(message);
        }
        else if (colleague == Cannery)
        {
            Shop.SellKetchup(message);
        }
    }
}
```

```
class Cannery : Colleague
{
    public Cannery(Mediator mediator)
        : base(mediator)
    {
    }

    public void MakeKetchup(string message)
    {
        string ketchup = message + "Ketchup";
        Console.WriteLine(this.GetType().Name
            + " produced " + ketchup);
        mediator.Send(ketchup, this);
    }
}
```

```
class Shop : Colleague
{
     public Shop(Mediator mediator)
        : base(mediator)
    {
    }

    public void SellKetchup(string message)
    {
        Console.WriteLine(this.GetType().Name
            + " sold " + message);
    }
}
```

См. Пример к главе: \017_Mediator\001_Mediator

Структура паттерна на языке UML



См. Пример к главе: \017_Mediator\001_Mediator

Структура паттерна на языке C#

```
class Program
{
    static void Main()
    {
        var mediator = new ConcreteMediator();
        var colleague1 = new ConcreteColleague1(mediator);
        var colleague2 = new ConcreteColleague2(mediator);

        mediator.Colleague1 = colleague1;
        mediator.Colleague2 = colleague2;
        colleague1.Send("Message 1");
        colleague2.Send("Message 2");
    }
}
```

```
abstract class Mediator
{
    public abstract void Send(string msg, Colleague colleague);
}
```

```
class ConcreteMediator : Mediator
{
    public ConcreteColleague1 Colleague1 { get; set; }
    public ConcreteColleague2 Colleague2 { get; set; }

    public override void Send(string msg, Colleague colleague)
    {
        if (Colleague1 == colleague)
            Colleague2.Notify(msg);
        else
            Colleague1.Notify(msg);
    }
}
```

```
abstract class Colleague
{
    protected Mediator mediator;

    public Colleague(Mediator mediator)
    {
        this.mediator = mediator;
    }
}
```

```
class ConcreteColleague1 : Colleague
{
    public ConcreteColleague1(Mediator mediator)
        : base(mediator)
    {
    }

    public void Send(string message)
    {
        mediator.Send(message, this);
    }

    public void Notify(string message)
    {
        Console.WriteLine(message);
    }
}
```

```
class ConcreteColleague2 : Colleague
{
    public ConcreteColleague2(Mediator mediator)
        : base(mediator)
    {
    }

    public void Send(string message)
    {
        mediator.Send(message, this);
    }

    public void Notify(string message)
    {
        Console.WriteLine(message);
    }
}
```

См. Пример к главе: \017_Mediator\001_Mediator

Участники

- `Mediator` - **Посредник**:
  Предоставляет интерфейс для организации процесса по обмену информацией между объектами типа `Colleague`.

- `ConcreteMediator` - **Конкретный посредник**:
  Реализует алгоритм взаимодействия между объектами-коллегами.

- `Colleague` - **Коллега**:
  Предоставляет интерфейс для организации процесса взаимодействия объектов-коллег с объектом типа `Mediator`.

- `ConcreteColleague` - **Конкретный коллега**:
  Каждый объект-коллега знает только об объекте-медиаторе. Все объекты-коллеги обмениваются информацией только через посредника (медиатора).

Отношения между участниками

Отношения между классами

- Класс `ConcreteMediator` связан связью отношения наследования с абстрактным классом `Mediator` и связями отношения ассоциации с классами `ConcreteColleague`.

- Абстрактный класс `Colleague` связан связью отношения ассоциации с абстрактным классом `Mediator`.

- Конкретные классы `ConcreteMediator` связаны связью отношения наследования с абстрактным классом `Colleague`.

Отношения между объектами

- Объекты-коллеги (`ConcreteColleague`) посылают запросы объекту-посреднику (`ConcreteMediator`), а он в свою очередь перенаправляет полученные запросы другим объектам-коллегам, тем самым организуя процесс взаимодействия между объектами-коллегами.

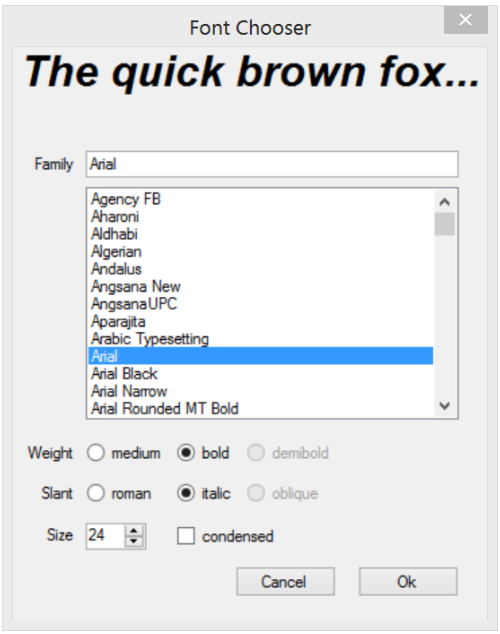
Мотивация

Объектно-ориентированное проектирование предоставляет возможности распределять поведение (функциональность) между различными объектами. При этом между имеющимися объектами может возникать очень много связей отношений, каждый объект (в худшем случае) может ссылаться на множество других объектов, тем самым превращая представление связей отношений в сложно понимаемую и сложно сопровождаемую сеть из связей. Если между объектами связей отношений много, то такая система превращается в монолит и объектно-ориентированный подход к разработке не даст никаких преимуществ. В системах со множеством связей между объектами, очень сложно вносить изменения, поскольку функциональность распределена между многими объектами и соответственно работа одних объектов зависит от функциональных возможностей других объектов. Часто при попытке внесения изменений в определенный класс приходится вносить изменения сразу в несколько других классов.

Использование подхода, предлагаемого паттерном Mediator позволит устранить сильную связанность между объектами при этом упростить правила взаимодействия между объектами, что соответственно сделает систему проще для понимания, сопровождения и расширения.

Предлагается рассмотреть реализацию диалогового окна выбора и настройки шрифта. В диалоговом окне может располагаться ряд элементов управления: кнопки (`Button`), поля ввода (`TextBox`) и т.д.

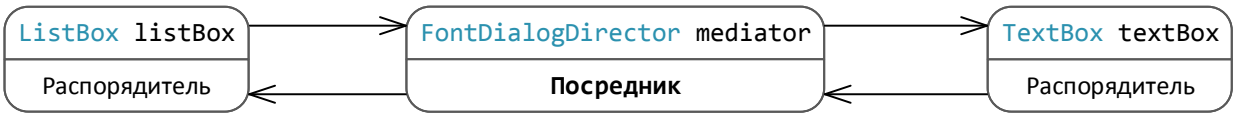
Часто между разными элементами управления в диалоговом окне имеется много зависимостей (одни элементы обращаются к другим). Например, если поле ввода пустое, то определенная кнопка может быть недоступна «загреена», а если в поле ввести текст, кнопка становится активной. На рисунке ниже показано, что при выборе элемента списка, изменяется содержимое поля ввода (выбрав в списке шрифт Arial, имя этого шрифта отображается в поле ввода). И наоборот, ввод текста в поле может автоматически привести к выбору элемента списка. Этот пример показывает, что некоторые элементы управления диалогового окна взаимодействуют друг с другом.



В разных диалоговых окнах взаимодействие между элементами управления может отличаться. Несмотря на то что во всех окнах используется элементы управления из стандартного набора (представленных в окне Toolbox, в Visual Studio), просто взять и использовать существующие классы элементов управления без дополнительной настройки взаимодействия не получится. Индивидуальная настройка каждого элемента управления для организации его взаимодействия с другими элементами управления часто приводит к запутанной логике взаимодействий, особенно когда взаимодействующих элементов управления много.

Проблем со сложностью организации взаимодействия между элементами управления можно избежать, если реализацию логики взаимодействия вынести в отдельный объект-посредник, с которым будут взаимодействовать все элементы управления. Использование объекта-посредника избавит элементы управления от необходимости ссылаться друг на друга. Элементы управления будут взаимодействовать только с объектом-посредником, который будет перенаправлять запросы от одного элемента управления другому.

Например, экземпляр класса `FontDialogDirector` может выступать в качестве посредника между элементами управления в диалоговом окне. Экземпляр класса `FontDialogDirector` «знает» обо всех элементах управления, используемых в контексте диалогового окна и руководит взаимодействием между этими элементами управления, другими словами выполняет функции центра взаимодействий.



См. Пример к главе: \017_Mediator\002_Motivation_Mediator

Применимость паттерна

Паттерн Mediator рекомендуется использовать, когда:

- Имеется некоторое количество объектов, связанных друг с другом при этом связи между этими объектами запутаны и сложны для понимания.

- Возникают трудности повторного использования объекта из-за его тесных связей с другими объектами.

Результаты

Использование паттерна Mediator предоставляет следующие возможности:

- **Уменьшает число создаваемых подклассов.**
  Класс `ConcreteMediator` собирает в себе все поведение (логику взаимодействия между коллегами), которое в противном случае пришлось бы распределять между объектам-коллегами. Для изменения взаимодействия между коллегами потребуется создать производный класс от класса `Mediator`, при этом классы коллег `ConcreteColleague` можно использовать повторно без внесения в них изменений.

- **Устранение сильной связанности между коллегами.**
  Класс `Mediator` обеспечивает слабую связанность между коллегами. Изменять классы посредников `ConcreteMediator` и коллег `ConcreteColleague` можно независимо друг от друга.

- **Упрощение правил (протокола и церемониала) взаимодействия между объектами коллегами.**
  Класс `Mediator` заменяет технику взаимодействия «*все со всеми*» техникой «*один со всеми*», то есть один посредник взаимодействует со всеми коллегами. Отношения «*один ко многим*» проще для понимания, сопровождения и расширения. Коллеги ничего не знают друг о друге. Отношение «*все со всеми*» представляют сложную запутанную сеть, которую сложно анализировать и сопровождать.

- **Алгоритмическое управление связями отношений между коллегами.**
  Механизм «*посредничества*», который расположен внутри объекта-посредника (`ConcreteMediator`), позволяет программисту сосредоточиться именно на взаимодействии объектов-коллег (в общем) а не на их индивидуальных отношениях друг с другом. Посредник использует технику алгоритмического управления связями отношений между объектами, самостоятельно переадресовывая запросы от одного коллеги другому.

- **Централизация управления отношениями между коллегами.**
  Класс (`ConcreteMediator`) сосредотачивает в себе сложные отношения и взаимодействия между коллегами (`ConcreteColleague`). Так как посредник содержит в себе алгоритмическое представление правил взаимодействия между коллегами, то сам посредник может оказаться сложнее отдельных коллег. В результате посредник может оказаться сложным монолитом, что затрудняет его сопровождение.

Реализация

Полезные приемы реализации паттерна Mediator:

- **Отказ от использования абстрактного класса `Mediator`.**
  Если в программной подсистеме коллеги взаимодействуют только с одним посредником (`ConcreteMediator`), то можно не создавать базовый абстрактный класс `Mediator`. Но использование абстрактного класса `Mediator` позволит коллегам работать с разными классами `ConcreteMediator` благодаря технике абстрагирования вариантов использования.

- **Обмен информацией между посредником и коллегами.**
  Коллеги должны обмениваться информацией с посредником тогда, когда возникает надобность послать сообщение об изменении состояния коллеги. Общаясь с посредником, коллега передает ссылку на себя в качестве аргумента вызываемого метода на посреднике, тем самым давая возможность посреднику определить отправителя сообщения и выбрать коллегу-получателя, которому переадресовать сообщение, полученное от коллеги-отправителя.

Пример кода

См. Пример к главе: \017_Mediator\002_Motivation_Mediator