.NET - O padrão de projeto Mediator

Os padrões de projeto são soluções prontas e testadas que podem ser aplicadas a situações específicas em um projeto de software.

Você tem um problema ???

Parabéns, seu problema já foi resolvido !!!

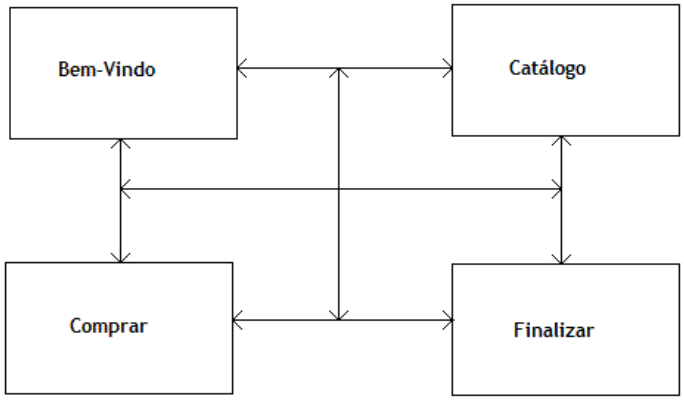
Basta encontrar o padrão que se encaixa ao seu problema...

Porque perder tempo tentando escrever código para um problema que você pensa que seja somente seu quando na verdade ele é um problema genérico que outros programadores já enfrentaram e JÁ resolveram ?

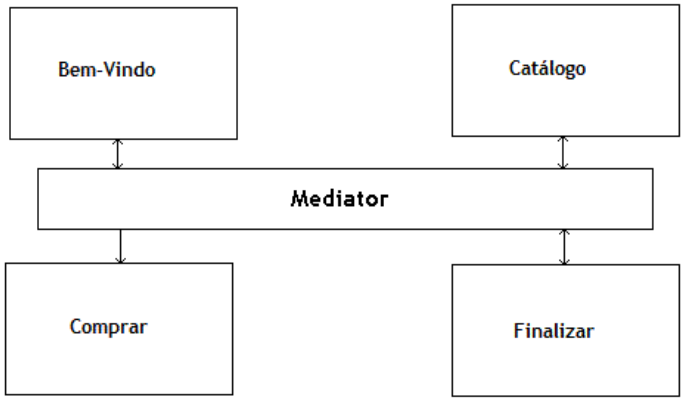
Hoje eu vou tratar do **padrão Mediator.**

Imagine a seguinte situação:

- Você desenvolveu um website para e-commerce bem simples onde existem 4 páginas onde os usuários podem consultar um catálogo de produtos e realizar compras;
- Neste cenário um usuário pode navegar entre as páginas e ai começam os problemas;
- O código em cada página precisa saber quando ir para uma nova página bem como ativar essa nova página;



- Dessa forma , e mesmo com apenas 4 páginas, existem muitas possibilidades de conexões e de navegação entre as páginas;
- Esse cenário costuma a gerar uma grande quantidade de código duplicado em cada página;
- Para resolver o problema o padrão Mediator pode ser usado para encapsular todo o código da navegação em um objeto separado de forma;
- Dessa forma cada página deverá apenas reportar qualquer alteração de estado para o objeto mediator que saberá qual página deve enviar;



Definição Formal

Define um objeto que encapsula como um conjunto de objetos interage.

O padrão **Mediator** promove um baixo acoplamento evitando que os objetos façam referência uns aos outros de forma explícita permitindo a você variar usa interação de forma independente.[**GoF**]

Problema

Como permitir que um grupo de objetos se comunique entre si sem que haja acoplamento entre eles ?

Como remover o forte acoplamento presente em relacionamentos muitos para muitos ?

Solução

Introduzir um **mediator** : Objetos podem se comunicar sem se conhecer.

- Um objeto Mediador deve encapsular toda a comunicação entre um grupo de objetos

- Cada objeto participante conhece o mediador mas ignora a existência dos outros objetos;
- O mediador conhece cada um dos objetos participantes;

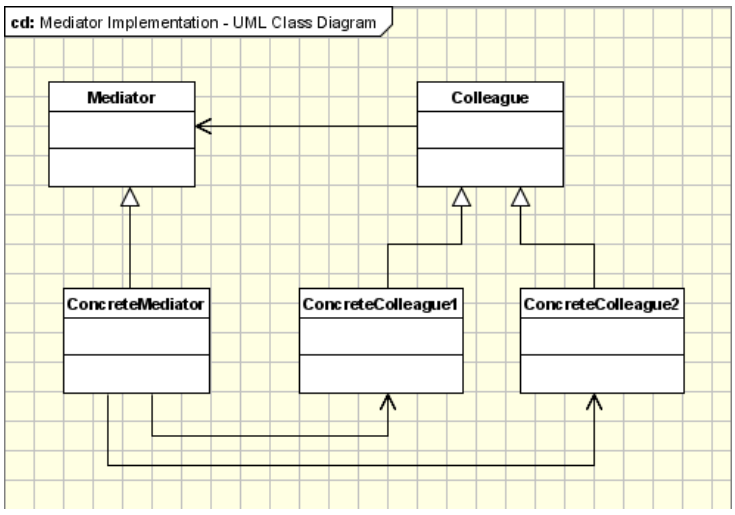
- A interface do Mediador é usada para iniciar a comunicação e receber notificações

- O mediador recebe requisições dos remetentes;
- O mediador repassa as requisições aos destinatários;

CheckList

- Identificar uma coleção de objetos que interagem e que se beneficiariam com o desacoplamento mútuo;
- Encapsular estas interações na abstração de uma nova classe;
- Criar uma instância desta nova classe e refazer todos os 'colegas' de objetos para interagir com um único Mediador;
- Equilibrar o princípio do desacoplamento com o princípio da distribuição de responsabilidade de maneira uniforme;
- Tomar cuidado para criar um 'Controlador' ou um objeto 'deus';

Diagrama de Classes UML



(fonte: http://www.oodesign.com/mediator-pattern.html)

Participantes (Classes)

- **Mediator -** Define uma interface para comunicação com os objetos Colleague;
- **ConcreteMediator -** Conhece as classes Colleague e mantêm uma referência aos objetos Colleague e implementa a comunicação e transferência de mensagens entre as classes Colegue;
- **Classes Colleague -** Mantém uma referência ao seu objeto **Mediator** - se comunicam com o Mediator sempre que necessário; de outra forma se comunica com um Colleague;

Vantagens

- Desacoplamento entre os diversos participantes da rede de comunicação: participantes não se conhecem;
- Eliminação de relacionamentos muitos para muitos (são todos substituídos por relacionamentos um para muitos);
- A política de comunicações está centralizada no mediador e pode ser alterada sem mexer nos colaboradores;

Desvantagens

- A centralização pode ser uma fonte de gargalos de desempenho e de risco para o sistema em caso de falha;
- Na prática os mediadores tendem a se tornarem mais complexos;

(fonte: 2003, Helder L. S da Rocha)

Padrões Relacionados

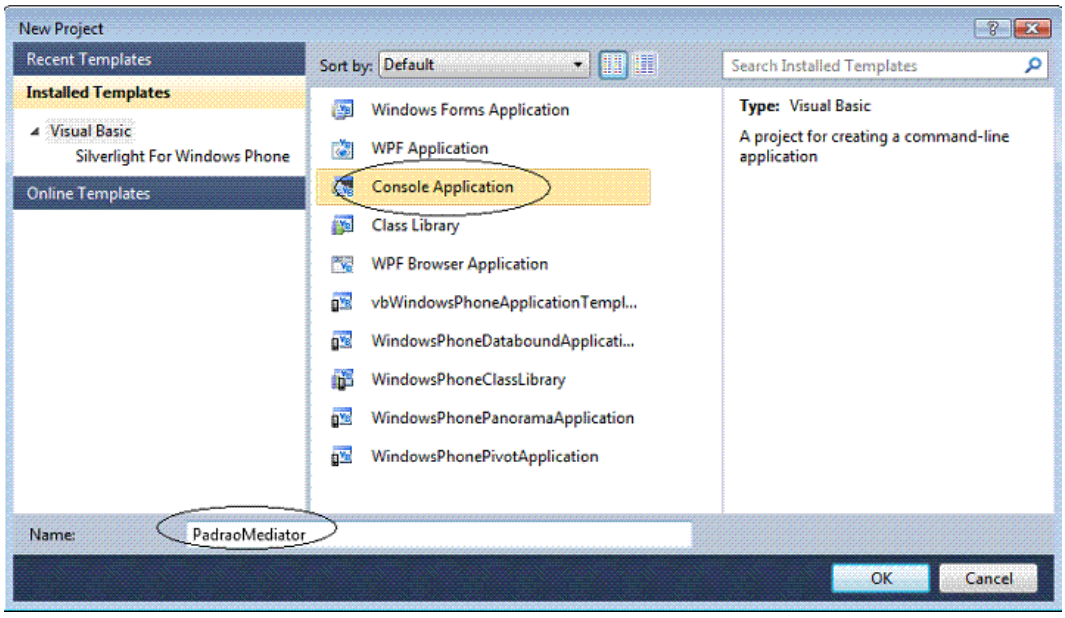
- **Facade** - Um mediator simplificado torna-se um padrão Facade se o médiador for a única classe ativa e as classes Colleagues forem classes passivas;
- **Adapter** - O padrão Mediator apenas media os pedisos entre as classes Colleague;
- **Observer** - Os padrões Mediator e Observer são semalhantes, resolvendo o mesmo problema.

Exemplo de Implementação em VB .NET

O exemplo a seguir demonstra a utilização do padrão **Mediator** para facilitar a comunicação usando o baixo acoplamento entre os diferentes **participantes de um Chat.**

Este exemplo foi adaptado de original encontrado em: http://www.dofactory.com/Patterns/PatternMediator.aspx

Abra o Visual Basic 2010 Express Edition e crie um novo projeto do tipo **Console Application** com o nome **PadraoMediator**:



Vamos criar 5 classes em nosso projeto da seguinte forma : no menu **Project** clique em **Add Class** e a seguir informe o nome da classe e clique no botão **Add**:

As classes são:

- **Participante** - representa a classe abstrata *Colleague* -Mantém uma referência ao seu objeto *Mediator* - se comunicam com o *Mediator* sempre que necessário; de outra forma se comunica com um participante;
- **Membro** - representa a classe *ConcreteColleague* e herda de *Participante*;
- **NaoMembro** - representa a classe *ConcreteColleague* e herda de *Participante*;
- **AbstractChatSala** - representa a classe *Mediator* - Define uma interface para comunicação com os objetos *Participante*;

- **ChatSala** - representa a classe concreta *ConcreteMediator* - Conhece as classes Participante e mantêm uma referência aos objetos *Participante* e implementa a comunicação e transferência de mensagens entre as objetos da classes *Participante*;

Vejamos a seguir o código de cada uma dessas classes:

1- Participante.vb

```
''' <summary>
'''  A classe 'AbstractColleague'
''' </summary>
MustInherit Class Participante

    Private _chatsala As Chatsala
    Private _nome As String

    ' Construtor
    Public Sub New(ByVal nome As String)
        Me._nome = nome
    End Sub

    ' Pega o nome do participante
    Public ReadOnly Property Nome() As String
        Get
            Return _nome
        End Get
    End Property

    ' Pega a sala de chat
    Public Property Chatsala() As Chatsala
        Get
            Return _chatsala
        End Get
        Set(ByVal value As Chatsala)
            _chatsala = value
        End Set
    End Property

    ' Envia mensagem para um dado participante
    Public Sub Enviar(ByVal [para] As String, ByVal mensagem As String)
        _chatsala.Enviar(_nome, [para], mensagem)
    End Sub

    ' Recebe mensagem de um participante
    Public Overridable Sub Receber(ByVal [de] As String, ByVal mensagem As String)
        Console.WriteLine("{0} para {1}: '{2}'", [de], Nome, mensagem)
    End Sub

End Class
```

2- Membro.vb

```
''' <summary>
''' A classe 'ConcreteColleague'
''' </summary>
Class Membro
    Inherits Participante

    ' Construtor
    Public Sub New(ByVal nome As String)
        MyBase.New(nome)
    End Sub
    'sobrescreve o método Receber
    Public Overrides Sub Receber(ByVal [de] As String, ByVal mensagem As String)
        Console.Write("para Membro : ")
        MyBase.Receber([de], mensagem)
    End Sub
End Class
```

3- NaoMembro.vb

```
''' <summary>
''' A classe 'ConcreteColleague'
''' </summary>
Class NaoMembro
    Inherits Participante

    ' Construtor
    Public Sub New(ByVal nome As String)
        MyBase.New(nome)
    End Sub
    'sobrescreve o método Receber
    Public Overrides Sub Receber(ByVal [de] As String, ByVal mensagem As String)
        Console.Write("Para NaoMembro : ")
        MyBase.Receber([de], mensagem)
    End Sub
End Class
```

4- **AbstractChatSala**

```
''' <summary>
''' A classe abstrata 'Mediator'
''' </summary>
MustInherit Class AbstractChatSala
    Public MustOverride Sub Registro(ByVal participante As Participante)
    Public MustOverride Sub Enviar(ByVal [de] As String, ByVal [para] As String, ByVal message As String)
End Class
```

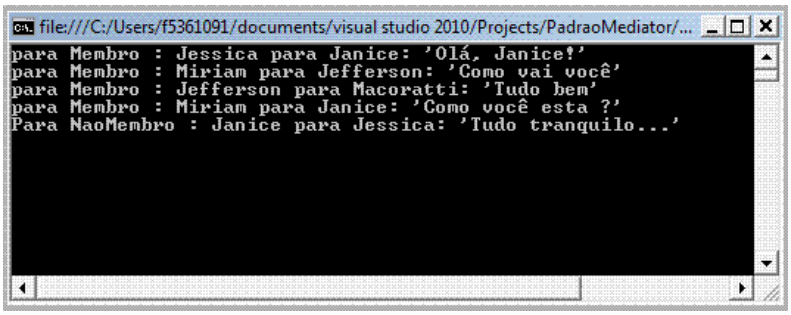
5- ChatSala

```
''' <summary>
''' A classe concreta  'ConcreteMediator'
''' </summary>
Class Chatsala
    Inherits AbstractChatSala

    Private _participantes As New Dictionary(Of String, Participante)()

    Public Overrides Sub Registro(ByVal _participante As Participante)
        If Not _participantes.ContainsValue(_participante) Then
            _participantes(_participante.Nome) = _participante
        End If

        _participante.Chatsala = Me
    End Sub

    Public Overrides Sub Enviar(ByVal [de] As String, ByVal [para] As String, ByVal mensagem As String)
        Dim _participante As Participante = _participantes([para])
        If _participante IsNot Nothing Then
            _participante.Receber([de], mensagem)
        End If
    End Sub
End Class
```

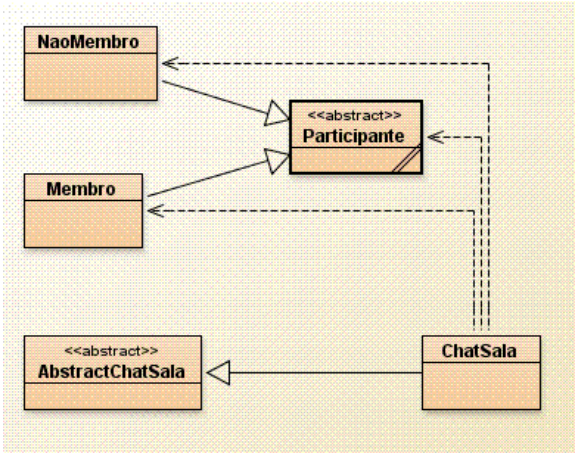
No módulo do projeto inclua na código abaixo no método **Main()** que irá testar a nossa pequena aplicação:

```
Module Module1

    Sub Main()
        'Cria uma sala de chat (chatsala)
        Dim chatsala As New Chatsala()

        ' cria participantes e faz o registro
        Dim Macoratti As Participante = New Membro("Macoratti")
        Dim Miriam As Participante = New Membro("Miriam")
        Dim Jefferson As Participante = New Membro("Jefferson")
        Dim Janice As Participante = New Membro("Janice")
```

```
        Dim Jessica As Participante = New NaoMembro("Jessica")

        'registra os participantes
        chatsala.Registro(Macoratti)
        chatsala.Registro(Miriam)
        chatsala.Registro(Jefferson)
        chatsala.Registro(Janice)
        chatsala.Registro(Jessica)

        ' Inicia o chat
        Jessica.Enviar("Janice", "Olá, Janice!")
        Miriam.Enviar("Jefferson", "Como vai você")
        Jefferson.Enviar("Macoratti", "Tudo bem")
        Miriam.Enviar("Janice", "Como você esta ?")
        Janice.Enviar("Jessica", "Tudo tranquilo...")

        ' aguarda...
        Console.ReadKey()
    End Sub

End Module
```



O diagrama de classes para o nosso exemplo feito no **BlueJ** e é exibido a seguir:



O padrão **mediator** é muito útil para programadores Visual Basic *(principalmente das versões 5 e 6*) pois ele é um atalho para a falta de herança. Ele é tipicamente usado em formulários onde o mesmo media por seus controle e componentes.

Pegue o projeto completo aqui: **PadraoMediator.zip**

"*Passará o céu e a terra, mas as minhas palavras jamais passarão.*" *(Mateus 24:35)*

Referências:

- Padrões de Projeto - O modelo MVC - Model View Controller
- Design Patterns - o padrão Factory
- Conceitos sobre projetos - Decomposição
- ASP .NET - MVC - Introdução
- http://en.wikipedia.org/wiki/Strategy_pattern
- Encapsulation and Inheritance in Object-Oriented Programming Languages
- http://martinfowler.com/bliki/DesignedInheritance.html
- http://sourcemaking.com/design_patterns/strategy
- Seção Padrões de projeto do site Macoratti.net

José Carlos Macoratti