



Chain of Responsibility

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini

Chain of Responsibility

Padrões de Projeto Comportamental II
despansore of respnsioinaneno

Jefferson Antonio Peressni



Acinbrie
Oacholin

Deutanier
Bistrudieie
Doseriaties

Amode
Reesemib

Micenialles
Imgoemmemuldo
rexpieretia

Bo
En

Jon
Re
ka

O padrão **Chain of Responsibility** evita o acoplamento do remetente de uma solicitação ao seu receptor, dando a mais de um objeto a oportunidade de tratar a solicitação. Ele encadeia os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto a trate.

Aplicabilidade

- Quando mais de um objeto pode tratar uma solicitação e não se sabe qual objeto fará tal tratamento. O objeto que trata a solicitação deve ser escolhido automaticamente.
- Ao ser necessário fazer uma solicitação para um dentre vários objetos sem especificar explicitamente para qual.
- Quando um conjunto de objetos que pode tratar uma solicitação deve ser especificado dinamicamente.

Componentes

- **Cliente:** Objeto que faz a solicitação.
- **Manipulador:** Define uma interface para tratar solicitações e pode implementar o elo (link) ao seu sucessor.
- **ManipuladorConcreto:** Classes candidatas a atender à solicitação do cliente. Pode acessar seu sucessor, portanto ele tenta atender a solicitação do cliente, caso não consiga, passa tal solicitação ao seu sucessor.

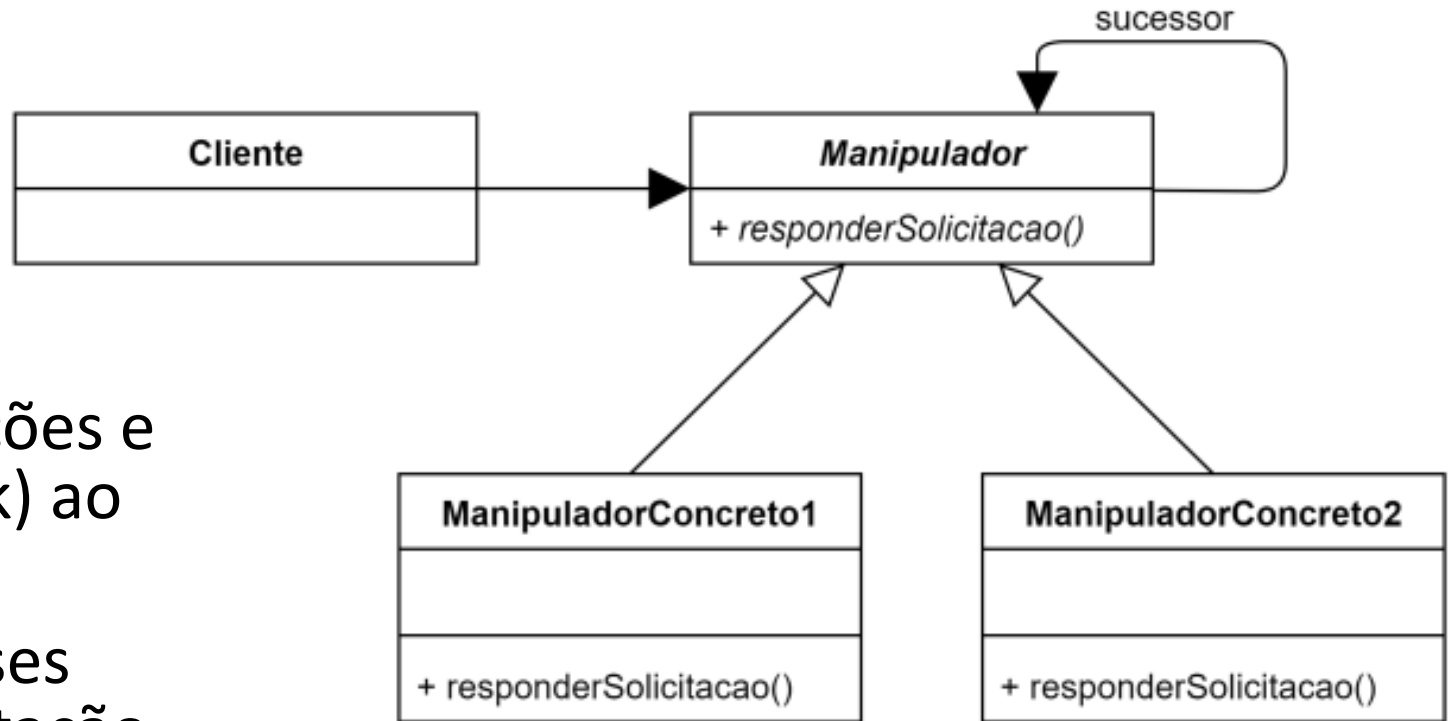


Diagrama de Classes

Motivação

- Durante o desenvolvimento de software é comum o surgimento de situações onde é necessário que apenas um tratamento entre muitos seja aplicado a um determinado fluxo. Isso pode parecer abstrato demais, então, apenas para exemplificar considere o código

```
public class CalculadorDePontos
{
    4 references
    public int CalcularPontos(PedidoLanche pedido, int dia)
    {
        int pontos = 0;
        if (pedido.valor >= 70)
        {
            pontos = (int) pedido.valor / 5;
        } else if (pedido.valor >= 40)
        {
            pontos = (int)pedido.valor / 7;
        } else if (pedido.valor >= 20)
        {
            pontos = (int)(pedido.valor / 10);
        } else
        {
            pontos = 0;
        }

        if(dia >= 16 && dia <= 31)
        {
            pontos *= 2;
        }
        return pontos;
    }
}
```

Motivação

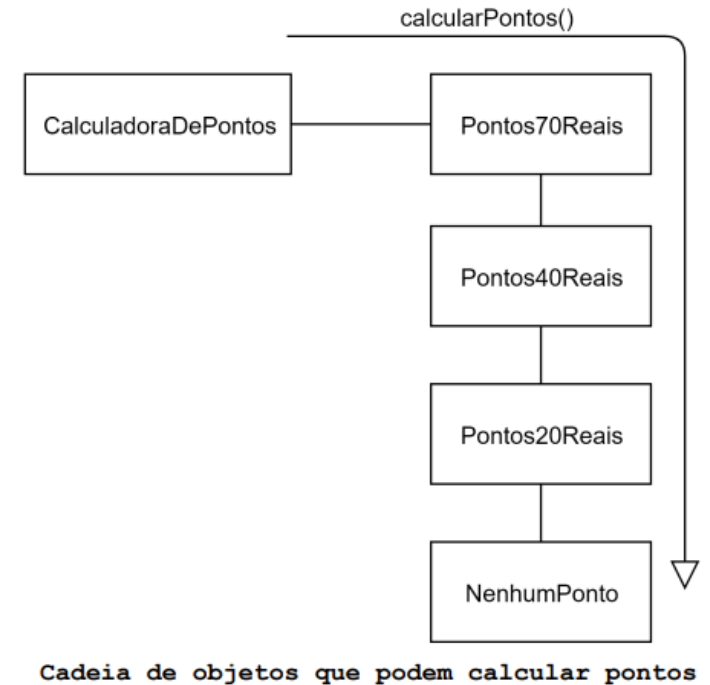
- O código demonstrado especifica a seguinte situação em um programa de fidelização por pontos:

Critérios	Pontos ganhos na (Primeira quinzena)	Pontos ganhos na (Segunda quinzena)
Pedido acima de R\$69,99	1 Ponto a cada 5 reais	2 Pontos a cada 5 reais
Pedido acima de R\$39,99	1 Ponto a cada 7 reais	2 Pontos a cada 7 reais
Pedido acima de R\$19,99	1 Ponto a cada 10 reais	2 Pontos a cada 10 reais
Pedidos abaixo de R\$20,00	0 pontos	0 Pontos

- É esperado que apenas uma das condições do método calcularPontos() seja satisfeita. Assim que a primeira for satisfeita não é mais necessário verificar as demais condições, ou seja, apenas um tratamento entre muitos é necessário.
- Códigos como o de cima, quando mal implementados, podem se tornar difíceis de ler e conter repetições de código dentro de cada condição.
- Códigos como o de cima, quando mal implementados, podem se tornar difíceis de ler e conter repetições de código dentro de cada condição.

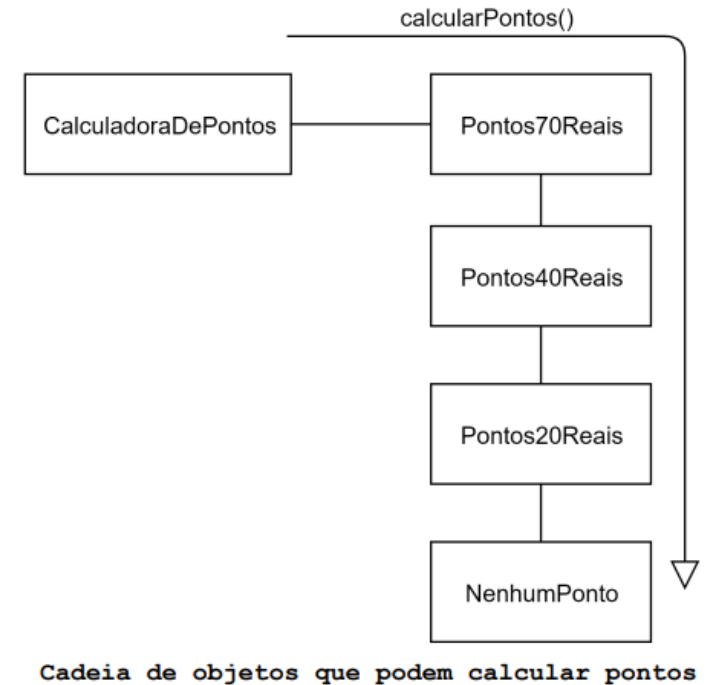
Motivação

- O Padrão **Chain of Responsibility** tem como objetivo evitar o acoplamento entre o objeto solicitante de uma chamada a um objeto solicitado.
- Isso é possível devido ao fato do padrão **Chain of Responsibility** dar a mais de um objeto a oportunidade de tratar a solicitação por meio de um encadeamento.
- A princípio pode ser que você pense que não existem grandes problemas na classe `CalculadoraDePontos`, ela realmente não é uma classe tão grande ou difícil de entender.
- Porém se o dono da hamburgueria decidir tornar seu plano de fidelidade mais complexo, ou então adicionar novos critérios de pontuação, a classe `CalculadoraDePontos` pode crescer de forma descontrolada.
- **Um dos principais objetivos dos padrões de projeto é facilitar a manutenção de código. Vamos ver como ficaria a solução deste problema utilizando o padrão Chain of Responsibility.**
- Precisamos criar uma cadeia de objetos que calculam pontos, se um objeto não puder calcular ele delega para o próximo objeto calculador de pontos da cadeia.



Motivação

- A classe CalculadoraDePontos não deve conhecer os objetos da cadeia de cálculo de pontos, ela só precisa ter a certeza que eles são capazes de tentar calcular pontos, ou seja, serem capazes de executar o método calcularPontos().
- A CalculadoraDePontos também precisa saber que caso o objeto da cadeia, por qualquer motivo, não consiga calcular os pontos ele possa passar a responsabilidade para outro objeto tentar fazer o cálculo.



Motivação

- A estrutura do projeto ficará conforme o diagrama.
- A cadeia de decisão (responsabilidade) ficará a cargo de nossa classe CalculadoraDePontos.
- E cada um dos tipos de cálculo ficará a cargo dos Calculadores Concretos (Pontos70reais, Pontos40reais, etc)

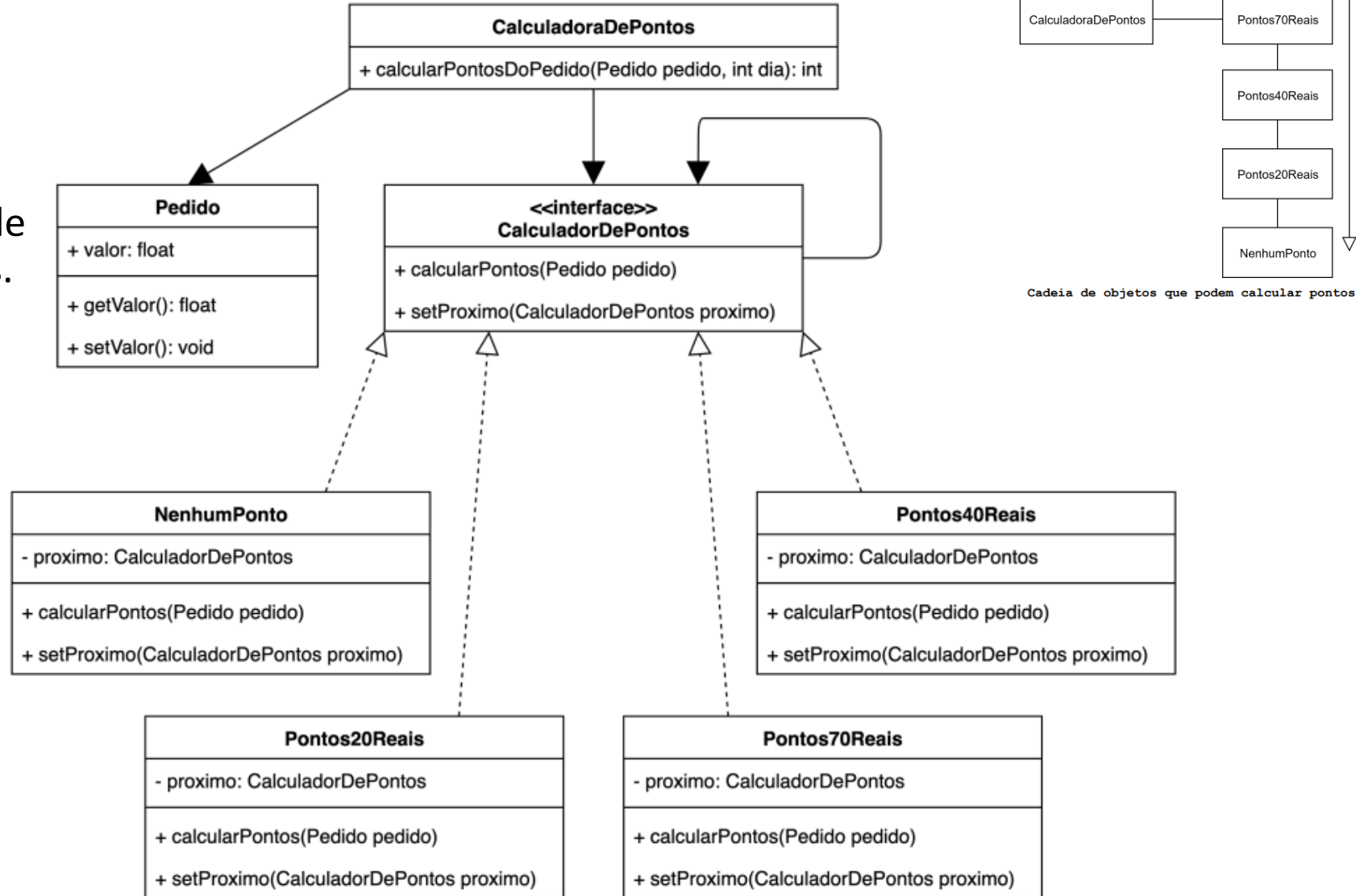


Diagrama de classes completo do exemplo



Chain of Responsibility – Implementação C#

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini

Chain of Responsibility

Padrões de Projeto Comportamental II
despanore of respnsioinaneno

Jefferson Antonio Peressni



Acinbrie
Oacholin

Deutanier
Bistrudie
Doserlaties

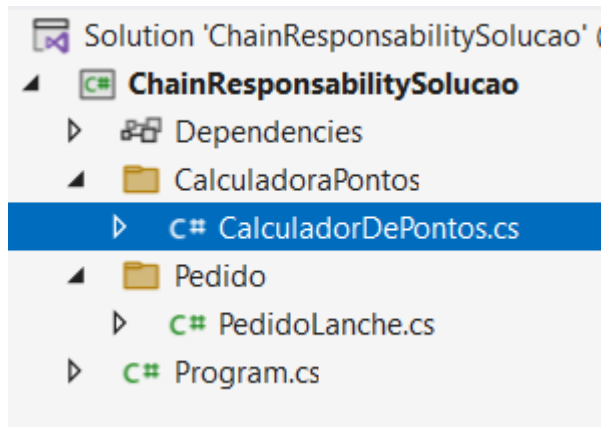
Amode
Reesemib

Micenialles
Imgoemmemuldo
rexpieretia

Bo
En
Jon
Re
ka

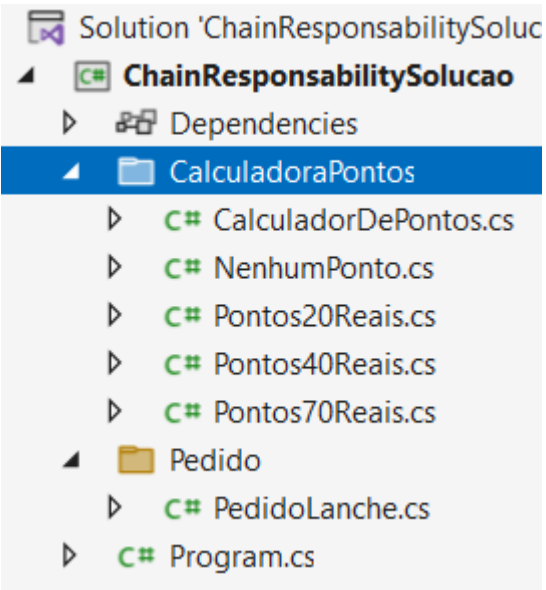
C#

- Para iniciar nosso desenvolvimento precisamos então de uma interface comum para todos os objetos calculadores de pontos, tal interface irá garantir para a **CalculadoraPontos** *que o objeto recebido por ela sabe calcular pontos e caso o cálculo não seja feito ele passará para o próximo objeto da cadeia de cálculo.*



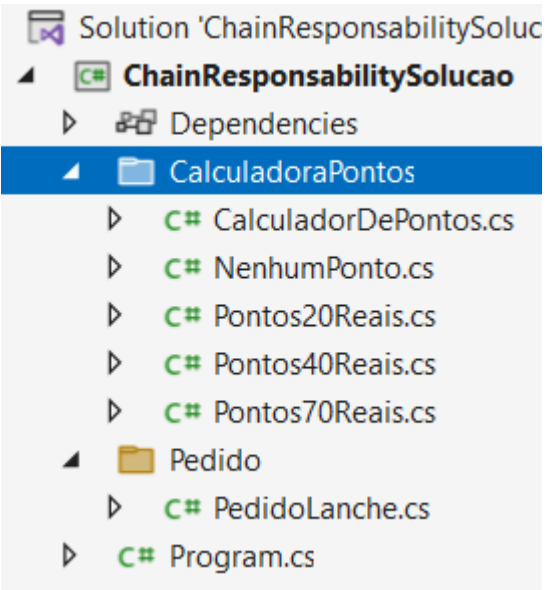
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using ChainResponsabilitySolucao.Pedido;
7
8  namespace ChainResponsabilitySolucao.CalculadoraPontos
9  {
10     3 references
11     public interface CalculadorDePontos
12     {
13         0 references
14         public int calcularPontos(PedidoLanche pedido);
15         0 references
16         public void setProximo(CalculadorDePontos proximo);
17     }
18 }
```

- Agora vamos criar as classes que calculam os pontos elas implementam a interface CalculadorDePontos.



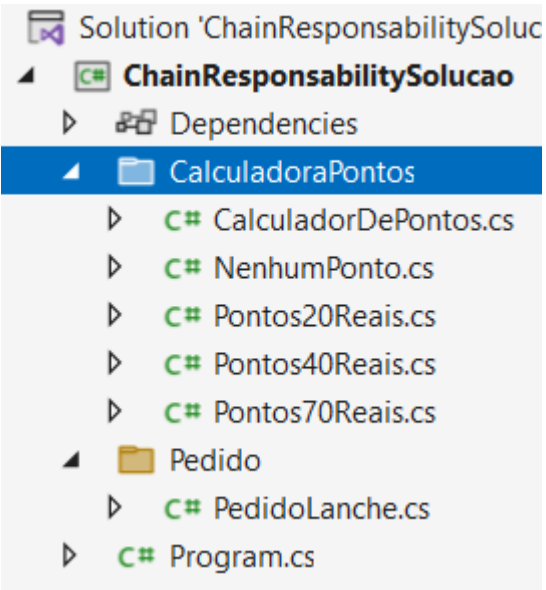
```
1  using ChainResponsabilitySolucao.Pedido;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace ChainResponsabilitySolucao.CalculadoraPontos
9  {
10     0 references
11     public class Pontos70Reais : CalculadorDePontos
12     {
13         private CalculadorDePontos proximoCalculadorDePontos;
14
15         4 references
16         public int calcularPontos(PedidoLanche pedido)
17         {
18             if (pedido.valor >= 70)
19             {
20                 return (int)(pedido.valor / 5);
21             }
22
23             return this.proximoCalculadorDePontos.calcularPontos(pedido);
24
25         1 reference
26         public void setProximo(CalculadorDePontos proximo)
27         {
28             this.proximoCalculadorDePontos = proximo;
29         }
30     }
31 }
```

- Agora vamos criar as classes que calculam os pontos elas implementam a interface CalculadorDePontos.



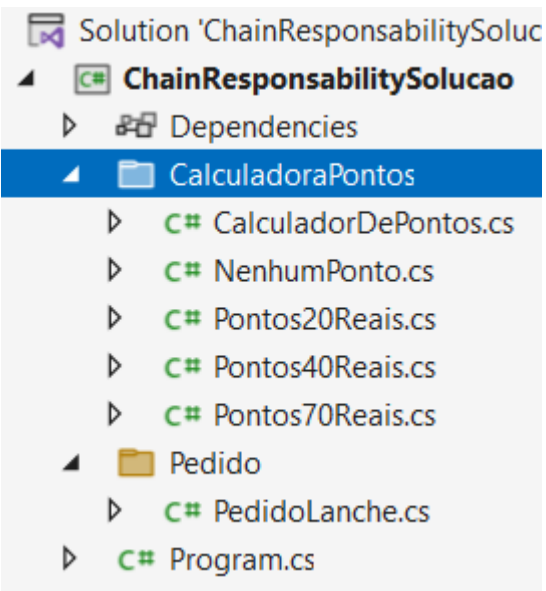
```
1  using ChainResponsabilitySolucao.Pedido;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace ChainResponsabilitySolucao.CalculadoraPontos
9  {
10     0 references
11     public class Pontos40Reais : CalculadorDePontos
12     {
13         private CalculadorDePontos proximoCalculadorDePontos;
14
15         4 references
16         public int calcularPontos(PedidoLanche pedido)
17         {
18             if (pedido.valor >= 40)
19             {
20                 return (int)(pedido.valor / 7);
21             }
22
23             return this.proximoCalculadorDePontos.calcularPontos(pedido);
24
25         1 reference
26         public void setProximo(CalculadorDePontos proximo)
27         {
28             this.proximoCalculadorDePontos = proximo;
29         }
30     }
31 }
```


- Agora vamos criar as classes que calculam os pontos elas implementam a interface CalculadorDePontos.



```
1  using ChainResponsabilitySolucao.Pedido;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace ChainResponsabilitySolucao.CalculadoraPontos
9  {
10     0 references
11     public class Pontos20Reais : CalculadorDePontos
12     {
13         private CalculadorDePontos proximoCalculadorDePontos;
14
15         4 references
16         public int calcularPontos(PedidoLanche pedido)
17         {
18             if (pedido.valor >= 20)
19             {
20                 return (int)(pedido.valor / 10);
21             }
22             return this.proximoCalculadorDePontos.calcularPontos(pedido);
23         }
24
25         1 reference
26         public void setProximo(CalculadorDePontos proximo)
27         {
28             this.proximoCalculadorDePontos = proximo;
29         }
30     }
31 }
```

- Agora vamos criar as classes que calculam os pontos elas implementam a interface CalculadorDePontos.



```
1  using ChainResponsabilitySolucao.Pedido;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace ChainResponsabilitySolucao.CalculadoraPontos
9  {
10     0 references
11     public class NenhumPonto : CalculadorDePontos
12     {
13         4 references
14         public int calcularPontos(PedidoLanche pedido)
15         {
16             return 0;
17         }
18
19         1 reference
20         public void setProximo(CalculadorDePontos proximo)
21         {
22             //fim da cadeia
23         }
24     }
25 }
```


C#

- Agora vamos criar as classes que calculam os pontos elas implementam a interface **CalculadorDePontos**.
- Para manter o foco no conceito do padrão, as classes calculadoras de pontos do nosso exemplo possuem os métodos **calcularPontos()** muito parecidos.
- É importante dizer que dependendo do contexto estes métodos podem ser completamente diferentes um do outro, essa é a “mágica” do padrão Chain of Responsibility.
- Poderíamos ter calculadores de pontos que levam em consideração a idade do cliente, o tipo de hambúrguer, forma de pagamento entre outros fatores.
- A classe **NenhumPonto** precisa implementar a interface **CalculadorDePontos** para ser aceita na cadeia, porém sua instância sempre será o último objeto da cadeia, então ela não terá um próximo objeto. Repare que o método **setProximo()** existe para manter o contrato da interface mas sua implementação é vazia.

C#

- Todos os calculadores de pontos já estão implementados, mas ainda falta o **orquestrador da cadeia de processamento**. Vejamos como fica a implementação da classe **CalculadoraDePontos** seguindo o padrão Chain of Responsibility.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using ChainResponsabilitySolucao.Pedido;
7
8  namespace ChainResponsabilitySolucao.CalculadoraPontos
9  {
10     0 references
11     public class CalculadoraDePontos
12     {
13         0 references
14         public int calcularPontos(PedidoLanche pedido, int dia)
15         {
16             //inicializa os calculadores da cadeia
17             CalculadorDePontos pontos70 = new Pontos70Reais();
18             CalculadorDePontos pontos40 = new Pontos40Reais();
19             CalculadorDePontos pontos20 = new Pontos20Reais();
20             CalculadorDePontos nenhumPonto = new NenhumPonto();
21
22             //define a ordem da cadeia de calculo.
23             pontos70.setProximo(pontos40);
24             pontos40.setProximo(pontos20);
25             pontos20.setProximo(nenhumPonto);
26
27             if (dia >= 16 && dia <= 31)
28             {
29                 return pontos70.calcularPontos(pedido) * 2;
30             }
31
32             return pontos70.calcularPontos(pedido);
33         }
34     }
35 }

```

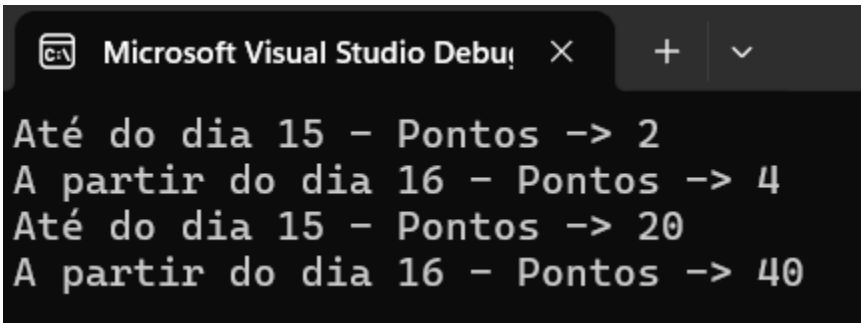
```

1 // See https://aka.ms/new-console-template for more information
2 using ChainResponsabilitySolucao.CalculadoraPontos;
3 using ChainResponsabilitySolucao.Pedido;
4
5 PedidoLanche pedido = new PedidoLanche(21);
6 CalculadoraDePontos calcPontos = new CalculadoraDePontos();
7
8 Console.Write("Até do dia 15 - Pontos -> ");
9 Console.WriteLine(calcPontos.calcularPontos(pedido, 15));
10 Console.Write("A partir do dia 16 - Pontos -> ");
11 Console.WriteLine(calcPontos.calcularPontos(pedido, 16));
12
13 pedido.valor = 100;
14 Console.Write("Até do dia 15 - Pontos -> ");
15 Console.WriteLine(calcPontos.calcularPontos(pedido, 15));
16 Console.Write("A partir do dia 16 - Pontos -> ");
17 Console.WriteLine(calcPontos.calcularPontos(pedido, 16));
18

```

C#

- Nosso teste não modificou a partir da mudança da implementação do padrão de projeto.
- Agora, caso o dono da hamburgueria decida criar novos critérios de pontuação, sejam eles simples ou complexos, a classe CalculadoraDePontos não irá crescer desenfreadamente.
- Será necessário apenas criar novas classes para cada novo critério e os adicionar na cadeia de processamento. Cada critério ficará encapsulado em sua própria classe facilitando a manutenção e entendimento do código.



```

Microsoft Visual Studio Debug Console
Até do dia 15 - Pontos -> 2
A partir do dia 16 - Pontos -> 4
Até do dia 15 - Pontos -> 20
A partir do dia 16 - Pontos -> 40

```



Chain of Responsibility – Implementação Java

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini

Chain of Responsibility

Padrões de Projeto Comportamental II
despanore of respnsioinaneno

Jefferson Antonio Peressni



Acinbrie
Oacholin

Deutaniar
Bistrudiarie
Doserlaties

Amode
Reesemib

Micenialles
Imgoemmemuldo
rexpieretia

Bo
En

Jon
Re
ka



Chain of Responsibility – Consequências

Padrões de Projeto Comportamental II

Prof. Me Jefferson Passerini

Chain of Responsibility

Padrões de Projeto Comportamental II
despanore of respnsioinaneno

Jefferson Antonio Peressni



Acinbrie
Oacholin

Deutanier
Bistrudiarie
Doserlaties

Amode
Reesemib

Micentialles
Imgoemmemuldo
rexpieretia

Bo
En

Jon
Re
ka

Consequências

- Redução de acoplamento
 - O padrão permite que um objeto não precise saber que outro objeto lida com uma solicitação, precisando saber apenas que uma solicitação será tratada de forma adequada.
 - O receptor e o remetente não têm conhecimento explícito um do outro, e um objeto na cadeia não precisa conhecer toda a estrutura da cadeia.
 - Deste modo a cadeia de responsabilidade simplifica as interconexões de objetos. Onde cada objeto mantém uma única referência, ao seu sucessor, ao invés de manter referência a todos os receptores candidatos.
- Maior flexibilidade na atribuição de responsabilidades aos objetos
 - A Cadeia de responsabilidade oferece flexibilidade adicional na distribuição de responsabilidades entre objetos.
 - Sendo possível adicionar ou alterar responsabilidades para manipular uma solicitação, adicionando ou alterando a cadeia em tempo de execução.

Consequências

- A resposta da solicitação não é garantida
 - Como uma solicitação não possui destinatário explícito, não há garantia de que ela será tratada. A solicitação pode chegar ao final da cadeia e não ser tratada. Uma solicitação também pode não ser tratada quando a cadeia estiver configurada de forma incorreta.