Adapter

Padrões de Projeto Estrutural I

Prof. Me Jefferson Passerini

O padrão ***Adapter*** converte a interface de uma classe para outra interface que o cliente espera encontrar. O Adapter permite que classes com interfaces incompatíveis trabalhem juntas.

Motivação – Por que utiilzar?

O padrão ***Adapter*** serve para tornar compatíveis classes que antes não poderiam serem utilizadas em conjunto devido a suas diferenças de interface.

Este padrão é utilizado frequentemente em manutenções de códigos legados, muitas vezes novas classes não são compatíveis com as existentes no código, de modo que um cliente não pode as utilizar de forma transparente.

Situação Problema

Para tornar a explicação mais ilustrativa, suponha que trabalhamos na empresa FreteExpress, ela possui um serviço de agendamento de fretes por aplicativo e faz a cobrança de seus clientes antecipadamente por gateway de pagamentos.

Situação Problema

Os serviços de PagFacil tem sido utilizados desde a fundação da FreteExpress, porém surgiu um novo fornecedor no mercado, a TopPagamentos.

Ela cobra uma taxa fixa maior que a PagFacil por cada pagamento, porém cobra juros menores por parcelamento no cartão de crédito

Valores das taxas cobradas pelos *gateways* (Apenas para ilustração)		
	PagFácil	TopPagamentos
Taxa fixa por pagamento	R$0,40	R$5,00
Juros ao mês (parcelamento)	5%	1%

Situação Problema

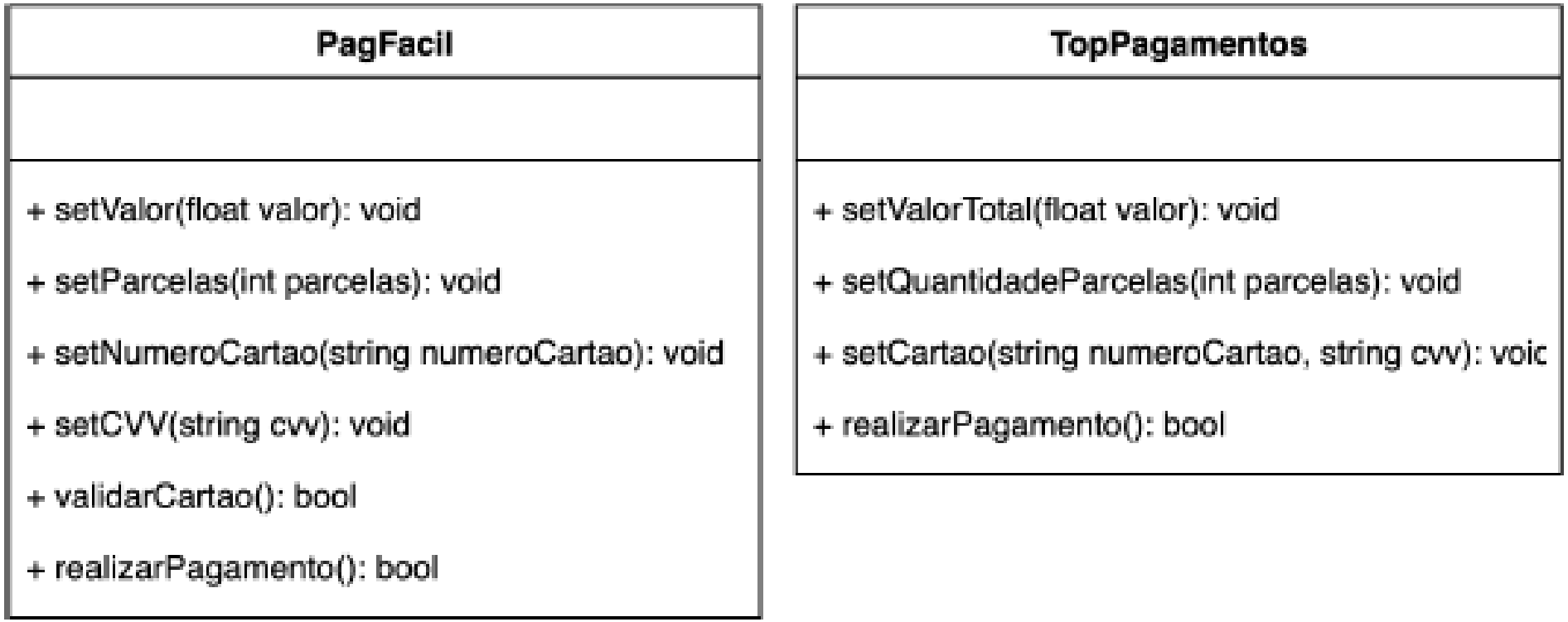
A FreteExpress decidu aderir os serviços da TopPagamentos em conjunto com a PagFacil.

- Para pagamentos à vista deverá ser utilizado o serviço da PagFacil, já que a taxa fixa do pagamento é consideravelmente menor que a da TopPagamentos. Além disso, a taxa de 5% ao mês da PagFacil não será aplicada a pagamentos à vista.

- Já para pagamentos parcelados o serviço da TopPagamentos deverá ser utilizado. Ela cobra uma taxa fixa de R$5,00 mas esse valor se justifica em relação ao baixo juros ao mês (1%) que incide sobre as parcelas

Situação Problema

A documentação dos gateways de pagamento diz que as seguintes classes estão disponíveis para seus clientes (não podem ser alteradas).



Classes oferecidas pelos *gateways* de pagamentos

Situação Problema

O projeto atual implementa a seguinte estrutura consumindo a classe do gateway PagFacil.

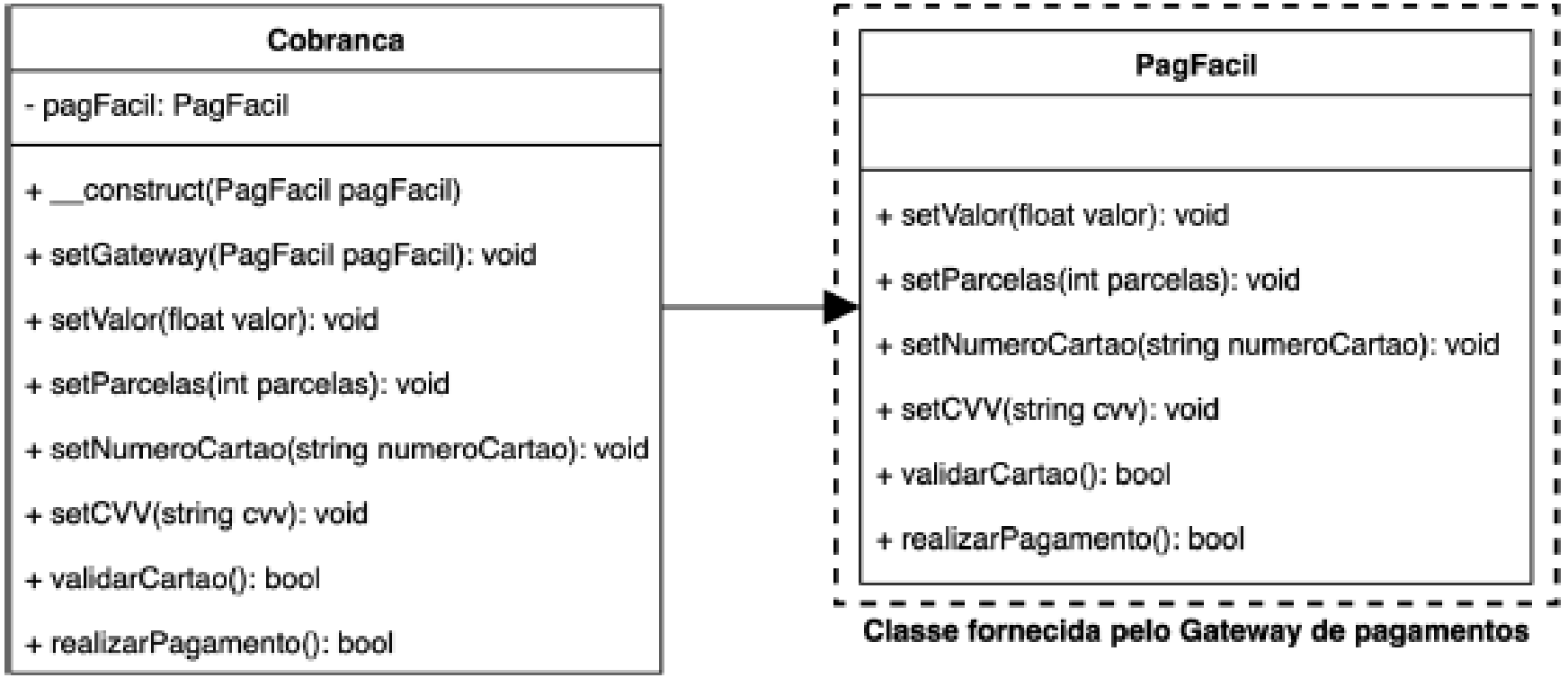


Diagrama de classes do aplicativo da FreteExpress

Situação Problema

A classe Cobranca é a responsável por solicitar a cobrança dos clientes por meio do gateway de pagamentos.

A empresa PagFacil foi o primeiro gateway a ser utilizado, portanto, o projeto da classe Cobrança foi baseado na classe PagFacil

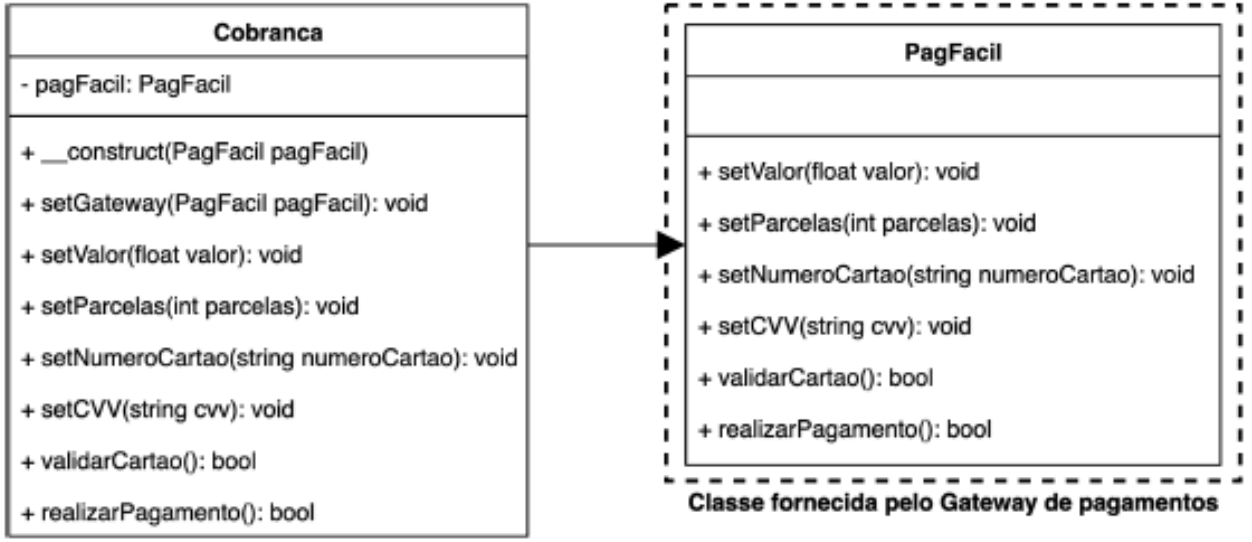


Diagrama de classes do aplicativo da FreteExpress

Situação Problema

No momento a classe Cobranca depende diretamente da classe PagFacil que é uma classe concreta fornecida por um terceiro.

Neste cenário seria mais prudente que a classe Cobrança dependesse de uma abstração, como um interface por exemplo.

Temos um forte acoplamento entre a classe Cobrança e a classe PagFacil.

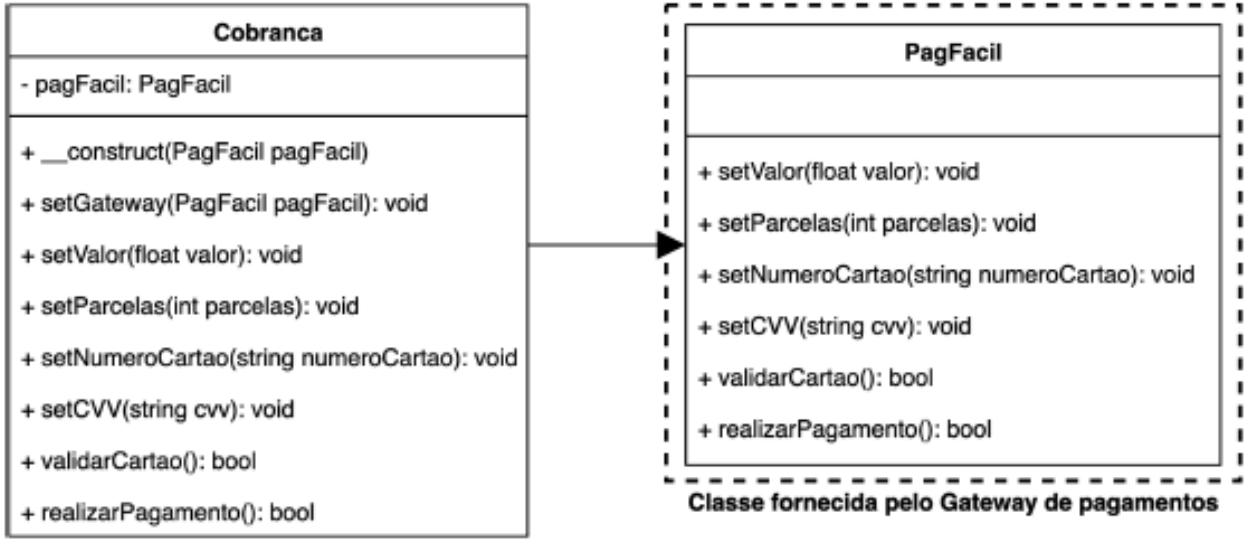


Diagrama de classes do aplicativo da FreteExpress

Padrão Adapter – Quando utilizar?

- Quando existe a necessidade de utilizar uma classe existente e sua interface é diferente da esperada.

- Quando se deseja criar uma classe reutilizável que coopera com classes não relacionadas a ela ou que não foram previstas, ou seja, classes que não necessariamente tem interfaces compatíveis.

- (Somente para adaptadores de Objeto) Quando é necessário usar várias sub-classes existentes, mas é impraticável adaptar sua interface sub-classificando cada uma delas. Um adaptador de objeto pode adaptar a interface de sua superclasse.

Padrão Adapter – Quando utilizar?

- No padrão de projeto Adapter, que é utilizado para fazer a interface de uma classe existente compatível com outra interface que os clientes esperam, existem duas variantes principais:
  - o **Adaptador de Objetos**
  - e o **Adaptador de Classes.**
- Cada um deles aborda o problema de incompatibilidade de interfaces de maneira diferente, aproveitando os recursos específicos da linguagem de programação, como a herança e a composição.

Padrão Adapter – Adaptador de Objetos

- - **Composição**: O adaptador de objetos utiliza composição, ou seja, ele contém uma instância da classe que está adaptando. Este padrão delega chamadas da interface alvo para a instância da classe adaptada.
- - **Relação Dinâmica**: A relação entre o adaptador e a classe adaptada é estabelecida em tempo de execução, proporcionando uma ligação mais flexível e dinâmica.
- - **Maior Flexibilidade**: Como o adaptador de objetos trabalha com instâncias, ele pode adaptar múltiplas instâncias de várias classes (inclusive subclasses da classe original que está sendo adaptada), desde que elas compartilhem a mesma interface. Isso permite que o adaptador de objetos seja mais flexível e reutilizável em diferentes contextos.

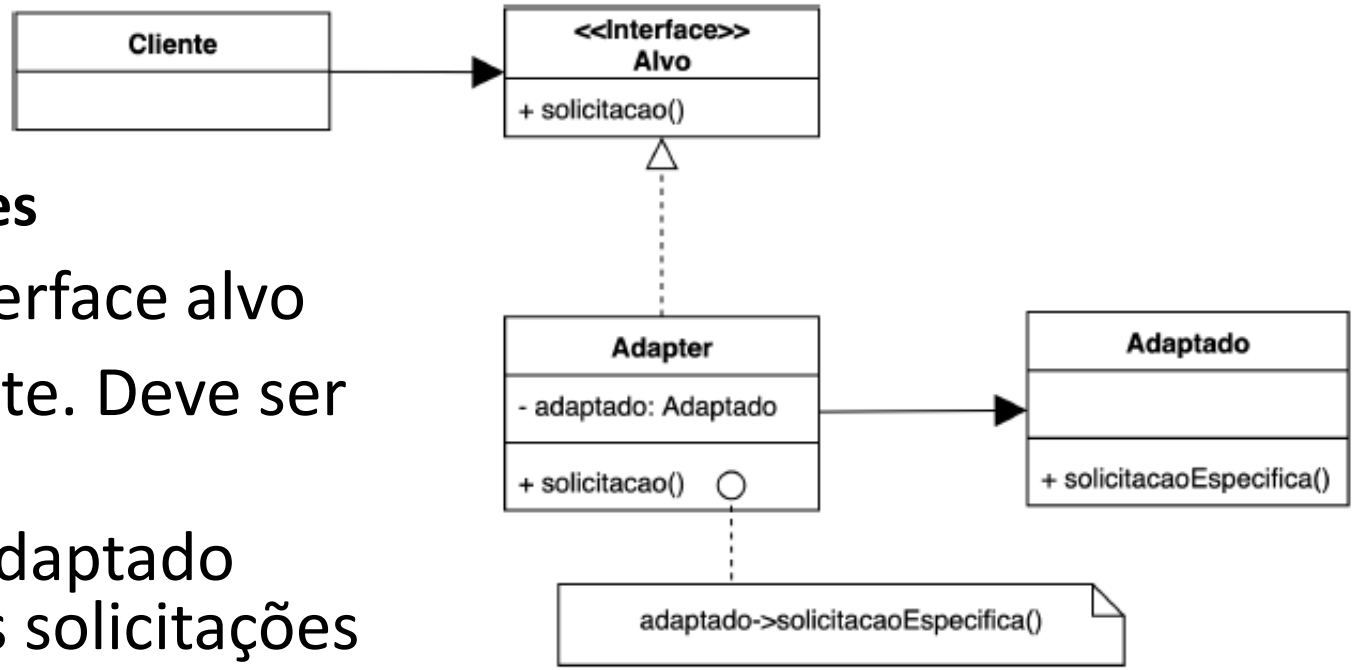


Diagrama de Classes (Adaptador de Objetos)

Padrão Adapter – Componentes

- **Cliente** → é a classe que espera a interface alvo
- **Alvo** → interface esperada pelo Cliente. Deve ser implementada pelo Adapter
- **Adapter** → converte a interface de Adaptado para a interface Alvo. Delega todas as solicitações para Adaptado.
- **Adaptado**→ classe que possui interface incompatível com o cliente e por isso precisa ser adaptada.

Padrão Adapter – Adaptador de Classes

- **Herança Múltipla**: Para implementar um adaptador de classes, a linguagem de programação deve suportar herança múltipla (ou interfaces, em linguagens que diferenciam entre classes e interfaces), pois o adaptador herda tanto da interface que deseja expor quanto da classe que está adaptando.
- - **Relação Estática**: A relação entre o adaptador e a classe adaptada é estabelecida em tempo de compilação, resultando em uma ligação estática.
- - **Extensibilidade Limitada**: Como a herança é utilizada, o adaptador de classes pode ter dificuldades para lidar com futuras modificações tanto na classe adaptada quanto na interface alvo, uma vez que qualquer alteração pode requerer uma revisão do adaptador.

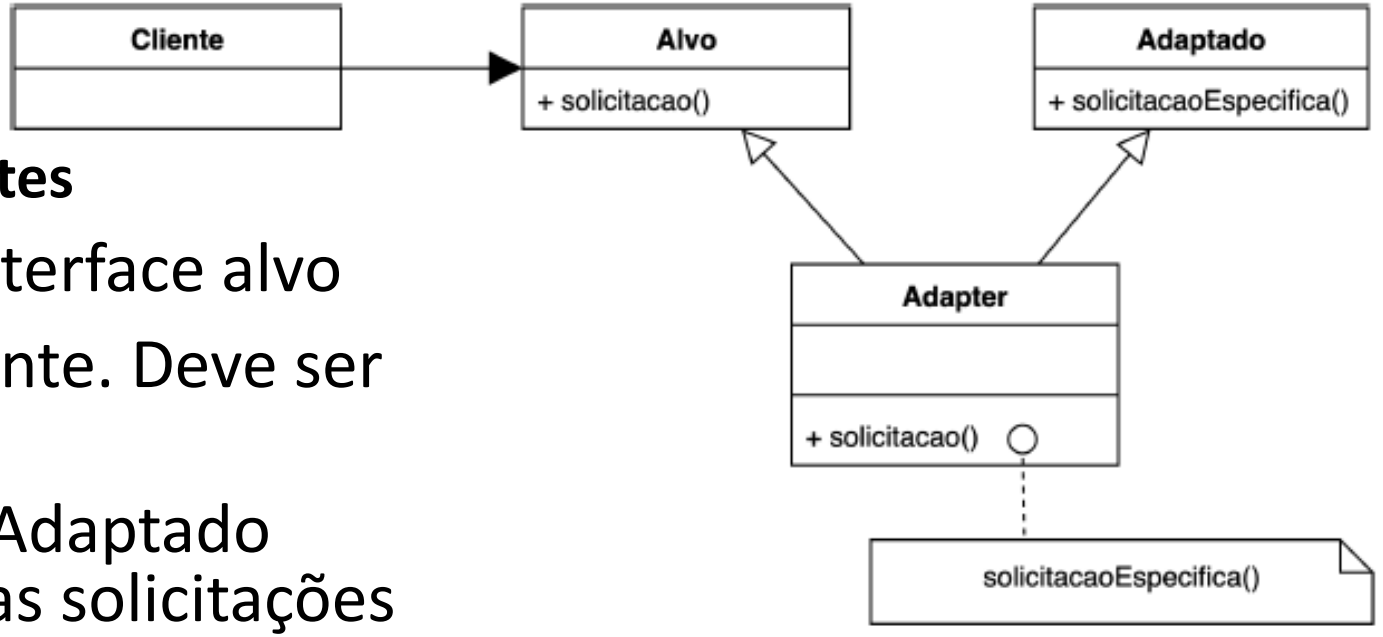


Diagrama de Classes (Adaptador de Classes)

Padrão Adapter – Componentes

- **Cliente** → é a classe que espera a interface alvo
- **Alvo** → interface esperada pelo Cliente. Deve ser implementada pelo Adapter
- **Adapter** → converte a interface de Adaptado para a interface Alvo. Delega todas as solicitações para Adaptado.
- **Adaptado**→ classe que possui interface incompatível com o cliente e por isso precisa ser adaptada.

Padrão Adapter – Escolha entre Adaptador de Objetos e de Classes

- A escolha entre usar um adaptador de objetos ou de classes depende de vários fatores, incluindo as capacidades da linguagem de programação em uso;
- Os requisitos específicos do projeto, e a necessidade de flexibilidade versus o desejo por uma relação mais direta e menos indireta entre a interface alvo e a classe adaptada.
- O adaptador de objetos, sendo mais flexível e dinâmico, é frequentemente preferido em situações onde a composição é mais adequada do que a herança.

Padrão Adapter – Escolha entre Adaptador de Objetos e de Classes

- Um adaptador de objetos (por composição):
  - Permite que um único adaptador funcione com muitos adaptados, ou seja, o próprio adaptado e todas as suas subclasses (se houver). O adaptador também pode adicionar funcionalidade a todos os adaptados de um só vez.
  - Dificulta sobrescrever (override) o comportamento de um Adaptado. Isso exigirá um sub-classificação do Adaptado e fará com que o adaptador se refirá à subclasse em vez do próprio Adaptado.

Padrão Adapter – Escolha entre Adaptador de Objetos e de Classes

- Um adaptador de classes (por herança):
  - Adapta a classe Adaptado ao Alvo comprometendo-se com uma classe concreta Adapter. Como consequência, um adaptador de classe não funcionará quando queremos adaptar uma classe e todas as suas subclasses.
  - Permite que o Adapter substitua parte do comportamento do Adaptado, pois o Adapter é um subclasse Adaptado.
  - Introduz apenas um objeto, e não é necessário nenhum direcionamento adicional do ponteiro (manter a referência para a classe Adaptado).

Padrão Adapter – Consequências

- A quantidade de trabalho que o Adapter faz depende de quão semelhante a interface do Alvo é do Adaptado.

- Há uma variação no volume de trabalhos que executar para adaptar uma classe para a interface alvo, vai desde a simples conversão de interface, por exemplo, alterando os nomes das solicitações, até o suporte a um conjunto de solicitações totalmente diferentes.

Padrão Adapter – Consequências

- Uma classe é mais reutilizável quanto menor for a quantidade de suposições que outras classes devem fazer para usá-la, o Adapter elimina tais suposições;

- Em outras palavras, um adapter de interface permite incorporar uma classe em sistemas existentes que podem esperar interfaces diferentes para a classe.

Padrão Adapter – Consequências

- Um problema em potencial com adaptadores é que eles não são transparentes para todos os clientes;
- Um objeto adaptado não oferece a interface do objeto original, portanto, ele não pode ser usado onde o objeto original é esperado.
- Adaptadores bidirecionais podem fornecer essa transparência. Especificamente são uteis quando dois clientes diferentes precisam exibir um objeto de maneira diferente.
- Para implementar um adaptador bidirecional que atua simultaneamente como interface antiga e a interface nova, basta que ele implemente ambas as interfaces envolvidas. Em alguns casos é preciso utilizar herança multipla.

Padrão Adapter – Consequências

- Um problema em potencial com adaptadores é que eles não são transparentes para todos os clientes;
- Um objeto adaptado não oferece a interface do objeto original, portanto, ele não pode ser usado onde o objeto original é esperado.
- Adaptadores bidirecionais podem fornecer essa transparência. Especificamente são uteis quando dois clientes diferentes precisam exibir um objeto de maneira diferente.
- Para implementar um adaptador bidirecional que atua simultaneamente como interface antiga e a interface nova, basta que ele implemente ambas as interfaces envolvidas. Em alguns casos é preciso utilizar herança multipla.

Adapter - Solução

Padrões de Projeto Estrutural I

Prof. Me Jefferson Passerini

Situação Problema

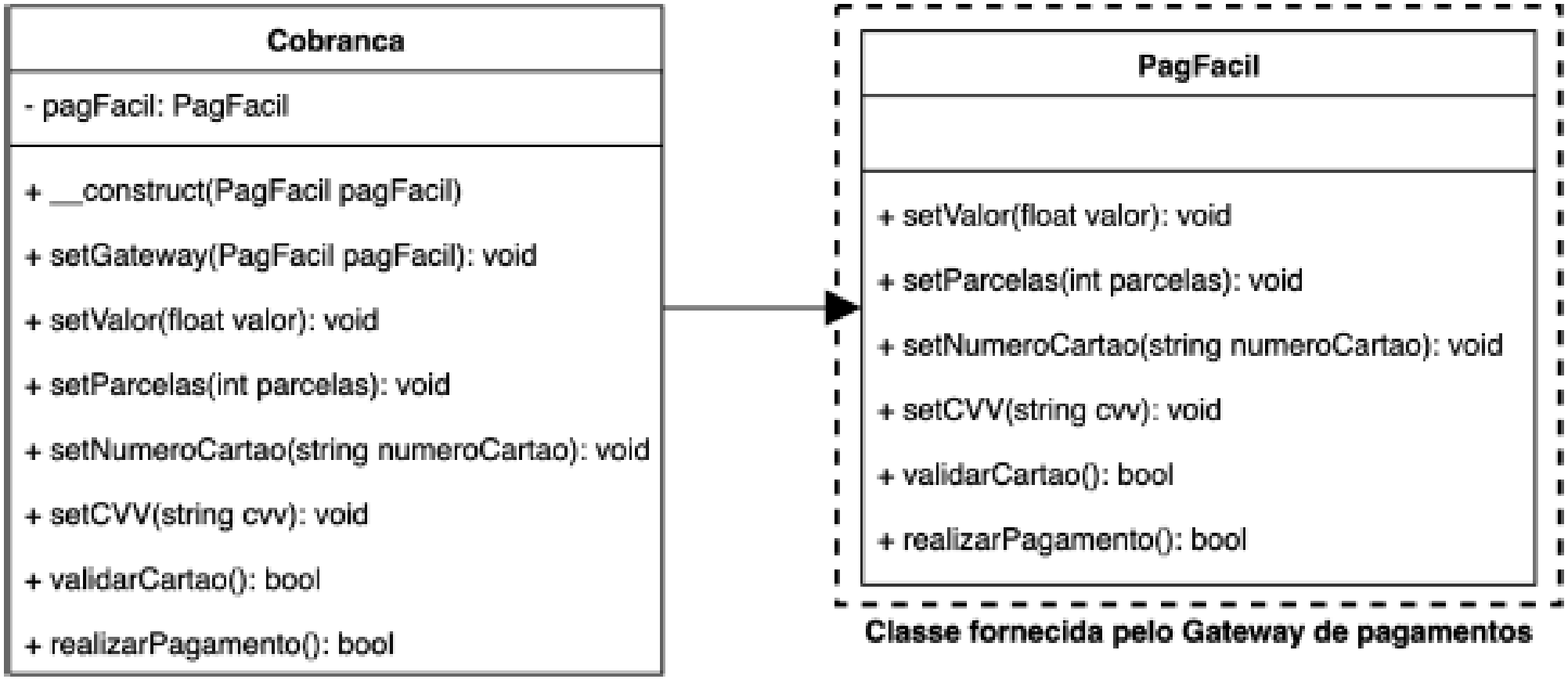
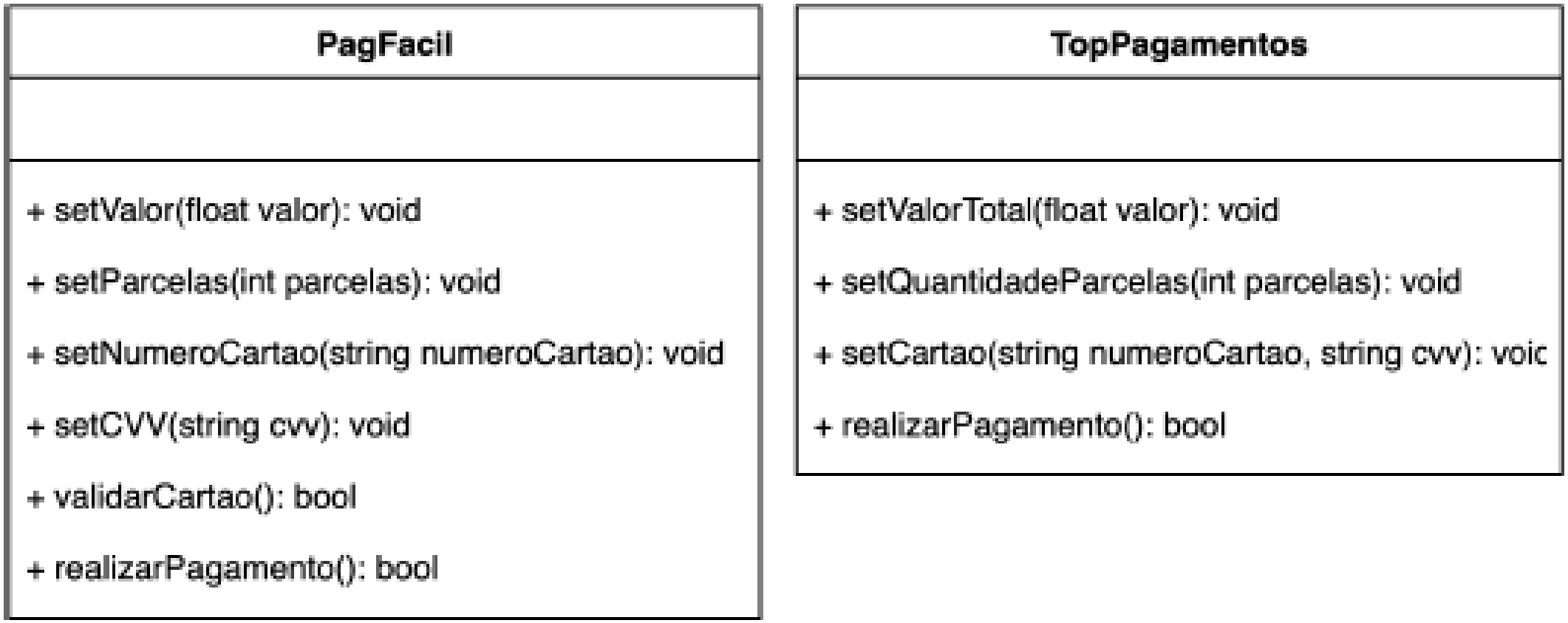


Diagrama de classes do aplicativo da FreteExpress

Situação Problema – novo gateway de pagamentos



Classes oferecidas pelos gateways de pagamentos

Comparação de equivalência entre os métodos da interface PagFacil e TopPagamentos

PagFacil	TopPagamentos
setValor(float valor): void	setValorTotal(float valor): void
setParcelas(int parcelas): void	setQuantidadeParcelas(int parcelas): void
setNumeroCartao(string numeroCartao): void	setCartao(string numero, string cvv): void
setCVV(string cvv): void	
validarCartao(): bool	
realizarPagamento(): bool	realizarPagamento(): bool

Padrões de Projetos Estrutural – Adapter

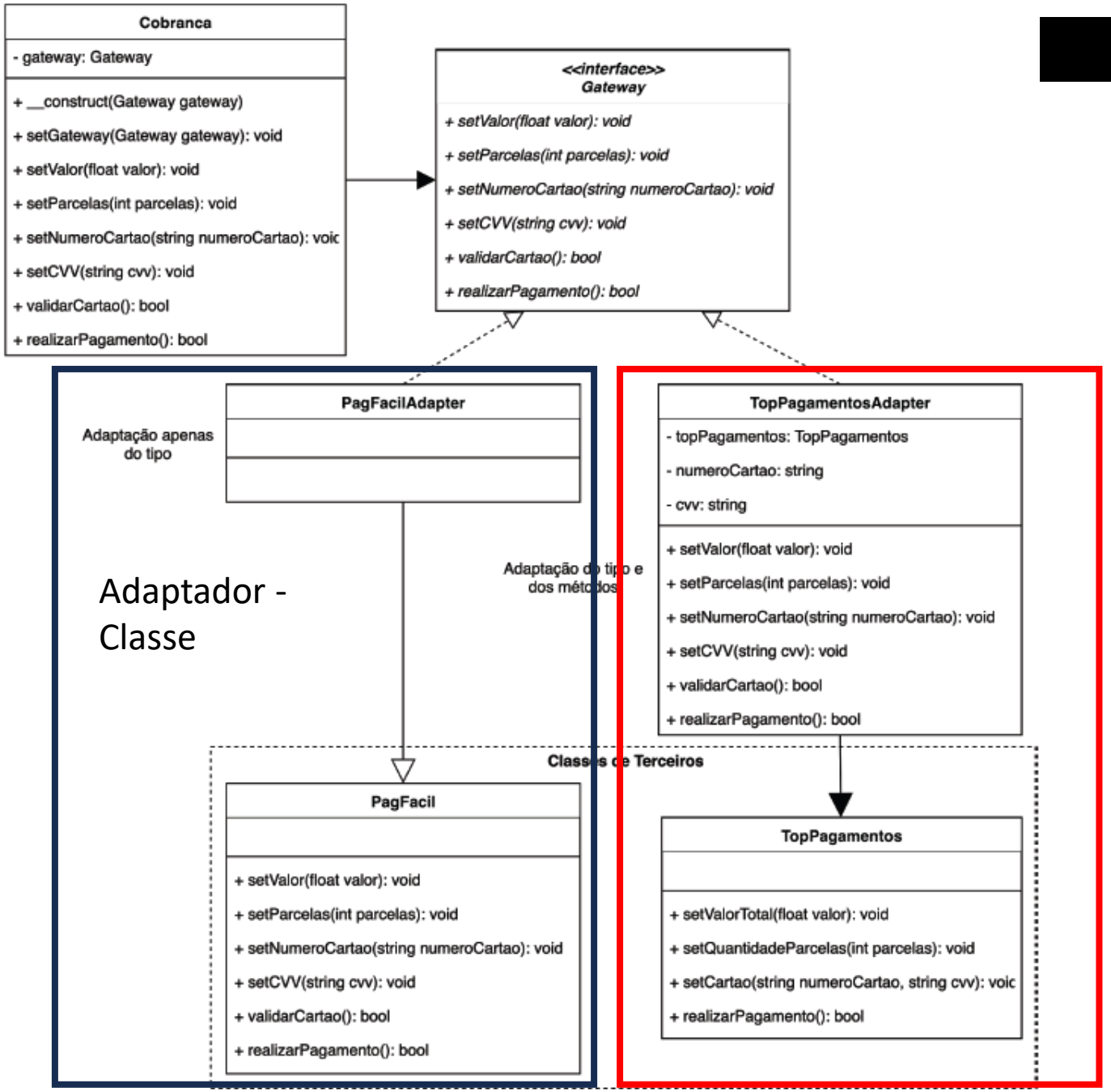


Diagrama de classes do aplicativo da FreteExpress após aplicação do adapter

Adapter – Solução – C#

Padrões de Projeto Estrutural I

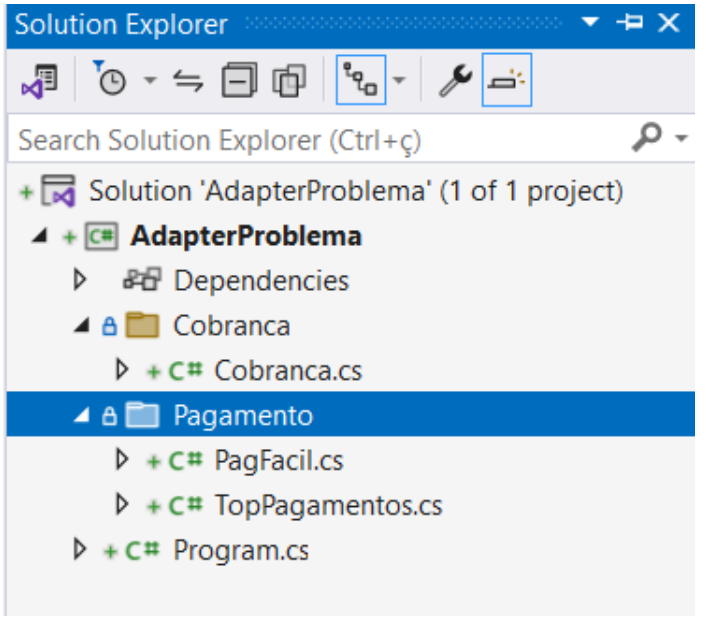
Prof. Me Jefferson Passerini

O projeto atual que utiliza apenas a classe PagFacil possui a estrutura representada na figura (a).

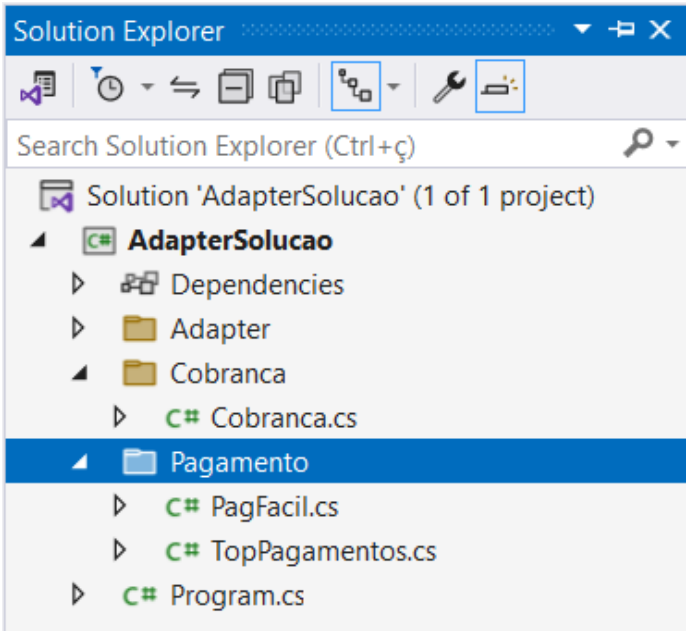
Crie as novas pastas para comportar a estrutura da refatoração que iremos realizar no projeto.

Criar a pasta → Adapter

a)



b)



Vamos criar uma interface para intermediar a relação entre a classe cobrança e as classes gateways de pagamentos.

Para minimizar os impactos causados pela refatoração, vamos manter a mesma nomenclatura de métodos da classe PagFacil nos métodos da interface

Vamos refatorar primeiro a classe PagFacil pois já existia no sistema e garantir que continue funcionando

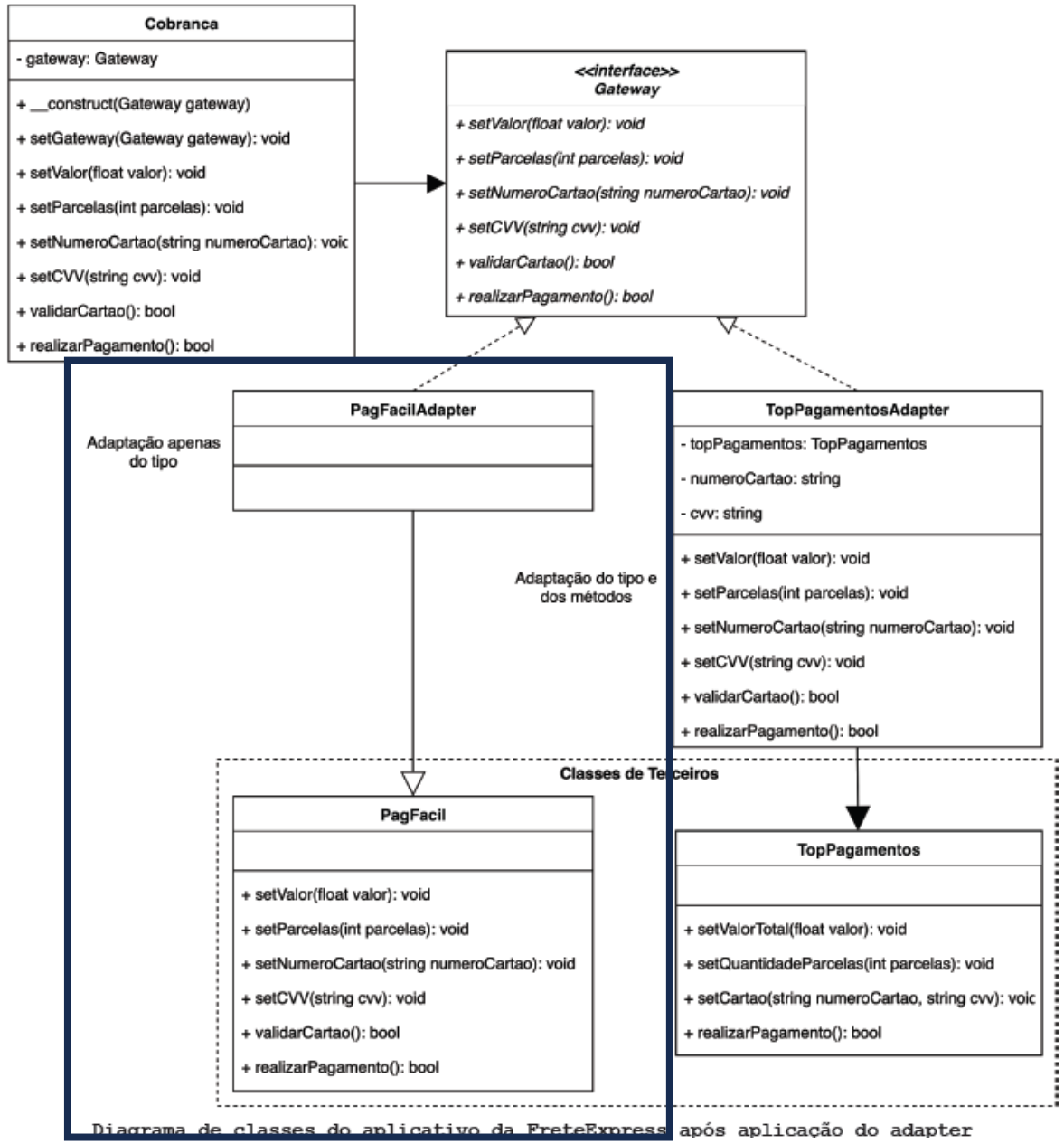
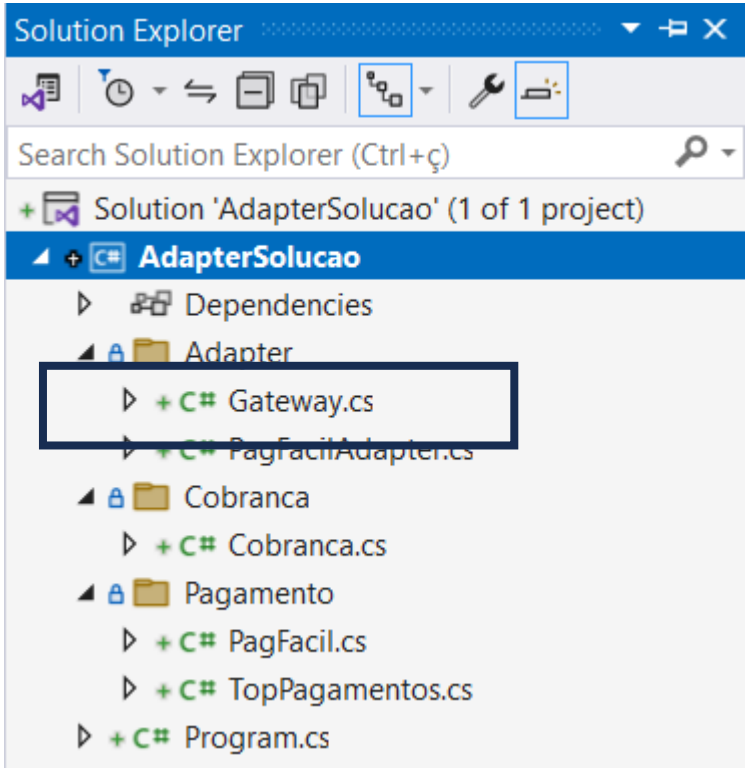


Diagrama de classes do aplicativo da FreteExpress após aplicação do adapter

Na pasta Adapter vamos criar a Interface Gateway;

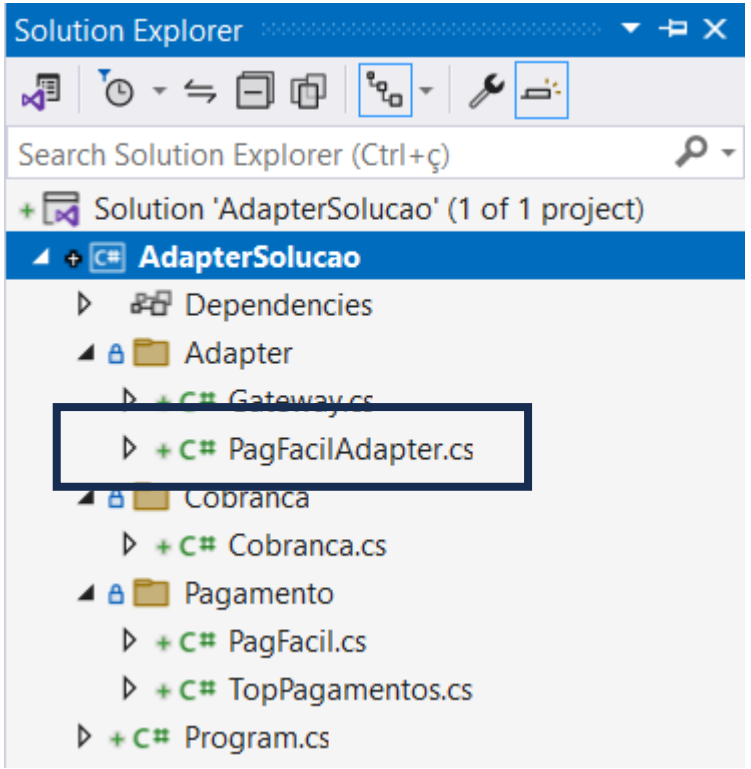
E desenvolva o código para essa interface.




```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace AdapterSolucao.Adapter
8   {
        4 references
9       public interface Gateway
10      {
            3 references
11          public void setValor(double valor);
            3 references
12          public void setParcelas(int parcelas);
            3 references
13          public void setNumeroCartao(string numeroCartao);
            3 references
14          public void setCVV(string cvv);
            4 references
15          public bool validarCartao();
            4 references
16          public bool realizarPagamento();
17      }
18  }
19
```

Agora na pasta Adapter vamos criar a classe PagFacilAdapter;

Como nossa Gateway replica a interfaces da classe PagFacil

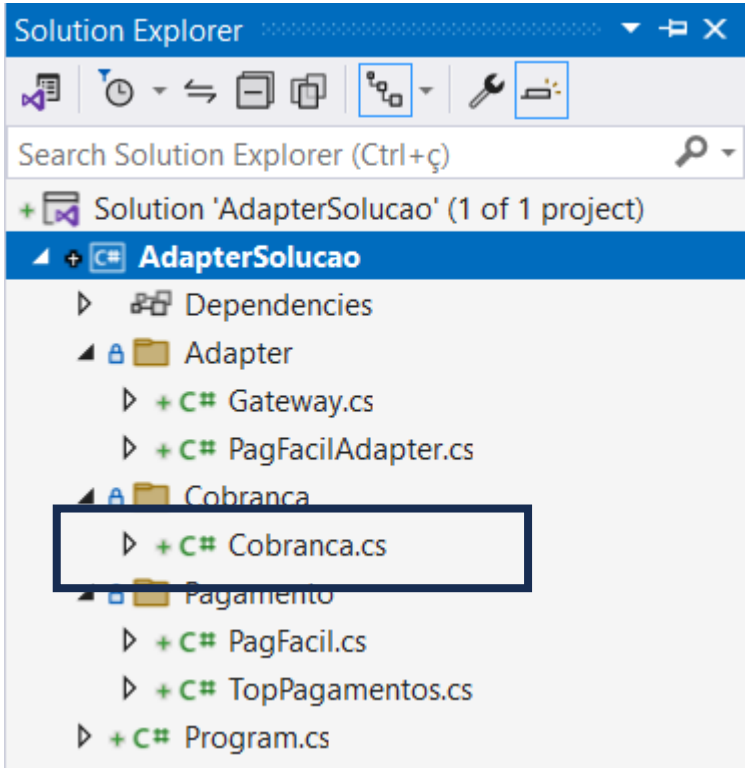



```
1   using AdapterSolucao.Pagamento;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace AdapterSolucao.Adapter
9   {
        1 reference
10      public class PagFacilAdapter : PagFacil, Gateway
11      {
12
13      }
14  }
```

Apenas extendemos (herança) a classe original **PagFacil** para nossa **PagFacilAdapter**;

E fazemos ela implementar a interface **Gateway**.

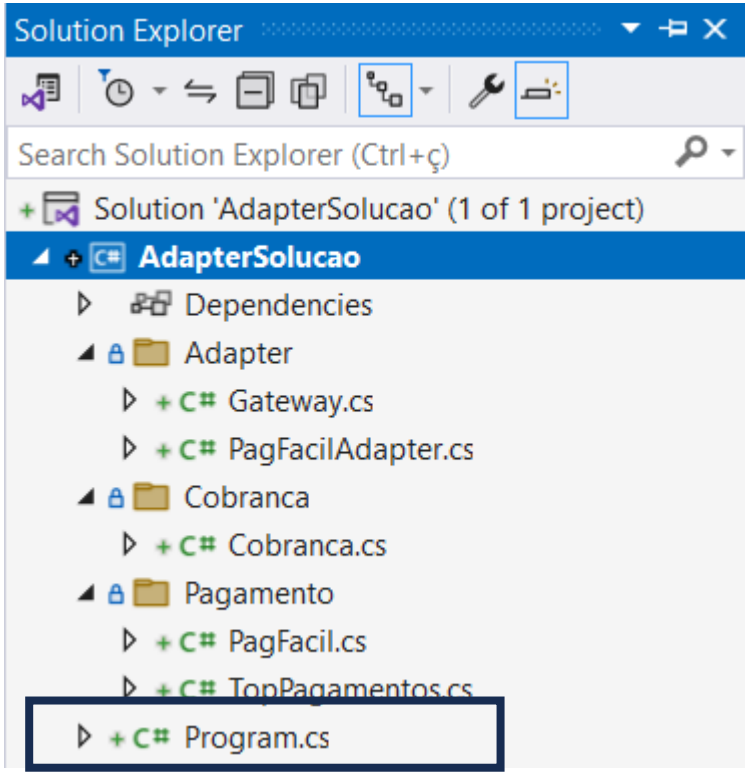
Na classe Cobrança existente vamos deixar as linhas indicadas como no exemplo.



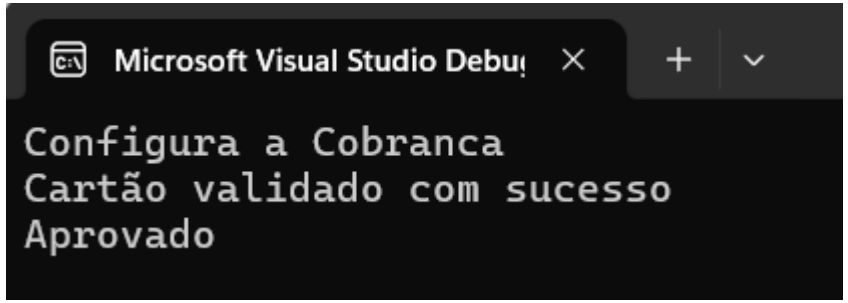

```
1   using ...
8
9   namespace AdapterSolucao.Cobranca
10  {
11      public class Cobranca: Gateway
12      {
13          public Gateway gateway { get; set; }
14          public double valor { get; set; }
15          public int parcelas { get; set; }
16          public string numeroCartao { get; set; }
17          public string cvv { get; set; }
18
19          public Cobranca(){ }
20
21          public void setGateway(Gateway gateway)
22          {
23              this.gateway = gateway;
24              gateway.setValor(valor);
25              gateway.setNumeroCartao(numeroCartao);
26              gateway.setCVV(cvv);
27              gateway.setParcelas(parcelas);
28          }
29
```

Agora faça uma pequena alteração na classe Program.

Altere a linha indicada!




```
1   // See https://aka.ms/new-console-template for
2   using AdapterSolucao.Adapter;
3   using AdapterSolucao.Cobranca;
4   using AdapterSolucao.Pagamento;
5
6   Console.WriteLine("Configura a Cobranca");
7   Cobranca cobranca = new Cobranca();
8   cobranca.setValor(100);
9   cobranca.setNumeroCartao("999999999999");
10  cobranca.setCVV("163");
11  cobranca.setGateway(new PagFacilAdapter());
12  cobranca.validarCartao();
13  cobranca.realizarPagamento();
14
```



Refatoramos nossa Gateway PagFacil e o sistema permanece funcionando.

Agora vamos implementar o acesso ao novo meio de pagamentos **TopPagamentos**

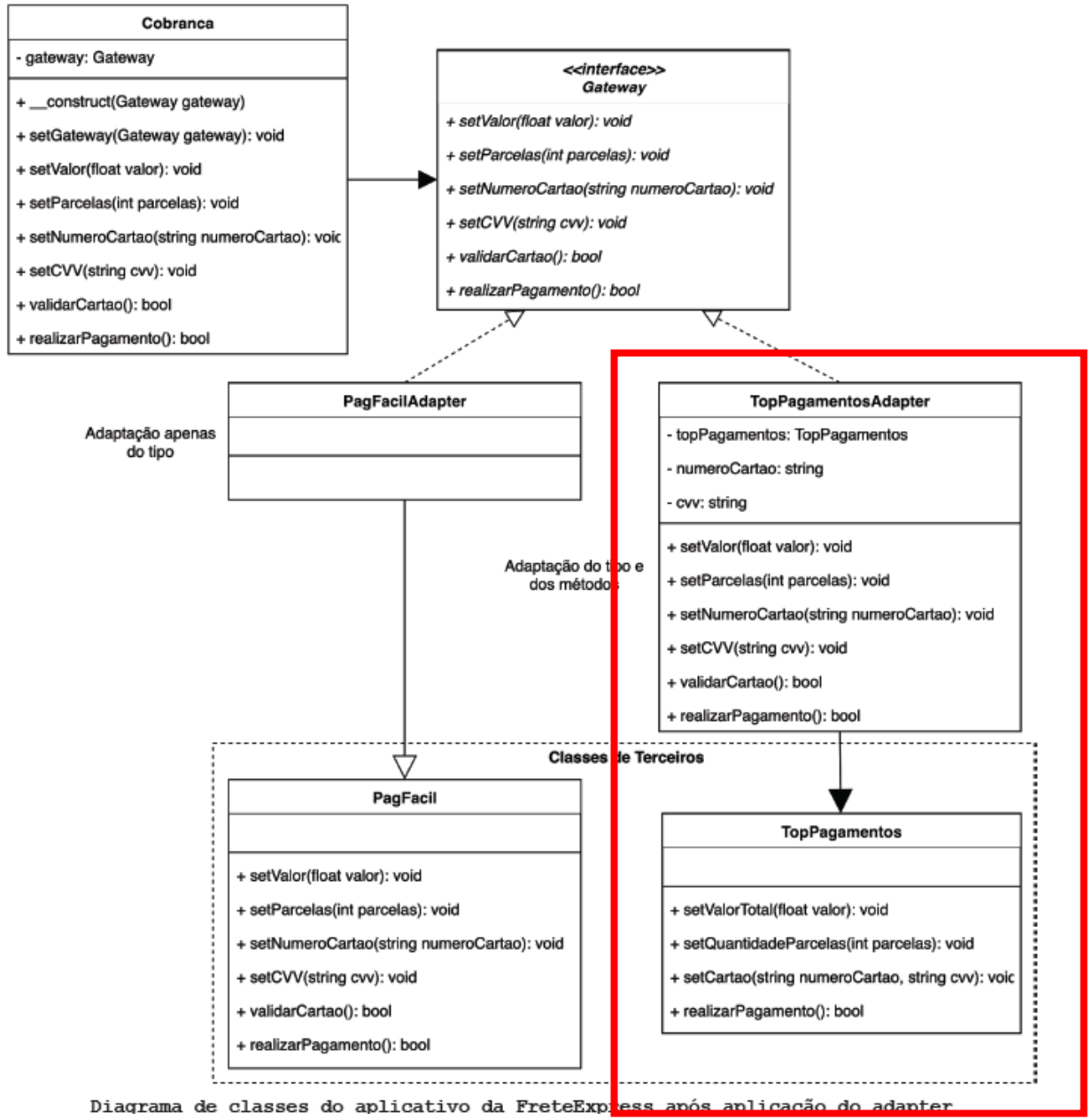
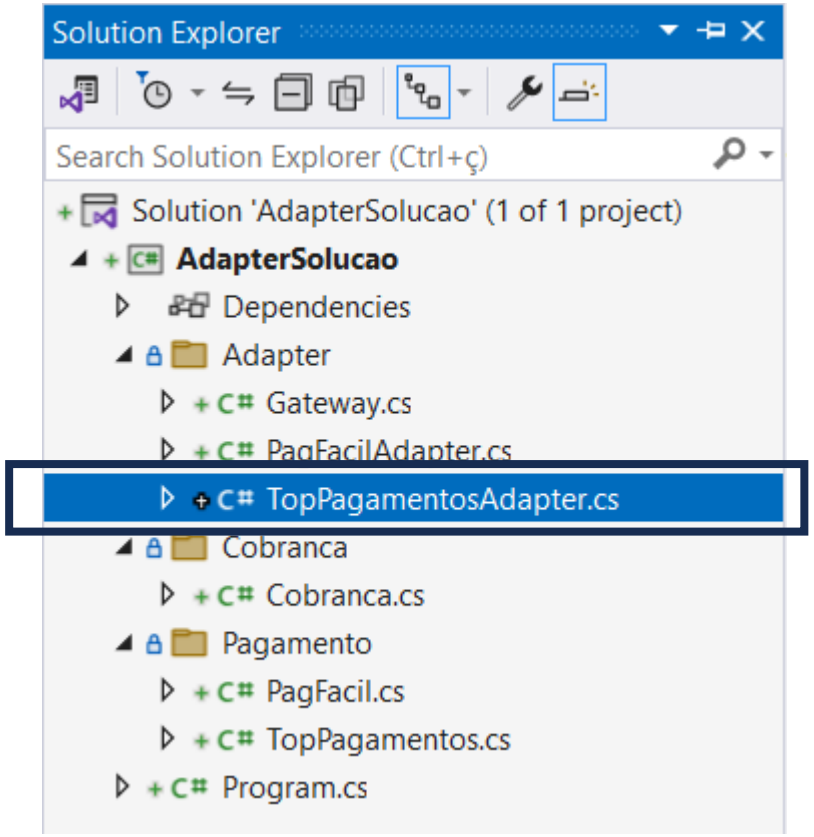


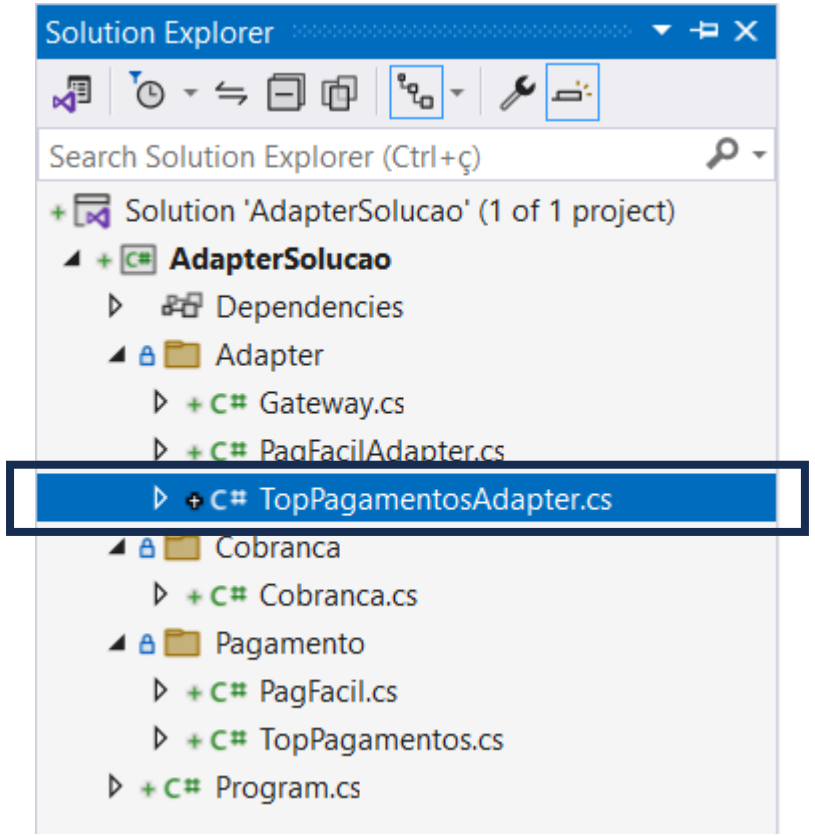
Diagrama de classes do aplicativo da FreteExpress após aplicação do adapter

Primeiro passo é criar a classe **TopPagamentosAdapter** na pasta Adapter.



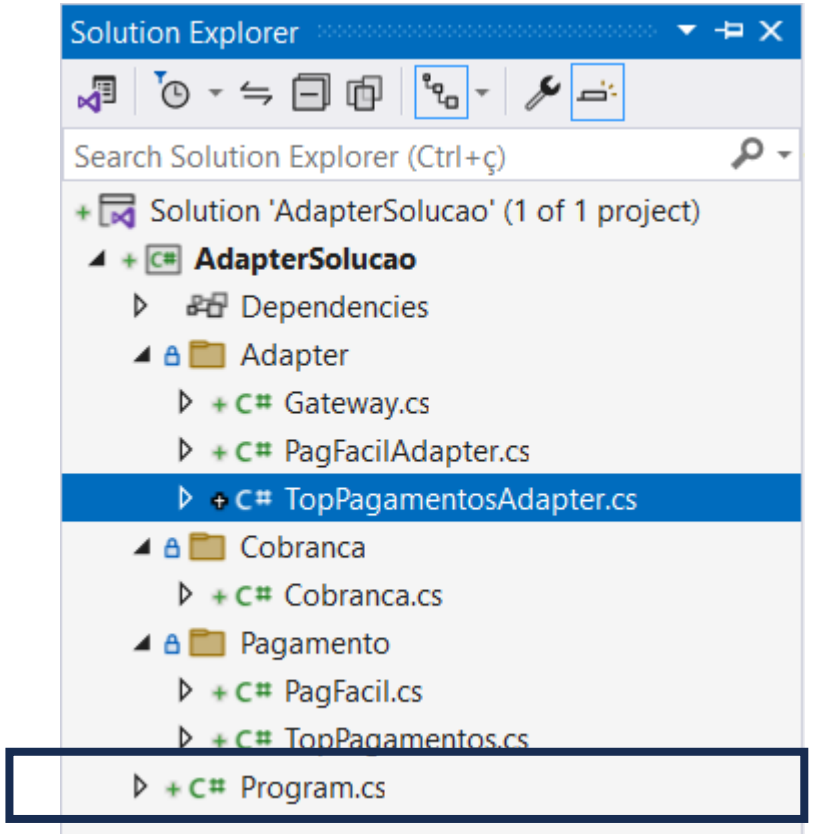

```
1   using ...
7
8   namespace AdapterSolucao.Adapter
9   {
        2 references
10      public class TopPagamentosAdapter : Gateway
11      {
12          public TopPagamentos topPagamentos;
                2 references
13          public string numeroCartao { get; set; }
                2 references
14          public string cvv { get; set; }
                2 references
15          public double valor {  get; set; }
                2 references
16          public int parcelas { get; set; }
17
                1 reference
18          public TopPagamentosAdapter() {
19              this.topPagamentos = new TopPagamentos();
20          }
21
                2 references
22          public void setCVV(string cvv)
23          {
24              this.cvv = cvv;
25          }
26
                2 references
27          public void setNumeroCartao(string numeroCartao)
28          {
29              this.numeroCartao = numeroCartao;
30          }
31
```

Continuação da implementação da classe **TopPagamentosAdapter** na pasta Adapter.

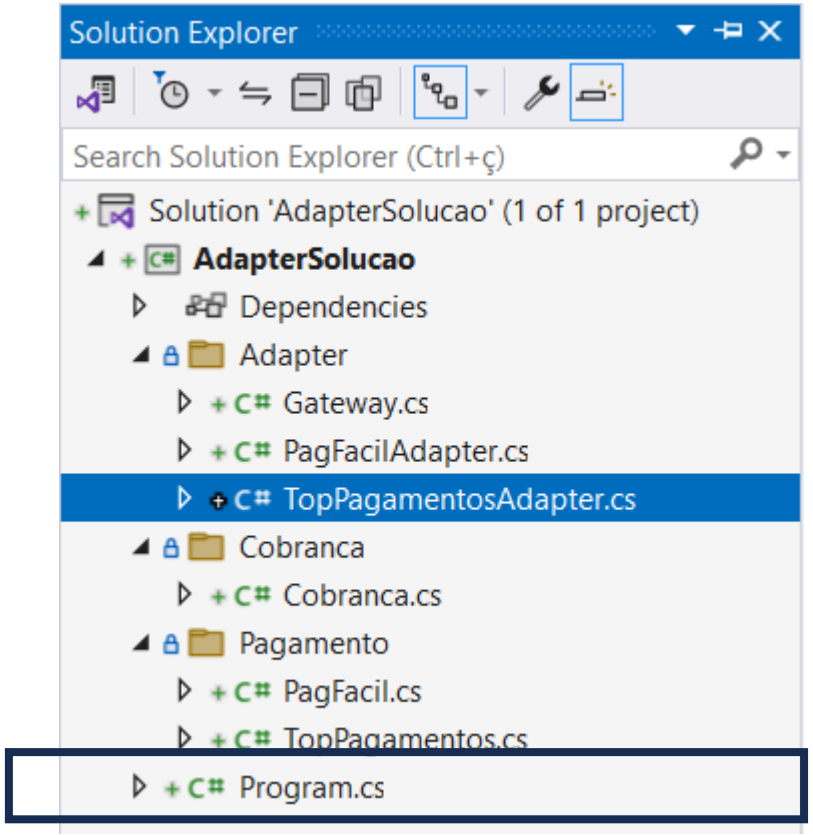



```
2 references
public void setParcelas(int parcelas)
{
    this.parcelas = parcelas;
}

2 references
public void setValor(double valor)
{
    this.valor = valor;
}

2 references
public bool validarCartao()
{
    return true;
}

2 references
public bool realizarPagamento()
{
    this.topPagamentos.setValorTotal(this.valor);
    this.topPagamentos.setCartao(this.numeroCartao, this.cvv);
    this.topPagamentos.setQuantidadeParcelas(this.parcelas);

    return this.topPagamentos.realizarPagamento();
}
}
}
```

Agora implemente o teste na classe Program




```
1   // See https://aka.ms/new-console-template for m
2   using AdapterSolucao.Adapter;
3   using AdapterSolucao.Cobranca;
4   using AdapterSolucao.Pagamento;
5
6   Console.WriteLine("Configura a Cobranca");
7   Cobranca cobranca = new Cobranca();
8   cobranca.setValor(100);
9   cobranca.setNumeroCartao("999999999999");
10  cobranca.setCVV("163");
11  //teste PagFacil
12  cobranca.setGateway(new PagFacilAdapter());
13  cobranca.validarCartao();
14  cobranca.realizarPagamento();
15
16  //teste TopPagamentos
17  //teste PagFacil
18  cobranca.setGateway(new TopPagamentosAdapter());
19  cobranca.validarCartao();
20  cobranca.realizarPagamento();
21
```




```
Configura a Cobranca
Cartão validado com sucesso
Aprovado
Cartão validado com sucesso
Aprovado
```

Ambos os Gateways de pagamento estão funcionando.

Adapter – Solução – Java

Padrões de Projeto Estrutural I

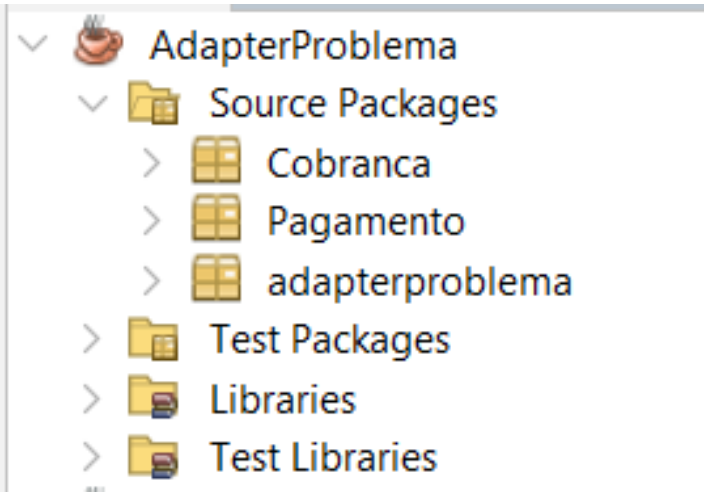
Prof. Me Jefferson Passerini

O projeto atual que utiliza apenas a classe PagFacil possui a estrutura representada na figura (a).

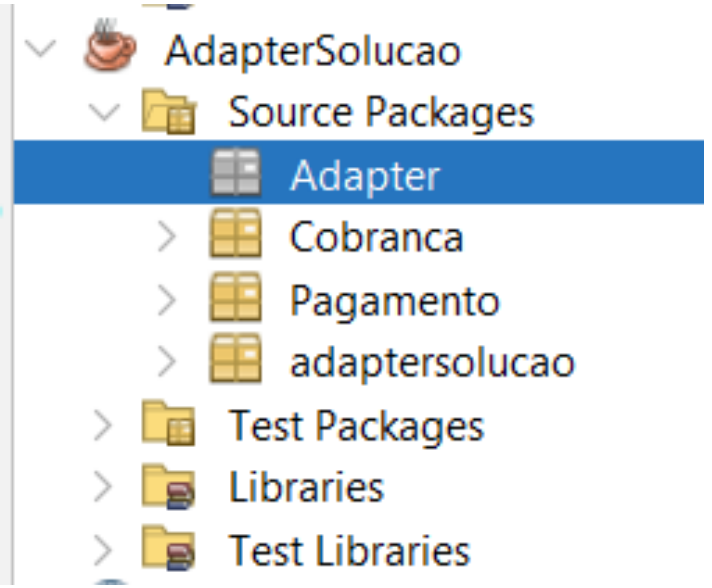
Crie as novas pastas para comportar a estrutura da refatoração que iremos realizar no projeto.

Criar a pasta → Adapter

a)



b)



Vamos criar uma interface para intermediar a relação entre a classe cobrança e as classes gateways de pagamentos.

Para minimizar os impactos causados pela refatoração, vamos manter a mesma nomenclatura de métodos da classe PagFacil nos métodos da interface

Vamos refatorar primeiro a classe PagFacil pois já existia no sistema e garantir que continue funcionando

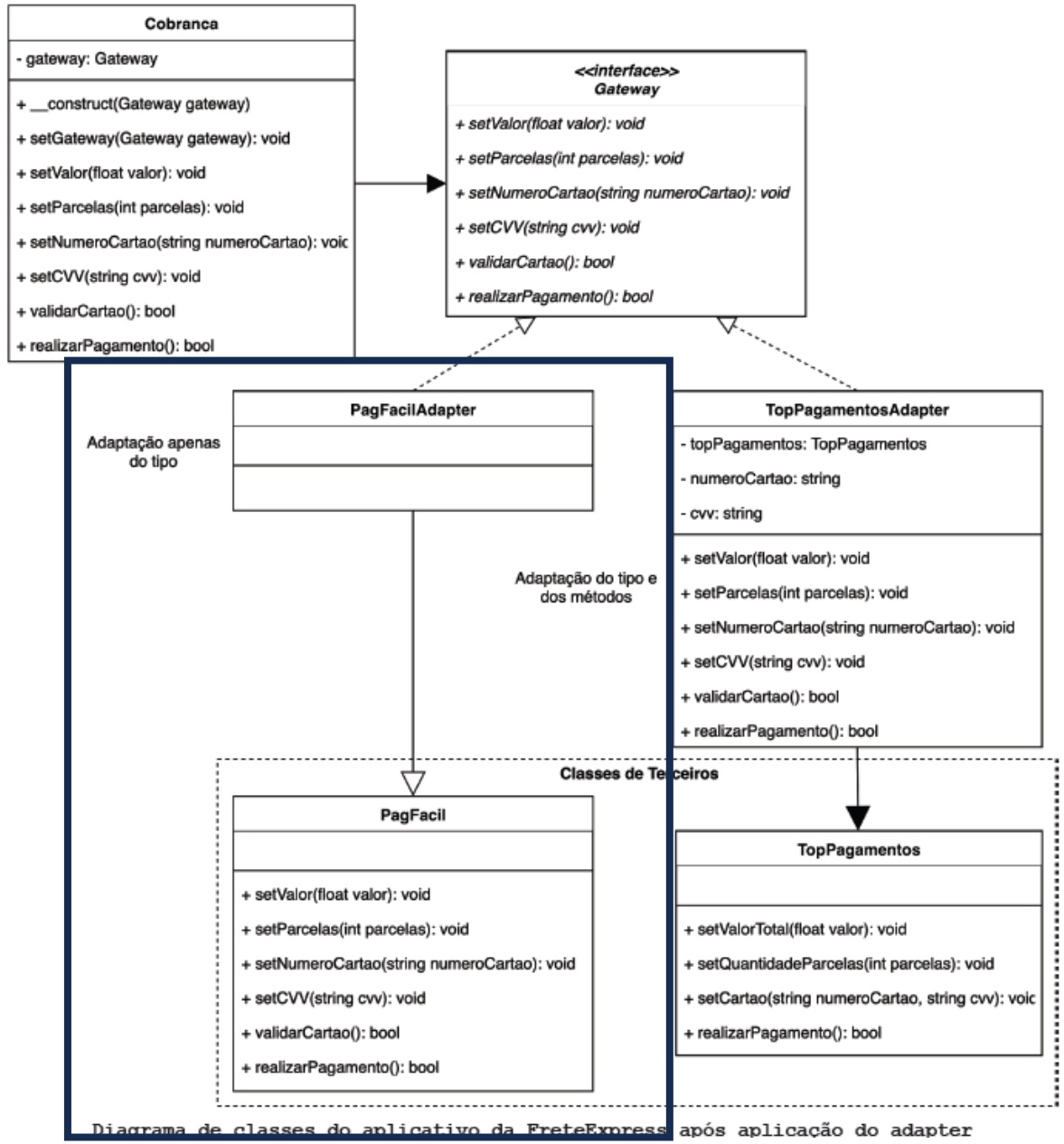
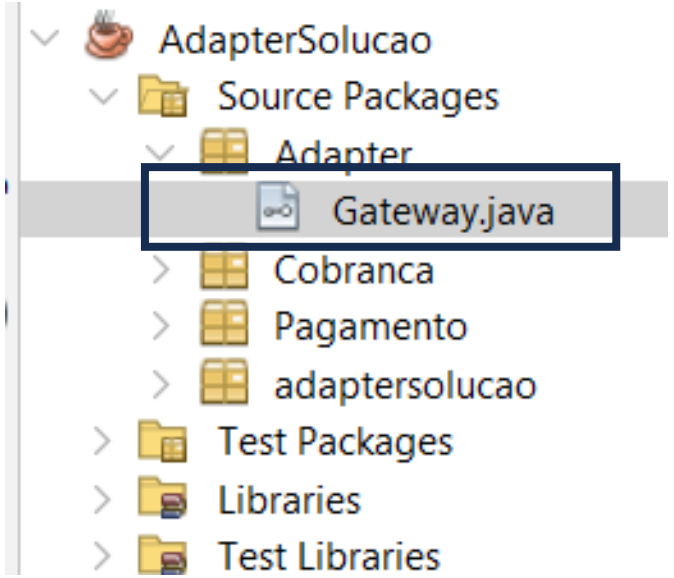


Diagrama de classes do aplicativo da FreteExpress após aplicação do adapter

Na pasta Adapter vamos criar a Interface Gateway;

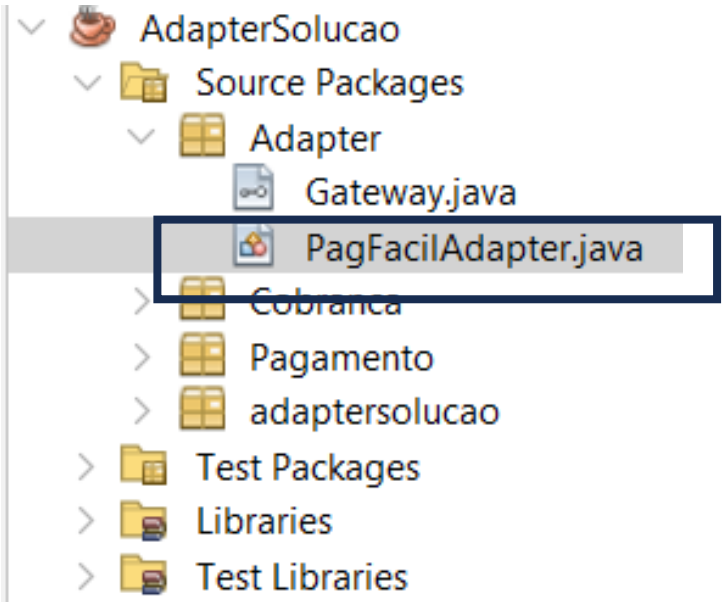
E desenvolva o código para essa interface.




```java
package Adapter;

/**
 *
 * @author jeffe
 */
public interface Gateway {

    public void setValor(double valor);
    public void setParcelas(int parcelas);
    public void setNumeroCartao(String numeroCartao);
    public void setCVV(String cvv);
    public boolean validarCartao();
    public boolean realizarPagamento();

}
```

Agora na pasta Adapter vamos criar a classe PagFacilAdapter;

Como nossa Gateway replica a interfaces da classe PagFacil

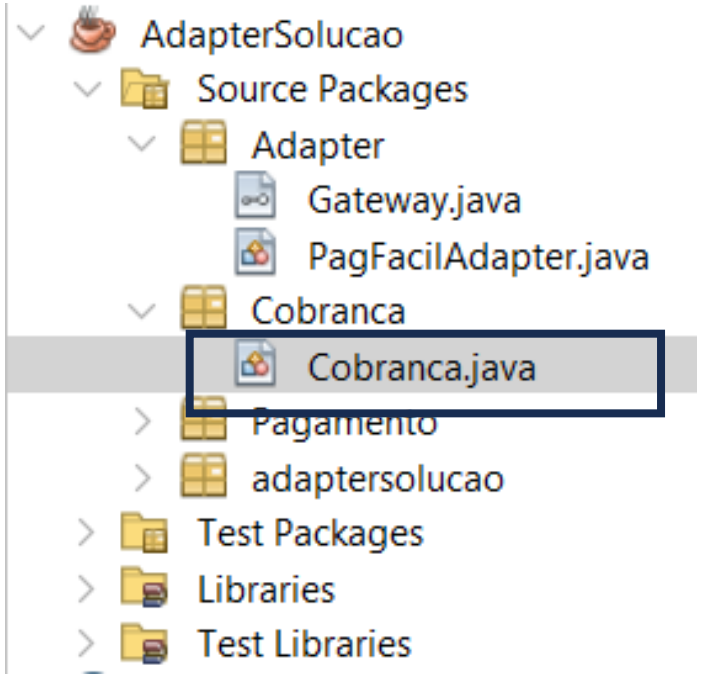



```
1  ...4 lines
5  package Adapter;
6
7  import Pagamento.PagFacil;
8
9  /**
10  *
11  * @author jeffe
12  */
13 public class PagFacilAdapter extends PagFacil
14         implements Gateway {
15
16 }
```

Apenas extendemos (herança) a classe original **PagFacil** para nossa **PagFacilAdapter**;

E fazemos ela implementar a interface **Gateway**.

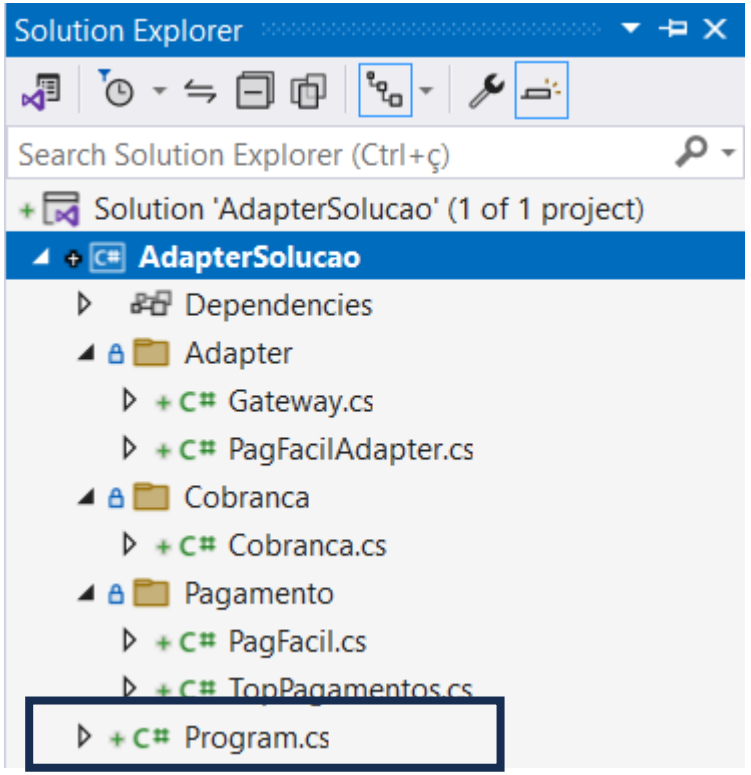
Na classe Cobrança existente vamos deixar as linhas indicadas como no exemplo.




```
14  public class Cobranca implements Gateway {
15
16      public Gateway gateway;
17      public double valor;
18      public int parcelas;
19      public String numeroCartao;
20      public String cvv;
21
22      public Cobranca() { }
23
24      public void setGateway(Gateway gateway)
25      {
26          this.gateway = gateway;
27          gateway.setValor(valor);
28          gateway.setNumeroCartao(numeroCartao);
29          gateway.setCVV(cvv);
30          gateway.setParcelas(parcelas);
31      }
```

Agora faça uma pequena alteração na classe Program.

Altere a linha indicada!




```
20      public static void main(String[] args) {
21          System.out.println("Configura a Cobranca");
22
23          Cobranca cobranca = new Cobranca();
24          cobranca.setValor(100);
25          cobranca.setNumeroCartao("999999999999");
26          cobranca.setCVV("163");
27          cobranca.setGateway(new PagFacilAdapter());
28          cobranca.validarCartao();
29          cobranca.realizarPagamento();
30      }
```

```
run:
Configura a Cobranca
Cartão validado com sucesso
Aprovado
BUILD SUCCESSFUL (total time: 0 seconds)
```

Refatoramos nossa Gateway PagFacil e o sistema permanece funcionando.

Agora vamos implementar o acesso ao novo meio de pagamentos **TopPagamentos**

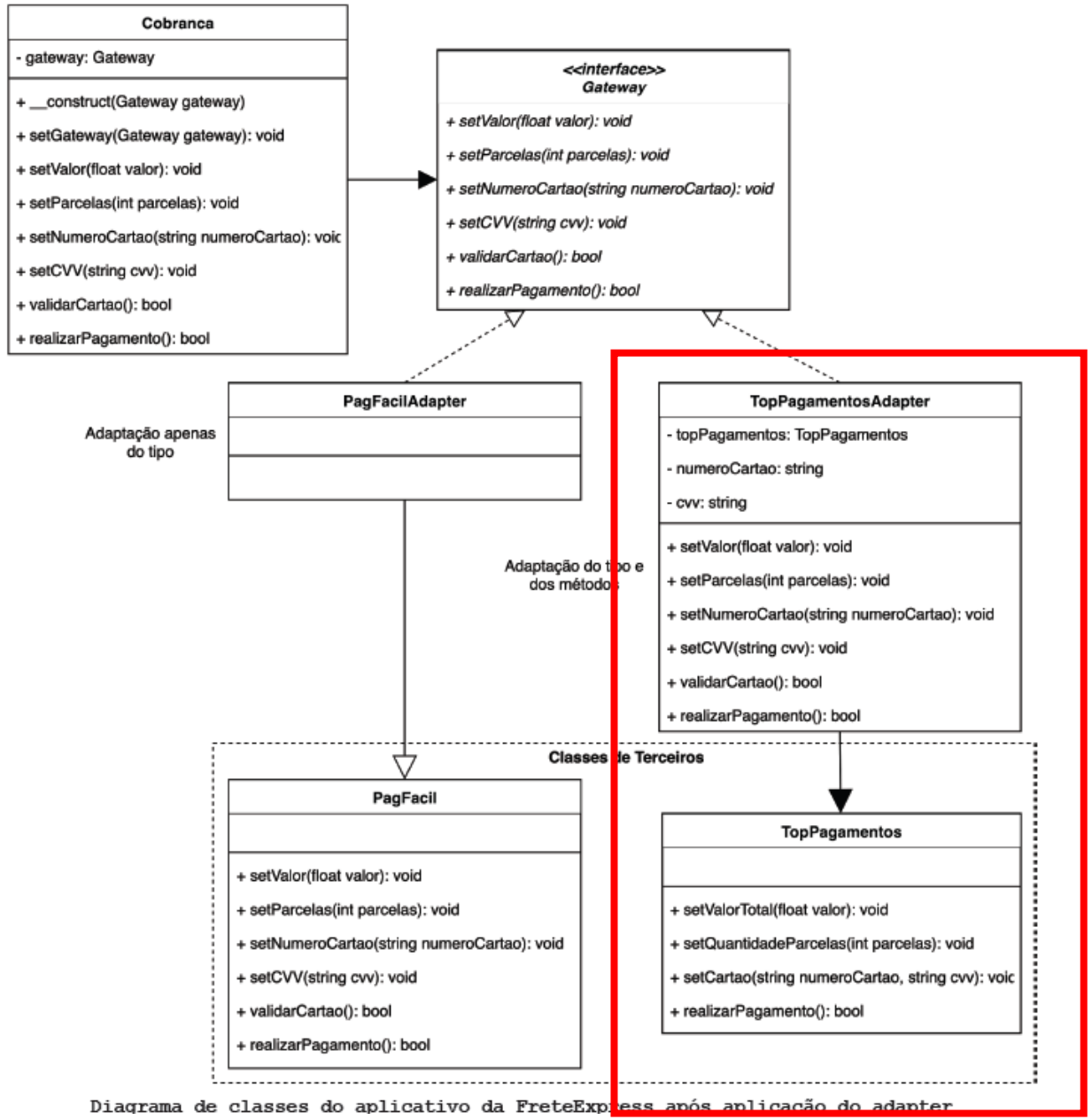
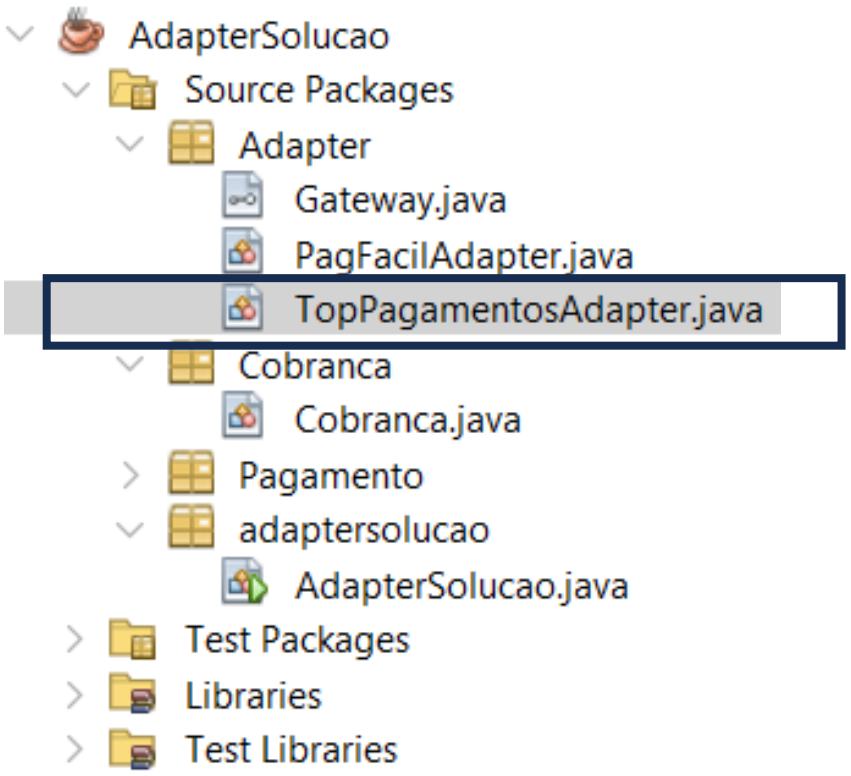


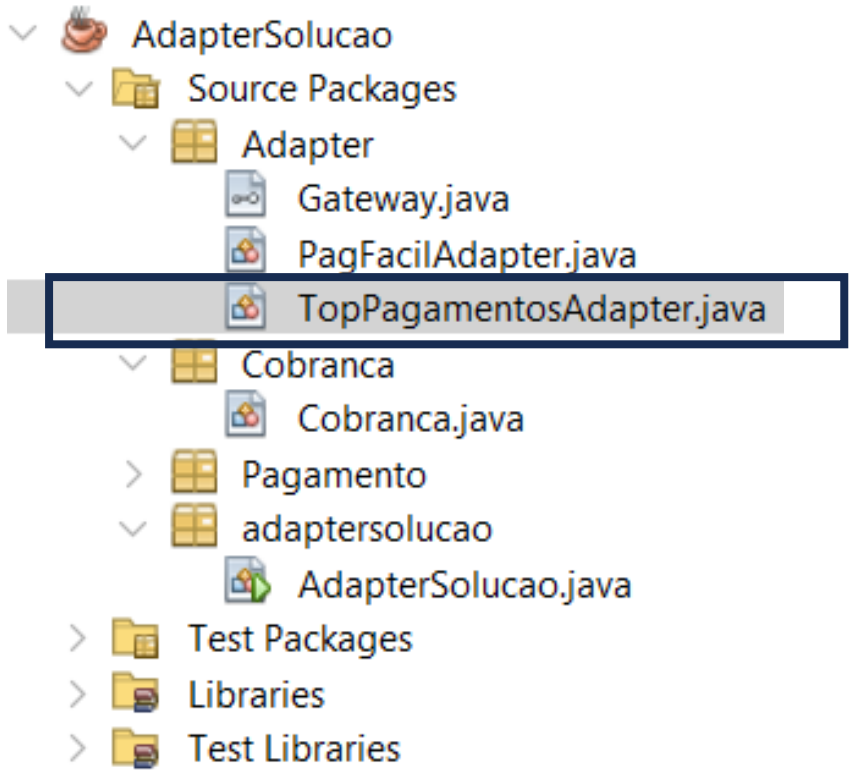
Diagrama de classes do aplicativo da FreteExpress após aplicação do adapter

Primeiro passo é criar a classe **TopPagamentosAdapter** na pasta Adapter.



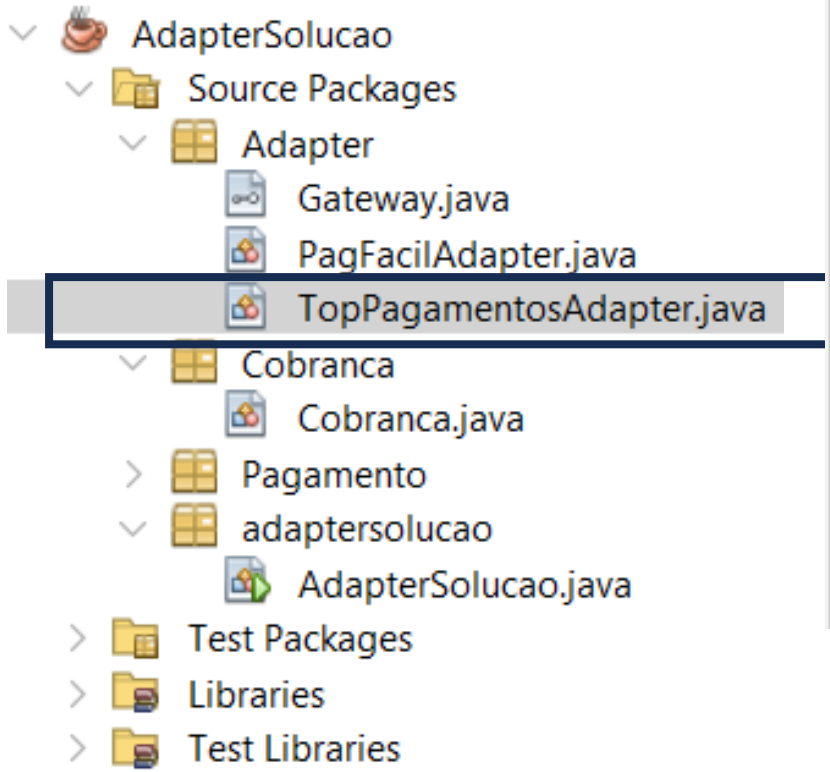

```
1  ...4 lines
5  package Adapter;
6
7  import Pagamento.TopPagamentos;
8
9  /**...4 lines */
13 public class TopPagamentosAdapter implements Gateway {
14
15     public TopPagamentos topPagamentos;
16     public String numeroCartao;
17     public String cvv;
18     public double valor;
19     public int parcelas;
20
21     public TopPagamentosAdapter() {
22         this.topPagamentos = new TopPagamentos();
23     }
```

Continuação da implementação da classe **TopPagamentosAdapter** na pasta Adapter.



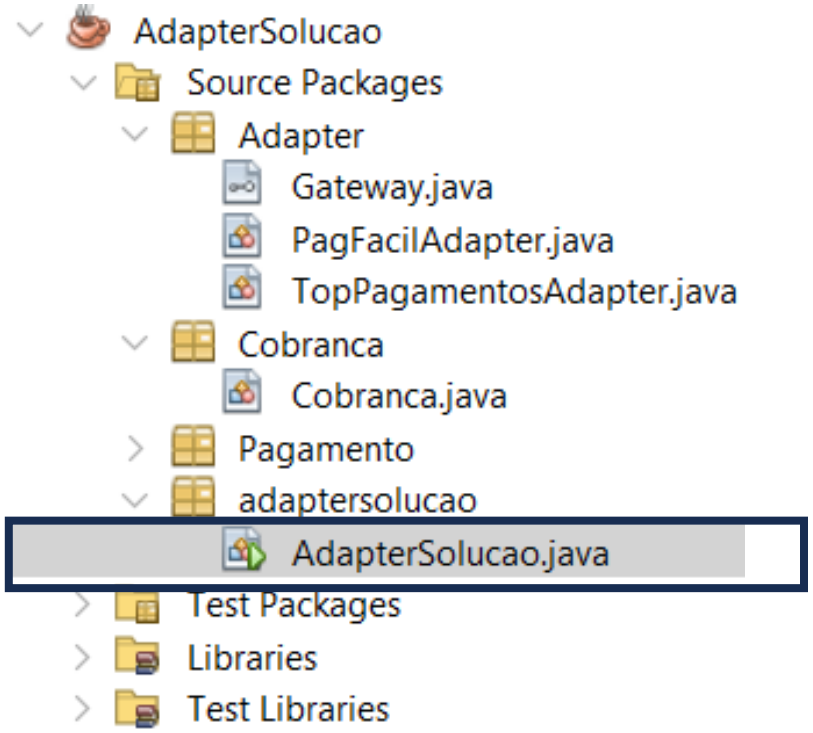

```
24
25      public void setCVV(String cvv)
26      {
27          this.cvv = cvv;
28      }
29
30      public void setNumeroCartao(String numeroCartao)
31      {
32          this.numeroCartao = numeroCartao;
33      }
34
35      public void setParcelas(int parcelas)
36      {
37          this.parcelas = parcelas;
38      }
39
```

Continuação da implementação da classe **TopPagamentosAdapter** na pasta Adapter.

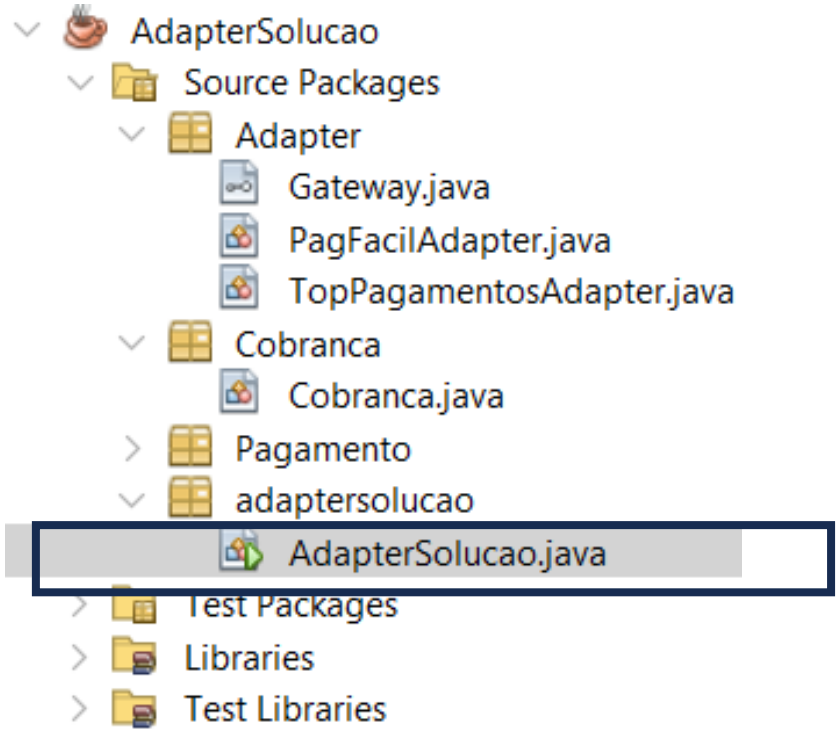



```
41      public void setValor(double valor)
42      {
43          this.valor = valor;
44      }
45
46      public boolean validarCartao()
47      {
48          return true;
49      }
50
51      public boolean realizarPagamento()
52      {
53          this.topPagamentos.setValorTotal(this.valor);
54          this.topPagamentos.setCartao(this.numeroCartao, this.cvv);
55          this.topPagamentos.setQuantidadeParcelas(this.parcelas);
56          return this.topPagamentos.realizarPagamento();
57      }
}
```

Agora implemente o teste na classe Program




```
21      public static void main(String[] args) {
22          System.out.println("Configura a Cobranca");
23
24          Cobranca cobranca = new Cobranca();
25          cobranca.setValor(100);
26          cobranca.setNumeroCartao("999999999999");
27          cobranca.setCVV("163");
28          //pagfacil
29          cobranca.setGateway(new PagFacilAdapter());
30          cobranca.validarCartao();
31          cobranca.realizarPagamento();
32          //toppagamentos
33          cobranca.setGateway(new TopPagamentosAdapter());
34          cobranca.validarCartao();
35          cobranca.realizarPagamento();
36      }
```




```
run:
Configura a Cobranca
Cartão validado com sucesso
Aprovado
Cartão validado com sucesso
Aprovado
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ambos os Gateways de pagamento estão funcionando.