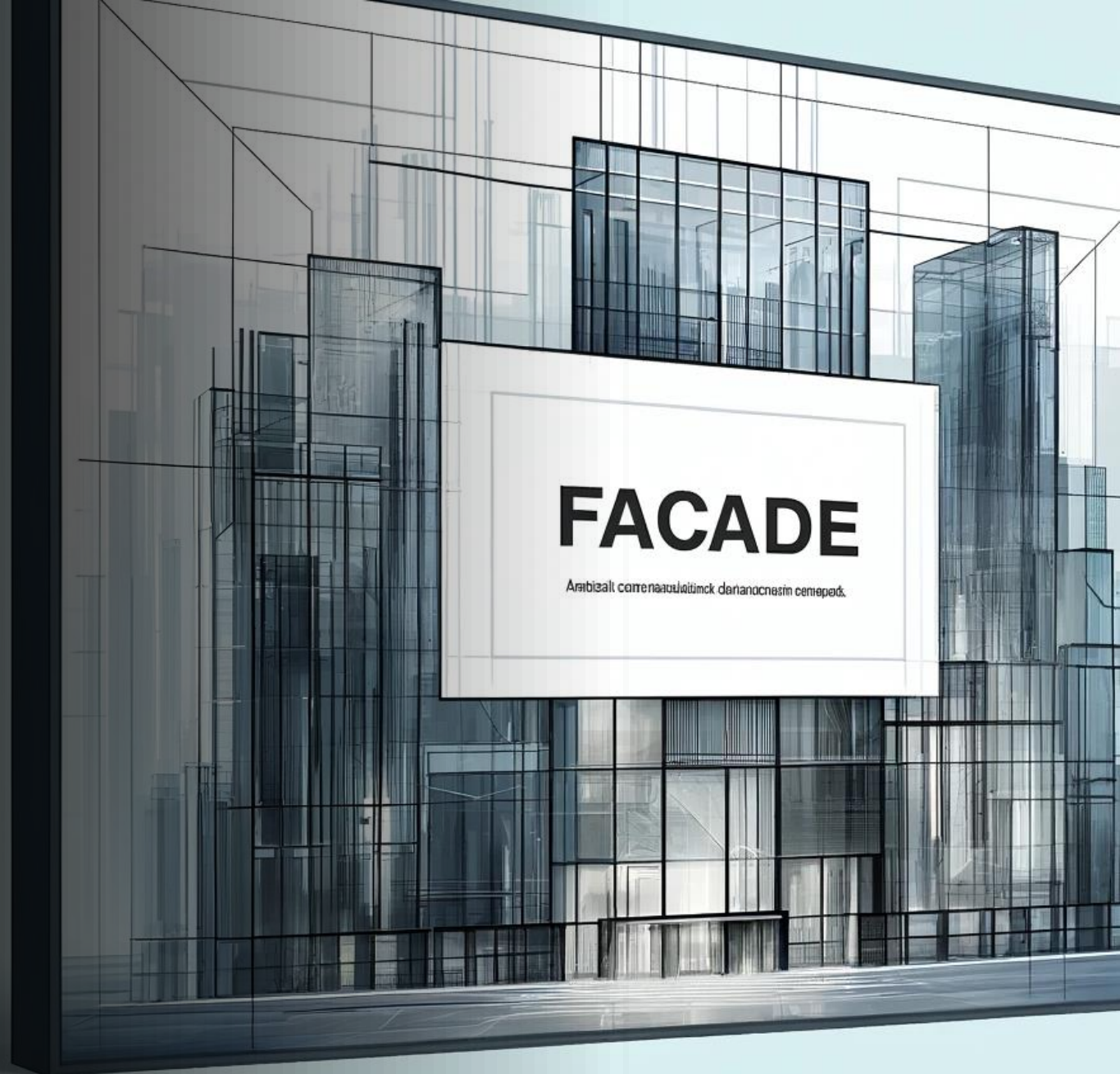




Facade

Padrões de Projeto Estrutural I

Prof. Me Jefferson Passerini



O padrão **Facade** fornece uma interface unificada para um conjunto de interfaces em um subsistema. O Facade define uma interface de nível mais alto que facilita a utilização do subsistema.

Motivação (Por que utilizar?)

A divisão de um sistema complexo em partes menores, ou seja, subsistemas, ajuda a reduzir a complexidade facilitando seu entendimento e manutenção.

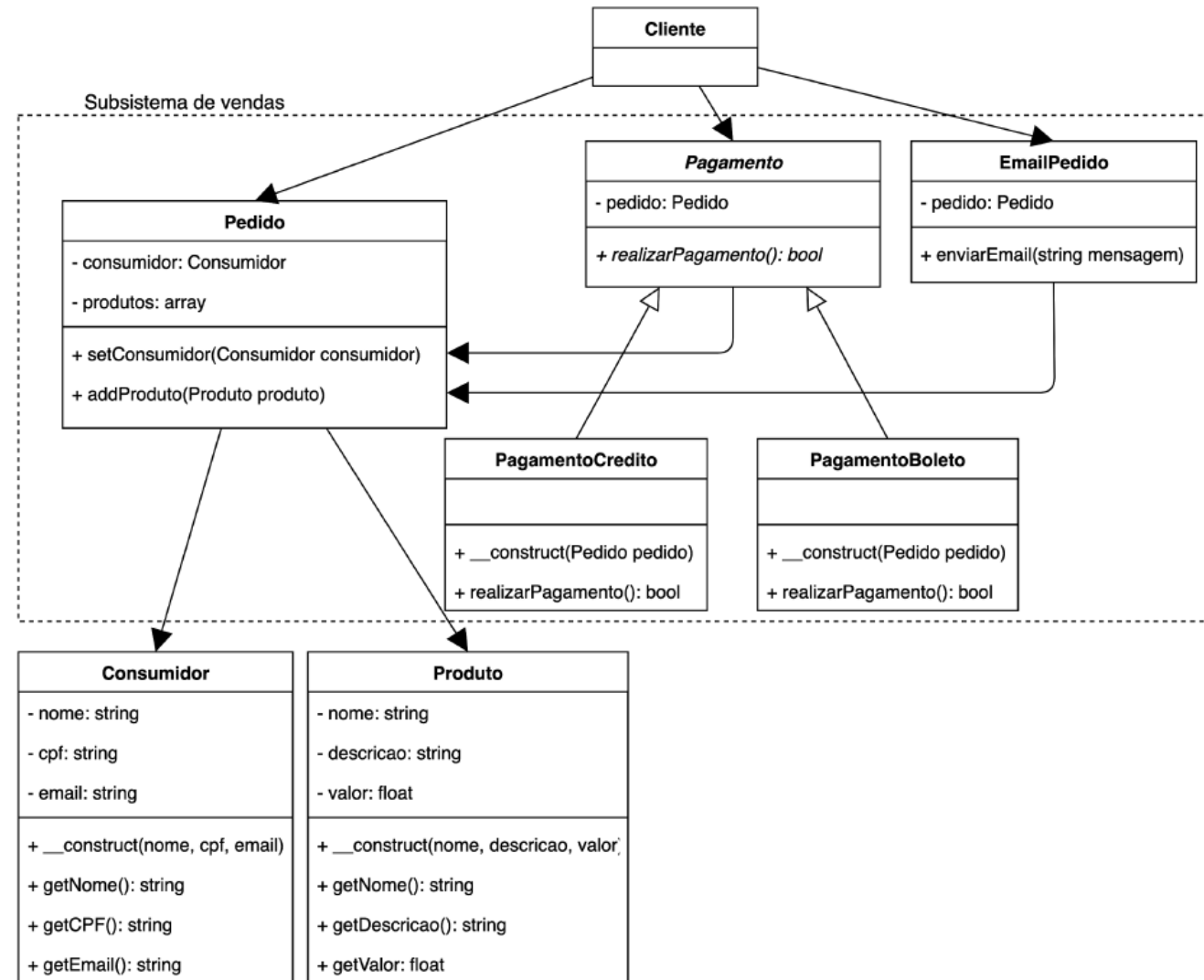
É importante que a comunicação e dependências entre tais subsistemas seja mínima para que sua complexidade não volte a crescer.

Isso pode ser alcançado por meio da introdução de um objeto de fachada, que por sua vez fornece uma interface única e simplificada para o uso de um determinado subsistema

Situação Problema

No subsistema proposto uma venda passa pelos seguintes estágios:

- O **Cliente** cria um **Pedido** que é composto por um **Consumidor** e um ou mais **Produtos**.
- Depois o **Cliente** solicita o pagamento com cartão de crédito ou boleto por meio das classes **PagamentoCredito** ou **PagamentoBoleto**.
- Por fim, envia, por meio da classe **EmailPedido**, um email para o **Consumidor** com os dados do **Pedido**.

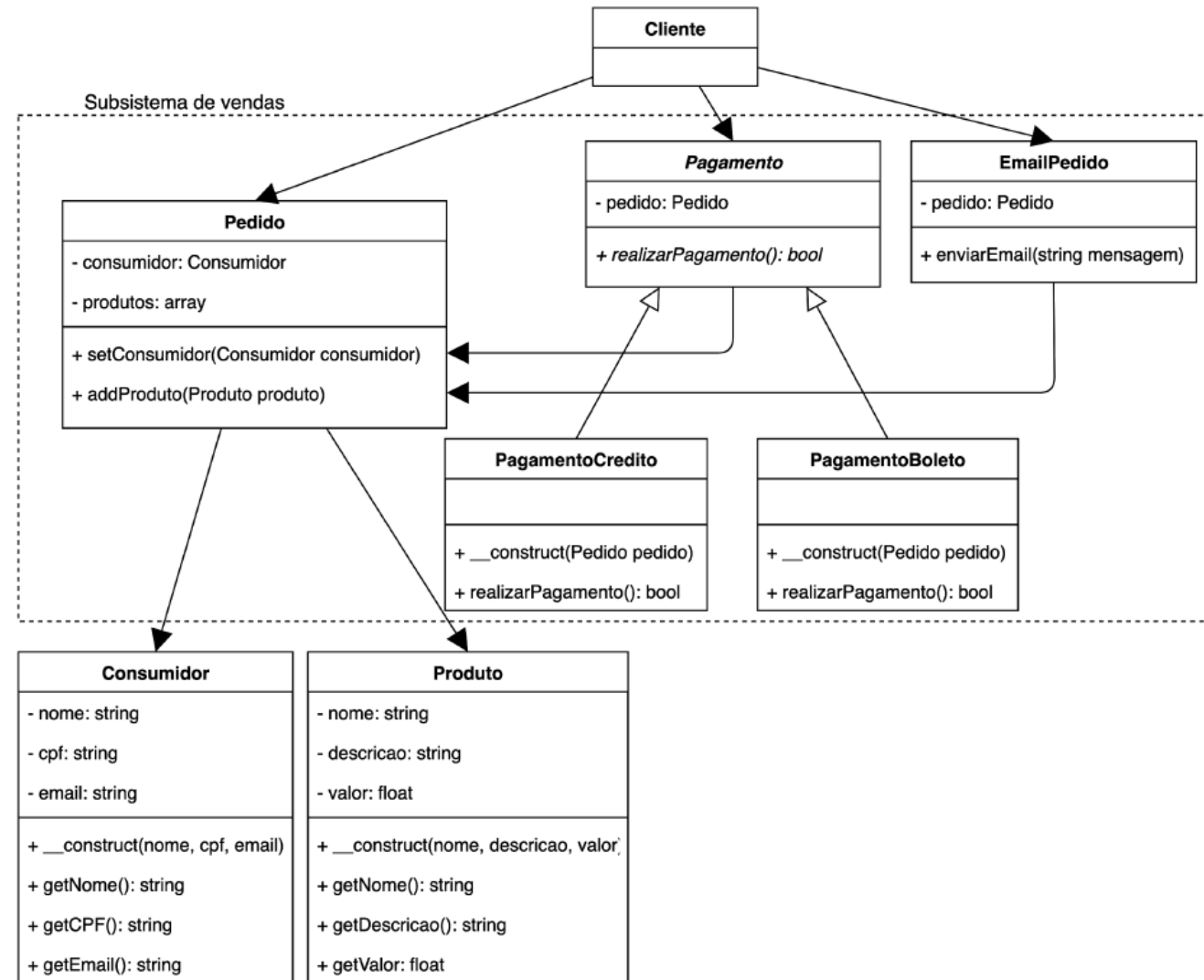


Exemplo de Subsistema de vendas (Sem o padrão Facade)

Situação Problema

Repare na grande quantidade de classes que o cliente precisa conhecer.

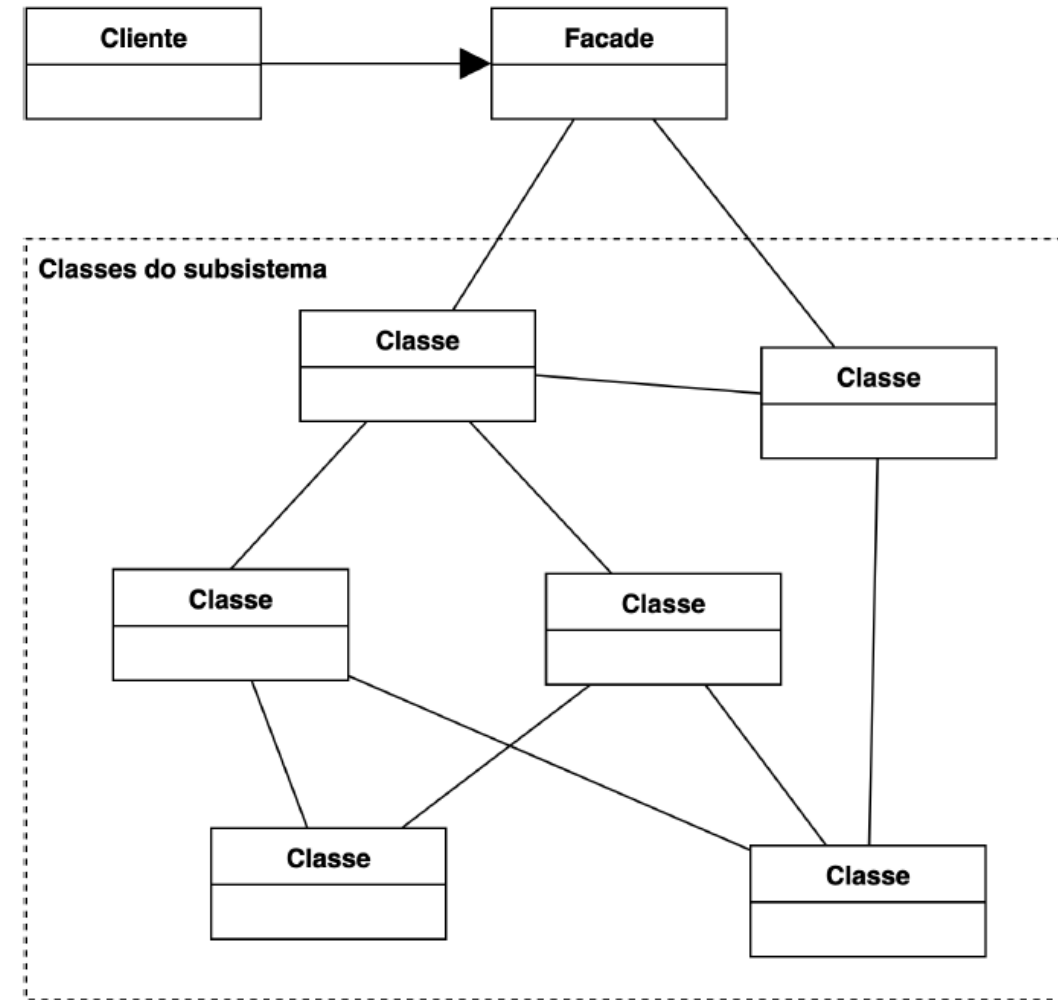
Isso cria um forte acoplamento no cliente. Se algo mudar no subsistema o cliente também terá que mudar, imagine que vários clientes distintos utilizem o subsistema de vendas, todos eles teriam que mudar também.



Exemplo de Subsistema de vendas (Sem o padrão Facade)

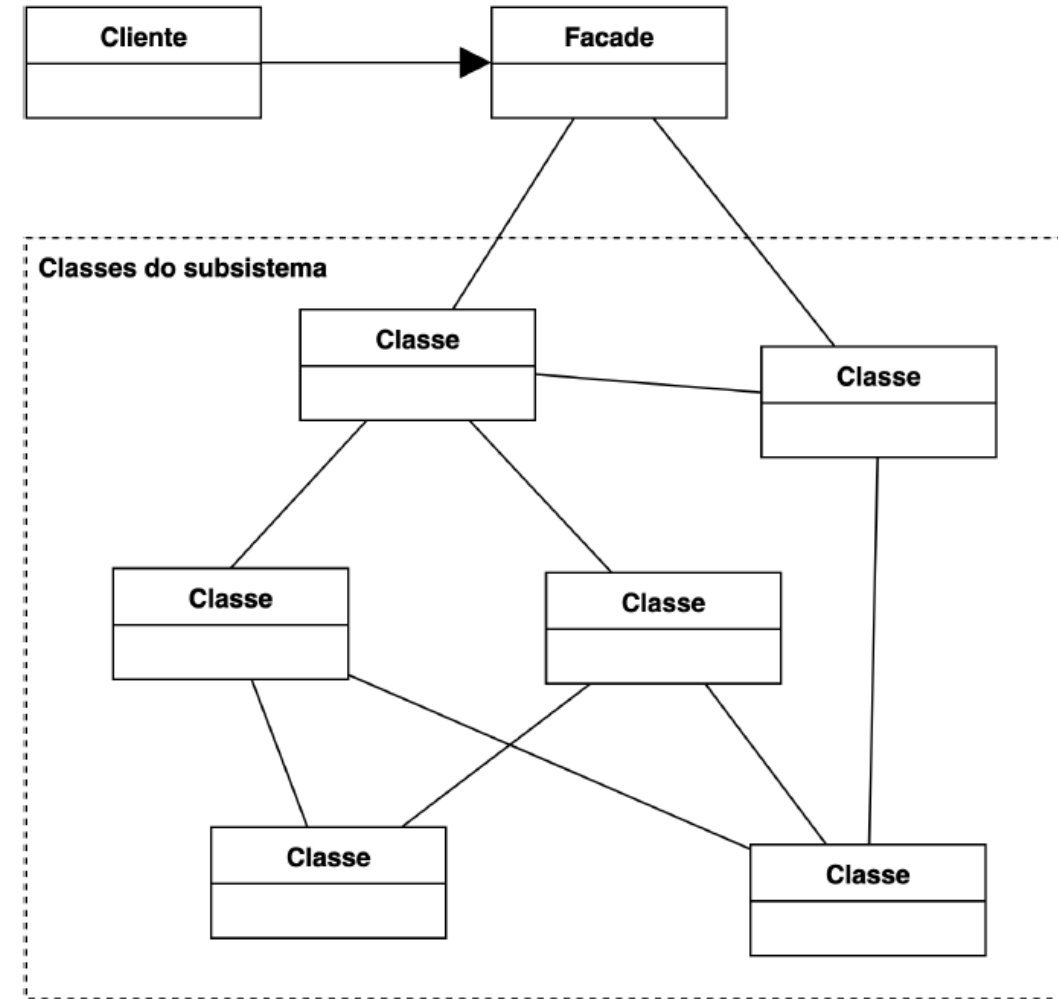
O Padrão FACADE - Componentes

- **Cliente:** é a classe que utiliza as facilidades oferecidas pela classe Facade.
- **Facade:** classe que fornece uma interface unificada e mais simplificada para o cliente. Ela sabe quais classes de subsistema são responsáveis por uma solicitação feita pelo Cliente, delega tais solicitações aos objetos apropriados do subsistema.
- **Subsistema mais complexo:** possui as implementações das funcionalidades do subsistema. É o responsável por responder às solicitações feitas pela classe Facade. Não sabe que Facade existe.



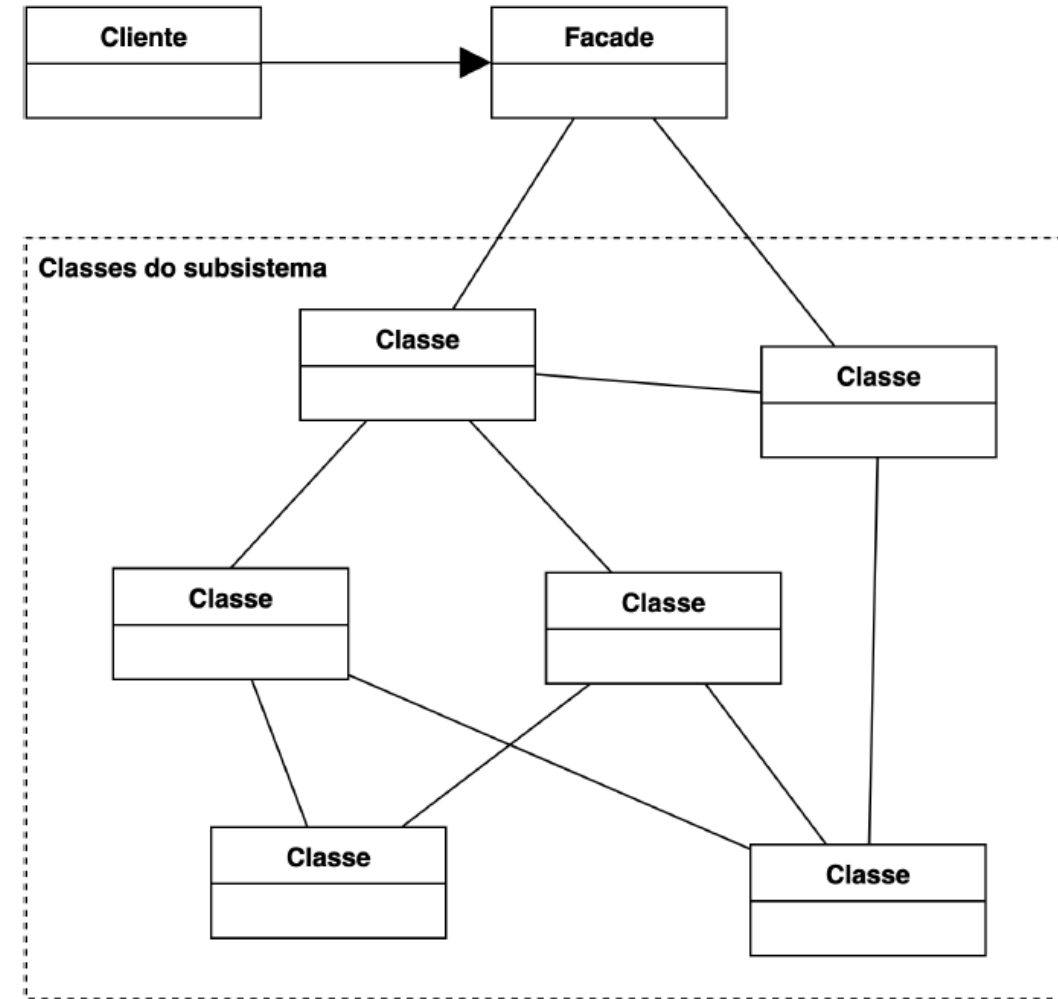
O Padrão FACADE - Aplicabilidade

- Quando é necessário fornecer uma interface simples para um sistema complexo. Os subsistemas geralmente ficam mais complexos à medida que evoluem.
- A maioria dos padrões, quando aplicados, resulta em classes cada vez menores. Isso torna o subsistema reutilizável e fácil de personalizar, mas também se torna mais difícil de usar para clientes que não precisam personalizá-lo.



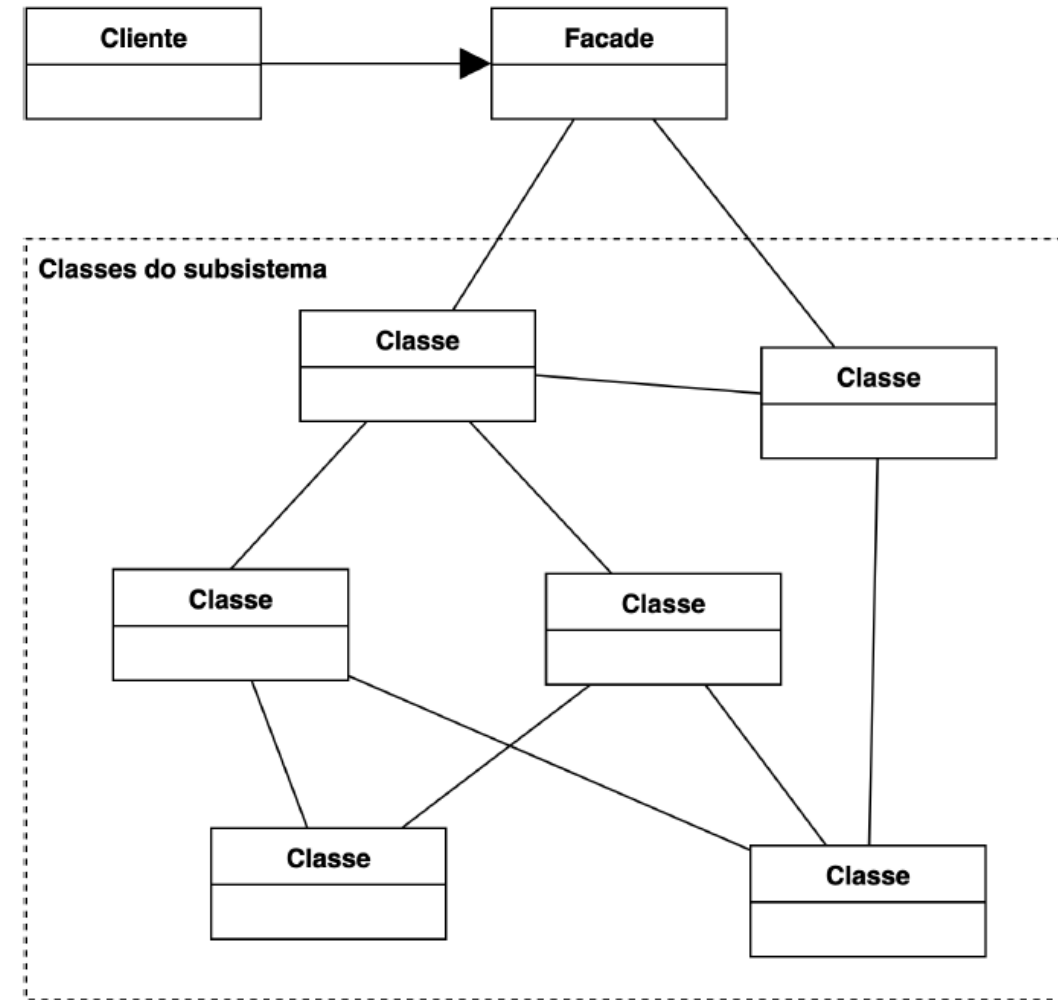
O Padrão FACADE - Aplicabilidade

- Uma fachada (FACADE) pode fornecer uma visualização padrão simples do subsistema que é boa o suficiente para a maioria dos clientes.
- Somente clientes que precisam de mais personalizações precisam olhar além da fachada.



O Padrão FACADE - Aplicabilidade

- Quando existem muitas dependências entre clientes e as classes de implementação de um abstração. Introduzir uma fachada para desacoplar o subsistema dos clientes e outros subsistemas, promove a independência e a portabilidade do sistema.
- Quando se deseja estruturar subsistemas em camadas. Uma fachada define um ponto de entrada para cada nível do subsistema. Se os subsistemas são dependentes, será possível simplificar as dependências entre eles, fazendo-os se comunicar apenas através de suas fachadas.



O Padrão FACADE - Consequências

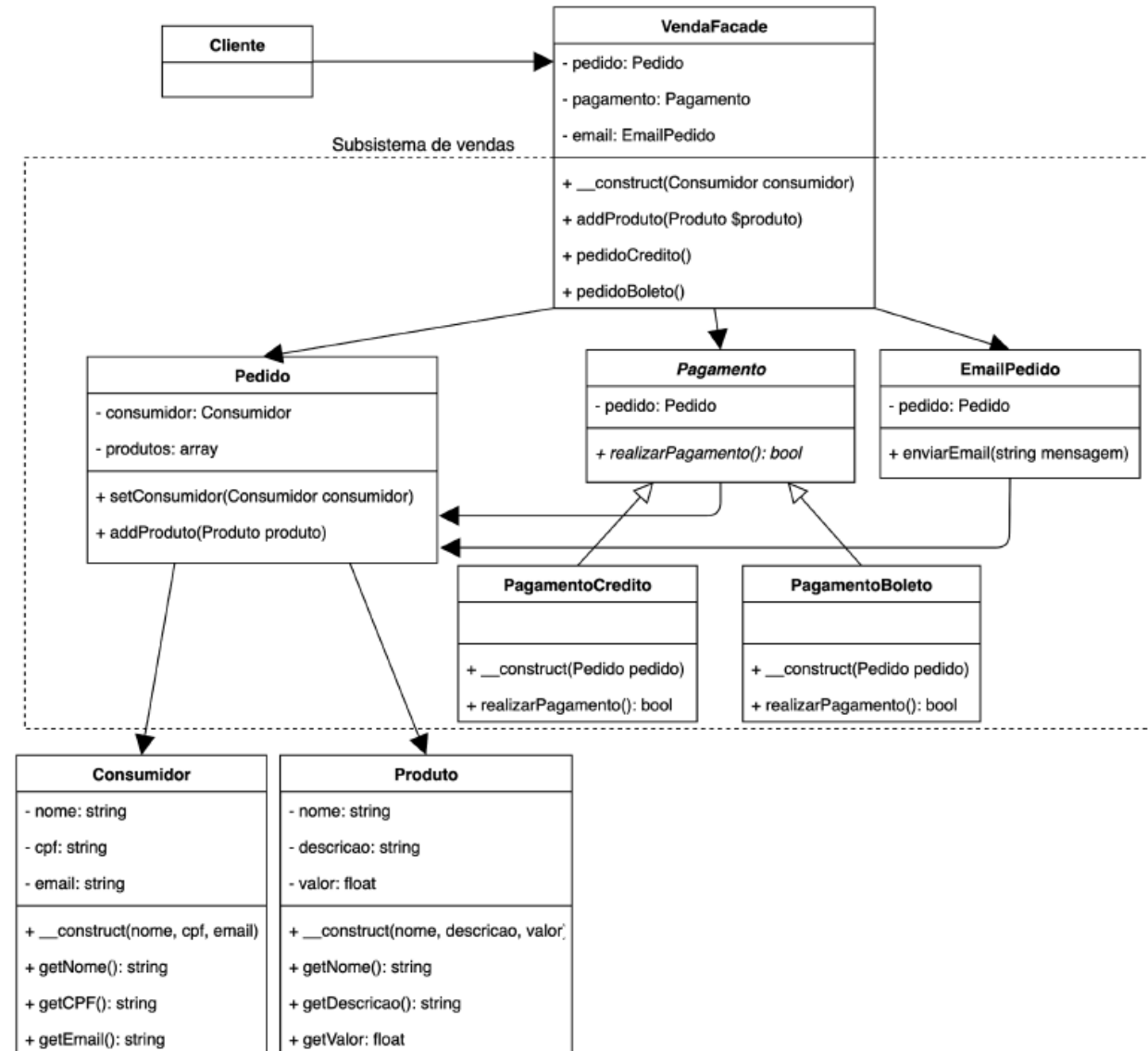
- O padrão Facade cria uma camada entre clientes e componentes do subsistema, isso reduz o número de objetos com os quais os clientes lidam e facilita o uso do subsistema.
- Promove fraco acoplamento entre o subsistema e seus clientes. Geralmente, os componentes em um subsistema são fortemente acoplados. O acoplamento fraco permite variar os componentes do subsistema sem afetar seus clientes. O padrão Facade ajuda eliminar dependências complexas ou circulares entre clientes e o subsistema. Isso pode ser uma consequência importante quando eles são implementados independentemente.

O Padrão FACADE - Consequências

- O padrão Facade não impede que os clientes usem as classes do subsistema se necessário. Assim, é possível escolher entre a dificuldade de uso proporcionada pelo padrão Facade ou o controle total disponível diretamente no subsistema.

O Padrão FACADE - Solução

- O padrão Facade tem o objetivo de remover o forte acoplamento e simplificar o processo de vendas para o cliente.
- Isso é alcançado por meio de um interface simplificada. Vejamos o código.



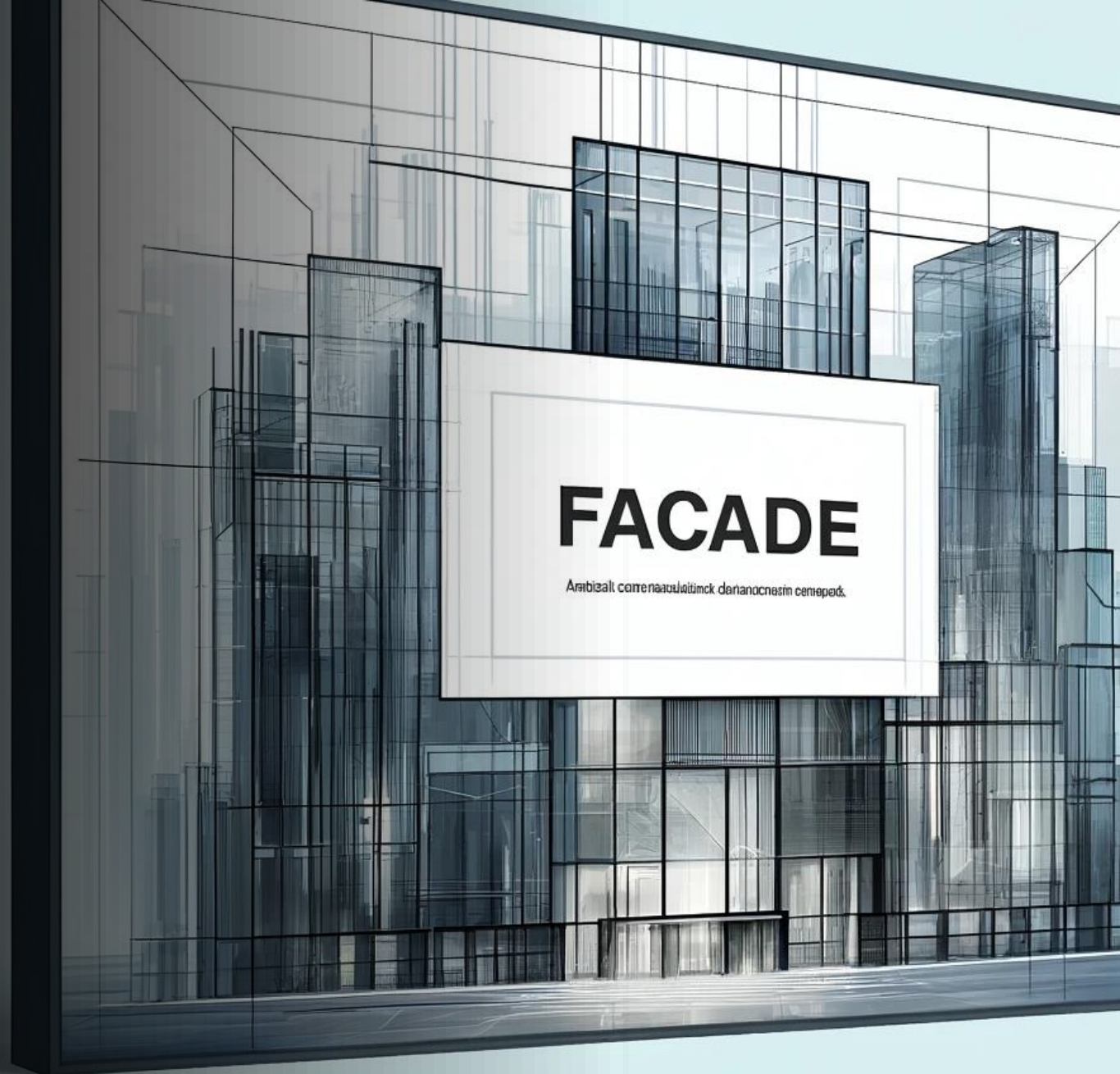
Exemplo de subsistema vendas (Com o padrão Facade)



Implementação em C#

Facade
Padrões de Projeto Estrutural I

Prof. Me Jefferson Passerini



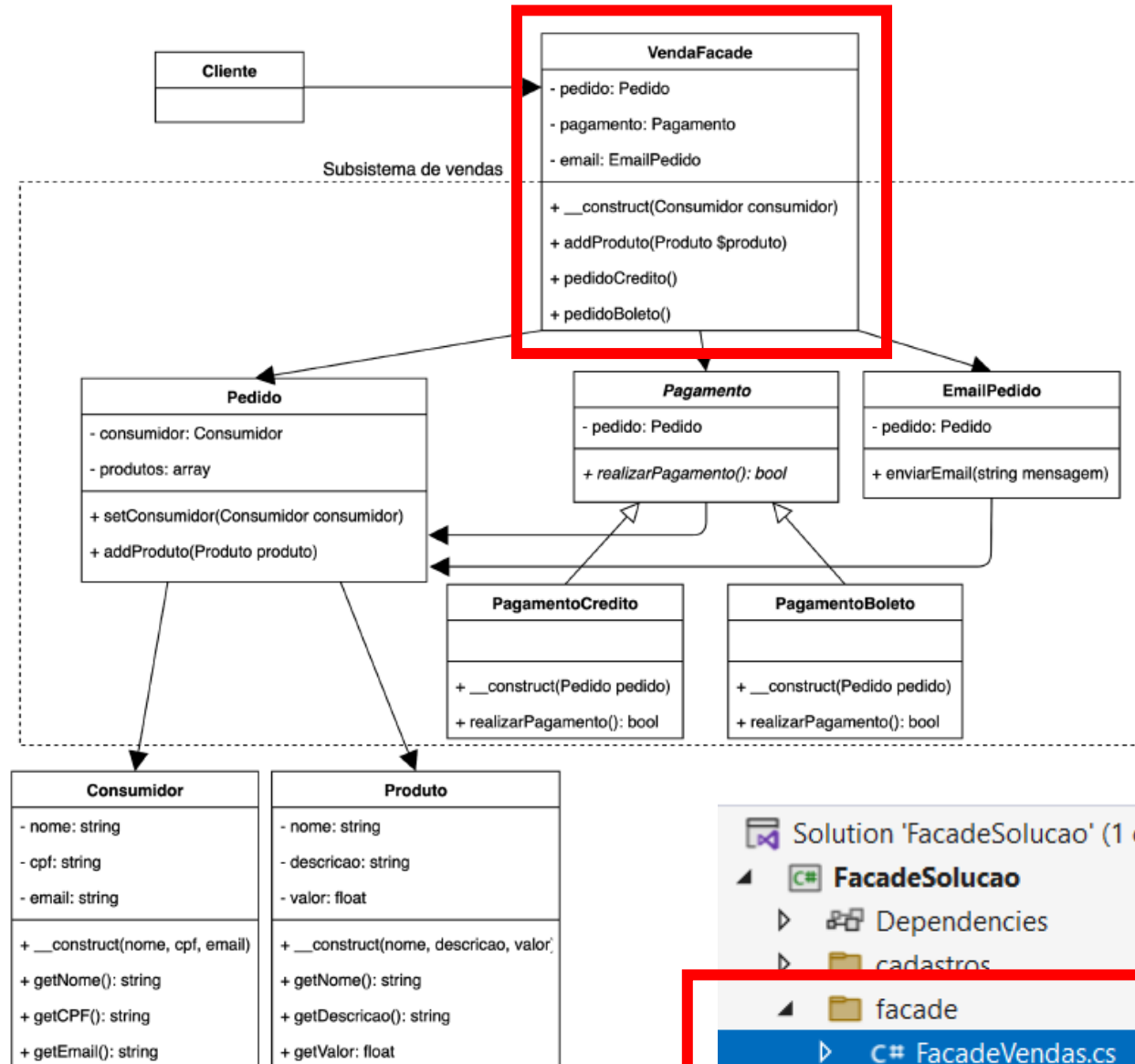
O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

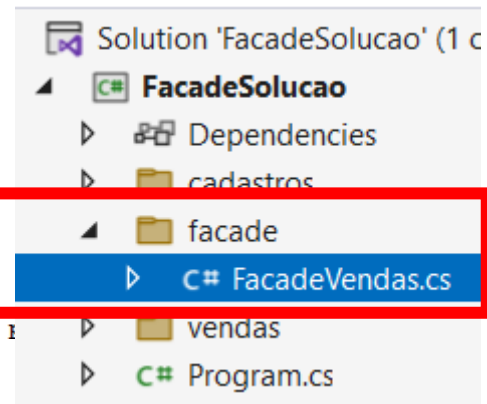
```

1  using FacadeProblema.cadastros;
2  using FacadeProblema.vendas;
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace FacadeSolucao.facade
10 {
11     3 references
12     public class FacadeVendas
13     {
14         private Pedido pedido;
15         private Pagamento pagamento;
16         private EmailPedido email;
17
18         1 reference
19         public FacadeVendas(Consumidor consumidor)
20         {
21             this.pedido = new Pedido();
22             this.pedido.consumidor = consumidor;
23
24             this.email = new EmailPedido(pedido);
25         }
26     }

```



Exemplo de subsistema vendas (Com o I



O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

```

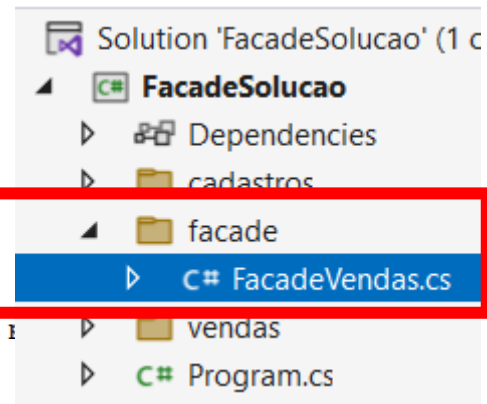
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43

public void addProduto(Produto produto)
{
    this.pedido.addProduto(produto);
}

0 references
public void pedidoCredito()
{
    this.pagamento = new PagamentoCredito(this.pedido);
    if (this.pagamento.realizarPagamento())
    {
        this.email.enviarEmail("Pagamento realizado "
            + "com sucesso (crédito)");
    }
    else
    {
        this.email.enviarEmail("Falha no pagamento "
            + "do pedido");
    }
}
    
```



Exemplo de subsistema vendas (Com o I



O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

```

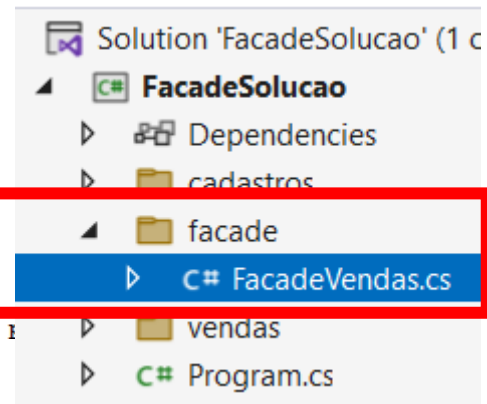
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60

1 reference
public void pedidoBoleto()
{
    this.pagamento = new PagamentoBoleto(this.pedido);
    if (this.pagamento.realizarPagamento())
    {
        this.email.enviarEmail("Pagamento realizado "
            + "com sucesso (boleto)");
    }
    else
    {
        this.email.enviarEmail("Falha no pagamento"
            + " do pedido");
    }
}

```



Exemplo de subsistema vendas (Com o I

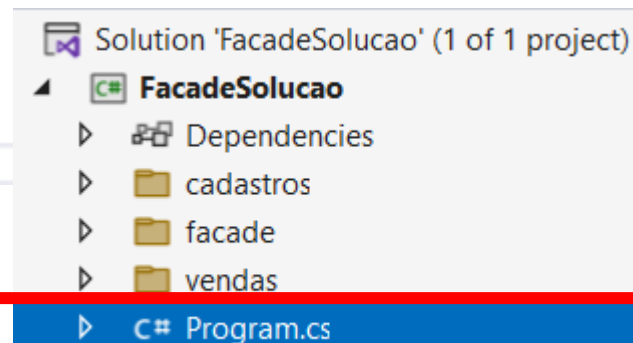
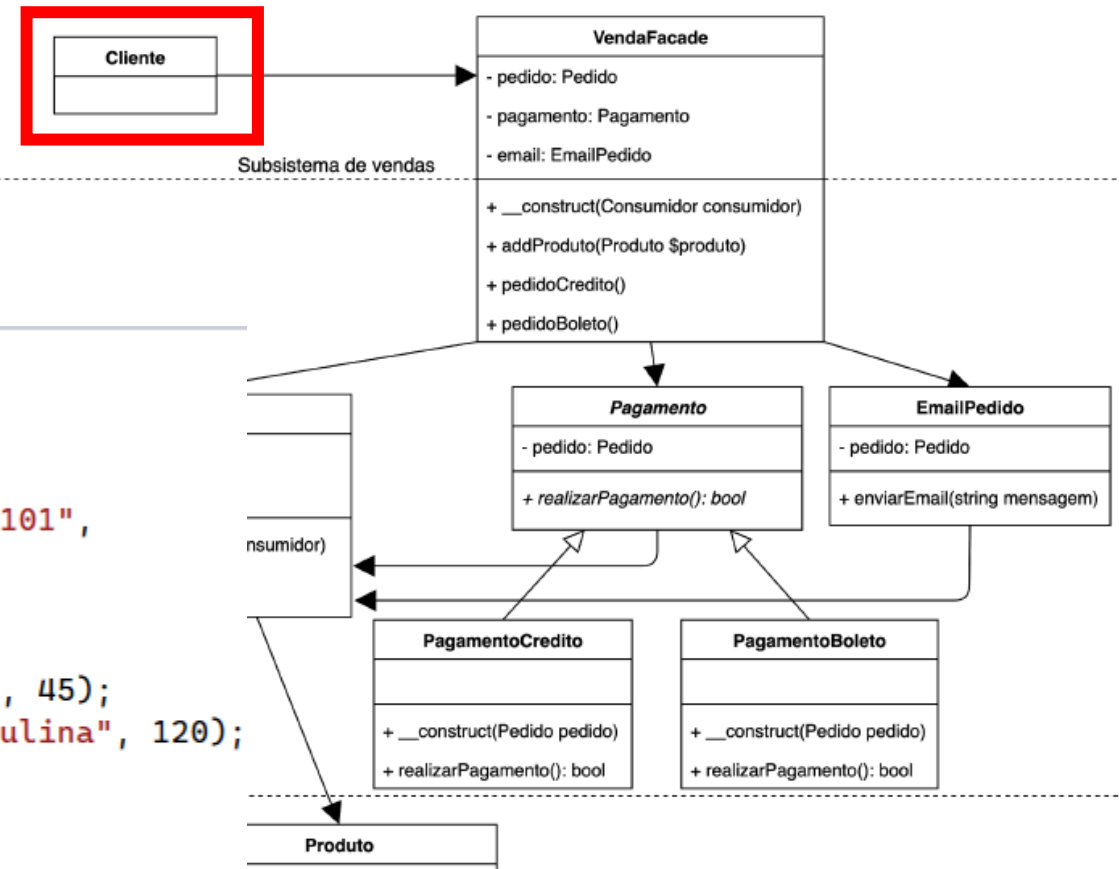


O Padrão FACADE - Solução

- Altere os testes na **main** do projeto.

```

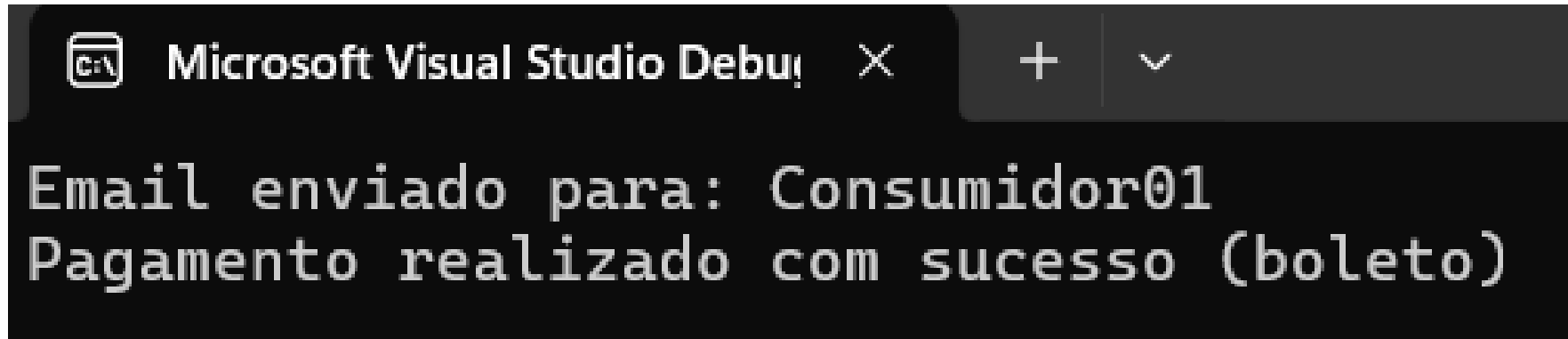
1  using FacadeProblema.cadastros;
2  using FacadeProblema.vendas;
3  using FacadeSolucao.facade;
4
5  Consumidor consumidor = new Consumidor("Consumidor01", "0101010101",
6      "consum@gmail.com");
7
8  Produto produto1 = new Produto("Blusa", "Blusa feminina", 80);
9  Produto produto2 = new Produto("Camiseta", "Camiseta Masculina", 45);
10 Produto produto3 = new Produto("Calça Jeans", "Calça Jeans Masculina", 120);
11
12 FacadeVendas pedido = new FacadeVendas(consumidor);
13 pedido.addProduto(produto1);
14 pedido.addProduto(produto2);
15 pedido.addProduto(produto3);
16
17 pedido.pedidoBoleto();
  
```



(Com o padrão Facade)

O Padrão FACADE - Solução

- Agora o Cliente não depende mais diretamente do subsistema.
- A classe FacadeVendas simplifica o processo para o cliente, se houver modificação no subsistema de venda basta editar essa classe.
- Embora o Facade simplifique a interface, o Cliente ainda é capaz de acessar o subsistema de forma direta se necessário.
- Cumpre o objetivo de simplificar a interface.



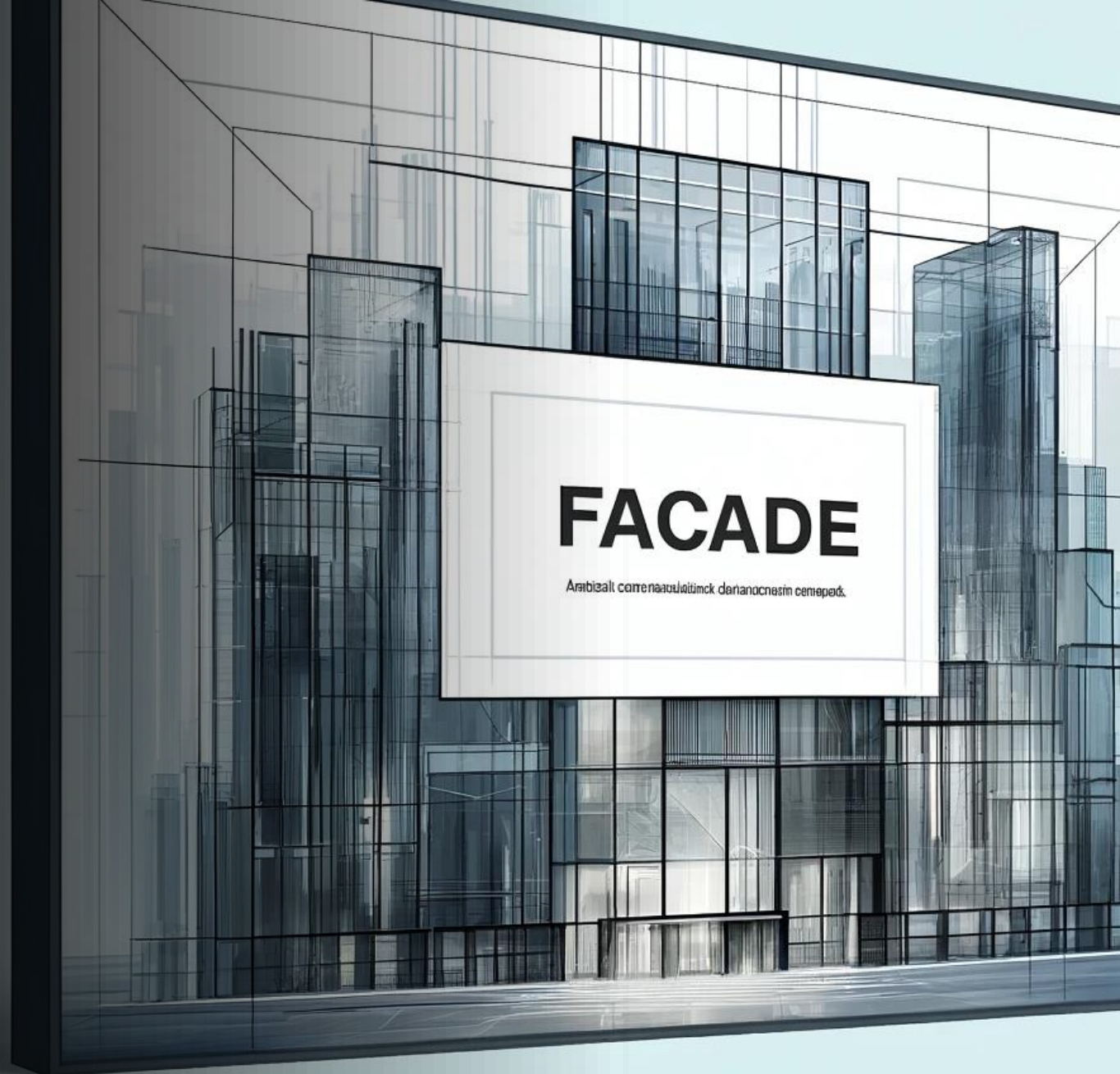
```
Microsoft Visual Studio Debug Console
Email enviado para: Consumidor01
Pagamento realizado com sucesso (boleto)
```



Implementação em Java

Facade
Padrões de Projeto Estrutural I

Prof. Me Jefferson Passerini



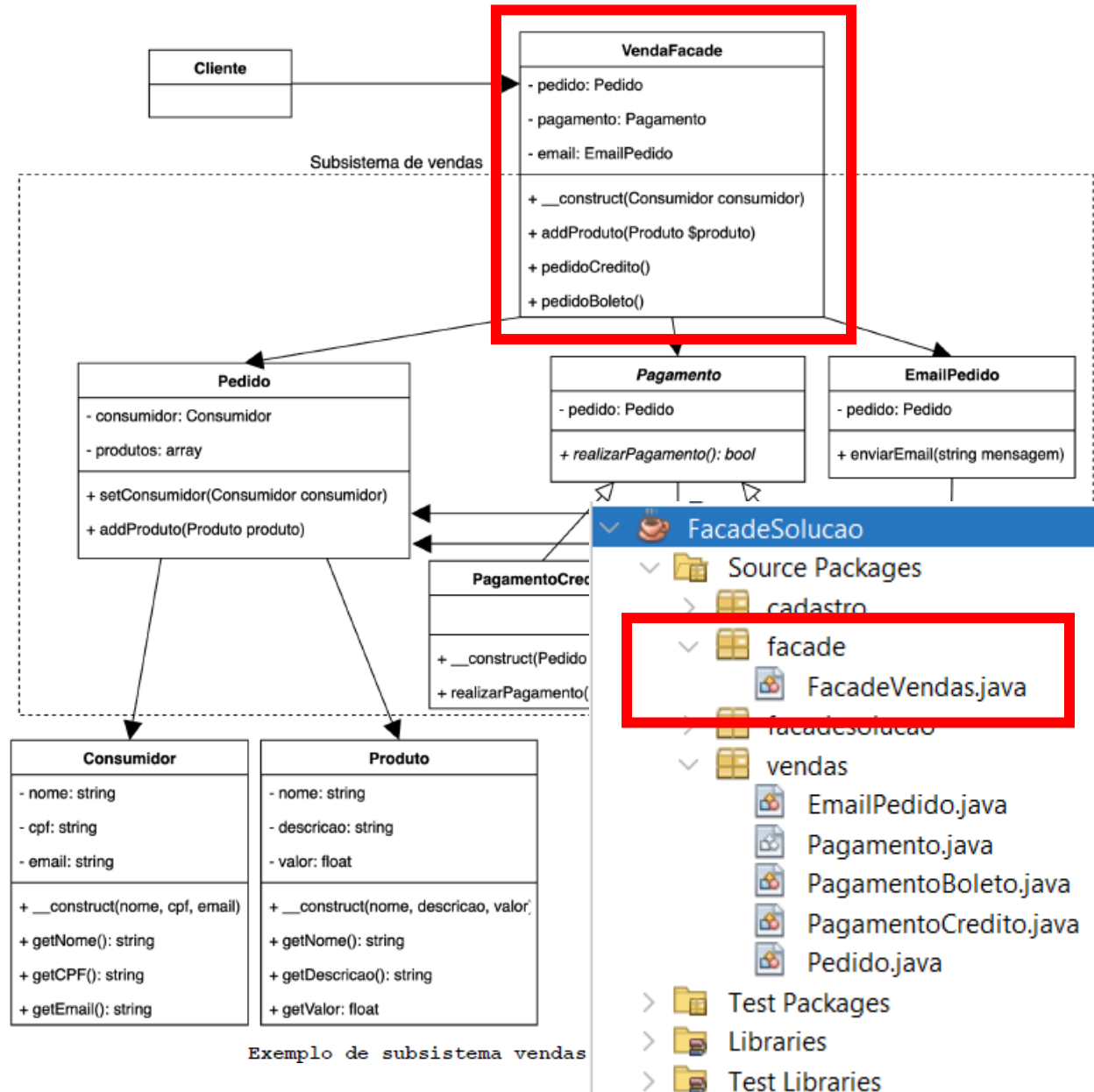
O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

```

1  ...4 lines
5  package facade;
6
7  import cadastro.Consumidor;
8  import cadastro.Produto;
9  import vendas.EmailPedido;
10 import vendas.Pagamento;
11 import vendas.PagamentoBoleto;
12 import vendas.PagamentoCredito;
13 import vendas.Pedido;
14
15 /**...4 lines */
19 public class FacadeVendas {
20
21     private Pedido pedido;
22     private Pagamento pagamento;
23     private EmailPedido email;
24

```

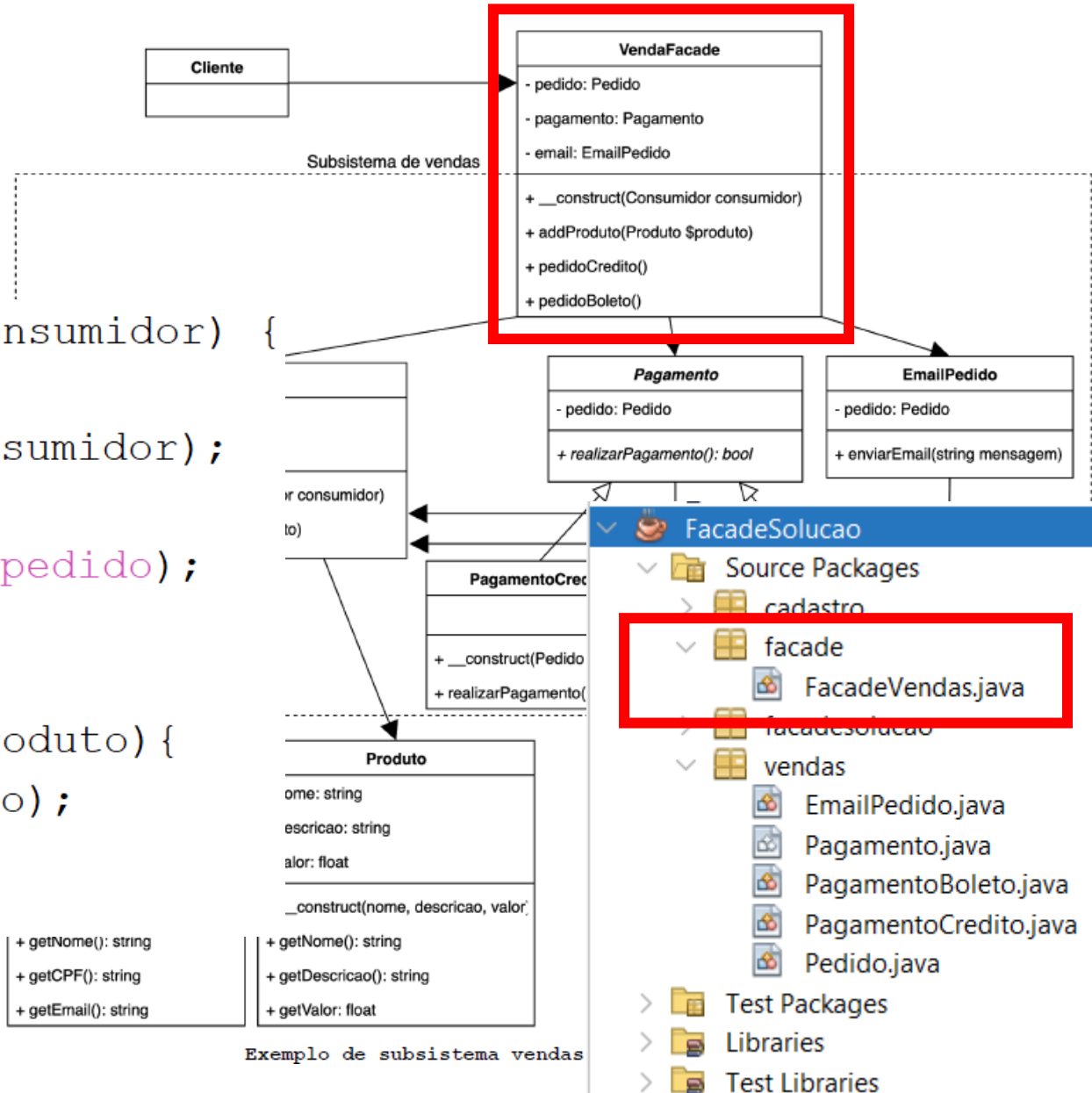


O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

```

25 public FacadeVendas(Consumidor consumidor) {
26     this.pedido = new Pedido();
27     this.pedido.setConsumidor(consumidor);
28
29     this.email = new EmailPedido(pedido);
30 }
31
32 public void addProduto(Produto produto) {
33     this.pedido.addProduto(produto);
34 }
35
    
```



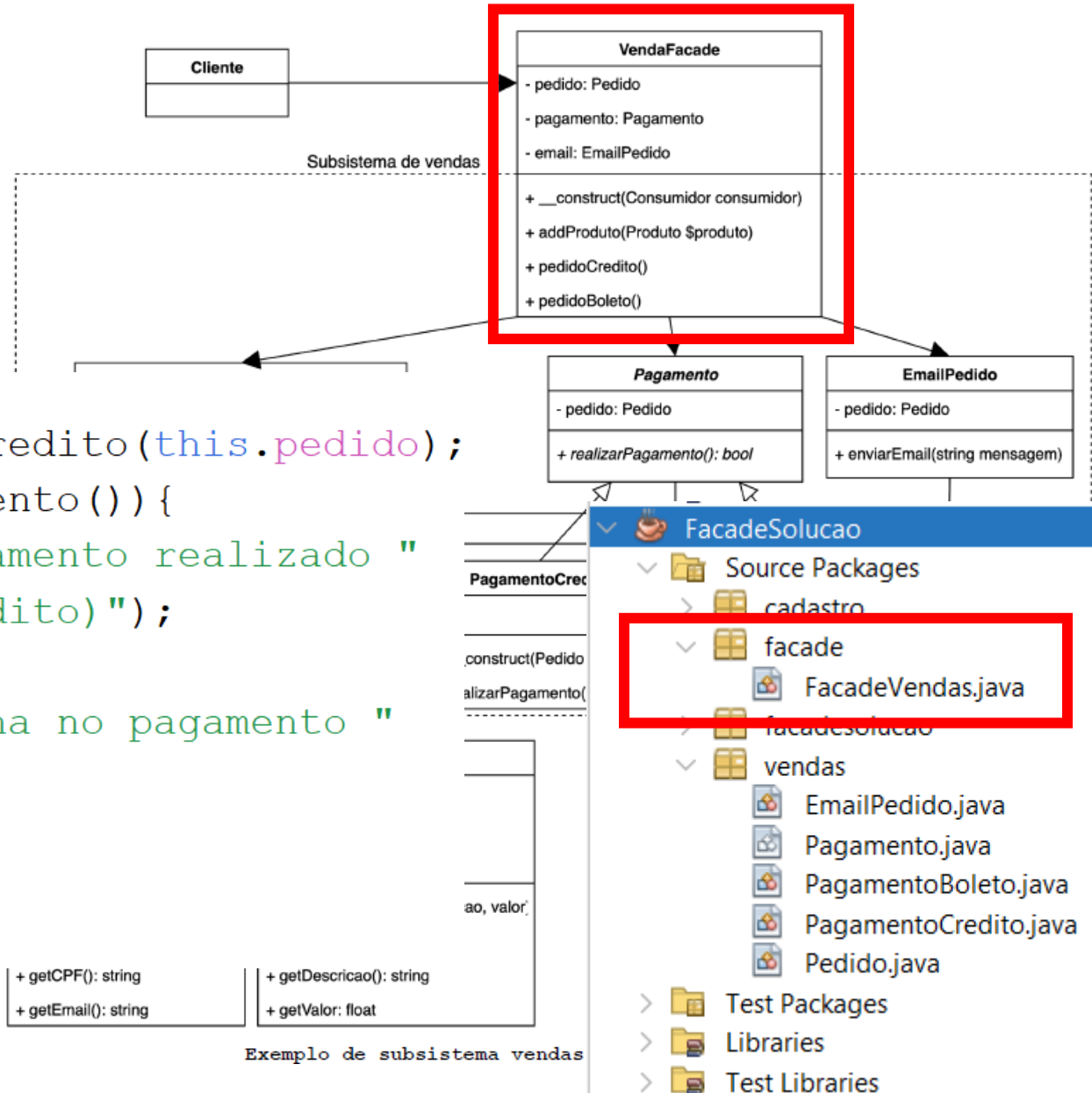
O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

```

36 public void pedidoCredito() {
37     this.pagamento = new PagamentoCredito(this.pedido);
38     if(this.pagamento.realizarPagamento()) {
39         this.email.enviarEmail("Pagamento realizado "
40             + "com sucesso (crédito)");
41     } else {
42         this.email.enviarEmail("Falha no pagamento "
43             + "do pedido");
44     }
45 }
46

```

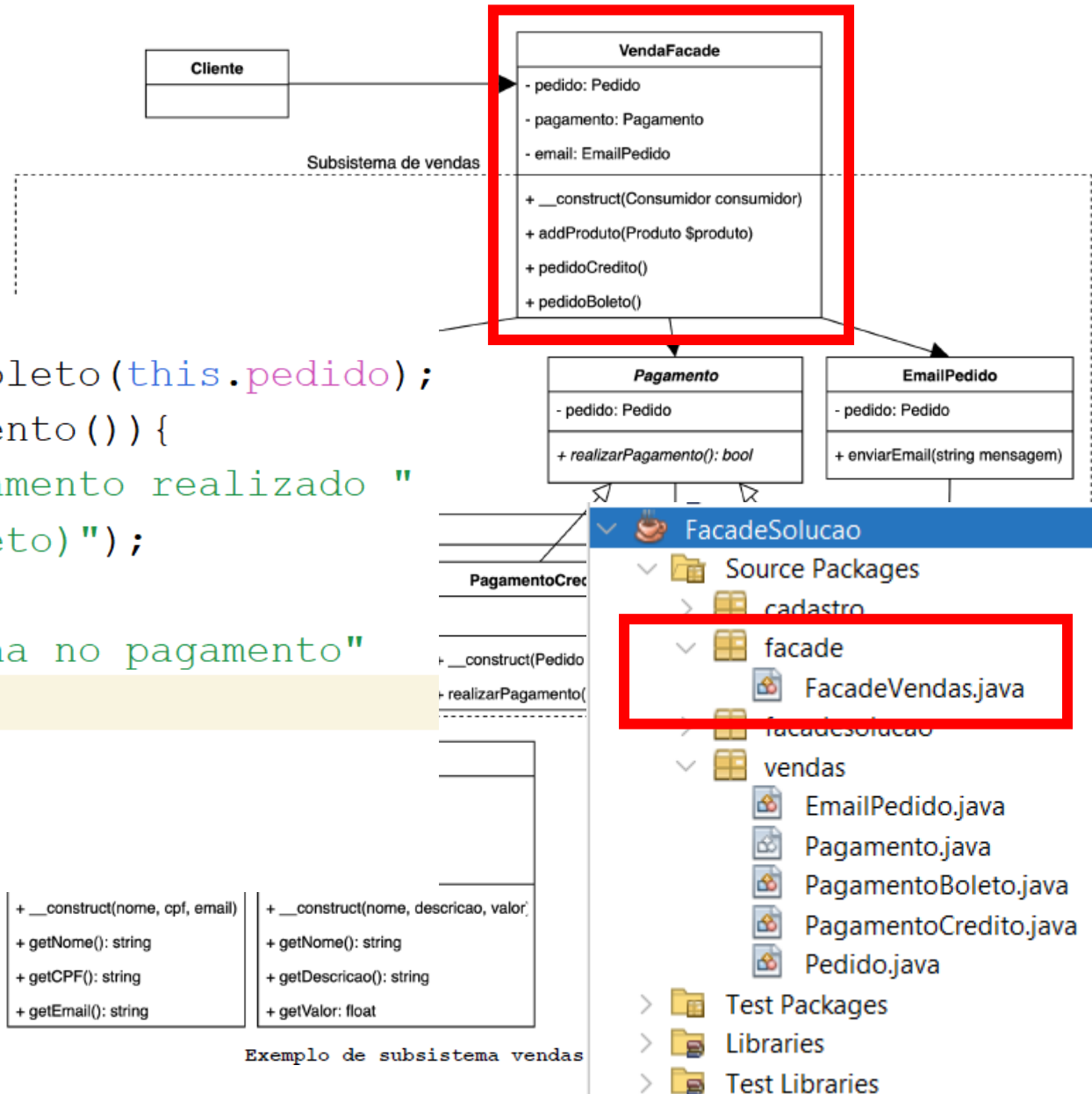


O Padrão FACADE - Solução

- Para refatorar o código crie a pasta **facade** e crie a classe **FacadeVendas**.

```

47 public void pedidoBoleto() {
48     this.pagamento = new PagamentoBoleto(this.pedido);
49     if(this.pagamento.realizarPagamento()) {
50         this.email.enviarEmail("Pagamento realizado "
51             + "com sucesso (boleto)");
52     } else {
53         this.email.enviarEmail("Falha no pagamento"
54             + " do pedido");
55     }
56 }
57 }
    
```

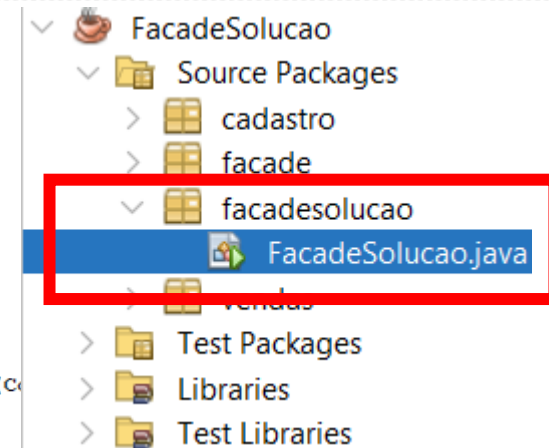
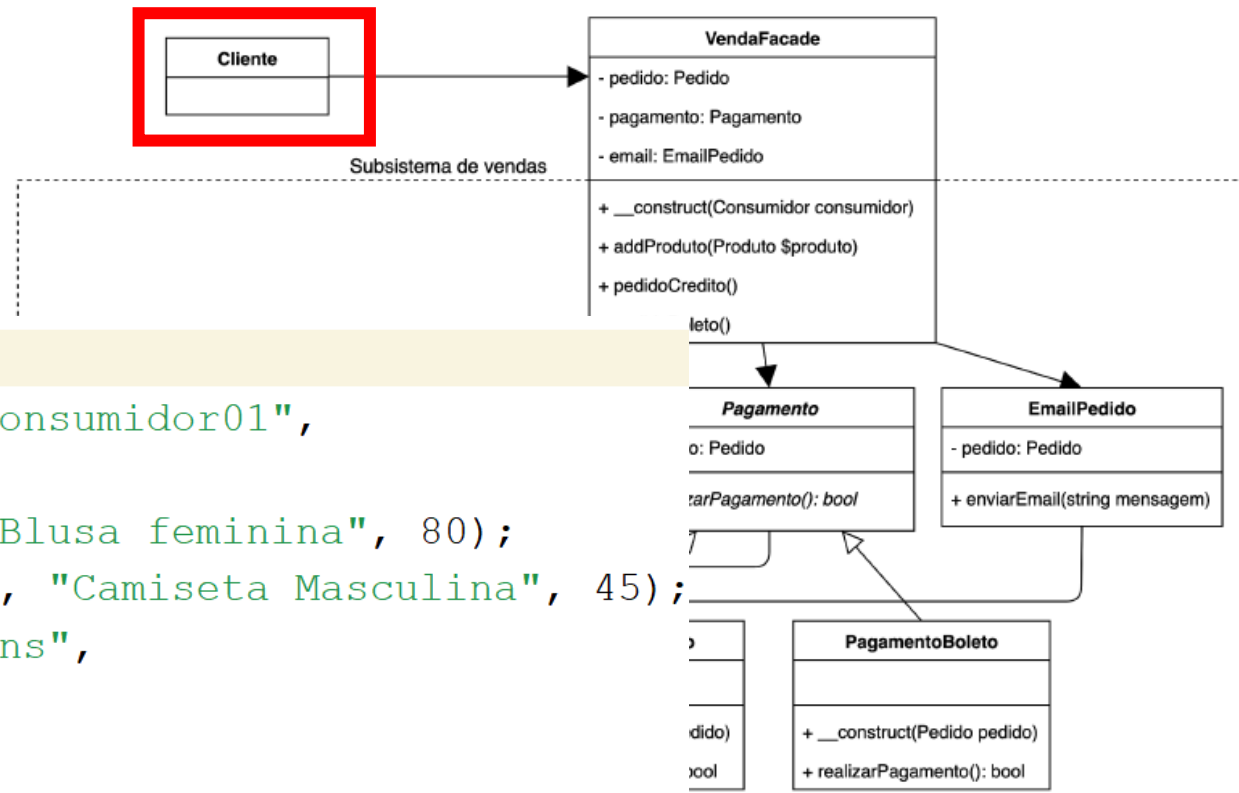


O Padrão FACADE - Solução

- Altere os testes na **main** do projeto.

```
public static void main(String[] args) {
    Consumidor consumidor = new Consumidor("Consumidor01",
        "0101010101", "consum@gmail.com");
    Produto produto1 = new Produto("Blusa", "Blusa feminina", 80);
    Produto produto2 = new Produto("Camiseta", "Camiseta Masculina", 45);
    Produto produto3 = new Produto("Calça Jeans",
        "Calça Jeans Masculina", 120);

    FacadeVendas pedido = new FacadeVendas(consumidor);
    pedido.addProduto(produto1);
    pedido.addProduto(produto2);
    pedido.addProduto(produto3);
    pedido.pedidoBoleto();
}
```



+ getEmail(): string + getValor(): float

Exemplo de subsistema vendas (C)

O Padrão FACADE - Solução

- Agora o Cliente não depende mais diretamente do subsistema.
- A classe FacadeVendas simplifica o processo para o cliente, se houver modificação no subsistema de venda basta editar essa classe.
- Embora o Facade simplifique a interface, o Cliente ainda é capaz de acessar o subsistema de forma direta se necessário.
- Cumpre o objetivo de simplificar a interface.

Output - FacadeSolucao (run)



run:

Email enviado para: Consumidor01

Pagamento realizado com sucesso (boleto)

BUILD SUCCESSFUL (total time: 0 seconds)

||