Padrões GRASP

Projeto de Software

Fabio Morais

Baseado no material dos Professores Nazareno Andrade e Hyggo Almeida

Atribuição de Responsabilidades

- **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns
  - Dividir responsabilidade entre objetos

- Responsabilidade de um objeto fazer algo
- Responsabilidade de conhecer/guardar algo

Principais padrões GRASP

- Information Expert
- Creator
- Baixo Acoplamento (Low Coupling)
- Alta Coesão (High Cohesion)

Information Expert

Information Expert

Princípio fundamental para atribuir responsabilidades

- Quem é o especialista da informação envolvida na operação?

Problema

- A qual classe deve ser atribuída uma responsabilidade?

Information Expert

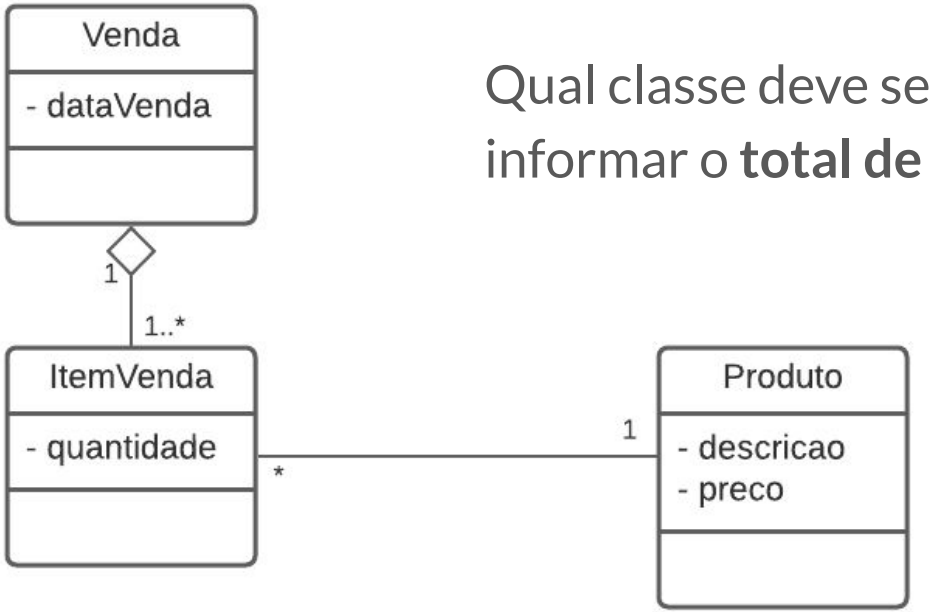
Problema

- A qual classe deve ser atribuída uma responsabilidade?

Exemplo do Mercado

- Qual classe deve ser responsável por saber / informar o total de uma venda?

Information Expert



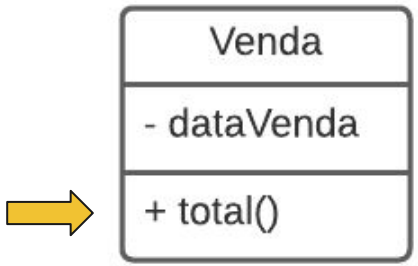
Qual classe deve ser responsável por saber / informar o **total de uma venda**?

Information Expert

Solução

- A classe que possui a **informação necessária** para realizar a operação

Qual classe deve **saber total de uma venda**?



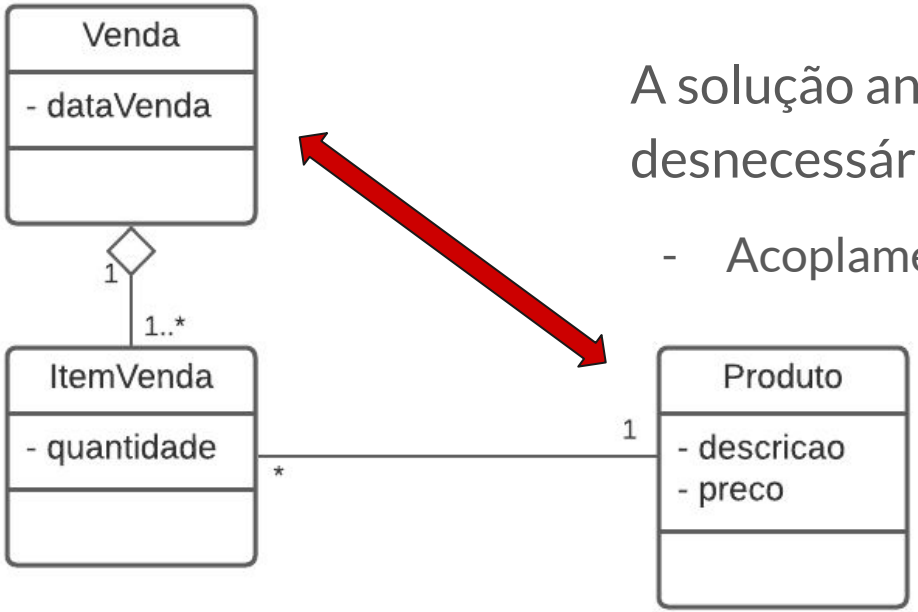
- A classe **Venda** é o especialista

Uma possível solução

```
public double total() {

    double total = 0;

    for (ItemVenda item : itens) {
        Produto produto = item.getProduto();
        int quantidade = item.getQuantidade();
        double preco = produto.getPreco();

        total += preco * quantidade;
    }
    return total;
}
```

Cada item consiste em uma **parte do total**

Uma possível solução

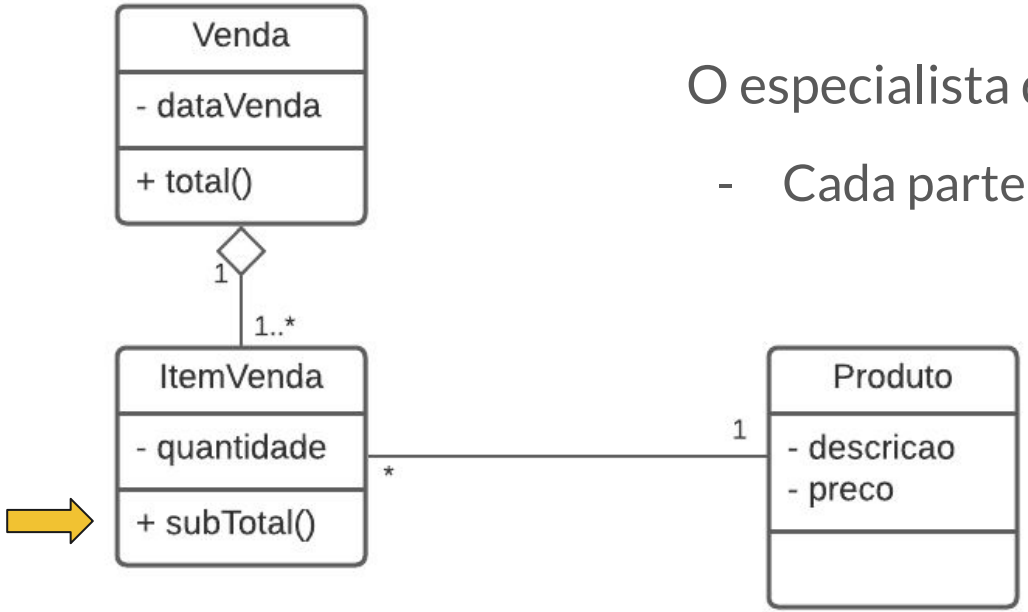


A solução anterior cria uma dependência desnecessária entre Venda e Produto

- Acoplamento

Quem é o especialista em cada parte?

Information Expert



O especialista de cada parte é **ItemVenda**

- Cada parte tem um **subtotal**

Solução usando Expert

Venda

```
public double total() {

    double total = 0;

    for (ItemVenda item : itens) {
        total += item.getSubTotal();
    }
    return total;
}
```

ItemVenda

```
public double getSubTotal() {
    return quantidade * produto.getPreco();
}
```

Evoluindo o sistema

- Cálculo do item de venda vai considerar um desconto
  - Quantidade de produtos maior que 200
  - Desconto de até 20% no valor do item de venda
  - O desconto depende do item de venda
- Onde o desconto deve ser calculado?

Evoluindo o sistema

- Quem deve manter a informação do valor de desconto?
- Seria natural considerar um atributo de produto
    - Desconto depende do item de venda e não do produto

Evoluindo o sistema

- Como seria a solução com a primeira implementação? 😬
  - Qual o número de classes impactadas?

- Como seria a implementação na solução expert?!

Expert

Consequências

- **Encapsulamento** é mantido
- **Baixo acoplamento** e facilidade de manutenção
- **Alta coesão**
    - Objetos realizam operações com base nas informações que possuem

Expert

Discussão

- É intuitivo - mas você tem que entender o contexto
- Vários especialistas “parciais” podem colaborar
- Objetos realizam sub operações no seu domínio
- Existem contraindicações (pode reduzir coesão)

Creator

Creator

Problema

- Quem deve criar novas instâncias de uma classe?

Exemplo do Mercado

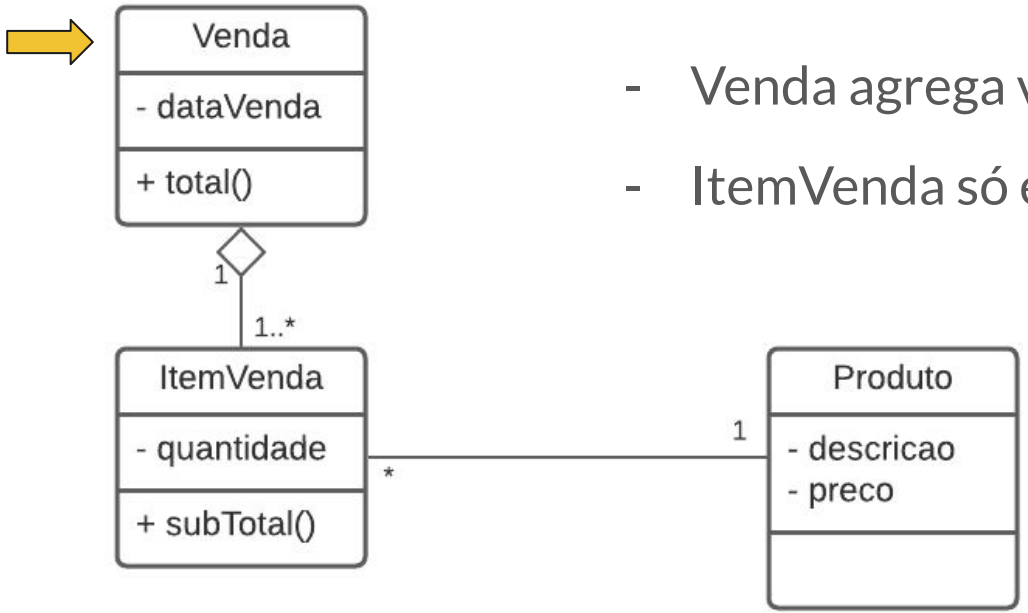
- Qual classe deve ser responsável por criar instâncias de ItemVenda?

Creator

Atribuir a uma classe **A** a responsabilidade de criar instâncias da classe **B** se:

- **A** contém objetos de **B**
- **A** registra instâncias de **B**
- **A** usa muitos objetos da classe **B**
- **A** possui os dados de inicialização de **B**

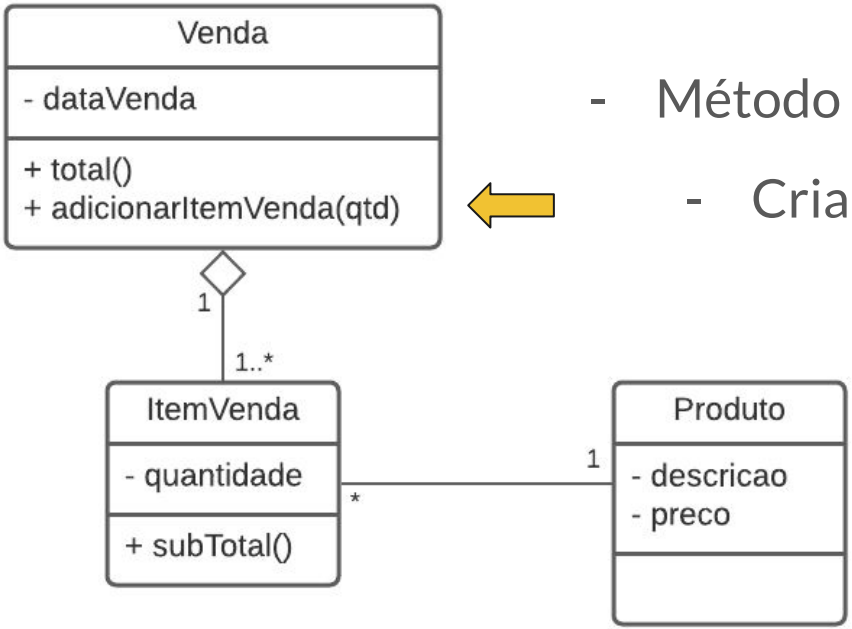
Creator



- Venda agrega vários ItemVenda
- ItemVenda só existem em Venda

Creator

```
public void adicionarItemVenda(int quantidade){
    this.itens.add(new ItemVenda(quantidade));
}
```



- Método **adicionarItemVenda** em **Venda**
  - Cria uma instância de Item Venda

Creator

Consequência

- **Baixo acoplamento**
- Venda já estaria acoplado com ItemVenda de toda forma

Low Coupling

Low Coupling (Baixo Acoplamento)

Acoplamento é uma medida de relação ou dependência entre elementos (classes)

Alto acoplamento: uma classe que depende de muitas outras

- Mais difícil de entender isoladamente
- Mais difícil de reutilizar (depende das outras classes)
- Sensível a mudanças nas outras classes (gelatina)

Low Coupling (Baixo Acoplamento)

Problema

- Como minimizar dependências e maximizar reúso?

Solução

- Atribuir responsabilidades visando minimizar o acoplamento

Low Coupling (Baixo Acoplamento)

Exemplo do Mercado Fácil

- Como permitir um pagamento associado a uma venda?

Uma classe Pagamento associada à classe Venda

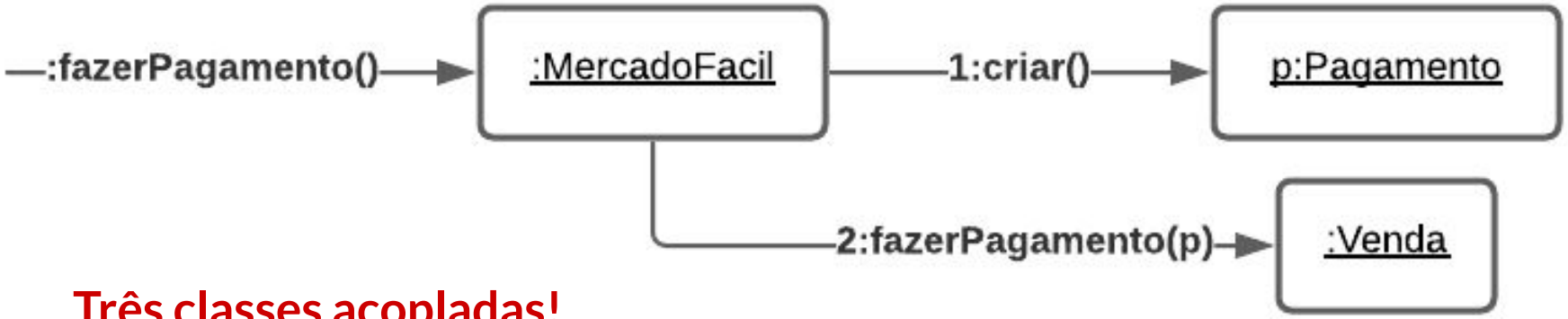
- MercadoFacil cria Pagamento e repassa a Venda (creator)

Solução de Pagamento

- MercadoFacil cria Pagamento e repassa a Venda (creator)

Solução de Pagamento

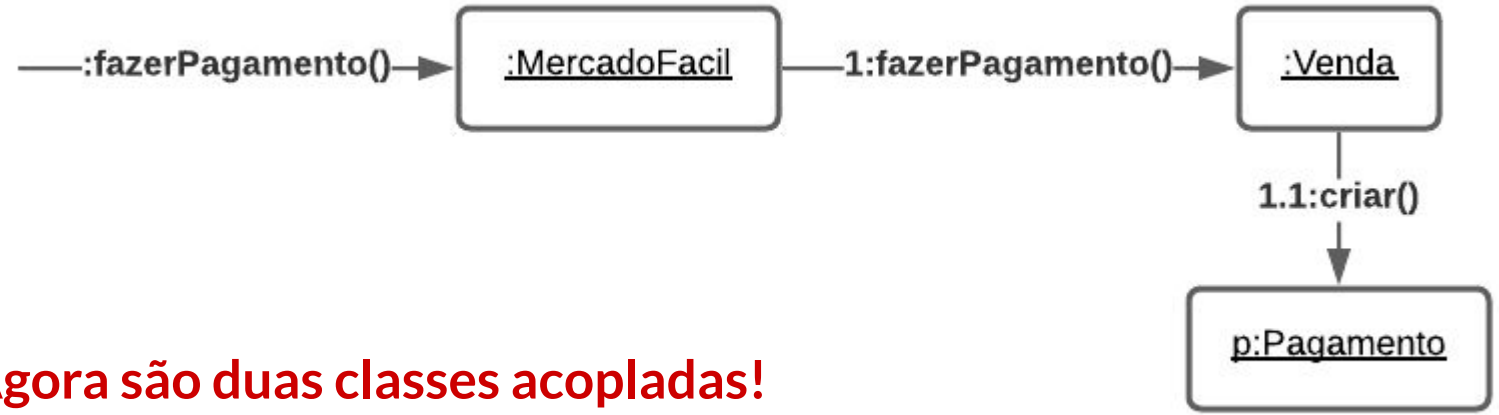
- MercadoFacil cria Pagamento e repassa a Venda (creator)



Três classes acopladas!

Solução de Pagamento sem Creator

- Ignorar o Creator em nome do baixo acoplamento



Agora são duas classes acopladas!

Low Coupling (Baixo Acoplamento)

Consequências

- Uma classe com baixo acoplamento não é afetada (ou é pouco afetada) por mudanças em outras classes
- Simples de entender isoladamente
- Facilita o reúso

Low Coupling (Baixo Acoplamento)

- Baixo acoplamento é o objetivo da maioria dos padrões
- Diferentes tipos de acoplamento
  - Acoplamento de passagem de dados
  - Acoplamento de controle
  - Acoplamento de dados globais
  - Acoplamento de dados internos

Acoplamento de passagem de dados

- O objeto **A** passa dados para **B**
  - Parâmetros
  - Estruturas de dados
- Exemplo comum
  - Objeto **A** passa o objeto **X** para **B** (**X** e **B** acoplados)
  - Mudanças na interface de **X** pode levar a mudanças em **B**

Acoplamento de controle

- *Flags* de controle entre objetos
  - Um objeto controla etapas de execução de outro objeto
- Exemplo comum
  - Objeto **A** manda uma mensagem para o objeto **B**
  - **B** usa um parâmetro da mensagem para decidir o que fazer

Acoplamento de controle

```java
public class Lampada {

    public final static int ON = 0;
    public void setLampada(int valor) {

        if (valor == ON) {
            // liga lampada
        } else if (valor == 1) {
            // desliga lampada
        } else if (valor == 2) {
            // pisca
        }
    }
}
```

```java
Lampada lampada = new Lampada();
lampada.setLampada(Lampada.ON);
lampada.setLampada(2);
```

Acoplamento de controle

- A solução é decompor em operações primitivas

```
public class Lampada {

    public void on() { /* liga lampada */ }
    public void off() { /* desliga lampada */ }
    public void pisca() { /* pisca */ }

}
```

Acoplamento de controle

- Outro exemplo comum
  - Objeto **A** manda uma mensagem para o objeto **B**
  - **B** retorna informação de controle para **A**
  - Por exemplo, um código de erro a ser tratado
- A solução é utilizar **exceções** para o tratamento de erros

Acoplamento de dados globais

- Objetos compartilham estruturas de dados globais
- Acoplamento não direto (escondido)
    - Uma chamada de método altera o dado global e o código não deixa isso evidente

Acoplamento de dados internos

- Um objeto altera dados locais de outro objeto
- Problemas de encapsulamento
    - Dados públicos, **package** ou mesmo **protected** (Java)

High Cohesion

High Cohesion (Alta Coesão)

Problema

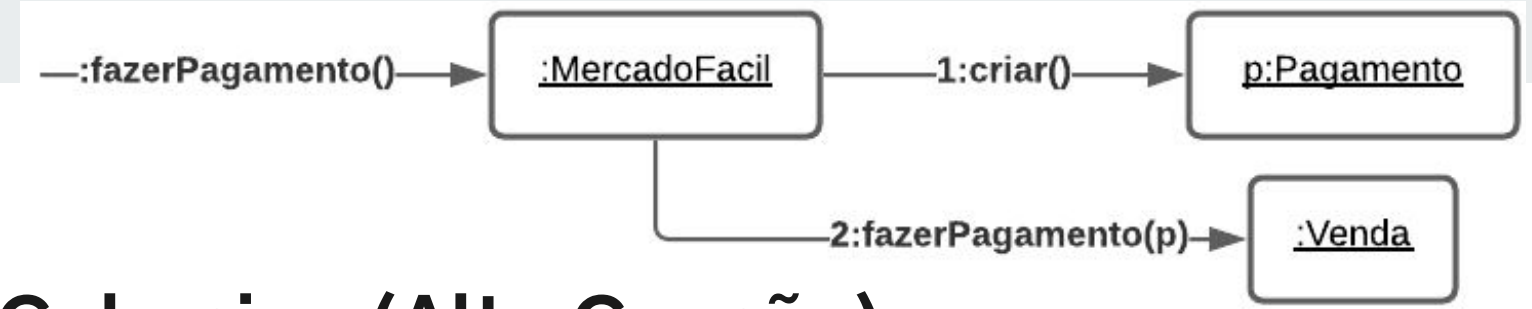
- Como gerenciar complexidade?
    - Responsabilidades no lugar certo?
    - Funcionalidades implementadas nas classes corretas?

Solução

- Buscar alta coesão pela atribuição de responsabilidades

High Cohesion (Alta Coesão)

- Classes com responsabilidades não relacionadas são:
  - Mais difíceis de entender
  - Mais complicadas para manter e reusar
  - Mais impactadas por mudanças em outras classes

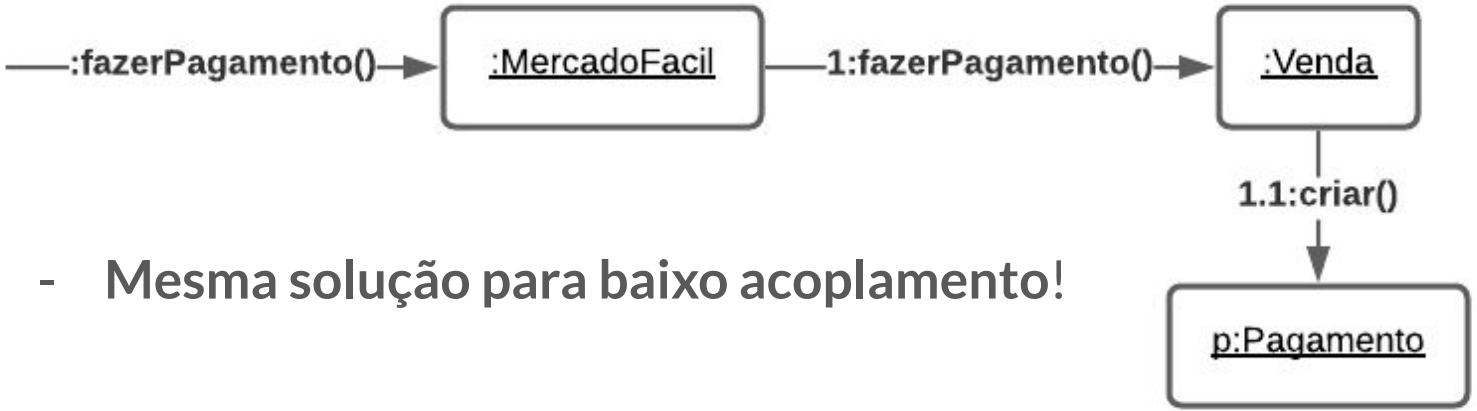


High Cohesion (Alta Coesão)

- Exemplo de MercadoFacil criando Pagamento (creator)
- Isso pode ocorrer também para outras classes
- MercadoFacil acumula responsabilidades não relacionadas
  - **Baixa coesão!**

High Cohesion (Alta Coesão)

- Delegar **fazerPagamento()** aumenta a coesão



- **Mesma solução para baixo acoplamento**!

High Cohesion (Alta Coesão)

Consequências

- Maior facilidade de compreensão do projeto
- Simplificação da manutenção do código
- Relação positiva com o baixo acoplamento