

# Projetando Software com Responsabilidade

Matheus Barbosa  
matheus.barbosa@dcx.ufpb.br

Um sistema orientado a objetos  
é composto de objetos que  
enviam mensagens a outros  
objetos para completar  
operações

Responsabilidades são  
atribuídas aos objetos durante o  
design

Responsabilidade é um  
**contrato** ou obrigação de um  
tipo, ou classe.

## Tipos de responsabilidades dos objetos

- De conhecimento (*knowing*): sobre dados privados e encapsulados; sobre objetos relacionados; sobre coisas que pode calcular ou derivar.
- De realização (*doing*): fazer alguma coisa em si mesmo; iniciar uma ação em outro objeto; controlar e coordenar atividades em outros objetos.

Projetar software orientado a  
objetos é **difícil**, mas projetar  
software **reutilizável** orientado a  
objetos é ainda **mais**  
**complicado**

O seu projeto deve ser  
específico para o problema a  
resolver, mas também genérico  
o suficiente para atender  
problemas e requisitos futuros

Os melhores projetistas sabem  
que não devem resolver  
problemas a partir de princípios  
elementares ou do zero.



Em vez disso, eles reutilizam  
soluções que funcionaram no  
passado.

Consequentemente, você encontrará **padrões**, de classes e de comunicação entre objetos, que reaparecem frequentemente

Padrões são uma maneira  
testada ou documentada de  
alcançar um objetivo qualquer

Padrões de projeto são soluções típicas para problemas comuns em projeto de software

# Padrões tem um formato estruturado:

- Nome
- Problema que soluciona
- Solução do problema

# Também podem aparecer:

- Consequências
- Estruturas
- Exemplos de código

O objetivo dos padrões é  
codificar conhecimento  
existente de uma forma que  
possa ser **reaplicado** em  
contextos diferentes

É difícil achar um sistema  
orientado a objetos que não use  
pelo menos dois desses  
padrões, e grandes sistemas  
usam quase todos eles



## Por que aprender padrões?

- Aprender com a experiência dos outros
- Aprender a programar bem com orientação a objetos
- Desenvolver software de melhor qualidade
- Vocabulário comum
- Ajuda na documentação e na aprendizagem

Há vários catálogos de padrões em software:

Ex: <https://refactoring.guru/pt-br/design-patterns>

Martin Fowler, Analysis Patterns -  
Reusable Object Models,  
Addison-Wesley, 1997.

# ANALYSIS PATTERNS

REUSABLE OBJECT MODELS

**MARTIN FOWLER**

Forewords by  
Ward Cunningham and Ralph Johnson



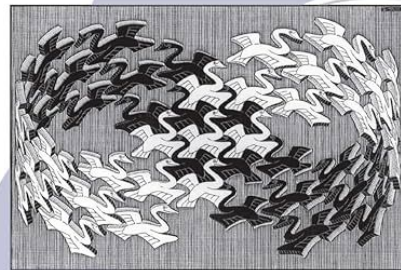
# Padrões GoF

O livro "**Design Patterns**" (1994) de Erich Gamma, John Vlissides, Ralph Johnson e Richard Helm, descreve 23 padrões de design

## Design Patterns

Elements of Reusable  
Object-Oriented Software

Erich Gamma  
Richard Helm  
Ralph Johnson  
John Vlissides



Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch



# GRASP: General Responsibility and Assignment Software Patterns

Introduzidos por Craig  
Larman em seu livro  
“Applying UML and  
Patterns”

## APPLYING UML AND PATTERNS

An Introduction to Object-Oriented Analysis and Design  
and Iterative Development

THIRD EDITION



“People often ask me which is the best book to introduce them to the world of OO design. Ever since I came across it, *Applying UML and Patterns* has been my unreserved choice.”

—Martin Fowler, author of *UML Distilled* and *Refactoring*

**CRAIG LARMAN**

Foreword by Philippe Kruchten



```
public class App {  
    public static void main(String[] args) {  
        String nome = "Caneta Azul";  
        double preco = 2.50;  
  
        // Validação misturada  
        if (nome == null || nome.isEmpty() || preco ≤ 0) {  
            System.out.println("Erro: dados inválidos.");  
            return;  
        }  
  
        // Simula persistência  
        System.out.println("Salvando no banco...");  
        System.out.println("Produto salvo: " + nome + " - R$" + preco);  
    }  
}
```

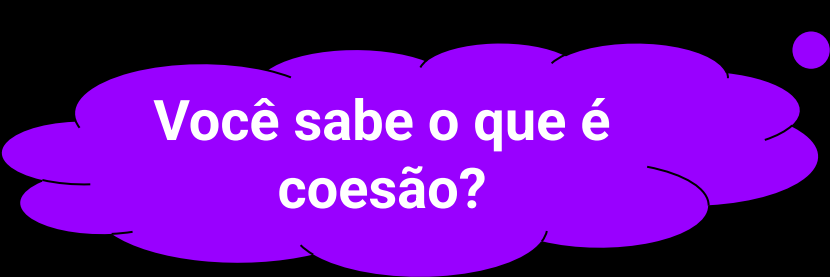
## Padrões Grasp básicos

- Information Expert (Especialista de Informação)
- Creator (Criador)
- High Cohesion (Coesão alta)
- Low Coupling (Baixo acoplamento)
- Controller (Controlador)

## High Cohesion (Coesão alta)

**Problema:** Como manter a complexidade sob controle? Classes que fazem muitas tarefas não relacionadas são mais difíceis de entender, de manter e de reusar, além de serem mais vulneráveis à mudança.

**Solução:** Atribuir uma responsabilidade para que a coesão se mantenha alta



Você sabe o que é coesão?



## Coesão

Uma medida de quão **relacionadas** estão as responsabilidades de um elemento.

Uma classe **Cão** é coesa se tem operações relacionadas ao Cão (morder, correr, comer, latir) e apenas ao Cão (não terá por exemplo, validar, imprimirCao, listarCaes)

## Low Coupling (Baixo acoplamento)

**Problema:** Como suportar dependência baixa e aumentar a reutilização?

**Solução:** Atribuir uma responsabilidade para que o **acoplamento** mantenha-se fraco.

# Acoplamento

É uma medida de quanto um elemento está conectado a, ou depende de outros elementos

Uma classe com acoplamento forte depende de muitas outras classes: tais classes podem ser indesejáveis

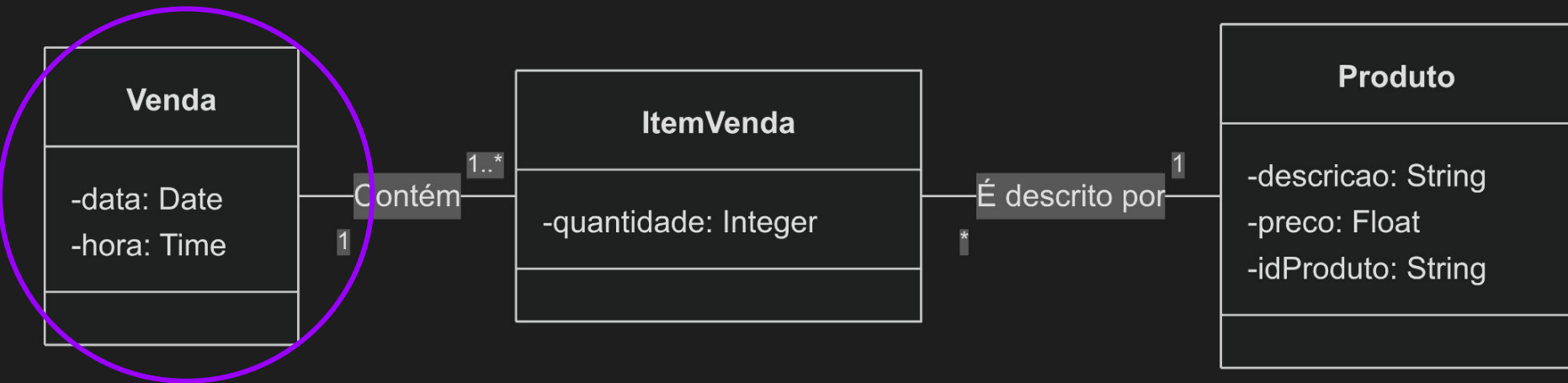
O acoplamento está associado à coesão: maior coesão, menor acoplamento e vice-versa.

## Expert (especialista de informação)

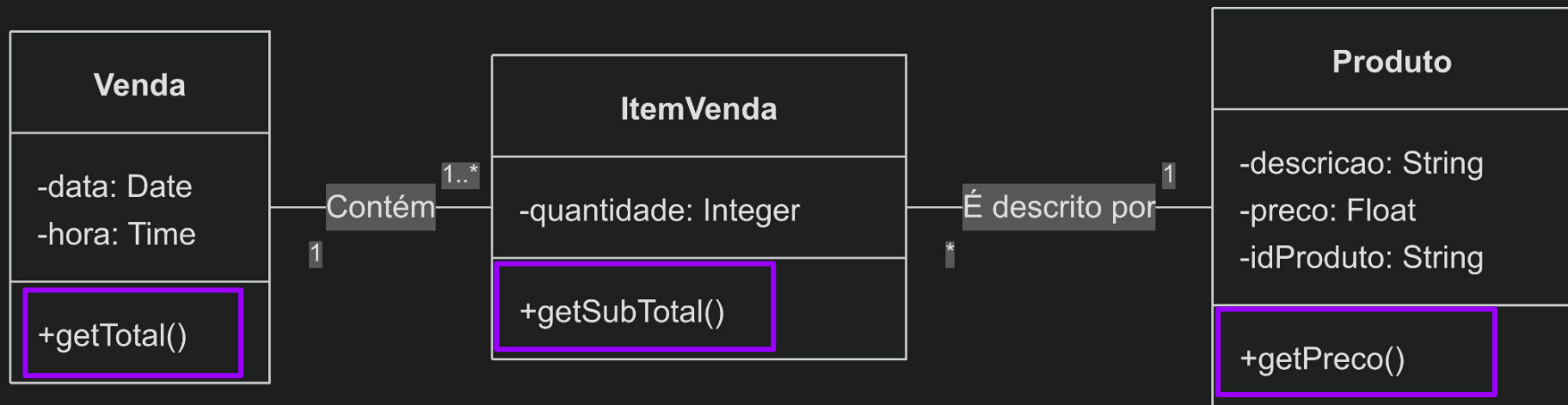
**Problema:** Dado um comportamento (responsabilidade) a qual classe essa **responsabilidade** deve ser alocada?

**Solução:** Atribuir responsabilidade ao **especialista da informação** – a classe que tem a informação necessária para satisfazer a responsabilidade.

Quem deve ser o **responsável** por gerar o valor total da venda?



Que **informação é necessária** para obter o valor total?



A classe **Venda** vai calcular o valor total, somando os subtotais. A responsabilidade para cada subtotal é atribuída ao objeto **ItemVenda**. O subtotal depende do preço. O objeto **Produto** é o especialista que conhece o preço, portanto a responsabilidade é dele.

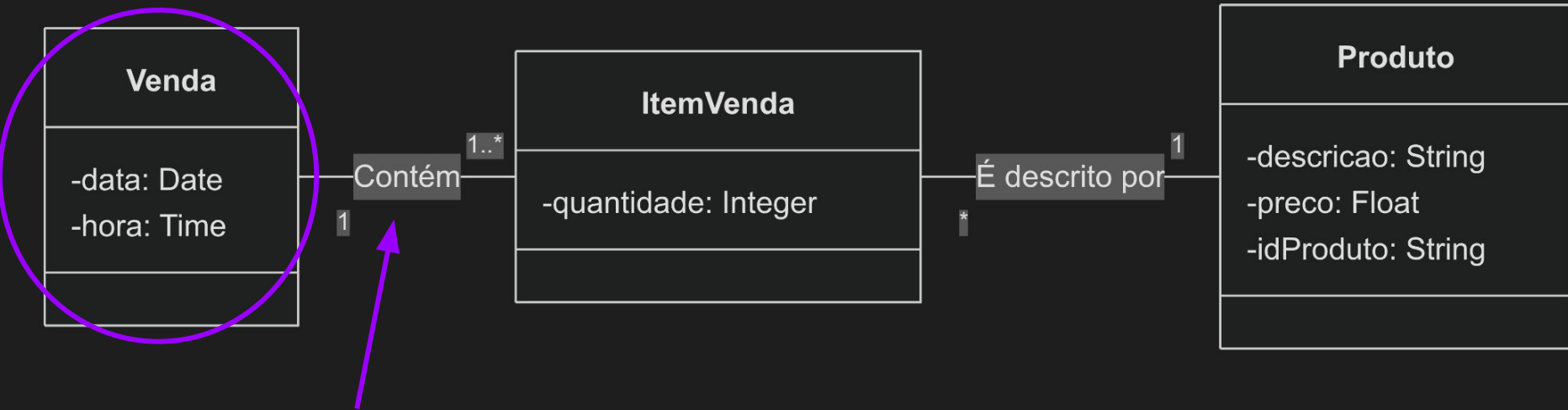
## Creator (Criador)

**Problema:** quem deve ser o responsável por criar instâncias de uma determinada classe?

**Solução:** Atribua a **B** a responsabilidade de criar **A** se:

- **B agrega A**
- **B contém A**
- **B guarda instâncias de A**
- **B faz uso de A**
- **B possui dados para inicialização de A**, quando ele for criado.

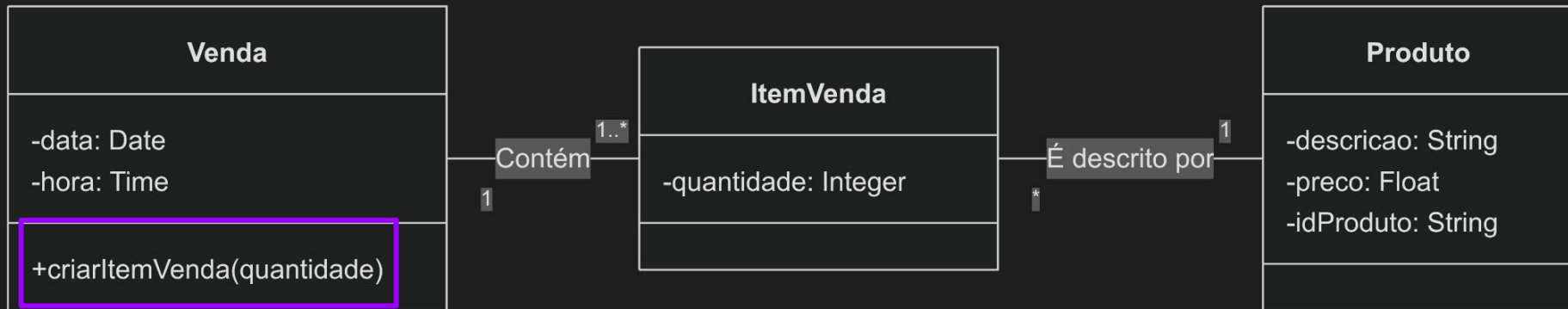
Que classe deve ser responsável por criar uma instância do objeto **ItemVenda** abaixo



**Venda** **agrega** muitos **ItemVenda**



Um novo método `criarItemVenda(...)` deve ser criado na classe `Venda`

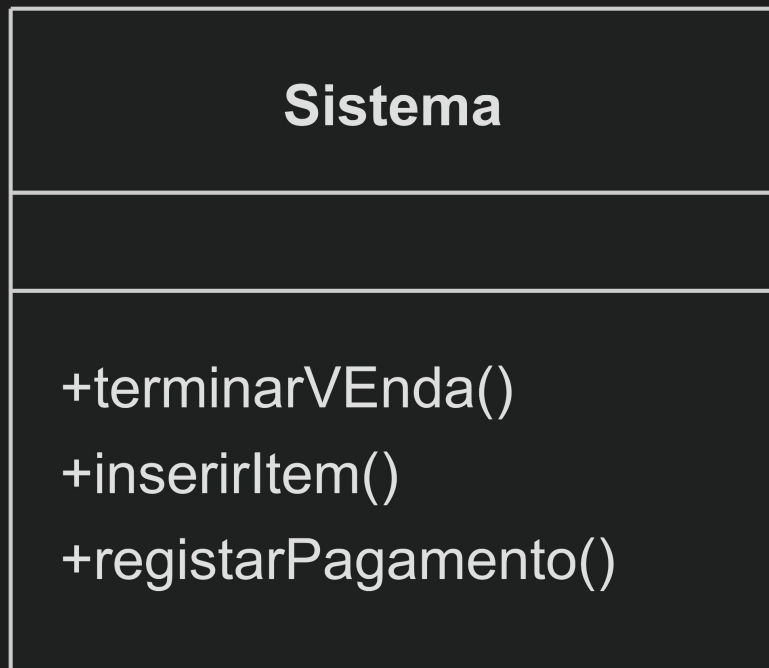


## Controller (Controlador)

**Problema:** Quem deve ser o responsável por lidar com um **evento** de uma **interface de entrada**?

**Solução:** Atribuir responsabilidades para receber ou lidar com um evento do sistema para uma classe que representa todo o sistema (controlador de fachada – front controller), um subsistema e um cenário de caso de uso (controlador de caso de uso ou de sessão).

**Que classe é responsável por criar um pagamento e associá-lo a uma venda?**



```
public class App {  
    public static void main(String[] args) {  
        String nome = "Caneta Azul";  
        double preco = 2.50;  
  
        // Validação misturada  
        if (nome == null || nome.isEmpty() || preco ≤ 0) {  
            System.out.println("Erro: dados inválidos.");  
            return;  
        }  
  
        // Simula persistência  
        System.out.println("Salvando no banco...");  
        System.out.println("Produto salvo: " + nome + " - R$" + preco);  
    }  
}
```

```
package domain;

public class Produto {
    private String nome;
    private double preco;

    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    // Metodos get e set...

    public boolean dadosValidos() {
        return nome != null && !nome.isEmpty() && preco > 0;
    }
}
```

```
package service;

import domain.Produto;

public class ProdutoService {
    public boolean cadastrar(String nome, double preco) {
        Produto produto = new Produto(nome, preco);

        if (!produto.dadosValidos()) {
            return false;
        }
        // Simulando persistência
        salvar(produto);
        return true;
    }

    private void salvar(Produto produto) {
        System.out.println("[Service] Salvando produto: " + produto.getNome()
            + " - R$ " + produto.getPreco());
    }
}
```

```
package controller;

import service.ProdutoService;

public class ProdutoController {
    private ProdutoService service;

    public ProdutoController() {
        this.service = new ProdutoService();
    }

    public void cadastrarProduto(String nome, double preco) {
        boolean sucesso = service.cadastrar(nome, preco);
        if (sucesso) {
            System.out.println("[Controller] Produto cadastrado com sucesso.");
        } else {
            System.out.println("[Controller] Erro ao cadastrar produto.");
        }
    }
}
```



```
public class Main {  
    public static void main(String[] args) {  
        ProdutoController controller = new ProdutoController();  
        controller.cadastrarProduto("Caneta Azul", 2.50);  
    }  
}
```



```
src/  
├─ Main.java  
├─ controller/  
│   └─ ProdutoController.java  
├─ domain/  
│   └─ Produto.java  
├─ service/  
│   └─ ProdutoService.java
```

**Atividade: “Quem é o Responsável?”**

**“Um sistema de estacionamento precisa controlar entrada de carros, calcular valor a pagar e registrar saída”**

**Atividade: “Quem é o Responsável?”**

Criar um novo Ticket quando o carro entra.

**Atividade: “Quem é o Responsável?”**

Guardar o horário de entrada e saída.

**Atividade: “Quem é o Responsável?”**

**Calcular o valor do estacionamento.**

**Atividade: “Quem é o Responsável?”**

**Calcular o valor do estacionamento.**

**Atividade: “Quem é o Responsável?”**

Registrar a saída do carro.