Análise e Projeto Orientados por Objetos

Aula 03 – Padrões de Projeto GRASP

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

Padrões de Projeto de Software

- Problemas no desenvolvimento de software se repetem...
  - Bons desenvolvedores aplicam boas soluções para resolve-los.
- Por que não reutilizar as **boas soluções** em outros projetos?
  - Padrões de software!
- É fundamental ter um repertório de soluções, de Padrões de Projeto.

Padrões de Projeto de Software

- **O que são padrões de software?**
  - A **descrição** de um problema que ocorre com freqüência;
  - A **base de uma solução** para este problema com um nome;

- Baseado na **reutilização de ideias** (não de código).

- Padrões e mais padrões...
  - Padrões de análise, padrões de testes, padrões de negócio, padrões pedagógicos, padrões arquiteturais, padrões de projeto (Design Patterns)...

Elementos Essenciais de um Padrão de Software

- **Nome:**
  - Resume em uma ou duas palavras: o problema, as soluções e consequências do uso do padrão;
  - Deve ser facilmente lembrado, reflete o conteúdo do padrão;
- **Problema:**
  - Descreve quando aplicar o padrão. Explica o problema, seu contexto, sintomas e condições;
- **Solução:**
  - Elementos que constituem o design, seus relacionamentos, responsabilidades e colaboradores.
- **Consequências:**
  - Resultados e compromissos decorrentes da aplicação do padrão;
  - Impactos sobre a flexibilidade, extensibilidade, portabilidade ou desempenho do sistema.

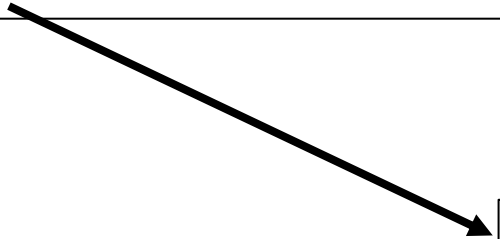
Elementos Essenciais de um Padrão de Software

- **Outros elementos importantes:**
  - Um exemplo de uso;
  - A razão que justifica a solução escolhida;
  - Padrões relacionados;
  - Uso conhecido do padrão;
  - Lista de outros nomes para o padrão;
  - Amostra de código em uma linguagem de programação.

Elementos Essenciais de um Padrão de Software

- **Nome:**
  - Deve ser sugestivo/representativo do problema ou solução;
  - Apenas pelo nome, sabe-se a "**natureza**" do padrão;
  - Todo mundo conhece os padrões:
    - Nomes padronizados em inglês;
    - Independente da equipe, estabelece-se comunicação de alto nível;

Que padrão deve ser utilizado para que se tenha apenas **uma instância** de uma classe?



Singleton

Elementos Essenciais de um Padrão de Software

- **Nome:**
  - Permite projetar em um nível mais alto de abstração;
  - Exemplos de conversa entre desenvolvedores:
    - “*Cara, acho melhor usar o Template Method aqui!!!*”;
    - “*Não sei não... um Strategy funcionaria melhor!!!*”;
    - “*Maria, coloca um Observer aí que resolve!!!*”;
  - Simplifica a documentação;

Elementos Essenciais de um Padrão de Software

- **Nome:**
  - Simplifica a documentação evitando longas descrições:

```
/** Nesta classe utiliza-se um
* construtor privado, com um método
* estático que retorna a única instância
* desta classe, sincronizado para evitar
* que outra instância seja recuperada
* por outra linha de execução...
* @author Joao da Silva
* @version 1.0
*/
public class Calendar{
...
```

```
/** Nesta classe implementa-se
* o padrão Singleton...
* @author Joao da Silva
* @version 1.0
*/
public class Calendar{
```

Elementos Essenciais de um Padrão de Software

- **Problema:**
  - Em que situações o padrão pode ser aplicado?
  - Em que situações o padrão traz flexibilidade/elegância ao projeto?
  - Quando não utilizá-lo?
  - O seu problema é similar ao descrito no padrão??? Se sim, basta uma adaptação da solução!!!
  - Geralmente o problema tem exemplos específicos de aplicação:
    - Como definir uma instância única de uma classe???
    - Como representar algoritmos como objetos???

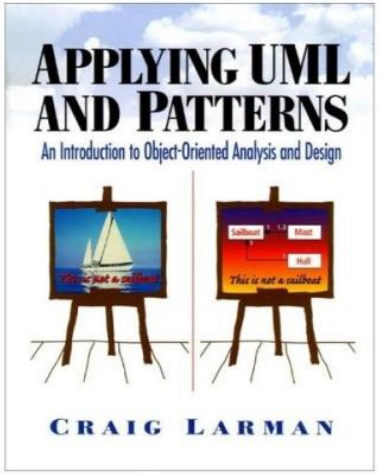
Elementos Essenciais de um Padrão de Software

- **Solução:**
  - Descreve os elementos que compõem o projeto da solução, suas responsabilidades e colaborações:
    - Modelo conceitual;
    - Diagrama de classes;
    - Diagramas de interação;
  - Solução não descreve uma implementação concreta, apenas um modelo genérico para que possa ser reutilizado
    - Dependendo do seu problema, deverá ser adaptado
  - Padrão é um gabarito de solução:
    - “Se João fez dessa forma e funcionou... porque não funcionaria comigo???”

Elementos Essenciais de um Padrão de Software

- **Consequências:**
  - Resultados do uso do padrão;
  - Vantagens e desvantagens;
  - Visão crítica: “o que se perde?”;
  - Análise de impactos sobre;
    - Flexibilidade;
    - Extensibilidade
    - Reuso;
    - Desempenho;

Padrões GRASP



- GRASP: General Responsibility Assignment Software Patterns.
- Padrões de análise catalogados por Craig Larman.

- Fornecem uma abordagem sistemática para a atribuição de responsabilidades às classes do projeto:
  - **De conhecimento (knowing):** sobre dados privativos 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.

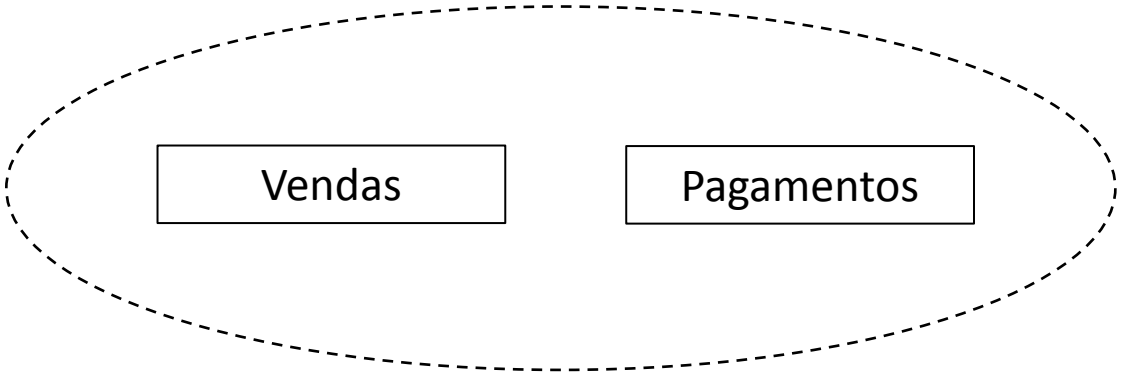
Padrões GRASP

- **Padrões básicos:**
  - Information Expert
  - Creator
  - High Cohesion
  - Low Coupling
  - Controller
- **Padrões avançados:**
  - Polymorphism
  - Pure Fabrication
  - Indirection
  - Protected Variations

Exemplo de Sistema

- Para a ilustração dos padrões GRASP analisaremos o exemplo de um sistema de vendas e pagamentos.
  - Padrões GRASP são utilizados quase sempre durante a etapa de análise.

- Sistema:
  - Grava vendas e gera pagamentos

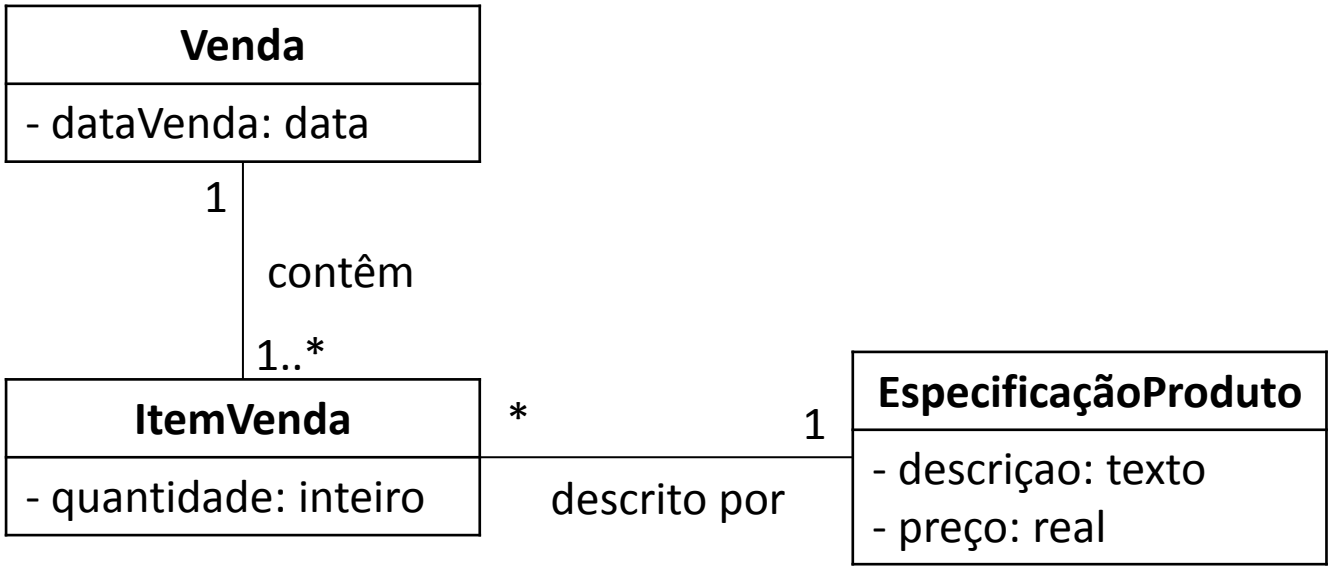


GRASP: Information Expert

- Princípio fundamental para atribuir responsabilidade.
- **Problema:** Qual é o princípio geral para a atribuição de responsabilidades aos objetos?
- **Solução:** Atribua a responsabilidade ao especialista: a classe que tem as informações necessárias para assumir a responsabilidade.

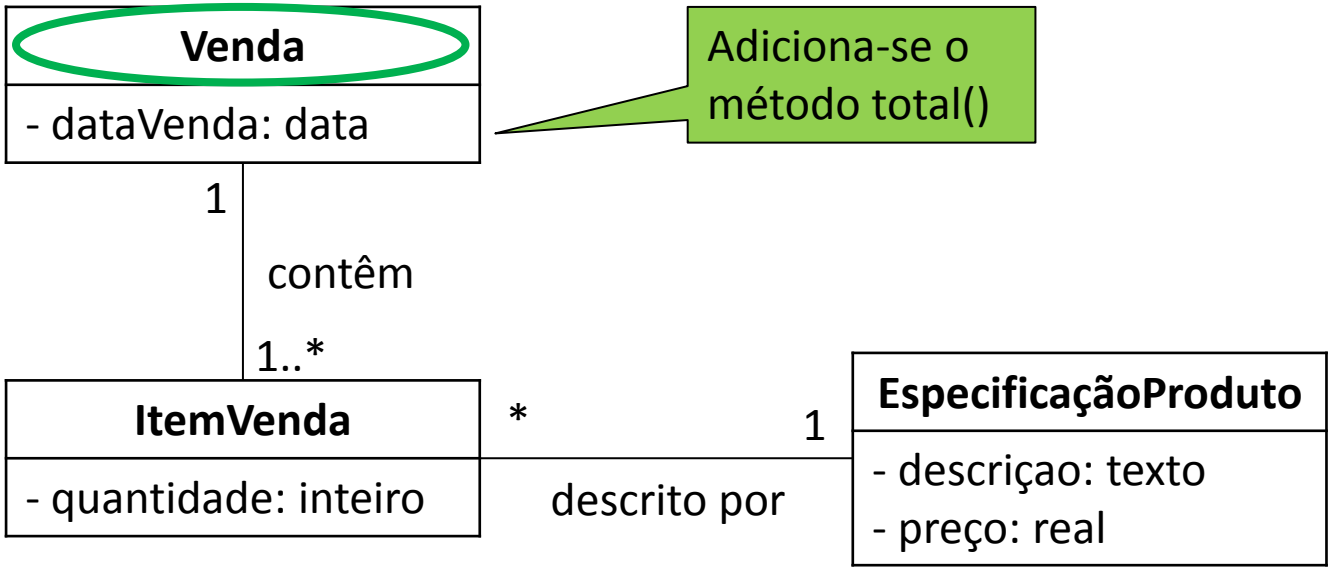
GRASP: Information Expert

- Quem deve ser responsável por conhecer o total da venda?



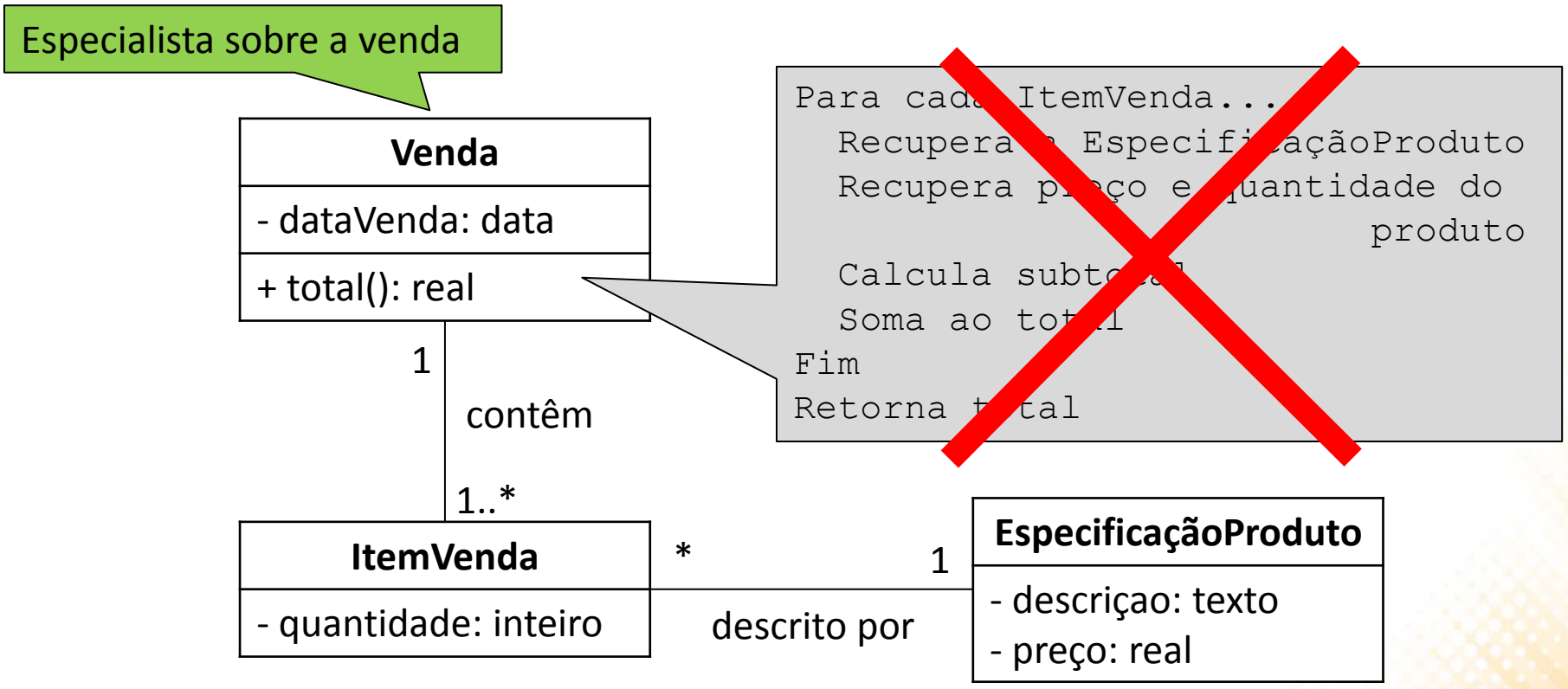
GRASP: Information Expert

- Quem deve ser responsável por conhecer o total da venda?



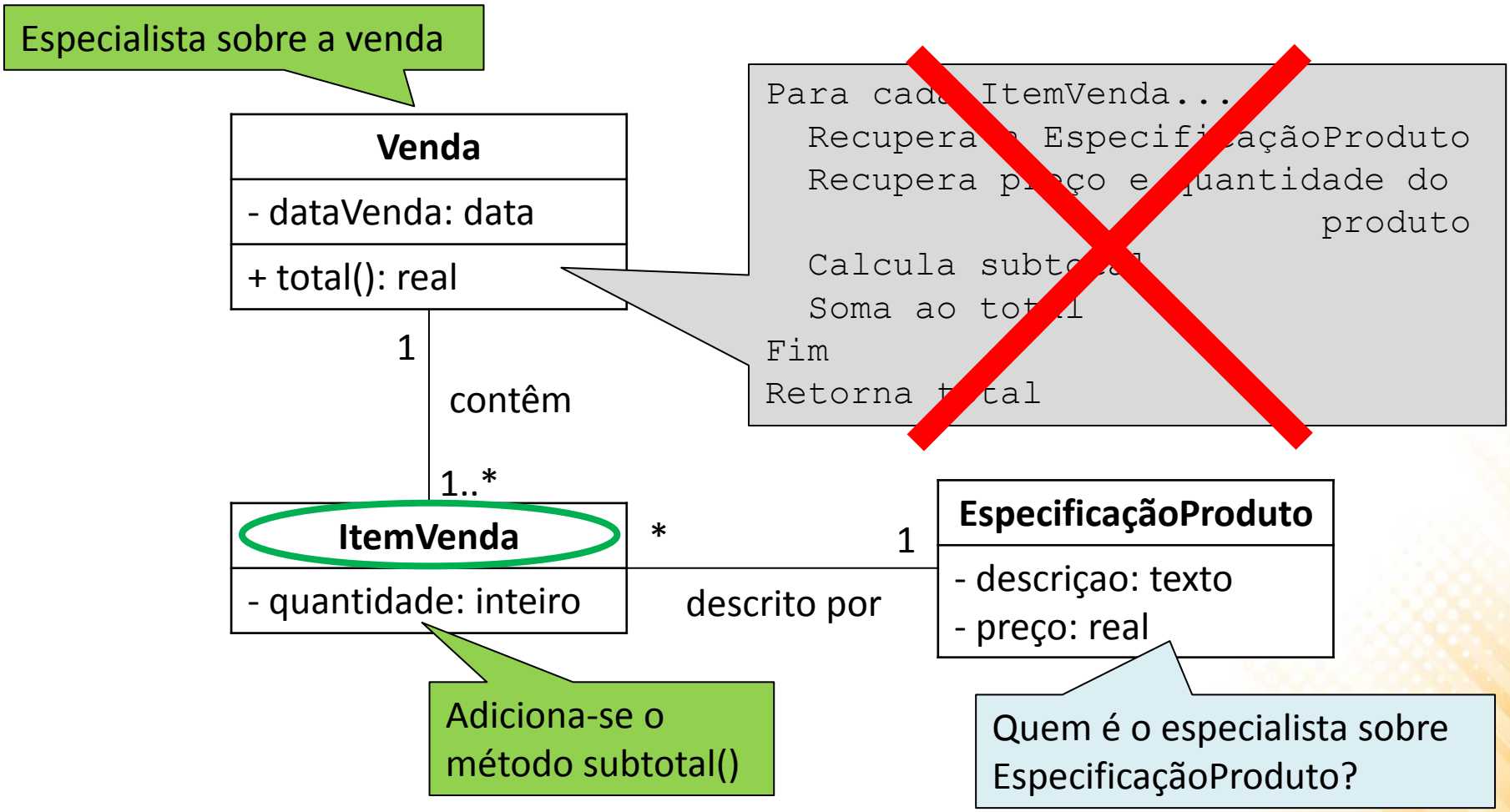
GRASP: Information Expert

- Quem deve ser responsável por conhecer o total da venda?



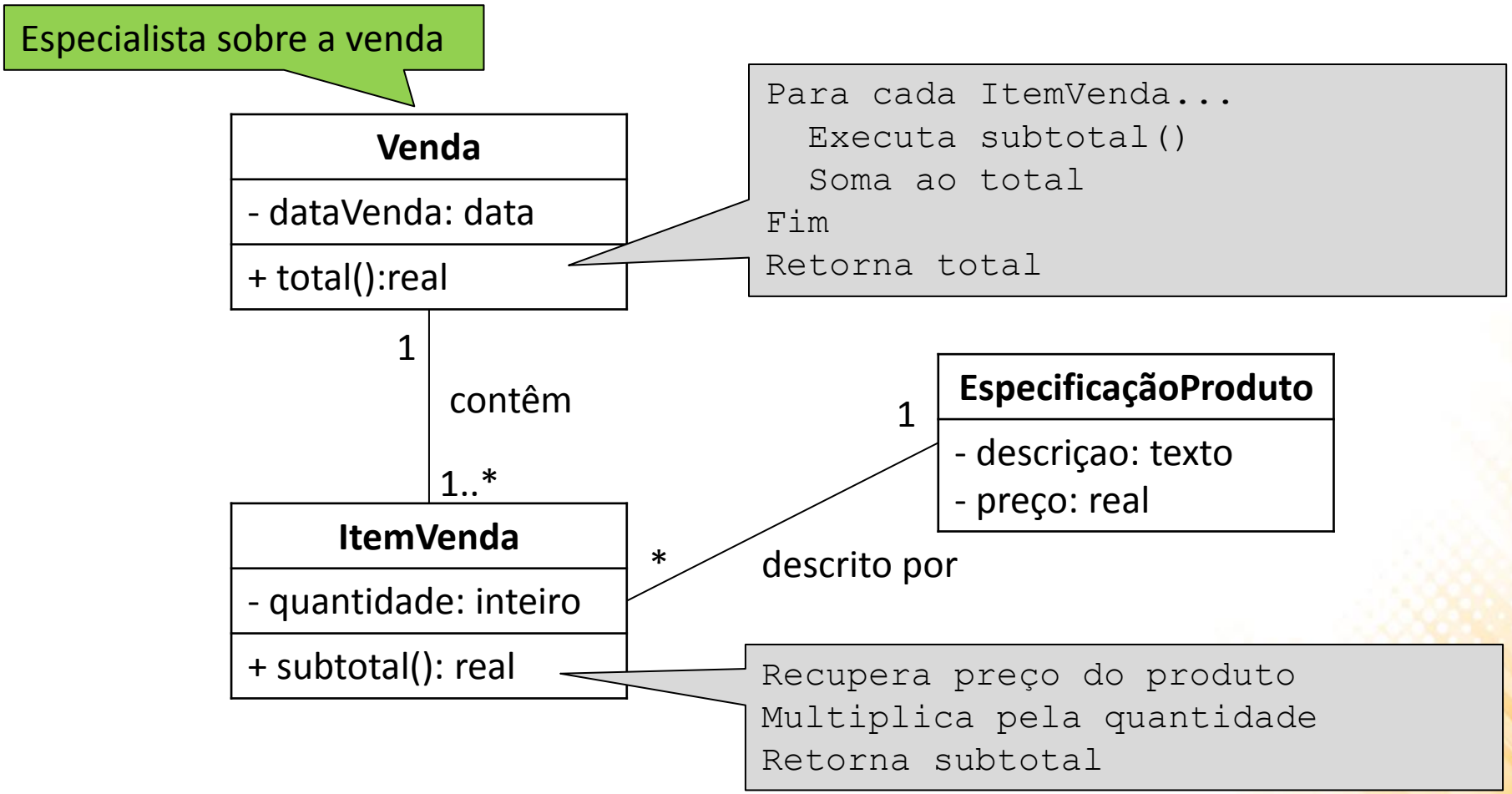
GRASP: Information Expert

- Quem deve ser responsável por conhecer o total da venda?



GRASP: Information Expert

- Quem deve ser responsável por conhecer o total da venda?



GRASP: Information Expert

- **Benefícios:**
  - Encapsulamento é mantido;
  - Fraco acoplamento (facilidade de manutenção);
  - Alta coesão (objetos fazem tudo relacionado à sua própria informação).

- No fim, vários especialistas "parciais" podem colaborar;

- Com o Expert, vários objetos inanimados podem ter ações;

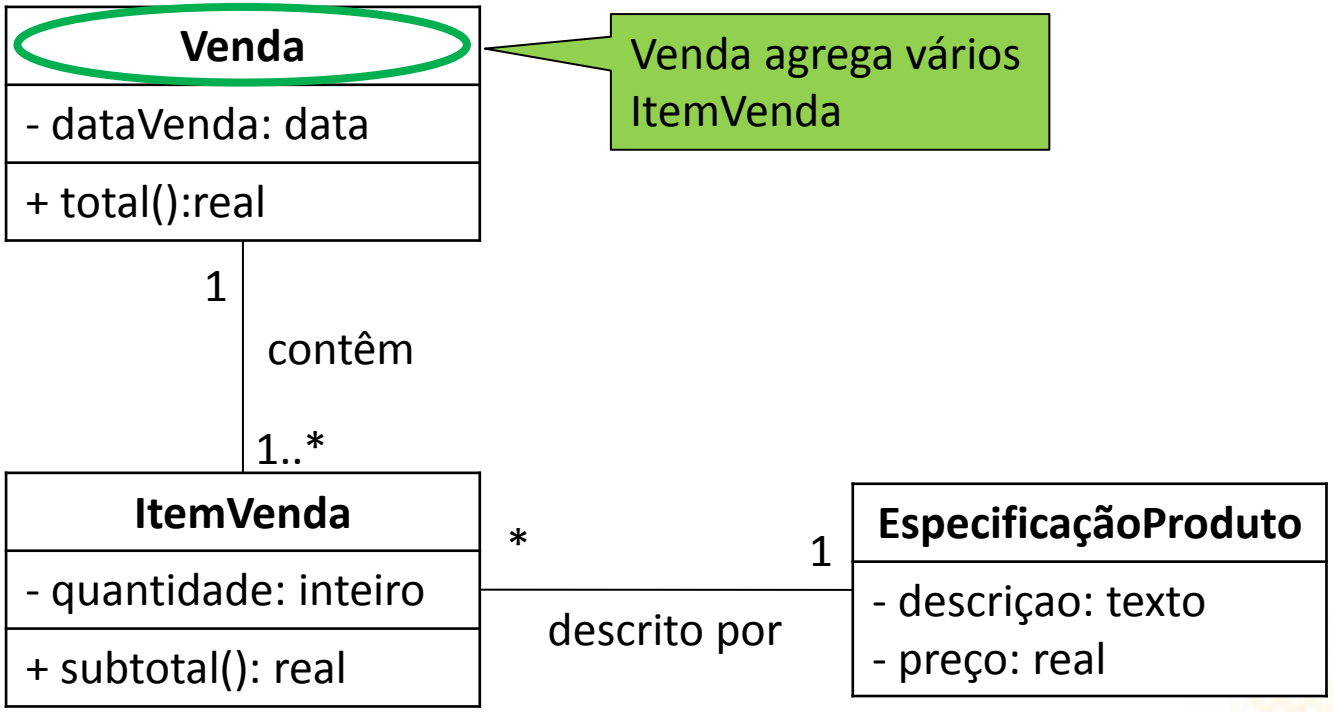
GRASP: Creator

- **Problema:** Quem deve ser responsável por criar uma nova instância de uma classe?

- **Solução:** Atribua à classe B a responsabilidade de criar uma instância de A se pelo menos um desses for verdadeiro (quanto mais melhor):
  - B contém ou agrega objetos de A;
  - B registra instancias de A;
  - B usa muitos objetos de A;
  - B tem os dados necessários para a inicialização de A que serão passados ao construtor de A;

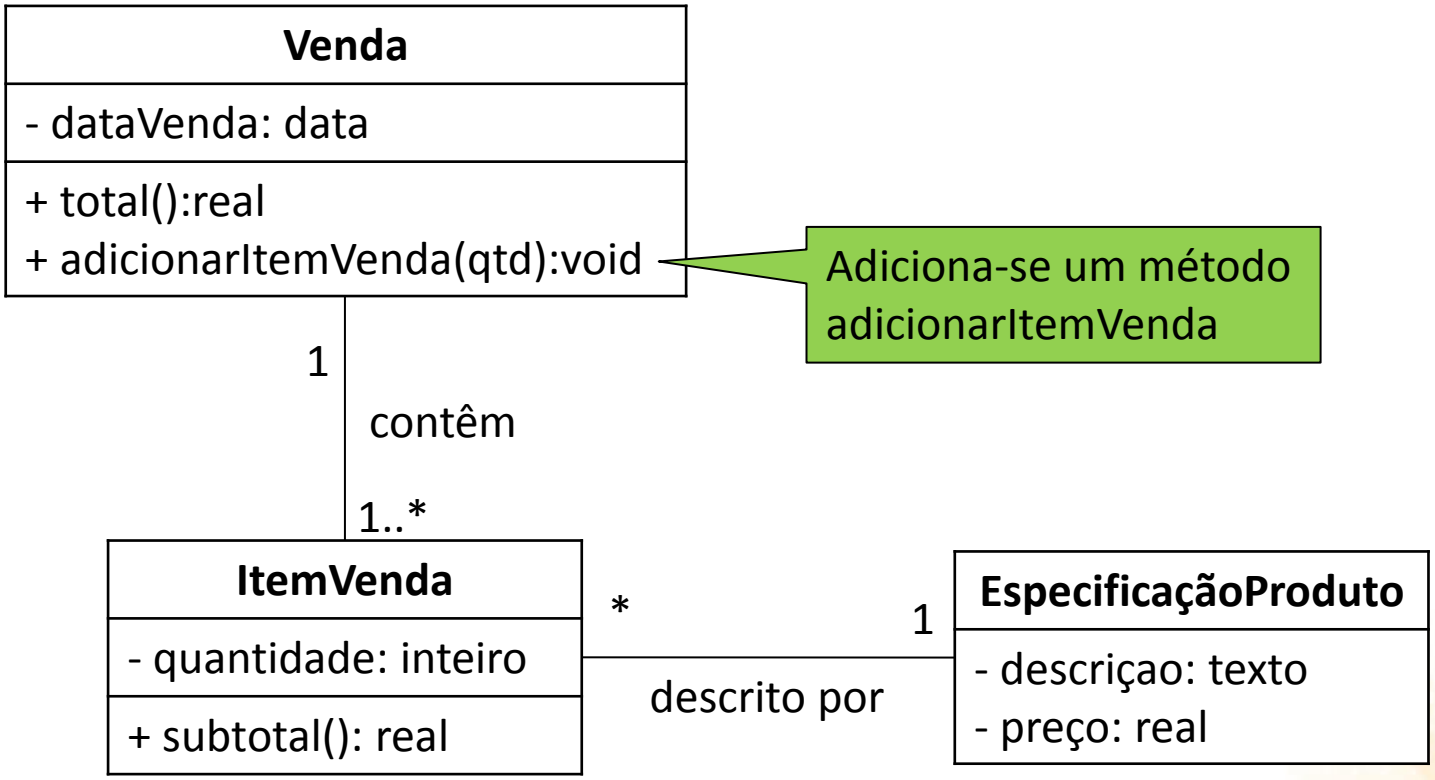
GRASP: Creator

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



GRASP: Creator

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



GRAPS: Low Coupling (Baixo Acoplamento)

- 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.
  - Por isto, acoplamento forte é indesejável.

- **Problema:** Como prover baixa dependência entre classes, reduzir o impacto de mudanças e obter alta reutilização?

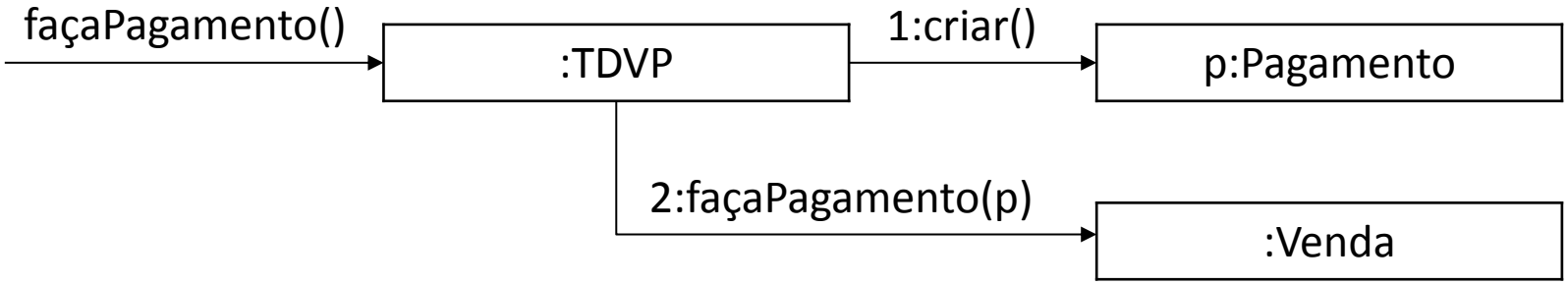
- **Solução:** Atribua as responsabilidades de modo que o acoplamento entre classes permaneça baixo. Use este princípio para avaliar alternativas.

GRAPS: Low Coupling (Baixo Acoplamento)

- Uma classe com acoplamento forte é:
  - Mais difícil de compreender isoladamente;
  - Mais difícil de reutilizar (seu uso depende da reutilização das outras classes da qual ela depende);
  - Sensível a mudanças nas classes associadas.

GRAPS: Low Coupling (Baixo Acoplamento)

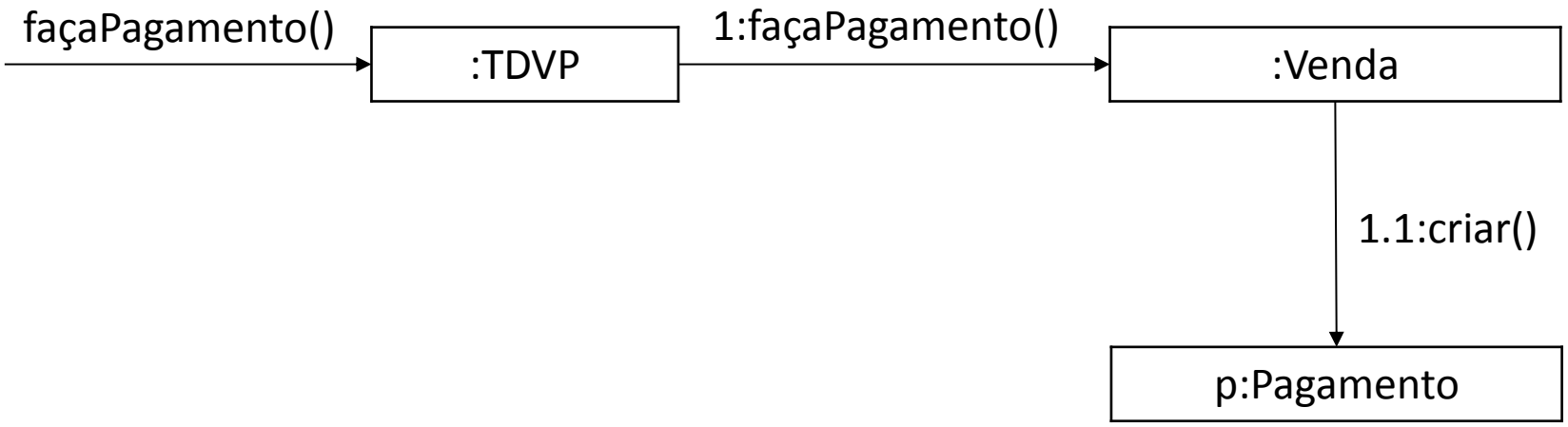
- Suponha que temos de criar um objeto pagamento e associá-lo à venda. Que classe deve ser responsável por isso?
  - Segundo o padrão Creator, o Terminal de Vendas e Pagamentos (TDVP) deveria criar Pagamento e repassá-lo a Venda.



- **Problema:** três classes acopladas!

GRAPS: Low Coupling (Baixo Acoplamento)

- Então, em nome do baixo acoplamento, ignoramos o Creator!



- Agora temos acoplamento entre duas classes apenas.

GRAPS: Low Coupling (Baixo Acoplamento)

- **Acoplamento de dados:**
  - Objeto a passa objeto x para objeto b;
  - Objeto x e b estão acoplados;
  - Uma mudança na interface de x pode acarretar mudanças em b;
- **Acoplamento de controle:**
  - Passar flags de controle entre objetos de forma que um objeto controle as etapas de processamento de outro objeto;
  - Ocorrência comum:
    - Objeto a manda uma mensagem para objeto b;
    - b usa um parâmetro da mensagem para decidir o que fazer;

GRAPS: Low Coupling (Baixo Acoplamento)

```
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
    }
  }
}
```

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

GRAPS: Low Coupling (Baixo Acoplamento)

- **Solução:** decompor em operações primitivas

```
class Lampada {

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

GRAPS: Low Coupling (Baixo Acoplamento)

- **Acoplamento de dados globais:**
  - Dois ou mais objetos compartilham estruturas de dados globais;
  - É um acoplamento muito ruim pois está escondido;
  - Uma chamada de método pode mudar um valor global e o código não deixa isso aparente;

- **Acoplamento de dados internos:**
  - Um objeto altera os dados locais de um outro objeto;
  - Dados públicos, protegidos...

GRASP: High Coesion (Coesão Alta)

- **Problema:** como manter a complexidade sob controle?
  - Classes que fazem muitas tarefas não relacionadas são:
    - Mais difíceis de entender;
    - Mais dificeis de manter e de reusar;
    - São mais vulneráveis à mudança.

- **Solução:** atribuir uma responsabilidade para que a coesão se mantenha alta.

GRASP: High Coesion (Coesão Alta)

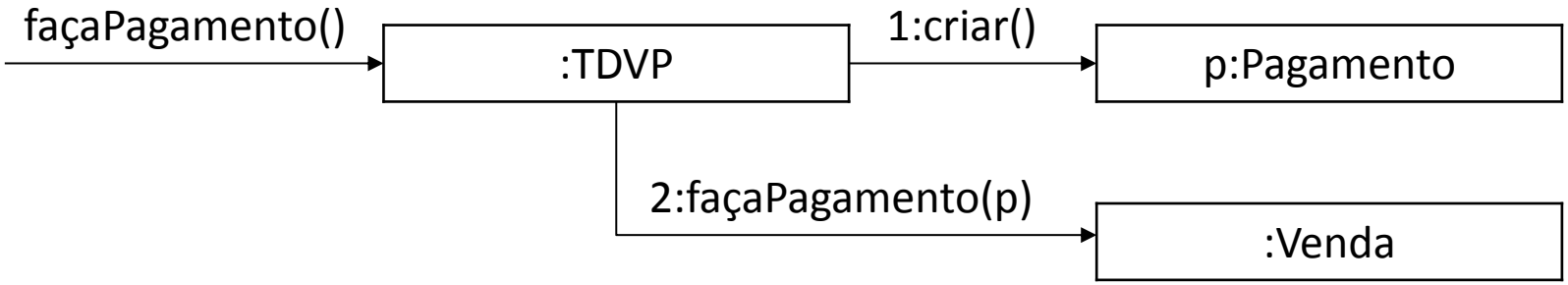
- A coesão é uma medida do quão fortemente **relacionadas** e **focalizadas** são as responsabilidades de uma classe.

- **Exemplo:** uma classe Cachorro é coesa se:
  - Tem operações relacionadas ao Cachorro (morder, correr, comer, latir);
  - E apenas ao Cão (não terá por exemplo listarCaes);

- Alta coesão promove design modular.

GRASP: High Coesion (Coesão Alta)

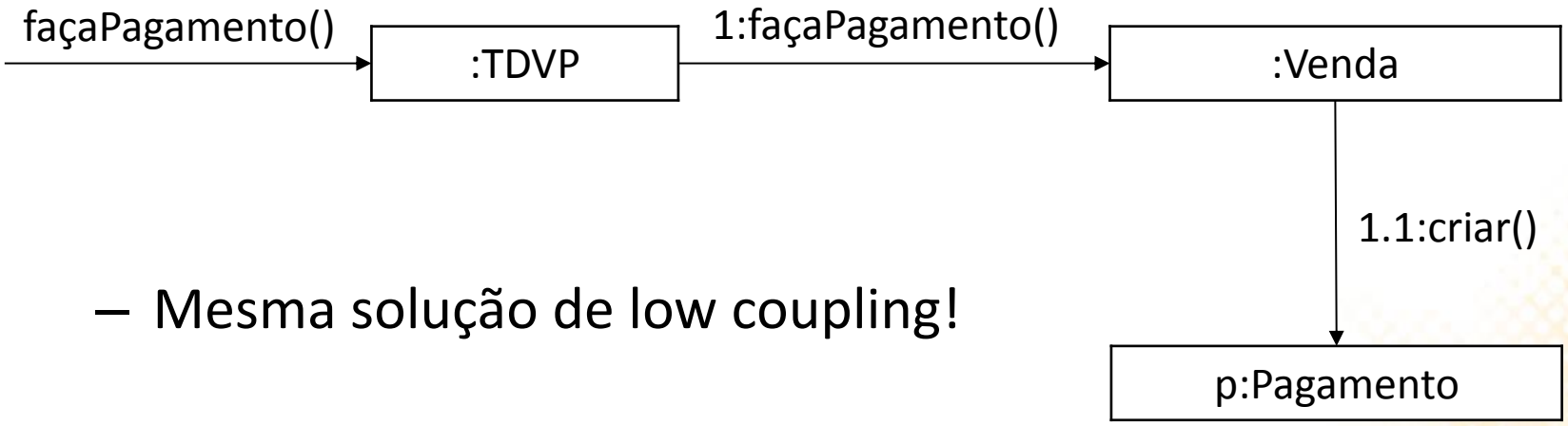
- Que classe é responsável por criar um pagamento e associá-lo a uma venda?
  - De acordo com o Creator, novamente o TDVP deveria criar Pagamento.



- Suponha que isto ocorra várias vezes (outras classes):
  - TDVP acumula métodos não relacionados a ele;
  - Baixa coesão.

GRASP: High Coesion (Coesão Alta)

- Faz mais sentido que o pagamento seja parte de Venda.
  - E não do TDVP, como aparecia na solução anterior;
  - Logo, TDVP delega a responsabilidade a Venda, aumentando a coesão de TDVP;



  - Mesma solução de low coupling!

GRASP: High Coesion (Coesão Alta)

- Em um bom projeto OO, cada classe não deve fazer muito trabalho.
- Como perceber que a coesão de uma classe está baixa?
  - Quando alguns atributos começam a depender de outros.
  - Quando há subgrupos de atributos correlacionados na classe.
- **Consequências:**
  - Melhor claridade e facilidade de compreensão do projeto;
  - Simplificação da manutenção;
  - Frequentemente causa baixo acoplamento;

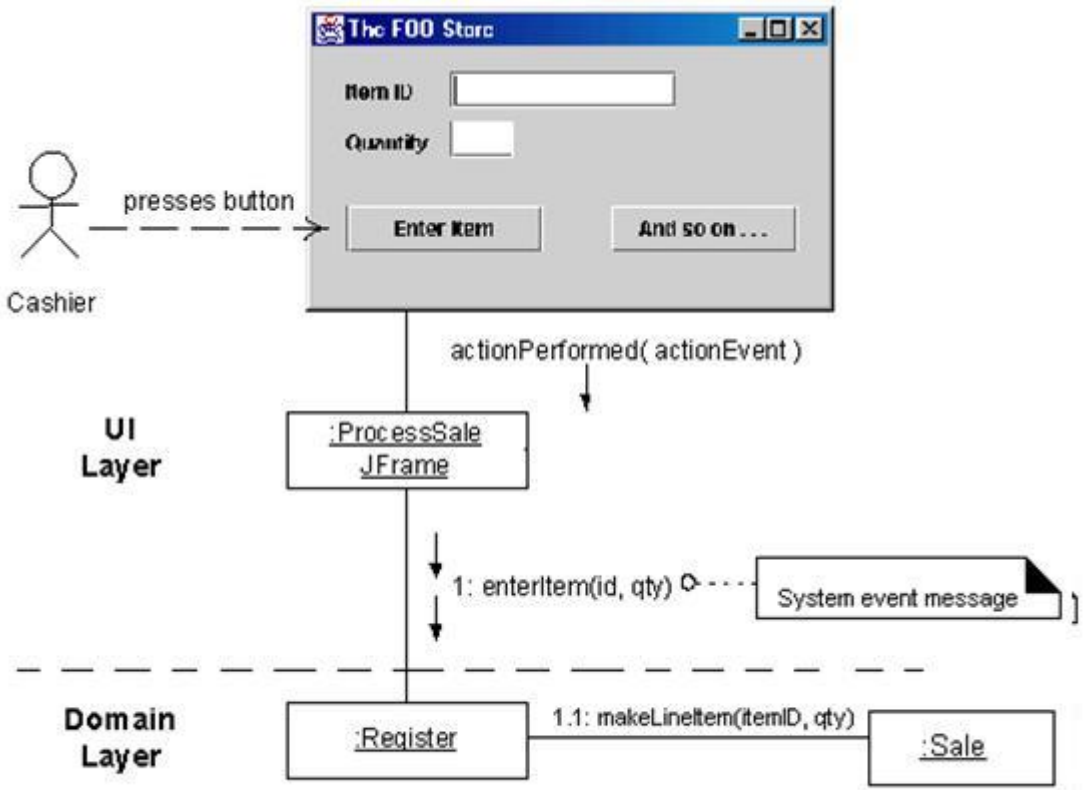
GRASP: Controller

- **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 ou subsistema (façade controller);
  - Uma classe que representa cenário de caso de uso (controlador de caso de uso ou de sessão).

GRASP: Controller

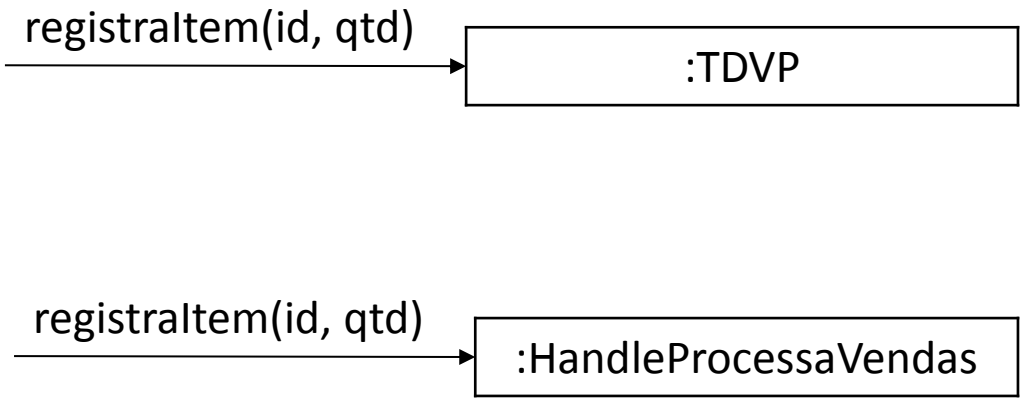
- Que classe deve ser responsável por receber eventos do sistema?



GRASP: Controller

- Normalmente, o controlador não realiza o trabalho, mas delega para outras subpartes do sistema.

- Possiveis escolhas:



GRASP: Controller

- **Benefícios:**
  - Diminui a sensibilidade da camada de apresentação em relação à lógica de domínio;
  - Oportunidade para controlar o estado do caso de uso;

- É necessário balancear a quantidade de controladores:
  - O problema mais comum é ter poucos controladores (controladores sobrecarregados).

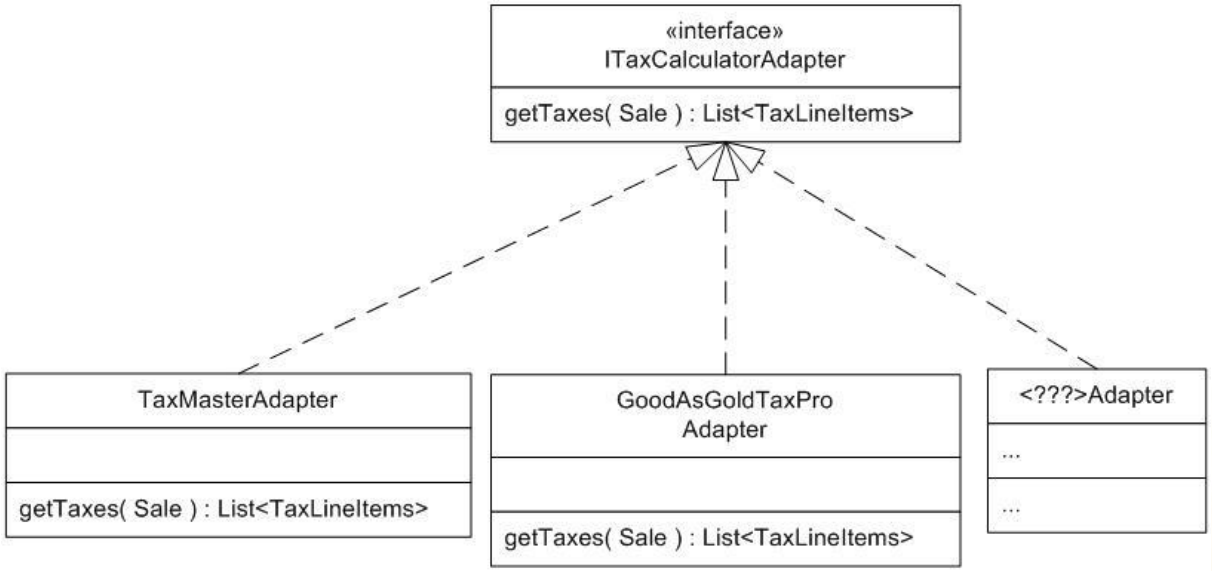
GRASP: Polymorphism

- **Problema:** como tratar alternativas baseadas no tipo? Como criar componentes de software "plugáveis"?

- **Solução:** quando alternativas ou comportamentos relacionados variam com o tipo (classe), atribua as responsabilidades aos tipos usando operações polimórficas.

GRASP: Polymorphism

- No sistema de vendas existem vários **tipos de taxas**, cujo comportamento varia de acordo seu tipo.
  - Através de polimorfismo, podemos criar vários adaptadores semelhantes (mesma interface), que recebem uma venda e se adaptam para diferentes calculos de taxas.



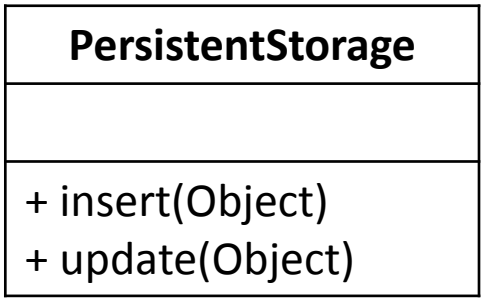
GRASP: Pure Fabrication

- **Problema:** que objeto deve ter a responsabilidade quando você não quer violar "Alta Coesão" e "Baixo Acoplamento", mas as soluções oferecidas pelo "Expert" não são apropriadas?

- **Solução:** atribua um conjuto coeso de responsabilidades a uma classe artificial que não representa um conceito no domínio da aplicação, uma classe fictícia que possibilite alta coesão, baixo acoplamento e o reuso.

GRASP: Pure Fabrication

- Qual classe é responsavel por salvar uma venda no banco de dados?
  - Apesar de Venda ser a candidata lógica para ser a Expert para salvar a si mesma em um banco de dados, isto levaria o projeto a ter baixo acoplamento, alta coesão e baixo reuso.
  - Uma solução seria criar uma classe responsável somente por isto.



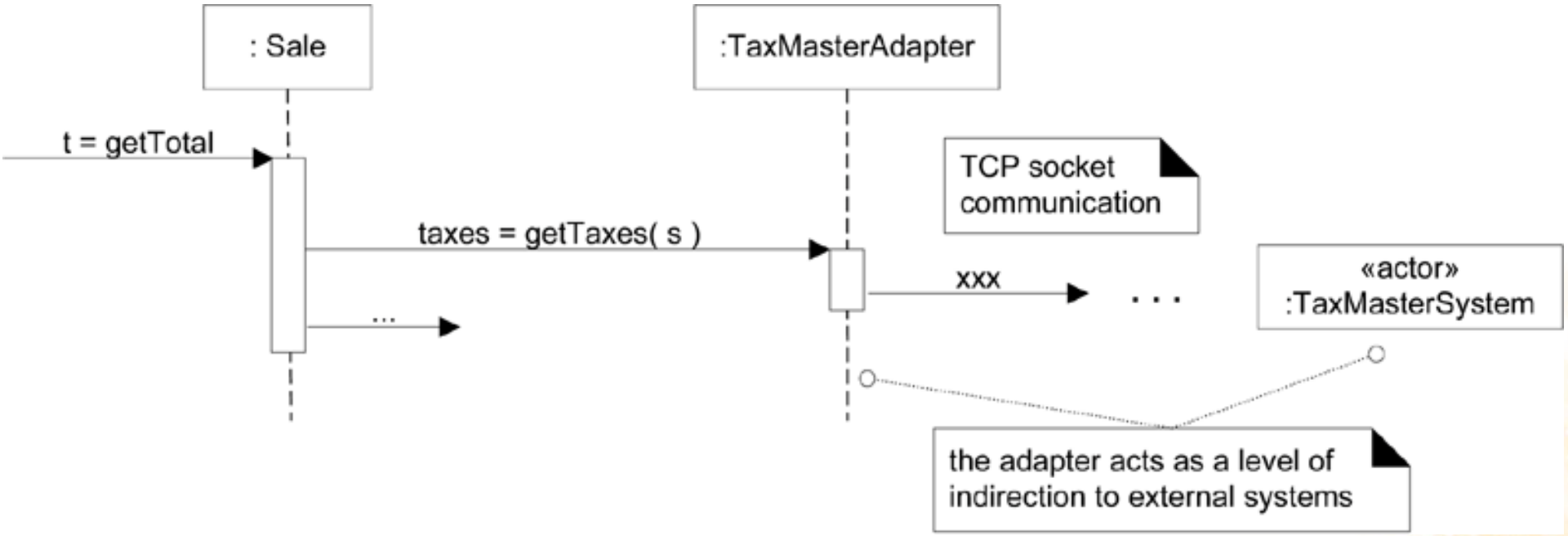
GRASP: Indirection

- **Problema:** Onde colocar uma responsabilidade de modo a evitar o acoplamento direto entre duas ou mais classes? Como desacoplar objetos de modo a possibilitar o baixo acoplamento e manter alta a possibilidade de reuso?

- **Solução:** Atribua a responsabilidade a um objeto intermediário que faça a mediação entre componentes ou serviços de modo que eles não sejam diretamente acoplados.

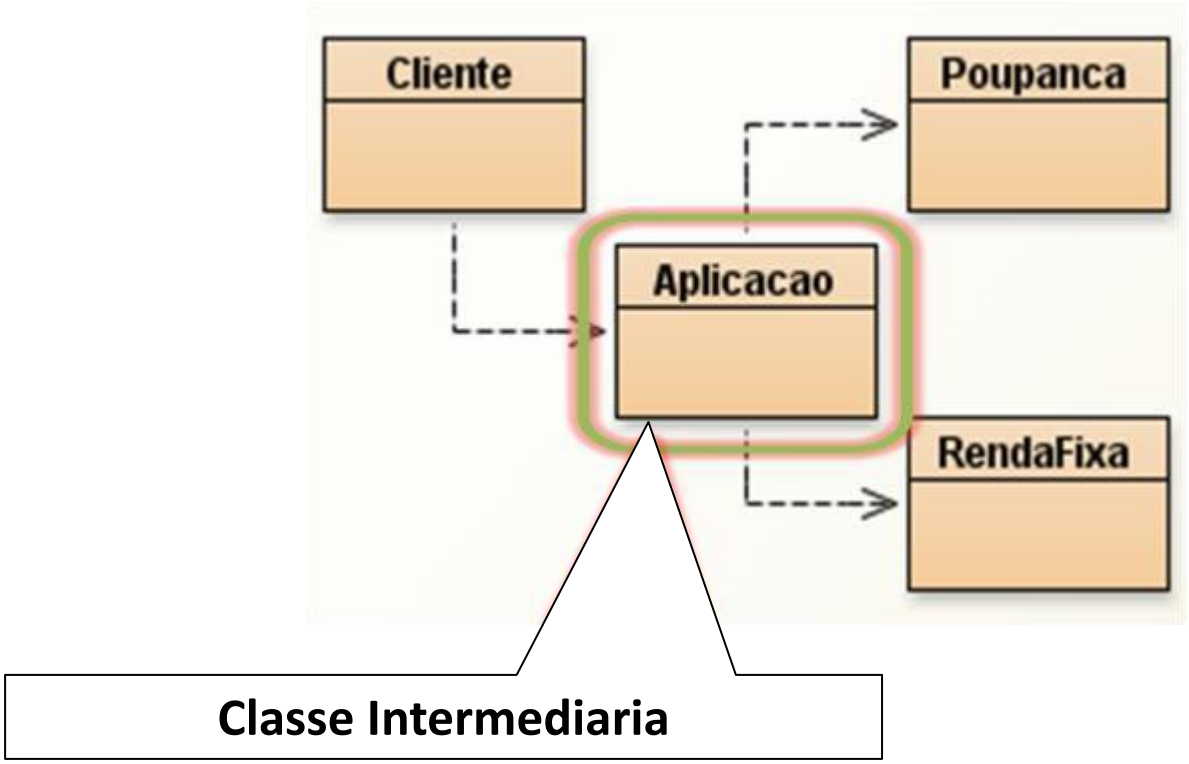
GRASP: Indirection

- Exemplo: Indireção através de um adaptador



GRASP: Indirection

- Exemplo: Indireção em um Sistema Bancário



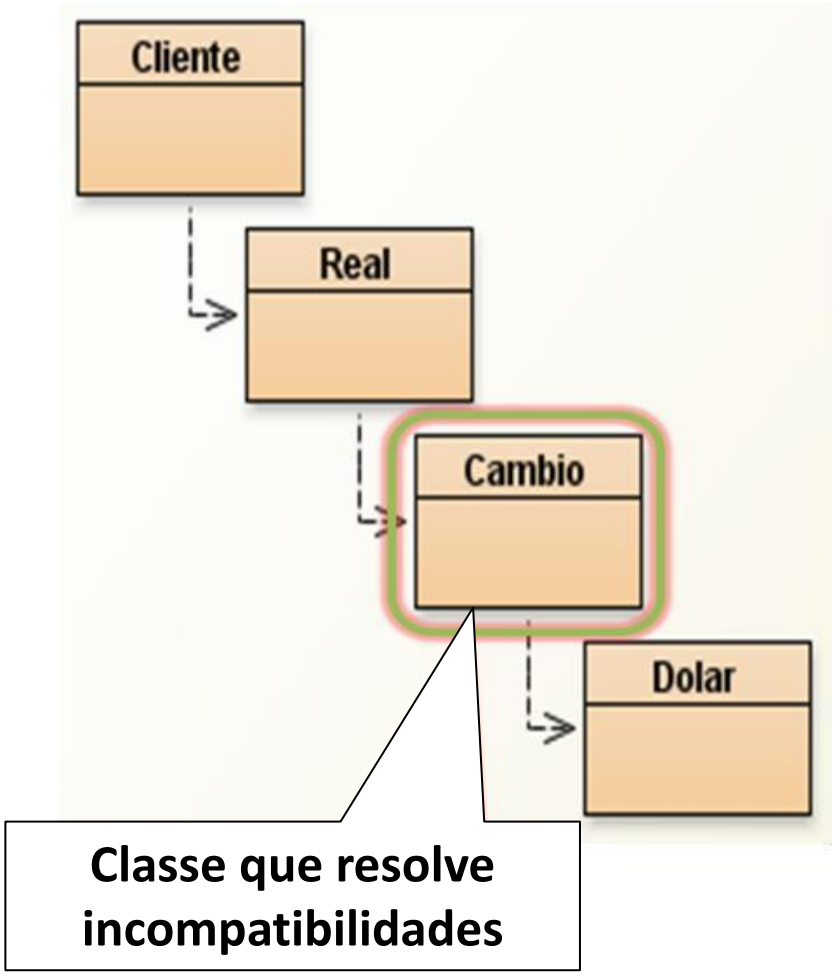
GRASP: Protected Variations

- **Problema:** Como projetar objetos, subsistema e sistemas para que as variações ou instabilidades nesses elementos não tenha um impacto indesejável nos outros elementos?

- **Solução:**
  - Identificar pontos de variação ou instabilidade potenciais e atribuir responsabilidades para criar uma interface estável em volta desses pontos;
  - Encapsulamento, interfaces, polimorfismo, indireção e padrões são motivados por este princípio;
  - Evite enviar mensagens a objetos muito distantes.

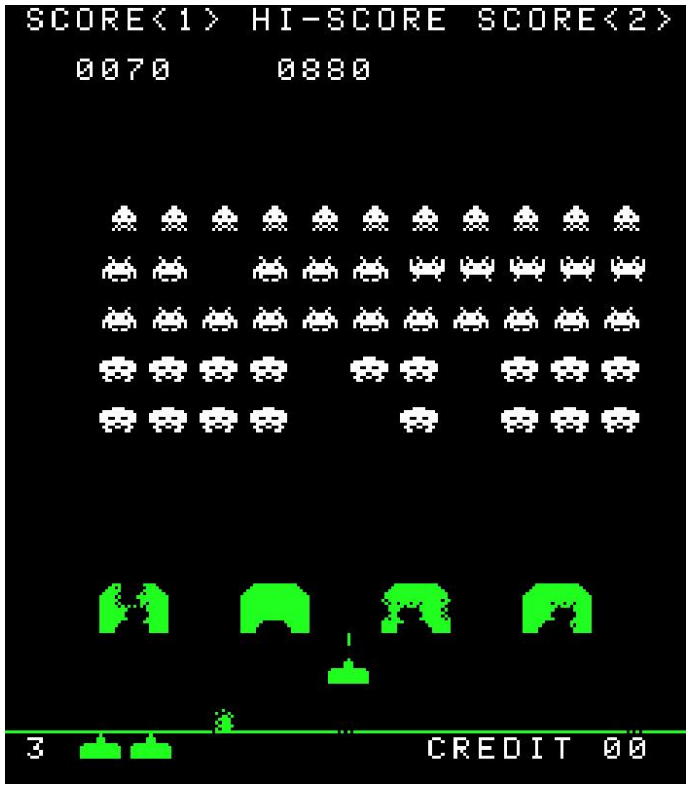
GRASP: Protected Variations

- Exemplo:



GRASP: Exercício

- Análise e projete o jogo Space Invaders aplicando os padrões GRASP na modelagem do sistema.



Exercícios

Lista de Exercícios 03

http://www.inf.puc-rio.br/~elima/poo/