
Residência em Software
Residência em Tecnologia da Informação e Comunicação
Design Patterns
Professor:
Alvaro Degas Coelho
Java
INSTITUIÇÃO EXECUTORA
cepedi
UESC
COORDENADORA
MCTI FUTURO
Softex
APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

O Custo da Manutenção

- Um programa gasta uma certa quantidade de tempo para ser desenvolvido
- E uma quantidade maior para ser mantido
- Constatação antiga
  - Swanson, E. Burton. "The dimensions of maintenance." Proceedings of the 2nd international conference on Software engineering. 1976.

O Custo da Manutenção

- Tipicamente se aceita a seguinte métrica
  - Desenvolvimento: 30%
  - Manutenção: 70%
- Porque?
  - Porque o sistema nunca vai ficar perfeito
    - Bugs are a plague
  - Porque o Mundo muda
    - E as responsabilidades do software também

Palavra chave: MUDANÇA

Como lidar com a Mudança

- Que o Software vai mudar não resta dúvida
- Que as mudanças demandarão mais trabalho que o desenvolvimento, não resta dúvida
- A quesrão é: como se antecipar?
  - Usar um vidente → pouco prático e de resultados questionáveis
  - Construir artefatos que sejam mais facilmente modificáveis

Software Adaptáveis

- Conceber um software com um olho no queijo e o outro no gato
  - Queremos terminar o mais rápido possível
    - Reduzir os custos
  - Queremos ter a capacidade de evoluir o software depois
    - Reduzir MUITO os custos

Padrões e Evolução

- A tarefa de evoluir um software é custosa por dois motivos principais
  - Compreensão do código de outro programador
    - Ou seu mesmo, mas feito em priscas eras!
  - Software Espaguetti

Padrões de Projeto

- Um Padrão é uma estratégia de desenvolvimento que permite facilmente que uma ideia seja compreendida e reproduzida
- "Cada padrão descreve um problema que ocorre frequentemente e então descreve o cerne da solução ao problema de forma a poder reusar a solução milhares de vezes em situações diferentes”
  Prof. Jacques Sauvé

Reuso: o conceito fundamental

- De ideias
  - Não de objetos, nem de códigos
- Micro-arquiteturas
  - Classes, Objetos, Papéis e Colaborações
- Um problema que já existiu, e já foi resolvido
- Visto em outro contexto
  - Adaptação

Mais que padrões de Projeto de Software?

- Sim!
- Padrões exiostem os mais variados
- Analysis Patterns
- Testing Patterns
- Business Patterns
- Pedagogical Patterns
- ...

Mais que padrões de Projeto de Software?

- Sim!
- Padrões exiostem os mais variados
- Analysis Patterns
- Testing Patterns
- Business Patterns
- Pedagogical Patterns
- ...

Meta-Padrão

- Ideia geral de padrões de projeto: aproveitar uma ideia
- Meta-Padrão imprescindível: saber isolar o que é igual e o que não é. Saber adaptar o que é adaptável e o que não é. Saber construir o que faltar.
- “Entre várias situações, isolar o que muda do que é igual”
  Prof. Jacques Sauvé

Padrões de Responsabilidades

- Atribuir responsabilidades
- Um sistema OO é composto de objetos que enviam mensagens uns para os outros
- Uma mensagem é um método executado no contexto de um objeto
- Escolher como distribuir as responsabilidades entre objetos (ou classes) é crucial para um bom projeto

Padrões de Responsabilidades

- Porque?
  - Uma má distribuição → sistemas e componentes frágeis e difíceis de entender, manter, reusar e estender
- Alguns padrões de distribuição de responsabilidades
  - Padrões GRASP (General Responsibility Assignment Software Patterns) – Craig Larman
- Ao mostrar padrões, apresentaremos princípios de um bom projeto OO
- Veremos mais padrões de projeto adiante

Responsabilidades

- Responsabilidades são obrigações de um tipo ou de uma classe
- Obrigações de fazer algo
  - Fazer algo a si mesmo
  - Iniciar ações em outros objetos
  - Controlar ou coordenar atividades em outros objetos

Responsabilidades

- Obrigações de conhecer algo
  - Conhecer dados encapsulados
  - Conhecer objetos relacionados
  - Conhecer coisas que se pode calcular

Responsabilidades

- Exemplos
- Um objeto Venda tem a responsabilidade de criar linha de detalhe (fazer algo)
- Um objeto Venda tem a responsabilidade de saber sua data (conhecer algo)

Responsabilidades

- Granularidade
- Uma responsabilidade pode envolver um único método (ou poucos)
  - Exemplo: Criar uma linha de detalhe de uma Venda
- Uma responsabilidade pode envolver dezenas de classes e métodos
  - Exemplo: Responsabilidade de fornecer acesso a um BD
- Uma responsabilidade não é igual a um método
  - Mas métodos são usados para implementar responsabilidades

O Padrão Expert

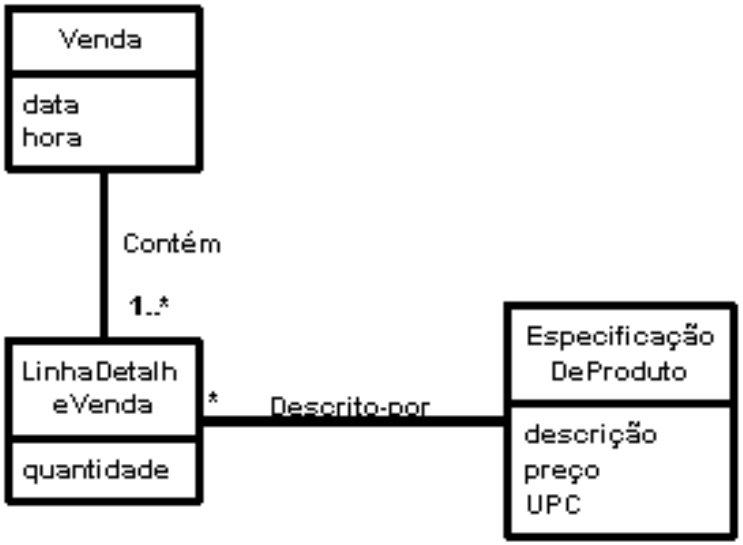
Expert

- Qual o princípio mais fundamental para atribuição de responsabilidades?

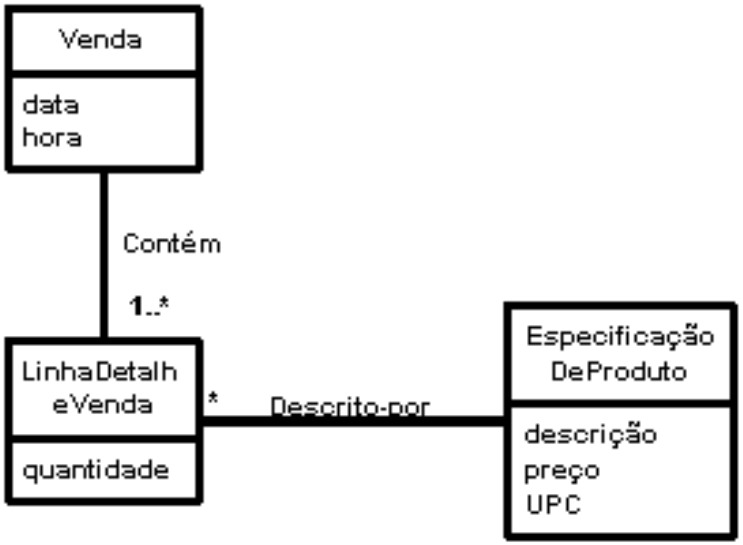
Expert

- Qual o princípio mais fundamental para atribuição de responsabilidades?
- O dono da Informação
- Classe que possui (ou tem acesso) a TODA a informação necessária para preencher a responsabilidade

Exemplo


Venda
data
hora
Contém
1..*
LinhaDetalh
eVenda
quantidade
*
Descrito-por
Especificação
DeProduto
descrição
preço
UPC

Exemplo



A quem caberia saber o total de uma venda?

Hipótese 1: Espec. De Produto

```
number totalVenda(data, hora)
        total = 0
        for each EscecificacaoProduto ep in ListaEspecificacaoProduto()
                for each LinhaDetalheVenda ld in ep.descritoPor()
                        Venda v = ld.getVenda()
                        if v.data==data && v.hora==hora
                                total += ld.quantidade*ep.preço
        return total
```

Hipótese 2: Venda

```
number totalVenda(data, hora)
       total=0
       for each LinhaDetalheVenda ld in this.Contem()
              EspecificacaoProduto ep =
ld.getEspecificacaoProduto()
              total += ld.quant*ep.preço
       return total
```

Qual a Informação Necessária?

Necessitamos ter acesso a todos os objetos do tipo LinhaDetalheVenda

Quem é a *Information Expert*?

Qual a Informação Necessária?

Necessitamos ter acesso a todos os objetos do tipo LinhaDetalheVenda

Quem é a *Information Expert*?

Venda

Not finished yet

- Qual a informação necessária para determinar o subtotal de um item?
  - Nova aplicação do Expert
- Precisamos dos atributos LinhaDeVenda.quantidade e EspecificacaoDeProduto.preço
- Quem é o *Information Expert*?

Not finished yet

- Qual a informação necessária para determinar o subtotal de um item?
  - Nova aplicação do Expert
- Precisamos dos atributos LinhaDeVenda.quantidade e EspecificacaoDeProduto.preço
- Quem é o *Information Expert*?

- **LinhaDeVenda**

Prosseguindo

- Quem é o *InformationExpert* para saber o preço de um produto?

Prosseguindo

- Quem é o *InformationExpert* para saber o preço de um produto?

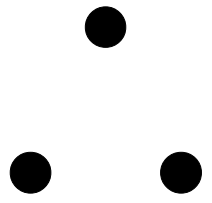
- **EspecificaçãoDeProduto**

Discussão

- Expert → padrão mais utilizado
- Fato: a informaçãonormalmente se espalha entre vários objetos
  - Experts parciais
- Colaborações: mensagens (métodos get)
- Resultado não é “natural”
  - No mundo real, uma “Venda” não calcula nada!

Porque o Expert

- Altíssimo encapsulamento



Baixo Acoplamento

Como Minimizar dependências e maximizar o reuso?

- Acoplamento: quanto um artefato computacional se liga a outro
  - Está conectado
  - Possui conhecimento
  - Depende
- Artefato para nós = Classe
- Fraco Acoplamento → uma classe não é dependente de outra(s)

Porque acoplamento forte é ruim?

Porque acoplamento forte é ruim?

- Mudanças numa classe A relacionada à classe B força mudanças na classe B
- Uma classe possui semântica menos evidente
  - Difícil de entender seu papel
- Reuso mais complicado
  - Depende de outras classes

Porque acoplamento forte é ruim?

- Mudanças numa classe A relacionada à classe B força mudanças na classe B
- Uma classe possui semântica menos evidente
  - Difícil de entender seu papel
- Reuso mais complicado
  - Depende de outras classes

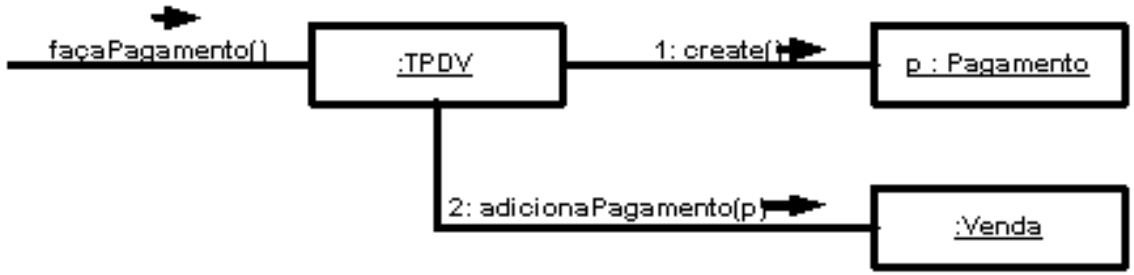
Atribuir responsabilidades visando minimizar o acoplamento

Ainda no exemplo do TDV

- Supondo que tenhamos uma classe **TPDV** (Terminal de Ponto de Venda)
- Criar a classe **Pagamento**
- Segundo o padrão Creator, quem deveria fazer isso?

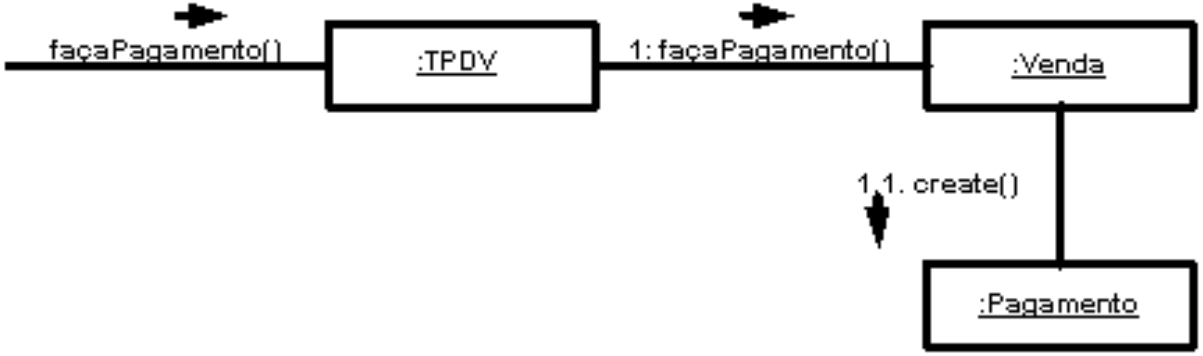
Duas alternativas

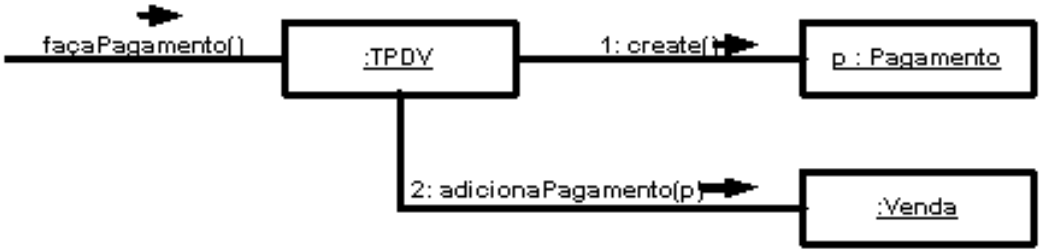
- Alternativa 1: posto que, no mundo real, um pagamento é registrado no TPDV, atribuir esta responsabilidade à classe TPDV

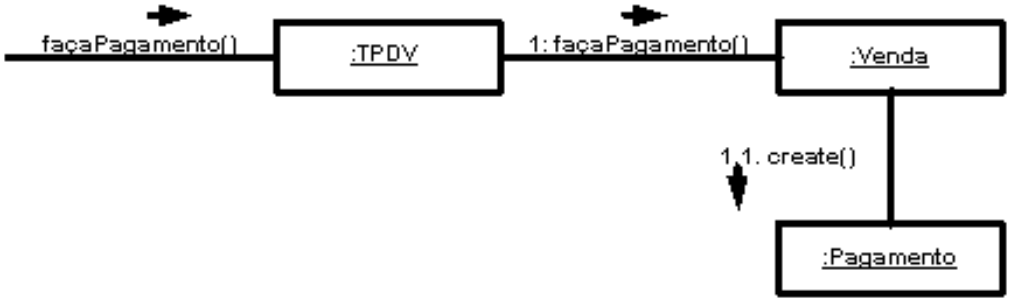


Duas alternativas

- Alternativa 2: Como o pagamento é associado à Venda, criá-lo com Venda




façaPagamento()
:TPDV
1: create()
p : Pagamento
2: adicionaPagamento(p)
:Venda


façaPagamento()
:TPDV
1: façaPagamento()
:Venda
1.1. create()
:Pagamento

Minimizar acoplamento é um dos princípios de ouro do projeto OO

- Acoplamento se manifesta de várias formas:
  - X tem um atributo que referencia uma instância de Y
  - X tem um método que referencia uma instância de Y
- Pode ser parâmetro, variável local, objeto retornado pelo método
  - X é uma subclasse direta ou indireta de Y
  - X implementa a interface Y

Minimizar acoplamento é um dos princípios de ouro do projeto OO

- A herança é um tipo de acoplamento particularmente forte
  - Aprofundaremos o assunto
- Não se deve minimizar acoplamento criando alguns poucos objetos monstruosos (God classes)
  - Vide POG

Alta Coesão

Gerenciamento da Complexidade

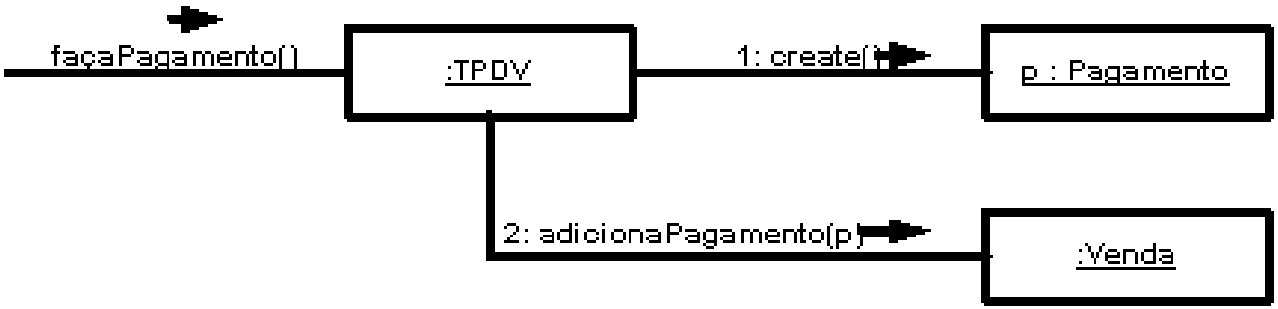
- A coesão mede quão relacionadas ou focadas estão as responsabilidades da classe
  - Também chamada de "coesão funcional"
- A classe “se encaixa” nas responsabilidades

Baixa Coesão

- Classe faz muitas coisas não relacionadas e leva aos seguintes problemas:
  - Difícil de entender
  - Difícil de reusar
  - Difícil de manter
  - "Delicada": constantemente sendo afetada por outras mudanças
- Uma classe com baixa coesão assumiu responsabilidades que pertencem (pertenceriam?) a outras classes

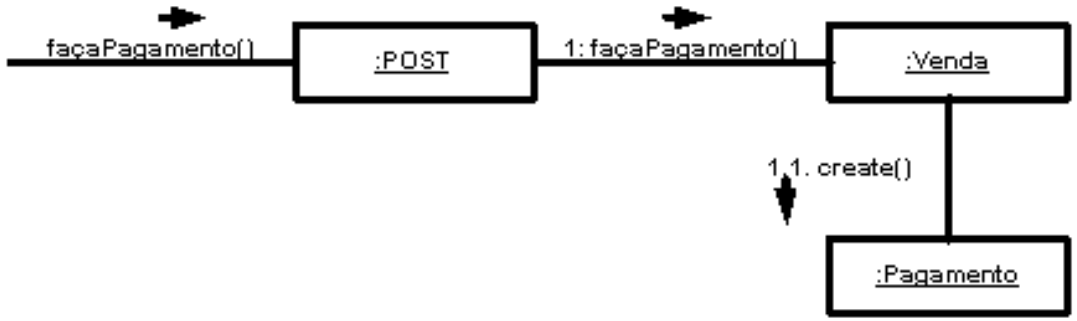
Exemplo

- Mesmo exemplo usado para Low Coupling
- Na primeira alternativa, TPDV assumiu uma responsabilidade de efetuar um pagamento (método `façaPagamento()`)



Exemplo

- Suponha que o mesmo ocorra com várias outras operações de sistema
  - CreditoCelular, CancelaCompra, DevolveProduto, etc.
- TPDV vai acumular um monte de métodos não muito focados
- Solução: delegar



Coesão: outra regra de ouro

- Manter um olho na coesão durante todo o projeto
- Refletir sobre a coesão antes de optar pela criação de qualquer classe
- E, principalmente, de qualquer método!

Padrão Creator

Quem deve criar instâncias?

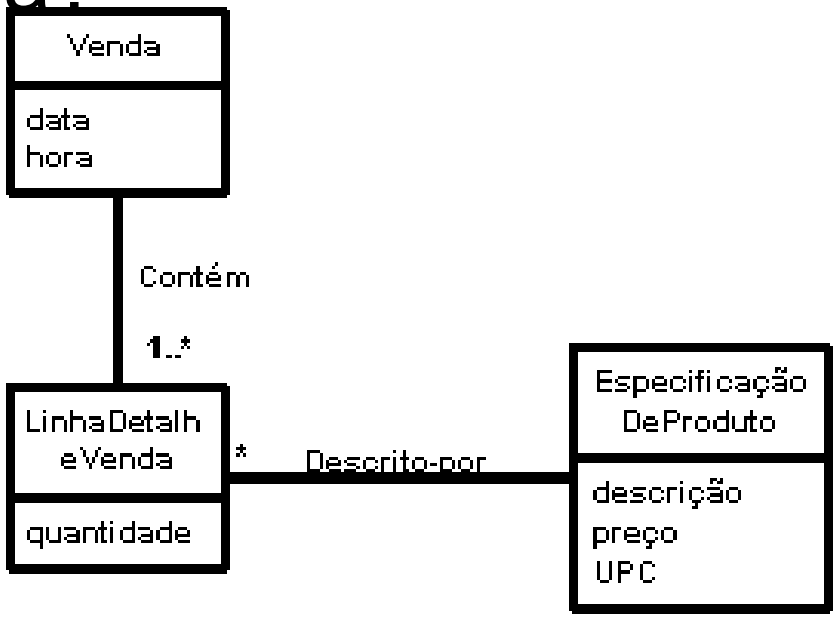
- Atribuir à classe B a responsabilidade de criar instância da classe A se
  - B agrega objetos da classe A, ou
  - B contém objetos da classe A, ou
  - B registra instâncias da classe A, ou
  - B usa muitos objetos da classe A, ou
  - B possui os dados usados para inicializar A

B é “creator” de A

- Criador de A significa que um objeto do tipo B é responsável por azer surgir objetos do tipo A
- Caso mais de um objeto seja candidato → escolher o que (potencialmente) agregue ou contenha mais objetos A

Exemplo

- Ainda usando o TDV
- Quem deve ser “creator” de LinhaDetalheVenda?



Reflexão

- Criador estará conectado ao objeto criado
  - Não aumenta o acoplamento
- O mesmo se um criador possui os dados para inicialização
- Exemplo
  - Quando formos criar uma instância de **Pagamento**
    - Entre outras informações, possui o total da Venda