Padrão para atribuir responsabilidades: Expert

Introdução

- 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
  - Uma má distribuição leva a sistemas e componentes frágeis e difíceis de entender, manter, reusar e estender
- Mostraremos alguns *padrões* de distribuição de responsabilidades
  - Padrões GRASP (General Responsibility Assignment Software Patterns)
  - São padrões de 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
- Obrigações de conhecer algo
  - Conhecer dados encapsulados
  - Conhecer objetos relacionados
  - Conhecer coisas que se pode calcular

- 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)
- 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

Problema

- Qual é o princípio mais fundamental para atribuir responsabilidades?

Solução

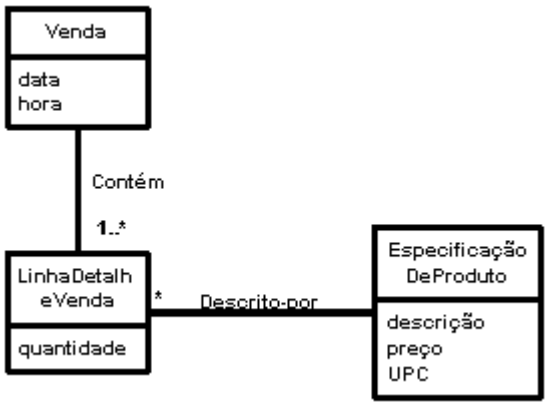
- Atribuir uma responsabilidade ao expert de informação - a classe que possui a informação necessária para preencher a responsabilidade

Exemplo

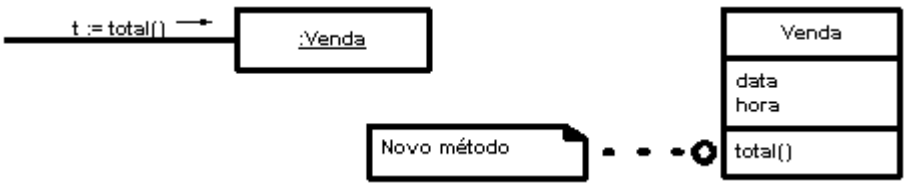
- Num sistema de TPDV, alguma classe precisa saber o total de uma venda
- Podemos dizer isso sobre a forma de uma responsabilidade:

- Quem deveria ser responsável pelo conhecimento do total de uma venda?

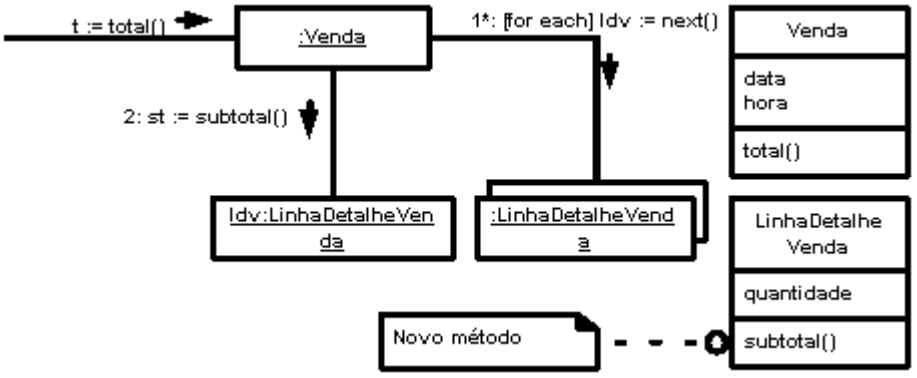
- Pelo padrão Expert, escolhemos a classe que possui a informação necessária para determinar o total
- Considere uma parte de um modelo conceitual:



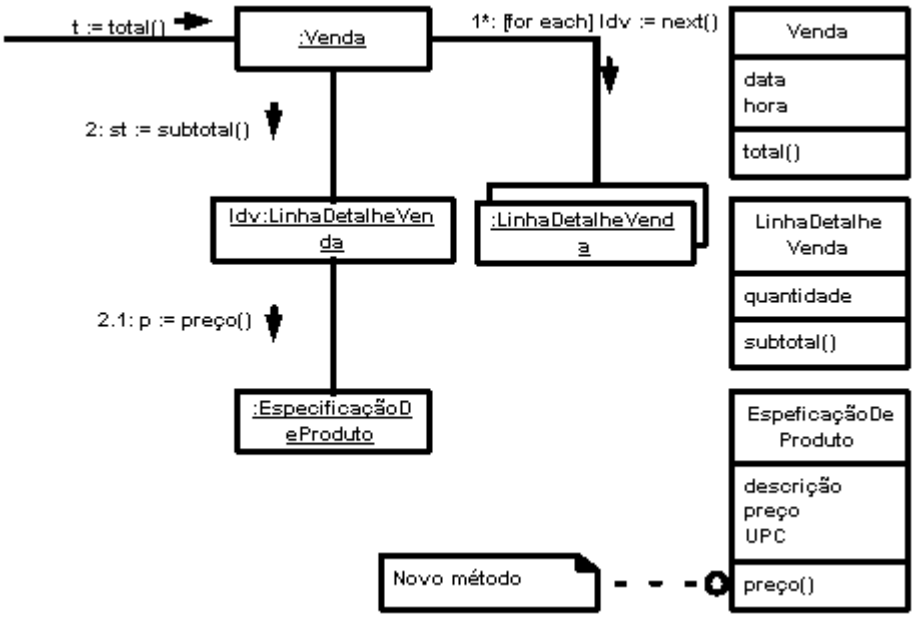
- Qual é a informação necessária?
  - Precisamos conhecer (ter acesso a) todos os LinhaDetalheVenda
  - Qual é information expert?
  - É a classe Venda
- Podemos agora fazer parte do diagrama de colaboração e do diagrama de classes



- Ainda não terminamos. Qual informação é necessária para determinar o subtotal para um item (uma linha de detalhe)?
  - Precisamos de LinhaDeVenda.quantidade e de EspecificaçãoDeProduto.preço
  - Quem é o information expert?
  - É a classe LinhaDeVenda
- Chegamos aos seguintes diagramas:



- Qual é o information expert para saber o preço de um produto?
  - É EspecificaçãoDeProduto
- Chegamos aos seguintes diagramas:



- Claro que tudo isso pode ser decidido sem ter que fazer diagramas UML

Discussão

- É o padrão mais usado de todos para atribuir responsabilidades
- A informação necessária freqüentemente está espalhada em vários objetos
  - Portanto, tem muitos experts parciais
  - Exemplo: determinar o total de uma venda requer a colaboração de 3 objetos, em 3 classes diferentes

- Mensagens são usadas para estabelecer as colaborações
- O resultado final é diferente do mundo real
  - No mundo real, uma venda não calcula seu próprio total
  - Isso seria feito por uma pessoa (se não houvesse software)
  - Mas no mundo de software, não queremos atribuir essa responsabilidade ao Caixa ou ao Terminal Ponto-De-Venda!
  - No mundo de software, coisas inertes (ou até conceitos como uma Venda) podem ter responsabilidades: Tudo está vivo!

Conseqüências

- A encapsulação é mantida, já que objetos usam sua própria informação para cumprir suas responsabilidades
  - Leva a fraco acoplamento entre objetos e sistemas mais robustos e fáceis de manter
- Leva a alta coesão, já que os objetos fazem tudo que é relacionado à sua própria informação

Também conhecido como

- "Colocar as responsabilidades com os dados"
- "Quem sabe, faz"
- "Animação"
- "Eu mesmo faço"
- "Colocar os serviços junto aos atributos que eles manipulam"

Exercício para casa

- No código abaixo, onde o padrão Expert está sendo furado?
  - Melhore o programa, aplicando o padrão Expert

```
public class Fita {
```

```
  public static final int NORMAL = 0;
  public static final int LANÇAMENTO = 1;
  public static final int INFANTIL = 2;

  private String título;
  private int códigoDePreço;

  public Fita(String título, int códigoDePreço) {
    this.título = título;
    this.códigoDePreço = códigoDePreço;
  }

  public String getTítulo() {
    return título;
  }

  public int getCódigoDePreço() {
    return códigoDePreço;
  }

  public void setCódigoDePreço(int códigoDePreço) {
    this.códigoDePreço = códigoDePreço;
  }
}

public class Aluguel {
  private Fita fita;
  private int diasAlugada;

  public Aluguel(Fita fita, int diasAlugada) {
    this.fita = fita;
    this.diasAlugada = diasAlugada;
  }

  public Fita getFita() {
    return fita;
  }

  public int getDiasAlugada() {
    return diasAlugada;
  }
}

public class Cliente {
  private String nome;
  private Collection fitasAlugadas = new Vector();

  public Cliente(String nome) {
```

```
  this.nome = nome;
}

public String getNome() {
  return nome;
}

public void adicionaAluguel(Aluguel aluguel) {
  fitasAlugadas.add(aluguel);
}

public String extrato() {
  final String fimDeLinha = System.getProperty("line.separ
  double valorTotal = 0.0;
  int pontosDeAlugadorFrequente = 0;
  Iterator alugueis = fitasAlugadas.iterator();
  String resultado = "Registro de Alugueis de " + getNome(
  while(alugueis.hasNext()) {
    double valorCorrente = 0.0;
    Aluguel cada = (Aluguel)alugueis.next();

    // determina valores para cada linha
    switch(cada.getFita().getCódigoDePreço()) {
    case Fita.NORMAL:
      valorCorrente += 2;
      if(cada.getDiasAlugada() > 2) {
        valorCorrente += (cada.getDiasAlugada() - 2) * 1.5
      }
      break;
    case Fita.LANÇAMENTO:
      valorCorrente += cada.getDiasAlugada() * 3;
      break;
    case Fita.INFANTIL:
      valorCorrente += 1.5;
      if(cada.getDiasAlugada() > 3) {
        valorCorrente += (cada.getDiasAlugada() - 3) * 1.5
      }
      break;
    } //switch
    // trata de pontos de alugador frequente
    pontosDeAlugadorFrequente++;
    // adiciona bonus para aluguel de um lançamento por pe
    if(cada.getFita().getCódigoDePreço() == Fita.LANÇAMENT
       cada.getDiasAlugada() > 1) {
       pontosDeAlugadorFrequente++;
    }

    // mostra valores para este aluguel
```

```
      resultado += "\t" + cada.getFita().getTítulo() + "\t"
      valorTotal += valorCorrente;
    } // while
    // adiciona rodapé
    resultado += "Valor total devido: " + valorTotal + fimDe
    resultado += "Voce acumulou " + pontosDeAlugadorFrequent
              " pontos de alugador frequente";
    return resultado;
  }
}
```

programa