Programação Orientada por Objectos

Design de Aplicações: Coesão e Acoplamento

Prof. Rui César das Neves, Prof. José Cordeiro

Departamento de Sistemas e Informática
Escola Superior de Tecnologia de Setúbal – Instituto Politécnico de Setúbal
2014/2015

Sumário

- Motivação
  - Manutenção
  - Flexibilidade
- Coesão
  - O que é ?
  - Maximizar a Coesão
  - Coesão de Classes
  - Classes não coesas por:
    - Problema de Mistura de Instâncias (mixed-instance)
    - Problema de Mistura de Domínios (mixed-domain)
    - Problema de Mistura de Papéis (mixed-role)
  - Coesão de Métodos
- Acoplamento
  - O que é ?
  - Como Minimizar?
  - Problemas de Acoplamento:
    - Acoplamento de Identidades
    - Acoplamento de Representação
    - Acoplamento de Subclasses
    - Acoplamento de Herança

Motivação

- Um programa pode ser usado durante décadas:
  - Um Programa não é um romance que se escreve uma vez e nunca mais é reescrito.
  - Um Programa requere manutenção ao longo da sua existência:
    - É adaptado a novas exigências
    - É expandido com novas funcionalidades
    - É corrigido e expurgado de erros
    - É transportado para outras e novas plataformas
  - A manutenção é realizada por diferentes pessoas
    - A legibilidade do código é fundamental.
- **O lema de um programa é:**
  - **"Change or die":**
  - Um programa difícil de manter acaba no lixo!

Motivação

- Qualidade do Código:
  - Dois importante conceitos para melhorar a qualidade e portanto facilitar a manutenção do código:
    - Coesão
    - Acoplamento
  - Flexibilidade através da simplicidade:
    - A arquitetura do sistema deve ser sempre o mais simples possível.
  - Mais flexibilidade com menos complexidade:
    - **maximizar a coesão e**
    - **minimizar o acoplamento**

Coesão – O que é?

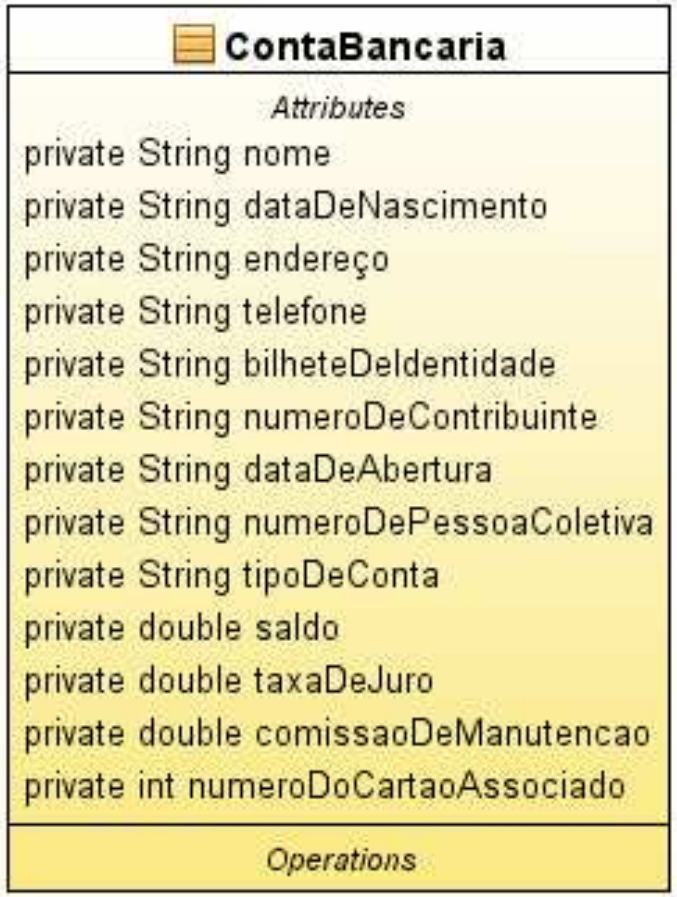
- **Coesão** - é uma medida da quantidade e diversidade dos "tópicos“ (ou tarefas) pelos quais uma entidade é responsável.
  - A coesão aplica-se a Classes e Métodos.
  - Quanto menor a diversidade de assuntos abordados por uma Classe/Método, maior a sua coesão.
  - Se uma entidade é responsável por uma única tarefa, dizemos que ela é coesa (ou que tem elevada coesão).
  - Uma entidade que englobe diferentes tarefas, temas, tópicos é uma entidade com baixa coesão (pouco coesa).
  - O nosso **objetivo** é **Maximizar a Coesão** das Classes e Métodos

Coesão – Maximizar para quê?

- Classes e Métodos devem ser o mais coesos possível:
  - Uma Classe deve representar apenas uma abstração.
  - Um Método deve executar uma única tarefa.
- Classes e Métodos pouco coesos dificultam a manutenção:
  - Dificultam a reutilização do código
  - Dificultam a adaptação a novos requisito e a expansão de funcionalidades
  - Dificultam a correção de problemas
  - Dificultam o transporte para outras plataformas
- Quando uma Classe modela mais de uma abstração …
  - É porque estamos a assumir implicitamente que as várias abstrações representadas na classe têm sempre uma relação de 1 para 1 entre si.
    - Quando a relação entre as várias abstrações representadas pela classe se revela não ser de 1 para 1, o mais provável é virmos a necessitar de posteriores especializações para as diferentes dimensões impostas pelas diferentes abstrações representadas na classe

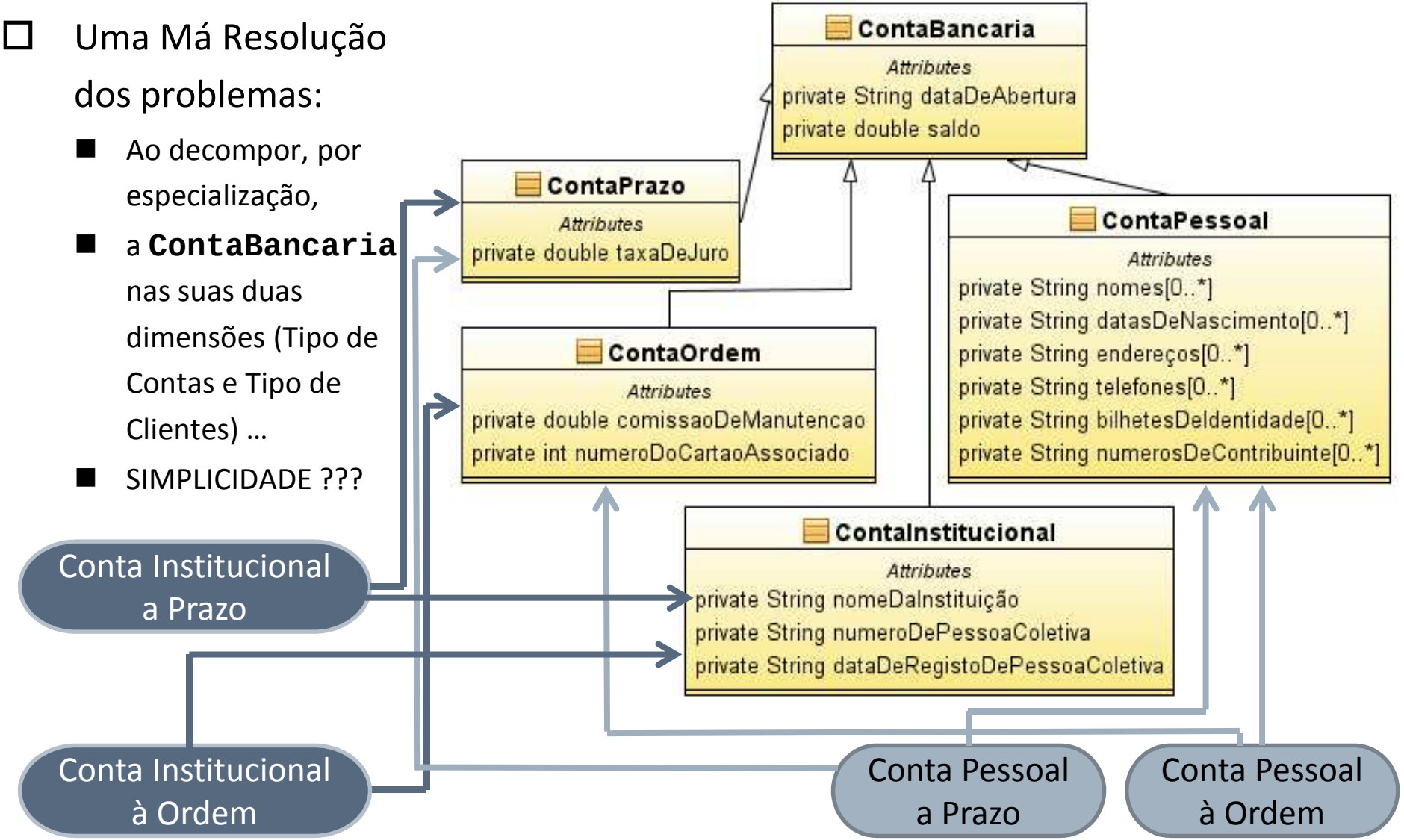
Coesão – de Classes

- Um Mau Exemplo:
  - Uma classe **`ContaBancaria`** que inclui informações sobre o utilizador (nome, endereço, etc.)
- Erro:
  - Mistura os clientes com as contas (engloba duas entidades distintas numa mesma classe)
- Problemas (só alguns):
  - Não permite contas conjuntas,
  - Um cliente com várias contas terá os seus dados repetidos em cada conta (o que gerará problemas de atualização e consistência),
  - Diferentes tipos de clientes (individuais e institucionais, por exemplo) exigem diferentes caracterizações.
- Uma Má Solução:
  - vamos tentar resolver os problemas sem ir à sua raiz, isto é sem "corrigir o erro".



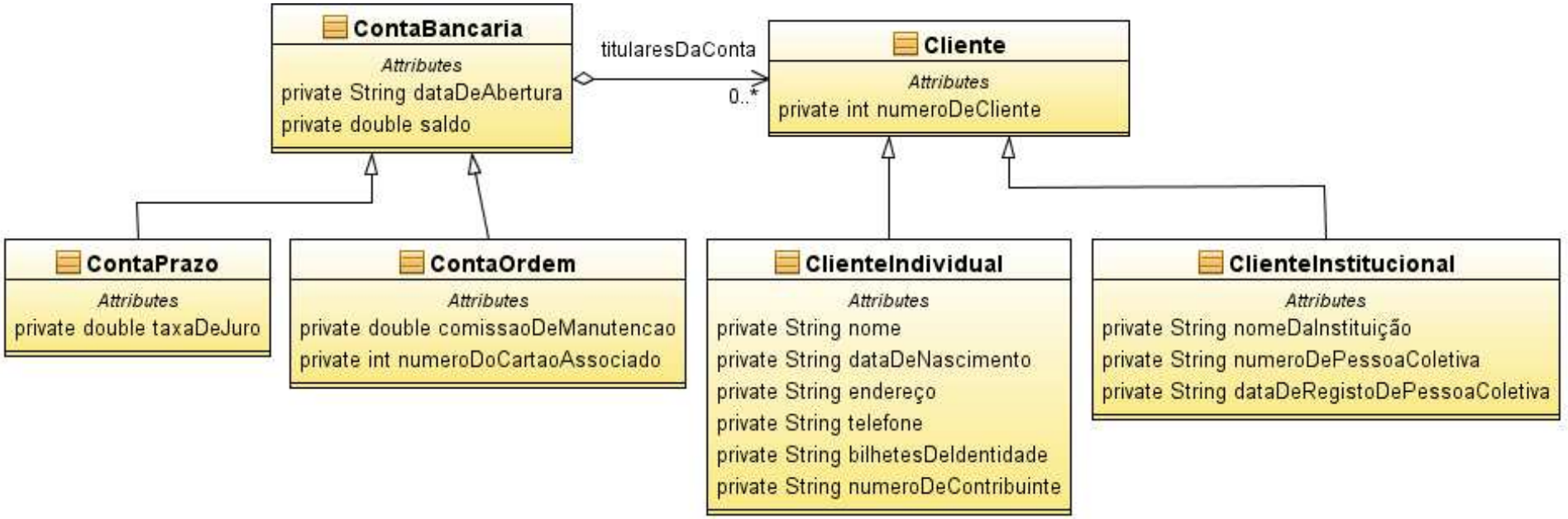
Coesão de Classes - Maximização

- Uma Má Resolução dos problemas:
  - Ao decompor, por especialização,
  - a **`ContaBancaria`** nas suas duas dimensões (Tipo de Contas e Tipo de Clientes) …
  - SIMPLICIDADE ???



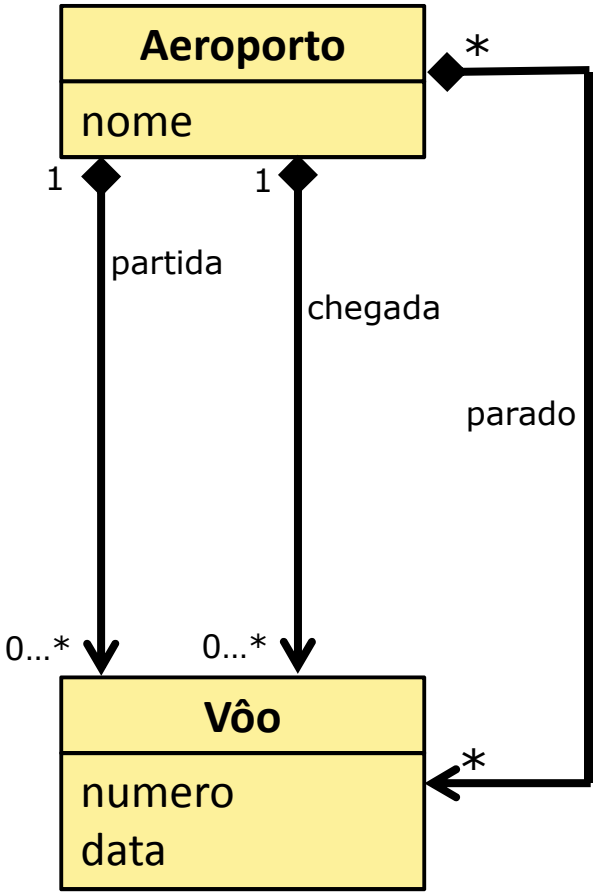
Coesão de Classes - Maximização

- Uma Boa Solução:
  - Atacar “o mal pela raiz”
  - Corrigir o “Erro”
  - Separar as entidades distintas
  - Maximizar a coesão!



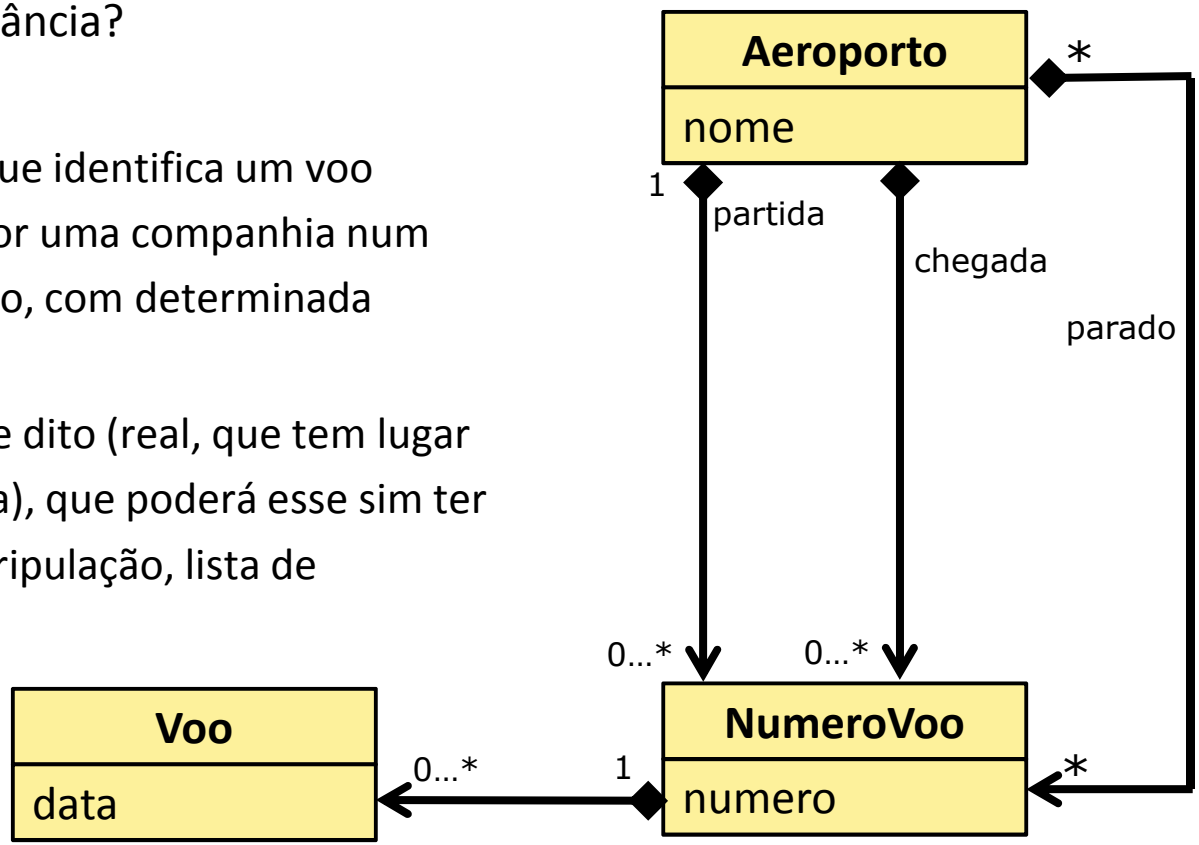
Coesão de Classes - Redundância

- Modelação das Relações entre Voos e Respetivos Aeroportos:
  - Um Aeroporto tem voos
    - a partir,
    - a chegar e eventualmente
    - voos parados.
    - Ou dito de outra forma, um voo parte de um aeroporto, chega a outro ou está parado.
- Problemas (redundância):
  - se o voo tem lugar todos os dias, estamos a repetir os objetos que representam o percurso (partida/chegada) todos os dias.
  - Isto acontece porque estamos a representar um percurso e uma instância do voo na mesma classe.
- Como eliminar a redundância?



Coesão de Classes - Redundância

- Modelação das Relações entre Voos e Respetivos Aeroportos:
  - Como eliminar a redundância?
  - Separando
    - O Numero Do Voo (que identifica um voo abstrato, realizado por uma companhia num determinado percurso, com determinada regularidade)
    - Do Voo propriamente dito (real, que tem lugar num determinado dia), que poderá esse sim ter informação sobre a tripulação, lista de passageiros etc.

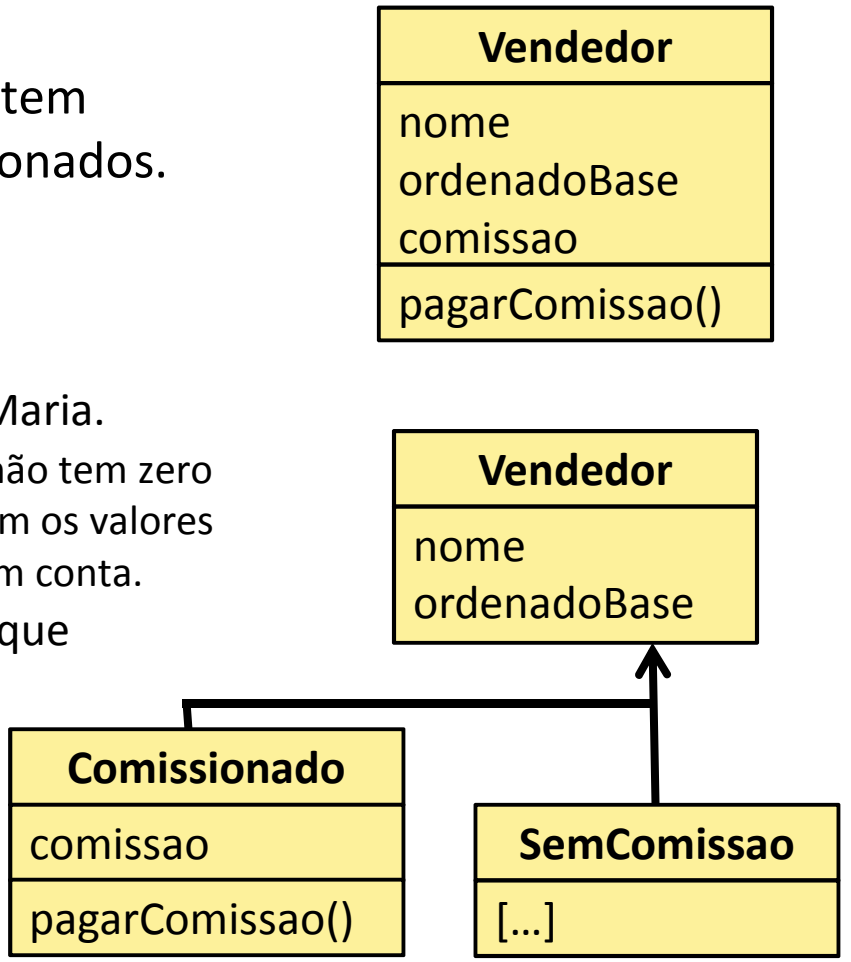


Coesão de Classes - Redundância

- Redundância:
  - Contras:
    - Dificulta a atualização dos dados.
    - Permite a existência de inconsistências nos dados.
    - Desperdiça espaço.
  - Prós:
    - Mas, cuidado, por outro lado, redundância pode melhorar o desempenho do sistema.
  - Podemos eliminar a redundância aumentando a coesão das entidades envolvidas.
  - Normalmente **aumentando a coesão** das entidades envolvidas **diminuímos a redundância**

Coesão de Classes – Tipos de Problemas

- **Mistura de instâncias (mixed-instance cohesion)** - Uma classe possui atributos que não são definidos para alguns dos objetos da classe.
- Problema: Um departamento de vendas tem vendedores comissionados e não comissionados.
  - Uma classe Vendedor possui o método **`pagarComissao()`**.
  - José é comissionado e Maria não.
  - Poderíamos atribuir zero à comissão de Maria.
    - Mas isso não seria correto, já que Maria não tem zero de comissão e os métodos que calculassem os valores a ambos os vendedores teriam de o ter em conta.
  - Poderíamos criar um atributo **`boolean`** que indicaria se o vendedor é comissionado
    - mas isso seria um péssimo design.
- Solução: especializar a classe nos dois tipos de vendedores



Coesão de Classes – Tipos de Problemas

- **Mistura de Domínios (mixed-domain cohesion)** - Uma classe contém um elemento que a sobrecarrega com uma classe extrínseca de um domínio diferente.
  - Uma classe A é extrínseca a B se A puder ser totalmente definida sem conhecimento de B.

- Problema:
  - Uma classe **`Real`** possui um método **`arcoTangente()`**.
  - Mas **`arcoTangente()`** não é uma característica de **`Real`** mas sim de **`Angulo`**.

- Quando uma classe de um dado domínio é especificada, não deve ser incluída em classes de domínios inferiores.
- É o que define **reusabilidade.**

Coesão de Classes – Tipos de Problemas

- **Mistura de Papéis (mixed role cohesion)** - Uma classe contém um ou mais elementos que fazem parte do domínio, mas não fazem parte da abstração representada por essa classe.
- Problema:
  - Uma classe **`Pessoa`** possui um método **`quantidadeCarros()`**.
  - Na realidade **`Carro`** não caracteriza **`Pessoa`**.
  - Como reutilizar **`Pessoa`** se não houver **`Carro`**s na nova aplicação?
  - E se continuássemos com essa filosofia de design? Onde pararíamos? **`quantidadeBarcos()`**, **`quantidadeGatos()`**, etc.
  - Todos estes atributos sobrecarregam a classe **`Pessoa`**.
  - É muito frequente cair neste tipo de problema, de falta de coesão, pois:
    - 1- É fácil (e lógico?) perguntar a uma Pessoa quantos carros tem:
      - `jose.quantidadeCarros();`
    - 2- Muitas abordagens de design indicam que **`quantidadeCarros()`** é um método de **`Pessoa`**;
    - 3- Na vida real, se quisermos saber quantos carros tem o José perguntamos-lhe.

Coesão de Classes - Resumindo

- Maximizar a Coesão …
  - Ao evitar, ou resolver, os problemas de coesão anteriormente mencionados
    - Mistura de instância (mixed instance cohesion)
    - Mistura de papéis (mixed role cohesion)
    - Mistura de domínios (mixed domain cohesion)
  - Estamos a modelar corretamente as nossas classes …
  - e a facilitar a sua manutenção e reutilização.
- Legibilidade do código
  - Obtemos classes com responsabilidades bem definidas;

Coesão - de Métodos

- **Coesão de Métodos** – Dizemos que um método é coeso quando executa apenas uma funcionalidade.

```
public void joga() {

    // Imprime as boas vindas
    System.out.println();
    System.out.println("Welcome to The World of Zuul!");
    System.out.println("Zuul é um incrivelmente aborrecido.");
    System.out.println("Escreva 'help' se precisar de ajuda.");
    System.out.println();
    System.out.println(currentRoom.getLongDescription());

    // Entra no loop principal. Le e executa comandos
    //repetidamente até acabar o jogo.
    boolean finished = false;
    while (! finished) {
        Command command = parser.getCommand();
        finished = processCommand(command);
    }

    System.out.println("Thank you for playing. Good bye.");
}
```

- Um método pouco coeso imprime as boas vindas e processa o jogo

```
public void joga() {
    imprimeBoasVindas();
    boolean finished = false;
    while (! finished) {
        Command command = parser.getCommand();
        finished = processCommand(command);
    }
    System.out.println("Thank you for playing. Good bye.");
}

Public void imprimeBoasVindas(){
    System.out.println();
    System.out.println("Welcome to The World of Zuul!");
    System.out.println("Zuul é um incrivelmente aborrecido.");
    System.out.println("Escreva 'help' se precisar de ajuda.");
    System.out.println();
    System.out.println(currentRoom.getLongDescription());
}
```

- Dois métodos mais coesos:
  - um só processa o jogo e
  - invoca outro que só imprime as boas vindas

Coesão de Métodos - Resumindo

- Maximizar a Coesão …
  - Ao atribuir a cada método uma só tarefa estamos a
    - Melhorar a legibilidade do código
    - Facilitar a sua reutilização
  - Estamos a modelar corretamente o comportamento dos nossos métodos …
  - e a facilitar a sua manutenção e reutilização.
- Legibilidade do código
  - Obtemos métodos com nomes e responsabilidades bem definidas;

Acoplamento O que é?

- **Acoplamento** - é uma medida do grau de dependência de umas partes (módulos/pacotes/classes) do sistema em relação a outras.
  - Refere-se às ligações existentes entre diferentes entidades do programa
  - Uma medida do grau de conhecimento que uma classe tem do meio envolvente.
  - Uma medida da Interdependência entre módulos (pacotes/classes).
  - Se duas entidades dependem muito uma da outra dizemos que estão estreitamente acopladas
  - O Objetivo é Minimizar o Acoplamento

Acoplamento – Minimizar para quê?

- Um sistema deve apresentar o mínimo de acoplamento:
  - As interdependências de um sistema devem ser minimizadas
- Sistemas com muitas interdependências são de difícil manutenção:
  - Aumentam a probabilidade da propagação de erros.
    - Os erros numa Classe ou Pacote passam a poder ter impacto nos restantes módulos dos quais estes dependem ou que deles são dependentes.
  - Dificultam o teste e debug de módulos isolados.
    - Obrigam-nos a testar em simultâneo vários módulos.
  - Dificultam a compreensão do código
    - Obrigam-nos a olhar para os vários módulos interdependentes a fim de compreender o funcionamento do programa.
  - Dificultam a expansão de capacidades
  - Dificultam a reutilização
    - Obrigam-nos a "levar atrás" vários módulos para reutilizar apenas um.

Acoplamento – Tipos de Problemas

- **Acoplamento de Identidades** - Um objeto de uma classe guarda uma referência para um objeto de outra classe.
  - O objeto conhece a identidade do outro e, portanto, está-lhe acoplado , está dele dependente.
  - Alterações no objeto referenciado podem ter impacto no que guarda a referência
- Como minimizar:
  - Reduz-se o acoplamento de identidades eliminando associações no diagrama de classes e implementando associações de forma unidirecional .

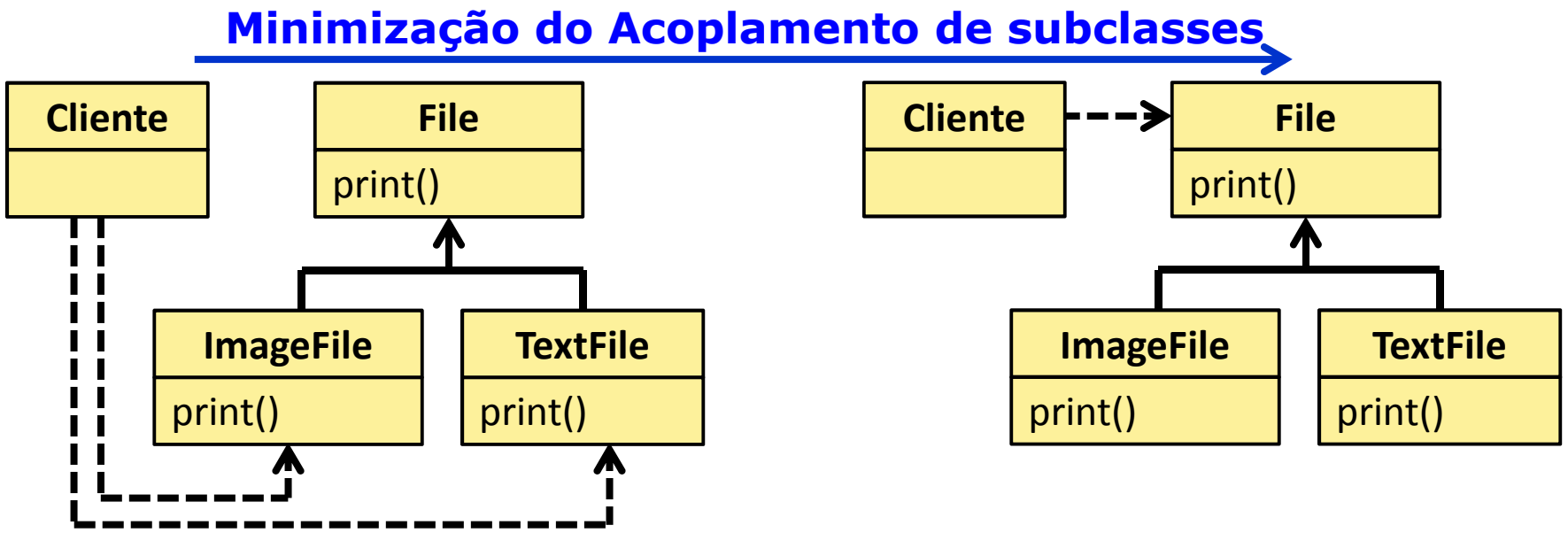
Acoplamento – Tipos de Problemas

- **Acoplamento de Representação** - Um objeto referencia outro, i.e., acede diretamente aos atributos de outro objeto da mesma classe.
  - Quanto mais específico o acesso mais forte o acoplamento
    - Lê (acede mas não pode alterar) atributos de um objeto
    - Lê e altera atributos de um objeto
- Como minimizar:
  - O encapsulamento através do uso dos inspetores e modificadores minimiza o acoplamento de representação (os métodos escondem a implementação do atributo).
  - Devemos sempre programar com a interface da classe e não com os seus atributos.
- Os atributos "private" impedem e acoplamento de representação?
  - Experimente no netBeans!

Acoplamento – Tipos de Problemas

- **Acoplamento de subclasses** - Um objeto acede a uma subclasse usando a interface da subclasse em vez de usar a interface da superclasse.

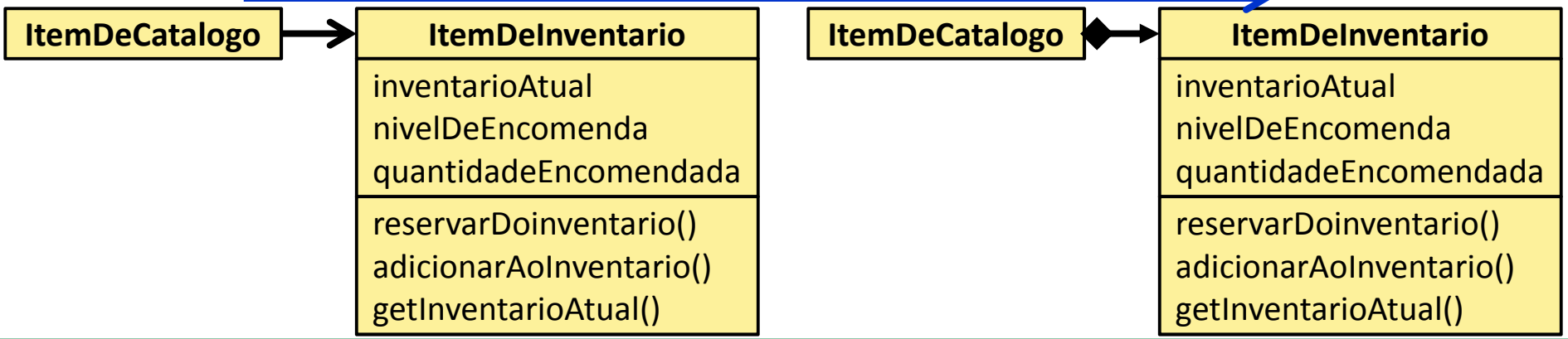
- Como minimizar:
  - Um cliente deve usar uma referência para o tipo mais genérico possível.



Acoplamento – Tipos de Problemas

- **Acoplamento por Herança** - Uma subclasse é acoplada à sua superclasse através de acoplamento de herança.
  - É provavelmente o tipo de acoplamento mais forte (dependências mais fortes).
  - Manifesta-se fortemente em tempo de compilação, mas também em tempo de execução:
    - O que uma classe herda em tempo de compilação não pode ser descartado em tempo de execução.
    - Neste aspeto difere de agregação em que um objeto se pode desfazer dos seus agregados em tempo de execução.
  - Como minimizar: Usar herança apenas quando, de facto, uma entidade "É" (também?) outra.

Minimização do Acoplamento por Herança



Refactoring – O Que é?

- **Refactoring -** Processo de alteração da estrutura do código sem alterar o comportamento externo.
- As classes tendem a crescer:
  - Durante a fase de manutenção de um programa é normal que as classes vão crescendo com a inclusão de novos atributos e de novos métodos.
  - Com alguma assiduidade torna-se necessário "refatorizar" essas classes por forma a manter a coesão (ou maximizá-la) e o acoplamento (ou minimizá-lo) conseguidos inicialmente
- O código começa a ser duplicado:
  - Ao verificar que determinado processamento (sequência de código) é necessário e está a ser "repetido" em vários métodos devemos criar um novo método privado que englobe todo ou parte desse processamento.
- Aumentar o reaproveitamento do código:
  - Já durante a manutenção frequentemente decidimos mover funcionalidades de uma dada classe para uma nova superclasse ou subclasse.

Refactoring – e Teste!

- Ao refatorizar o código:
  - Separe a refatorização da introdução de quaisquer outras alterações!
  - Antes de mais proceda à refatorização sem introduzir quaisquer novas funcionalidades.
  - Teste o programa antes e logo após a refatorização para se assegurar de que não foram introduzidos erros.

Resumindo

- Acoplamento
  - Uma medida da interdependência dos módulos.
  - Pretende-se minimizar o acoplamento para:
    - Diminuir a probabilidade da propagação de erros.
    - Facilitar o teste e debug de módulos isolados.
    - Tornar o código mais compreensível
    - Facilitar a manutenção
    - Simplificar a reutilização de módulos
- Problemas de Acoplamento:
  - 1. Acoplamento de Identidades - 2. Acoplamento de Representação – 3. Acoplamento de Subclasses – 4. Acoplamento de Herança
- Coesão e Acoplamento são medidas da qualidade de um programa.
  - O objetivo da modelação de um sistema orientado a objetos é garantir uma elevada coesão e um baixo acoplamento, facilitando alterações nas suas estruturas de dados e de funcionalidades internas.

Leitura Complementar

- Page-Jones, "*Fundamentals of Object-Oriented Design in UML*".
- Charles Richter, "*Designing Flexible Object-Oriented Systems with UML*" . *Capítulo 4: Flexibility Guidelines for Class Diagrams. Macmillan Technical Publishing, 1999.*
- Capítulo 5
  - Páginas 141 a 169
  - Página 189