Instituto de Computação

Princípio de POO (continuação)

Viviane Torres da Silva
viviane.silva@ic.uff.br

http://www.ic.uff.br/~viviane.silva/es1

Agenda

- Encapsulamento
- Projeto Estruturado
- Congeneridade
- Domínios
- Grau de dependência
- Coesão
- Contratos
- Interface de classes
- Perigos detectados em POO

Contratos I/XI

- Projeto por Contratos (*design by contract*) é uma abordagem utilizada para especificar as variações de espaço-estado (domínio de valores válidos para os atributos) possíveis para um método
- Um contrato é descrito pela combinação da invariante de classe com as pré e pós-condições de um método
- A pré-condições deve ser verdadeira antes da execução do método
- A pós-condição deve ser verdadeira após a execução do método

Contratos II/XI

- Antes da execução de cada método, a seguinte condição deve ser avaliada:
  - (invariantes de classe) e (pré-condições do método)
- Se a avaliação é falsa:
  - O contrato não está sendo cumprido pelo contratante
  - A execução não deverá ocorrer
  - O sistema entrará em uma condição de tratamento de exceções

- Após a execução de cada método, a seguinte condição deve ser avaliada:
  - (invariantes de classe) e (pós-condições do método)
- Se a avaliação é falsa:
  - O contrato não está sendo cumprido pelo contratado
  - O método deverá ser reimplementado
  - O sistema entrará em uma condição de tratamento de erro

Cláusulas Contratuais:

- Se o contratante (cliente) consegue garantir as pré-condições, então o contratado (fornecedor) deve garantir as pós-condições
- Se o contratante não conseguir garantir as pré-condições, então o contrato será cancelado para aquela chamada, podendo a operação não ser executada e não garantir a pós-condição

Contratos V/XI

- Exemplo de criação de contrato para a classe Pilha e para o método pop() utilizando linguagem natural
- A classe Pilha tem os atributos itens (coleção dos elementos da pilha) e limite (tamanho máximo da pilha)

Invariante: A pilha tem no máximo o número de itens definido pelo limite

Pré-condição do método pop(): A pilha não está vazia

Pós-condição do método pop(): O número de itens foi decrescido em uma unidade

- Como seria o contrato para o push() ??

- O uso de linguagem natural para descrever cláusulas contratuais leva a ambigüidade
- É possível utilizar formalismos como OCL para permite a descrição de forma não ambígua

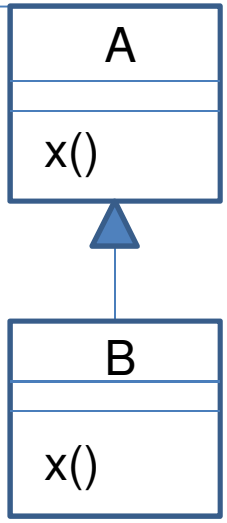
```
context Pilha
inv: not (self.itens->size > self.limite)

context Pilha::pop()
pre: self.itens-> notEmpty
post: self.itens-> size = (self.itens@pre->size - 1)
```

Contratos VII/XI

- Em contratos usados pela sociedade, só pode haver alteração se ambas as partes concordarem
  - Caso o contratante resolva modificar o contrato, as novas condições devem ser iguais às antigas ou mais flexíveis
  - Caso o contratado resolva modificar o contrato, as novas condições devem ser iguais às antigas ou mais restritivas

- A mudança contratual em projeto orientado a objetos ocorre quando é criada uma subclasse com métodos polimórficos

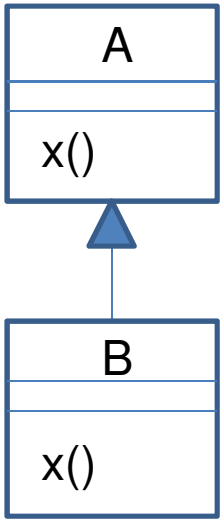
- As invariantes da subclasses devem ser iguais ou mais restritivas que da superclasse
  - Nem tudo que vale para A vale para B



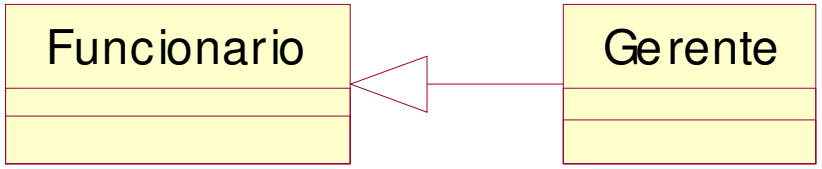
- **Contravariação**: As pré-condições dos métodos polimórficos da subclasse devem ser iguais ou menos restritivas que as pré-condições dos métodos da superclasse
  - As restrições para executar A.x() são maiores que para executar B.x(), isto é, nem todas as execuções de B.x() valem para A.x()

Contratos IX/XI

- **Covariação**: As pós-condições dos métodos polimórficos da subclasse devem ser iguais ou mais restritivas que as pós-condições dos métodos da superclasse
  - O resultado da execução de A.x() é menos restritivo que o resultado da execução de B.x(), isto é, o resultado de B.x() é mais especializado



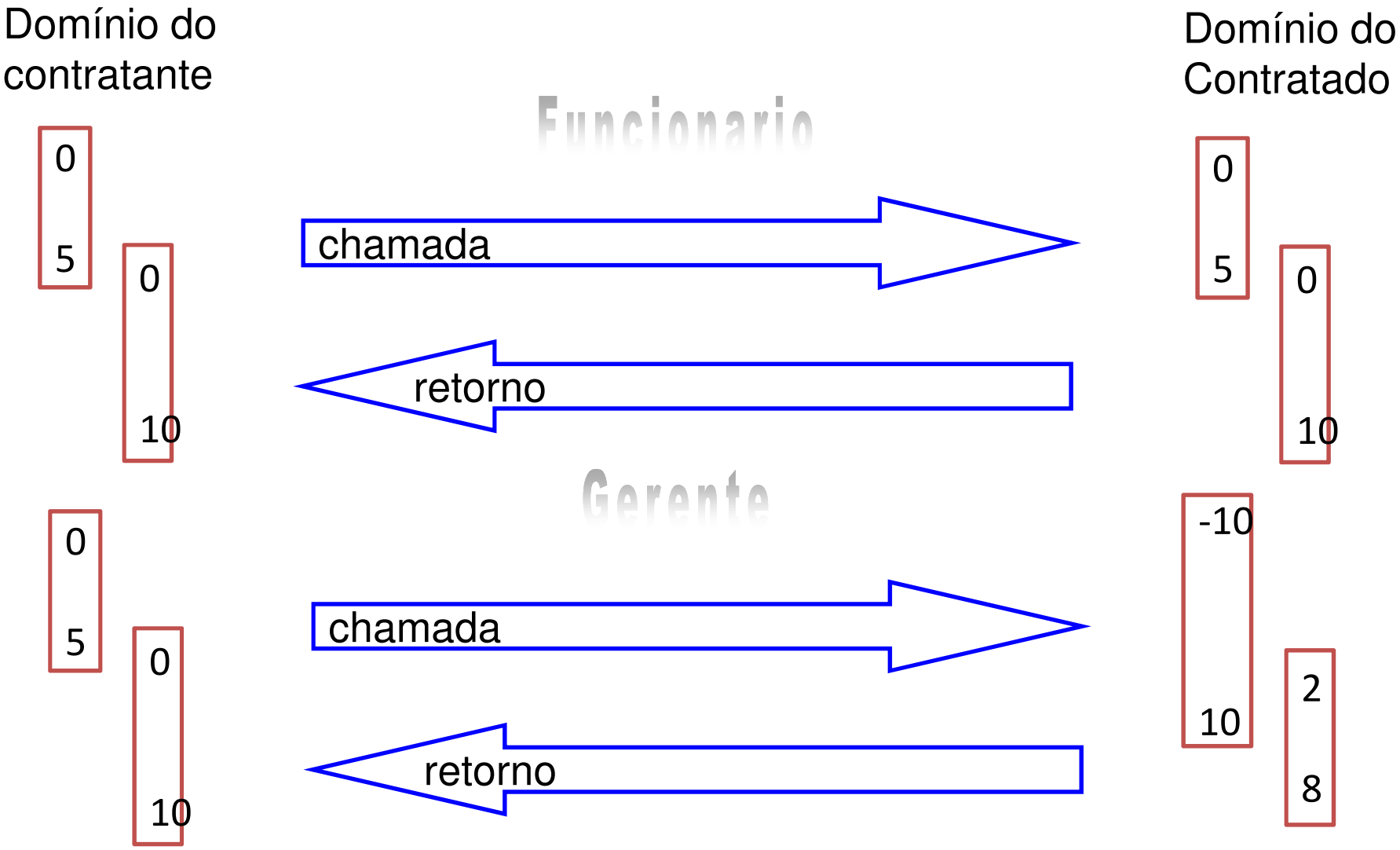
Contratos X/XI




```
context Funcionario
inv: self.anosEstudo >= 9

context Funcionario::calculaBonus(avaliacao:int):int
pre: (avaliacao >= 0) and (avaliacao <= 5)
post: (result >= 0) and (result <= 10)
```

```
context Gerente
inv: self.anosEstudo >= 12

context Gerente::calculaBonus(avaliacao:int):int
pre: (avaliacao >= -10) and (avaliacao <= 10)
post: (result >= 2) and (result <= 8)
```

Contratos XI/XI

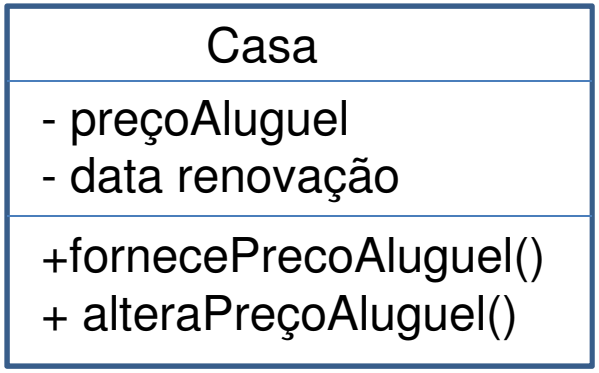


Interface de Classe I/VII

- A interface (pública) de uma classe define todos os métodos que serão visíveis às demais classe do sistema
- É através dessa interface que o comportamento da classe será definido
- Juntamente com o comportamento, as variações de estado da classe também são dependentes da interface
- As interfaces podem ser classificadas tanto em relação aos estados internos quanto em relação ao comportamento

Interface de Classe II/VII

- Em relação aos estados, existem quatro classificações de interface:
- **Interface com estados ilegais (com vazamento)**: exibe métodos privados como públicos
  - Exemplo: alteraPreçoAluguel() deveria ser um método privado chamado de dentro de fornecePrecoAluguel()



- **Interface com estados incompletos**: não possibilita alcançar todos os estados válidos do espaço-estado
  - Exemplo: não ser possível alterar a data de renovação

Interface de Classe III/VII

- **Interface com estados inapropriados**: permite acesso a estados que não fazem parte da abstração do objeto
  - Exemplo: Visualizar o enésimo elemento de uma Pilha
  - Exemplo: Colocar um elemento em qualquer ordem

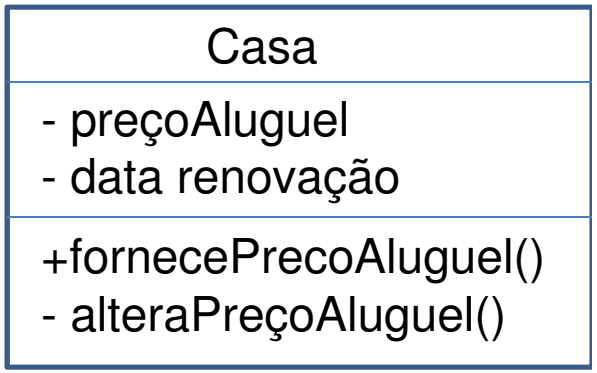
- **Interface com estados ideais**: um objeto consegue atingir qualquer estado válido da classe, mas somente os estados válidos
  - Exemplo: uma implementação de Pilha que permite as operações *pop()*, *push()*, *isEmpty()* e *isFull()*
  - Não permite colocar um elemento em qualquer ordem

Interface de Classe IV/VII

- Em relação aos comportamentos, existem sete classificações de interfaces
- **Interface com comportamento ilegal**: possibilita uma troca de estado não esperada na abstração da classe
  - Exemplo: inserir um objeto no meio de uma Fila
- **Interface com comportamento perigoso**: necessita que estados ilegais temporários sejam atingidos para fornecer um comportamento por inteiro
  - Exemplo: para mover um Retangulo, enviar quatro mensagens, uma para cada vértice

Interface de Classe V/VII

- **Interface com comportamento irrelevante**: contém método não prejudicial que não faz sentido para a abstração da classe
  - Exemplo: método *previsaoTempo()* em Casa



- **Interface com comportamento incompleto**: falta de comportamento que possibilite uma transição de estado válida
  - Exemplo: não é possível não alterar o preço do aluguel na data de renovação

Interface de Classe VI/VII

- **Interface com comportamento inábil**: necessita que estados não apropriados temporários sejam atingidos para fornecer um comportamento por inteiro
  - Exemplo: para trocar a data de um pedido aprovado, ser necessário transformar o pedido em pendente e aprovar novamente
- **Interface com comportamento replicado:** oferece mais de uma forma de se obter o mesmo comportamento
  - Exemplo: métodos *girarDireita()* e *girar(double angulo)* em Figura

Interface de Classe VII/VII

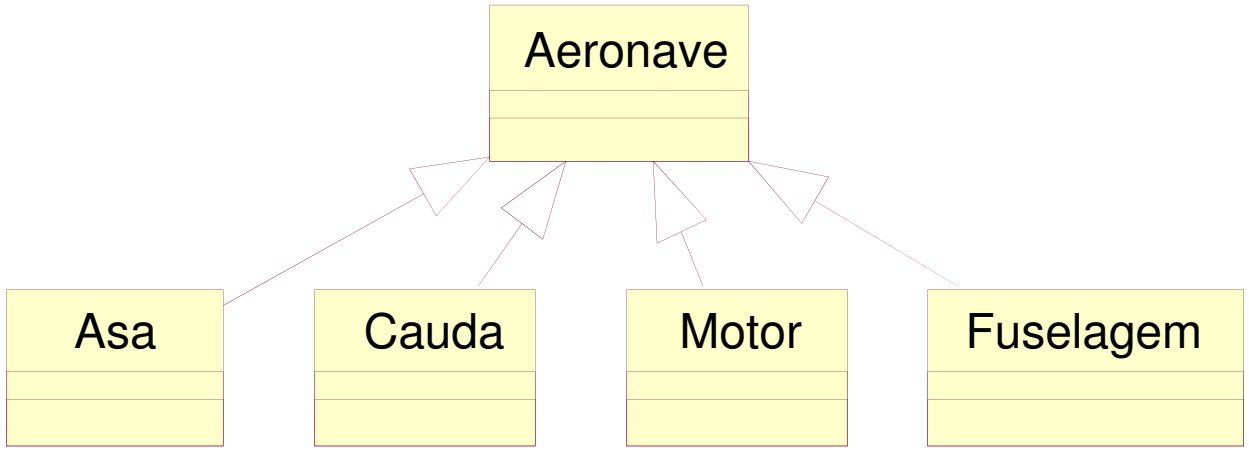
- **Interface com comportamento ideal**: permite que...
  - Objetos em estados válidos somente façam transição para outros estados válidos
  - Transições de estado sejam efetuadas somente através de comportamentos válidos
  - Exista somente uma forma de efetuar um comportamento válido
  - Exemplo: uma implementação de Pilha que permite as operações *pop()*, *push()*, *isEmpty()* e *isFull()*

Perigos Detectados em POO I/VI

- Herança e polimorfismo constituem uma ferramenta poderosa para a modelagem OO
- Entretanto, seu uso excessivo ou equivocado pode ser nocivo à qualidade do modelo produzido

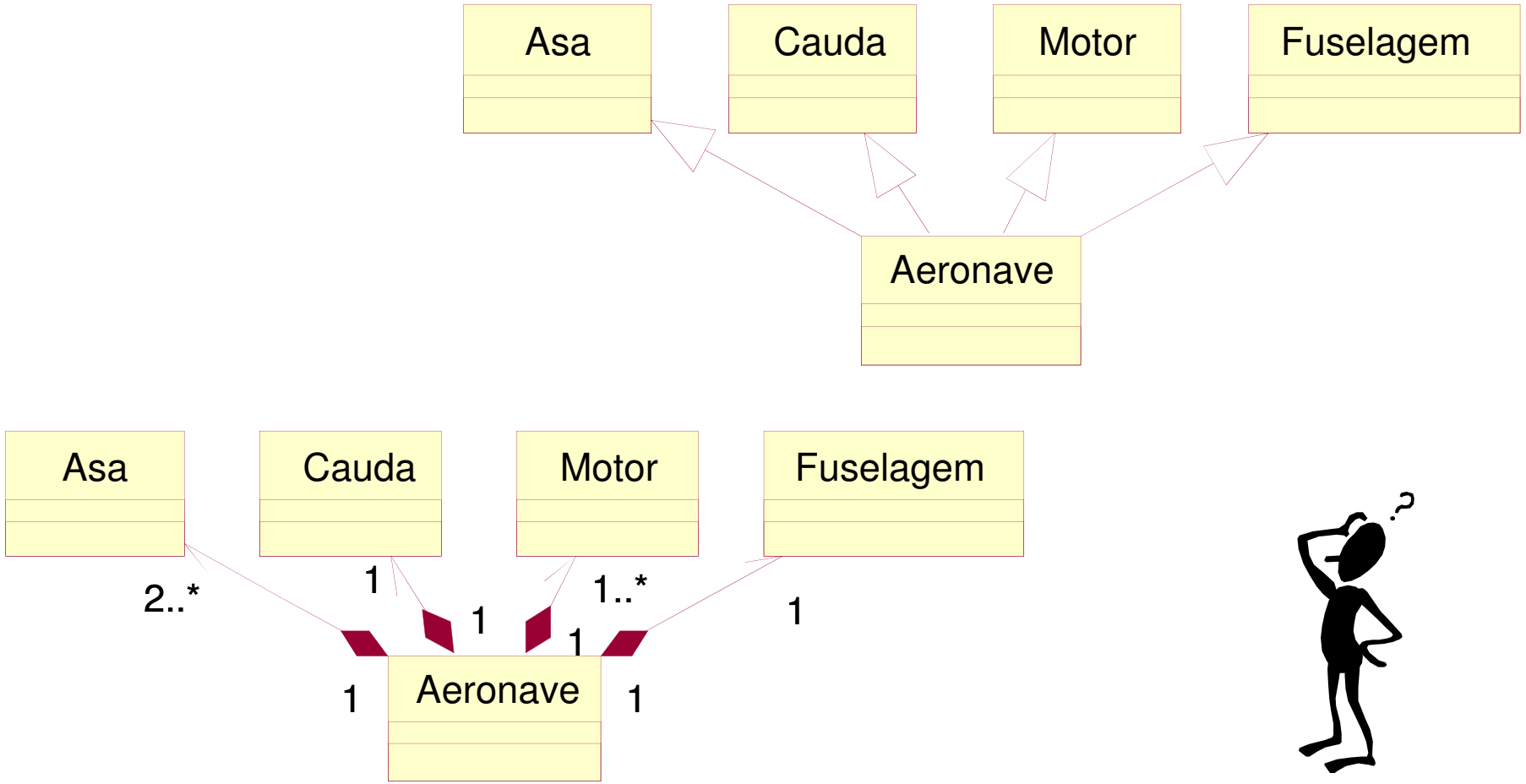
Perigos Detectados em POO II/VI

- Em certas situações, projetistas utilizam equivocadamente herança para mostrar que os sistema são OO



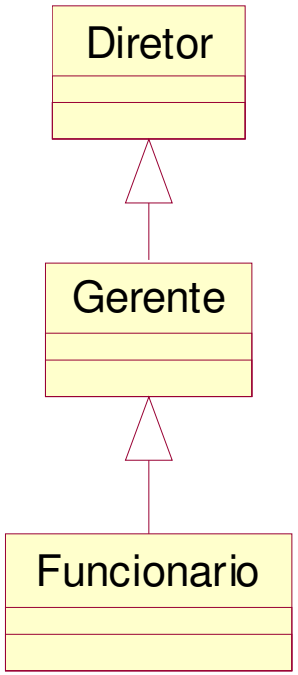
Perigos Detectados em POO III/VI

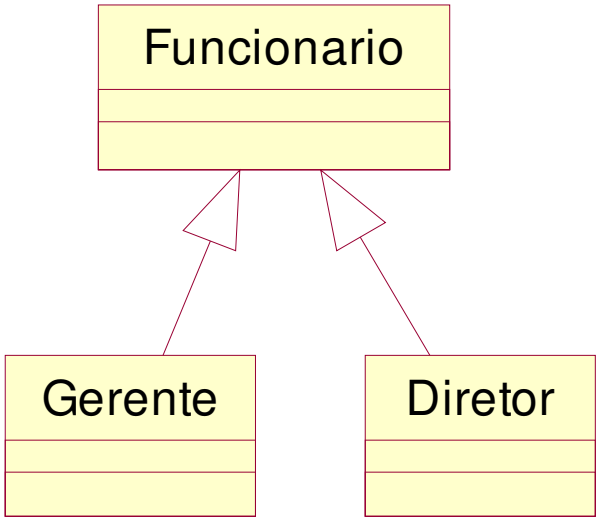
- Confusão entre os conceitos de herança ou composição



Perigos Detectados em POO IV/VI

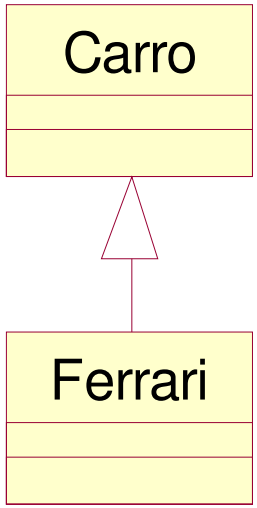
- Confusão entre estrutura hierárquica organizacional e hierarquia de classes OO





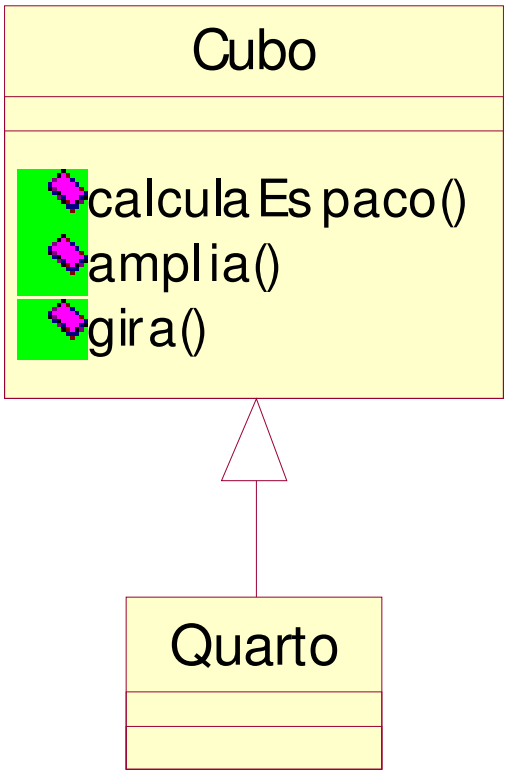
Perigos Detectados em POO V/VI

- Confusão entre os níveis de abstração dos elementos da estrutura (confusão entre classe e instancia)



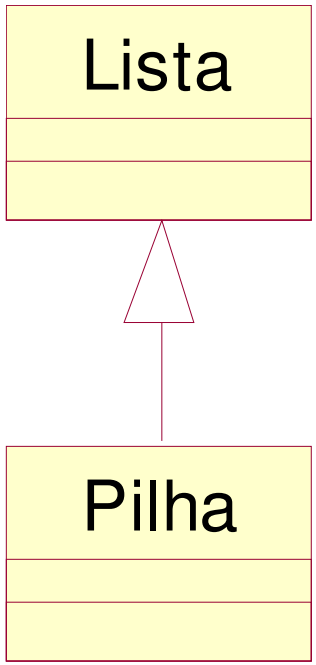
Perigos Detectados em POO VI/VI

- Utilização inadequada da herança (herança forçada)



Exercício

- Qual é o problema no projeto abaixo? Mude a sua estrutura para que ele não tenha mais o problema detectado.
- Descreva métodos e atributos para a nova estrutura.



Bibliografia

- “Fundamentos do Desenho Orientado a Objeto com UML”, Meilir Page-Jones, Makron Books, 2001

- Várias transparências foram produzidas por Leonardo Murta
  - http://www.ic.uff.br/~leomurta