

Diagrama de Classes

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

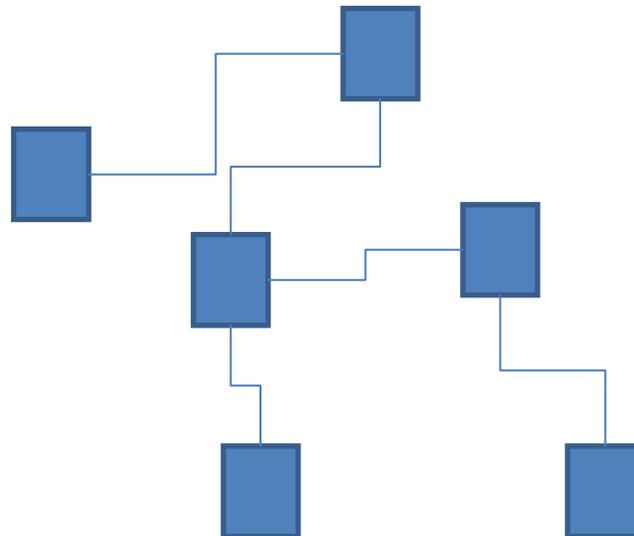
<http://www.ic.uff.br/~viviane.silva/2010.2/es1>

O que é?

- Diagrama mais utilizado da UML
- Representa os tipos (classes) de objetos de um sistema
 - Propriedades desses tipos
 - Funcionalidades providas por esses tipos
 - Relacionamentos entre esses tipos
- Pode ser mapeado diretamente para uma linguagem orientada a objetos
 - Ajuda no processo transitório dos requisitos para o código
 - Pode representar visualmente o código do sistema

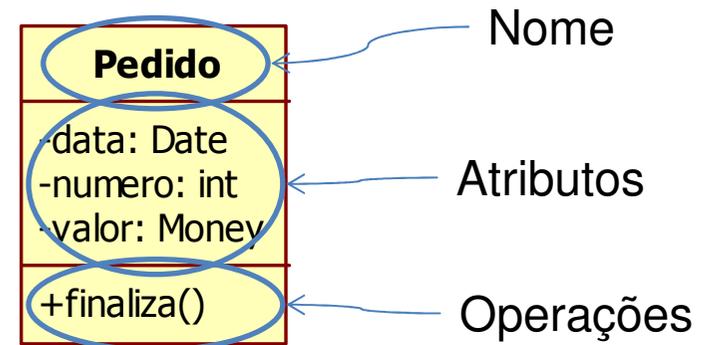
Idéia geral

- Caixas representando as classes
- Linhas representando os relacionamentos



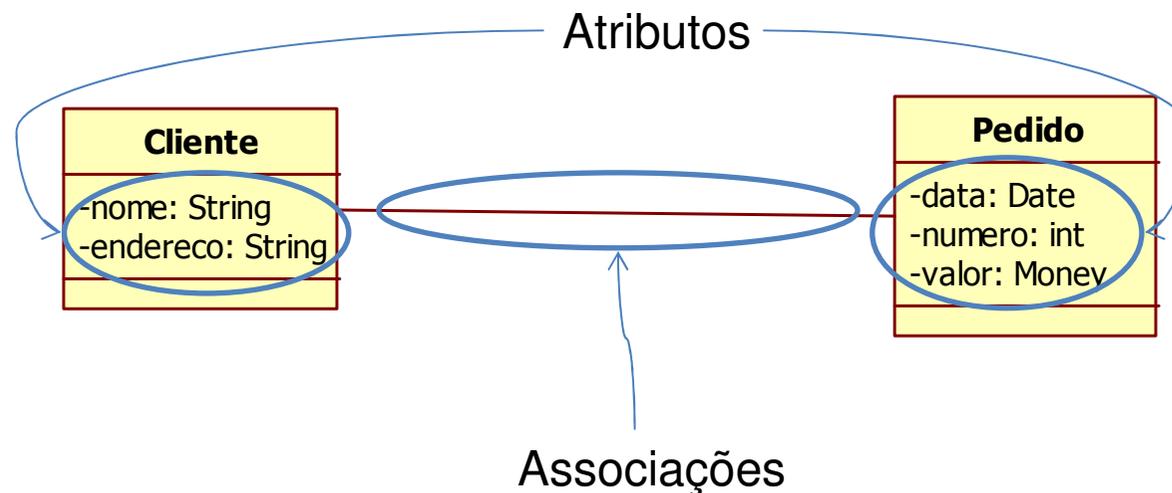
Classes

- As classes são representadas por caixas contendo
 - Nome (obrigatório)
 - Lista de atributos
 - Lista de operações



Propriedades

- Classes são descritas via suas propriedades
 - Primitivas: representadas como atributos
 - Compostas: representadas como associações para outras classes
- Quando transformadas para código, as propriedades se tornam sempre campos da classe



Atributos

- Visibilidade
- Nome
- Tipo
- Multiplicidade
- Valor padrão

- endereco : String[1] = “Sem Endereço”

Atributos: Visibilidade

- Privado (-)
 - Somente a própria classe pode manipular o atributo
 - Indicado na maioria dos casos
- Pacote (~)
 - Qualquer classe do mesmo pacote pode manipular o atributo
- Protegido (#)
 - Qualquer subclasse pode manipular o atributo
- Publico (+)
 - Qualquer classe do sistema pode manipular o atributo

- endereço : String

Atributos: Nome e tipo

- O nome do atributo corresponde ao nome que será utilizado no código fonte
 - É aceitável utilizar nomes com espaço e acentos na fase de análise
- O tipo do atributo corresponde ao tipo que será utilizado no código fonte
 - Tipos primitivos da linguagem
 - Classes de apoio da linguagem (String, Date, Money, etc.)

endereco String

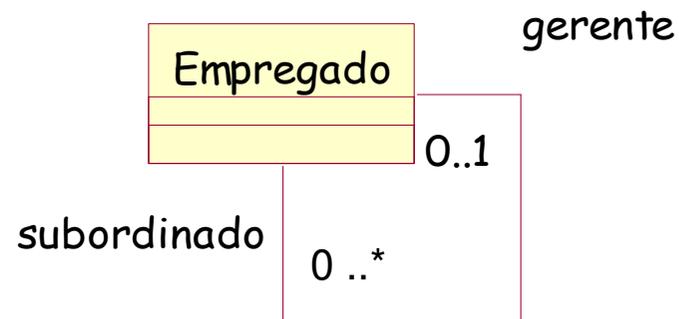
Atributos: Multiplicidade

- Representa o número de elementos de uma propriedade
- Estrutura X..Y onde
 - Opcional: X = 0
 - Mandatório: X = 1
 - Somente um valor: Y = 1
 - Multivalorado: Y > 1
- Valores clássicos
 - 0..1
 - 1 (equivalente a 1..1 → default)
 - * (equivalente a 0..*)
 - 1..*

- endereco : String[0..3]

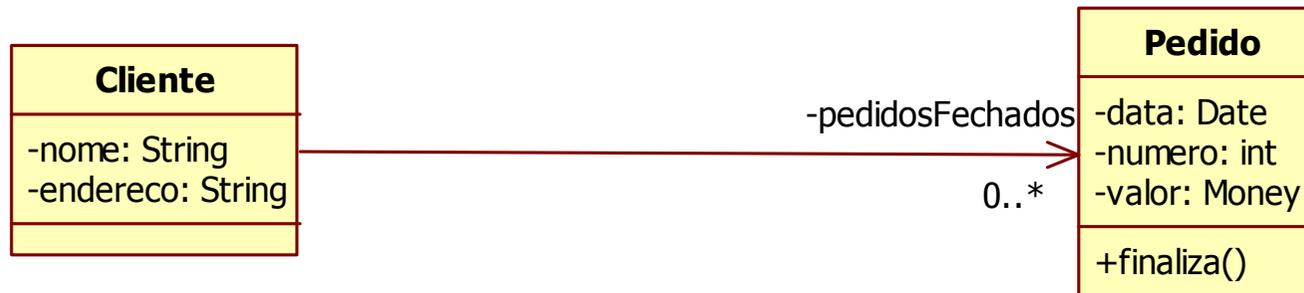
Associação

- Utilizada para relacionar duas classes
- Só as classes que estão relacionadas são as classes cujos objetos podem se comunicar
- Identifica o papel das classes na associação



Atributos de uma associação

- Nome
 - Nome da associação
- Papéis
 - Papéis das classes que estão relacionadas pela associação
 - O papel da classe A é o nome do atributo que a classe B possui que guarda o objetivo da classe A
- Multiplicidades
 - Quantidades de objetos associados a um papel
- Navegabilidade
 - Indica a direção da relação entre as classes



Operações ou métodos

- Operações são descritas via
 - Visibilidade
 - Nome
 - Lista de parâmetros
 - Tipo de retorno

+ finaliza(data : Date) : Money

Operações: Visibilidade

- Valem as mesmas regras de visibilidade de atributos
- Privado (-)
 - Funcionalidades de apoio à própria classe
- Pacote (~)
 - Funcionalidades de apoio a outras classes do pacote (ex. construção de um componente)
- Protegido (#)
 - Funcionalidades que precisam ser estendidas por outras classes (ex. construção de um *framework*)
- Publico (+)
 - Funcionalidades visíveis por todas as classes do sistema

⊕ finaliza(data : Date) : Money

Operações: Nome e tipo de retorno

- Valem as mesmas regras já vistas para atributos...
 - Normalmente o nome de uma operação é formado por um verbo (opcionalmente seguido de substantivo)
 - A ausência de um tipo de retorno indica que a operação não retorna nada (i.e., *void*)

+ finaliza(data : Date) : Money

Operações: Lista de parâmetros

- A lista de parâmetros pode ser composta por zero ou mais parâmetros separados por vírgula
 - Parâmetro: [direção] nome : tipo [= valor padrão]
 - Nome
 - Tipo
 - Primitivo
 - Classe
 - Valor padrão (opcional)

+ finaliza(data : Date) : Money

Análise x Design

- Em análise não se atenha aos detalhes mas em design sim
 - Visibilidade
 - Navegabilidade
 - Tipo
- Visibilidade pública em propriedades
 - Assume campo privado e métodos de acesso (get e set)
- Operações
 - Somente as responsabilidades óbvias das classes

Código: Classes + Associação

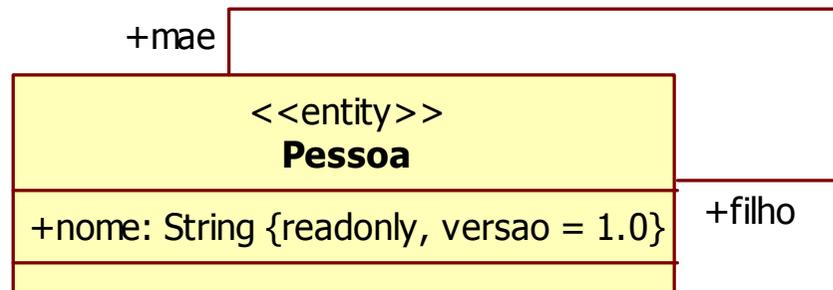
```
public class Cliente {  
    private String nome;  
    private String endereco;  
    private Pedido[] pedidosFechados;  
}
```

```
public class Pedido {  
    private Date data;  
    public Integer numero;  
    private Money valor;  
    public void finaliza() {  
    }  
}
```



Palavras-chave, propriedades e restrições

- Apóiam a linguagem gráfica com informações textuais
- Permitem dar mais semântica aos elementos do modelo
- Notação de palavra-chave (estereótipos)
 - Textual: <<palavra>> (ex.: <<interface>>)
 - Icônica: imagem representando a palavra-chave
- Notação de propriedades e restrições
 - {propriedade} (ex.: {readonly}) só operação de leitura
 - {nome = valor} (ex.: {versão = 1.0})
 - {restrição} (ex.: {Mãe deve ser do sexo feminino})



Propriedades de atributos e associações

Alguns exemplos...

➤ {readonly}

- Somente oferece operações de leitura

➤ {ordered}, {unordered}

- Indica se o atributo ou associação multivalorado mantém a seqüência dos itens inseridos

➤ {unique}, {nonunique}

- Indica se o atributo ou associação multivalorado permite repetição

- endereco : String = "Sem Endereço" {readonly}

Propriedades de operações

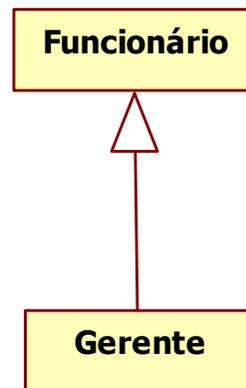
- {query}
 - Não modifica o estado do sistema após a execução
- {sequential}
 - A instância foi projetada para tratar uma thread por vez, mas não é sua responsabilidade assegurar que isso ocorra
- {guarded}
 - A instância foi projetada para tratar uma thread por vez, e é sua responsabilidade assegurar que isso ocorra (ex.: metodos synchronized em Java)
- {concurrent}
 - A instância é capaz de tratar múltiplas threads concorrentemente

Outros relacionamentos entre classes

- Além das associações, alguns outros tipos de relacionamentos são importantes
 - Generalização
 - Realização
 - Composição
 - Agregação
 - Dependência
 - Classes de associação

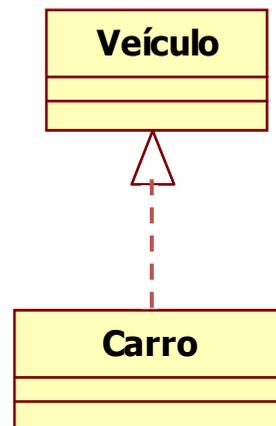
Generalização

- Visa estabelecer relações entre tipos
- Leitura: “é um”
- Se Gerente “é um” Funcionário
 - Todas as operações e propriedades (não privadas) de Funcionário vão estar disponíveis em Gerente
 - Toda instância de Gerente pode ser utilizada aonde se espera instâncias de Funcionário
 - Gera o efeito de herança e polimorfismo quando mapeado para código



Realização

- Visa estabelecer relações entre tipos
- Leitura: “implementa”
- É utilizada entre uma classe e uma interface
- Se Carro “implementa” Veículo
 - Todas as operações definidas em Veículos devem ser implementadas em Carro
 - Carro pode utilizar os atributos definidos em Veículo (que sempre são constantes).



Código: Generalização e Realização

➤ Generalização

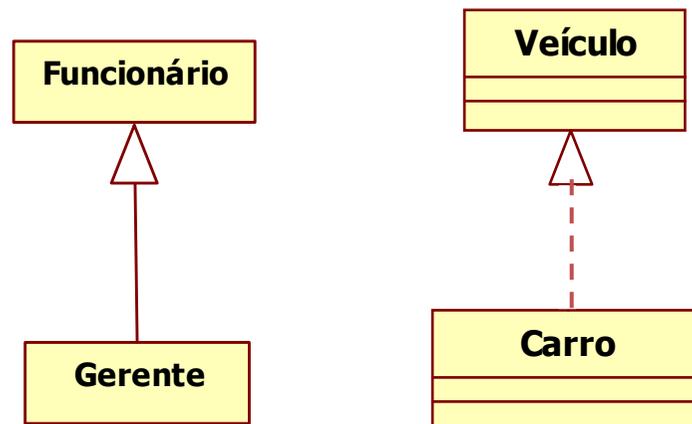
```
public class Funcionario { ... }
```

```
public class Gerente extends Funcionario { ... }
```

➤ Realização

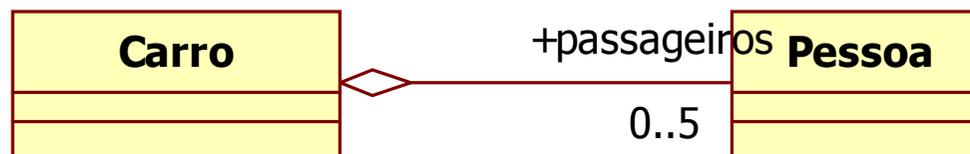
```
public class Veículo { ... }
```

```
public class Carro implements Veículo { ... }
```



Agregação

- É uma associação com a semântica de “contém”
- Serve como uma relação todo-parte fraca
- O todo existe sem as partes
- As partes existem sem o todo

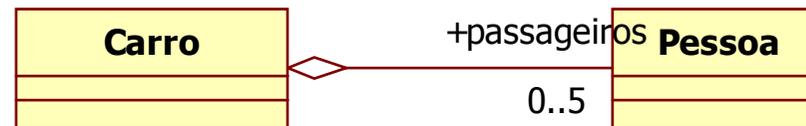


Código: Agregação

```
public class Pessoa {  
}
```

```
public class Carro implements Veículo {  
    public Pessoa[5] passageiros;  
}
```

- Qual é a diferença entre agregação e associação no código?
 - Nenhuma.



Composição

- É uma associação com a semântica de “é composto de”
- Serve como uma relação todo-parte forte
- As partes não existem sem o topo
 - As partes pertencem a somente um todo
 - A remoção do todo implica na remoção das partes



Código: Composição

```
public class Carro implements Veículo {  
    public Pessoa[5] passageiros;  
    public Peça[] peças;  
}
```

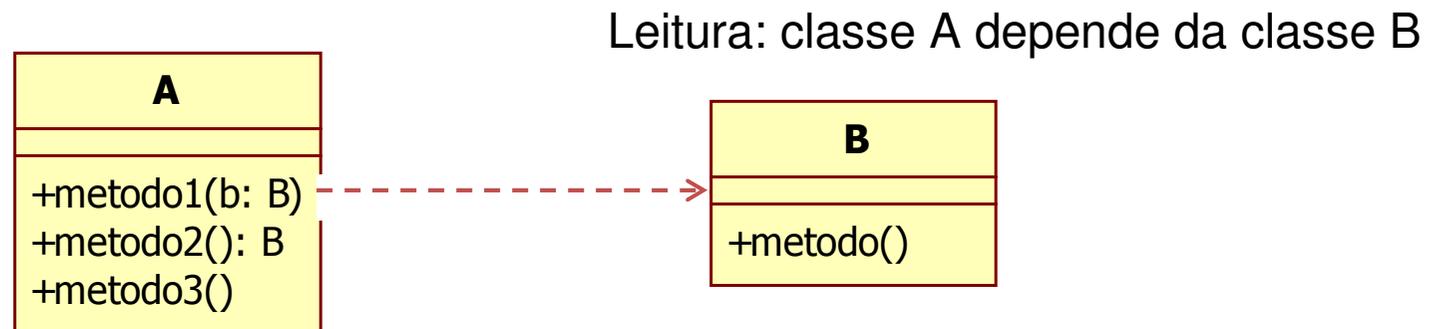
```
public class Peça {  
}
```

- Qual é a diferença entre composição e associação no código?
 - Na declaração das classes, métodos e atributos, nenhuma.
 - A diferença está na implementação dos métodos de destruição do todo e de criação das partes



Dependência

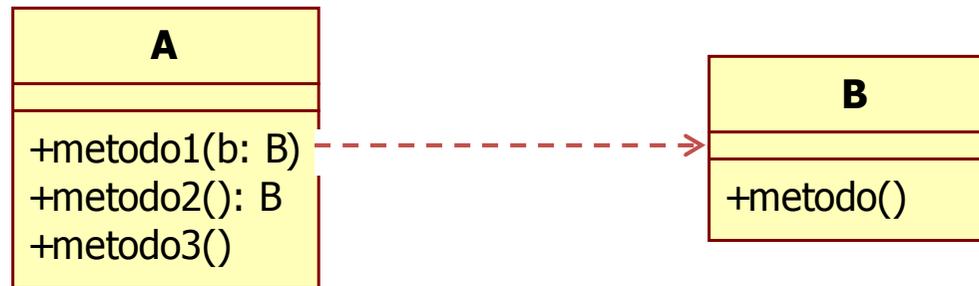
- Deixa explícito que mudanças em uma classe podem gerar conseqüências em outra classe
- Exemplos:
 - Uma classe chama métodos de outra (método 3)
 - Uma classe tem operações que retornam outra classe (método 2)
 - Uma classe tem operações que esperam como parâmetro outra classe (método 1)
- Outros relacionamento (ex.: associação com navegação) implicitamente determinam dependência



Código: Dependência

```
public class B {  
    public void metodo() {...}  
}  
public class A {  
    public void metodo1(B b) {...}  
    public B metodo2() {...}  
    public void metodo3() {  
        B b;  
        b = new B();  
        b.metodo();  
    }  
}}
```

➤ O relacionamento de dependência não é visível na definição dos atributos da classe, e sim nos métodos.

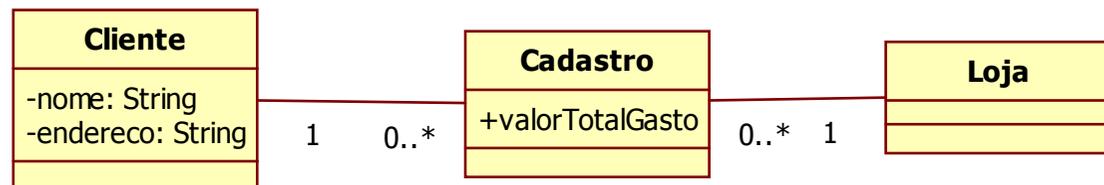
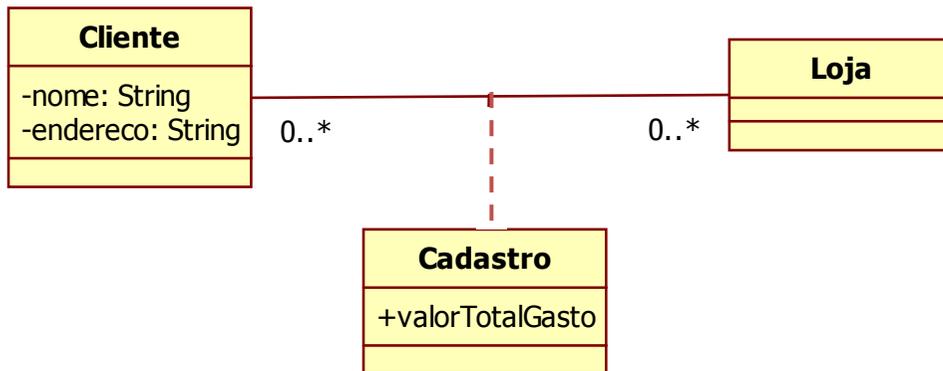


Classes de associação

- Permitem a adição de informações em uma associação
- Devem ser transformadas em classes comuns posteriormente para viabilizar implementação

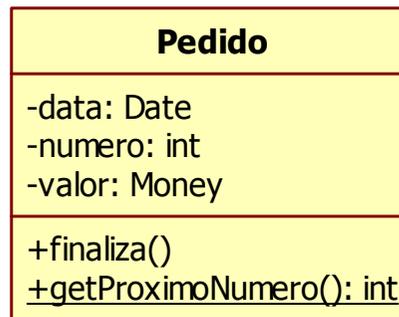


Qual o valor total gasto por um cliente em cada loja?



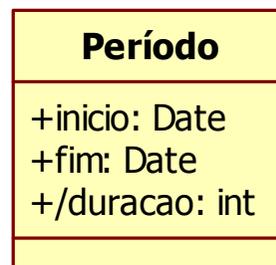
Propriedades e operações estáticas

- Propriedades que não são instanciadas nos objetos
- Operações que atuam somente sobre propriedades estáticas
- Ambos são acessados diretamente na classe
 - Ex.: Pedido.getProximoNumero()
- Não é necessário um objeto para acessar a propriedade
- São sublinhadas no diagrama



Propriedades derivadas

- São propriedades que na verdade não existem como atributos ou associações
- Podem ser inferidas por outras propriedades da classe
- É interessante explicitar através de nota ou restrição a fórmula de derivação
- São marcadas com o símbolo “/”



duração = fim - início

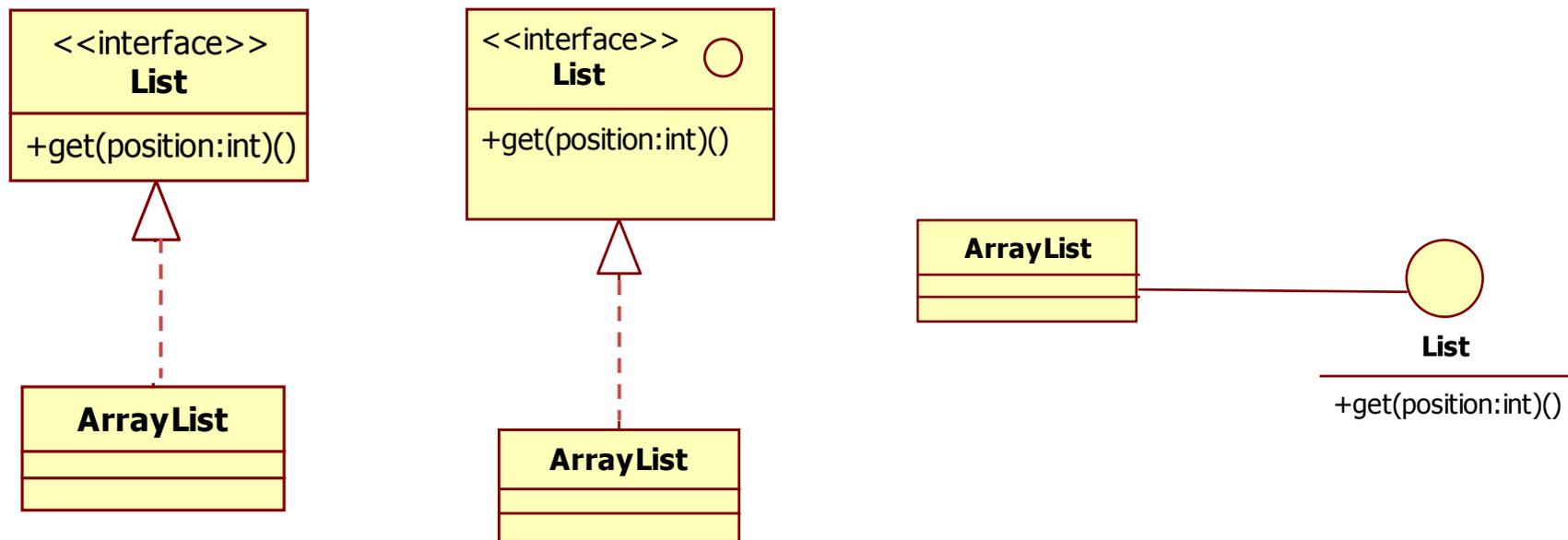
Classes e operações abstratas

- Classes que não podem ter instâncias
 - Usualmente têm operações abstratas, ou seja, sem implementação
- Suas subclasses usualmente são concretas
 - Implementam métodos com comportamentos específicos para as operações abstratas
- Utilizam nome em itálico



Interfaces

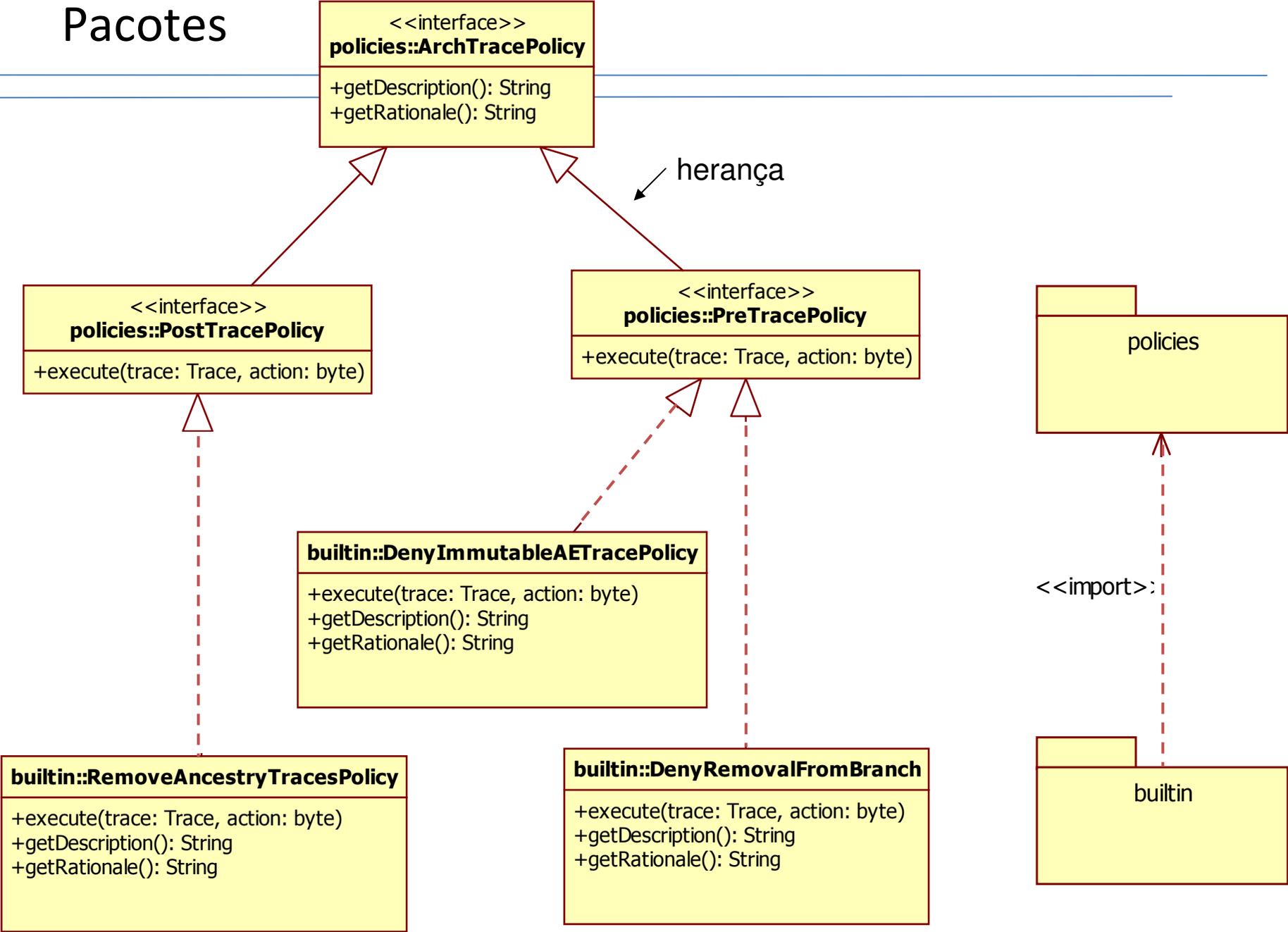
- Uma classe sem nenhuma implementação
 - Todas operações são abstratas
- Faz uso da palavra-chave <<interface>>
 - Pode ser representado também como um ícone
- O relacionamento de realização indica as classes que implementam a interface
 - Equivalente a generalização



Pacotes

- Em algumas situações se deseja ter uma visão geral das partes do sistema
- Para isso, o **diagrama de pacotes** é a ferramenta indicada
- Pacotes agregam classes e outros pacotes
 - Dependências podem ser inferidas indiretamente
- Exemplo
 - Classe C1 pertence ao pacote P1
 - Classe C2 pertence ao pacote P2
 - Classe C1 depende da classe C2
 - Logo, pacote P1 depende do pacote P2

Pacotes



Dicas

- Inicie com um diagrama simples
- O que normalmente tem em todo diagrama
 - Classes
 - Atributos
 - Operações
 - Associações
- Use os demais recursos da linguagem somente quando for realmente necessário

Dicas: Possíveis candidatos

➤ Classes

- Entidades externas que produzem ou consomem informações (ex.: sistema de validação do cartão de crédito)
- Coisas que são parte do problema e que são informações compostas (ex.: Produto)
- Eventos que ocorrem durante a operação do sistema (ex.: Pedido)
- Papeis que interagem com o sistema (ex.: Cliente)
- Unidades organizacionais relevantes (ex.: Rede de lojas)
- Lugares que fornecem o contexto do problema ou do sistema (ex.: Loja)
- Estruturas definidas no problema (ex.: Estoque)

Dicas: Possíveis candidatos

➤ Atributos

- Informação primitiva que precisa ser memorizada (ex.: Preço)

➤ Associações

- A classe A precisa se relacionar com a classe B para atender a operações específicas (ex.: Cliente – Pedido)

➤ Operações

- Funcionalidades que devem ser providas por uma classe para viabilizar o uso do sistema (ex.: calculaTotal em Pedido)

Dicas: Possíveis candidatos

➤ Atributos

- Informação primitiva que precisa ser memorizada (ex.: Preço)

➤ Associações

- A classe A precisa se relacionar com a classe B para atender a operações específicas (ex.: Cliente – Pedido)

➤ Operações

- Funcionalidades que devem ser providas por uma classe para viabilizar o uso do sistema (ex.: calculaTotal em Pedido)

Exemplo

- Uma loja que vende roupas possui um sistema capaz de controlar a venda e o estoque. Cada roupa possui um preço, um código e uma descrição. Um roupa pode possuir vários exemplares. Cada exemplar tem um tamanho e uma cor associados.
- Os clientes da loja são cadastrados pelo nome e quando a venda é realizada, o sistema guarda informação de qual(is) foi(foram) o(s) exemplar(es) vendido(s) para o cliente.

Exemplo 2

- Um sistema de gerenciamento de submissões pode gerenciar vários eventos ao mesmo tempo. Cada evento possui um único gerente, vários revisores, vários autores e é composto por um conjunto de artigos submetidos para o evento. Um mesmo artigo só pode ser submetido para um único evento.
- Todas as pessoas que acessam um sistema de gerenciamento de submissões de artigos possuem um nome, uma instituição à qual pertencem, um email, um login e uma senha. Um artigo tem um autor que escreveu o artigo, possui um título, um resumo, um corpo de texto e uma área. Note que um autor pode ter escrito vários artigos.
- Todo artigo tem no mínimo 2 e no máximo 4 revisores. Cada revisor, que está associado a uma única área, pode fazer a revisão de até 3 artigos. A revisão de um artigo feita por um revisor contém uma nota dada pelo revisor e um comentário.
- Faça um diagrama de classes que represente o sistema de submissão. Preocupe-se com o nome das classes, os atributos, os tipos das relações entre as classes e as multiplicidades. Não é necessário modelar os métodos.

Bibliografia

- Fowler, Martin. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. Addison-Wesley Professional.
- Pressman, Roger. 2004. *Software Engineering: A Practitioner's Approach*. 6th ed. McGraw-Hill.
- Várias transparências foram produzidas por Leonardo Murta
 - <http://www.ic.uff.br/~leomurta>