

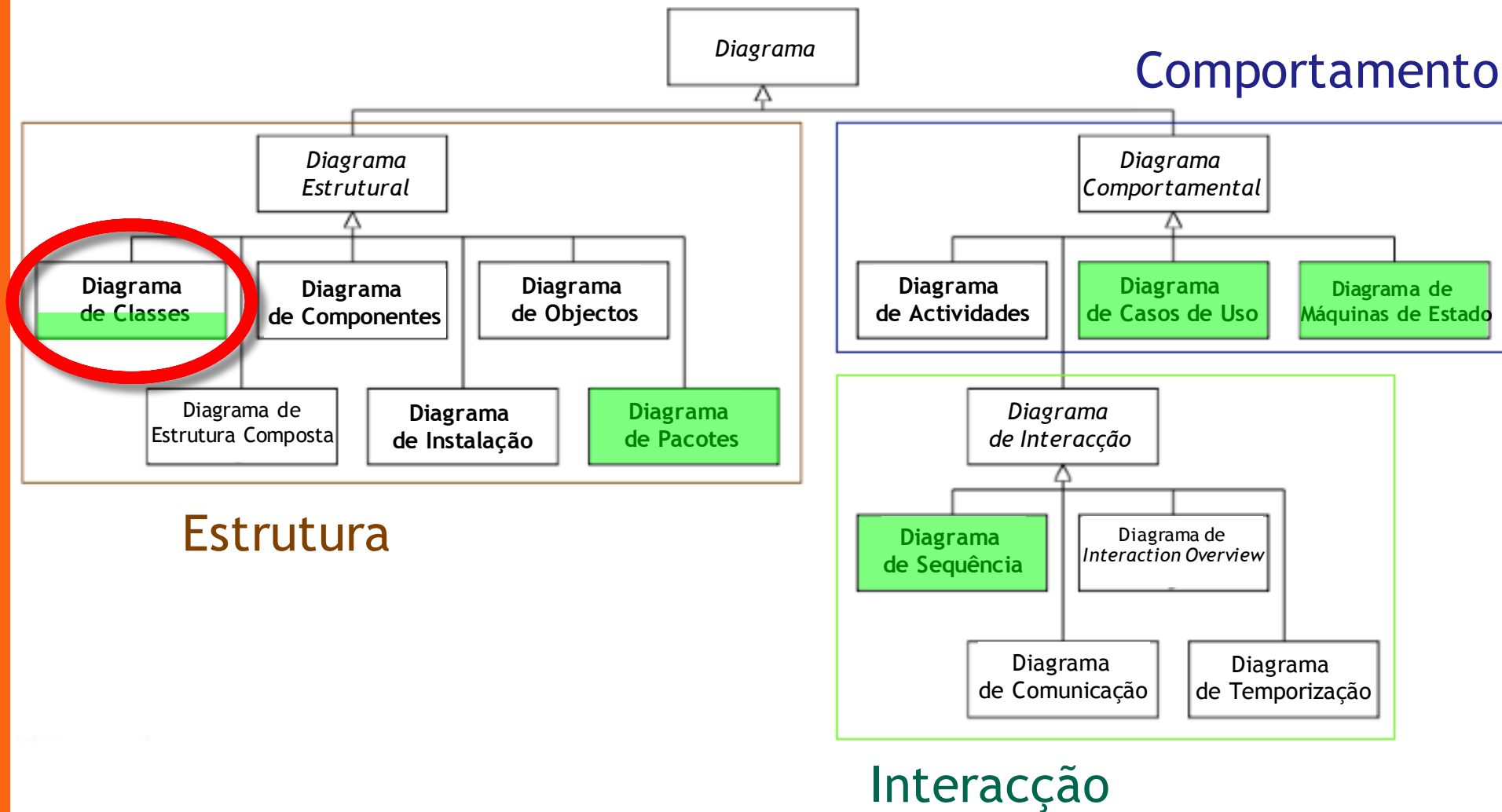


# Desenvolvimento de Sistemas Software

## Aula Teórica 16: Modelação Estrutural / Diagramas de Classe



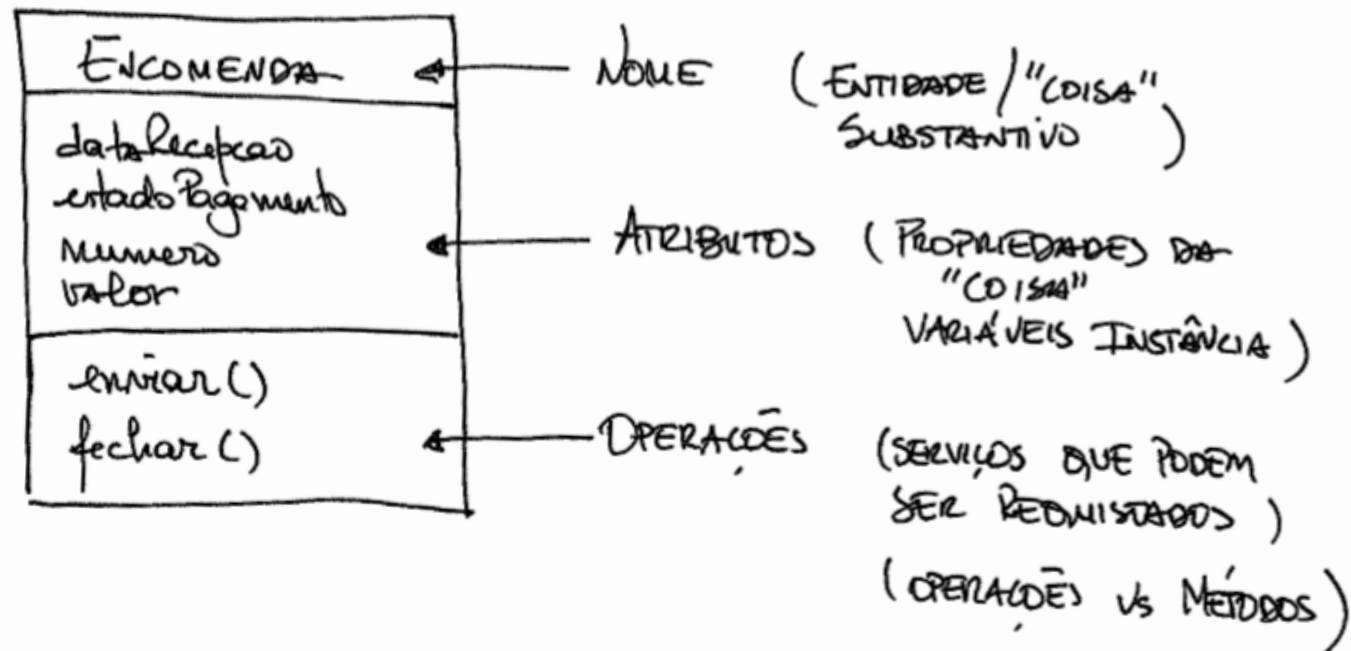
# Diagramas da UML 2.x



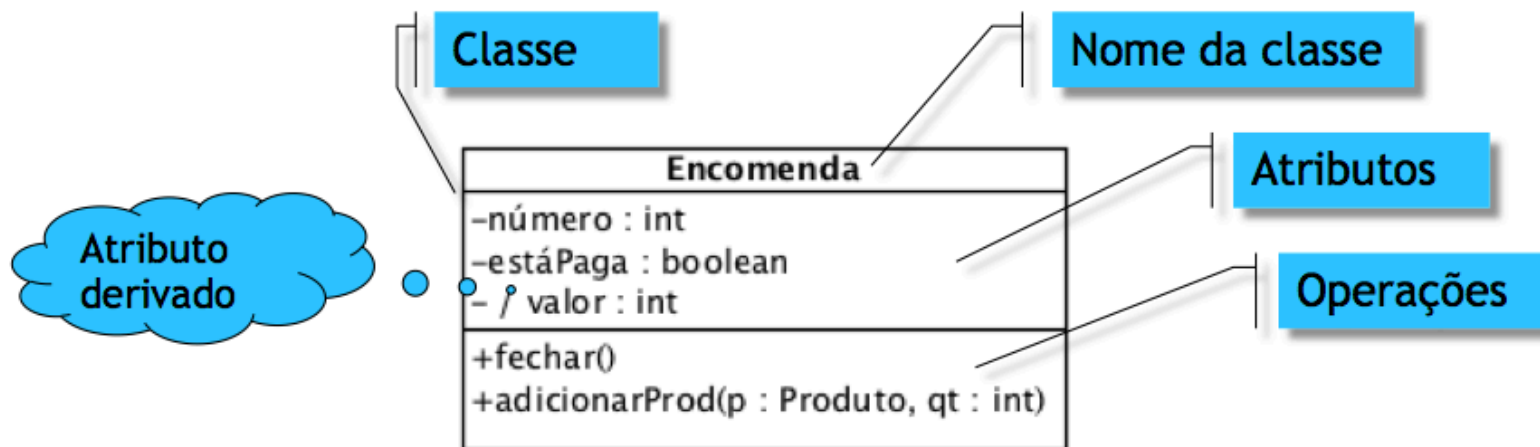


## Revisão do conceito de classe

- Base de um qualquer sistema OO
- Cada classe descreve um conjunto de objectos com a mesma estrutura de dados e comportamento
- Exemplo:



# Representação de classes em UML



- Compartimentos pré-definidos
  - Nome da classe – começa com maiúsculas / substantivo
  - Atributos (de instância) – representam propriedades das instâncias desta classe / começam com minúsculas / substantivos
  - Operações (de instância) – representam serviços que podem ser pedidos a instâncias da classe / começam com minúsculas / verbos
- Compartimentos podem ser omitidos – isso não significa que não exista lá informação!



# Níveis de modelação

- Podemos considerar 3 níveis de modelação:

- Conceptual
- Especificação
- Implementação

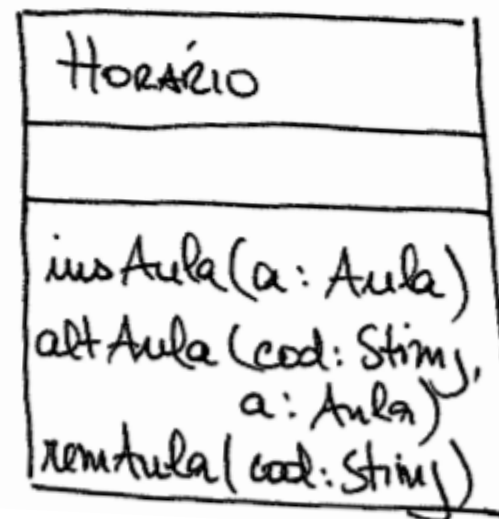
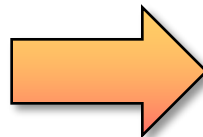
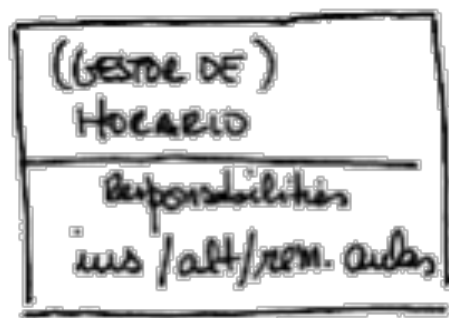


- Nível Conceptual

- Representação dos conceitos no domínio de análise
- Não corresponde necessariamente a um mapeamento directo para a implementação
- Cf. Modelo de Domínio

# Níveis de modelação

- Nível de especificação
  - Definição das interfaces (API's)
  - Identificar responsabilidades e modelá-las com operações/atributos
- Exemplo:

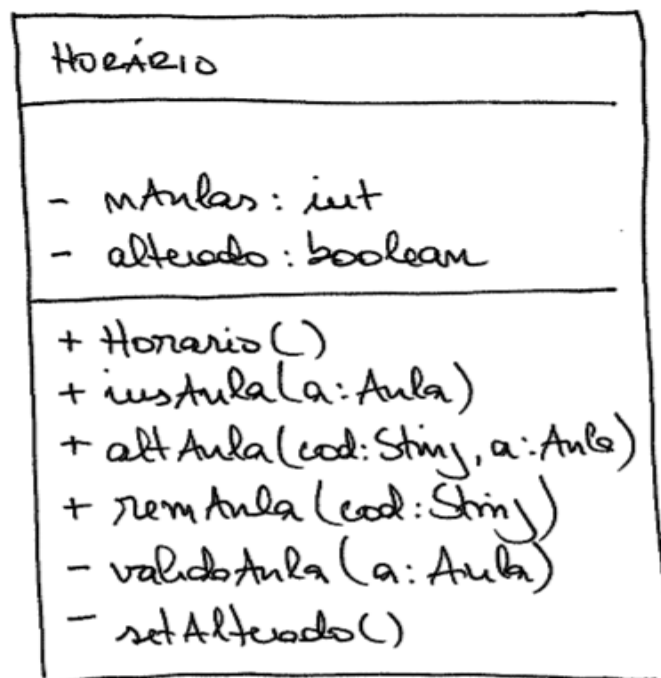




# Níveis de modelação

- Nível de implementação
  - Definição concreta das classes a implementar - geração de código
  - Definição dos relacionamentos estruturais entre as entidades

- Exemplo:





# Visibilidade de atributos e operações

- O nível de visibilidade (acesso) que se pretende para cada atributo/operação é representado com as seguintes anotações:
  - privado — só acessível ao objecto a que pertence (cf. encapsulamento)
  - # protegido — acessível a instâncias das sub-classes (atenção: em Java fica também acessível a instâncias de classes do mesmo *package*!)
  - pacote/*package* — acessível a instâncias de classes do mesmo *package* (nível de acesso por omissão)
  - + público — acessível a todos os objectos no sistema (que conheçam o objecto a que o atributo/operação pertence!)





# Declaração de atributos / operações

Só o nome é obrigatório!

- Atributos

«*esteréotipo*» *visibilidade* / nome : *tipo* [*multiplicidade*] = valorInic {propriedades}

- Exemplos

morada

- morada= “Braga” {addedBy=“jfc”, date=“18/11/2011”}
- morada: String [1..2] {leaf, addOnly, addedBy=“jfc”}

Propriedades comuns:

changeability:

changeable - pode ser alterado (o *default*)

frozen - não pode ser alterado (final em Java)

addOnly - para multiplicidades > 1 (só adicionar)

leaf - não pode ser redefinido

ordered - para multiplicidades > 1



# Declaração de atributos / operações

## • Operações

Obrigatório!

in | out | inout | return

«*esteréotipo*» *visibilidade* nome (direção nomeParam : tipo = valorOmiss) : *tipo*

{propriedades}

## • Exemplos

setNome

+ setNome(nome = "SCX") {abstract}

+ getNome() : String {isQuery, risco = baixo}

# getNome(out nome) {isQuery}

+ «create» Pessoa()

por omissão é "in"

in - parâmetro de entrada  
out - parâmetro de saída  
inout - parâmetro de entrada/saída  
return - operação retorna o parâmetro como um dos seus valores de retorno

Propriedades comuns:

abstract - operação abstrata

leaf - não pode ser redefinido

isQuery - não altera o estado do objecto



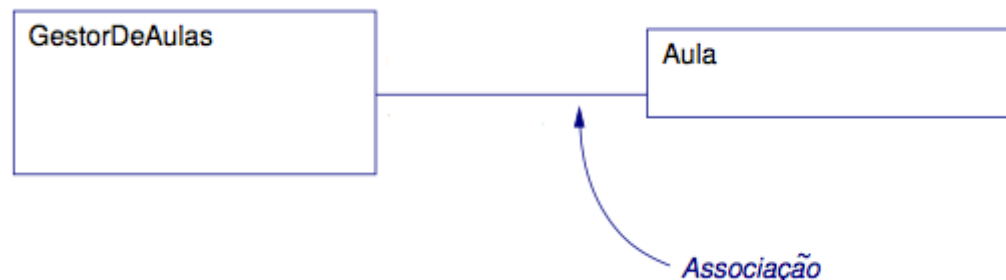
# Relações entre classes

- Três tipos de relações possíveis entre as classes:
  - Associação  
indica que existe algum tipo de ligação entre objectos das duas classes
  - Dependência  
indica que uma classe depende de outra
  - Generalização/Especialização  
relação entre classe mais geral e classe mais específica



# Relações entre classes - Associação

- Notação:



- Indica que objectos de uma estão ligados a objectos de outra — define uma relação entre os objectos
- Noção de navegabilidade (cf. diagramas E-R)
- Por omissão representam navegação bidireccional — mas pode indicar-se explicitamente o sentido da navegabilidade.



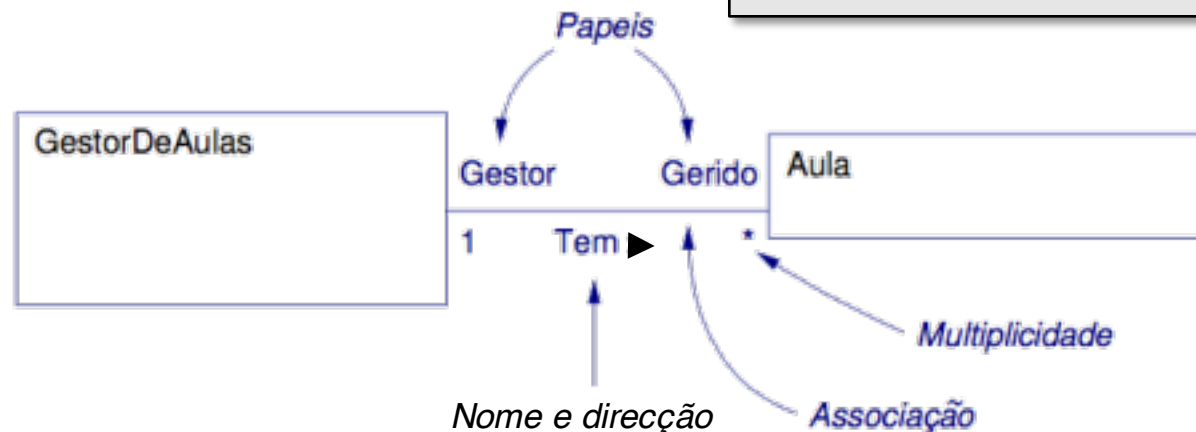


# Relações entre classes - Associação

- Três decorações possíveis:
  - nome** — descreve a natureza da relação (pode ter direcção)
  - papeis** — indica o papel que cada classe desempenha na relação definida pela associação (usualmente utilizado como alternativa ao nome)
  - multiplicidade** — quantos objectos participam na relação:
    - \* — zero ou mais objectos
    - n*** — ***n*** objectos (***n***  $\geq$  1)
    - n..m*** — entre ***n*** e ***m*** objectos (***n***  $<$  ***m***)

## Casos particulares:

- 1 — um objecto = objecto obrigatório
- 0..1 — zero ou um objectos = objecto opcional
- n*..\*** — mais de ***n*** objectos



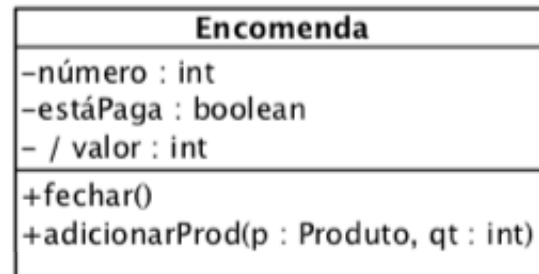


# Associações vs. Atributos

- Atributos (de instância) representam propriedades das instâncias das classes
  - são codificados como variáveis de instância
- Associações também representam propriedades das instâncias das classes
  - também são codificados como variáveis de instância
- Atributos devem ter tipos simples
  - **utilizar associações para tipos estruturados**

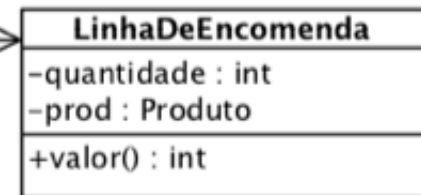
## Erro de Principiante!

Repetir as associações nos atributos.

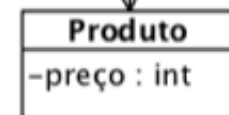


linhas

1..\*



1 prod



```
class LinhaDeEncomenda {
    private int quantidade;
    private Produto prod;
    private Produto prod;

    public int valor() {...}
}
```



# Relações entre classes - Dependência

- Notação:

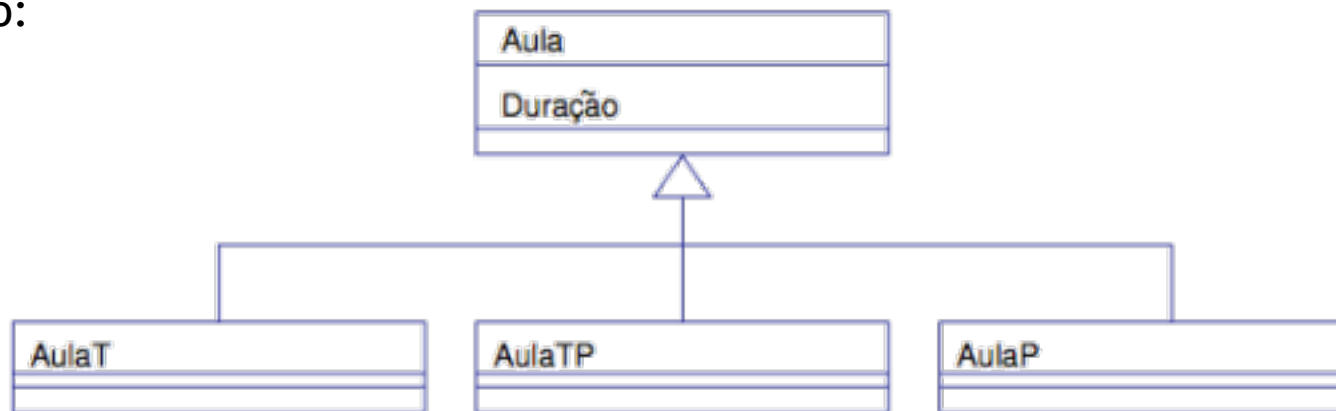


- Indica que a definição de uma classe está dependente da definição de outra.
- Utiliza-se normalmente para mostrar que instâncias da origem utilizam, de alguma forma, instâncias do destino (por exemplo: um parâmetro de um método)
- Uma alteração no destino (quem é usado) pode alterar a origem (quem usa)
- **Diminuir o número de dependências deve ser um objectivo.**



# Relações entre classes - Generalização/Especialização

- Indica a relação entre uma classe mais geral (super-classe) e uma classe mais específica (sub-classe).
- Noção de *is-a* – tipagem / substitubilidade
- Polimorfismo – duas sub-classes podem fornecer métodos diferentes para implementar uma operação da super classe.
- *Overriding* – sub-classe pode alterar o método associado a uma operação declarada pela super-classe
- Herança simples vs. herança múltipla
- Notação:

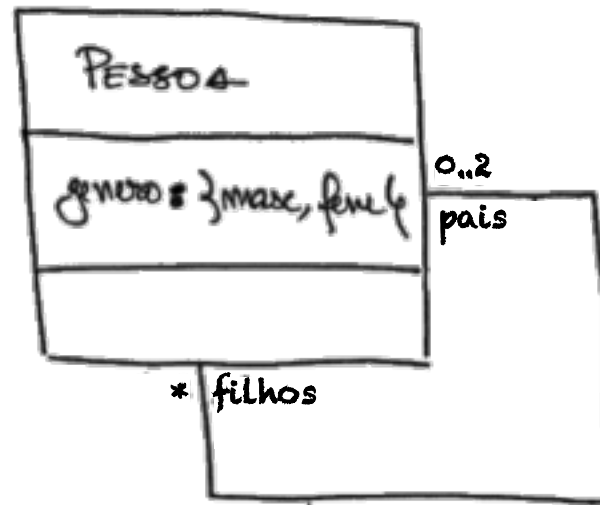






# Relações entre classes - mais sobre Associações

- Associação reflexiva
  - Define uma relação entre objectos da mesma classe



- Uma pessoa pode ter filhos / pai e mãe



# Relações entre classes - mais sobre Associações

- Por vezes a relação entre duas classes implica uma relação todo-parte
  - mais forte que simples associação
  - Exemplo: uma Turma é constituída por Alunos

- Agregação

- Os alunos fazem parte da estrutura interna da Turma
- Apesar disso, os Alunos tem existência própria



- Composição

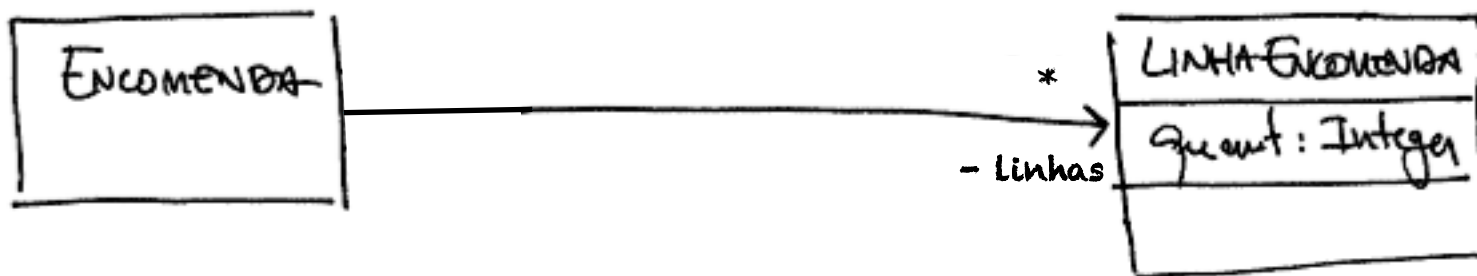
- Os alunos (da Turma) só existem no contexto da Turma
- Os alunos não têm existência para além da existência da Turma





## Associações qualificadas

- Produto é chave na relação entre Encomenda e LinhaEncomenda
  - Para cada produto p existe (no máximo) uma linha de encomenda

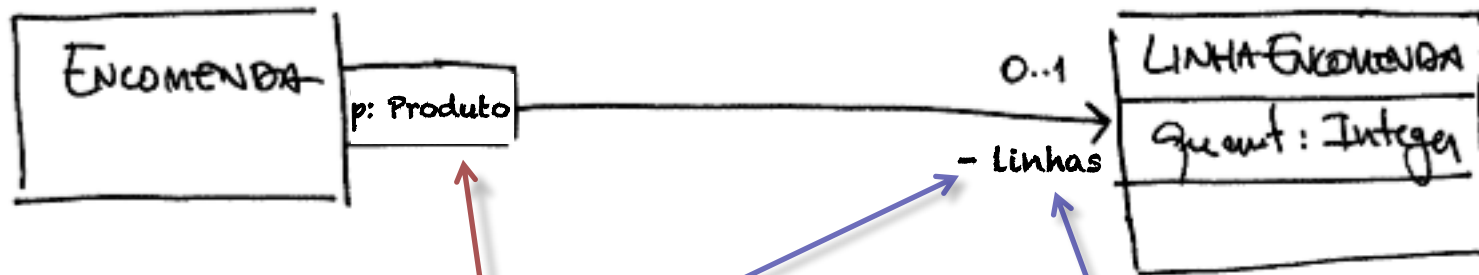


```
public class Encomenda {  
  
    private Map<Produto, LinhaEncomenda> linhas;  
    ...  
    public LinhaEncomenda getLinhaEnc(Produto umProd);  
    public void addLinhaEncomenda(Integer qt,  
                                    Produto umProd);  
    ...  
}
```



# Associações qualificadas

- Produto é chave na relação entre Encomenda e LinhaEncomenda
  - Para cada produto p existe (no máximo) uma linha de encomenda

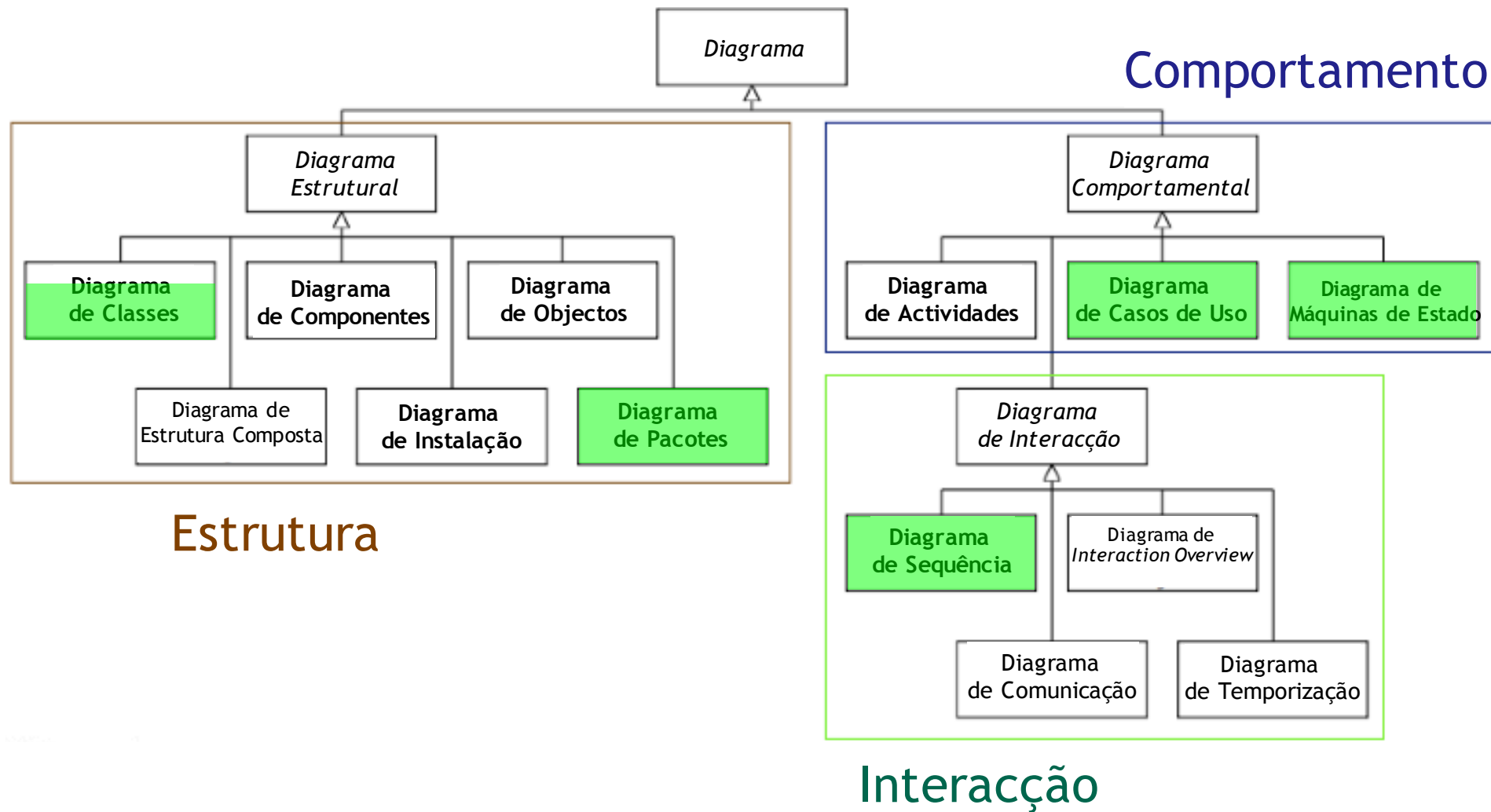


```

public class Encomenda {
    private Map<Produto, LinhaEncomenda> linhas;
    ...
    public LinhaEncomenda getLinhaEnc(Produto umProd);
    public void addLinhaEncomenda(Integer qt,
                                    Produto umProd);
    ...
}
  
```



# Diagramas da UML 2.x





# Modelação Estrutural

## Sumário

- Diagramas de Classe II
  - Níveis de modelação
  - Relações entre as classes
    - Herança/especialização
    - Dependências
    - Associação bidireccional vs. unidireccional
    - Agregação vs. Composição vs. Associação simples
    - Associações qualificadas