

Relacionamento de classes

Prof. Hugo de Paula



PUC Minas



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Engenharia de Software

Sumário

- 1 Relacionamento entre objetos
 - Associação
 - Agregação e composição
 - Cardinalidade das associações
- 2 Exemplo: Controle de Estoque
- 3 Projeto de relações
 - Papeis
 - Navegabilidade

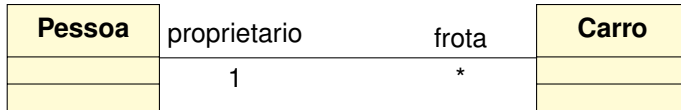
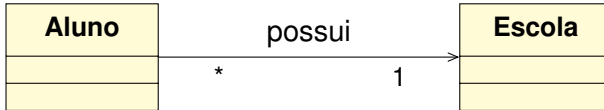
Uso e reuso

- Programar é uma atividade de repetição com pequenas modificações.
- Seria desejável construir um catálogo de componentes de software de tal forma a construir um novo sistema.
- Obstáculos: as repetições não são exatas.
- Reusabilidade é fundamental para reduzir custos e aumentar confiabilidade.
- Modularidade é a chave para atingir alto grau de reusabilidade.

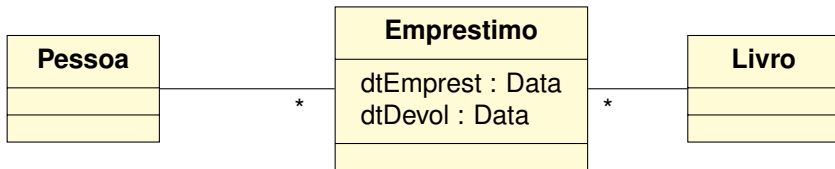
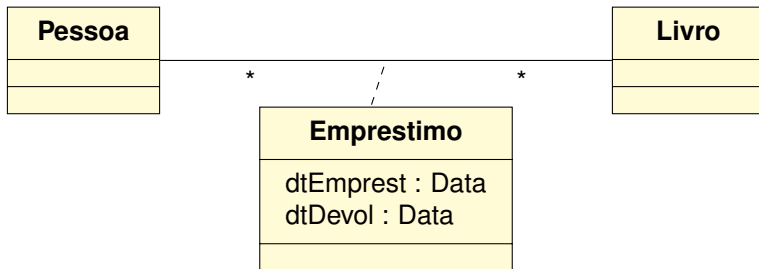
Relacionamentos entre objetos

- **Associação:** objeto “usa” serviços de outro objeto.
 - Mensagens disparam operações.
 - Operações (métodos) retornam resultados.
- **Agregação:** objeto definido em termos de seus componentes.
 - relação parte/todo (“tem um”).
- **Composição:** relação “está contido”.
 - dependência de tempo de vida entre a parte e o todo.

Associação



Classe de associação

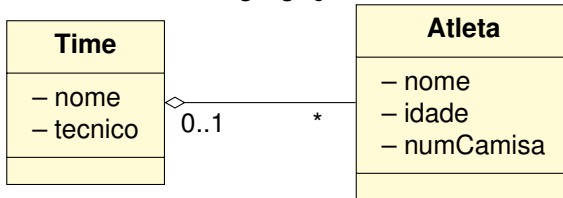


Agregação x composição

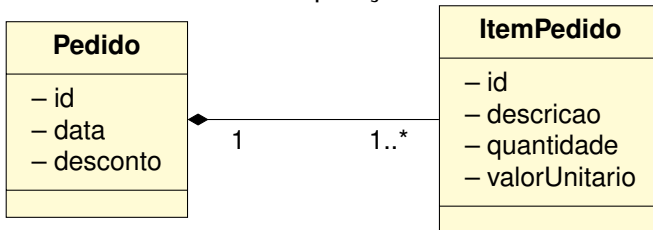
- **Agregação:** a existência da “parte” faz sentido, mesmo não existindo “todo”.
 - partes podem eventualmente pertencer a mais de um todo (não simultaneamente).
 - Ex: Atleta → Time
- **Composição:** relacionamento mais forte. A existência da parte **não** faz sentido se o todo não existir.
 - as partes não podem ser compartilhadas.
 - Ex: Itens → Pedido
- Relacionamentos do tipo “composição” indicam que se apaguem os objetos associados quando o todo for destruído/finalizado.

Exemplo de agregação e composição

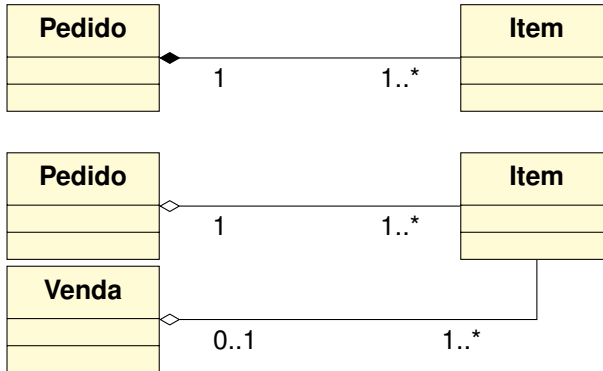
Agregação



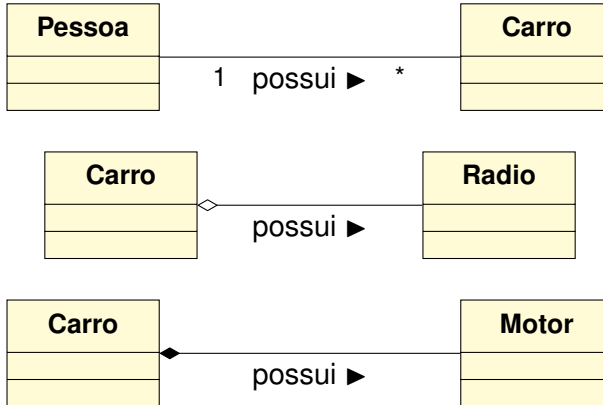
Composição



Exemplo de agregação e composição



Exemplo de associação, agregação e composição



Cardinalidade das associações

- *um-para-um*
 - Ex: um Curso tem um Coordenador
- *um-para-muitos*
 - Ex: um Departamento tem muitos Professores, mas um Professor está alocado a um Departamento apenas
- *muitos-para-muitos*
 - Ex: um professor tem muitos alunos e um aluno tem aulas com vários professores
- A cardinalidade *muitos* pode ser qualificada.
 - Ex: um aluno pode solicitar muitos livros na biblioteca, mas esse número é limitado a no máximo 5 ([0..5])

Multiplicidade na UML

1 – exatamente 1

0..1 – zero ou 1

* – zero ou mais

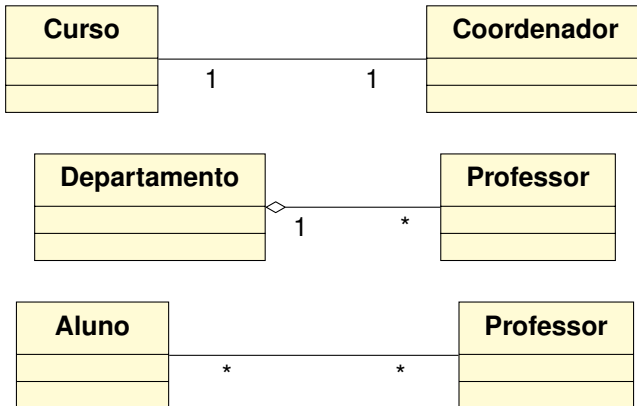
1..* – um ou mais

1..10 – um até dez

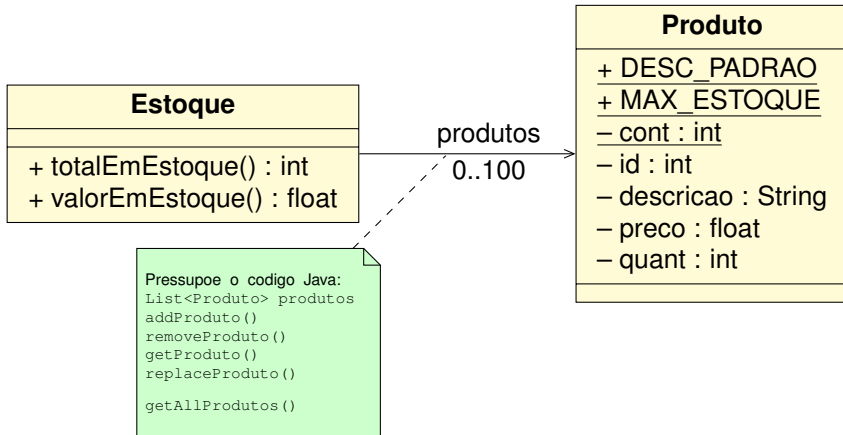
1,10 – um ou dez

0,5..10 – zero ou de 5 a dez

Cardinalidade



Exemplo: Controle de Estoque



Exemplo: Controle de Estoque

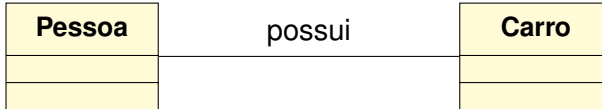
```
public class Estoque {  
    private static final int MAX_PRODUTOS = 100;  
    private Produto[] produtos;  
    private int numProdutos;  
  
    public void addProduto(Produto p) {  
        if (numProdutos < MAX_PRODUTOS) {  
            produtos[numProdutos++] = p;  
        }  
    }  
  
    public void removeProduto(String descricao) {  
        if (numProdutos > 0) {  
            for (int pos = 0; pos < numProdutos; pos++) {  
                if (descricao.equalsIgnoreCase(produtos[pos].getDescricao())) {  
                    // remove produto  
                    for (int i = pos + 1; i < numProdutos; i++)  
                        produtos[i - 1] = produtos[i];  
                }  
            }  
        }  
    }  
    ...  
}
```

Exemplo: Controle de Estoque

```
public int totalEmEstoque() {  
    int total = 0;  
    for (int i = 0; i < numProdutos; i++)  
        total += produtos[i].getQuant();  
    return total;  
}  
  
public float valorEmEstoque() {  
    float valor = 0;  
    for (int i = 0; i < numProdutos; i++)  
        valor += produtos[i].getQuant() * produtos[i].getPreco();  
    return valor;  
}  
  
public String exibirEstoque()  
{  
    StringBuilder valor = new StringBuilder();  
    for (int i = 0; i < numProdutos; i++)  
        valor.append("Produto: " + produtos[i].getId()  
            + "- " + produtos[i].getDescricao()  
            + " Preço: " + produtos[i].getPreco()  
            + " Quant.: " + produtos[i].getQuant() + "\n");  
    return valor.toString();  
}  
  
public Estoque() {  
    produtos = new Produto[MAX_PRODUTOS];  
    numProdutos = 0;  
}  
}
```


Associação e papeis

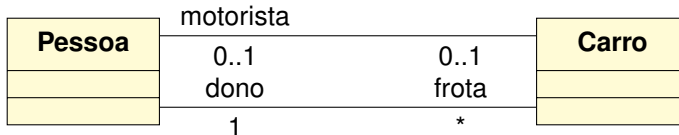
Nome de associação:



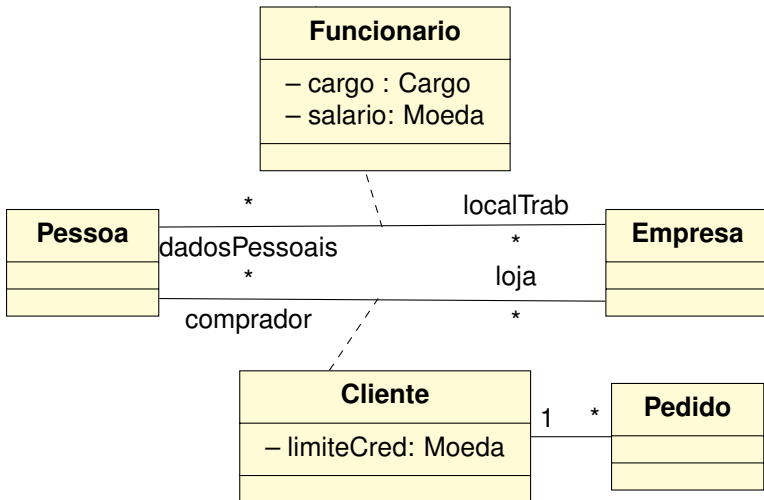
Nomes de papeis da associação:



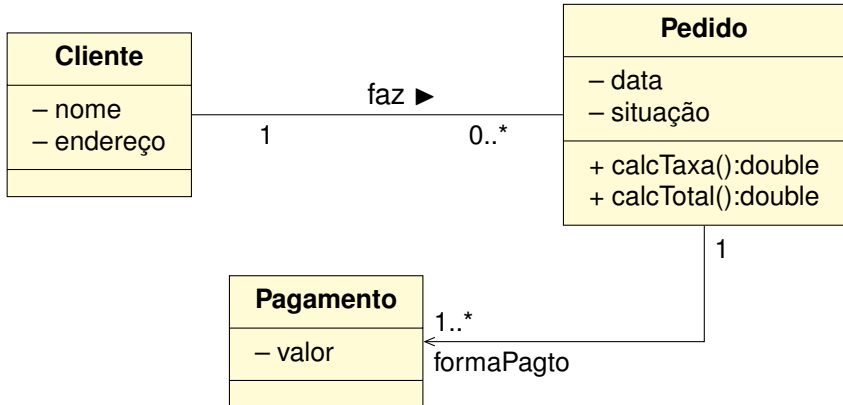
Múltiplos papeis:



Múltiplos papéis como classes de associação



Navegabilidade



Implementação de associações

- 1 – atributo simples (não pode ser `null`)
- 0..1 – atributo simples (pode ser `null`)
 - * – estruturas de dados ou coleções (listas, filas, pilhas, sets)

Navegabilidade e atributos

Associações unidirecionais sugerem a implementação de atributos apenas na origem. Mas alguns casos exigem uma análise mais aprofundada.



Operações de associações

Tipo	1	0..1	*
get	getPapel():Objeto	getPapel():Objeto	getPapel():Set
add	NSA	addPapel(obj)	addPapel(obj)
remove	NSA	removePapel(obj)	removePapel(obj)
replace	replacePapel(obj)	replacePapel(obj)	replacePapel(antigo, novo)
Tipo	* (com classe de associação)		
get	getPapel():Set getClasseDeAssociacao():Set getClasseDeAssociacao(obj):obj		
add	addPapel(obj)		
remove	removePapel(obj)		
remove	removeClasseDeAssociacao(obj)		
replace	replacePapel(antigo, novo)		

Exemplo: Associação unidirecional para 1



```
class Carro {
    private Pessoa dono;

    public Carro(Pessoa dono) {
        this.dono = dono;
    }

    public Pessoa getDono() {
        return this.dono;
    }

    public void replaceDono(Pessoa novoDono) {
        this.dono = novoDono;
    }
}
```

Exemplo: Associação unidirecional para 0..1



```
class Carro {  
    private Pessoa dono;  
  
    public Carro(Pessoa dono) {  
        this.dono = null;  
    }  
  
    public Pessoa getDono() {  
        return this.dono;  
    }  
}
```

```
    public Pessoa addDono(Pessoa novo) {  
        if (this.dono == null)  
            this.dono = novo;  
    }  
  
    public removeDono() {  
        this.dono = null;  
    }  
  
    public void replaceDono(Pessoa novo){  
        this.dono = novo;  
    }  
}
```

Associações reflexivas

- Uma classe pode estar associada a si mesma.

