

Relacionamento Entre Objetos



Prof. Bruno Gomes
bruno.gomes@ifrn.edu.br

Programação Orientada a Objetos

Introdução

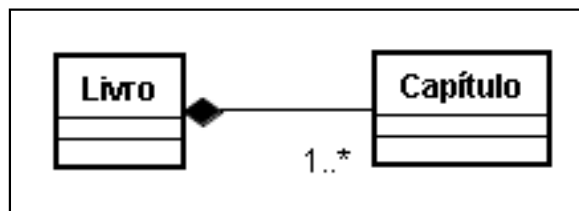
- Objetos não existem isolados
 - São **formados por outros objetos**
 - Objetos **usam outros objetos**
 - Um programa OO possui vários objetos que **interagem entre si**
 - Modelagem define quais objetos usamos em um programa

Introdução

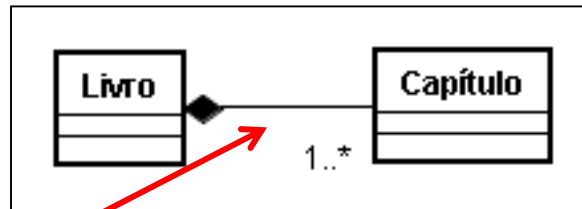
- Objetos possuem relacionamentos
 - Composição
 - Um objeto pode **ser formado por outros** objetos
 - Casa, livro, jardim, agenda de contatos, etc
 - Agregação
 - Um objeto pode **conter outros objetos**
 - Carro (motor, pneu, porta)
 - Associação
 - Objetos podem **usar outros objetos**
 - Trem usa estrada de ferro

Composição

- Um livro é composto de capítulos
 - Capítulo é parte essencial de livro
 - Se não existir capítulo, não existe livro
 - Capítulo não existe fora de livro
- Linha com losângulo preenchido na classe “dominante”
 - Livro é composto de 1 ou mais capítulos



Composição



Os atributos são derivados dos relacionamentos. Não existem no diagrama

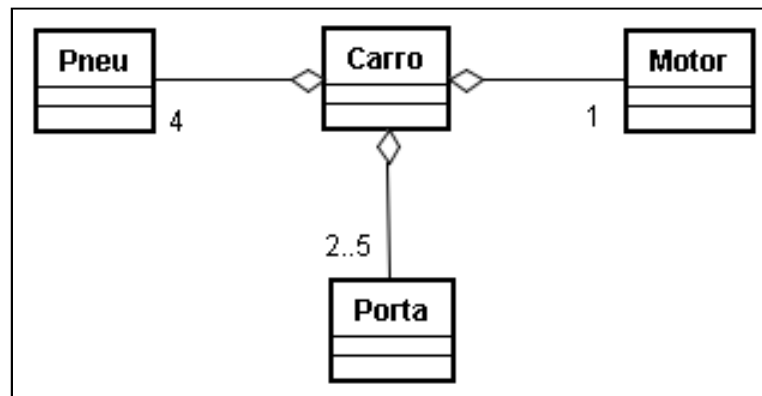
```
public class Livro {  
  
    private Capítulo[] capitulos;  
  
    public Livro(int qtdCapitulo){  
        capitulos = new Capítulo[qtdCapitulo];  
    }  
}
```

```
public class Capítulo {  
  
    private Livro livro;  
  
    /* Definição da classe Capítulo */  
}
```

Referência pode ou não ser bidirecional. Capítulo não precisa ter o atributo livro

Agregação

- Carro possui Pneu, Motor e Porta
 - Não são partes essenciais do carro
 - Retirando as portas um carro continua sendo um carro
 - Pneus/portas existem como objetos independentes
- Linha com losângulo vazio na classe “dominante”

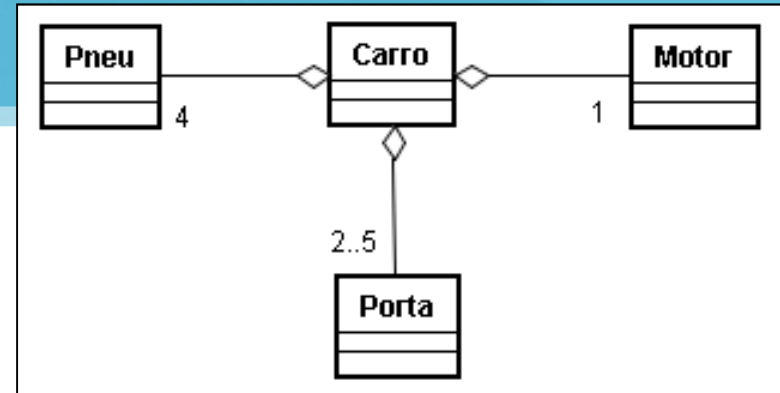


Agregação

```
public class Carro {  
  
    private Motor motor;  
    private Porta portas[];  
    private Pneu pneus[];  
    /* ... */  
}
```

Pode ser implementado
de mais de uma forma

```
public class Carro {  
  
    private Motor motor;  
    private Porta portas[];  
    private Pneu p1, p2, p3, p4;  
    /* ... */  
}
```



```
public class Motor {  
    /* ... */  
}
```

```
public class Pneu{  
    /* ... */  
}
```

```
public class Porta{  
    /* ... */  
}
```

Associação

- Objetos que usam outros objetos
 - Podem ser implementados como atributos



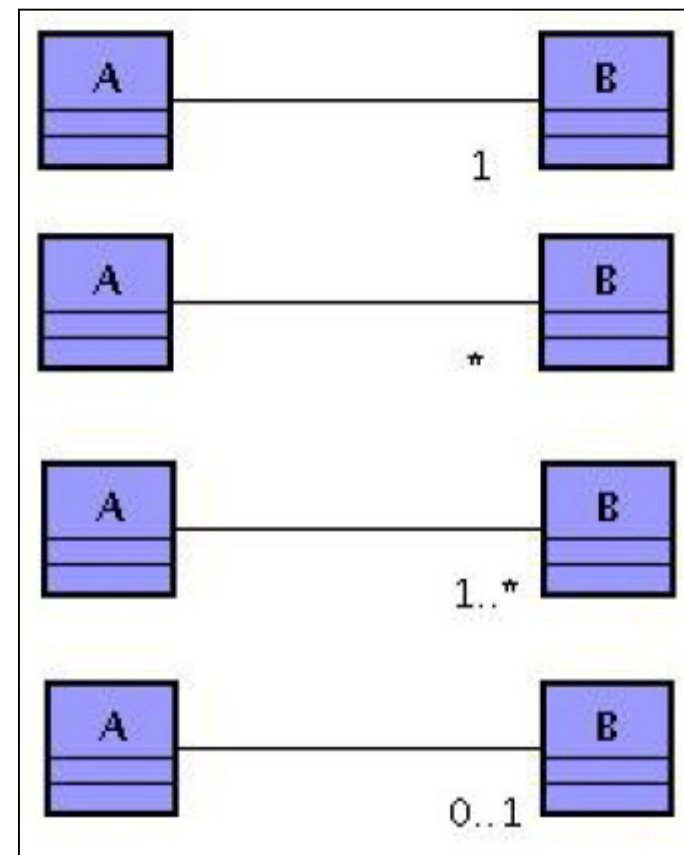
```
public class Trem {
    private EstradaDeFerro estradaDeFerro;
    /* ... */
    public void definirEstrada(EstradaDeFerro estradaDeFerro){
        this.estradaDeFerro = estradaDeFerro;
    }
}
```

Observações

- Composição, Agregação e Associação
 - Mesma forma de implementar
 - Muda apenas o **conceito**
 - Comportamento diferente
 - Muito comum usar apenas notação da associação
 - Sem o losângulo
 - Composições e Agregações são “tipos” de associações
 - Representam relacionamento “**tem um**”
 - Carro “tem uma” roda
 - Livro “tem um” capítulo
 - Trem “tem uma” estrada de ferro

Multiplicidade

- Indica quantidade de objetos referenciados
- Principais:
 - A possui exatamente 1 B
 - A possui vários B
 - A possui 1 ou mais B
 - A possui 0 ou 1 B



Nomes

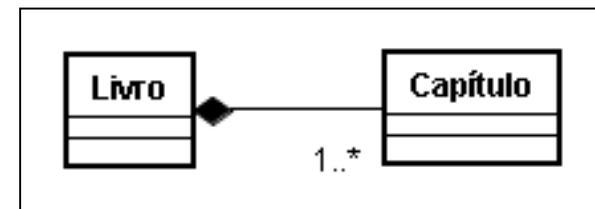
- Pode-se especificar o nome do atributo
 - Obrigar existência do atributo
 - Carro tem um atributo privado **motor** do tipo **Motor**

```
public class Carro {  
    private Motor motor;  
    /* ... */  
}
```

Nomes

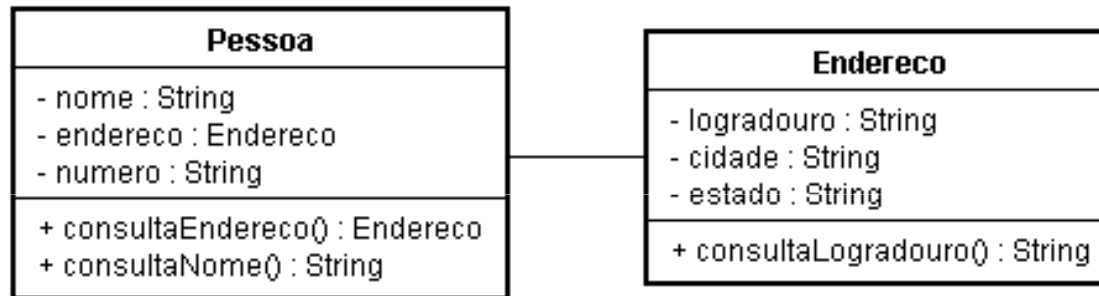
- Coleções
 - Atributos com multiplicidade * **podem ser** implementados de mais de uma forma
 - Array é uma delas

```
public class Livro {  
    private Capitulo cap[];  
}
```



Exercício 1

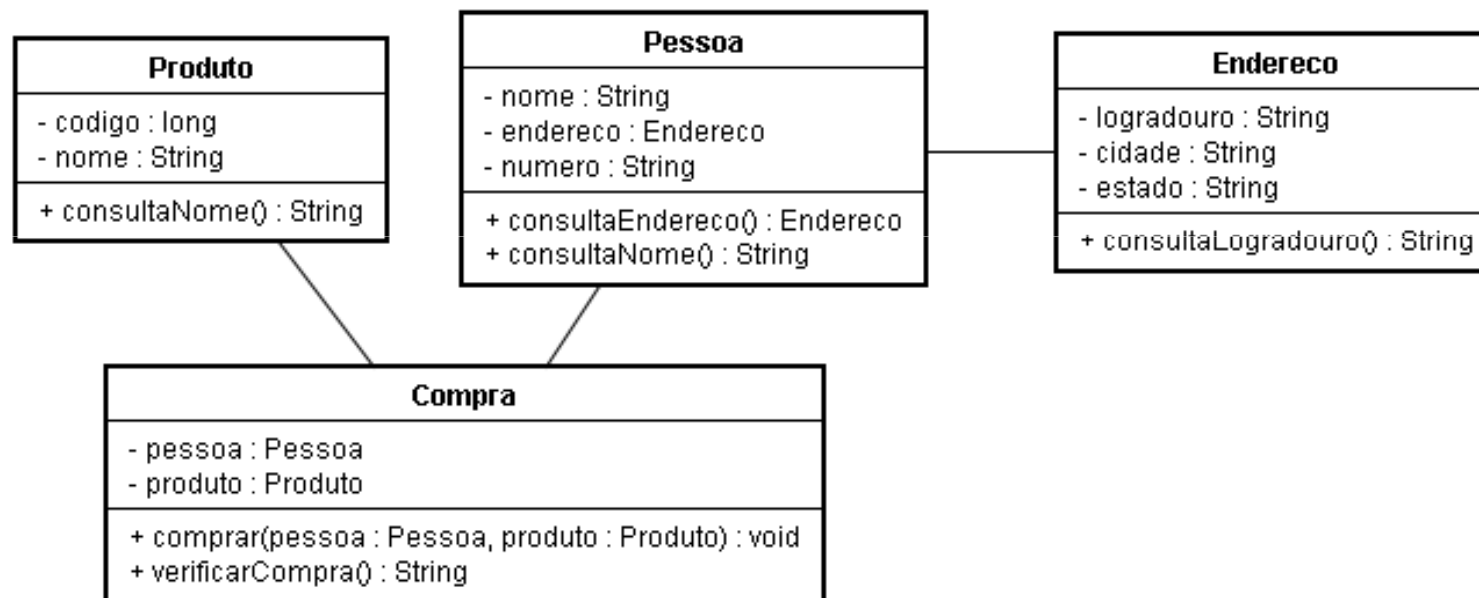
- Desenvolver o seguinte relacionamento:



- Pessoa** vai ter um objeto do tipo **Endereco**
- Classes com construtores parametrizados

Exercício 2

- Adicionar as seguintes classes ao relacionamento:



- Detalhes nos próximos Slides

Exercício 2

- A Classe **Compra** vai conter um objeto do tipo **Pessoa** e outro do tipo **Produto**
- É possível comprar acessando o método comprar, passando como parâmetro dois objetos, um do tipo **Pessoa** e outro do tipo **Produto**

Exercício 2

- Desenvolva uma Classe de teste:
 - Inicialmente, a classe deve criar 2 objetos do tipo **Produto** (crie os objetos com as informações que desejar)
 - O usuário faz um cadastro (criando um objeto do tipo **Pessoa**)
 - Logo após, o usuário seleciona entre os 2 produtos cadastrados anteriormente
 - Dependendo da escolha, é acessado o método comprar da classe **Compra**, e passado como parâmetro o objeto **Pessoa** que ele cadastrou e o do **Produto** escolhido
 - Logo após, é exibido no console ao usuário uma mensagem de confirmação, exibindo o nome dele e do produto comprado (acesso ao método verificarCompra()) e finaliza a aplicação

Trabalho de Pesquisa

- Pesquisar qual a finalidade, como utilizar e quais as principais diferenças entre as seguintes Classes:
 - ArrayList
 - Vector
 - HashMap