Programação Orientada a Objeto

10 - Relacionamentos

- Relacionamentos:
  - Dependência;
  - Associação Simples;
  - Classe de Associação;
  - Agrupamento;
  - Composição;

Dr. Márcia L. Aguena Castro
marcia@39dev.com

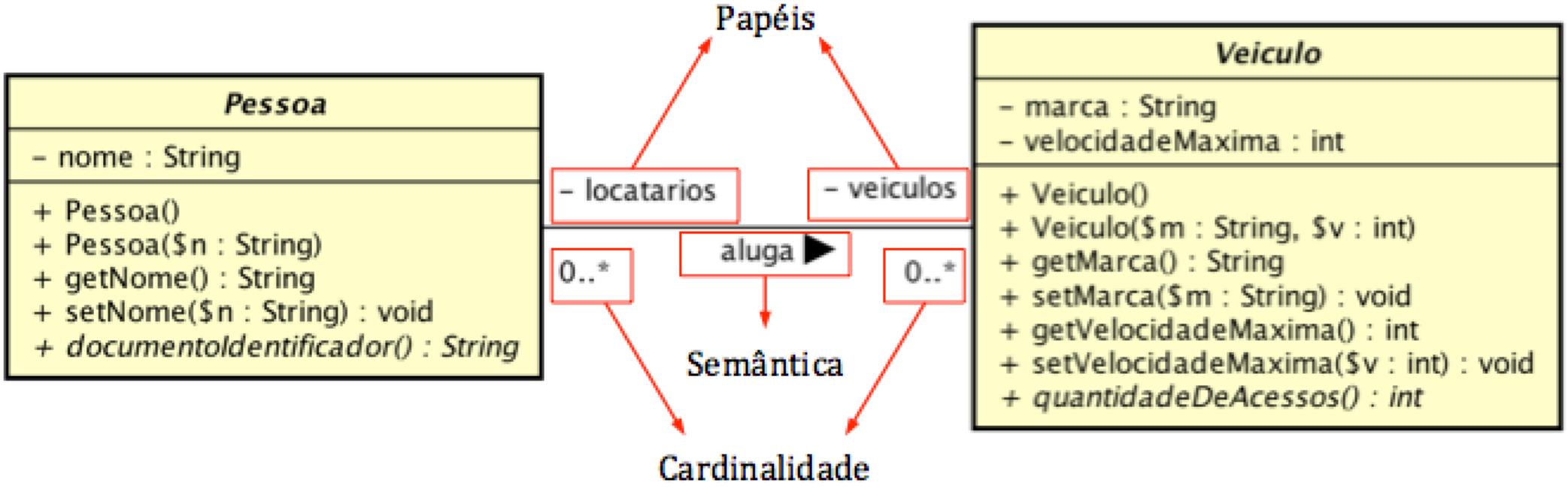
Me. Reinaldo de O. Castro
reinaldo@39dev.com

Relacionamentos

- Um relacionamento **conecta** duas ou mais classes;
- Um relacionamento define a **semântica** entre duas ou mais classes;
- Por meio de relacionamentos que as classes trabalham **colaborativamente** para resolver um problema;
- Um relacionamento pode ser **unidirecional** ou **bidirecional**;
- Um relacionamento sempre possui uma **cardinalidade** (implícita ou explícita);
- Um relacionamento pode definir os **papéis** de cada classe participante desse mesmo relacionamento;

Relacionamentos

- Representação em **UML**:



Relacionamentos

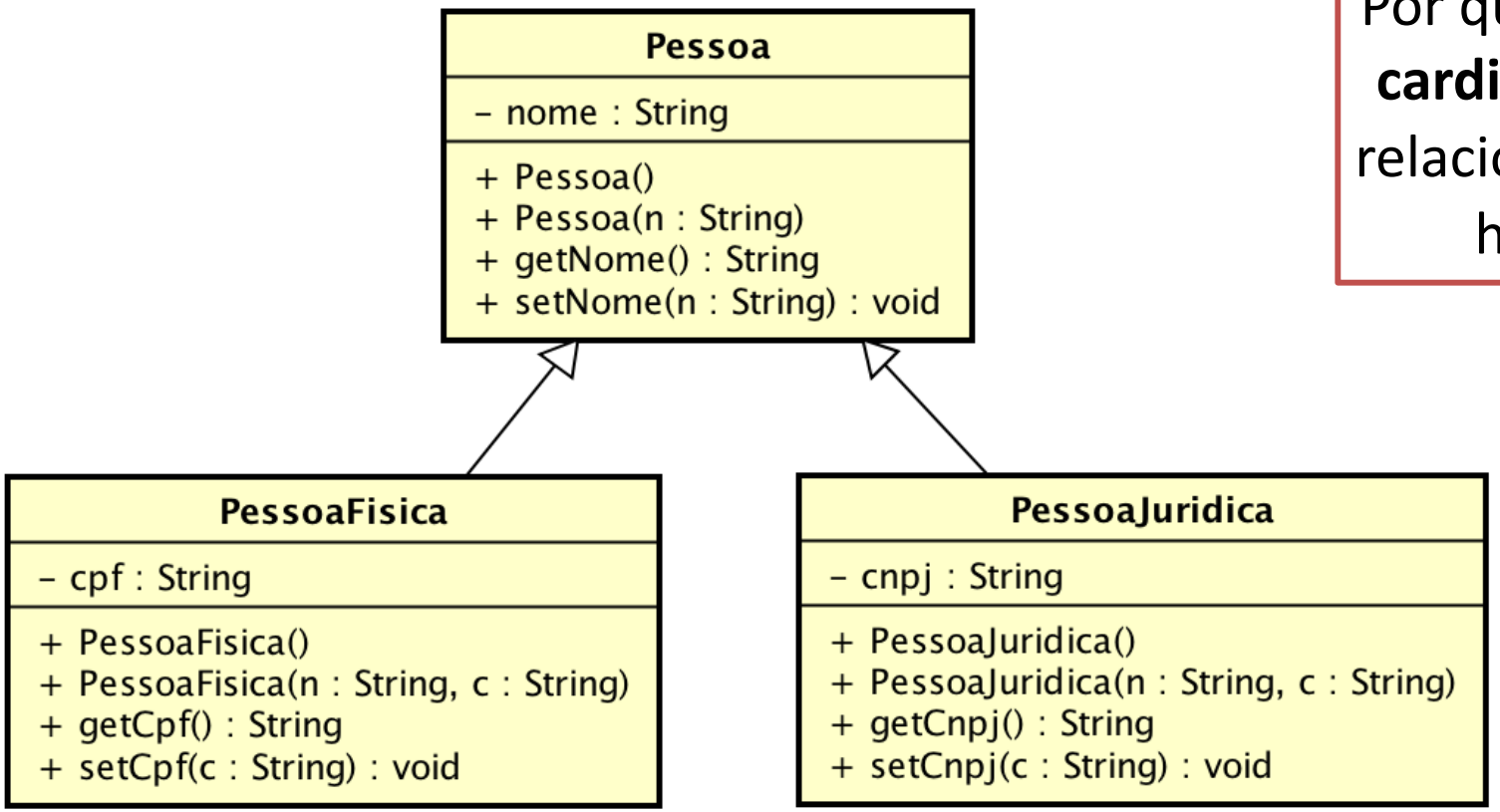
- Os relacionamentos descritos pela UML e implementáveis em Java são:
  - Generalização (Herança);
  - Realização (Interface).
  - Associação;
  - Dependência;
- Os conceitos são muitas vezes subjetivos e há várias interpretações teóricas diferentes sobre o assunto.
- O importante é entender o problema e saber implementá-lo.

Generalização ou Herança

- Herança até agora foi abordada como um conceito de OO, mas especificamente trata-se de um tipo de relacionamento entre duas classes chamado **generalização**.
- Relembrando:
  - É expressa em uma semântica do tipo "**é um**"/ "**é uma**";
  - Se dá por meio da palavra-chave `extends`;
  - Sua representação em um diagrama de classes é feito por uma **linha contínua com um triângulo apontado** para a superclasse;

Generalização ou Herança

- Representação em **UML**:



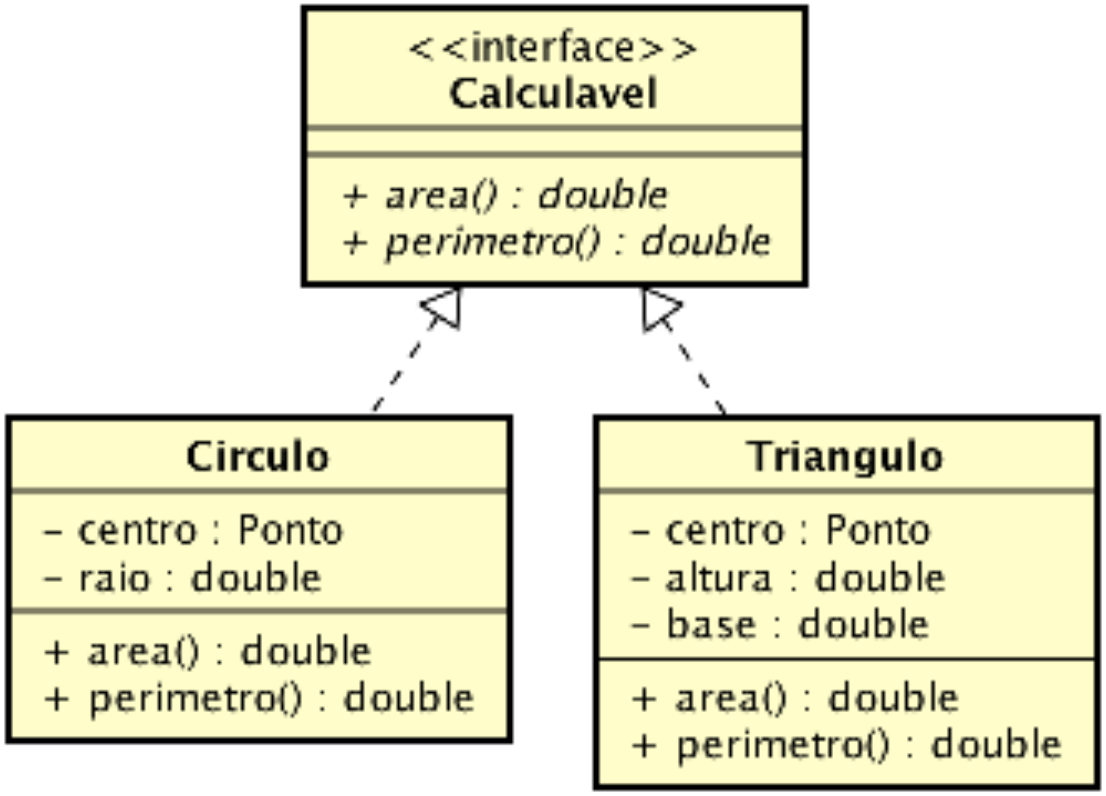
Por que não existe **cardinalidade** no relacionamento de herança?

Realização ou Interface

- Assim como a Herança, a **Interface** até agora foi abordada como um conceito de OO, mas especificamente trata-se de um tipo de relacionamento entre duas classes chamado **Realização**.
- Relembrando:
  - É expressa em uma semântica do tipo "**Assina um contrato de**";
  - Se dá por meio da palavra-chave `implements`;
  - Sua representação em um diagrama de classes é feito por uma **linha tracejada com um triângulo apontado** para a interface;

Realização ou Interface

- Representação em **UML**:



Dependência

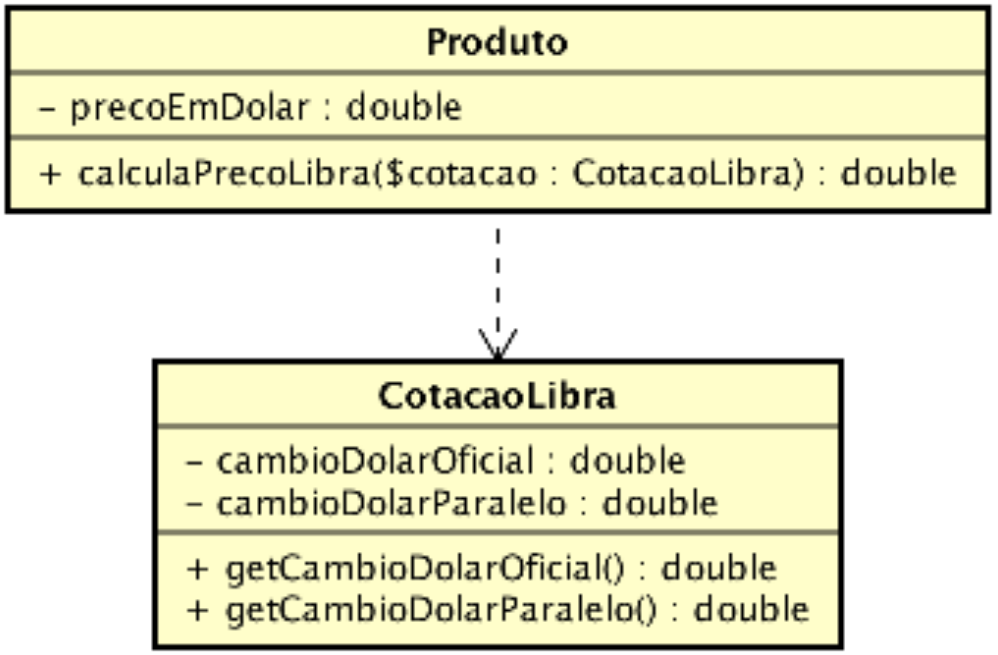
- A **dependência** acontece quando uma **alteração em uma classe afeta a outra**;
- O inverso não é verdade;
- Diz-se então que uma classe utiliza a outra como **argumento em sua assinatura**.
- É expressa em uma semântica "**usa**".
- A dependência entre classes indica que os objetos de uma classe **usam serviços** dos objetos de outra classe;

Dependência

- Na implementação não existe palavra chave para indicar uma dependência, ela ocorre quando uma classe possui um **método ou operação com a classe que é utilizada como um parâmetro ou valor de retorno** para essa operação.
- É representada por uma **linha tracejada e uma seta aberta na ponta**.

Dependência

- Representação em **UML**:




```
1  public class Produto{
2    private double precoEmDolar;
3
4    public double calculaPrecoLibra(CotacaoLibra $cotacao){
5      return this.precoEmDolar * $cotacao.cambioDolarParalelo;
6    }
7  }
```

Dependência

- Representação em **UML**:

```
1  package aula2G.Relacionamentos.Dependencia;
2
3  public class TestaDependencia {
4      public static void main(String[] args){
5      Produto p = new Produto(10.00);
6      CotacaoLibra c = new CotacaoLibra();
7      System.out.println(p.calculaPrecoLibra(c));
8      }
9  }
```

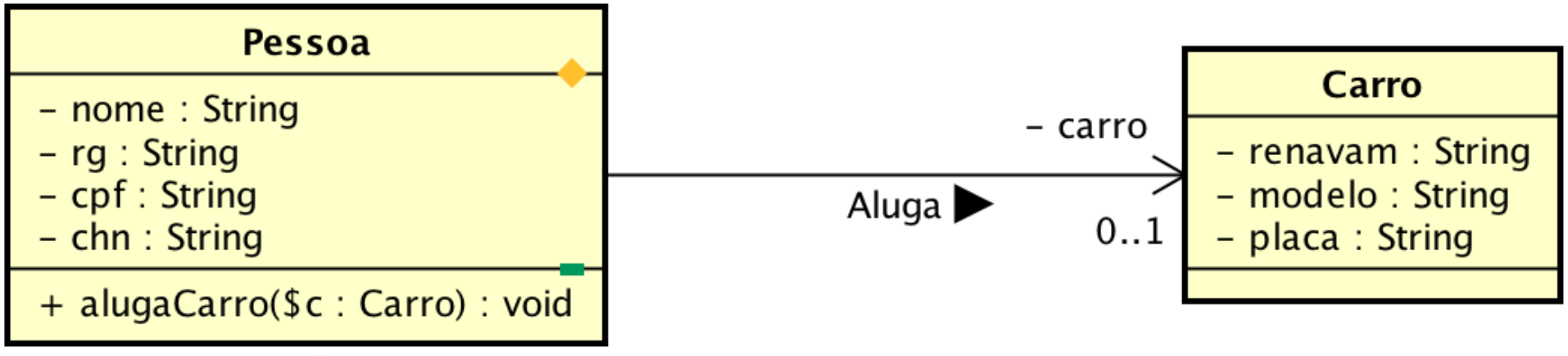
```
1  package aula2G.Relacionamentos.Dependencia;
2
3  public class TestaDependencia {
4      public static void main(String[] args){
5      Produto p = new Produto(10.00);
6      CotacaoLibra c = new CotacaoLibra();
7      System.out.println(p.calculaPrecoLibra(c));
8      }
9  }
```

Associação

- **Associação** é o relacionamento mais comum em OO;
- Uma classe tem **objeto como atributo**, mas a existência desse atributo-objeto é **independente** em relação a essa classe;
- Diz-se que o objeto da associação pode **enviar uma mensagem** (chamar um método público) ao objeto associado.
- Não existe palavra-chave para indicar a associação.
- É representada por uma **linha sólida entre duas ou mais classes**.
- Pode ser **uni** ou **bidirecional**, e a seta mostra o sentido.

Associação Unidirecional

- Representação em **UML**:



Associação Unidirecional

```
3  public class Pessoa{
4    private String nome;
5    private String rg;
6    private String cpf;
7    private String cnh;
8    private Carro carro;
9
10   public Pessoa(){}
11   public Pessoa(String $nome){
12     this.nome = $nome;
13   }
14   public String getNome(){
15     return this.nome;
16   }
17   public void setCarro(Carro $c){
18     this.carro=$c;
19   }
20   public void alugaUmCarro(Carro $c){
21     this.setCarro($c);
22   }
23   public String toString(){
24     return this.nome + " esta alugando " + this.carro.getPlaca();
25   }
26 }
```

- Implementação da Classe Pessoa - **Unidirecional**.

Associação Unidirecional

```
3  public class Carro{
4    private String renavam;
5    private String modelo;
6    private String placa;
7
8    public Carro(){}
9    public Carro(String $placa){
10     this.placa = $placa;
11   }
12   public String getPlaca(){
13     return this.placa;
14   }
15 }
```

- Implementação da Classe `Carro` - **Unidirecional**.

Associação Unidirecional

- Classe `TestaAssociacao` e saída de Tela - **Unidirecional**.

```
3  public class TestaAssociacao{
4    public static void main (String[] args){
5      Pessoa pessoa1  = new Pessoa("Marcos");
6      Carro  carro    = new Carro("TLP0000");
7
8      pessoa1.alugaUmCarro(carro);
9
10     System.out.println(pessoa1.toString());
11   }
12 }
```

```
Marcos esta alugando TLP0000
```

Associação Bidirecional

```
3  public class Pessoa{
4    private String nome;
5    private String rg;
6    private String cpf;
7    private String cnh;
8    private Carro carro;
9
10   public Pessoa(){}
11   public Pessoa(String $nome){
12     this.nome = $nome;
13   }
14   public String getNome(){
15     return this.nome;
16   }
17   public void setCarro(Carro $c){
18     this.carro=$c;
19   }
20   public void alugaUmCarro(Carro $c){
21     this.setCarro($c);
22   }
23   public String toString(){
24     return this.nome + " esta alugando " + this.carro.getPlaca();
25   }
26 }
```

- Implementação da clase Pessoa - **Bidirecional.**

Associação Bidirecional

```
3  public class Carro{
4    private String renavam;
5    private String modelo;
6    private String placa;
7    private Pessoa locatario;
8
9    public Carro(){}
10   public Carro(String $placa){
11     this.placa = $placa;
12   }
13   public String getPlaca(){
14     return this.placa;
15   }
16   public void eAlugado(Pessoa $locatario){
17       this.locatario = $locatario;
18   }
19   public String toString(){
20       return this.placa + " está alugado para " + this.locatario.getNome();
21   }
22 }
```

- Implementação da clase `Carro` - **Bidirecional.**

Associação Bidirecional

- Classe `TestaAssociacao` e saída de Tela - **Bidirecional**.

```
3  public class TestaAssociacao{
4    public static void main (String[] args){
5      Pessoa pessoa1  = new Pessoa("Marcos");
6      Carro  carro    = new Carro("TLP0000");
7
8      pessoa1.alugaUmCarro(carro);
9      carro.eAlugado(pessoa1);
10
11     System.out.println(pessoa1);
12     System.out.println(carro);
13   }
14 }
```

```
Marcos esta alugando TLP0000
TLP0000 está alugado para Marcos
```

Associação Bidirecional

- Uma associação bidirecional, depende de duas ações:
  - Associar a primeira classe à segunda;
  - Associar a segunda classe à primeira;
- Se essas duas ações não estiverem coordenada podem ocorrer problemas.

Associação Bidirecional

```
3  public class TestaAssociacaoErro{
4    public static void main (String[] args){
5      Pessoa pessoa1  = new Pessoa("Marcos");
6      Carro  carro    = new Carro("TLP0000");
7
8      pessoa1.alugaUmCarro(carro);
9      carro.eAlugado(pessoa1);
10
11     System.out.println(pessoa1);
12     System.out.println(carro);
13
14     Pessoa pessoa2 = new Pessoa("voSalvelina");
15     pessoa2.alugaUmCarro(carro);
16
17     System.out.println(pessoa2);
18     System.out.println(carro);
19   }
20 }
```

```
Marcos esta alugando TLP0000
TLP0000 está alugado para Marcos
voSalvelina esta alugando TLP0000
TLP0000 está alugado para Marcos
```

Associação Bidirecional

- **Nunca** crie um relacionamento **bidirecional** sem uma real necessidade.
- Um relacionamento bidirecional necessita que ambas as classes que se relacionem, estejam apontadas entre si.
- O problema se agrava se o relacionamento for 1..N ou N..M.
- Se precisar de um relacionamento bidirecional, faça com que **ambas as classes criem as duas partes do relacionamento** (mesmo que isso implique em redundância de código).

Associação Bidirecional

```
3  public class Carro{
4    private String renavam;
5    private String modelo;
6    private String placa;
7    private Pessoa locatario;
8
9    public Carro(){}
10   public Carro(String $placa){
11     this.placa = $placa;
12   }
13   public String getPlaca(){
14     return this.placa;
15   }
16   public void setLocatario(Pessoa $l){
17       this.locatario = $c;
18   }
19   public void eAlugado(Pessoa $locatario){
20       this.setLocatario($locatario);
21       $locatario.setCarro(this);
22   }
23   public String toString(){
24       return this.placa + " está alugado para " + this.locatario.getNome();
25   }
26 }
```

Associação Bidirecional

```
3  public class Pessoa{
4    private String nome;
5    private String rg;
6    private String cpf;
7    private String cnh;
8    private Carro carro;
9
10   public Pessoa(){}
11   public Pessoa(String $nome){
12     this.nome = $nome;
13   }
14   public String getNome(){
15     return this.nome;
16   }
17   public void setCarro(Carro $c){
18     this.carro=$c;
19   }
20   public void alugaUmCarro(Carro $c){
21     this.setCarro($c);
22     $c.setLocatario(this);
23   }
24   public String toString(){
25     return this.nome + " esta alugando " + this.carro.getPlaca();
26   }
27 }
```

Associação Bidirecional

```
1  public class TestaAssociacao{
2    public static void main (String[] args){
3      Pessoa pessoa1  = new Pessoa("Marcos");
4      Carro  carro    = new Carro("TLP0000");
5
6      pessoa1.alugaUmCarro(carro);
7
8      System.out.println(pessoa1.toString());
9      System.out.println(carro.toString());
10
11     Pessoa pessoa2 = new Pessoa("voSalvelina");
12     pessoa2.alugaUmCarro(carro);
13
14     System.out.println(pessoa2.toString());
15     System.out.println(carro.toString());
16   }
17 }
```

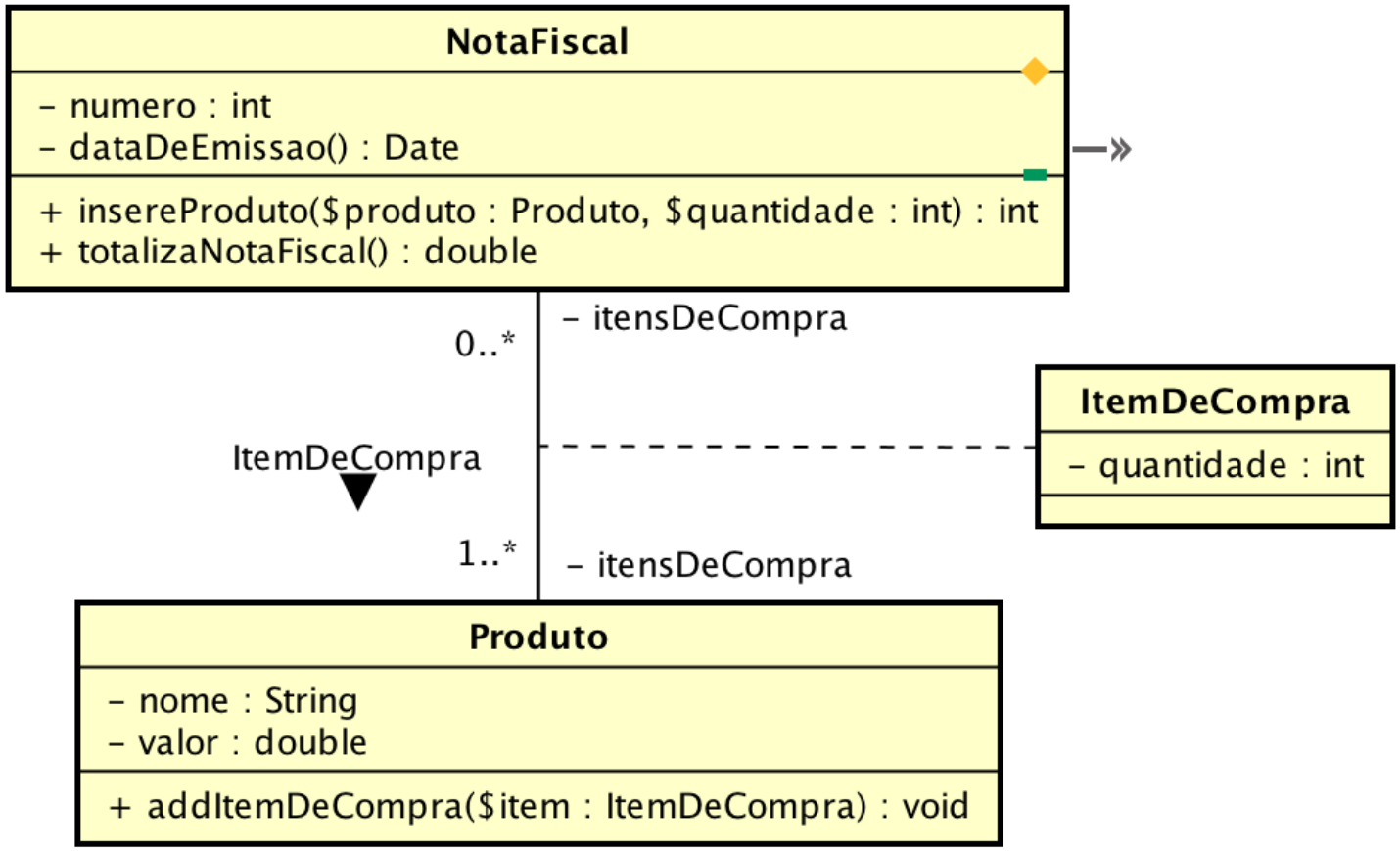
```
Marcos esta alugando TLP0000
TLP0000 está alugado para Marcos
voSalvelina esta alugando TLP0000
TLP0000 está alugado para voSalvelina
```

Classe de Associação

- Uma **classe de Associação** é uma classe que **surge de uma associação**;
- Armazena dados referentes **ao relacionamento** (ou seja, os dados não pertence a nenhuma das duas classes que compõe o relacionamento);
- Tem a mesma semântica de dados armazenados no relacionamento **Diagrama Entidade-Relacionamento**;
- Representada por uma **linha pontilhada** (**que inicia na linha contínua até a classe no final**);
- Na implementação, as classes que compõe o relacionamento **apontam para a classe de associação** e não mais uma para a outra.

Classe de Associação

- Representação **UML**:



Classe de Associação

```
1   import java.util.Collection;
2   import java.util.ArrayList;
3   import java.util.Date;
4   public class NotaFiscal{
5     private int numero;
6     private Date dataDeEmissao;
7     private Collection<ItemDeCompra> itensDeCompra;
8
9     public NotaFiscal(){}
10 ›   public NotaFiscal(int $numero, Date $data){…
```

Classe de Associação

```
15    public void insereProduto(Produto $produto,int $quantidade){
16      //ItemDeCompra
17      ItemDeCompra item = new ItemDeCompra($quantidade,this,$produto);
18      //Produto
19      item.getProduto().addItemDeCompra(item);
20      //NotaFiscal
21      this.itensDeCompra.add(item);
22    }
23    public double totalizaNota(){
24      double total=0;
25      System.out.println("NotaFiscal : " + this.numero);
26      for (ItemDeCompra item: itensDeCompra){
27        total += item.getQuantidade() * item.getProduto().getValor();
28        System.out.println(item.getProduto().getNome() + "-" +
29                           item.getQuantidade() + "-" +
30                           item.getProduto().getValor());
31      }
32      System.out.println("Total: "+ total);
33      return total;
34    }
35  }
```

Classe de Associação

```php
import java.util.Collection;
import java.util.ArrayList;
public class Produto{
  private String nome;
  private double valor;
  private Collection<ItemDeCompra> itensDeCompra;

  public Produto(){}
  public Produto(String $nome,double $valor){
  public double getValor(){
  public String getNome(){
  public void addItemDeCompra(ItemDeCompra $item){
      itensDeCompra.add($item);
  }
}
```

```php
public class ItemDeCompra{
  private int quantidade;
  private NotaFiscal notaFiscal;
  private Produto    produto;

  public ItemDeCompra(){}
  public ItemDeCompra(int $quantidade, NotaFiscal $nota, Produto $produto){
  public Produto getProduto(){
  public int getQuantidade(){
}
```

Classe de Associação

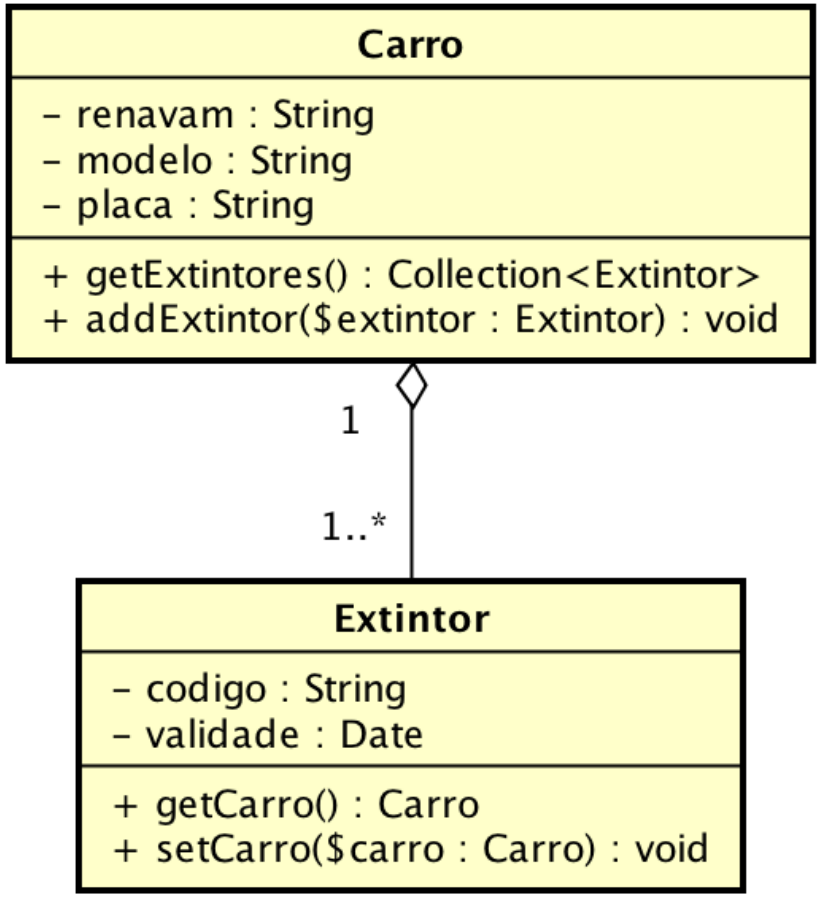
```
1  import java.util.Date;
2  public class TestaNotaFiscal{
3    public static void main(String[] args){
4      Date hoje = new Date();
5      NotaFiscal nota1 = new NotaFiscal(1234,hoje);
6      Produto produto1 = new Produto("lápis",1.50);
7      Produto produto2 = new Produto("borracha",2.00);
8      nota1.insereProduto(produto1,3);
9      nota1.insereProduto(produto2,1);
10     nota1.totalizaNota();
11   }
12 }
```

Agregação

- A agregação representa um relacionamento **todo** / **parte**, ex: veículo e motor;
- A semântica da agregação é um relacionamento "**tem um**" / "**tem uma**"
- O objeto todo **NÃO É** responsável pelo **ciclo de vida do objeto parte**; ex: remover um veículo do sistema não implica necessariamente em remover seu respectivo motor;
- Não existe nenhuma palavra-chave para indicar uma agregação;
- Representada por uma **linha contínua e um losango vazio** na classe todo;
- Em termos de implementação é **idêntico à associação**.

Agregação

- Representação **UML**:



Agregação

```
1  import java.util.Collection;
2  import java.util.ArrayList;
3  public class Carro{
4    private String renavam;
5    private String modelo;
6    private String placa;
7    private Pessoa locatario;
8    private Collection<Extintor> extintores;
9
10   public Carro(){
11     extintores = new ArrayList<>();
12   }
13   public Carro(String $placa){
14     this.placa = $placa;
15     extintores = new ArrayList<>();
16   }
```

Agregação

```
17 ›    public String getPlaca(){
20 ›    public void setLocatario(Pessoa $p){
23 ›    public String toString(){
26      public Collection<Extintor> getExtintores(){
27        return this.extintores;
28      }
29      public void addExtintor(Extintor $extintores){
30        this.extintores.add($extintores);
31      }
32  }
```

Agregação

```
1   import java.util.Date;
2   public class Extintor {
3     private String codigo;
4     private Date validade;
5     private Carro carro;
6
7     public Extintor(){}
8 ›   public Extintor(String $codigo){…
11 ›  public Carro getCarro(){…
14 ›  public void setCarro(Carro $carro){…
17  }
```

Agregação

```
1  public class TestaCarroExtintor{
2    public static void main(String[] args){
3      Carro carro1 = new Carro("TLP0000");
4
5      Extintor extintor1 = new Extintor("velhobom");
6      Extintor extintor2 = new Extintor("novoruim");
7
8      carro1.addExtintor(extintor1);
9      carro1.addExtintor(extintor2);
10
11     System.out.println(carro1.getExtintores());
12   }
13 }
```

Composição

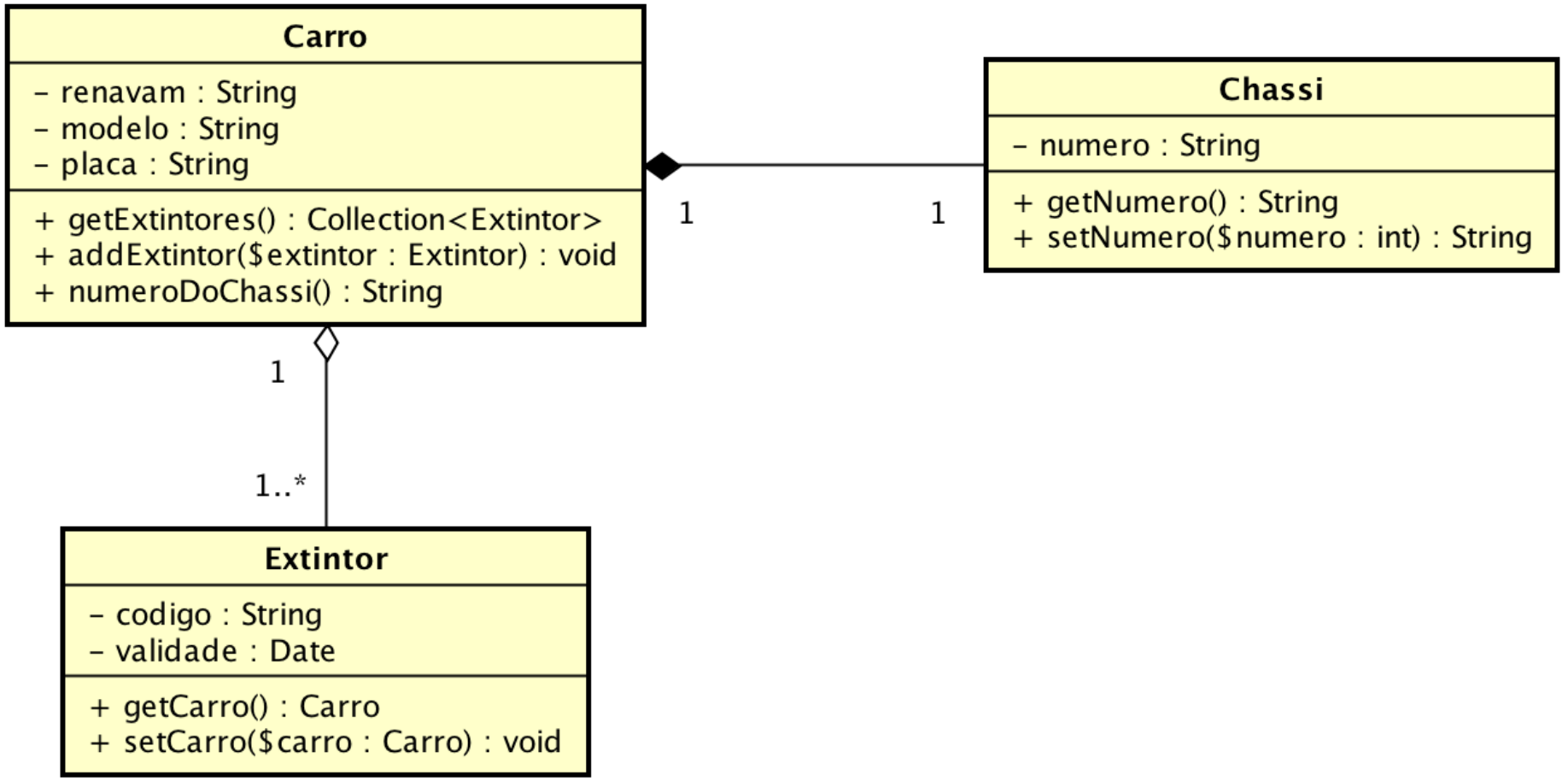
- A **composição** também representa um relacionamento **todo** / **parte**, ex: veículo e motor;
- Também demonstra um relacionamento "**tem um**" / "**tem uma**";
- O objeto todo **É responsável** pelo **ciclo de vida** do objeto parte; ex: remover um veículo do sistema implica em remover seu respectivo motor;
- Não existe nenhuma palavra-chave para indicar uma composição;
- Representada por uma **linha contínua e um losango preenchido** na classe todo.

Composição

- **Requer cuidados especiais** na implementação:
  - Geralmente o objeto todo **cria internamente** o objeto parte;
  - Nenhum método do objeto todo retorna **referências** do objeto parte, apenas informações sobre o objeto parte;
  - A **cardinalidade** entre o todo e a parte será **1:1** ou **1:N**.

Composição

- Representação **UML**:



Composição

```
 3  public class Carro{


 9    private Chassi chassi;
10
11    public Carro(String $placa, String $numeroChassi){
12      this.placa = $placa;
13      extintores = new ArrayList<>();
14      chassi = new Chassi($numeroChassi);
15    }


31    public String numeroDoChassi(){
32      return this.chassi.getNumero();
33    }
34  }
```

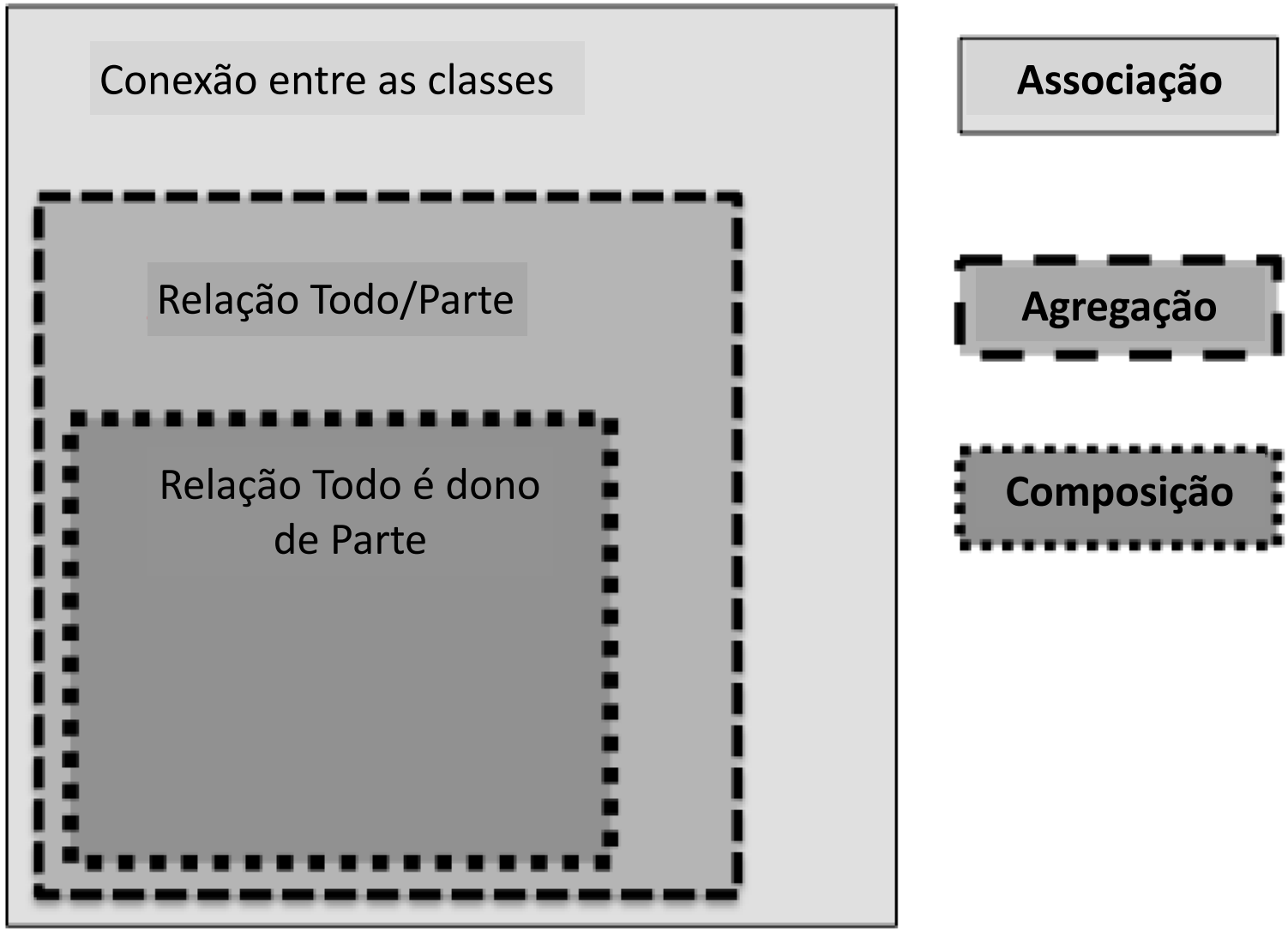
Composição

```
1  public class Chassi{
2    private String numero;
3
4    public Chassi(){}
5    public Chassi(String $numero){
6      this.numero = $numero;
7    }
8    public String getNumero(){
9      return this.numero;
10   }
11   public void setNumero(String $numero){
12     this.numero = $numero;
13   }
14
15 }
```

Composição

```
1  public class TestaCarroChassi{
2    public static void main(String[] args){
3      Carro carro1 = new Carro("TLP0000","000111");
4
5      Extintor extintor1 = new Extintor("velhobom");
6      Extintor extintor2 = new Extintor("novoruim");
7
8      carro1.addExtintor(extintor1);
9      carro1.addExtintor(extintor2);
10
11     System.out.println(carro1.getExtintores());
12     System.out.println(carro1.numeroDoChassi());
13   }
14 }
```

Associação - Agregação - Composição

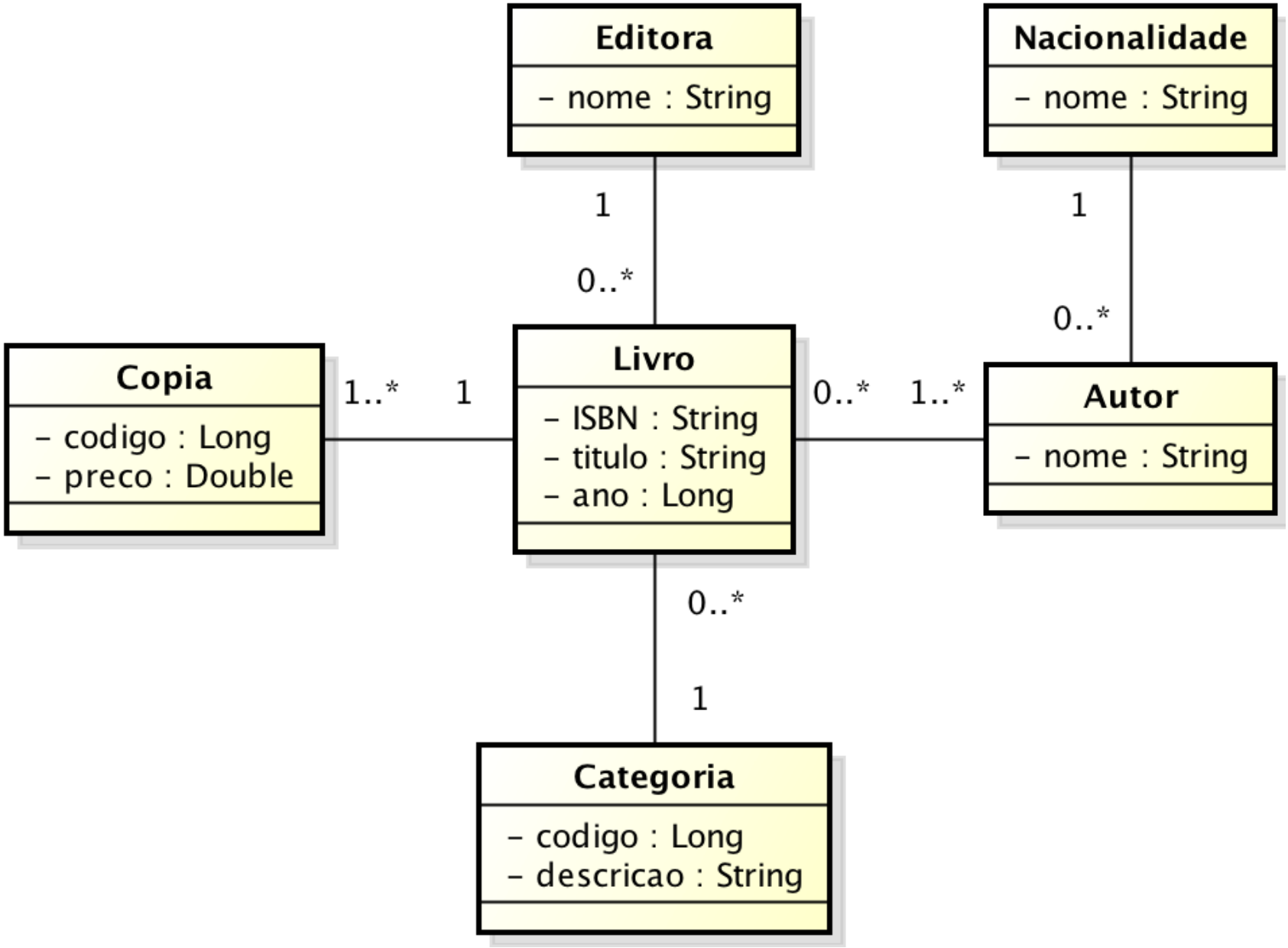


Exercícios - Relacionamentos

Exercício 1 - Leia a descrição, observe o diagrama de classes e implemente o código em Java.

- Uma biblioteca deseja manter informações sobre seus livros.
- Inicialmente, quer armazenar para os livros as seguintes características: ISBN, título, ano, editora e autores deste livro.
- Para os autores, deseja manter: nome e nacionalidade.
- Um autor pode ter vários livros, assim como um livro pode ser escrito por
- vários autores.
- Cada livro pertence a uma categoria, que possuem as informações: código da categoria e descrição.
- Uma categoria pode ter vários livros associados a ela.

Exercícios - Relacionamentos

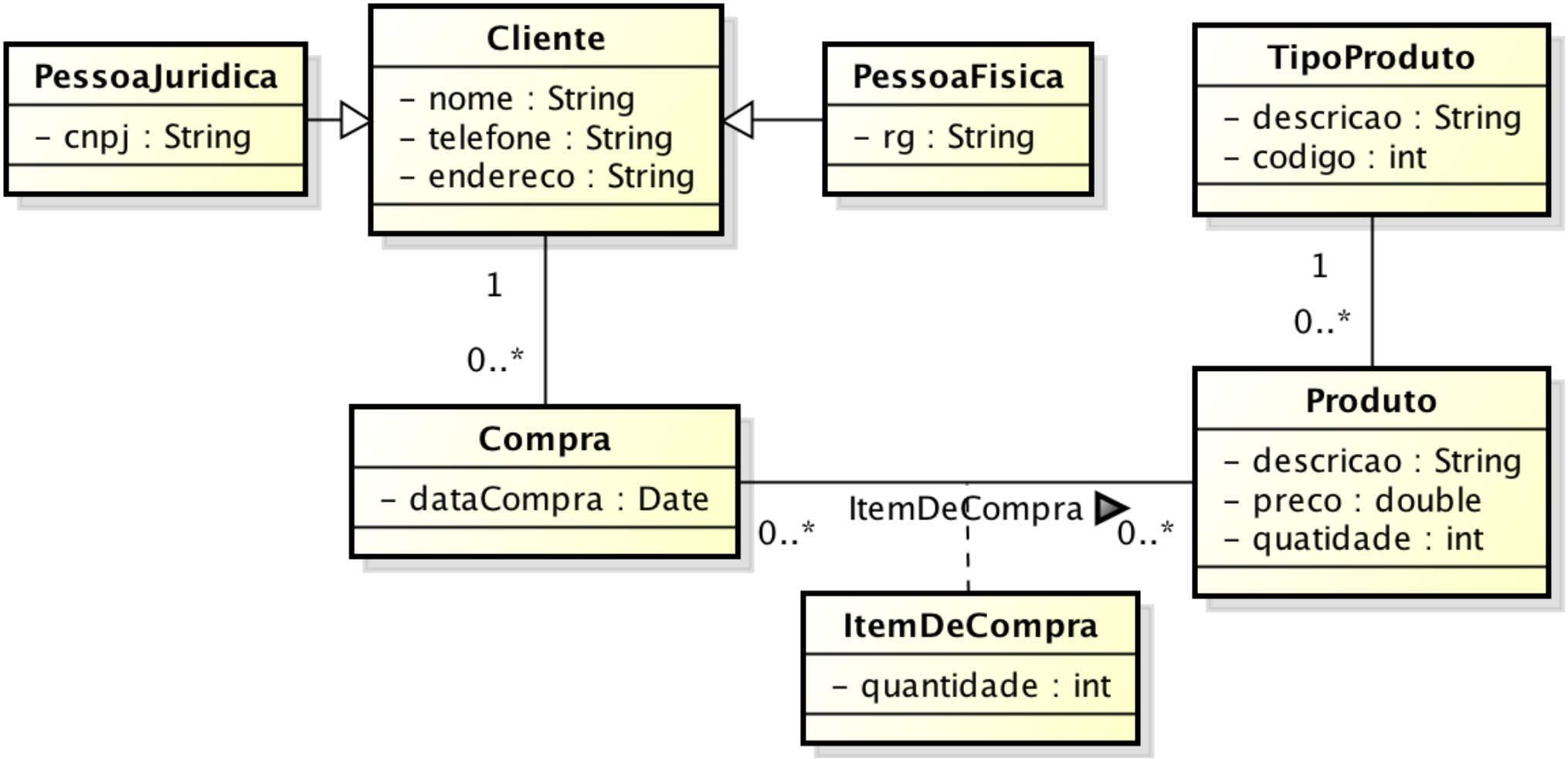


Exercícios - Relacionamentos

Exercício 2 - Leia a descrição, observe o diagrama de classes e implemente o código em Java.

- Uma floricultura deseja informatizar suas operações.
- Inicialmente, deseja manter um cadastro de todos os seus clientes, mantendo informações como: RG (se for um cliente pessoa física), CNPJ (se for um cliente pessoa jurídica), nome, telefone e endereço.
- Deseja também manter um cadastro contendo informações sobre os produtos que vende, tais como: nome do produto, tipo (flor, vaso, planta,...), preço e quantidade em estoque.
- Quando um cliente faz uma compra, a mesma é armazenada, mantendo informação sobre o cliente que fez a compra, a data da compra, o valor total e os produtos comprados.

Exercícios - Relacionamentos



Exercícios - Relacionamentos

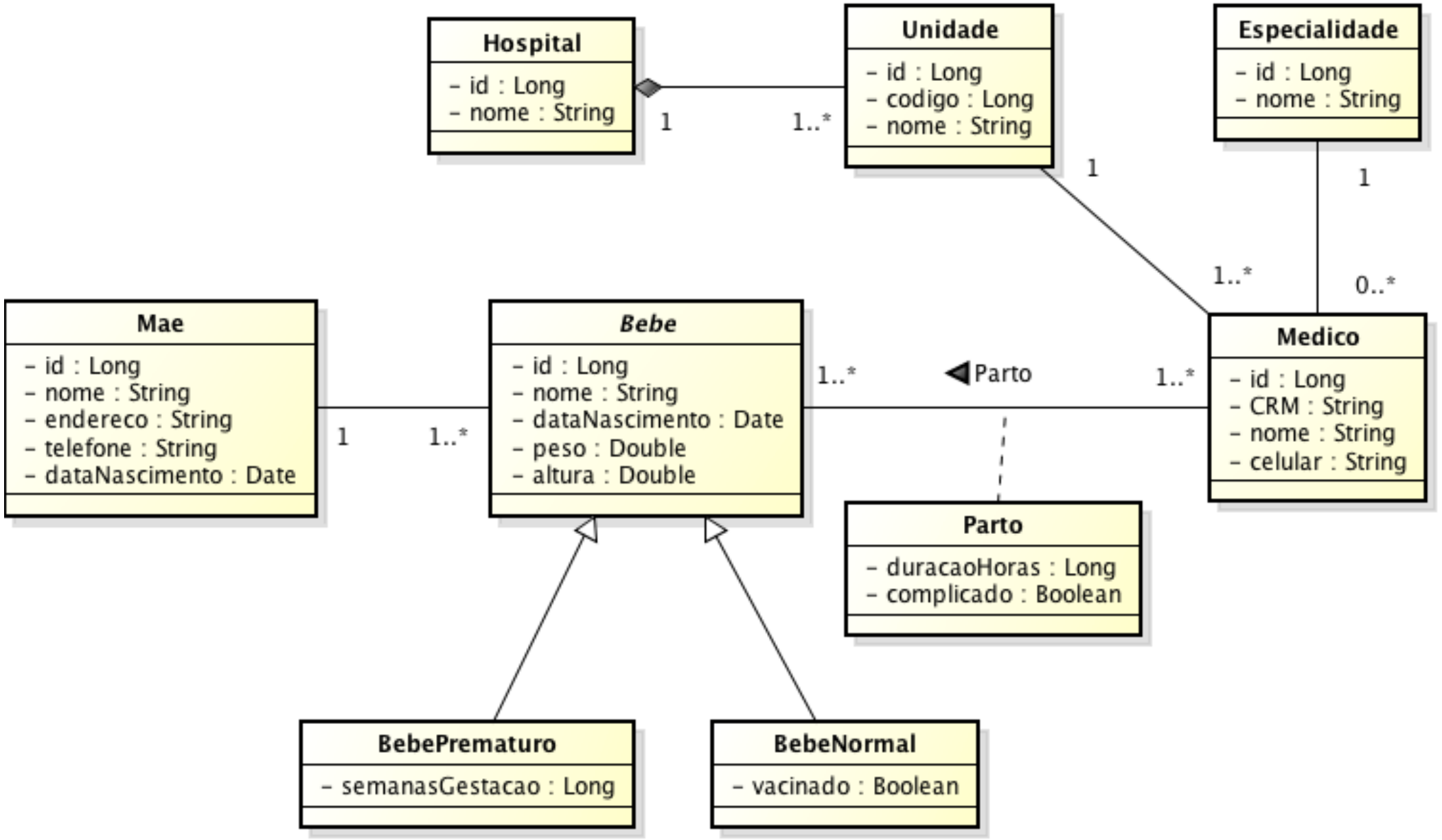
Exercício 3 - Leia a descrição, observe o diagrama de classes e implemente o código em Java.

- Um hospital-berçário, que possui diversas unidades de atendimento espalhadas na região, deseja informatizar suas operações.
- Quando um bebê nasce, algumas informações sobre ele são necessárias, tais como: nome, data do nascimento, peso do nascimento, altura, a mãe deste bebê e o médico que fez seu parto.
- Para as mães, o berçário também deseja manter um controle, guardando informações como: nome, endereço, telefone e data de nascimento.

Exercícios - Relacionamentos

- Para os médicos, é importante saber: CRM, nome, telefone celular, valor da hora de trabalho e especialidade. Sabe-se também que um médico atende em somente uma unidade, mas uma unidade pode ter vários médicos.
- Um detalhe é que quando um bebê nasce prematuramente, deseja-se saber quanto foi o período total da gestação em número de semanas. Bebês não prematuros podem ser vacinados
- Sobre o parto deseja-se armazenar qual foi a duração em horas e se foi um parto complicado ou não. Sabe-se também que um parto pode envolver um ou mais médicos.
- Sobre uma unidade deseja-se saber o código e seu nome.
- Sobre o hospital, simplesmente o nome.

Exercícios - Relacionamentos



Exercícios - Relacionamentos

Atenção: Não copie ou cole nenhum exercício. A repetição é intencional para criar fluência na linguagem.

1. Crie os relacionamentos do Diagrama de Classes a seguir.
2. Para cada classe envolvida em relacionamentos, faça um toString().
3. Crie instâncias na AplicacaoFinanceira para testar os relacionamentos.

Exercícios - Relacionamentos