Implementação - Relacionamento

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3

Relacionamento Entre Classes

- Associação
  - A classe dependente possui um atributo que é uma referência para a outra classe. Também chamada de dependência por atributo (ou estrutural)

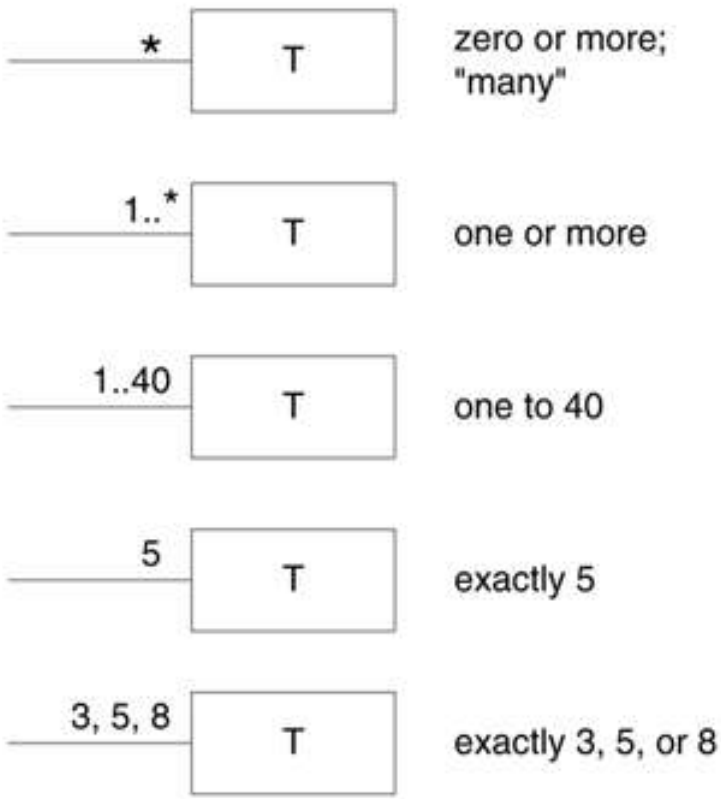
- Dependência
  - Indica que uma classe depende dos serviços (operações) fornecidos por outra classe.

Implementação - Associação

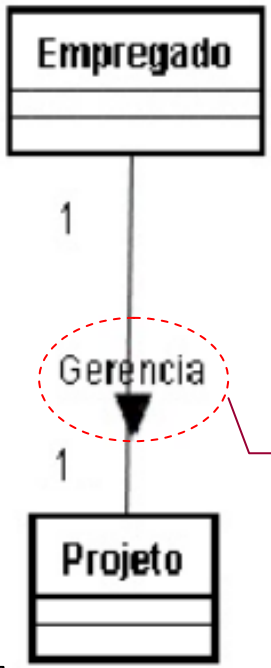
Associação

- Associações
  - Bidirecional
  - Unidirecional

Associação


*
T
zero or more;
"many"
1..*
T
one or more
1..40
T
one to 40
5
T
exactly 5
3, 5, 8
T
exactly 3, 5, or 8

Associação

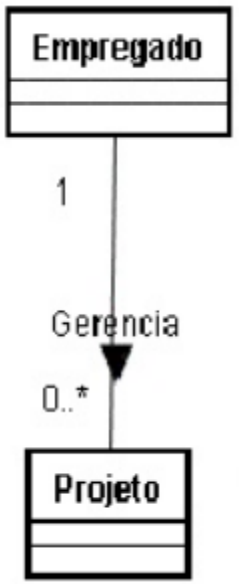



```
public class Empregado {

    private Projeto projeto;
}
```

Sentido de Leitura

```
public class Projeto {

    private Empregado empregado;
}
```

Associação



Exemplo de associação

Observe que um Empregado pode gerenciar nenhum ou vários projetos. Estamos lidando aqui com o conceito de coleções

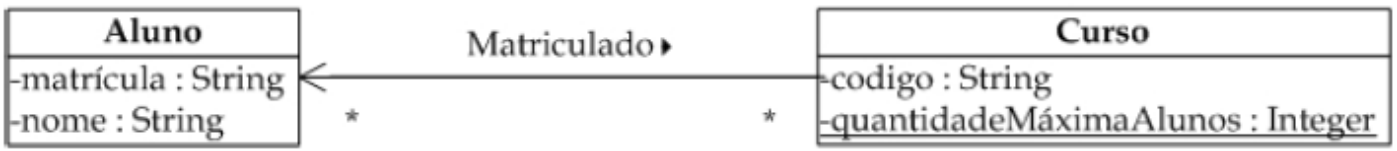
```
import java.util.ArrayList;

public class Empregado {

    private ArrayList<Projeto> projetos = new ArrayList<Projeto>();
}
```

Associações

- Associações
  - Associações podem ser bidirecionais ou unidirecionais.
  - Bidirecional
    - Indica que há um conhecimento mútuo entre os objetos associados.
  - Unidirecional
    - Indica que apenas um dos extremos da associação tem ciência da existência da mesma. Representada através da adição de um sentido à seta da associação.

Associações

- Associações
  - A escolha da navegabilidade de uma associação pode ser feita através do estudo dos diagramas de interação. O sentido de envio das mensagens entre objetos influencia na necessidade ou não de navegabilidade em cada um dos sentidos.

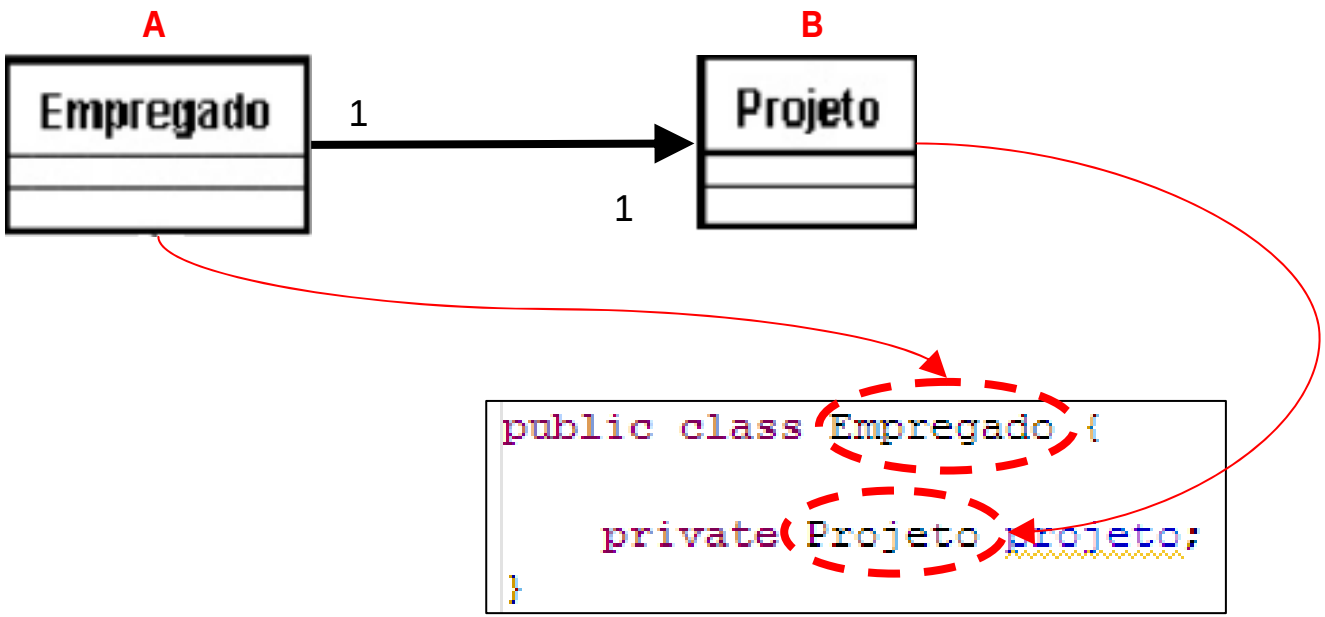


Implementação de Associações

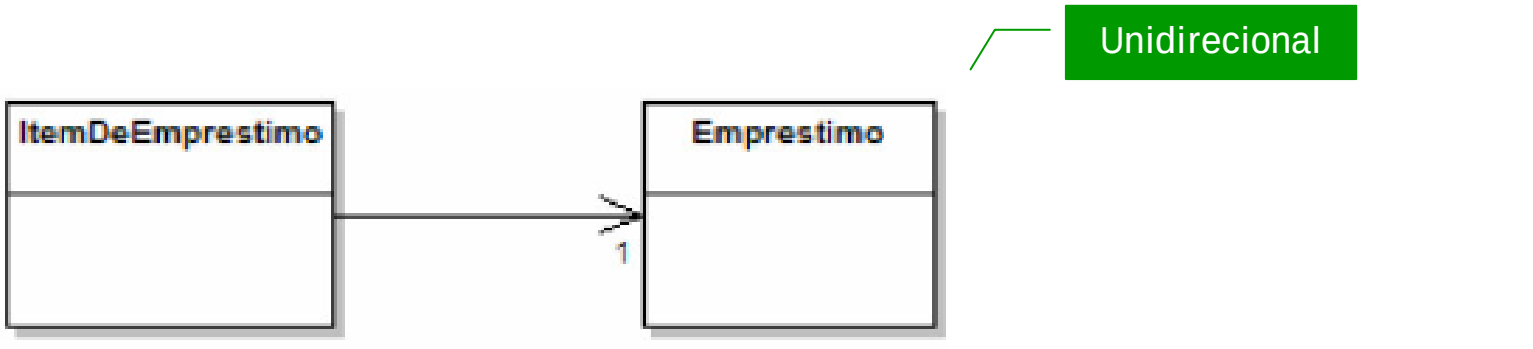
- Há três casos
  - Em função da conectividade: 1:1, 1:N e N:M
  - Para uma associação 1:X entre duas classes A e B:
    - Se a navegabilidade é unidirecional no sentido de A para B, é definido um atributo do tipo B na classe A.Se a navegabilidade é bidirecional, podemos aplicar o procedimento acima para as duas classes.

Relacionamento Unidirecional

Unidirecional Para 1



Unidirecional Para 1

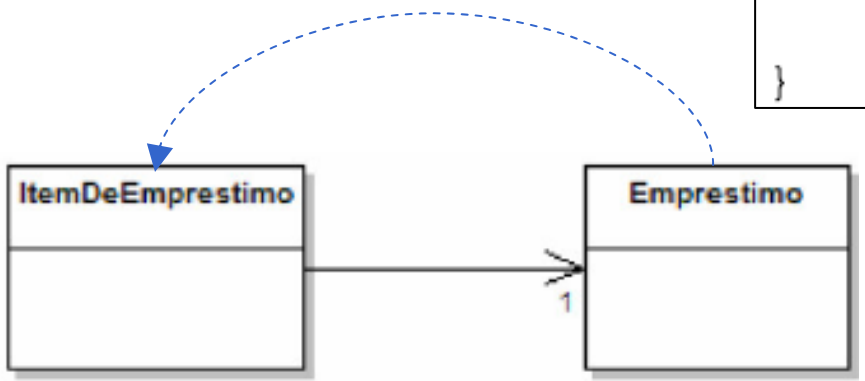


Como a associação 1 é obrigatória para o objeto origem,construtor da classe deve ter como parâmetro o elemento a ser associado, para que, desde o momento da criação, todas as instâncias da classe na origem da associação estejam consistentes.

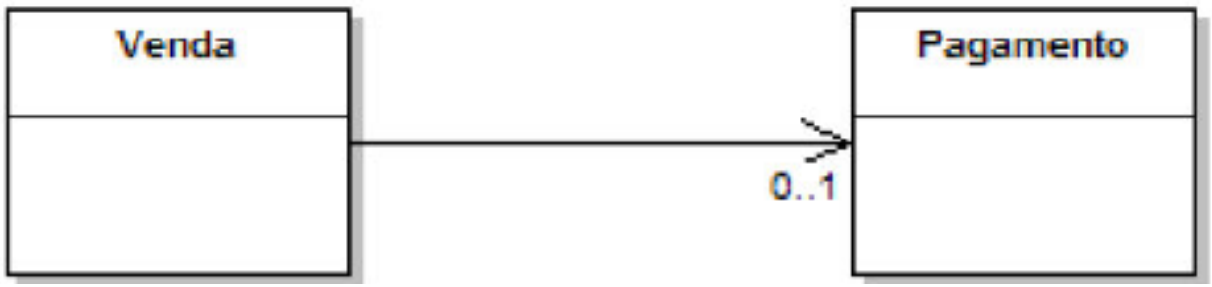
Unidirecional Para 1

Construtor de ItemDeEmprestimo

```
class itemDeEmprestimo
{
        private Emprestimo emprestimo;

        public ItemDeEmprestimo(Emprestimo emprestimo)
        {
                this.associaEmprestimo(emprestimo)
        }

        public void associaEmprestimo(Emprestimo emprestimo)
        {
                This.emprestimo = emprestimo;
        }

        public Emprestimo getEmprestimo()
        {
                return emprestimo;
        }
}
```



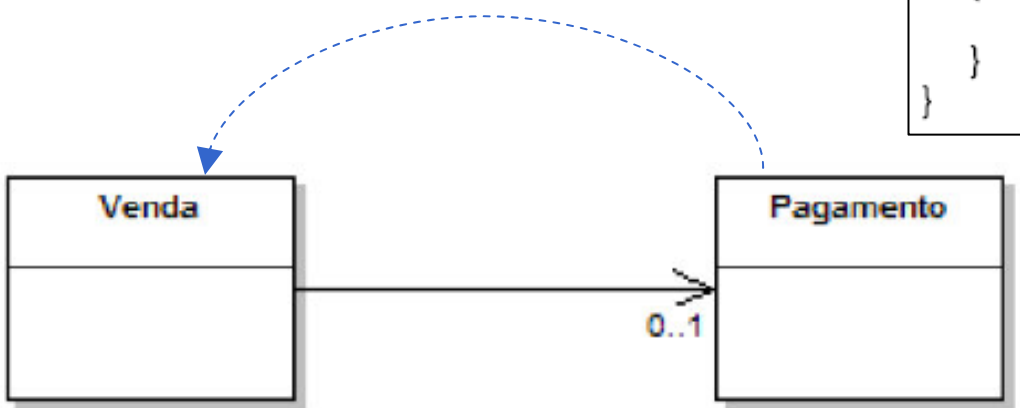
Unidirecional Para 0:1



Não é necessário passar um objeto como parâmetro para o método construtor, pois a associação 0..1 não é obrigatória

Unidirecional Para 0:1

```
class Venda
{
    private Pagamento pagamento;

    public Venda() {}

    public void associaPagamento(Pagamento pagamento)
    {
            this.pagamento = pagamento;
    }

    public void desassociaPagamento()
    {
            This.pagamento = null;
    }

    public Pagamento getPagamento()
    {
            return pagamento;
    }
}
```

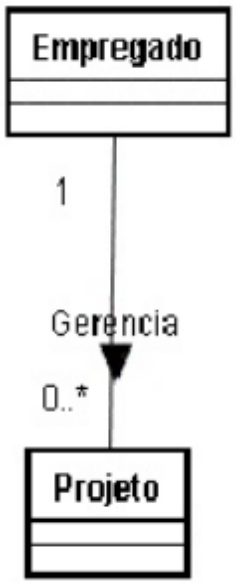


Unidirecional Para 0:1


```
public class Professor {
  private GradeDisciplinas grade;
  ...
}
```

Unidirecional Para *

- Para uma associação 1:N entre duas classes A e B:
  - São utilizados atributos cujos tipos representam coleções de elementos.
  - 1:N: essas coleções são codificadas na classe de multiplicidade 1



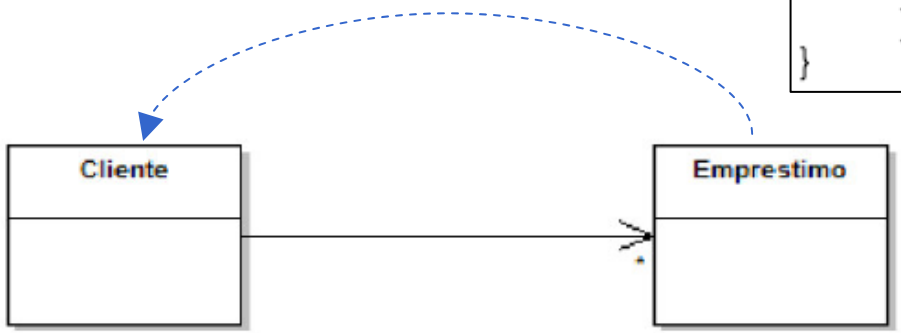

```
import java.util.Set;

public class Empregado {

    private Set<Projeto> projetos;
}
```

Unidirecional Para *

A implementação de uma associação desse tipo constitui-se pela implementação de uma variável do tipo Conjunto, no código Java da classe Cliente.

Unidirecional Para *

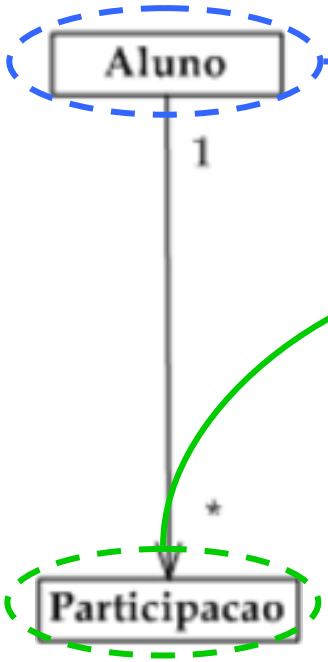
```
class Cliente {
        private Set emprestimos = new HashSet();
        public Cliente () { }

        public void adicionaEmprestimo(Emprestimo emprestimo)
        {
                this.emprestimos.add(emprestimo);
        }

        public void removeEmprestimo(Emprestimo emprestimo)
        {
                this.emprestimos.remove(emprestimo);
        }

        public Set getEmprestimos ()
         {
                return Collections.unmodifiableSet(emprestimos);
        }
}
```



Unidirecional Para *

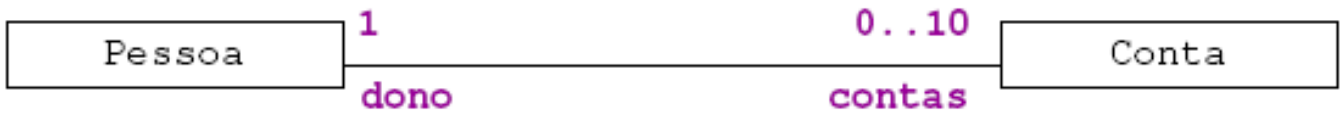

```
public class Aluno {
  private Set<Participacao> participacoes;
  ...
```

Unidirecional Para *




```
public class Aluno {
   private Set<Participacao> participacoes;
   ...

   public boolean adicionarParticipacao(Participacao p){
         ...
   }

   public boolean removerParticipacao(Participacao p) {
         ...
   }
}
```

Unidirecional – Valores Específicos




```
public class Pessoa {
   int numProxConta=0;
   static final int numMaxConta=10;
   Conta[] conta;
   ...
   Pessoa() {
        conta = new Conta[numMaxConta];
   }
   void associarConta(Conta conta){
        if (numProxConta<numMaxConta)
               this.conta[numProxConta++] = conta;
        else
               System.out.println("Número máximo atingido!");
   }
}
```

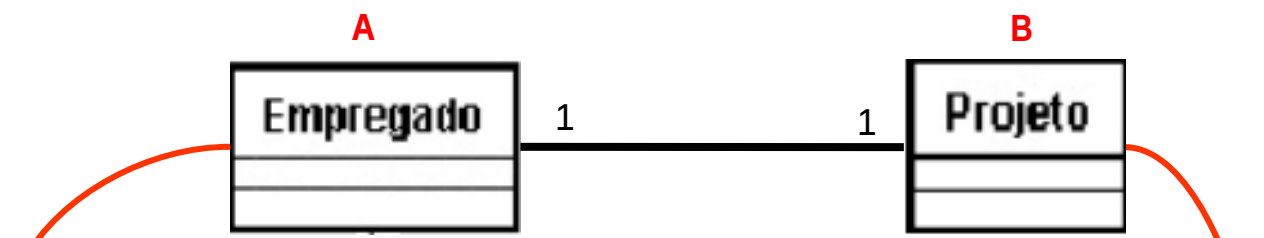
```
public class Conta{
   Pessoa dono;
   ...
   Conta(Pessoa dono){
        this.dono = dono;
   }
}
```

Relacionamento Bidirecional

Associação Bidirecional

- Associação bidirecional
  - Deve ser implementada em ambas as classes.
  - Independente de a multiplicidade ser para 1, para 0..1 ou para *

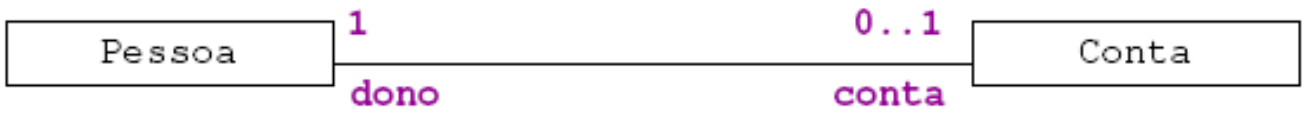
Associação Bidirecional 1:1




```
public class Empregado {

    private Projeto projeto;
}
```

```
public class Projeto {

    private Empregado empregado;
}
```

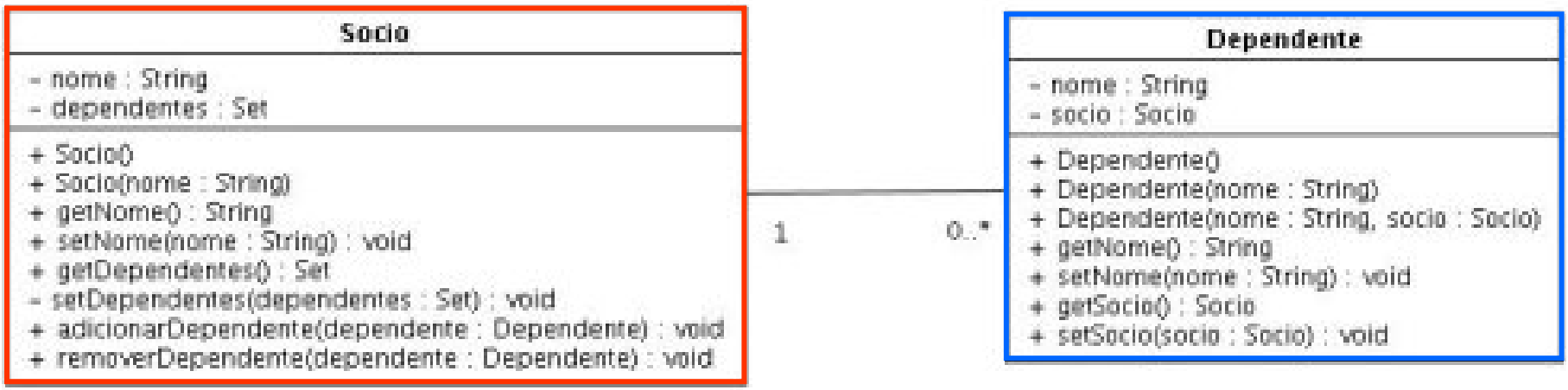
Bidirecional Para 0:1




```
public class Pessoa {
   Conta conta;
   ...
   Pessoa(){
        conta = null;
         ...
   }
   void associarConta(Conta conta) {
        this.conta = conta;
   }
}
```

```
public class Conta{
   Pessoa dono;
   ...
   Conta(Pessoa dono){
        this.dono = dono;
         ...
   }
}
```

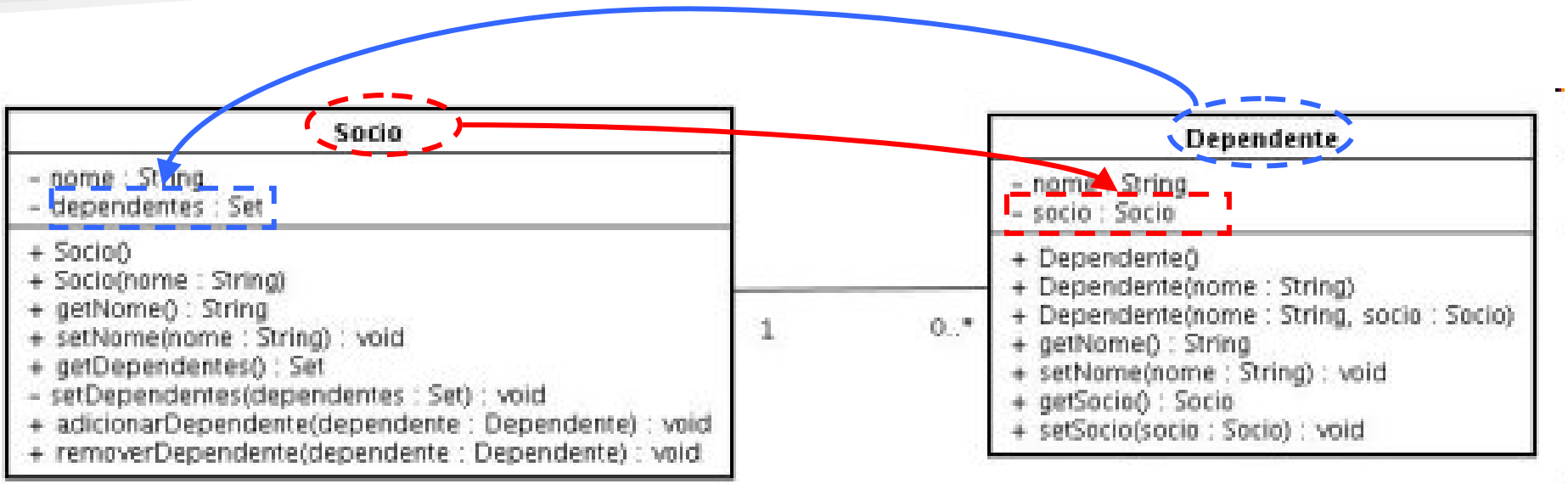
Associação Bidirecional 1:N




```
public class Socio {
        private String nome;
        private Set dependentes;
```

```
public class Dependente {
        private String nome;
        private Socio socio;
```

Implementação de Associações




```
public class Socio {
        private String nome;
        private Set dependentes;
```

```
public class Dependente {
        private String nome;
        private Socio socio;
```

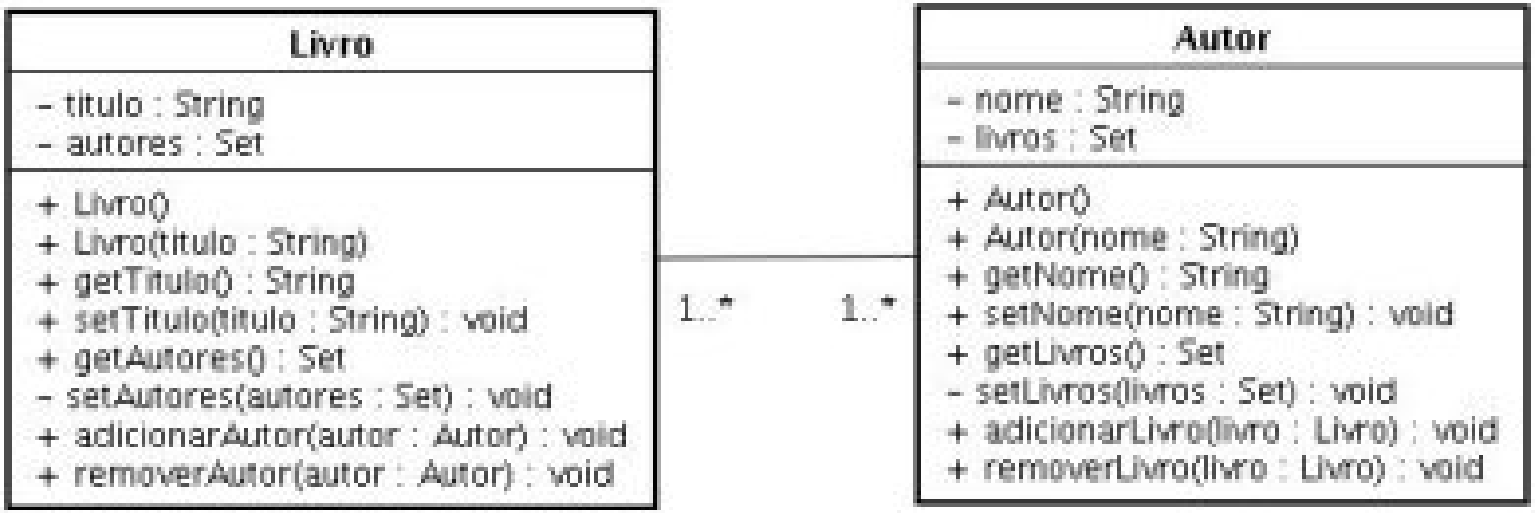
Associação Bidirecional 1:N

Associação Bidirecional 1:N

```
class Cliente
{
        private Set emprestimos = new HashSet();

        public Cliente () {}
```

```
class Emprestimo
{
        private Cliente cliente;

        public Emprestimo(Cliente cliente)
        {
                this.associaCliente(cliente);
        }
```

Associação Bidirecional N:N



- Implementar N:M
  - Insira, em ambas as classes, um atributo do tipo coleção
  - Insira os métodos get e set correspondentes ao atributo coleção;
  - Insira métodos para adicionar e remover objetos na coleção

Implementação de Associações

```
public class Livro {

private String titulo;
private Set autores;

…
}
```

1

```
public void setAutores(Set autores) {
...
}

public Set getAutores() {
...
}
```

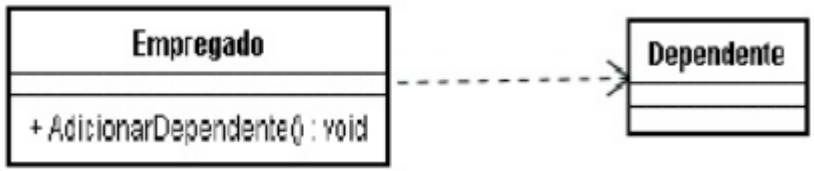
2

```
public void adicionarAutor(Autor autor){
…
}

public void removerAutor(Autor autor){
…
}
```

3

Implementação - Dependência

Dependência

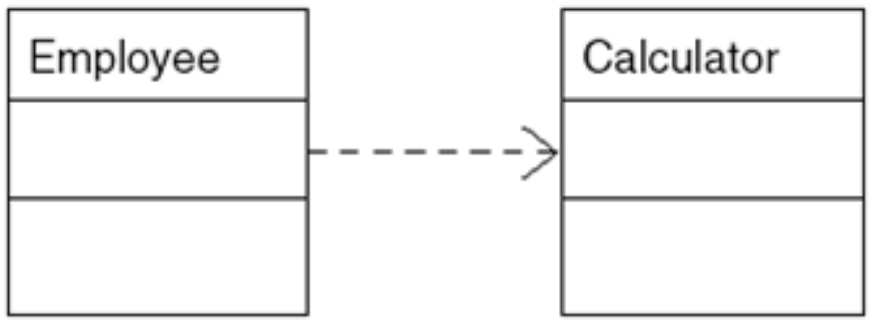



```
public class Empregado {

    public void adicionarDependente(Dependente dependente){
        //implementação do método
    }
}
```

Dependência por atributo

Dependência

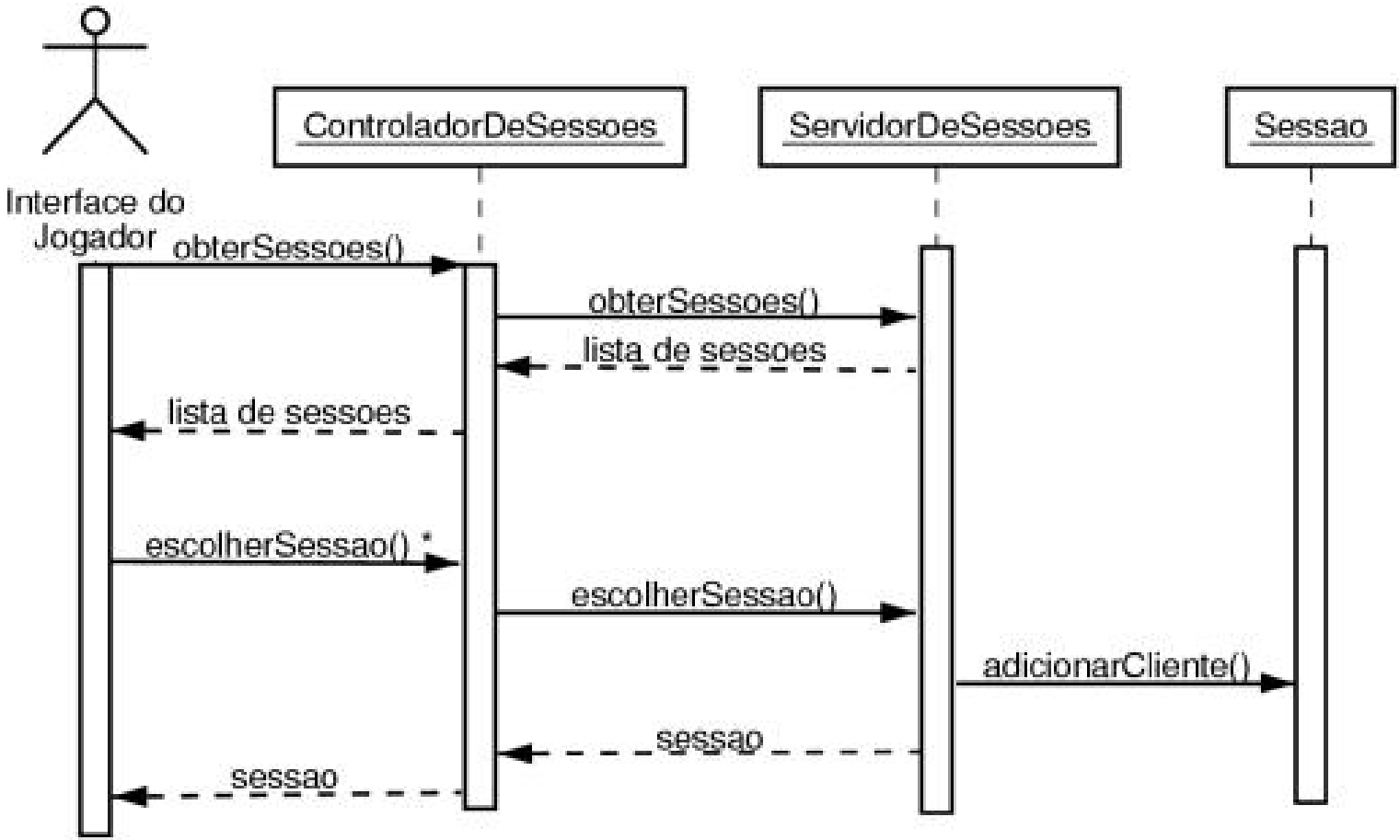
```
public class Employee {
        public void calcSalary(Calculator c) {
        ...
        }
}
```



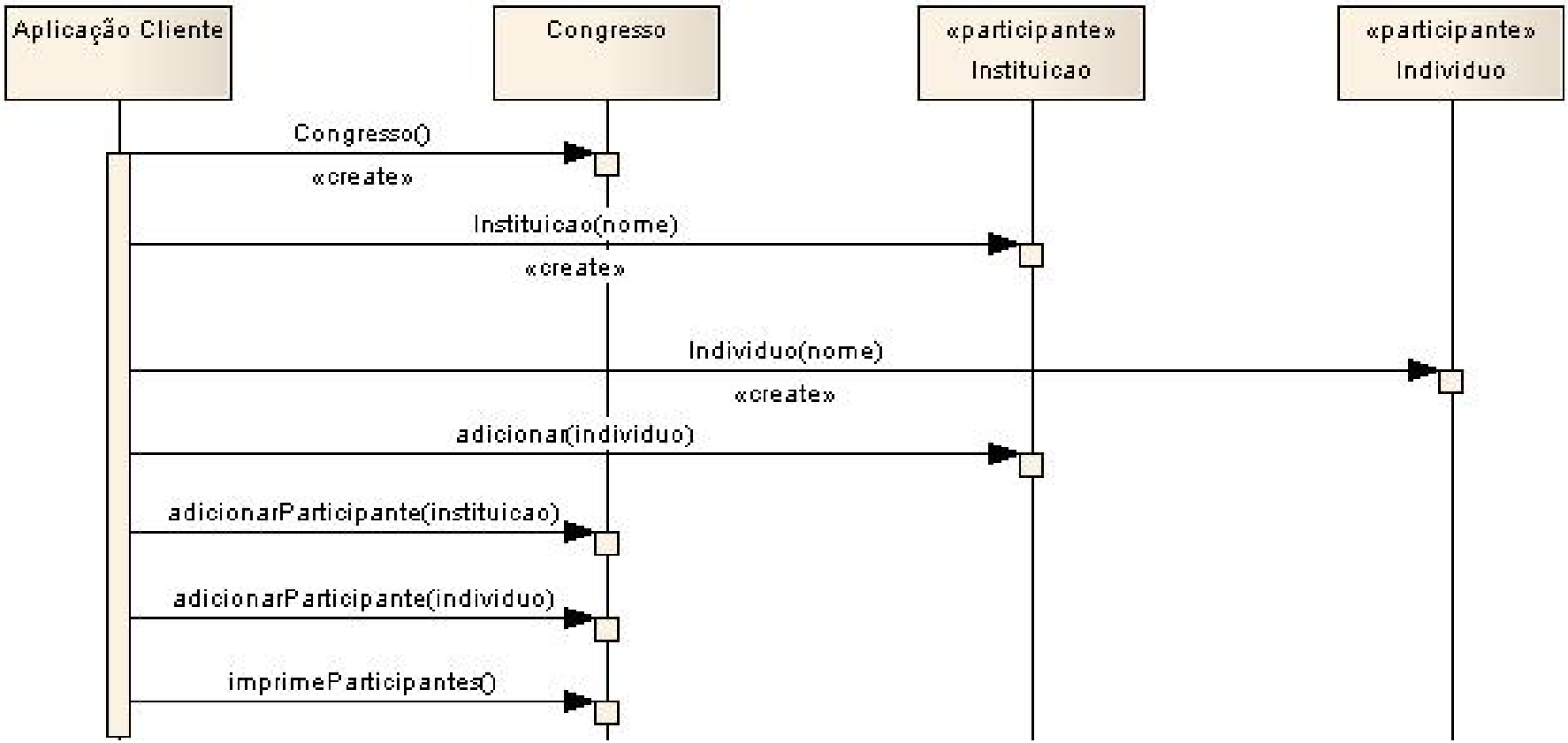
Encontrar Relacionamentos

Relações UML

Diagrama de Sequência



Relações UML



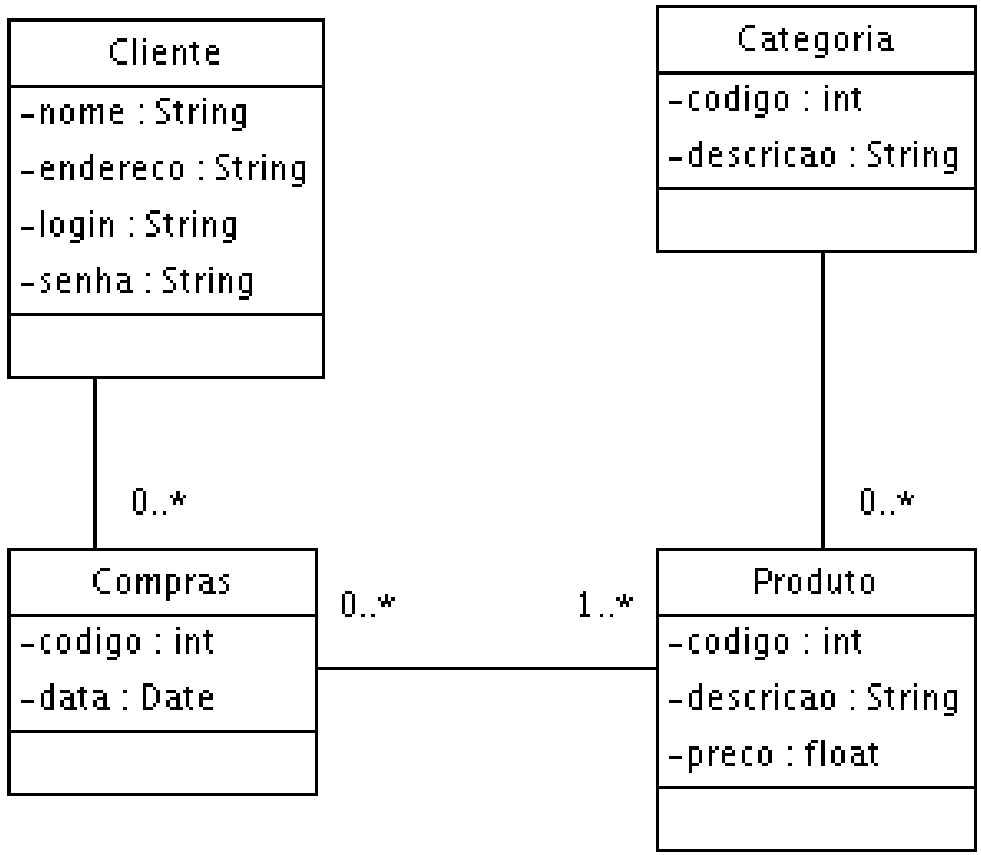
Exercícios

Exercício

- Exercício
  - Em um sistema de uma locadora de DVDs, um filme deve possuir ao menos uma cópia, podendo possuir diversas delas, porém cada cópia se refere exclusivamente a um único filme. Um sócio pode realizar muitas locações enquanto permanecer sócio, mas uma locação se refere exclusivamente a determinado sócio. Cada locação deve referenciar-se obrigatoriamente ao menos a uma cópia de um filme, podendo referenciar-se a muitas cópias. Já uma cópia pode ter sido locada muitas vezes
  - Crie as classses, especifique suas relações e, depois, implemente em Java
  - OBS: considere apenas relações bidirecionais
  - Procure analisar se alguma relação pode ser unidirecional

Exercício

Codificar as classes e associações



Exercício

```java
public class Point {
        private int X_POS = 0;
        private int Y_POS = 0;

        public int getXpos() {
            return this.X_POS;
        }

        public void setXpos(int xpos) {
            this.X_POS = xpos;
        }

        public int getYpos() {
            return this.Y_POS;
        }

        public void setYpos(int ypos) {
            this.Y_POS = ypos;
        }
}
```

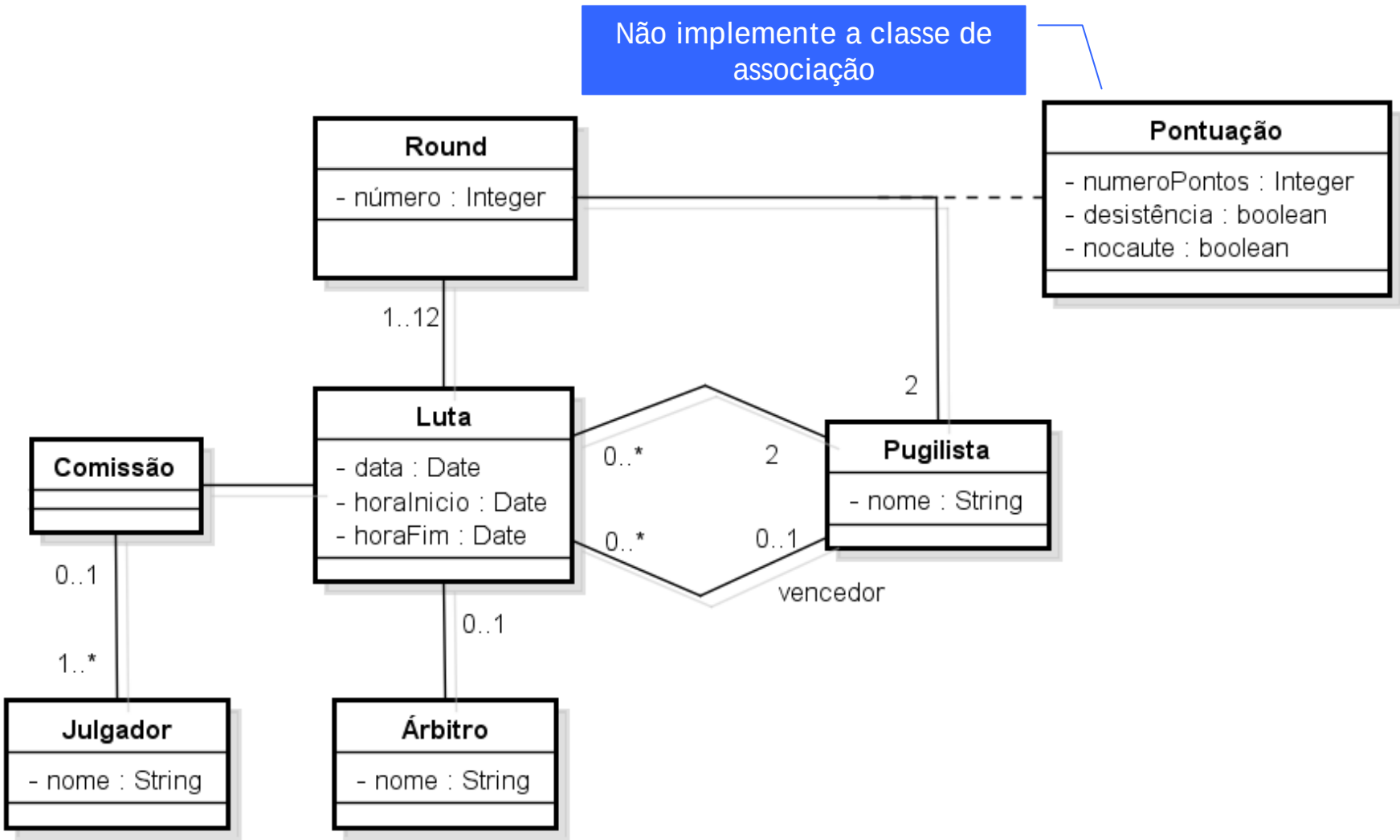
```java
public class Circle {
    private Point pointObj;

}
```

(a)

(b)

```java
public class Circle {
    private Point[] pointObj;

}
```

Exercício


Não implemente a classe de associação
Round
- número : Integer
Pontuação
- numeroPontos : Integer
- desistência : boolean
- nocaute : boolean
1..12
2
Luta
- data : Date
- horaInicio : Date
- horaFim : Date
Comissão
0..*
2
0..*
0..1
Pugilista
- nome : String
vencedor
0..1
1..*
0..1
Julgador
- nome : String
Árbitro
- nome : String

Implementação - Relacionamento

Orientação a Objetos – DCC025

Prof. Edmar Welington Oliveira
edmar.oliveira@ufjf.edu.br

Universidade Federal de Juiz de Fora – UFJF
Departamento de Ciência da Computação - DCC

2011.3