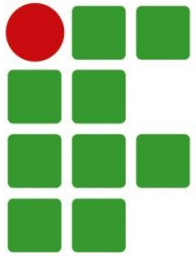


INSTITUTO FEDERAL
Goiás

Instituto Federal de Goiás
Câmpus Goiânia
Bacharelado em Sistemas de Informação
Disciplina: Programação Orientada a Objetos I

Associação de Classes
Agregação

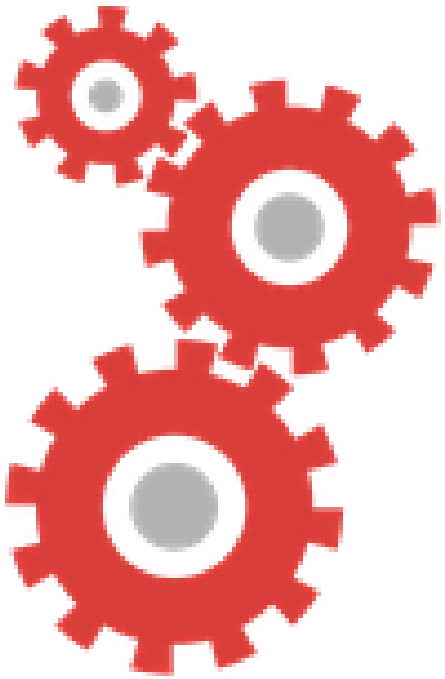
Prof. Ms. Renan Rodrigues de Oliveira
Goiânia - GO

Introdução

Ao construir abstrações, observa-se que há um número muito pequeno de classes que trabalham sozinhas. Em vez disso, a maioria das classes colaboram com outras de diversas maneiras.

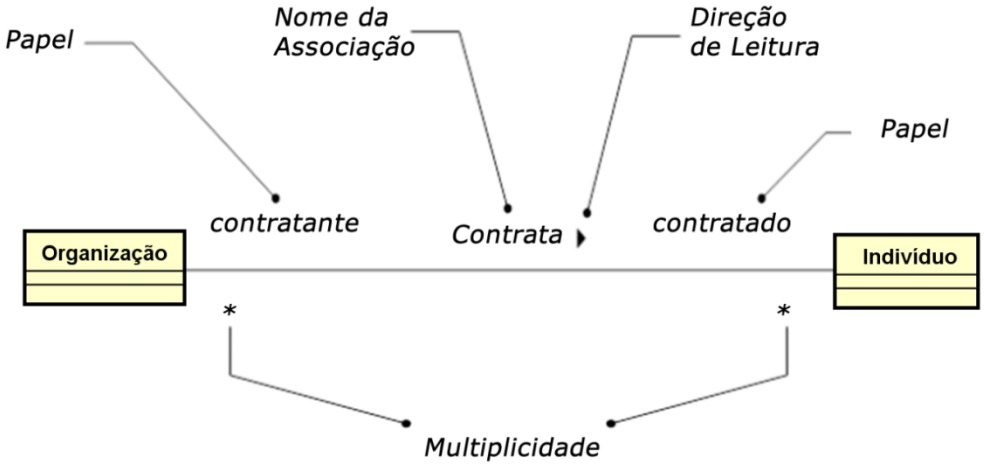
Podendo as Classes serem consideradas engrenagens ou peças de um quebra-cabeça, uma questão surge naturalmente:

Como essas peças poderiam ser relacionadas?



Associação

Uma associação é um relacionamento estrutural que especifica que objetos de um tipo são conectados a outro tipo.



Uma associação pode ter os seguintes elementos:

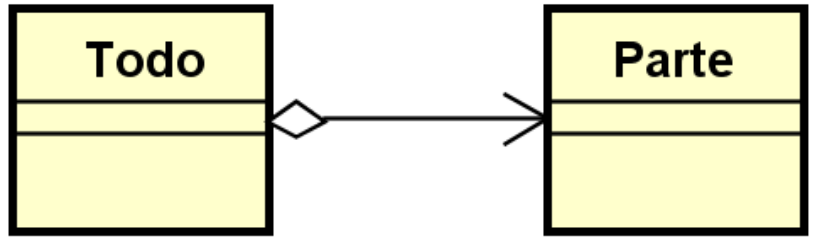
- **Nome da Associação:** descreve a natureza da associação;
- **Papel:** define o papel específico neste relacionamento;
- **Direção de Leitura:** indica como a associação deve ser lida;
- **Multiplicidade:** define quantos objetos estarão conectados a uma instância de uma associação.

Agregação

Indicada para representar um relacionamento entre “parte” e “todo”, onde o “todo” é formado por partes.

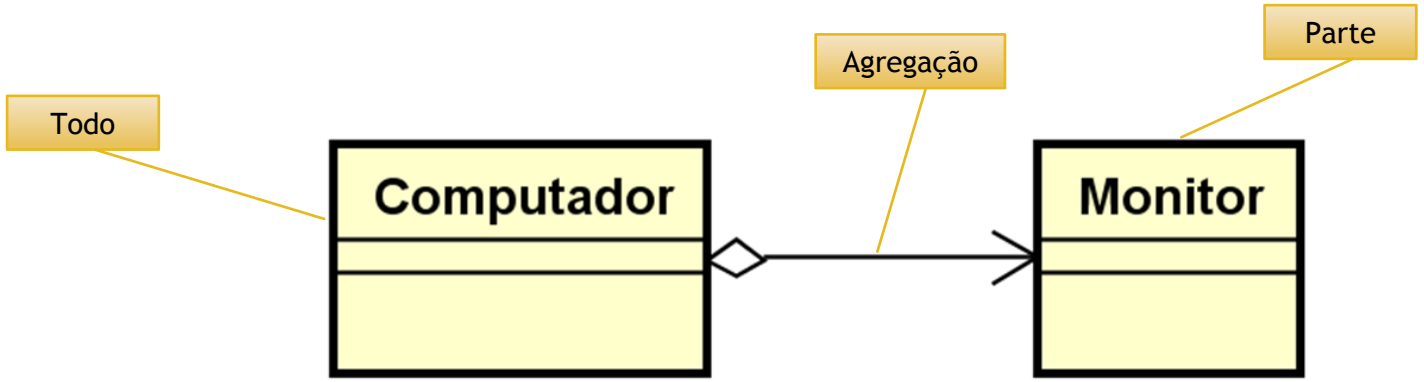
Este relacionamento é caracterizado pela parte poder existir sem o todo, ou seja, a parte deve existir antes que o vínculo seja realizado.

- Um objeto da classe parte integrante pode existir sem o todo, sendo que este último apenas agrega as partes já existentes;
- Tempo de vida da classe "parte" independente do tempo da classe "todo".



Exemplo: Agregação

Exemplo: O Computado e seu Monitor



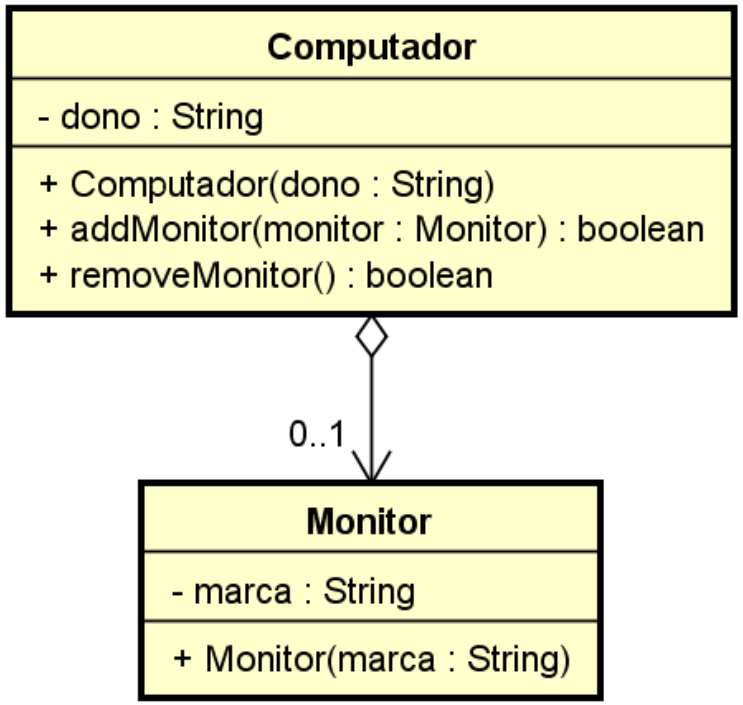
Para que a parte possa existir sem o todo, ela deve estar criada antes de estar agregada ao todo.

- Sua referência deve ser conhecida em outra parte do programa, de modo que, se o todo acabar, a parte continue podendo ser referenciada;
- O que será agregado (vinculado) ao objeto “todo” será a referência que representa o objeto “parte”.

Agregação: Multiplicidade 0..1

Exemplo: O Computado e seu Monitor

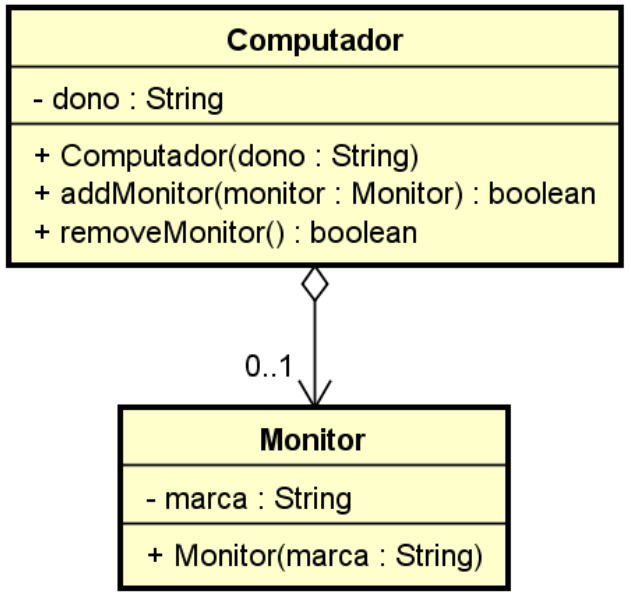
O Monitor pode fazer parte de um Computador. Se o Computador for jogado fora, o Monitor ainda continua existindo.



Agregação: Multiplicidade 0..1

Na multiplicidade 0..1, o "todo" pode nascer sem possuir nenhuma parte.

- Ao longo de seu ciclo de vida, uma "parte" pode agregar ao "todo", com o "todo" sabendo qual "parte" estará se relacionando com ele;
- Tempo de vida da classe "parte" não depende do tempo da classe "todo".



Implementação

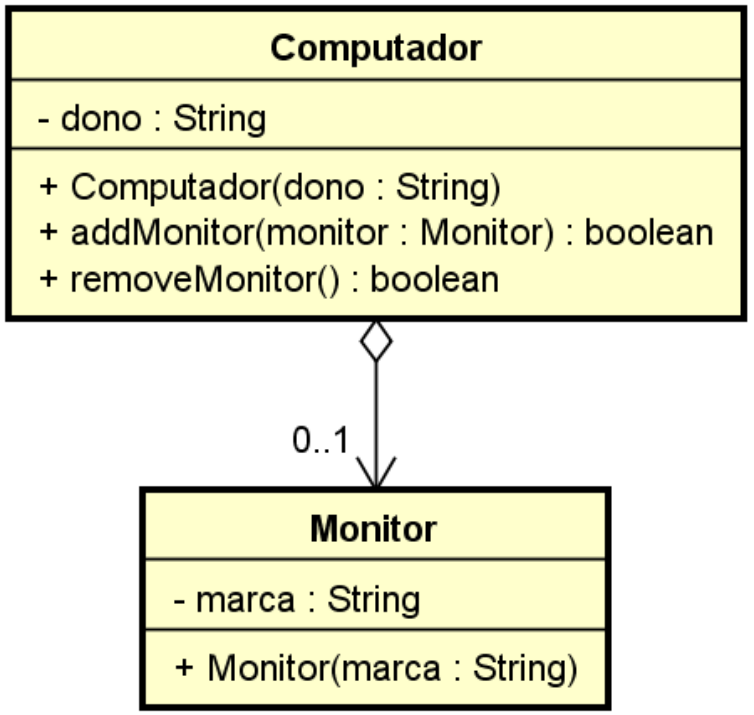
- Um Monitor agrega a um Computador;
  - O Computador pode ter 0 ou 1 Monitor;
  - O vínculo se dará no método addMonitor();
- Primeiro programe as partes, depois o relacionamento;
  - Crie o Monitor, para depois vincular ao Computador

É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.

Agregação: Multiplicidade 0..1

Implementando a Classe Monitor

```java
package br.com.renanrodrigues.agregacao;

public class Monitor {
    private String marca;

    public Monitor(String marca) {
        this.marca = marca;
    }

    public String getMarca() {
        return marca;
    }

    @Override
    public String toString() {
        return "Monitor [marca=" + marca + "]";
    }
}
```

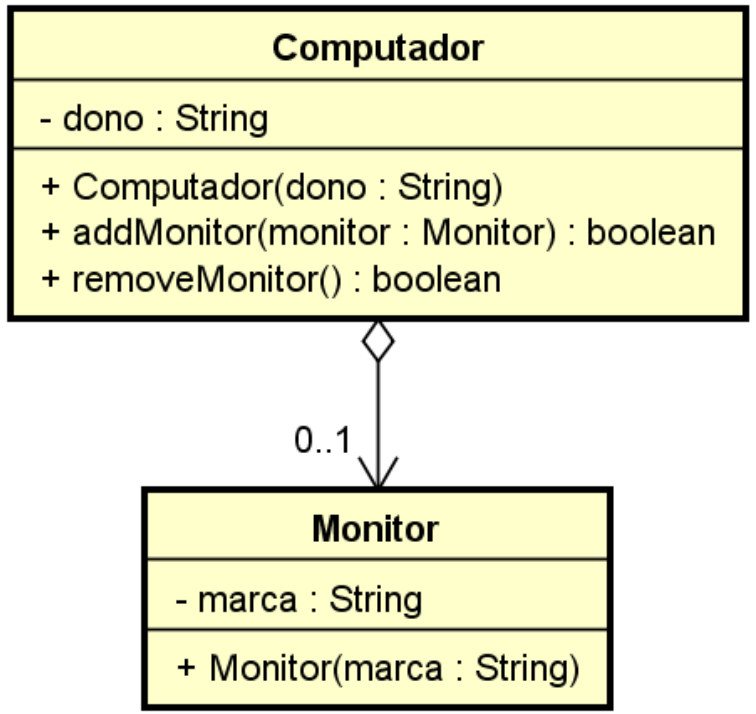


Agregação: Multiplicidade 0..1

Implementando a Classe Computador

```java
package br.com.renanrodrigues.agregacao;

public class Computador {
    private String dono;
    private Monitor monitor;

    public Computador(String dono) {
        this.dono = dono;
    }

    public boolean addMonitor(Monitor monitor) {

        boolean sucesso = false;

        if (this.monitor == null) {
            this.monitor = monitor;
            sucesso = true;
        }

        return sucesso;
    }
```

CONTINUA >>

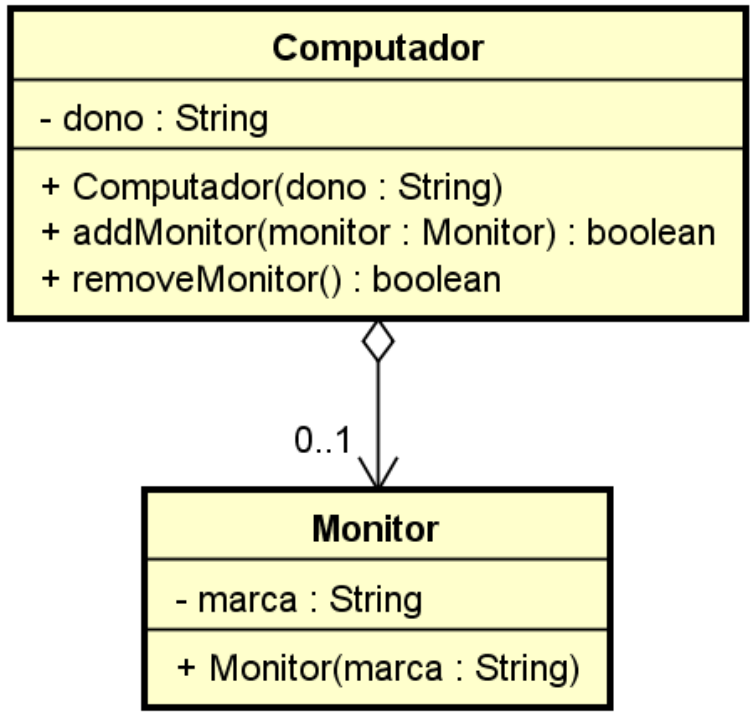


Agregação: Multiplicidade 0..1

Implementando a Classe Computador

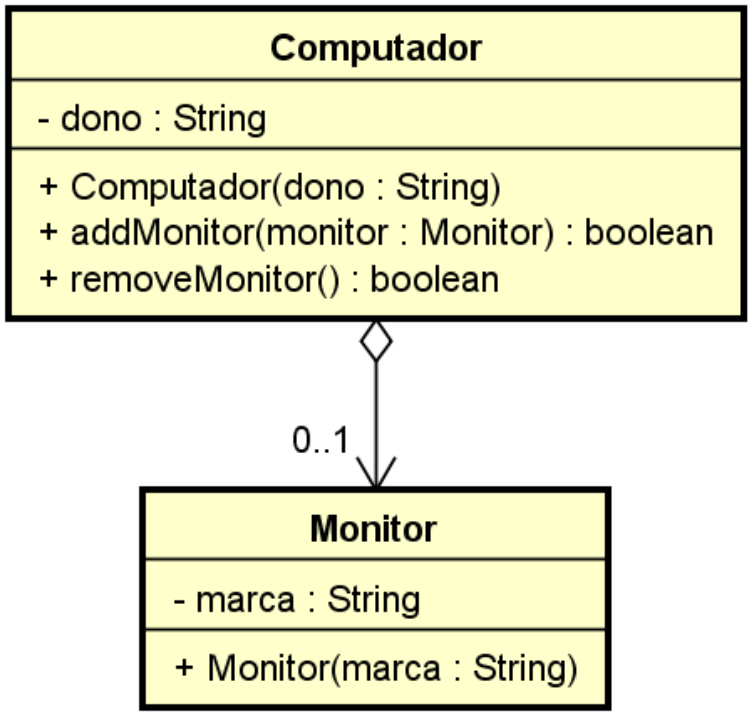
```
23    public boolean removeMonitor() {
24
25        boolean sucesso = false;
26
27        if (this.monitor != null) {
28            this.monitor = null;
29            sucesso = true;
30        }
31
32        return sucesso;
33    }
```

CONTINUA >>



Agregação: Multiplicidade 0..1

Implementando a Classe Computador




```
35      public Monitor getMonitor() {
36          return monitor;
37      }
38
39      public String getDono() {
40          return dono;
41      }
42
43      public void setDono(String dono) {
44          this.dono = dono;
45      }
```

```
47      @Override
48      public String toString() {
49          return "Computador [dono=" + dono + ", monitor=" + monitor + "]";
50      }
51  }
```

Agregação: Multiplicidade 0..1

Programa Principal

```
1  package br.com.renanrodrigues.agregacao;
2
3  public class TesteComputador {
4
5      public static void main(String[] args) {
6
7          Computador c = new Computador("Renan");
8          Monitor m1 = new Monitor("Samsung");
9          Monitor m2 = new Monitor("Philips");
10
11         System.out.println(c);
12
13         c.addMonitor(m1);
14         System.out.println(c);
15
16         c.addMonitor(m2);
17         System.out.println(c);
18
19         c.removeMonitor();
20         System.out.println(c);
21
22         c.addMonitor(m2);
23         System.out.println(c);
24     }
25 }
```

Saída do Programa

```
Computador [dono=Renan, monitor=null]
Computador [dono=Renan, monitor=Monitor [marca=Samsung]]
Computador [dono=Renan, monitor=Monitor [marca=Samsung]]
Computador [dono=Renan, monitor=null]
Computador [dono=Renan, monitor=Monitor [marca=Philips]]
```

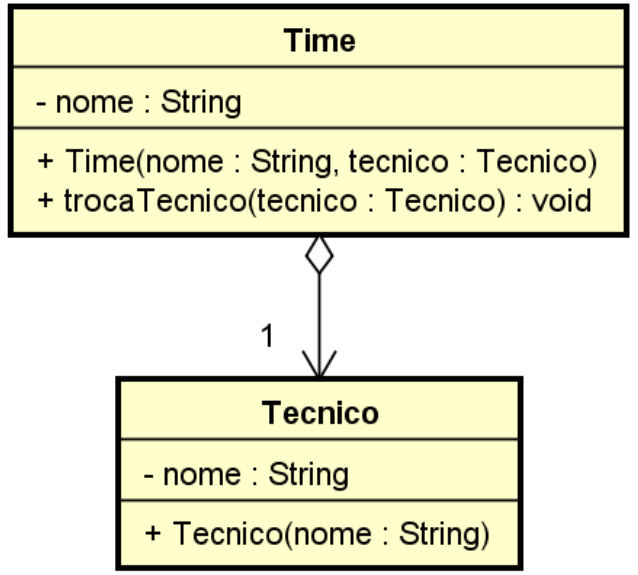
Agregação: Multiplicidade 1

Exemplo: Um Time e seu Técnico

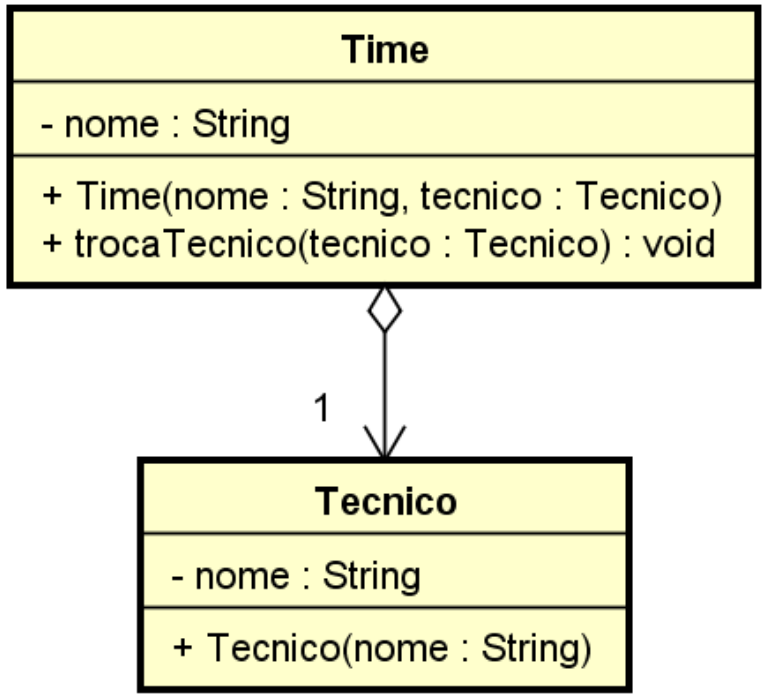
Um Time tem um Técnico. Se o Técnico perder o emprego ele ainda continua sendo um Técnico.

Na multiplicidade 1, o "todo" DEVE nascer possuindo uma parte.

- Assim sendo, neste caso, a parte deve existir antes do "todo";
- Ao longo de seu ciclo de vida, uma "parte" pode ser substituída, mas nunca removida.



Agregação: Multiplicidade 1



Implementação

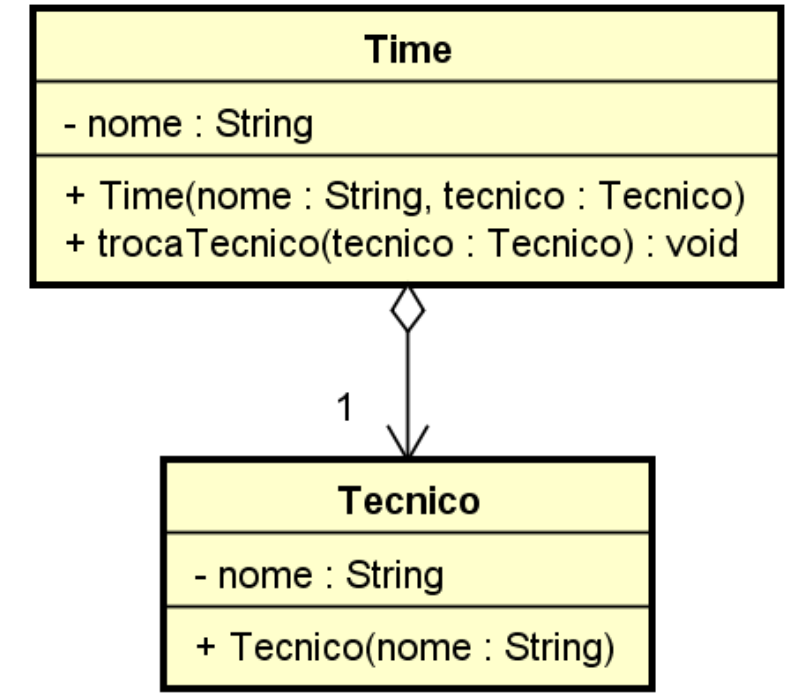
- Um Time possui um Técnico;
  - O vínculo se dará no construtor;
- Primeiro programe as partes, depois o relacionamento;
  - Crie o Técnico antes de criar o Time.

É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.

Agregação: Multiplicidade 1

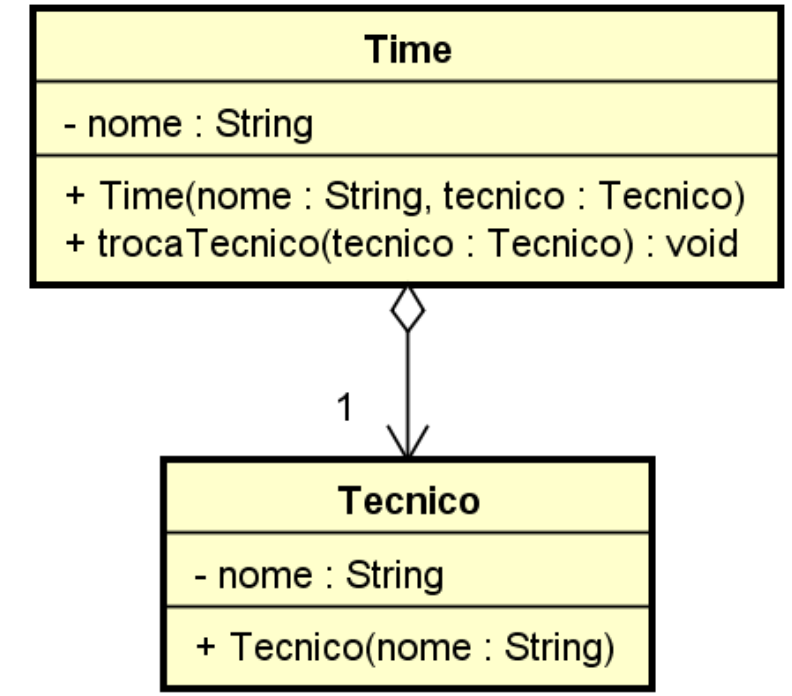
Implementando a Classe Técnico

```java
package br.com.renanrodrigues.agregacao;

public class Tecnico {

    private String nome;

    public Tecnico(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

    @Override
    public String toString() {
        return "Tecnico [nome=" + nome + "]";
    }
}
```



Agregação: Multiplicidade 1

Implementando a Classe Time



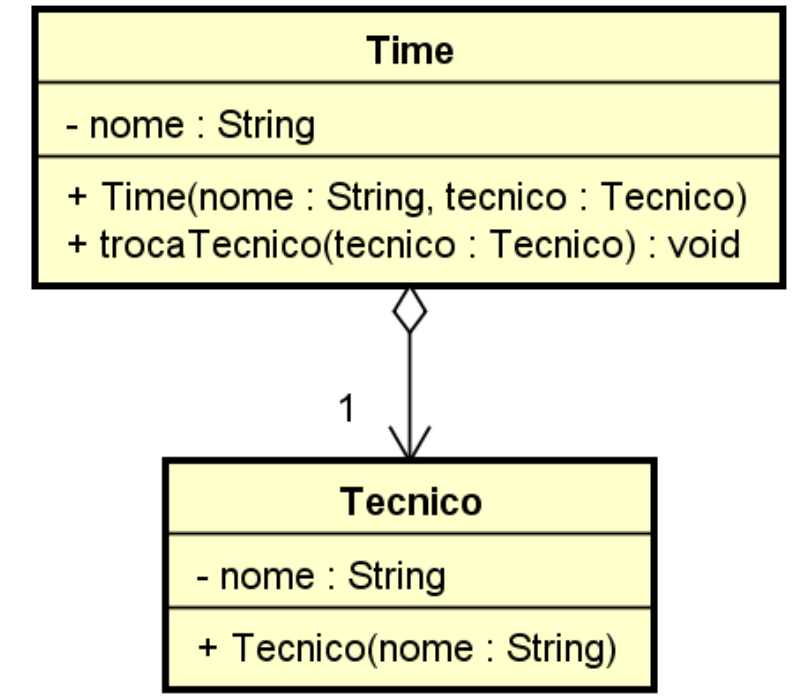

```java
package br.com.renanrodrigues.agregacao;

public class Time {

    private String nome;
    private Tecnico tecnico;

    public Time(String nome, Tecnico tecnico) {

        if (tecnico == null) {
            throw new NullPointerException("A referência do Técnico não pode ser nula!");
        }

        this.nome = nome;
        this.tecnico = tecnico;
    }
```

CONTINUA >>

Agregação: Multiplicidade 1
Implementando a Classe Time

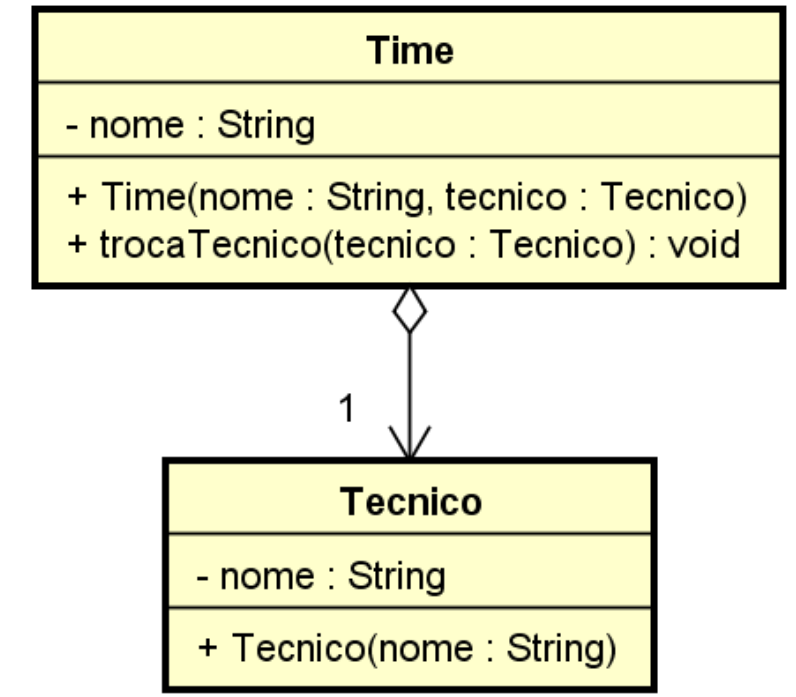
```
18    public boolean trocaTecnico(Tecnico tecnico) {
19
20        boolean sucesso = false;
21
22        if (tecnico != null) {
23            this.tecnico = tecnico;
24            sucesso = true;
25        }
26
27        return sucesso;
28    }
```

CONTINUA >>



Agregação: Multiplicidade 1
Implementando a Classe Time

```
30      public String getNome() {
31          return nome;
32      }
33
34      public Tecnico getTecnico() {
35          return tecnico;
36      }
37
38      @Override
39      public String toString() {
40          return "Time [nome=" + nome + ", tecnico=" + tecnico + "]";
41      }
42  }
```



Agregação: Multiplicidade 1

Programa Principal

```java
package br.com.renanrodrigues.agregacao;

public class TesteTime {

    public static void main(String[] args) {

        Tecnico t1 = null;
        Tecnico t2 = new Tecnico("Dunga");
        Tecnico t3 = new Tecnico("Felipão");

        Time tm;

        try {
            tm = new Time("Brasil", t1);
        } catch (NullPointerException e) {
            System.err.println(e.getMessage());
        }

            tm = new Time("Brasil", t2);
            System.out.println(tm);

            tm.trocaTecnico(t3);
            System.out.println(tm);
    }
}
```

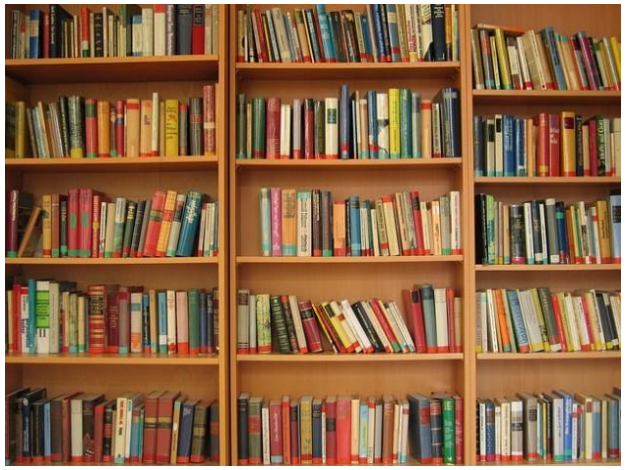
Saída do Programa

```
A referência do Técnico não pode ser nula!
Time [nome=Brasil, tecnico=Tecnico [nome=Dunga]]
Time [nome=Brasil, tecnico=Tecnico [nome=Felipão]]
```

Agregação: Multiplicidade 0..*

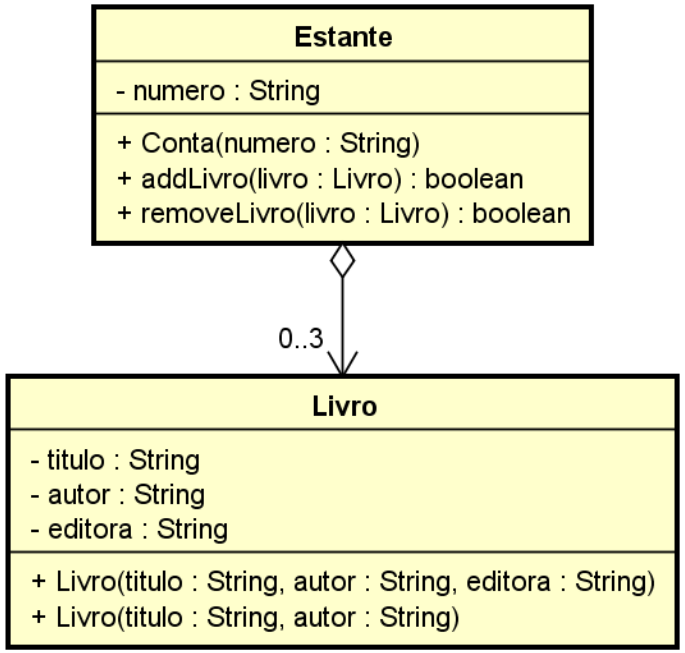
Exemplo: Uma Equipe e seus Atletas

Uma Equipe possui vários Atletas. Porém, deve existir antes de ser relacionado a Equipe.
Se um Atleta for desligado da Equipe, o Atleta ainda pode jogar em outra Equipe.

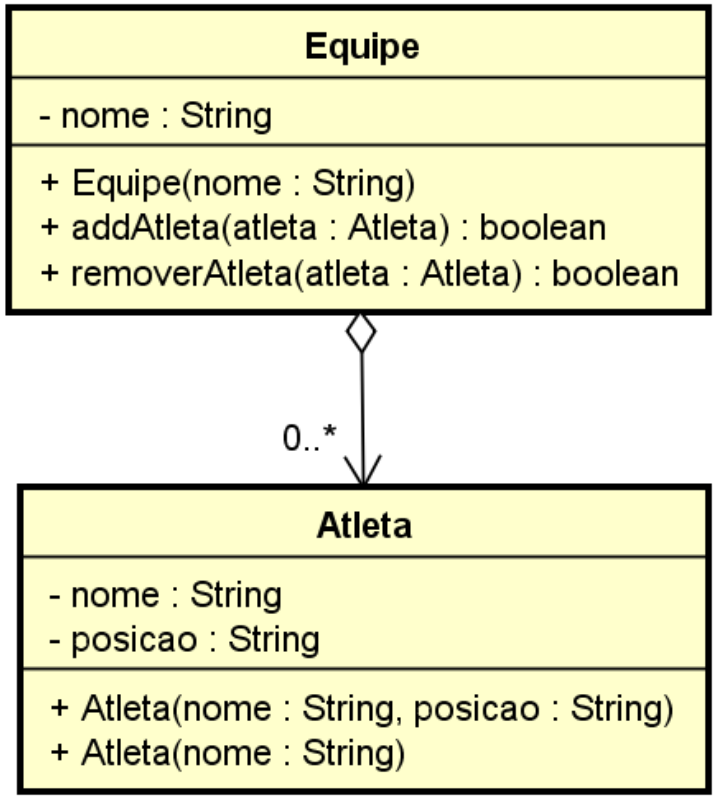


Na multiplicidade 0..*, o "todo" pode nascer sem possuir nenhuma parte.

- Ao longo de seu ciclo de vida, N "partes" podem agregar ao "todo", com o "todo" sabendo quais "partes" estarão se relacionando com ele.



Agregação: Multiplicidade 0..*



Implementação

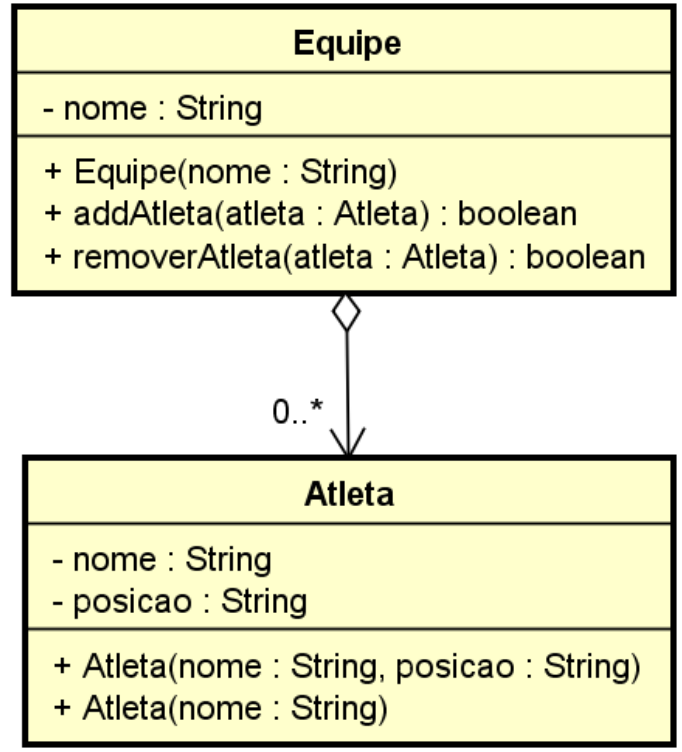
- Um Atleta agrega uma Equipe;
  - A Equipe pode ter muitos Atletas;
- O vínculo se dará no método addAtleta();
- Primeiro programe as partes, depois o relacionamento;
  - Crie o Atleta, para depois vincular a Equipe.

É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 0..*
Implementando a Classe Atleta

```java
package br.com.renanrodrigues.agregacao;

public class Atleta {

    private String nome;
    private String posicao;

    public Atleta(String nome, String posicao) {
        this.nome = nome;
        this.posicao = posicao;
    }

    public Atleta(String nome) {
        this.nome = nome;
    }
```

CONTINUA >>

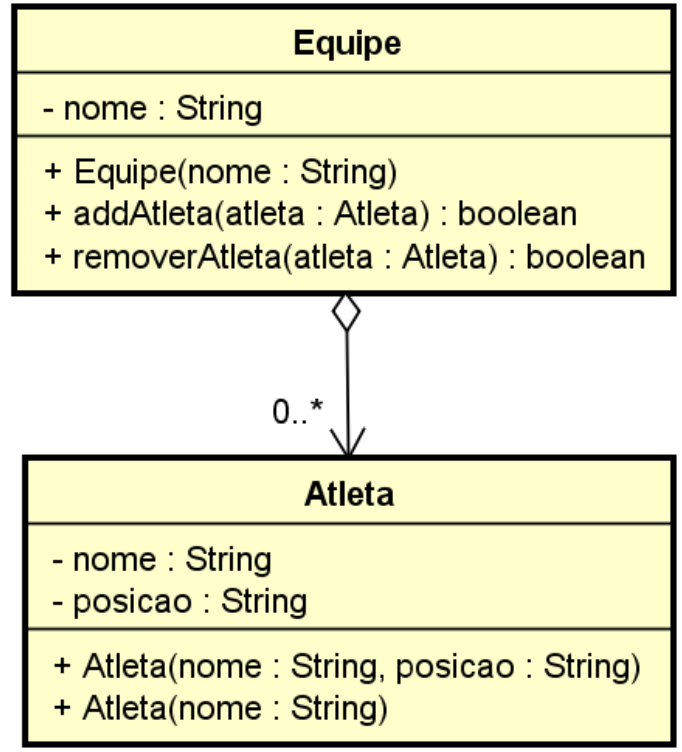


Agregação: Multiplicidade 0..*

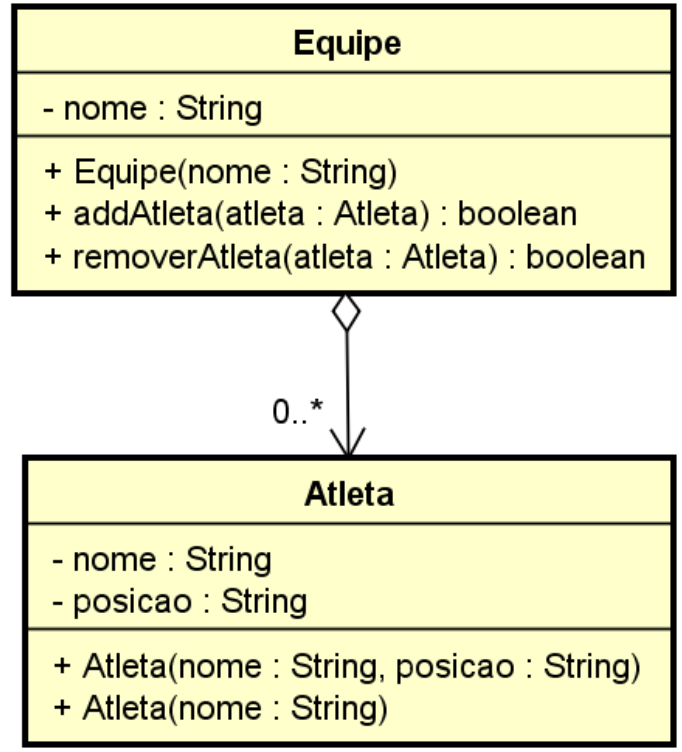
Implementando a Classe Atleta

```
17      public String getNome() {
18          return nome;
19      }
20
21      public String getPosicao() {
22          return posicao;
23      }
24
25      public void setPosicao(String posicao) {
26          this.posicao = posicao;
27      }
```

CONTINUA >>



Agregação: Multiplicidade 0..*
Implementando a Classe Atleta




```
29      @Override
30      public boolean equals(Object obj) {
31          if (this == obj)
32              return true;
33          if (obj == null)
34              return false;
35          if (getClass() != obj.getClass())
36              return false;
37          Atleta other = (Atleta) obj;
38          if (nome == null) {
39              if (other.nome != null)
40                  return false;
41          } else if (!nome.equals(other.nome))
42              return false;
43          return true;
44      }
```

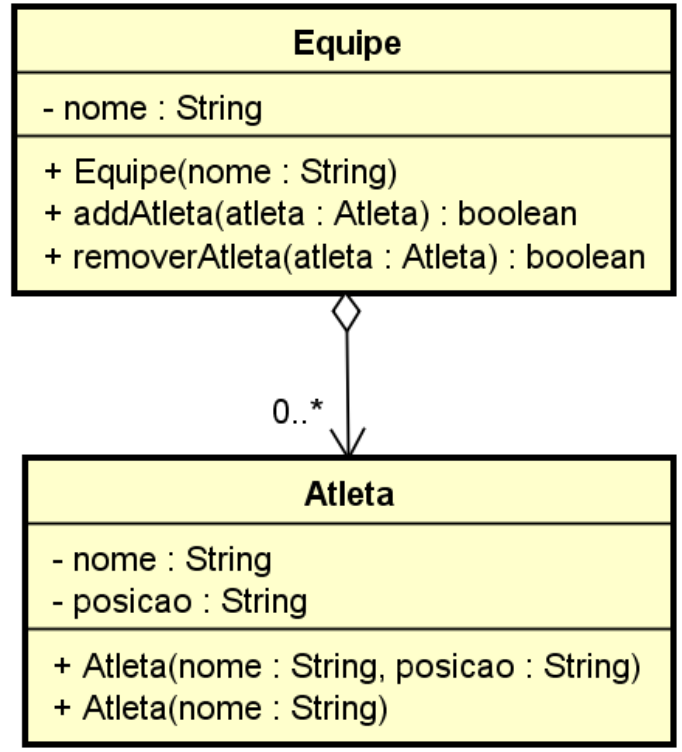
```
46      @Override
47      public String toString() {
48          return "Atleta [nome=" + nome + ", posicao=" + posicao + "]";
49      }
50  }
```

Agregação: Multiplicidade 0..*

Implementando a Classe Equipe

```java
package br.com.renanrodrigues.agregacao;

import java.util.ArrayList;
import java.util.List;

public class Equipe {

    private String nome;
    private List<Atleta> listaAtleta = new ArrayList<Atleta>();

    public Equipe(String nome) {
        this.nome = nome;
    }
```

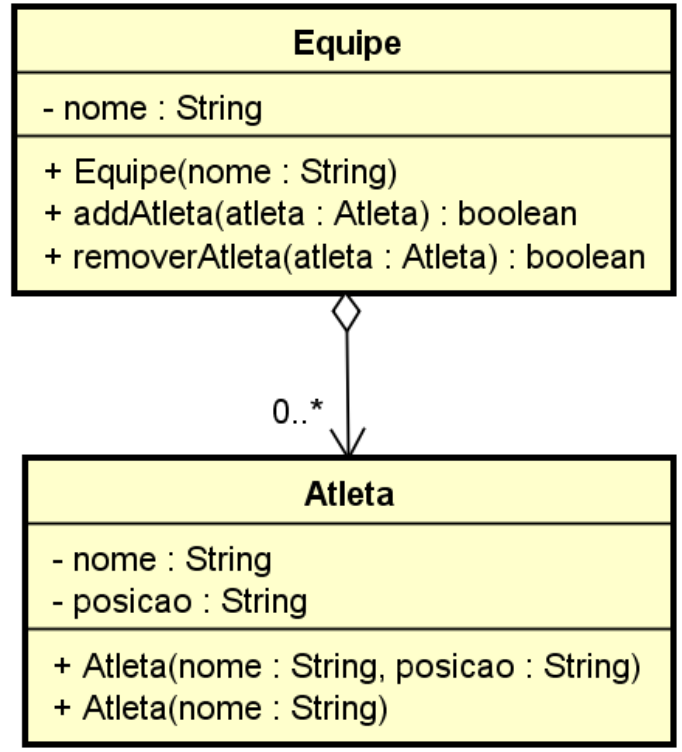
CONTINUA >>



Agregação: Multiplicidade 0..*

Implementando a Classe Equipe

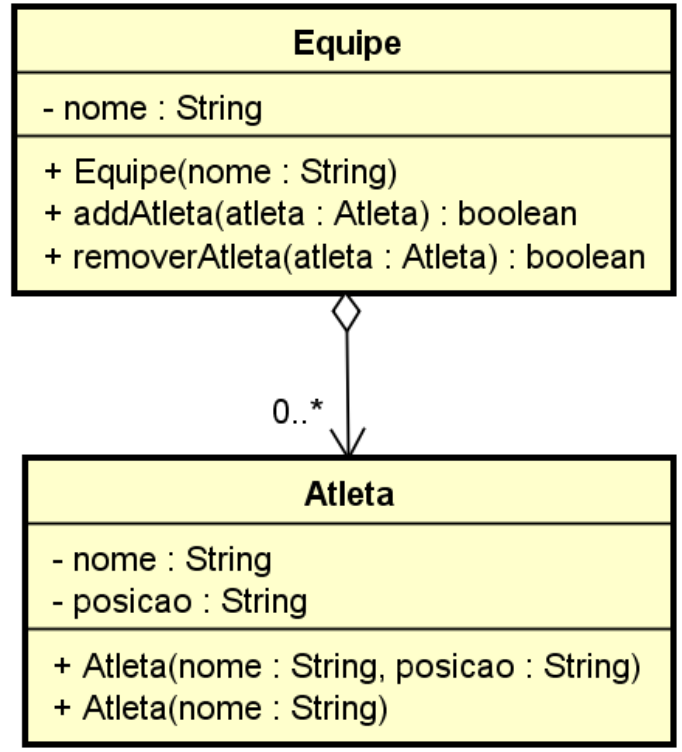
```
15    public boolean addAtleta(Atleta atleta) {
16        boolean sucesso = false;
17
18        if (atleta!= null && !listaAtleta.contains(atleta)) {
19            listaAtleta.add(atleta);
20            sucesso = true;
21        }
22        return sucesso;
23    }
25    public boolean removerAtleta(Atleta a) {
26        boolean sucesso = false;
27
28        if (listaAtleta.size() > 0 && listaAtleta.contains(a)) {
29            listaAtleta.remove(a);
30            sucesso = true;
31        }
32
33        return sucesso;
34    }
35
```



CONTINUA >>

Agregação: Multiplicidade 0..*
Implementando a Classe Equipe

```
36      public String getNome() {
37          return nome;
38      }
39
40      public void setNome(String nome) {
41          this.nome = nome;
42      }


44      public List<Atleta> getListaAtleta() {
45          return listaAtleta;
46      }
47
48      @Override
49      public String toString() {
50          return "Time [nome=" + nome + ", listaAtleta=" + listaAtleta + "]";
51      }
52  }
```



Agregação: Multiplicidade 0..*

Programa Principal

```java
package br.com.renanrodrigues.agregacao;

public class TesteEquipe {

    public static void main(String[] args) {

        Atleta a1 = new Atleta("Renan", "Goleiro");
        Atleta a2 = new Atleta("Rafael Foster", "Zagueiro");
        Atleta a3 = new Atleta("Renan", "Goleiro");
        Atleta a4 = null;
        Atleta a5 = new Atleta("Felipe Menezes", "Meia");

        Equipe t = new Equipe("Goiás");

        t.addAtleta(a1);
        t.addAtleta(a2);
        t.addAtleta(a3);
        t.addAtleta(a4);
        System.out.println(t);

        t.addAtleta(a5);
        System.out.println(t);

        t.removerAtleta(a2);
        System.out.println(t);
    }
}
```

Saída do Programa

```
Time [nome=Goiás, listaAtleta=[Atleta [nome=Renan, posicao=Goleiro], Atleta [nome=Rafael Foster, posicao=Zagueiro]]]
Time [nome=Goiás, listaAtleta=[Atleta [nome=Renan, posicao=Goleiro], Atleta [nome=Rafael Foster, posicao=Zagueiro],
                               Atleta [nome=Felipe Menezes, posicao=Meia]]]
Time [nome=Goiás, listaAtleta=[Atleta [nome=Renan, posicao=Goleiro], Atleta [nome=Felipe Menezes, posicao=Meia]]]
```

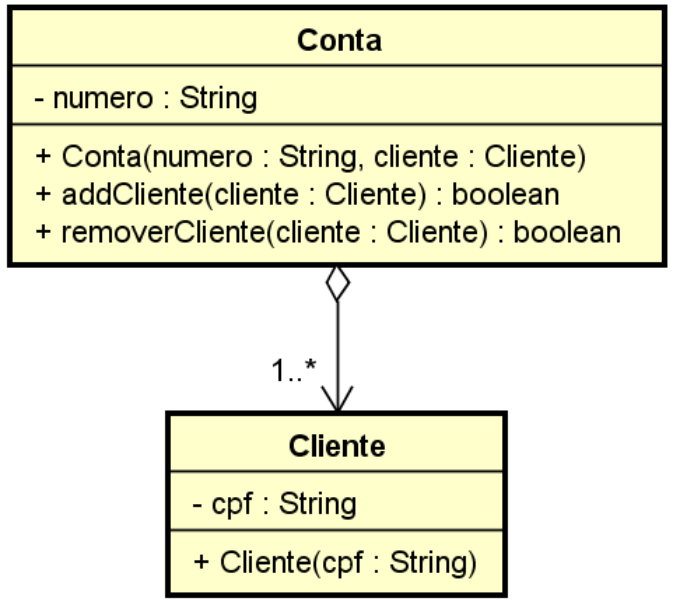
Agregação: Multiplicidade 1..*

Exemplo: Uma Conta e seus Cliente

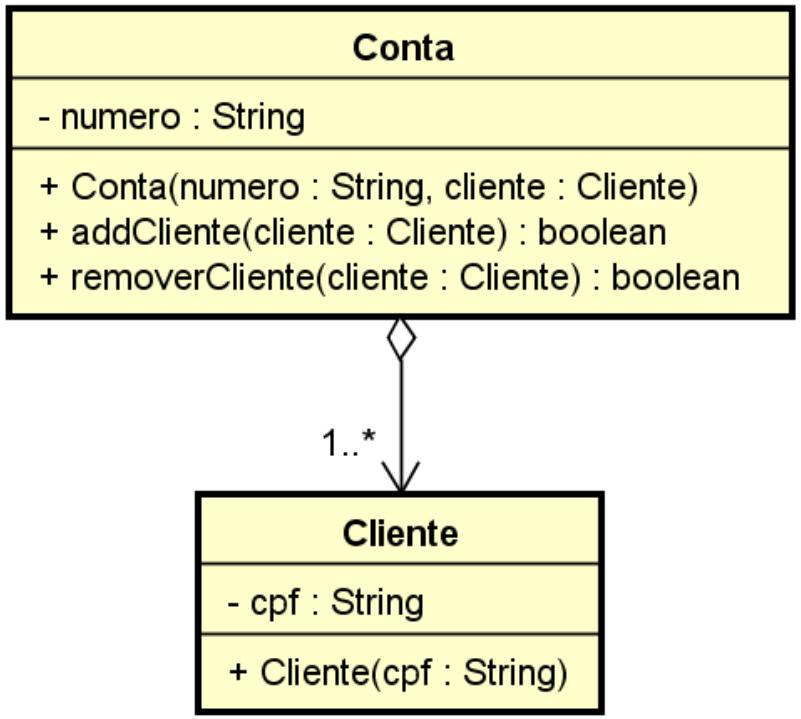
Uma Conta possui um ou mais Clientes. Se um Cliente for retirado da Conta, o Cliente ainda existe e pode abrir em outra Conta.

Na multiplicidade 1..*, o "todo" DEVE nascer possuindo uma parte.

- Ao longo de seu ciclo de vida, N "partes" podem agregar ao "todo", com o "todo" sabendo quais "partes" estarão se relacionando com ele.



Agregação: Multiplicidade 1..*



Implementação

- Um Cliente agrega uma Conta;
  - A Conta pode ter um ou mais Clientes;
- O vínculo se dará no método addCliente();
- Primeiro programe as partes, depois o relacionamento;
  - Crie o Cliente, para depois vincular a Conta.

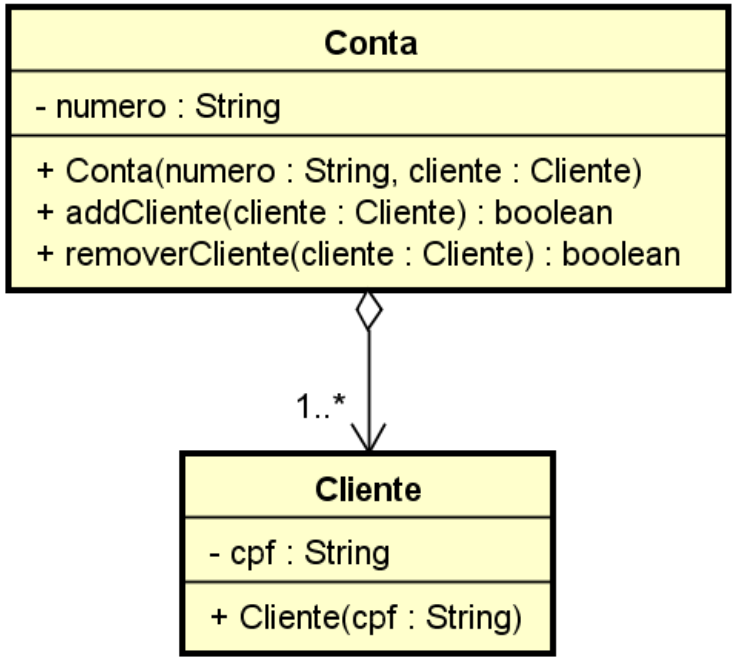
É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 1..*

Implementando a Classe Cliente

```java
package br.com.renanrodrigues.agregacao;

public class Cliente {

    private String cpf;

    public Cliente(String cpf) {
        this.cpf = cpf;
    }

    public String getCPF() {
        return cpf;
    }
```

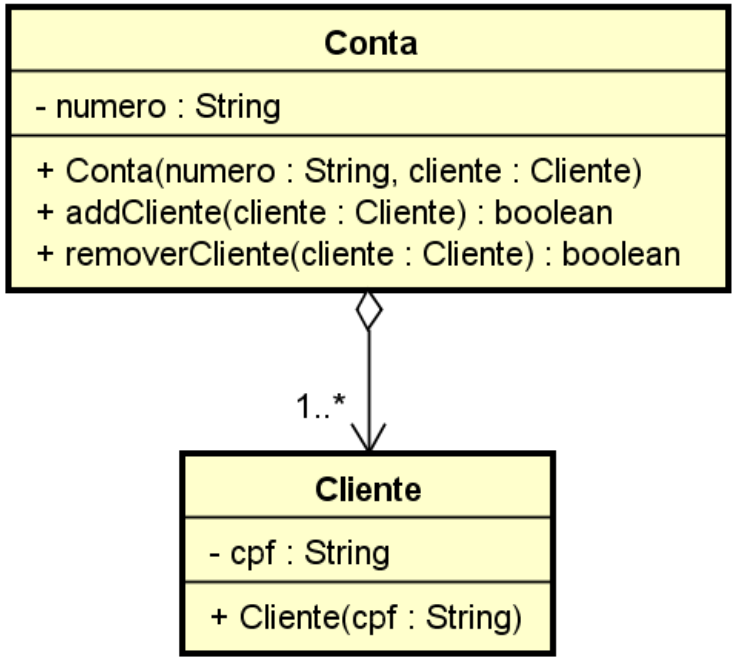
CONTINUA >>



Agregação: Multiplicidade 1..*

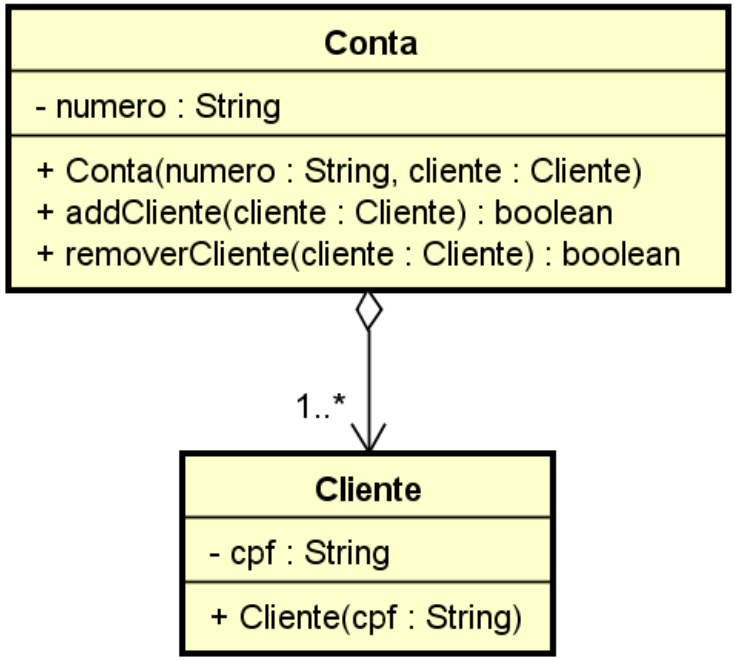
Implementando a Classe Cliente

```
15      @Override
16      public boolean equals(Object obj) {
17          if (this == obj)
18              return true;
19          if (obj == null)
20              return false;
21          if (getClass() != obj.getClass())
22              return false;
23          Cliente other = (Cliente) obj;
24          if (cpf == null) {
25              if (other.cpf != null)
26                  return false;
27          } else if (!cpf.equals(other.cpf))
28              return false;
29          return true;
30      }
```




```
32      @Override
33      public String toString() {
34          return "Cliente [cpf=" + cpf + "]";
35      }
36  }
```

Agregação: Multiplicidade 1..*
Implementando a Classe Conta

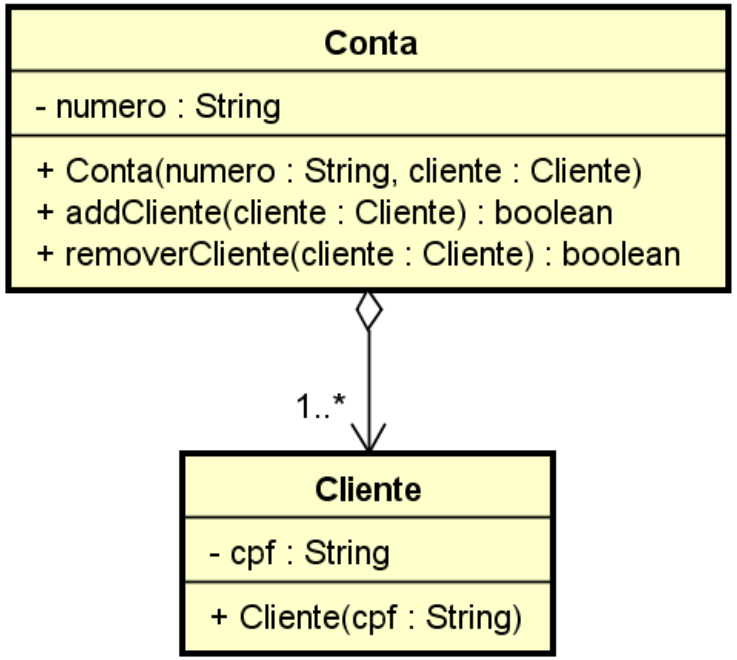



```
21    public boolean addCliente(Cliente titular) {
22        boolean sucesso = false;
23
24        if (titular!= null) {
25            listaCliente.add(titular);
26            sucesso = true;
27        }
28
29        return sucesso;
30    }
```

```
32    public boolean removerCliente(Cliente a) {
33        boolean sucesso = false;
34
35        if (listaCliente.size() > 1 && listaCliente.contains(a)) {
36            listaCliente.remove(a);
37            sucesso = true;
38        }
39
40        return sucesso;
41    }
```

CONTINUA >>

Agregação: Multiplicidade 1..*
Implementando a Classe Conta




```
43      public String getNumero() {
44          return numero;
45      }
46
47      public List<Cliente> getListaCliente() {
48          return listaCliente;
49      }
50
51      @Override
52      public String toString() {
53          return "Conta [numero=" + numero + ", listaCliente=" + listaCliente + "]";
54      }
55  }
```

Agregação: Multiplicidade 1..*

Programa Principal

```java
package br.com.renanrodrigues.agregacao;

public class TesteConta {

    public static void main(String[] args) {

        Conta c;
        Cliente t1 = new Cliente("111");
        Cliente t2 = new Cliente("222");
        Cliente t3 = null;
        Cliente t4 = new Cliente("333");

        try {
            c = new Conta("ABC", t3);
        } catch (NullPointerException e) {
            System.err.println(e.getMessage());
        }
        c = new Conta("ABC", t1);
        c.addCliente(t2);
        c.addCliente(t4);
        System.out.println(c);

        c.removerCliente(t1);
        c.removerCliente(t4);
        System.out.println(c);

        c.removerCliente(t2);
        System.out.println(c);
    }
}
```

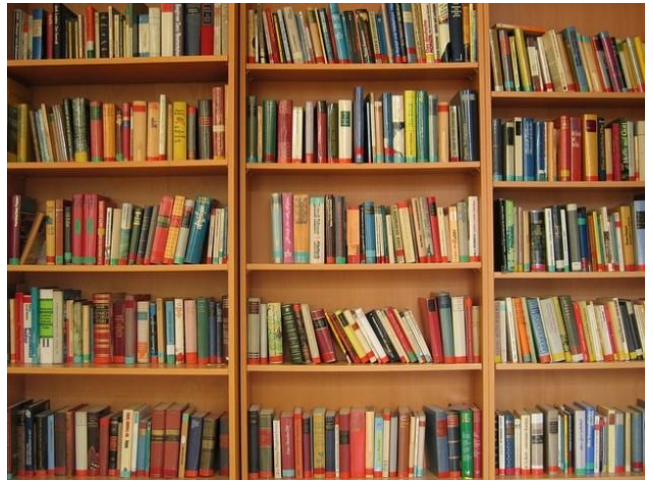
Saída do Programa

```
A referência do Cliente não pode ser nula!
Conta [numero=ABC, listaCliente=[Cliente [cpf=111], Cliente [cpf=222], Cliente [cpf=333]]]
Conta [numero=ABC, listaCliente=[Cliente [cpf=222]]]
Conta [numero=ABC, listaCliente=[Cliente [cpf=222]]]
```

Agregação: Multiplicidade 0..N

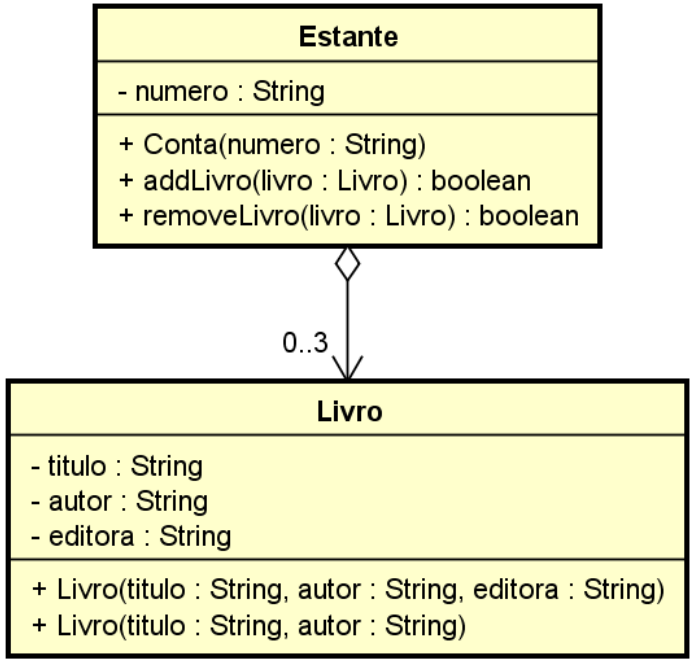
Exemplo: Uma Estante e seus Livros

Uma Estante possui vários Livros, podendo comportar uma quantidade finita. Se um Livro for retirado da Estante, o Livro ainda continua existindo.

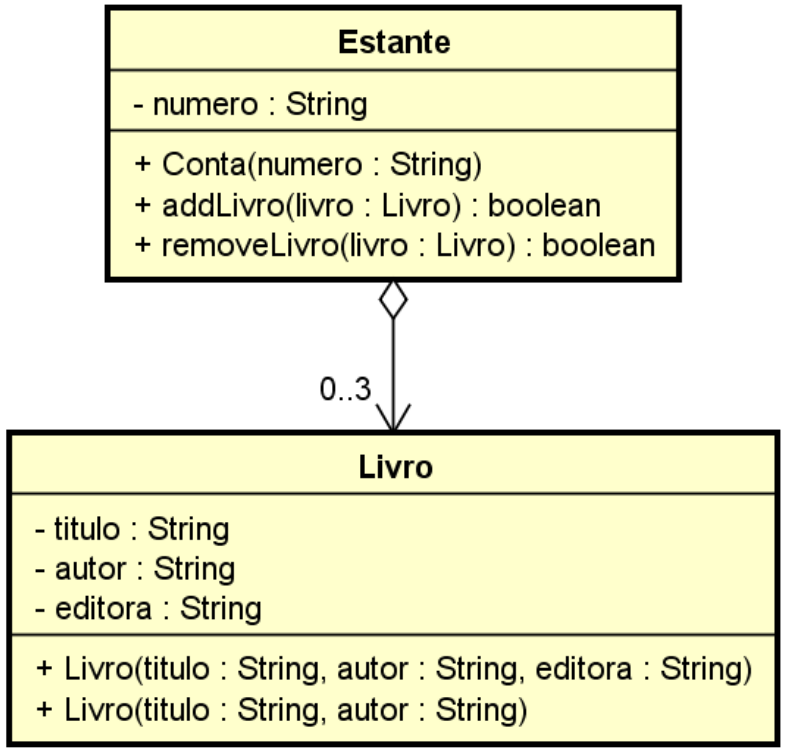


Na multiplicidade 0..N, o "todo" pode nascer sem possuir nenhuma parte.

- Ao longo de seu ciclo de vida, N "partes" podem agregar ao "todo", com o "todo" sabendo quais "partes" estarão se relacionando com ele.



Agregação: Multiplicidade 0..N



Implementação

- Um Livro agrega uma Estante;
  - A estante pode ter um ou mais Livro (no máximo 3);
- O vínculo se dará no método addLivro();
- Primeiro programe as partes, depois o relacionamento;
  - Crie o Livro, para depois vincular a Estante.

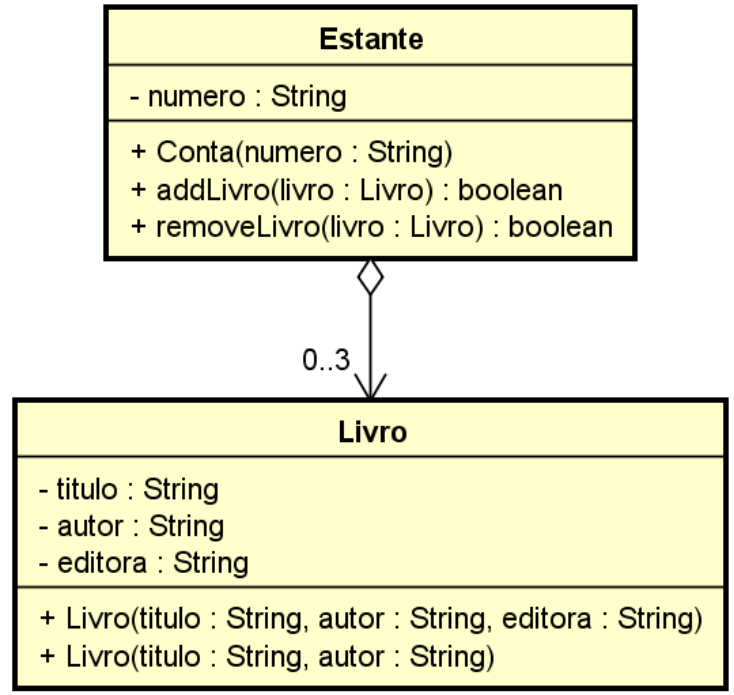
É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 0..N

Implementando a Classe Livro

```java
package br.com.renanrodrigues.agregacao;

public class Livro {

    private String titulo;
    private String autor;
    private String editora;

    public Livro(String titulo, String autor, String editora) {
        this.titulo = titulo;
        this.autor = autor;
        this.editora = editora;
    }

    public Livro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
    }
```

CONTINUA >>

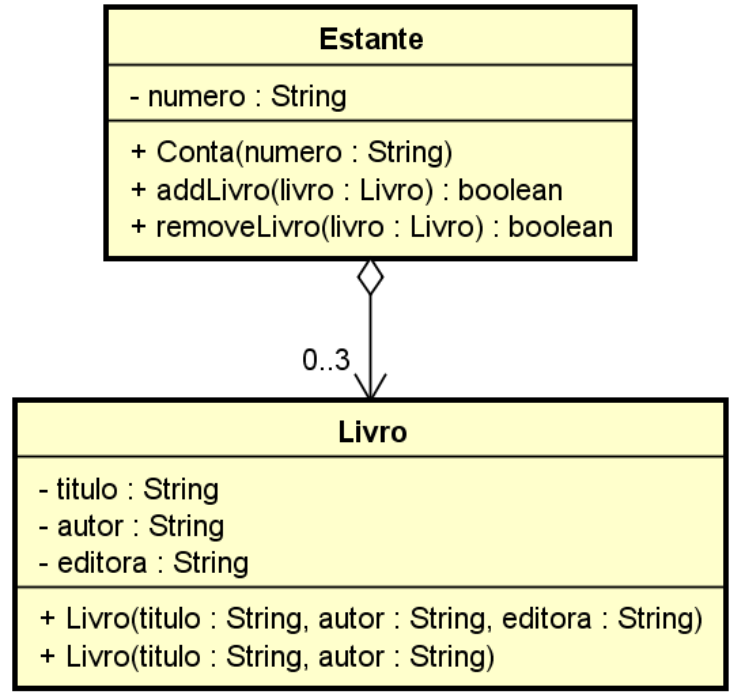


Agregação: Multiplicidade 0..N

Implementando a Classe Livro

```
20      public String getTitulo() {
21          return titulo;
22      }
23
24      public String getAutor() {
25          return autor;
26      }
27
28      public String getEditora() {
29          return editora;
30      }
```

CONTINUA >>

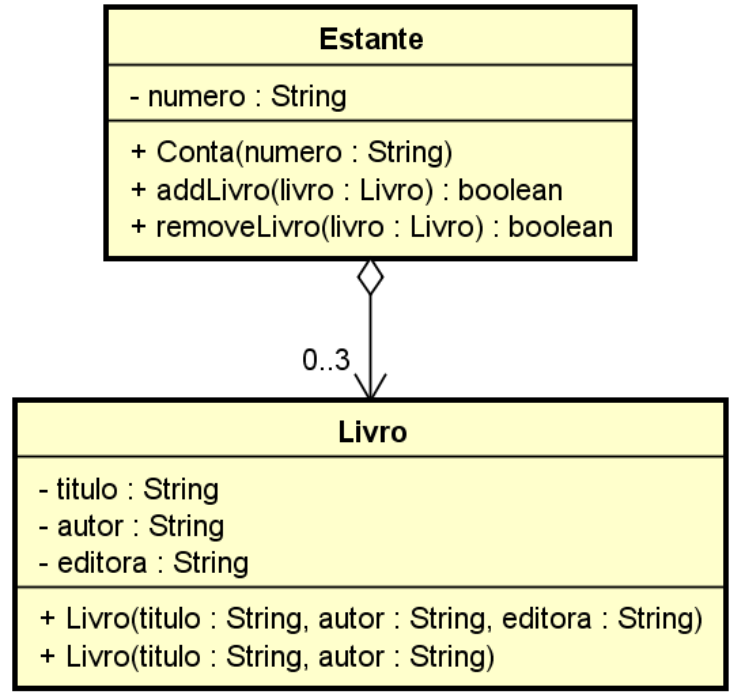


Agregação: Multiplicidade 0..N

Implementando a Classe Livro

```java
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Livro other = (Livro) obj;
        if (autor == null) {
            if (other.autor != null)
                return false;
        } else if (!autor.equals(other.autor))
            return false;
        if (editora == null) {
            if (other.editora != null)
                return false;
        } else if (!editora.equals(other.editora))
            return false;
        if (titulo == null) {
            if (other.titulo != null)
                return false;
        } else if (!titulo.equals(other.titulo))
            return false;
        return true;
    }
```

```java
    @Override
    public String toString() {
        return "Livro [titulo=" + titulo + ", autor=" + autor + ", editora="
                + editora + "]";
    }
}
```

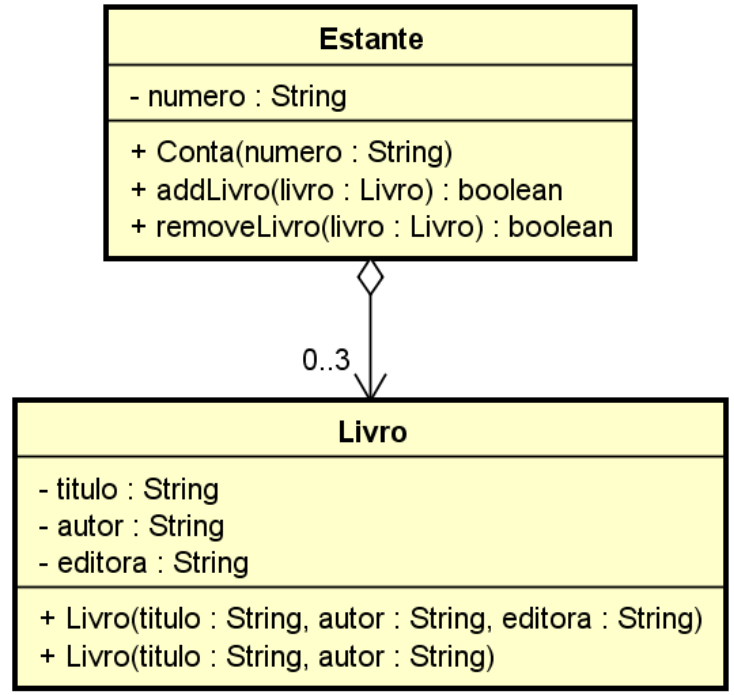


Agregação: Multiplicidade 0..N

Implementando a Classe Estante

```java
package br.com.renanrodrigues.agregacao;

import java.util.ArrayList;
import java.util.List;

public class Estante {

    private String numero;
    private List<Livro> listaLivro = new ArrayList<Livro>();

    public Estante(String numero) {
        this.numero = numero;
    }
```

CONTINUA >>

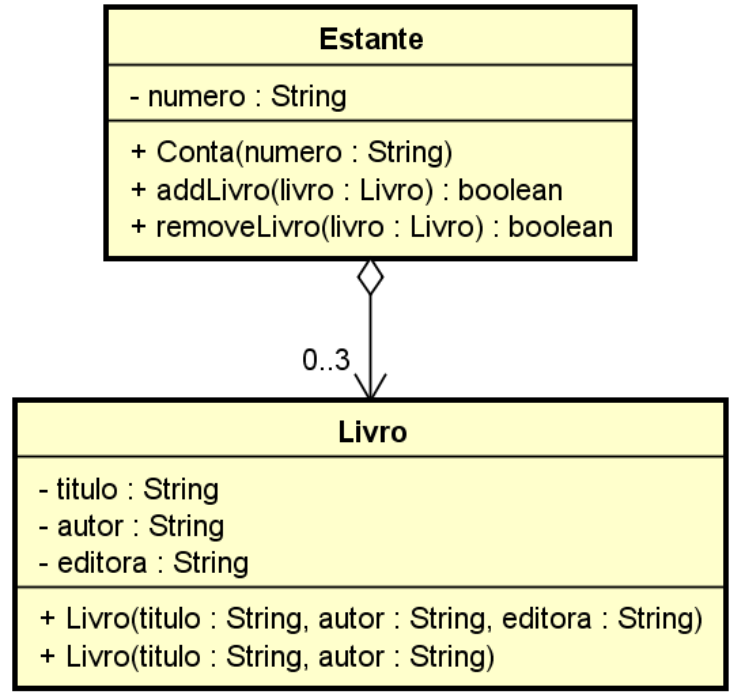


Agregação: Multiplicidade 0..N

Implementando a Classe Estante

```
15    public boolean addLivro(Livro livro) {
16        boolean sucesso = false;
17
18        if (livro!= null && listaLivro.size() < 3 ) {
19            listaLivro.add(livro);
20            sucesso = true;
21        }
22        return sucesso;
23    }

25    public boolean removerLivro(Livro livro) {
26        boolean sucesso = false;
27
28        if (listaLivro.size() > 0 && listaLivro.contains(livro)) {
29            listaLivro.remove(livro);
30            sucesso = true;
31        }
32
33        return sucesso;
34    }
```

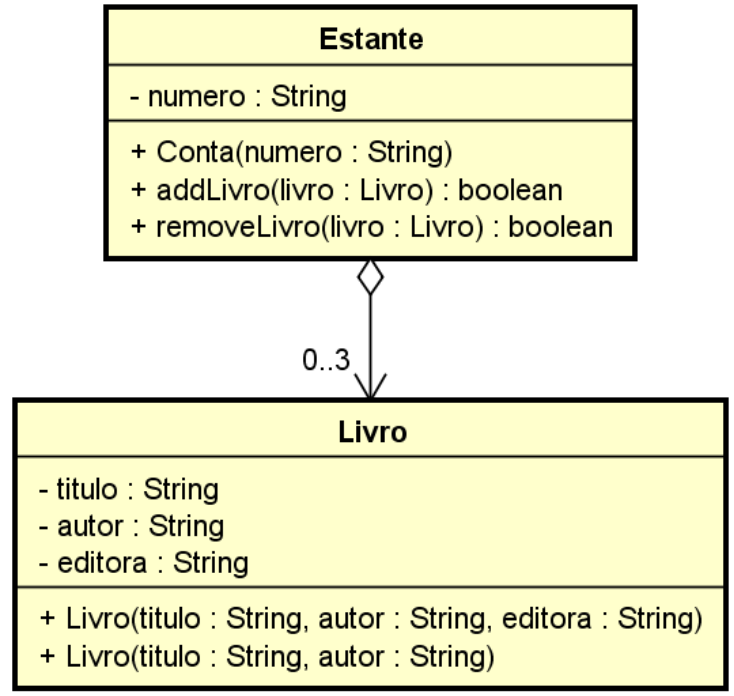
CONTINUA >>



Agregação: Multiplicidade 0..N

Implementando a Classe Estante

```
36      public String getNumero() {
37          return numero;
38      }
39
40      public void setNumero(String numero) {
41          this.numero = numero;
42      }
43
44      public List<Livro> getListaLivro() {
45          return listaLivro;
46      }
47
48      @Override
49      public String toString() {
50          return "Estante [numero=" + numero + ", listaLivro=" + listaLivro + "]";
51      }
52  }
```



Agregação: Multiplicidade 0..N
Programa Principal

```java
package agregacao;

public class TesteEstante {

    public static void main(String[] args) {

        Livro lv1 = new Livro("Introdução à Programação Usando Java", "Rafael Santos", "Câmpus");
        Livro lv2 = new Livro("Utilizando UML e Padrões", "Larman", "Bookman");
        Livro lv3 = new Livro("Java: Como Programar", "Deitel", "Pearson");
        Livro lv4 = new Livro("Programação Orientada a Objetos com Java", "Barnes", "Pearson");

        Estante e = new Estante("AB-123");

        e.addLivro(lv1);
        e.addLivro(lv2);
        e.addLivro(lv3);
        e.addLivro(lv4);
        System.out.println(e);

        e.removerLivro(lv2);
        e.addLivro(lv4);
        System.out.println(e);

    }
}
```

Saída do Programa

```
Estante [numero=AB-123, listaLivro=[Livro [titulo=Introdução à Programação Usando Java, autor=Rafael Santos, editora=Câmpus],
                                    Livro [titulo=Utilizando UML e Padrões, autor=Larman, editora=Bookman],
                                    Livro [titulo=Java: Como Programar, autor=Deitel, editora=Pearson]]]
Estante [numero=AB-123, listaLivro=[Livro [titulo=Introdução à Programação Usando Java, autor=Rafael Santos, editora=Câmpus],
                                    Livro [titulo=Java: Como Programar, autor=Deitel, editora=Pearson],
                                    Livro [titulo=Programação Orientada a Objetos com Java, autor=Barnes, editora=Pearson]]]
```