

INSTITUTO FEDERAL
Goiás

Instituto Federal de Goiás
Câmpus Goiânia

Bacharelado em Sistemas de Informação
Disciplina: Programação Orientada a Objetos I

Associação de Classes Agregação

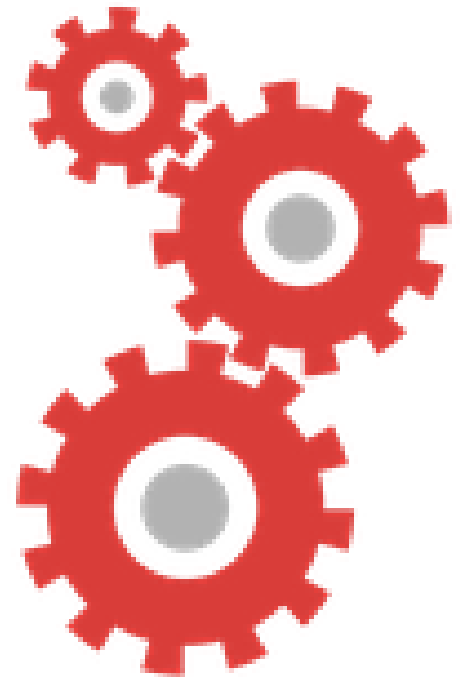
Prof. Ms. Renan Rodrigues de Oliveira
Goiânia - GO

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.

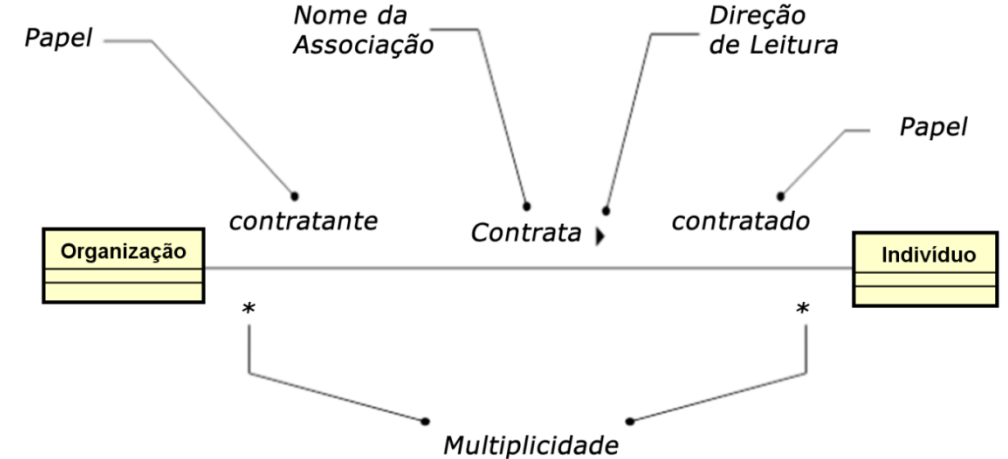
Podendo as Classes serem consideradas engrenagens ou peças de um quebra-cabeça, uma questão surge naturalmente:

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.

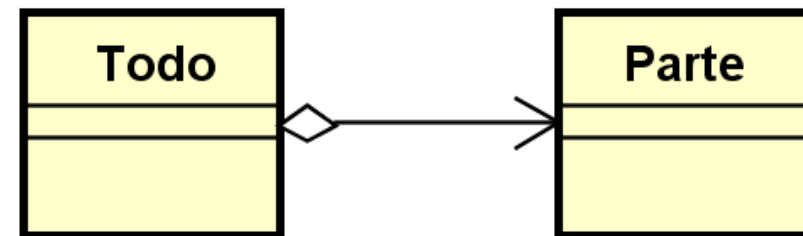
Agregação

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.

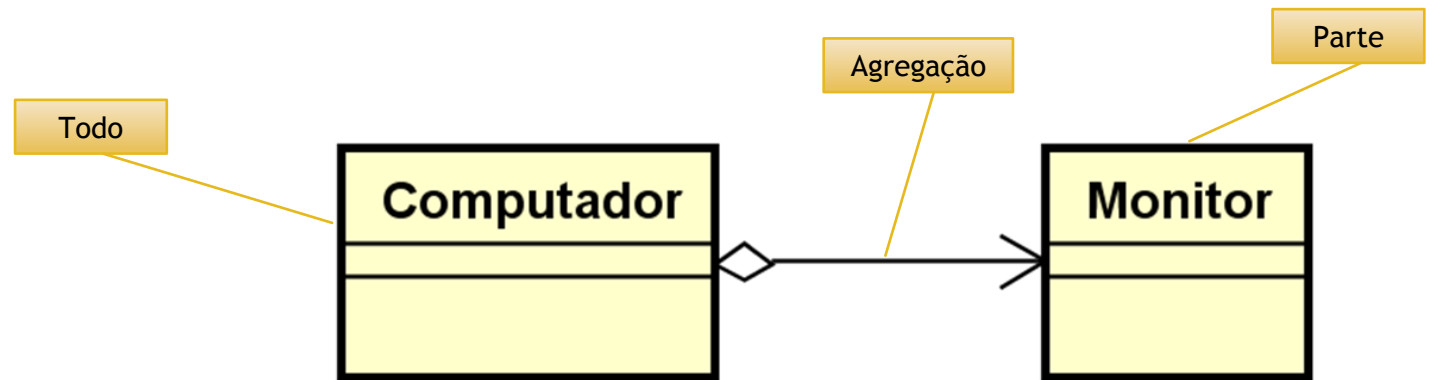
- ▶ Um objeto da classe parte integrante pode existir sem o todo, sendo que este último apenas agrega as partes já existentes;
- ▶ Tempo de vida da classe "parte" independente do tempo da classe "todo".



Exemplo: Agregação



Exemplo: O Computado e seu Monitor



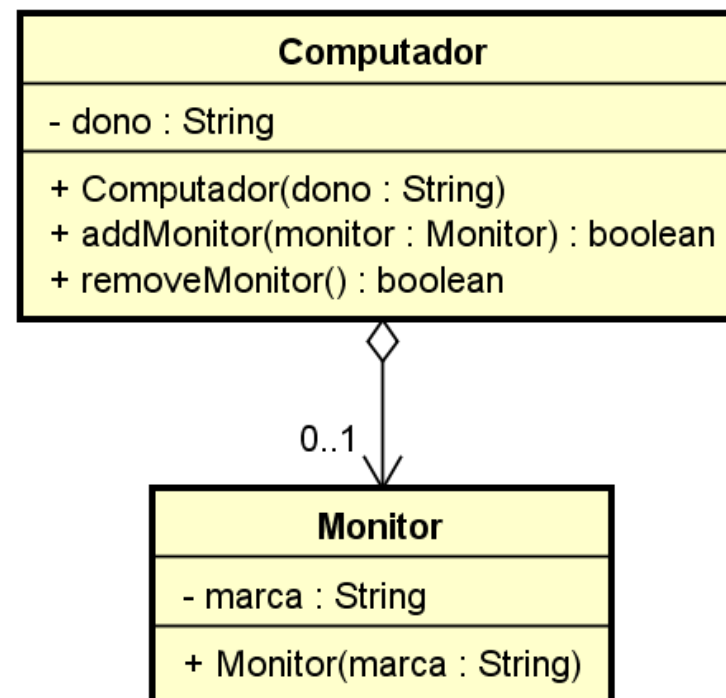
Para que a parte possa existir sem o todo, ela deve estar criada antes de estar agregada ao todo.

- ▶ 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.

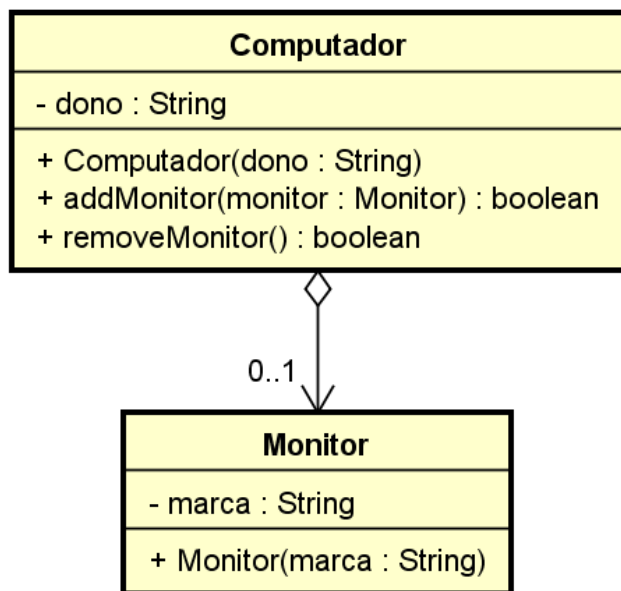


Agregação: Multiplicidade 0..1



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

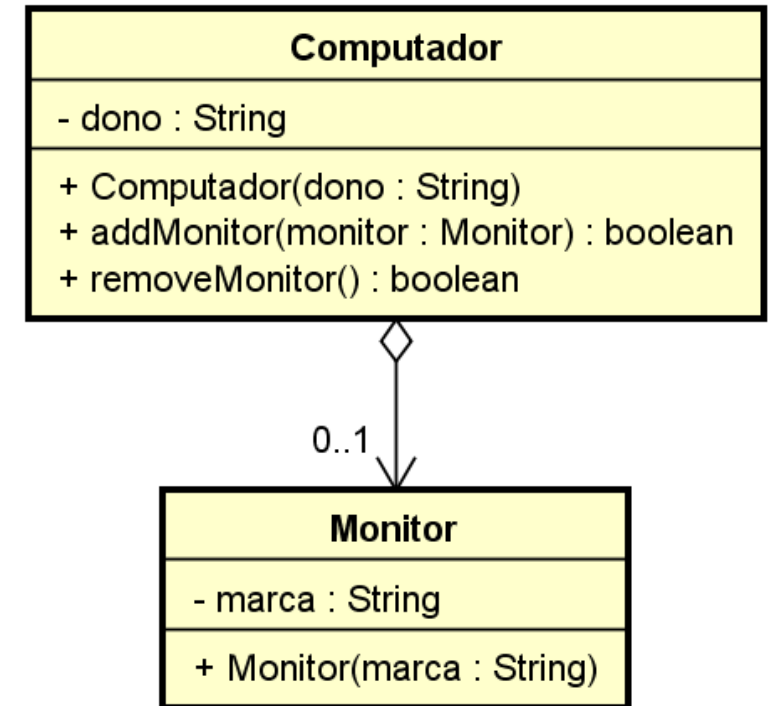
- ▶ Um Monitor agrega a um Computador;
 - ▶ O Computador pode ter 0 ou 1 Monitor;
 - ▶ O vínculo se dará no método addMonitor();
- ▶ Primeiro programe as partes, depois o relacionamento;
 - ▶ Crie o Monitor, para depois vincular ao Computador

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

Agregação: Multiplicidade 0..1

Implementando a Classe Monitor

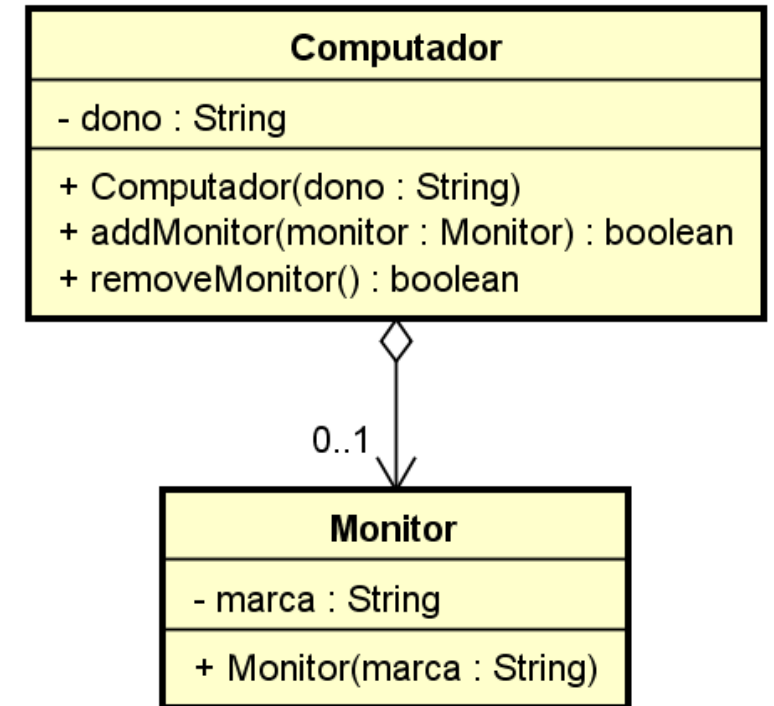
```
1 package br.com.renanrodrigues.agregacao;
2
3 public class Monitor {
4     private String marca;
5
6     public Monitor(String marca) {
7         this.marca = marca;
8     }
9
10    public String getMarca() {
11        return marca;
12    }
13
14    @Override
15    public String toString() {
16        return "Monitor [marca=" + marca + "]";
17    }
18 }
```



Agregação: Multiplicidade 0..1

Implementando a Classe Computador

```
1 package br.com.renanrodrigues.agregacao;
2
3 public class Computador {
4     private String dono;
5     private Monitor monitor;
6
7     public Computador(String dono) {
8         this.dono = dono;
9     }
10
11     public boolean addMonitor(Monitor monitor) {
12
13         boolean sucesso = false;
14
15         if (this.monitor == null) {
16             this.monitor = monitor;
17             sucesso = true;
18         }
19
20         return sucesso;
21     }
22 }
```

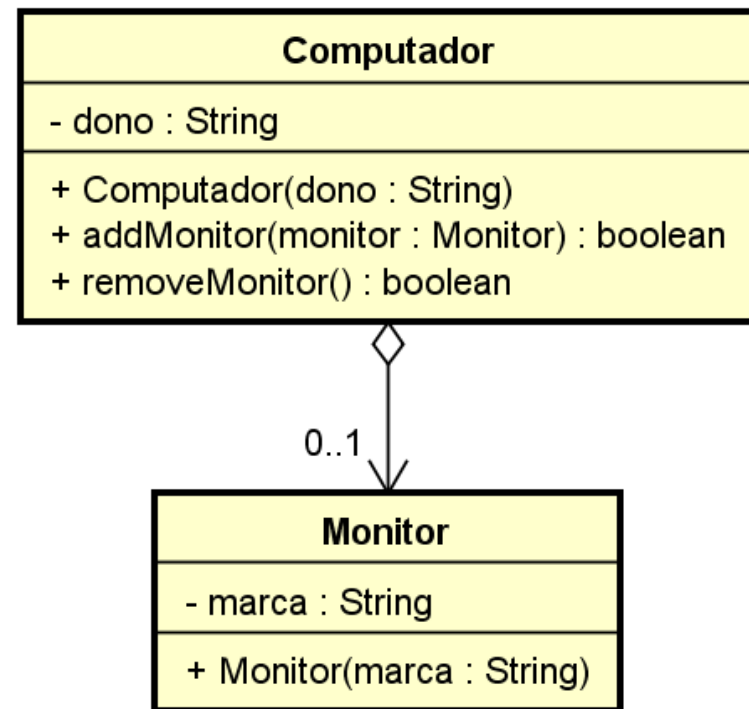


Agregação: Multiplicidade 0..1

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 >>

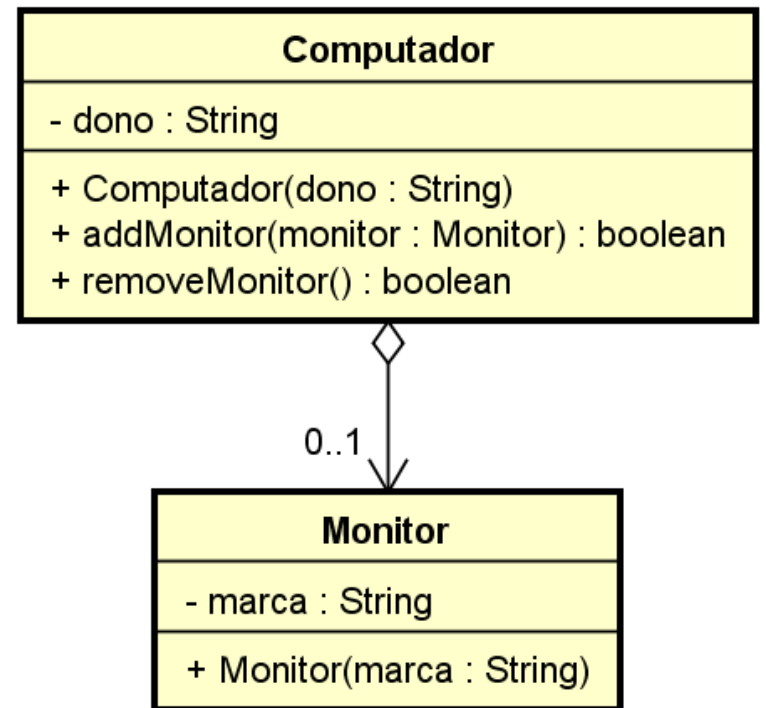


Agregação: Multiplicidade 0..1

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

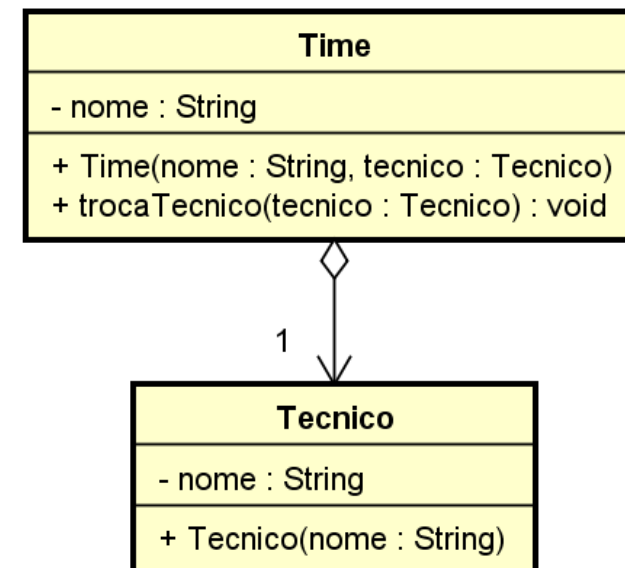
Exemplo: Um Time e seu Técnico

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

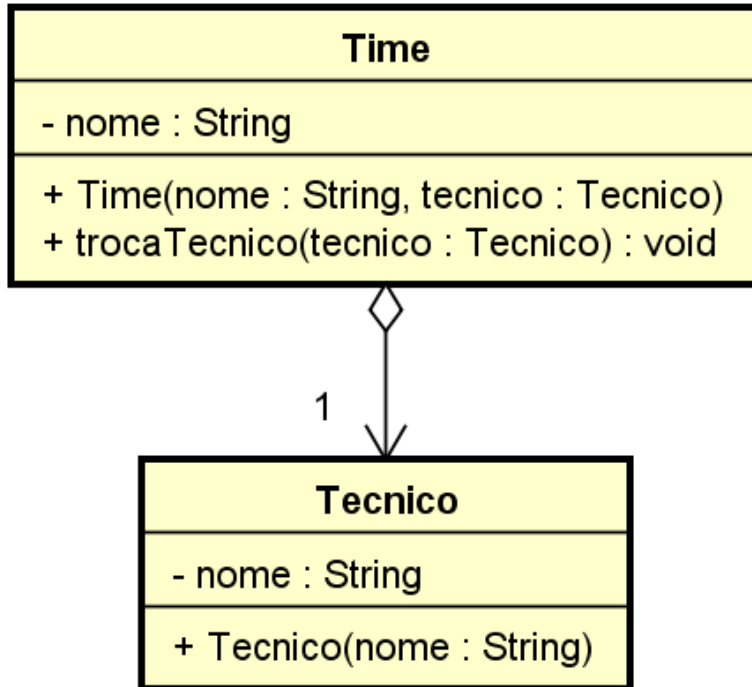


Na multiplicidade 1, o “todo” DEVE nascer possuindo uma parte.

- ▶ 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

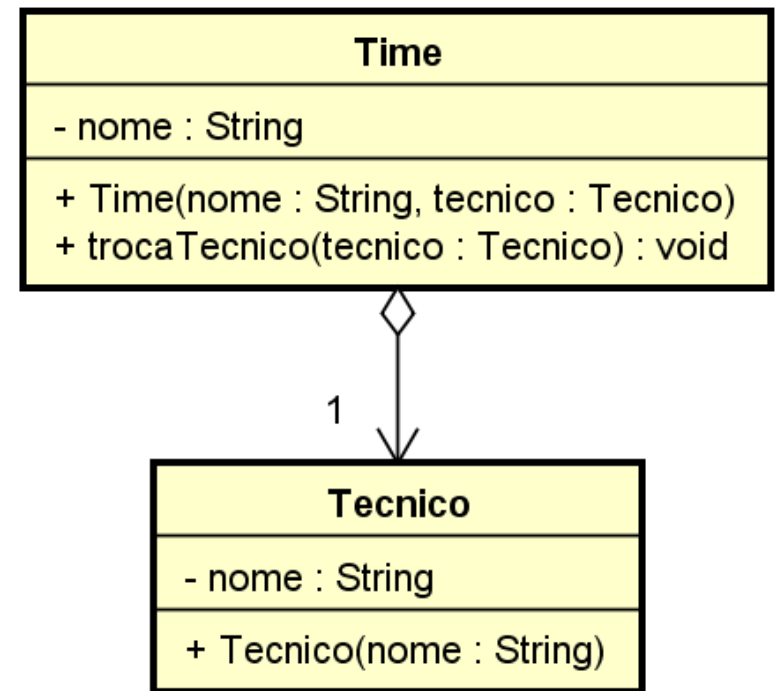
- ▶ Um Time possui um Técnico;
 - ▶ O vínculo se dará no construtor;
- ▶ Primeiro programe as partes, depois o relacionamento;
 - ▶ Crie o Técnico antes de criar o Time.

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

Agregação: Multiplicidade 1

Implementando a Classe Técnico

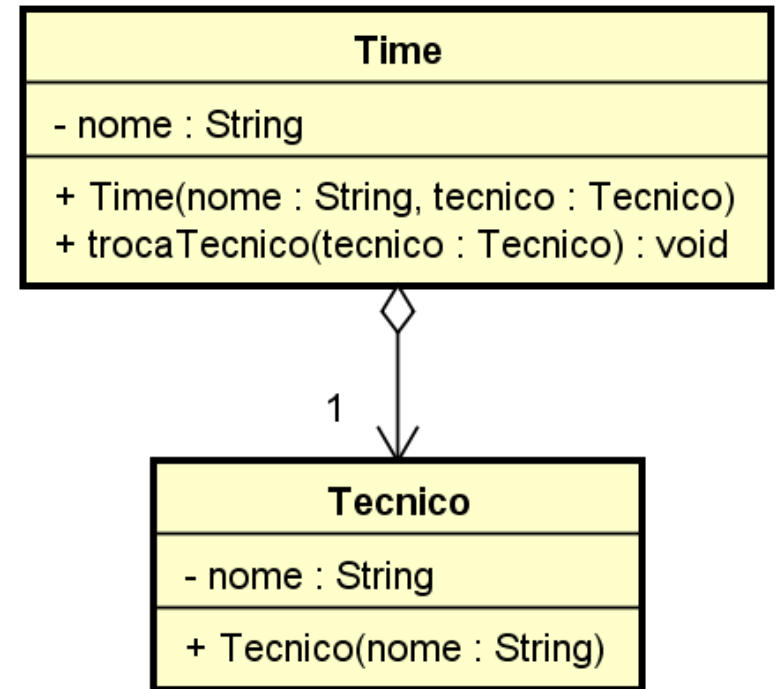
```
1 package br.com.renanrodrigues.agregacao;
2
3 public class Tecnico {
4
5     private String nome;
6
7     public Tecnico(String nome) {
8         this.nome = nome;
9     }
10
11     public String getNome() {
12         return nome;
13     }
14
15     @Override
16     public String toString() {
17         return "Tecnico [nome=" + nome + "]";
18     }
19 }
```



Agregação: Multiplicidade 1

Implementando a Classe Time

```
1 package br.com.renanrodrigues.agregacao;
2
3 public class Time {
4
5     private String nome;
6     private Tecnico tecnico;
7
8     public Time(String nome, Tecnico tecnico) {
9
10         if (tecnico == null) {
11             throw new NullPointerException("A referência do Técnico não pode ser nula!");
12         }
13
14         this.nome = nome;
15         this.tecnico = tecnico;
16     }
```



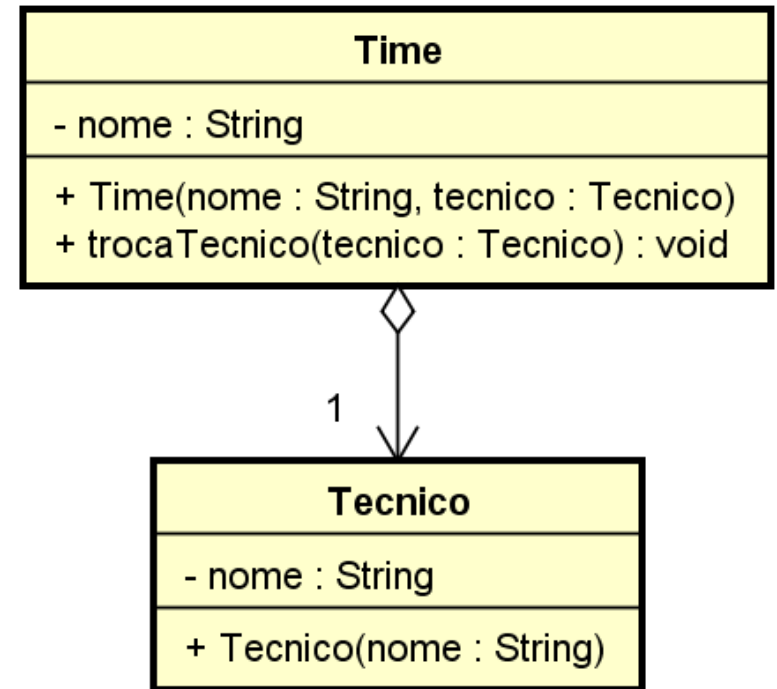
CONTINUA >>

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 }
```

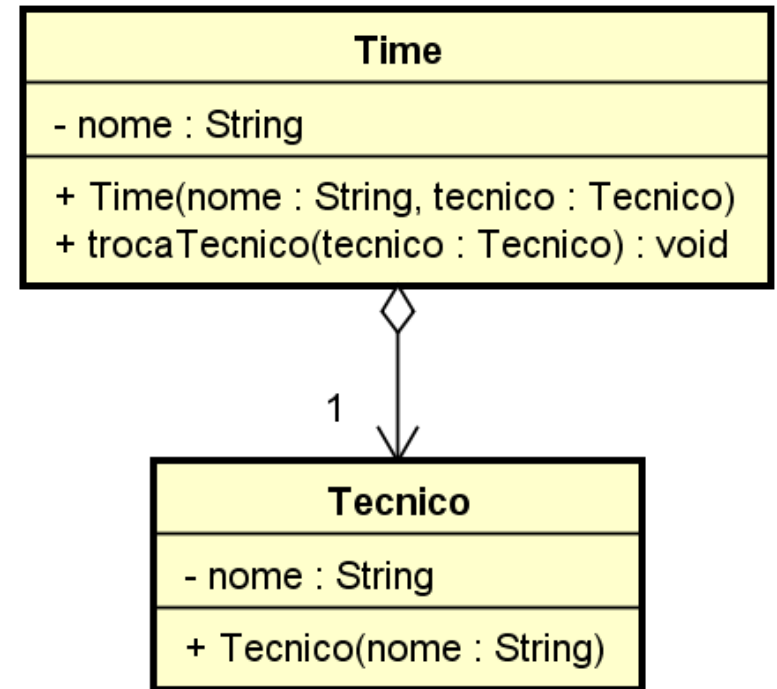
CONTINUA >>



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

```
1 package br.com.renanrodrigues.agregacao;
2
3 public class TesteTime {
4
5     public static void main(String[] args) {
6
7         Tecnico t1 = null;
8         Tecnico t2 = new Tecnico("Dunga");
9         Tecnico t3 = new Tecnico("Felipão");
10
11         Time tm;
12
13         try {
14             tm = new Time("Brasil", t1);
15         } catch (NullPointerException e) {
16             System.err.println(e.getMessage());
17         }
18
19         tm = new Time("Brasil", t2);
20         System.out.println(tm);
21
22         tm.trocaTecnico(t3);
23         System.out.println(tm);
24     }
25 }
```

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..*

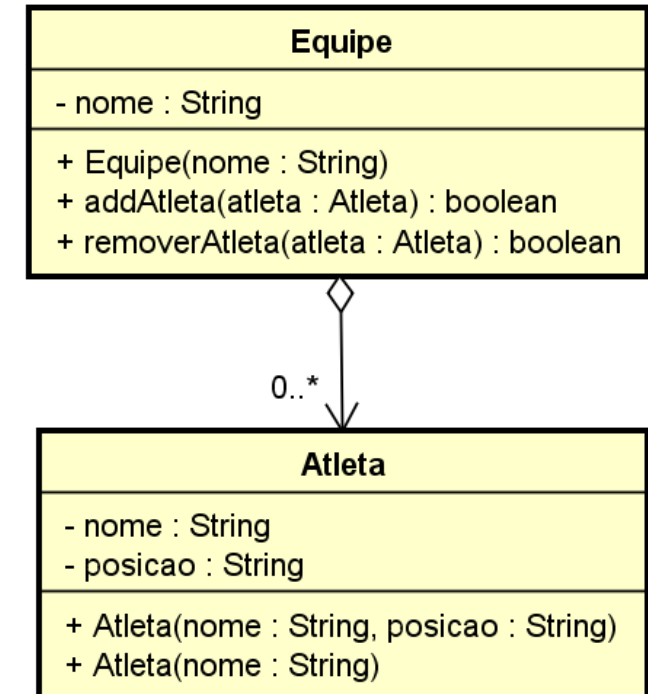
Exemplo: Uma Equipe e seus Atletas

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.

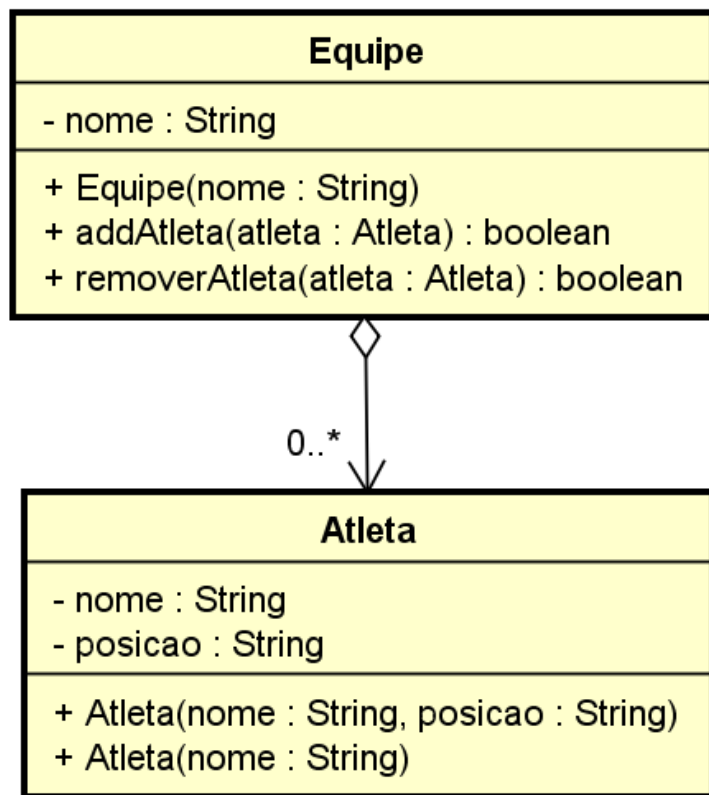


Na multiplicidade 0..*, o “todo” pode nascer sem possuir nenhuma parte.

- ▶ 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.

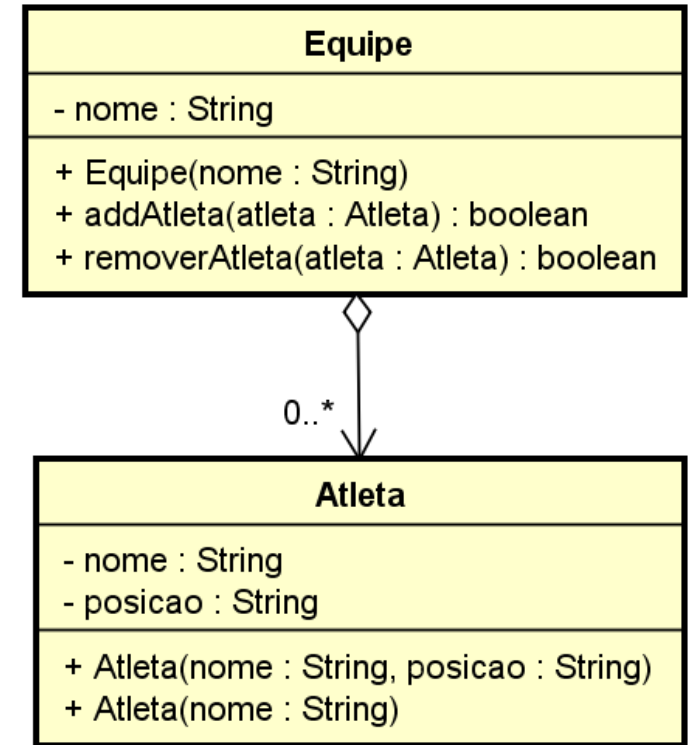
É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 0..*

Implementando a Classe Atleta

```
1 package br.com.renanrodrigues.agregacao;  
2  
3 public class Atleta {  
4  
5     private String nome;  
6     private String posicao;  
7  
8     public Atleta(String nome, String posicao) {  
9         this.nome = nome;  
10        this.posicao = posicao;  
11    }  
12  
13    public Atleta(String nome) {  
14        this.nome = nome;  
15    }
```

CONTINUA >>

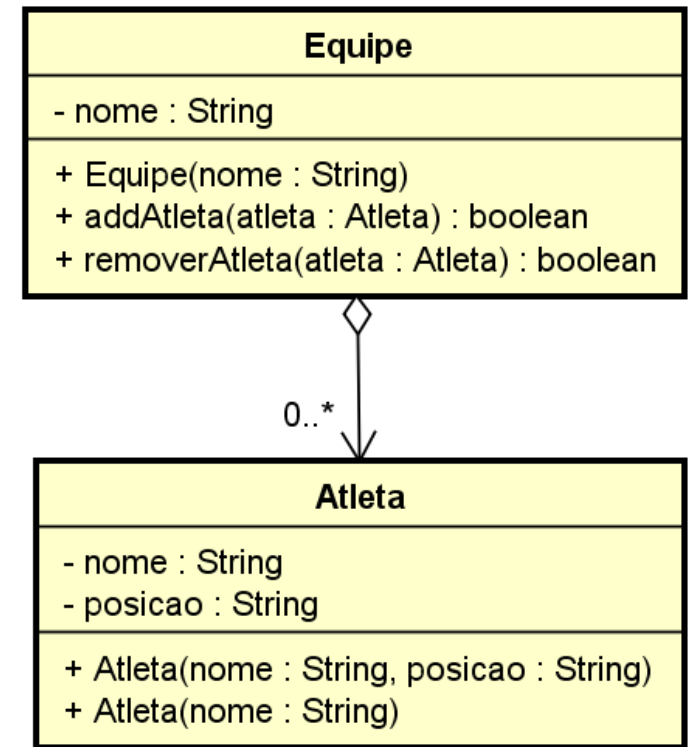


Agregação: Multiplicidade 0..*

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 >>

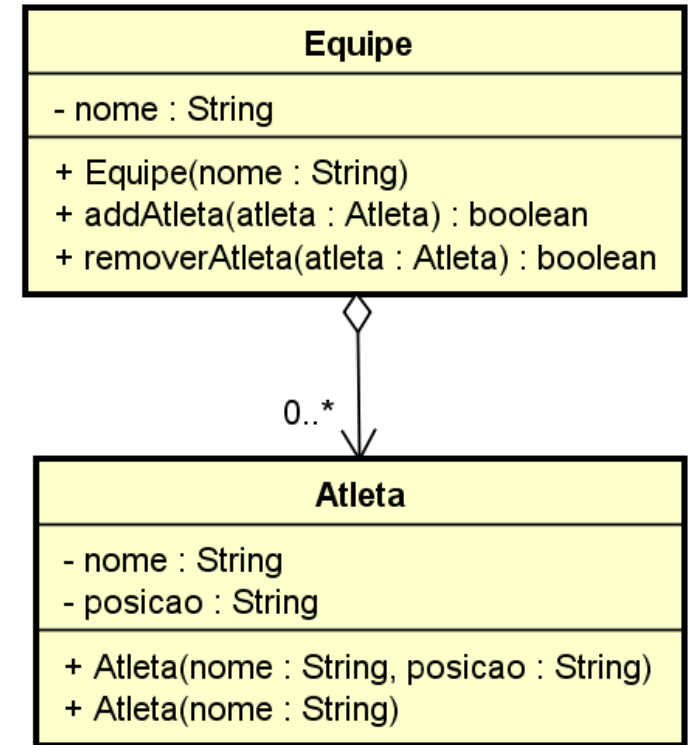


Agregação: Multiplicidade 0..*

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 }
```

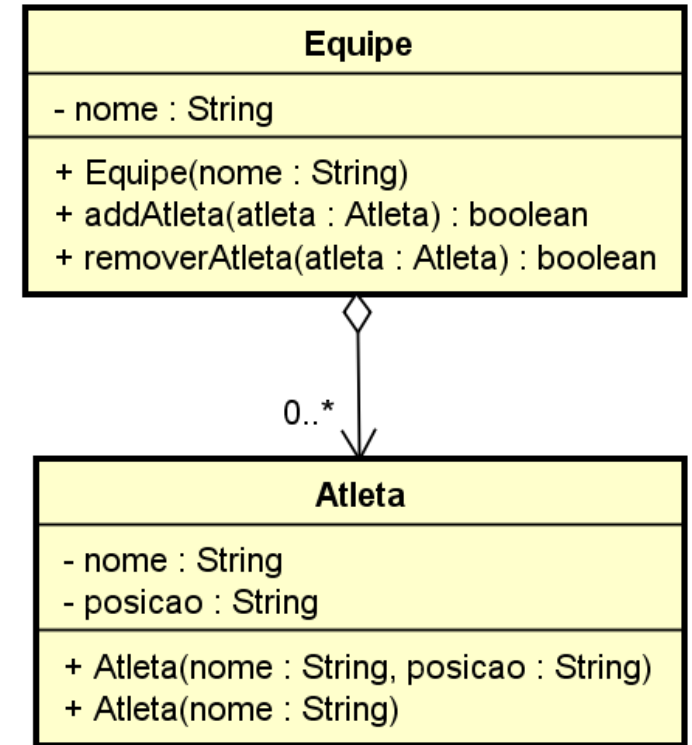


Agregação: Multiplicidade 0..*

Implementando a Classe Equipe

```
1 package br.com.renanrodrigues.agregacao;  
2  
3 import java.util.ArrayList;  
4 import java.util.List;  
5  
6 public class Equipe {  
7  
8     private String nome;  
9     private List<Atleta> listaAtleta = new ArrayList<Atleta>();  
10  
11     public Equipe(String nome) {  
12         this.nome = nome;  
13     }  
}
```

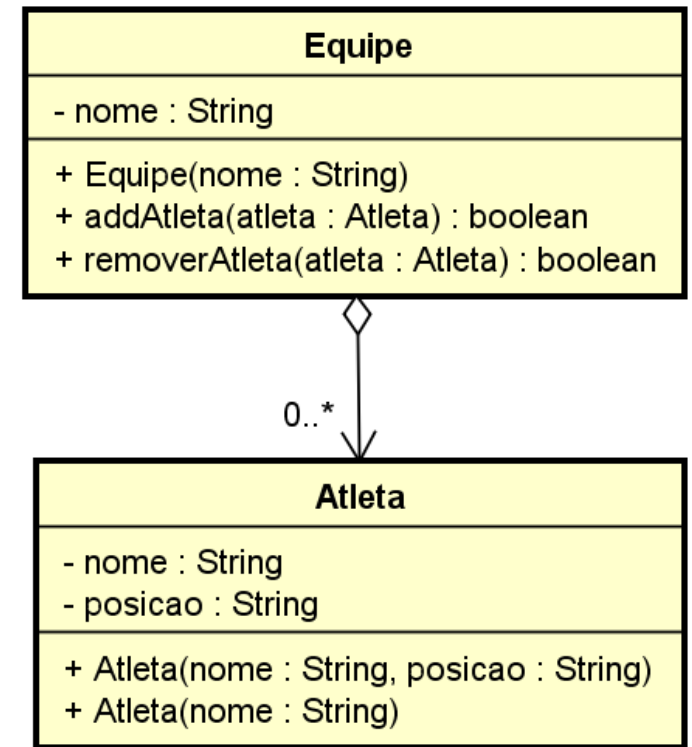
CONTINUA >>



Agregação: Multiplicidade 0..*

Implementando a Classe Equipe

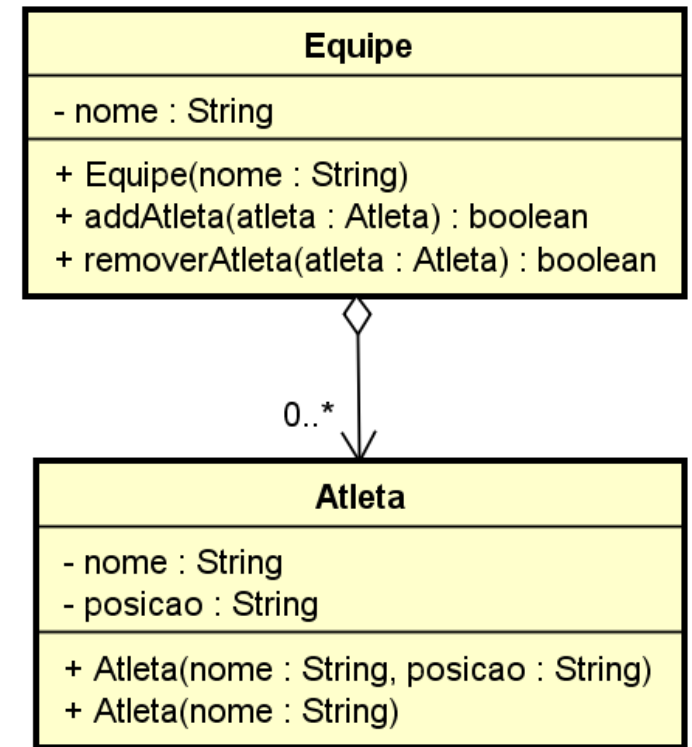
```
15 public boolean addAtleta(Atleta atleta) {  
16     boolean sucesso = false;  
17  
18     if (atleta != null && !listaAtleta.contains(atleta)) {  
19         listaAtleta.add(atleta);  
20         sucesso = true;  
21     }  
22     return sucesso;  
23 }  
  
25 public boolean removerAtleta(Atleta 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
```



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

```
1 package br.com.renanrodrigues.agregacao;
2
3 public class TesteEquipe {
4
5     public static void main(String[] args) {
6
7         Atleta a1 = new Atleta("Renan", "Goleiro");
8         Atleta a2 = new Atleta("Rafael Foster", "Zagueiro");
9         Atleta a3 = new Atleta("Renan", "Goleiro");
10        Atleta a4 = null;
11        Atleta a5 = new Atleta("Felipe Menezes", "Meia");
12
13        Equipe t = new Equipe("Goiás");
14
15        t.addAtleta(a1);
16        t.addAtleta(a2);
17        t.addAtleta(a3);
18        t.addAtleta(a4);
19        System.out.println(t);
20
21        t.addAtleta(a5);
22        System.out.println(t);
23
24        t.removerAtleta(a2);
25        System.out.println(t);
26    }
27 }
```

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..*

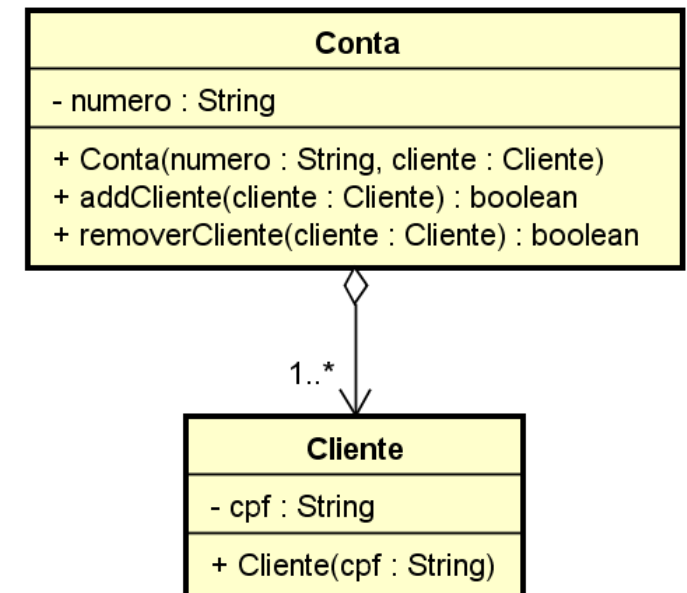
Exemplo: Uma Conta e seus Cliente

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

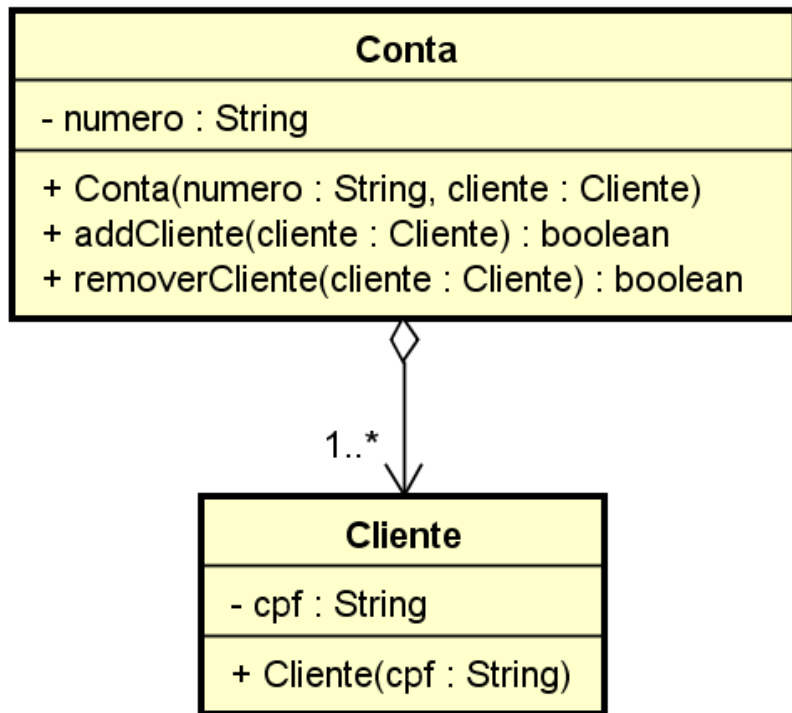


Na multiplicidade 1..*, o “todo” DEVE nascer possuindo uma parte.

- ▶ 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.

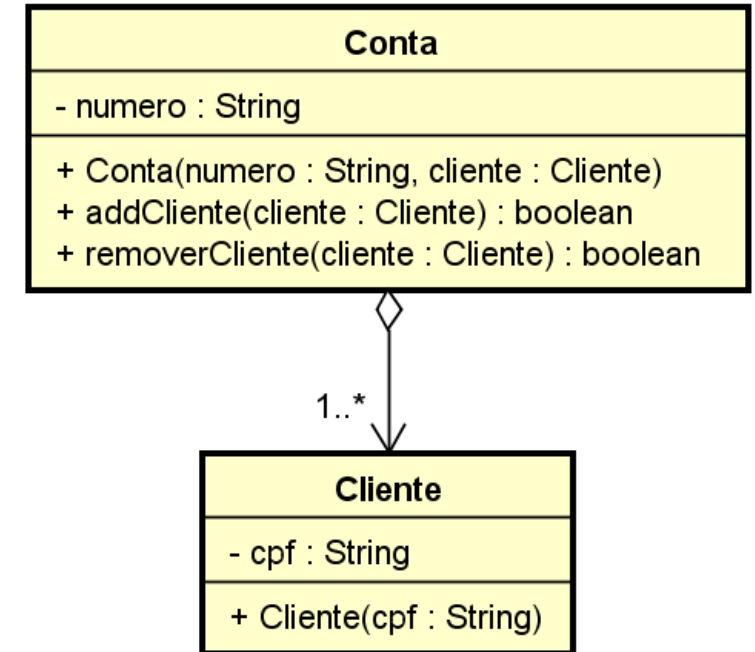
É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 1..*

Implementando a Classe Cliente

```
1 package br.com.renanrodrigues.agregacao;  
2  
3 public class Cliente {  
4  
5     private String cpf;  
6  
7     public Cliente(String cpf) {  
8         this.cpf = cpf;  
9     }  
10  
11     public String getCPF() {  
12         return cpf;  
13     }  
}
```

CONTINUA >>

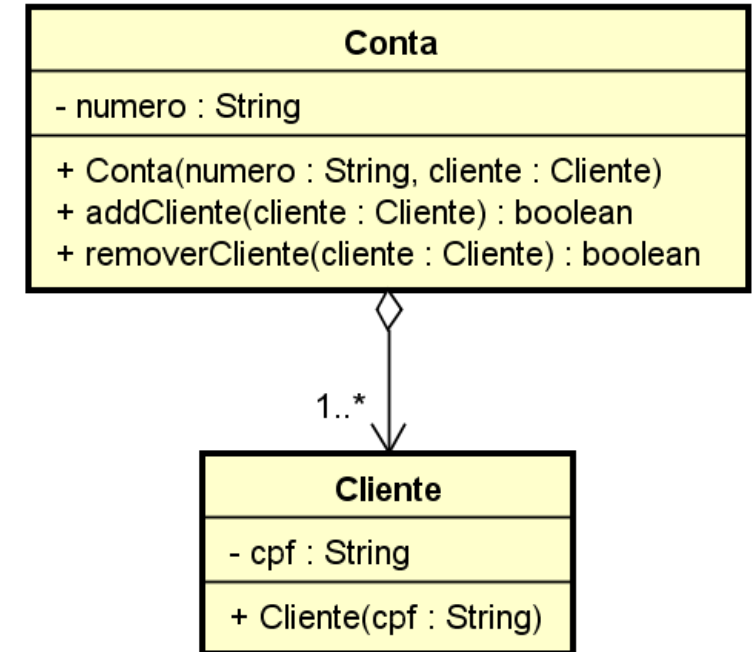


Agregação: Multiplicidade 1..*

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 }
```

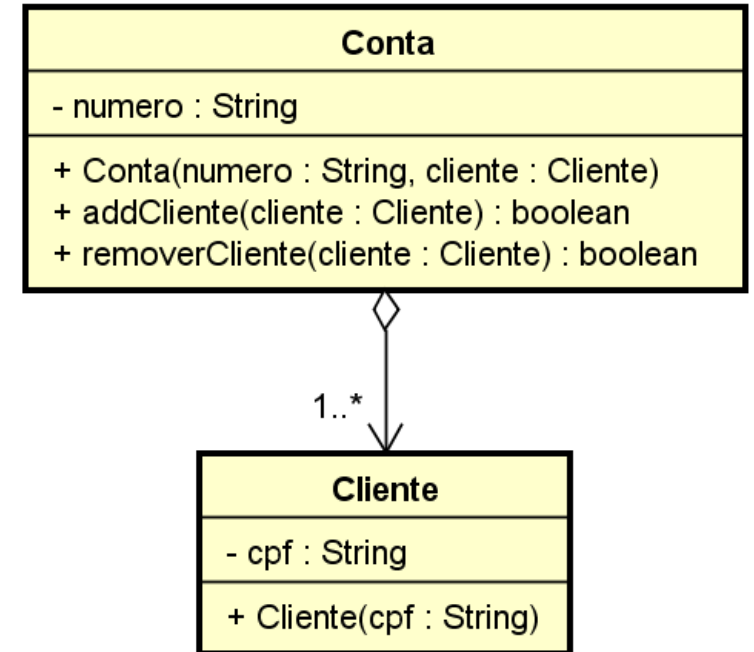


Agregação: Multiplicidade 1..*

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 }
```

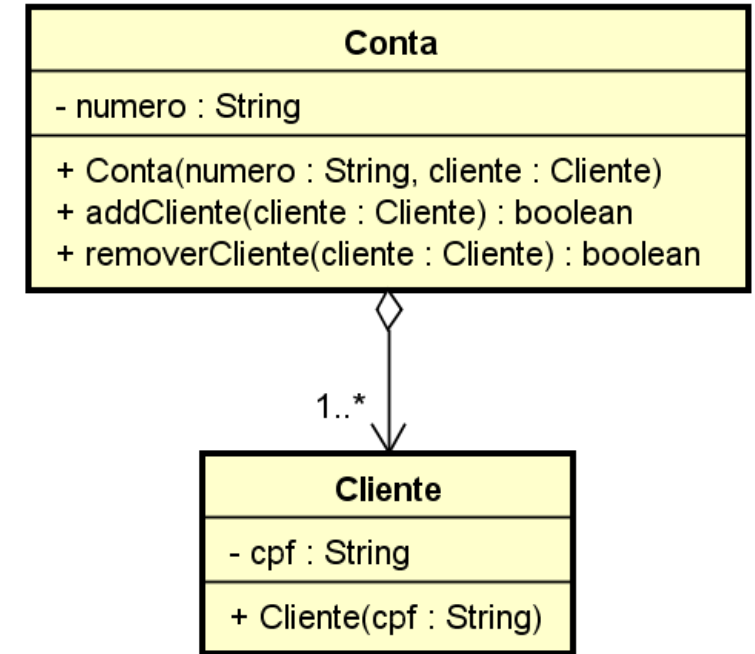


CONTINUA >>

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

```
1 package br.com.renanrodrigues.agregacao;
2
3 public class TesteConta {
4
5     public static void main(String[] args) {
6
7         Conta c;
8         Cliente t1 = new Cliente("111");
9         Cliente t2 = new Cliente("222");
10        Cliente t3 = null;
11        Cliente t4 = new Cliente("333");
12
13        try {
14            c = new Conta("ABC", t3);
15        } catch (NullPointerException e) {
16            System.err.println(e.getMessage());
17        }
18
19        c = new Conta("ABC", t1);
20        c.addCliente(t2);
21        c.addCliente(t4);
22        System.out.println(c);
23
24        c.removeCliente(t1);
25        c.removeCliente(t4);
26        System.out.println(c);
27
28        c.removeCliente(t2);
29        System.out.println(c);
30    }
31 }
```

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

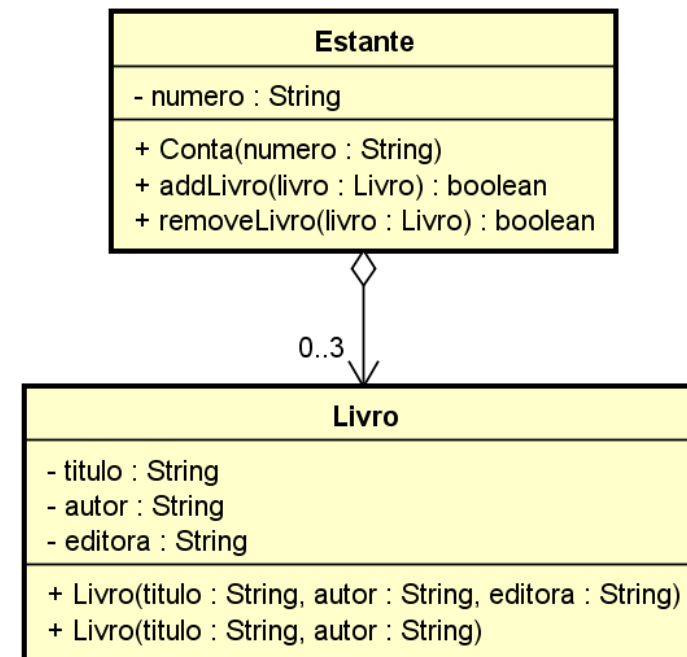
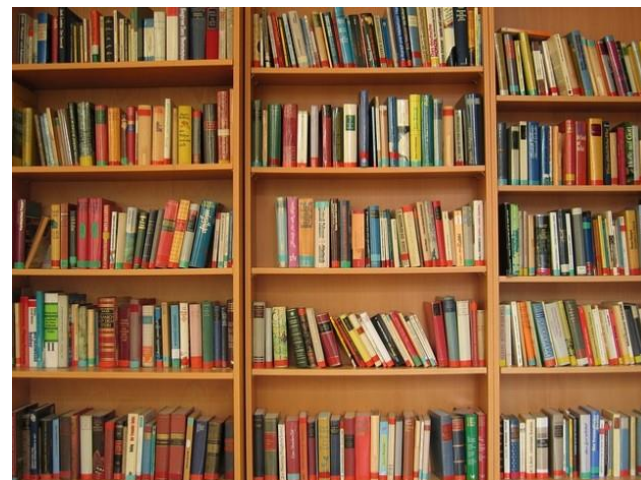
Exemplo: Uma Estante e seus Livros

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

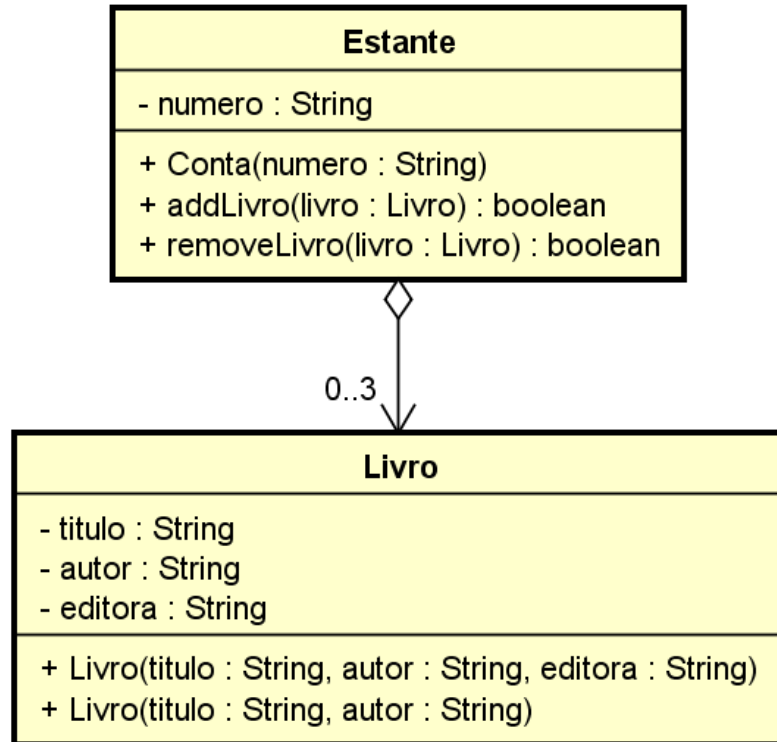


Na multiplicidade 0..N, o “todo” pode nascer sem possuir nenhuma parte.

- ▶ 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.

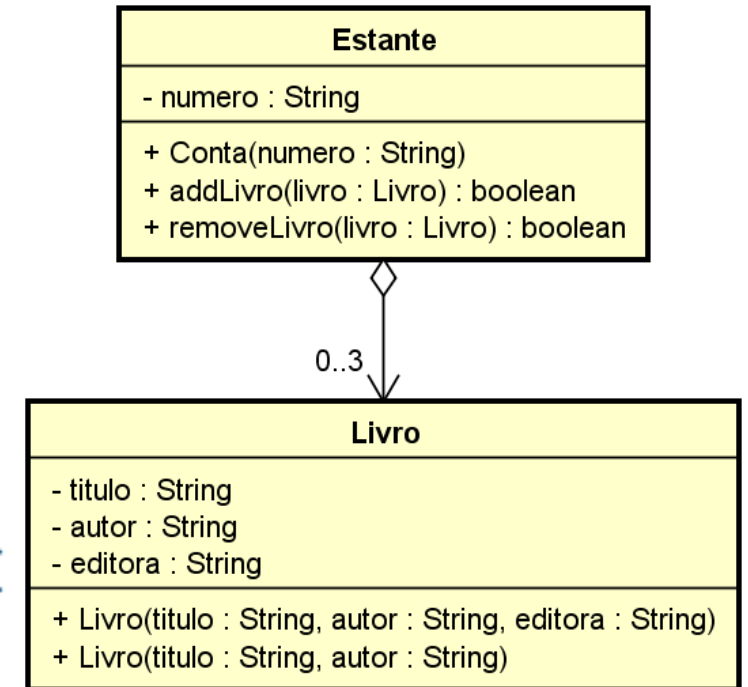
É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 0..N

Implementando a Classe Livro

```
1 package br.com.renanrodrigues.agregacao;
2
3 public class Livro {
4
5     private String titulo;
6     private String autor;
7     private String editora;
8
9     public Livro(String titulo, String autor, String editora) {
10         this.titulo = titulo;
11         this.autor = autor;
12         this.editora = editora;
13     }
14
15     public Livro(String titulo, String autor) {
16         this.titulo = titulo;
17         this.autor = autor;
18     }
19 }
```

CONTINUA >>

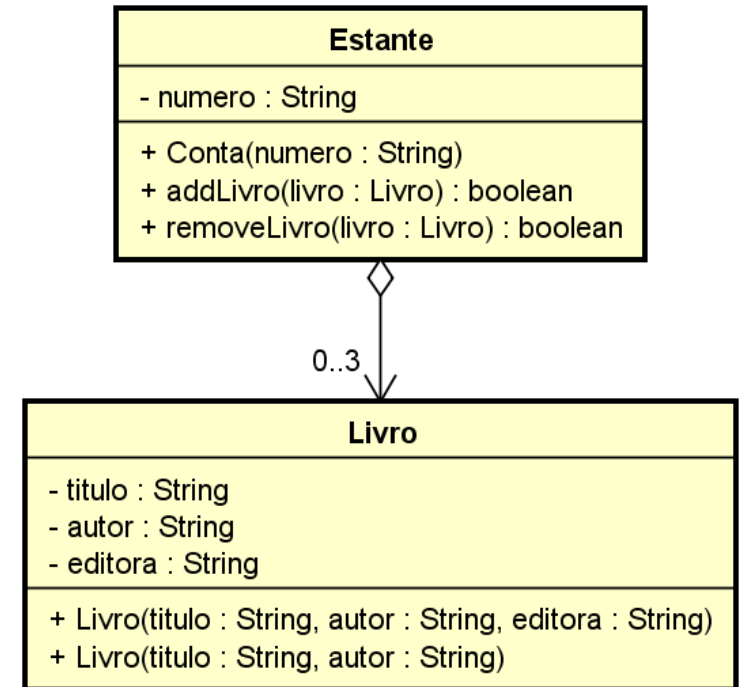


Agregação: Multiplicidade 0..N

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 >>

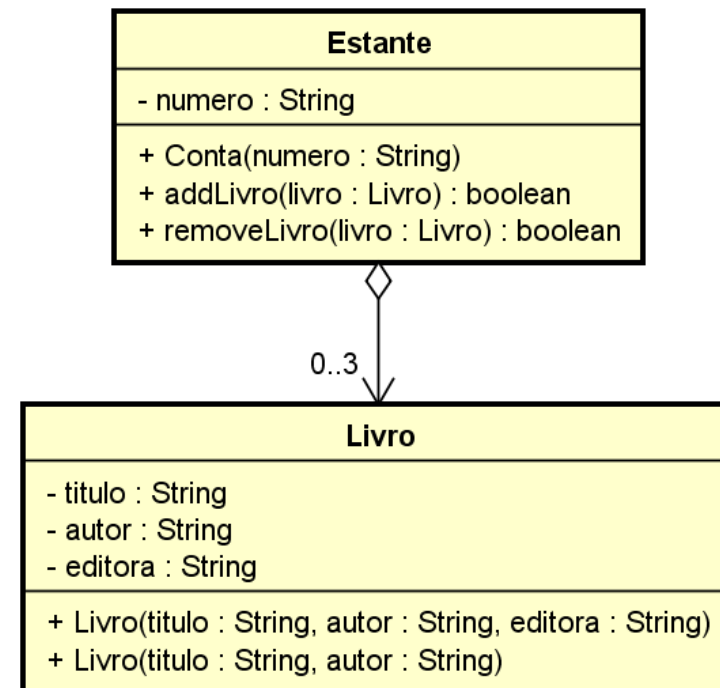


Agregação: Multiplicidade 0..N

Implementando a Classe Livro

```
32 @Override
33 public boolean equals(Object obj) {
34     if (this == obj)
35         return true;
36     if (obj == null)
37         return false;
38     if (getClass() != obj.getClass())
39         return false;
40     Livro other = (Livro) obj;
41     if (autor == null) {
42         if (other.autor != null)
43             return false;
44     } else if (!autor.equals(other.autor))
45         return false;
46     if (editora == null) {
47         if (other.editora != null)
48             return false;
49     } else if (!editora.equals(other.editora))
50         return false;
51     if (titulo == null) {
52         if (other.titulo != null)
53             return false;
54     } else if (!titulo.equals(other.titulo))
55         return false;
56     return true;
57 }
```

```
59 @Override
60 public String toString() {
61     return "Livro [titulo=" + titulo + ", autor=" + autor + ", editora="
62         + editora + "];"
63 }
64 }
```

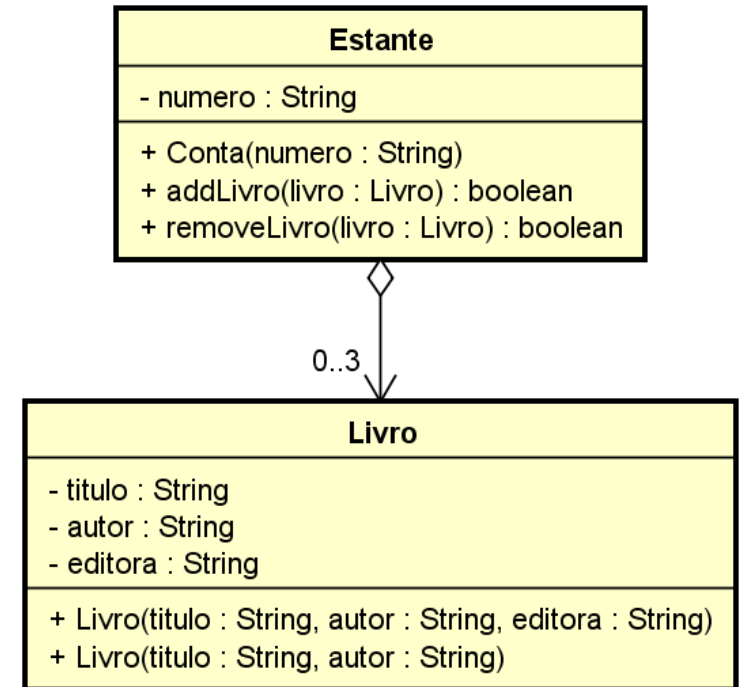


Agregação: Multiplicidade 0..N

Implementando a Classe Estante

```
1 package br.com.renanrodrigues.agregacao;  
2  
3 import java.util.ArrayList;  
4 import java.util.List;  
5  
6 public class Estante {  
7  
8     private String numero;  
9     private List<Livro> listaLivro = new ArrayList<Livro>();  
10  
11     public Estante(String numero) {  
12         this.numero = numero;  
13     }  
}
```

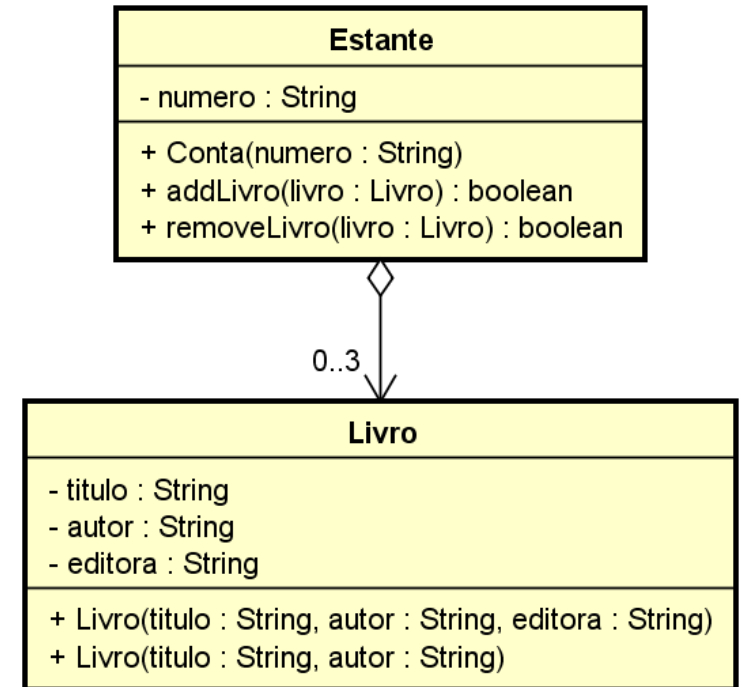
CONTINUA >>



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 }
24
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 }
```

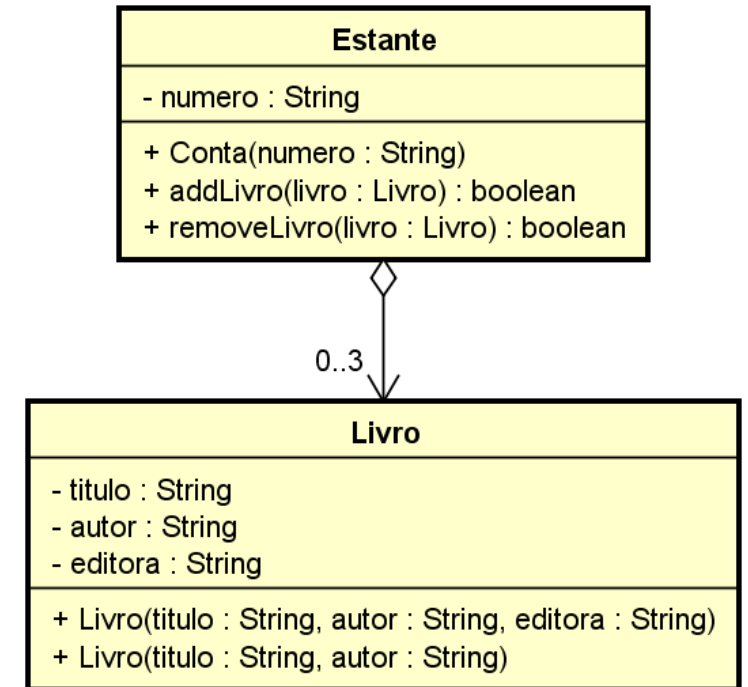


CONTINUA >>

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

```
1 package agregacao;
2
3 public class TesteEstante {
4
5     public static void main(String[] args) {
6
7         Livro lv1 = new Livro("Introdução à Programação Usando Java", "Rafael Santos", "Câmpus");
8         Livro lv2 = new Livro("Utilizando UML e Padrões", "Larman", "Bookman");
9         Livro lv3 = new Livro("Java: Como Programar", "Deitel", "Pearson");
10        Livro lv4 = new Livro("Programação Orientada a Objetos com Java", "Barnes", "Pearson");
11
12        Estante e = new Estante("AB-123");
13
14        e.addLivro(lv1);
15        e.addLivro(lv2);
16        e.addLivro(lv3);
17        e.addLivro(lv4);
18        System.out.println(e);
19
20        e.removeLivro(lv2);
21        e.addLivro(lv4);
22        System.out.println(e);
23
24    }
25 }
```

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]]]
```