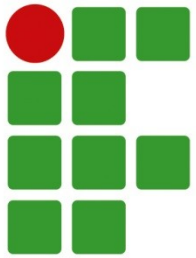


INSTITUTO FEDERAL
Goiás

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

Associação de Classes

Prof. Ms. Dory Gonzaga Rodrigues
Goiânia - GO

O que é uma Associação ?

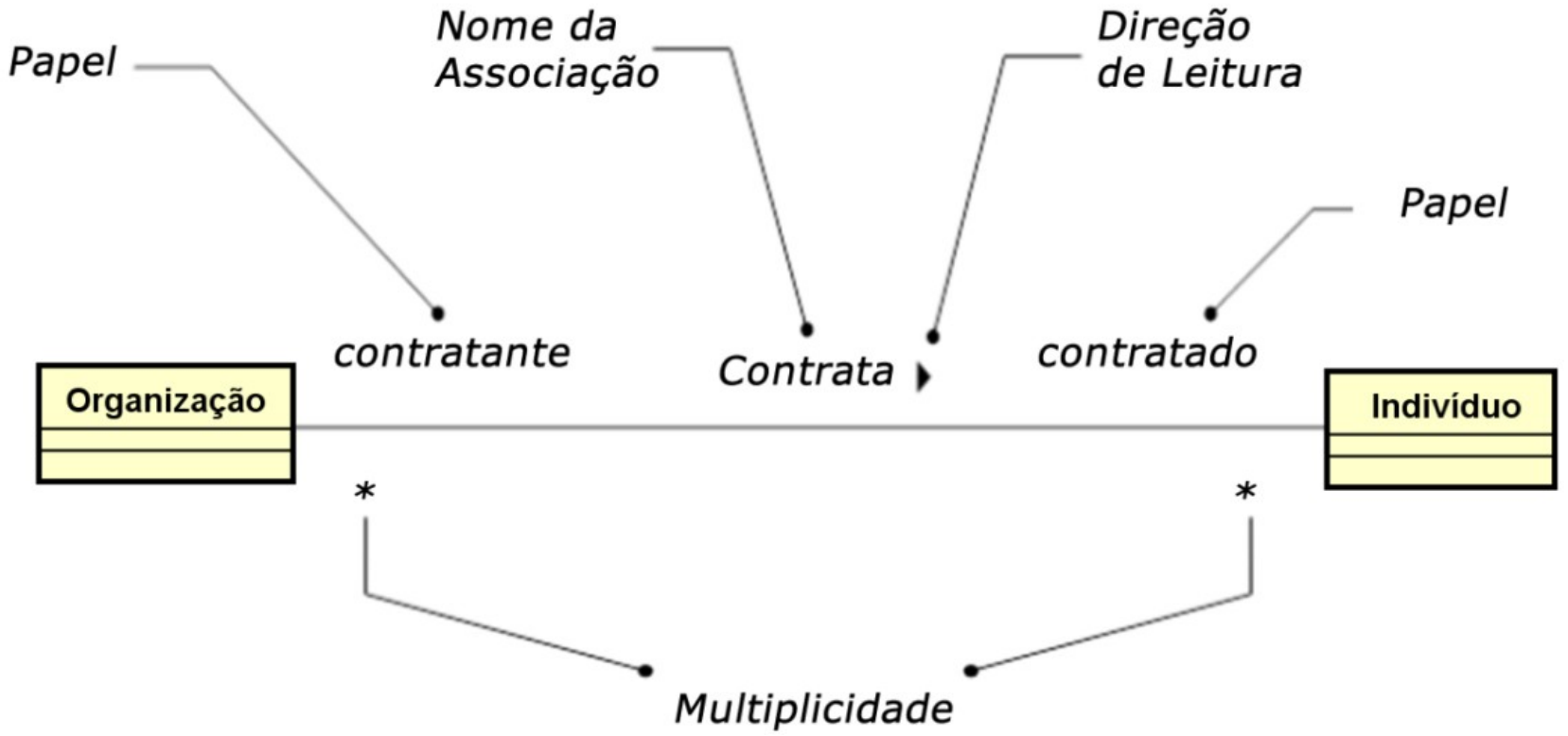
Uma associação entre duas classes é a forma de representar o relacionamento entre elas.

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

Como essas peças poderiam se relacionar ?

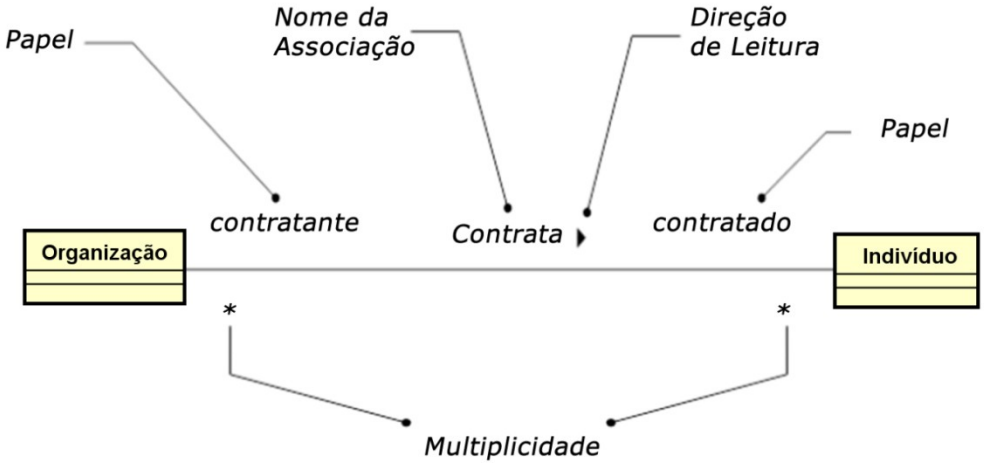
Associação

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



Associação

Somente as classes que estão relacionadas são as classes cujos objetos podem se comunicar.



Uma associação pode ter os seguintes elementos:

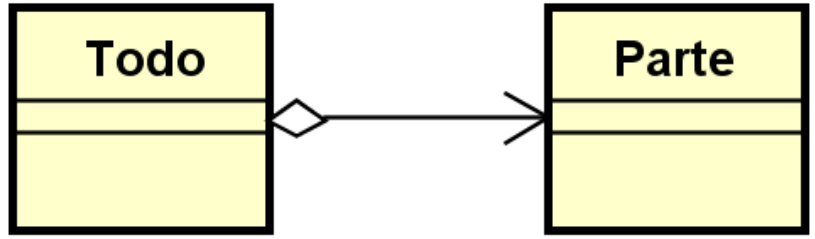
- **Nome da Associação:** descreve a natureza da associação;
- **Papel:** define o papel específico neste relacionamento;
- **Direção de Leitura:** indica como a associação deve ser lida;
- **Multiplicidade:** define quantos objetos estarão conectados a uma instância de uma associação.

Agregação

Agregação é tipo de associação 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".



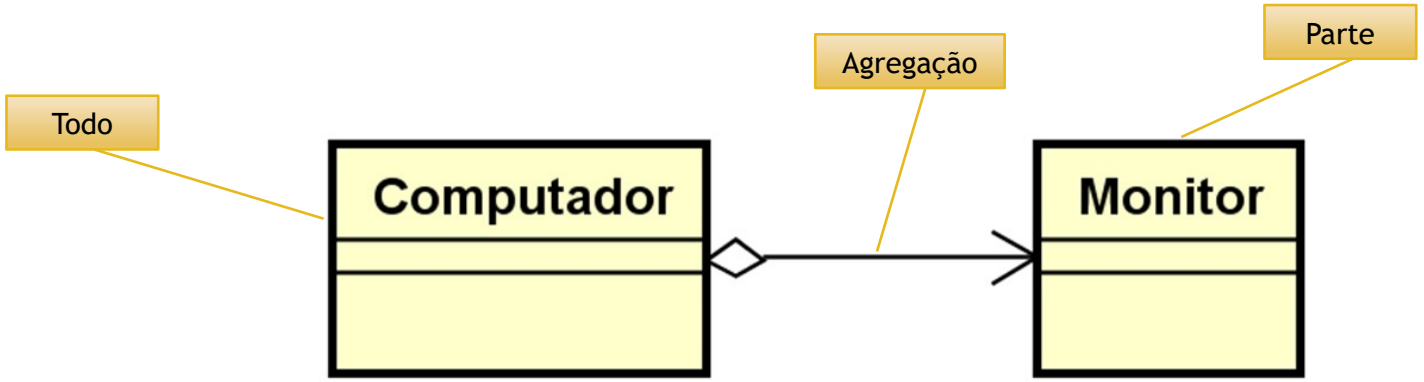
Agregação e a Multiplicidade

A Multiplicidade representa a quantidade de associações, ou seja, define a quantidade de "partes" que pode estar associada ao todo.

- Multiplicidade 0..1
- Multiplicidade 1..1
- Multiplicidade 0..*
- Multiplicidade 1..*
- Multiplicidade 1..N (onde o N >1 e definido. Ex: 1..3)

Exemplo: Agregação

Exemplo: O Computado e seu Monitor



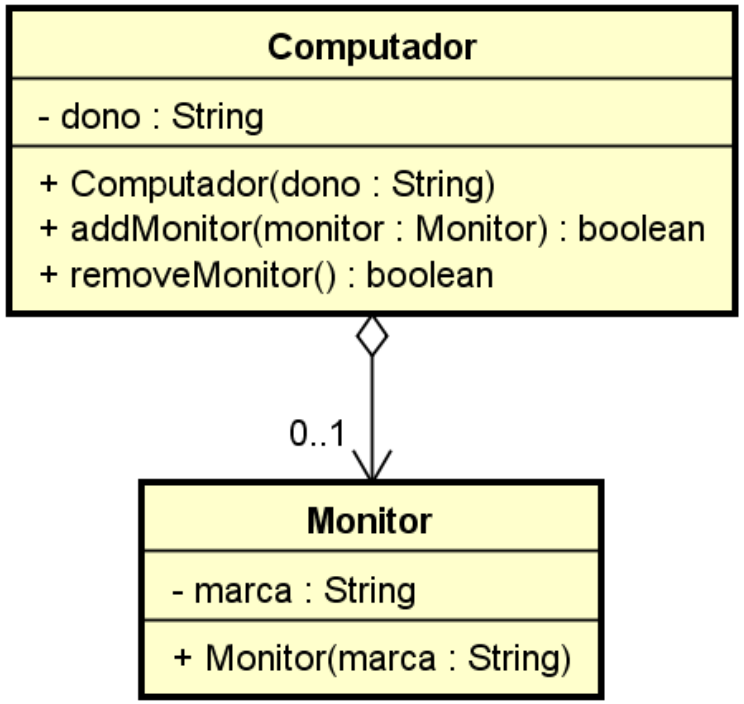
Para que a parte possa existir sem o todo, ela deve estar criada antes de estar agregada ao todo.

- 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

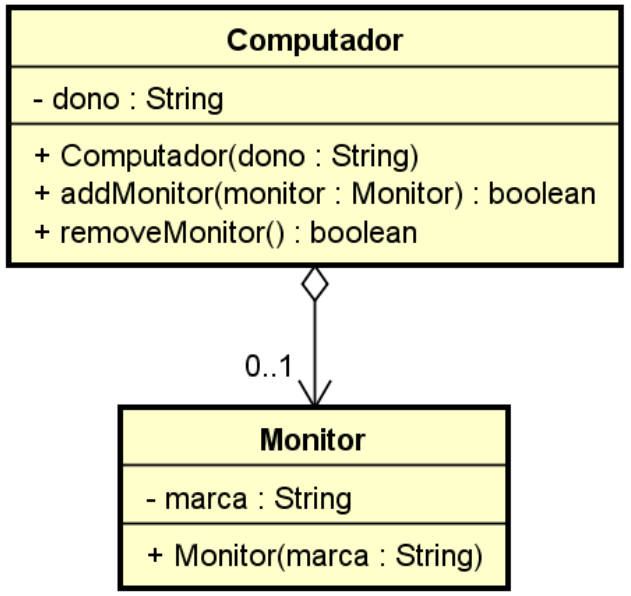
O Monitor pode fazer parte de um Computador. Se o Computador for jogado fora, o Monitor ainda continua existindo.



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

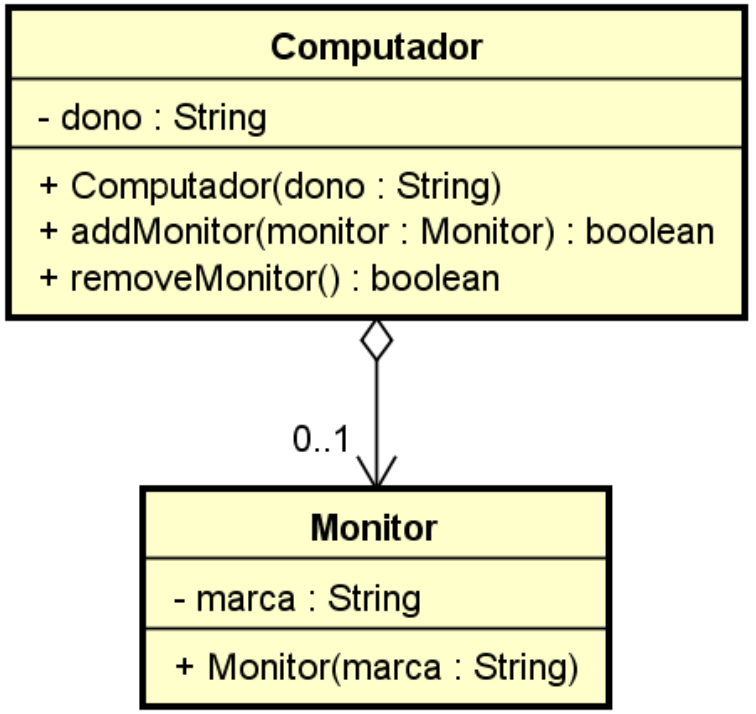
É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.

Agregação: Multiplicidade 0..1

Implementando a Classe Monitor

Monitor.java

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



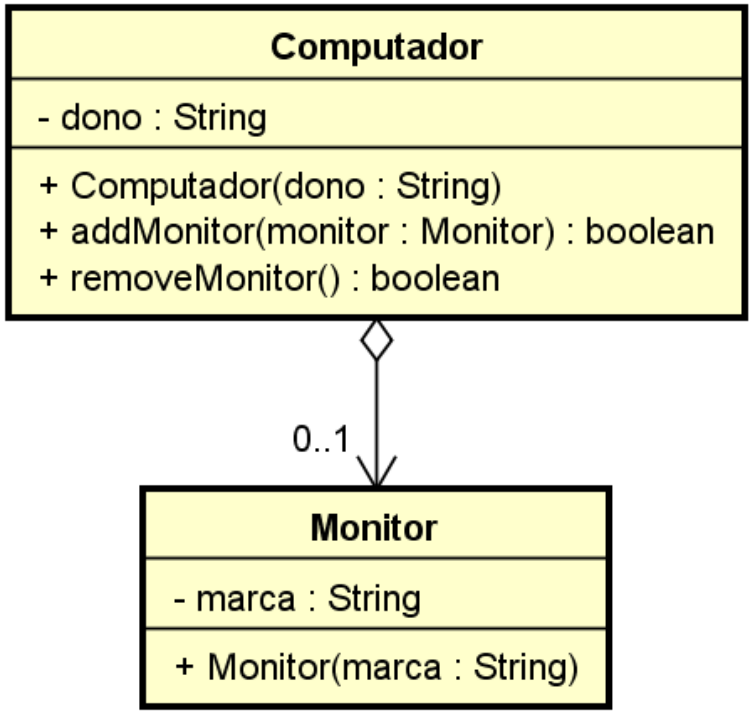
Agregação: Multiplicidade 0..1

Implementando a Classe Computador

Computador.java

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

CONTINUA >>

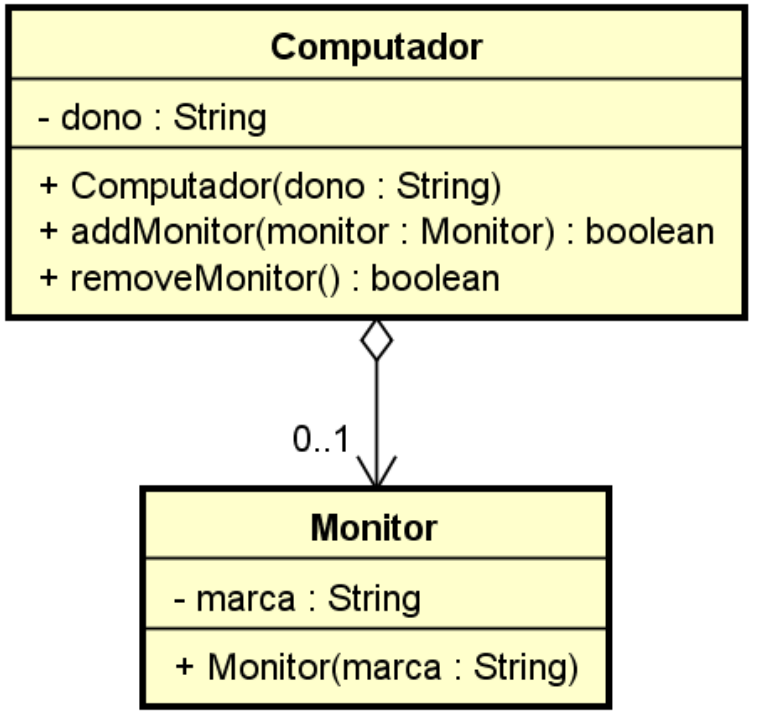


Agregação: Multiplicidade 0..1

Implementando a Classe Computador

Computador.java

```
20
21      public boolean removeMonitor() {
22
23          boolean sucesso = false;
24
25          if (this.monitor != null) {
26              this.monitor = null;
27              sucesso = true;
28          }
29
30          return sucesso;
31      }
```



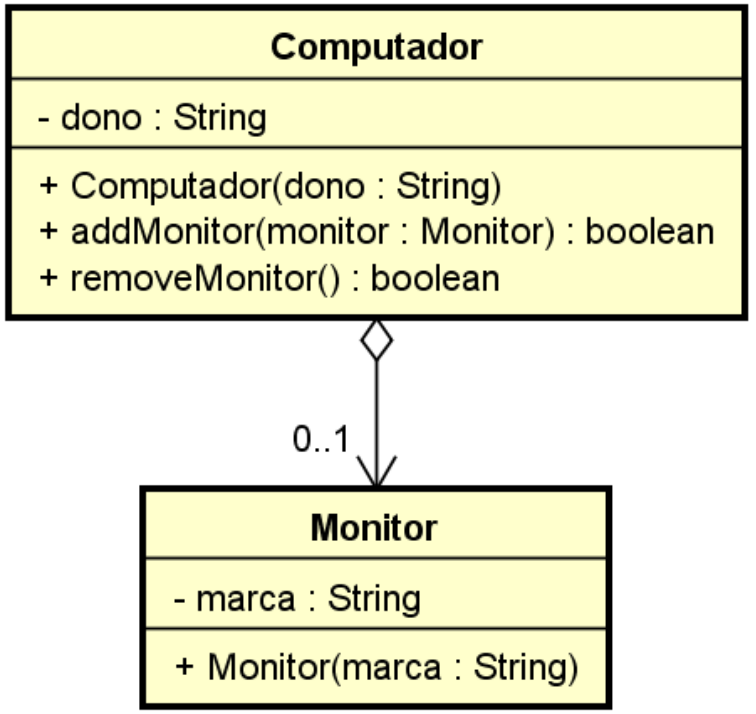
CONTINUA >>

Agregação: Multiplicidade 0..1

Implementando a Classe Computador

Computador.java

```
33      public Monitor getMonitor() {
34          return monitor;
35      }
36
37      public String getDono() {
38          return dono;
39      }
40
41      public void setDono(String dono) {
42          this.dono = dono;
43      }
44
45      @Override
46      public String toString() {
47          return "Computador [dono=" + dono + ", monitor=" + monitor + "]";
48      }
49  }
```



Agregação: Multiplicidade 0..1

Programa Principal

TesteComputador.java

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

Saída do Programa

Console

```
<terminated> TesteComputador [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (27 de set c
Computador [dono=Dory, monitor=null]
Computador [dono=Dory, monitor=Monitor [marca=Samsung]]
Computador [dono=Dory, monitor=Monitor [marca=Samsung]]
Computador [dono=Dory, monitor=null]
Computador [dono=Dory, monitor=Monitor [marca=Philips]]
```

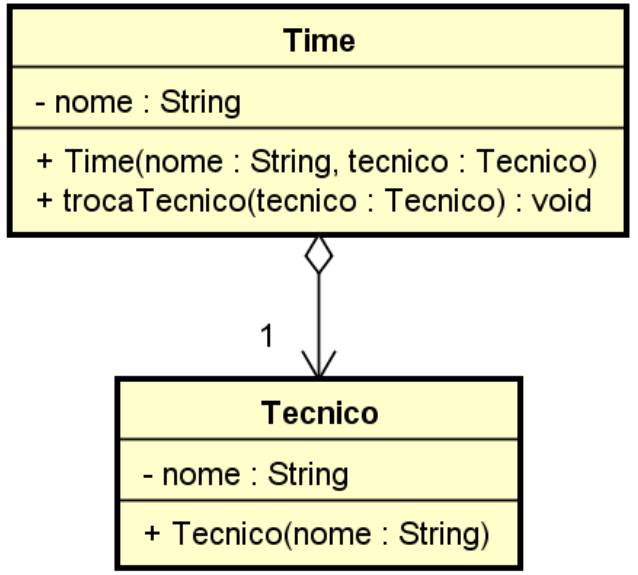
Agregação: Multiplicidade 1..1

Exemplo: Um Time e seu Técnico

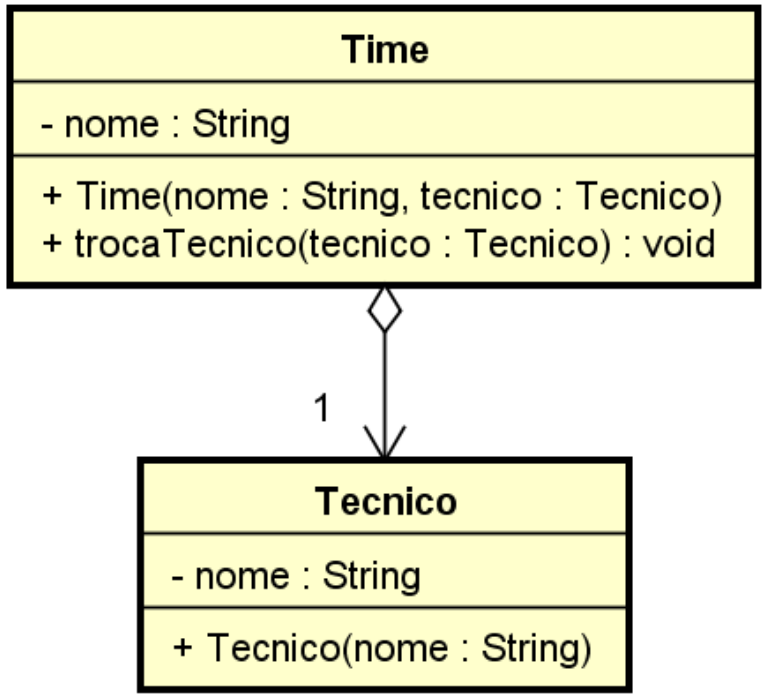
Um Time tem um Técnico. Se o Técnico perder o emprego ele ainda continua sendo um Técnico.

Na multiplicidade 1..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..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.

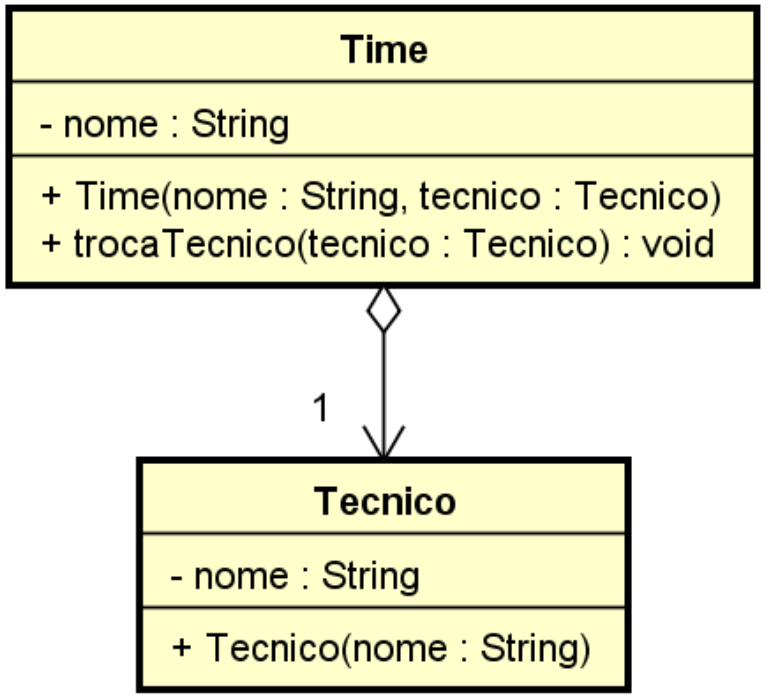
É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.

Agregação: Multiplicidade 1..1

Implementando a Classe Técnico

Tecnico.java

```java
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 + "]";
    }
}
```

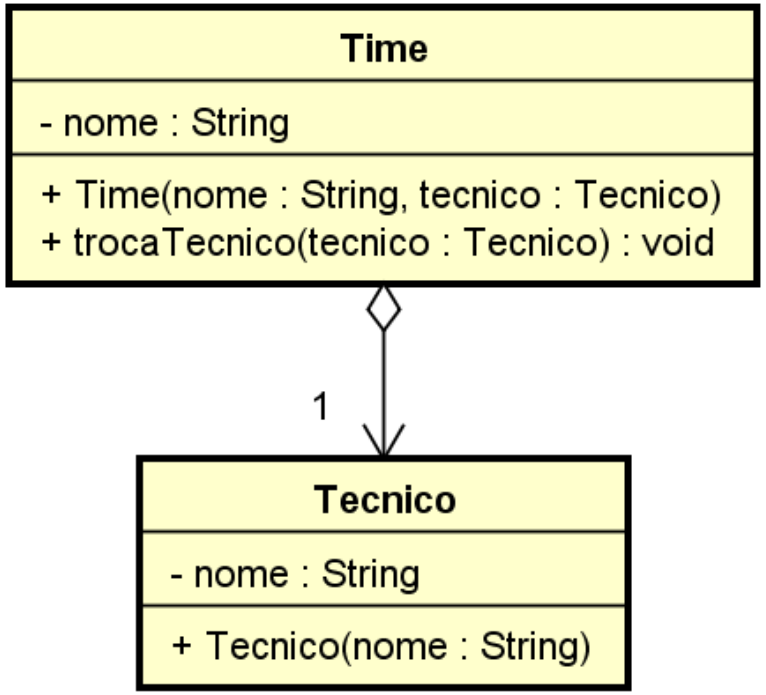


Agregação: Multiplicidade 1..1

Implementando a Classe Time

Time.java

```
1  public class Time {
2
3      private String nome;
4      private Tecnico tecnico;
5
6      public Time(String nome, Tecnico tecnico) {
7
8          if (tecnico == null) {
9              throw new NullPointerException("A referência do Técnico não pode ser nula!");
10         }
11
12         this.nome = nome;
13         this.tecnico = tecnico;
14     }
15
16     public boolean trocaTecnico(Tecnico tecnico) {
17
18         boolean sucesso = false;
19
20         if (tecnico != null) {
21             this.tecnico = tecnico;
22             sucesso = true;
23         }
24
25         return sucesso;
26     }
```



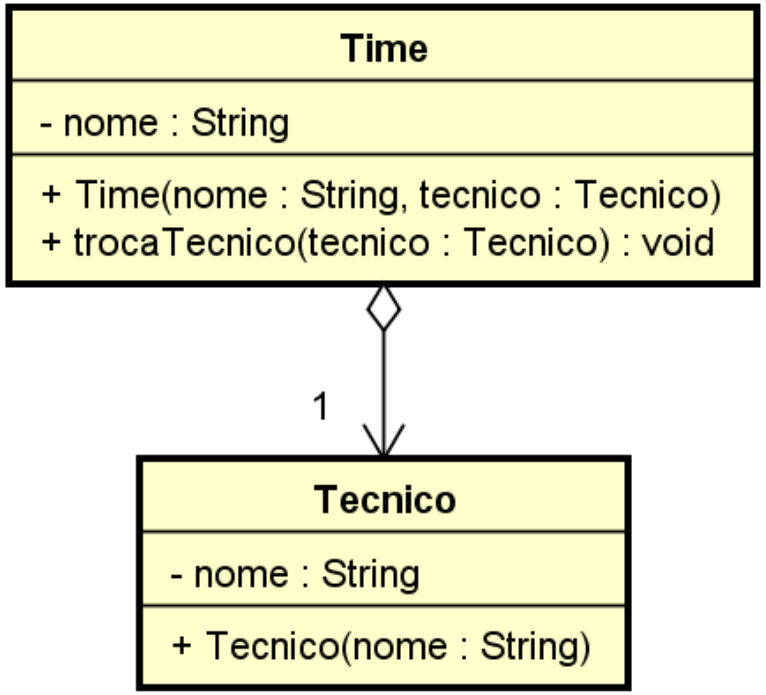
CONTINUA >>

Agregação: Multiplicidade 1..1

Implementando a Classe Time

Time.java

```
27
28      public String getNome() {
29          return nome;
30      }
31
32      public Tecnico getTecnico() {
33          return tecnico;
34      }
35
36      @Override
37      public String toString() {
38          return "Time [nome=" + nome + ", tecnico=" + tecnico + "]";
39      }
40  }
```



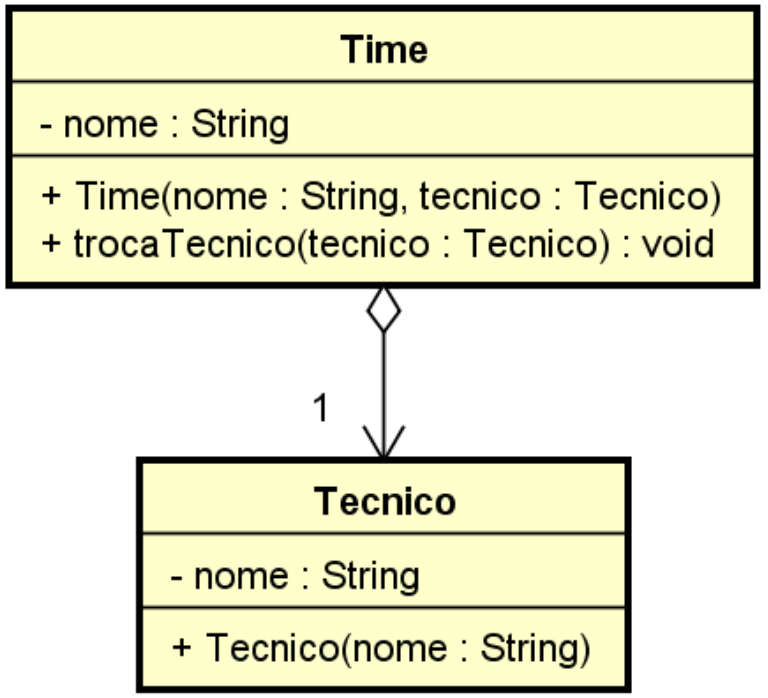
CONTINUA >>

Agregação: Multiplicidade 1..1

Programa Principal

TesteTime.java

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



Saída do Programa

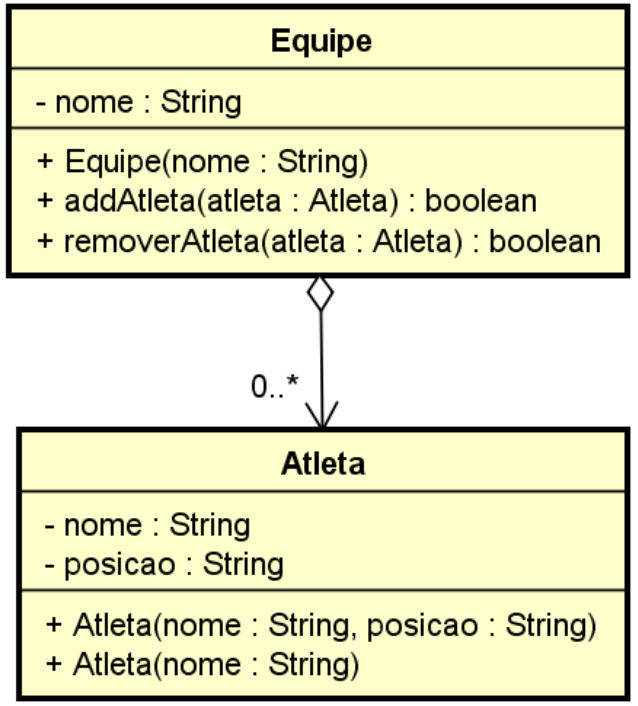
Console

```
<terminated> TesteTime [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.
A referência do Técnico não pode ser nula!
Time [nome=Brasil, tecnico=Tecnico [nome=Tite]]
Time [nome=Brasil, tecnico=Tecnico [nome=Caio]]
```

Agregação: Multiplicidade 0..*

Exemplo: Uma Equipe e seus Atletas

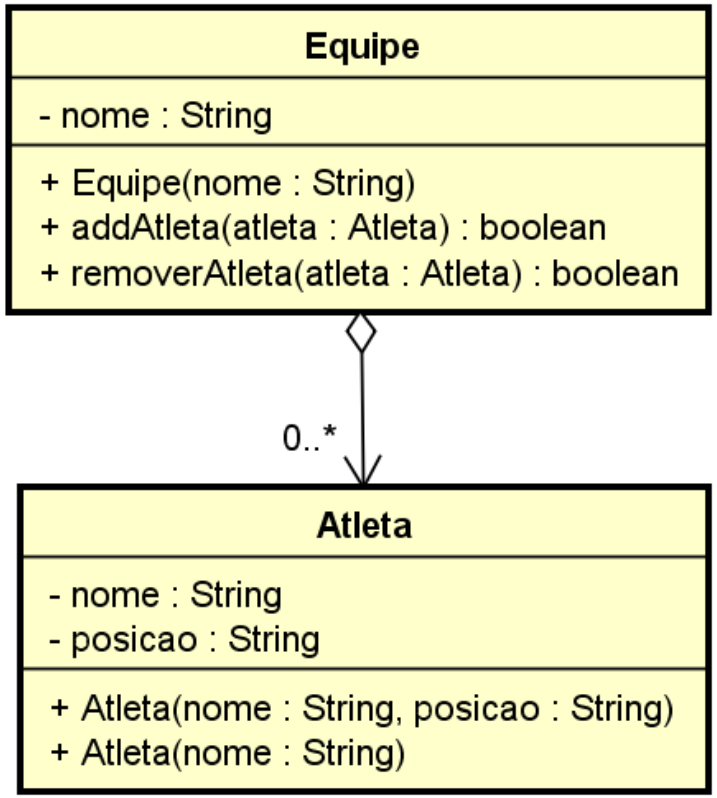
Uma Equipe possui vários Atletas. Porém, deve existir antes de ser relacionado a Equipe.
Se um Atleta for desligado da Equipe, o Atleta ainda pode jogar em outra Equipe.



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.

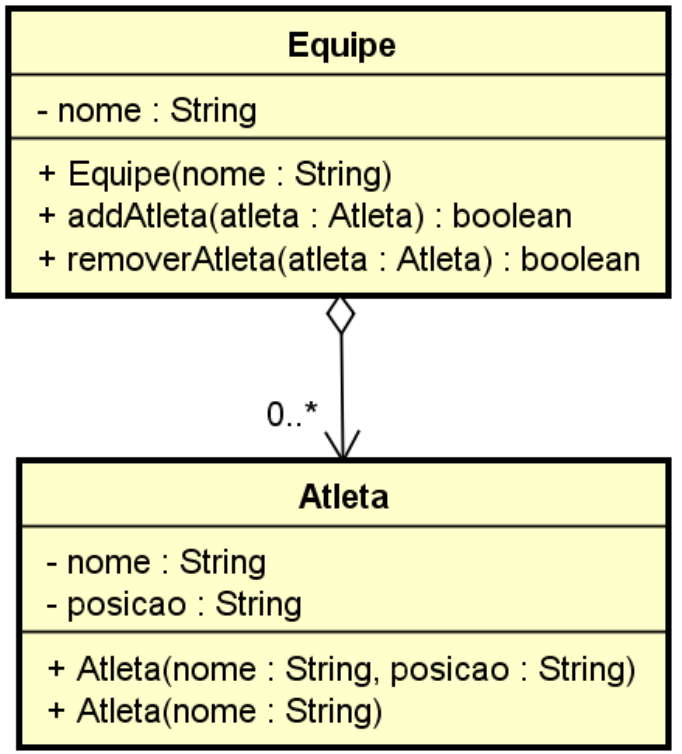
Agregação: Multiplicidade 0..*

Implementando a Classe Atleta

Atleta.java

```
1  public class Atleta {
2
3      private String nome;
4      private String posicao;
5
6      public Atleta(String nome, String posicao) {
7          this.nome = nome;
8          this.posicao = posicao;
9      }
10
11     public Atleta(String nome) {
12         this.nome = nome;
13     }
14
15     public String getNome() {
16         return nome;
17     }
18
19     public String getPosicao() {
20         return posicao;
21     }
22
23     public void setPosicao(String posicao) {
24         this.posicao = posicao;
25     }
```

CONTINUA >>

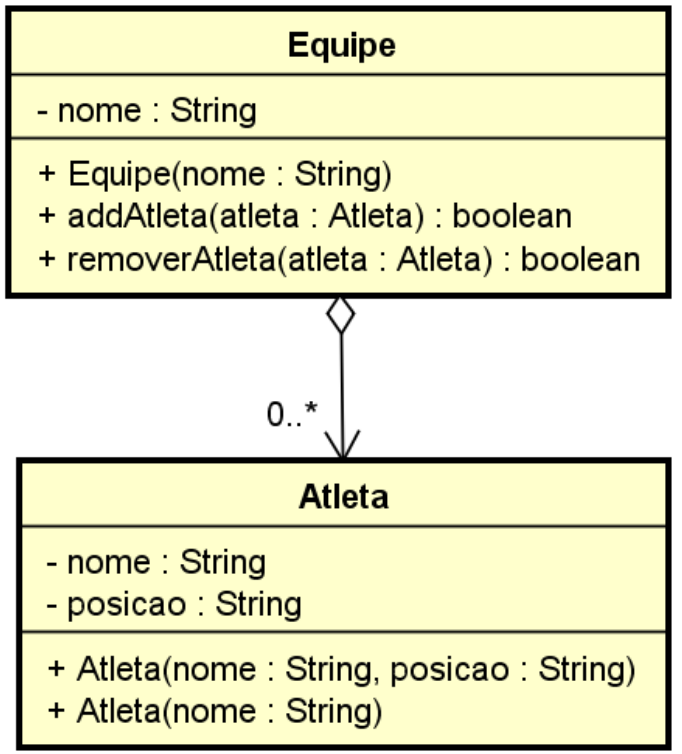


Agregação: Multiplicidade 0..*

Implementando a Classe Atleta

Atleta.java

```
26
27      @Override
28      public boolean equals(Object obj) {
29          if (this == obj)
30              return true;
31          if (obj == null)
32              return false;
33          if (getClass() != obj.getClass())
34              return false;
35          Atleta other = (Atleta) obj;
36          if (nome == null) {
37              if (other.nome != null)
38                  return false;
39          } else if (!nome.equals(other.nome))
40              return false;
41          if (posicao == null) {
42              if (other.posicao != null)
43                  return false;
44          } else if (!posicao.equals(other.posicao))
45              return false;
46          return true;
47      }
48
49      @Override
50      public String toString() {
51          return "Atleta [nome=" + nome + ", posicao=" + posicao + "]";
52      }
53  }
```

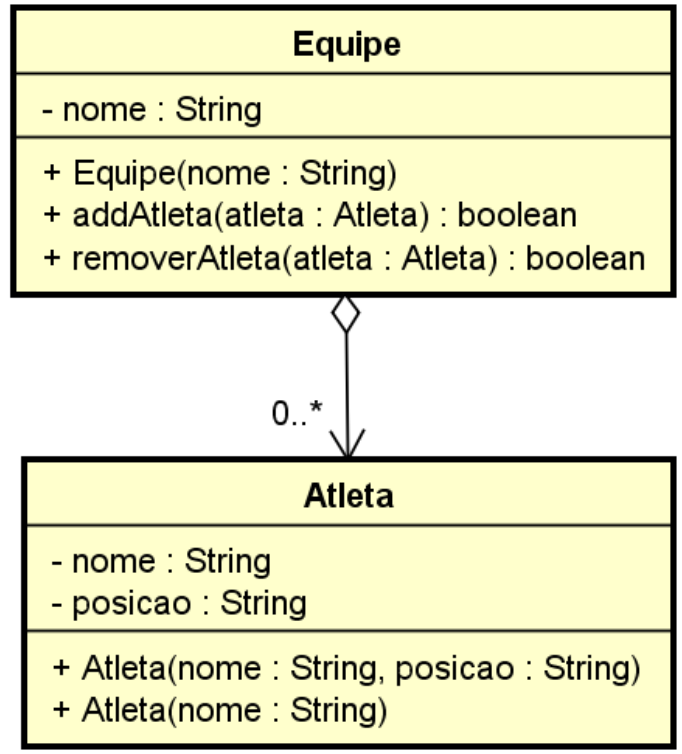


Agregação: Multiplicidade 0..*

Implementando a Classe Equipe

Equipe.java

```java
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;
    }

    public boolean addAtleta(Atleta atleta) {
        boolean sucesso = false;

        if (atleta!= null && !listaAtleta.contains(atleta)) {
            listaAtleta.add(atleta);
            sucesso = true;
        }
        return sucesso;
    }

```



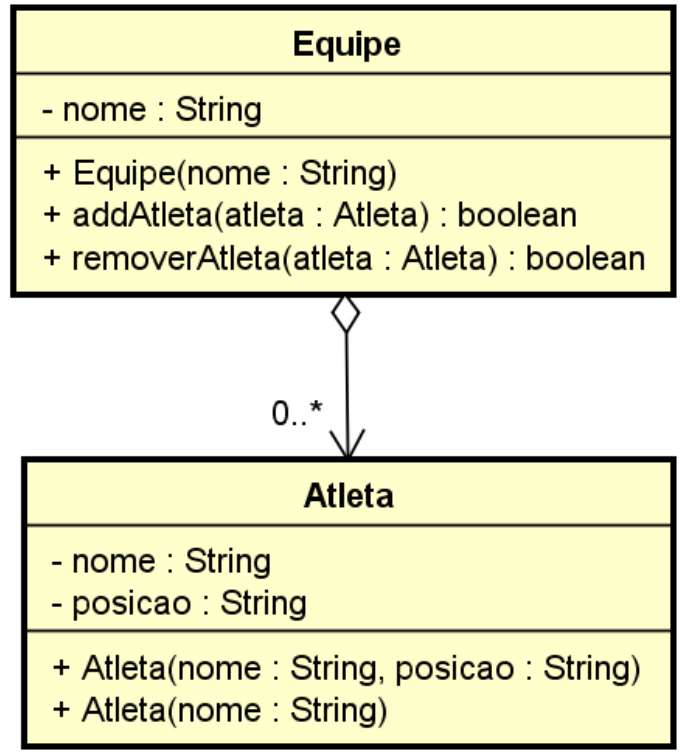
CONTINUA >>

Agregação: Multiplicidade 0..*

Implementando a Classe Equipe

Equipe.java

```
21
22      public boolean removerAtleta(Atleta a) {
23          boolean sucesso = false;
24
25          if (listaAtleta.size() > 0 && listaAtleta.contains(a)) {
26              listaAtleta.remove(a);
27              sucesso = true;
28          }
29          return sucesso;
30      }
31
32      public String getNome() {
33          return nome;
34      }
35
36      public void setNome(String nome) {
37          this.nome = nome;
38      }
39
40      public List<Atleta> getListaAtleta() {
41          return listaAtleta;
42      }
43
44      @Override
45      public String toString() {
46          return "Time [nome=" + nome + "\nlistaAtleta=" + listaAtleta + "\n]";
47      }
48  }
```



Agregação: Multiplicidade 0..*

Programa Principal

TesteEquipe.java

```
1  public class TesteEquipe {
2
3      public static void main(String[] args) {
4
5          Atleta a1 = new Atleta("Dory", "Atacante");
6          Atleta a2 = new Atleta("Rafael", "Zagueiro");
7          Atleta a3 = new Atleta("Dory", "Atacante");
8          Atleta a4 = null;
9          Atleta a5 = new Atleta("Felipe", "Meia");
10
11         Equipe t = new Equipe("Goiás");
12
13         t.addAtleta(a1);
14         t.addAtleta(a2);
15         t.addAtleta(a3);
16         t.addAtleta(a4);
17         System.out.println(t);
18
19         t.addAtleta(a5);
20         System.out.println(t);
21     }
22 }
```

Saída do Programa

Console

```
<terminated> TesteEquipe [Java Application] C:\Program Files\Java\jre1.8.0_171
Time [nome=Goiás
listaAtleta=[
Atleta [nome=Dory, posicao=Atacante],
Atleta [nome=Rafael, posicao=Zagueiro]]
]
Time [nome=Goiás
listaAtleta=[
Atleta [nome=Dory, posicao=Atacante],
Atleta [nome=Rafael, posicao=Zagueiro],
Atleta [nome=Felipe, posicao=Meia]]
]
```

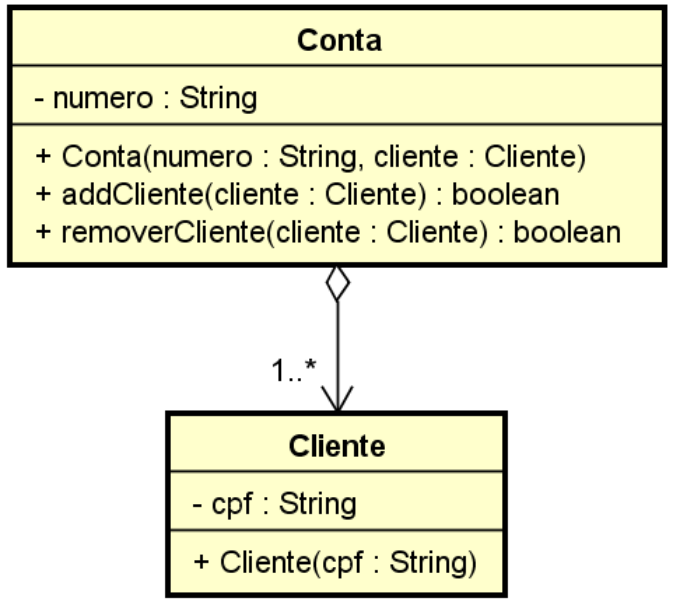
Agregação: Multiplicidade 1..*

Exemplo: Uma Conta e seus Cliente

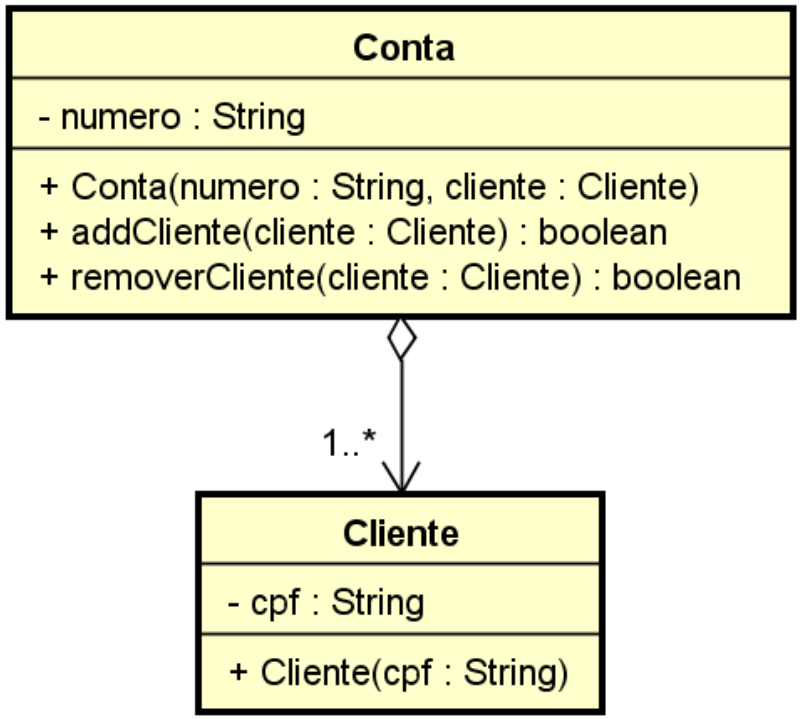
Uma Conta possui um ou mais Clientes. Se um Cliente for retirado da Conta, o Cliente ainda existe e pode abrir em outra Conta.

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.

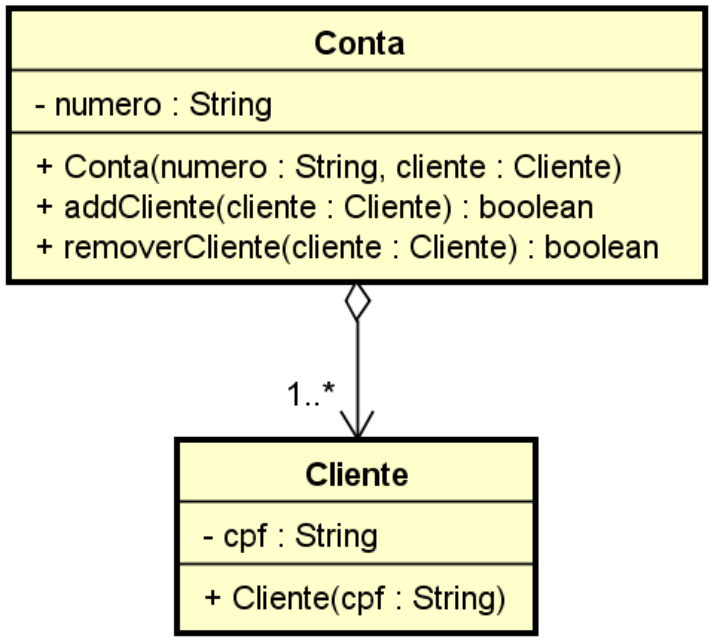
É de responsabilidade do desenvolvedor a manutenção da multiplicidade na lista de partes.

Agregação: Multiplicidade 1..*

Implementando a Classe Cliente

Cliente.java

```java
public class Cliente {

    private String cpf;

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

    public String getCPF() {
        return cpf;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Cliente other = (Cliente) obj;
        if (cpf == null) {
            if (other.cpf != null)
                return false;
        } else if (!cpf.equals(other.cpf))
            return false;
        return true;
    }
```



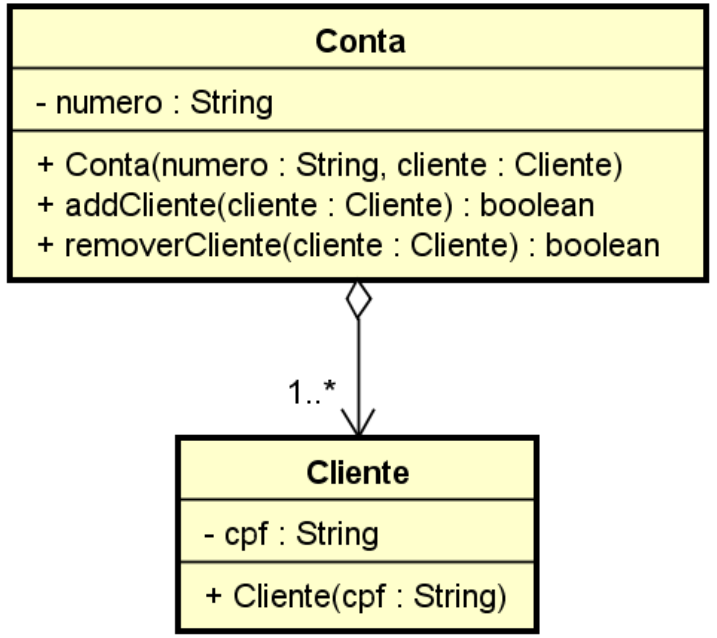
CONTINUA >>

Agregação: Multiplicidade 1..*

Implementando a Classe Cliente

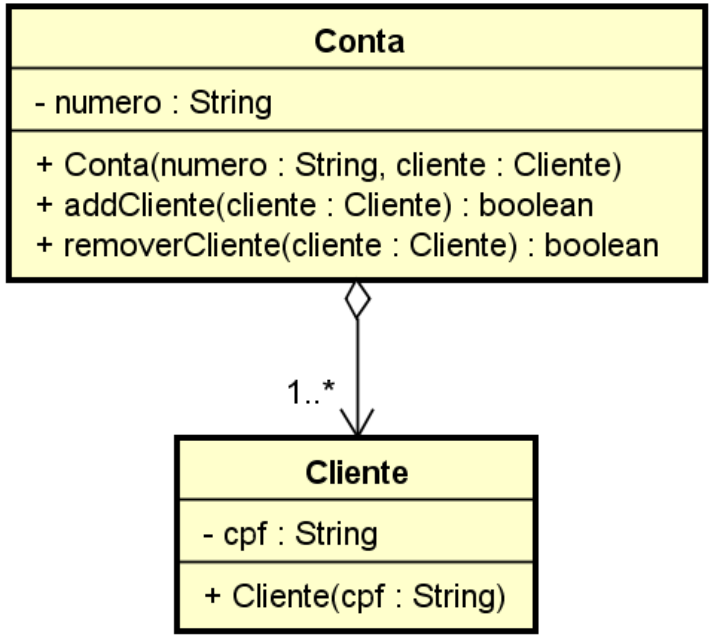
Cliente.java

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



Agregação: Multiplicidade 1..*

Implementando a Classe Conta



Conta.java

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class Conta {
5
6      private String numero;
7      private List<Cliente> listaCliente = new ArrayList<Cliente>();
8
9      public Conta(String numero, Cliente cliente) {
10
11         if (cliente == null) {
12             throw new NullPointerException("A referência do Cliente não pode ser nula!");
13         }
14         this.numero = numero;
15         this.addCliente(cliente);
16     }
17
18     public boolean addCliente(Cliente titular) {
19         boolean sucesso = false;
20
21         if (titular!= null) {
22             listaCliente.add(titular);
23             sucesso = true;
24         }
25         return sucesso;
26     }
```

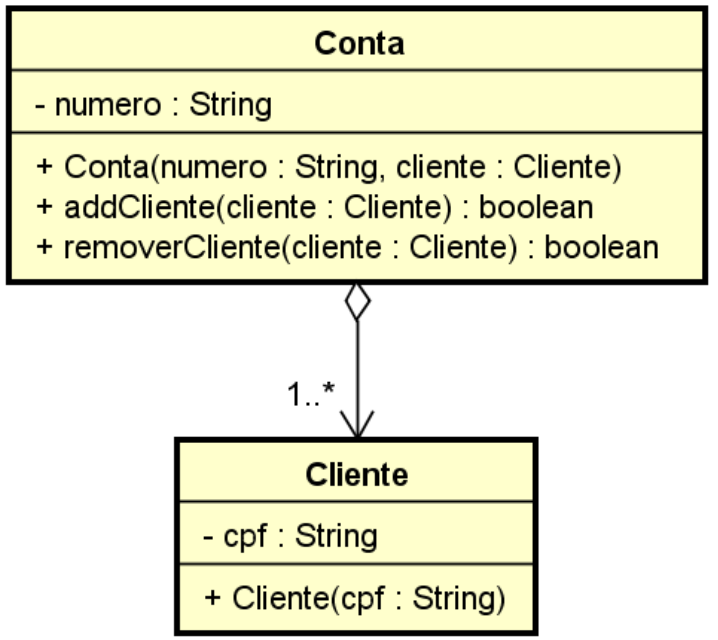
CONTINUA >>

Agregação: Multiplicidade 1..*

Implementando a Classe Conta

Conta.java

```
28      public boolean removerCliente(Cliente a) {
29          boolean sucesso = false;
30
31          if (listaCliente.size() > 1 && listaCliente.contains(a)) {
32              listaCliente.remove(a);
33              sucesso = true;
34          }
35          return sucesso;
36      }
37
38      public String getNumero() {
39          return numero;
40      }
41
42      public List<Cliente> getListaCliente() {
43          List<Cliente> listaRetorno = new ArrayList<Cliente>();
44          listaRetorno.addAll(listaCliente);
45          return listaRetorno;
46      }
47
48      @Override
49      public String toString() {
50          return "Conta [numero=" + numero + ", listaCliente=" + listaCliente + "]";
51      }
52  }
```



Agregação: Multiplicidade 1..*

Programa Principal

TesteConta.java

```
1  public class TesteConta {
2
3      public static void main(String[] args) {
4
5          Conta c;
6          Cliente t1 = new Cliente("123.456.789-00");
7          Cliente t2 = new Cliente("987.654.321-00");
8          Cliente t3 = null;
9
10         try {
11             c = new Conta("Ouro", t3);
12         } catch (NullPointerException e) {
13             System.err.println(e.getMessage());
14         }
15
16         c = new Conta("Platinum", t1);
17         c.addCliente(t2);
18         System.out.println(c);
19
20         c.removerCliente(t1);
21         System.out.println(c);
22     }
23 }
```

Saída do Programa

Console

```
<terminated> TesteConta [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (27 de set de 2018 04:02:35)
A referência do Cliente não pode ser nula!
Conta [numero=Platinum, listaCliente=[Cliente [cpf=123.456.789-00], Cliente [cpf=987.654.321-00]]]
Conta [numero=Platinum, listaCliente=[Cliente [cpf=987.654.321-00]]]
```

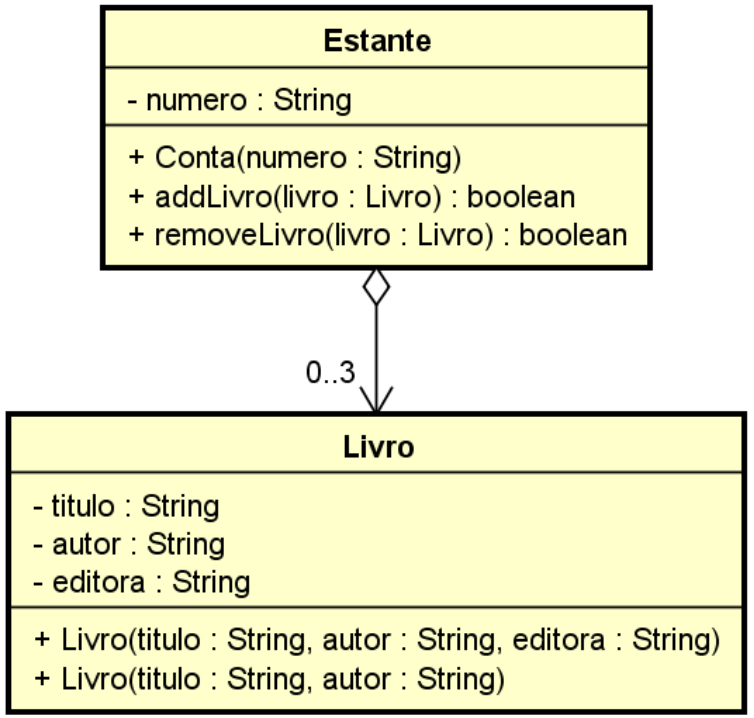
Agregação: Multiplicidade 0..3

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.

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.

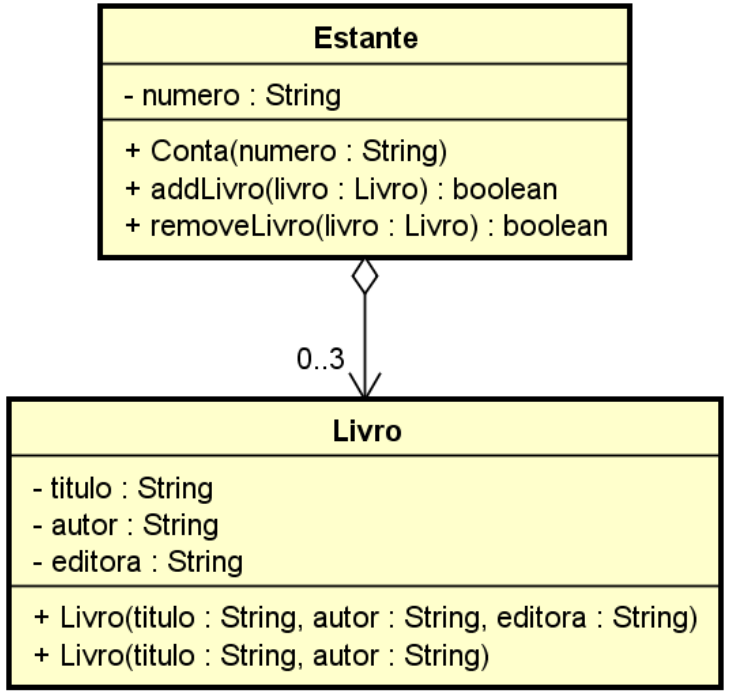
Agregação: Multiplicidade 0..3

Implementando a Classe Livro

Livro.java

```
1  public class Livro {
2
3      private String titulo;
4      private String autor;
5      private String editora;
6
7      public Livro(String titulo, String autor, String editora) {
8          this.titulo = titulo;
9          this.autor = autor;
10         this.editora = editora;
11     }
12
13     public Livro(String titulo, String autor) {
14         this.titulo = titulo;
15         this.autor = autor;
16     }
17
18     public String getTitulo() {
19         return titulo;
20     }
21
22     public String getAutor() {
23         return autor;
24     }
25
26     public String getEditora() {
27         return editora;
28     }
```

CONTINUA >>

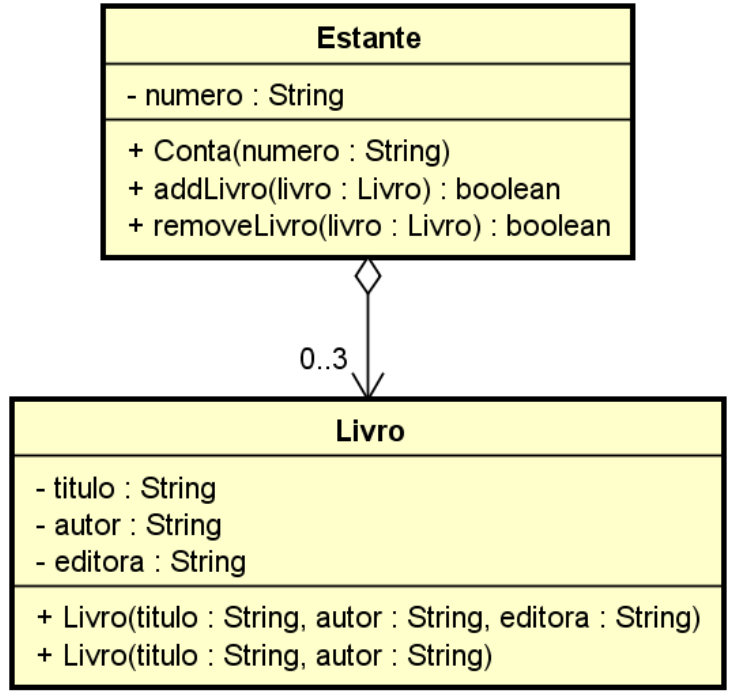


Agregação: Multiplicidade 0..3

Implementando a Classe Livro

Livro.java

```
30      @Override
31      public boolean equals(Object obj) {
32          if (this == obj)
33              return true;
34          if (obj == null)
35              return false;
36          if (getClass() != obj.getClass())
37              return false;
38          Livro other = (Livro) obj;
39          if (autor == null && other.autor != null)
40              return false;
41          if (!autor.equals(other.autor))
42              return false;
43          if (editora == null && other.editora != null)
44              return false;
45          if (!editora.equals(other.editora))
46              return false;
47          if (titulo == null && other.titulo != null)
48              return false;
49          if (!titulo.equals(other.titulo))
50              return false;
51          return true;
52      }
53
54      @Override
55      public String toString() {
56          return "\n\tLivro [titulo=" + titulo + ", autor=" + autor + ", editora=" + editora + "]";
57      }
58  }
```

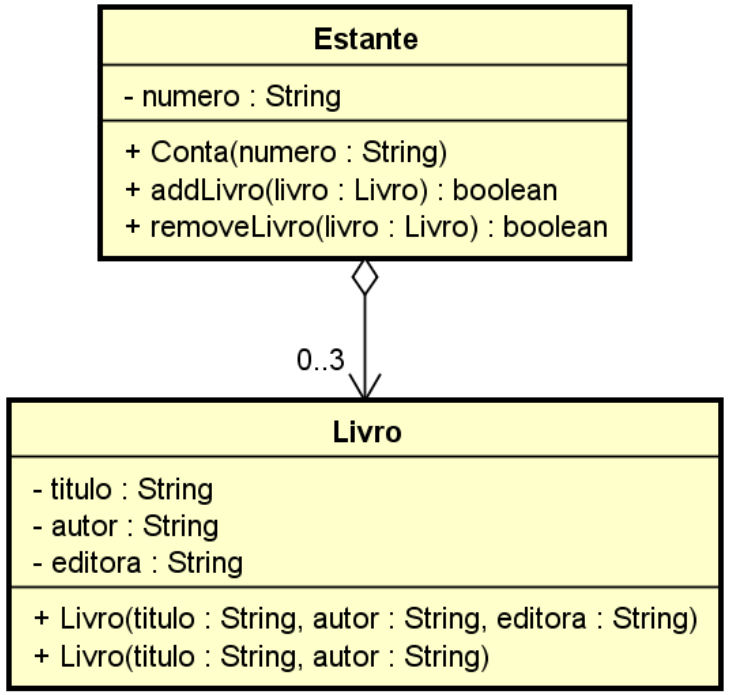


Agregação: Multiplicidade 0..3

Implementando a Classe Estante

Estante.java

```java
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;
    }

    public boolean addLivro(Livro livro) {
        boolean sucesso = false;

        if (livro!= null && listaLivro.size() < 3 ) {
            listaLivro.add(livro);
            sucesso = true;
        }
        return sucesso;
    }

```



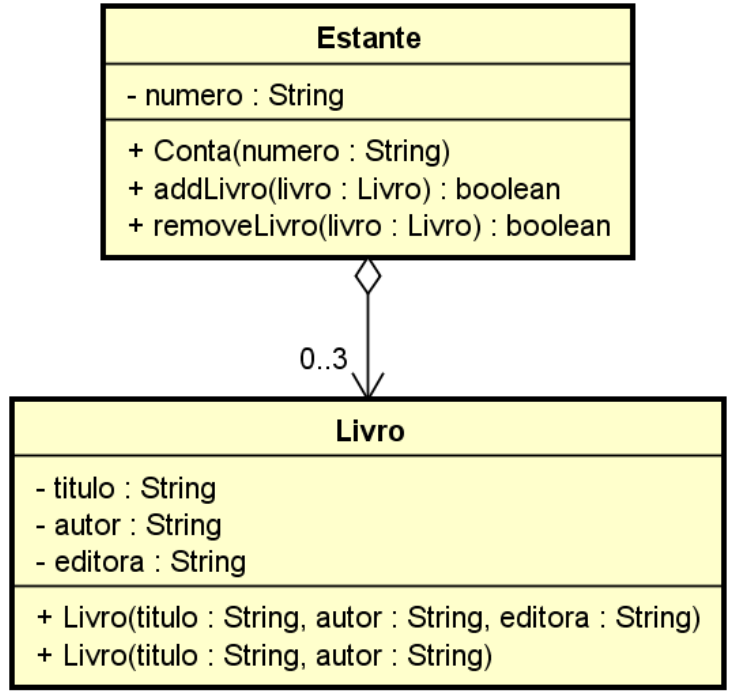
CONTINUA >>

Agregação: Multiplicidade 0..3

Implementando a Classe Estante

Estante.java

```
23      public boolean removerLivro(Livro livro) {
24          boolean sucesso = false;
25
26          if (listaLivro.size() > 0 && listaLivro.contains(livro)) {
27              listaLivro.remove(livro);
28              sucesso = true;
29          }
30          return sucesso;
31      }
32
33      public String getNumero() {
34          return numero;
35      }
36
37      public void setNumero(String numero) {
38          this.numero = numero;
39      }
40
41      public List<Livro> getListaLivro() {
42          List<Livro> listaRetorno = new ArrayList<Livro>();
43          listaRetorno.addAll(listaLivro);
44          return listaRetorno;
45      }
46
47      @Override
48      public String toString() {
49          return "Estante [numero=" + numero + ", listaLivro=" + listaLivro + "]";
50      }
51  }
```



CONTINUA >>

Agregação: Multiplicidade 0..3

Programa Principal

TesteEstante.java

```
1  public class TesteEstante {
2
3      public static void main(String[] args) {
4
5          Livro lv1 = new Livro("Programação Java", "Dory", "Câmpus");
6          Livro lv2 = new Livro("UML e Padrões", "Larman", "Bookman");
7          Livro lv3 = new Livro("Php", "Deitel", "Pearson");
8          Livro lv4 = new Livro("Java", "Barnes", "Pearson");
9
10         Estante e = new Estante("AB-123");
11
12         e.addLivro(lv1);
13         e.addLivro(lv2);
14         e.addLivro(lv3);
15         e.addLivro(lv4);
16         System.out.println(e);
17
18         e.removerLivro(lv1);
19         e.addLivro(lv4);
20         System.out.println(e);
21     }
22 }
```

Saída do Programa

Console

```
<terminated> TesteEstante [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (27 de set de 2018 04:28:18)
Estante [numero=AB-123, listaLivro=[
    Livro [titulo=Programação Java, autor=Dory, editora=Câmpus],
    Livro [titulo=UML e Padrões, autor=Larman, editora=Bookman],
    Livro [titulo=Php, autor=Deitel, editora=Pearson]]]
Estante [numero=AB-123, listaLivro=[
    Livro [titulo=UML e Padrões, autor=Larman, editora=Bookman],
    Livro [titulo=Php, autor=Deitel, editora=Pearson],
    Livro [titulo=Java, autor=Barnes, editora=Pearson]]]
```