

**INSTITUTO FEDERAL**  
Goiás

Instituto Federal de Goiás  
Câmpus Goiânia

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

# Associação de Classes Composição

Prof. Ms. Renan Rodrigues de Oliveira  
Goiânia - GO

# Composição

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

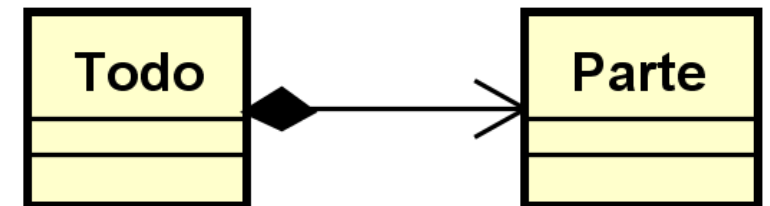


**Este relacionamento é caracterizado pela parte poder existir somente compondo o todo, ou seja, a parte deve ser criada dentro do vínculo.**

- ▶ A classe que compõe (parte) possui o mesmo tempo de vida da classe composta (todo);
- ▶ Se a classe composta morrer, suas partes também morrerão.

## Exemplo

- ▶ Um pedido e um item. Um pedido é composto por itens. Um item faz parte de um pedido, porém não existe fora do universo do pedido. Caso o pedido seja encerrado ou deixe de existir, o item do pedido não existirá mais.



# Composição



**A estrutura de dados utilizada, bem como o local do vínculo dependerão da multiplicidade.**

- ▶ Partes que compõem um todo não estarão criadas antes. Sua referência será conhecida somente dentro do todo;
- ▶ Os argumentos/parâmetros de métodos e/ou construtores que realizarão o vínculo serão os atributos da parte. Crie o objeto da classe parte dentro destas estruturas;
- ▶ Assim sendo, a única entidade que vai conhecer a referência da parte, quando vinculada, é o todo.

# Agregação x Composição



**Na composição, leia-se: um objeto é composto por outros objetos.**

- ▶ O todo é responsável pela criação e destruição de suas partes.
- ▶ Quando o objeto todo é destruído, suas partes também são.



**Na agregação, o todo e as partes são independentes.**

- ▶ Ao destruímos o objeto todo as partes permanecem na memória, por terem sido criadas fora do escopo da classe todo.

# Composição: Multiplicidade 0..1

## Exemplo: Um Fornecedor e seu Contato

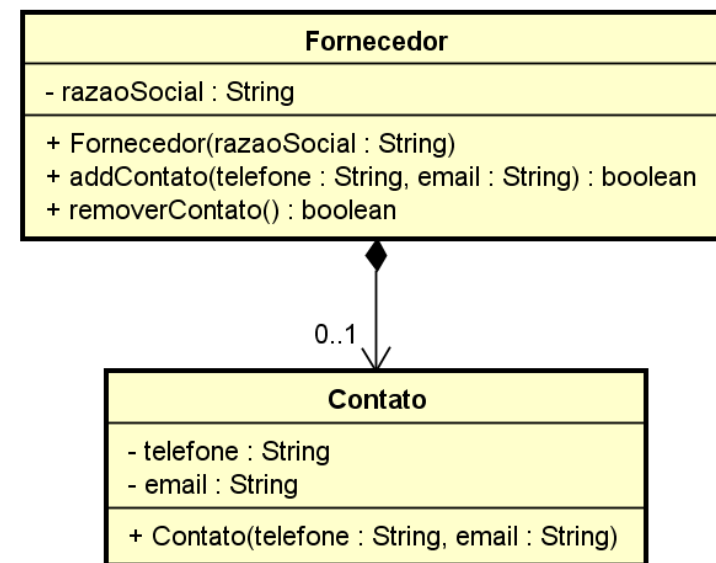


O Fornecedor pode ter um Contato. Se o Fornecedor for removido, o Contato também deve ser removido.

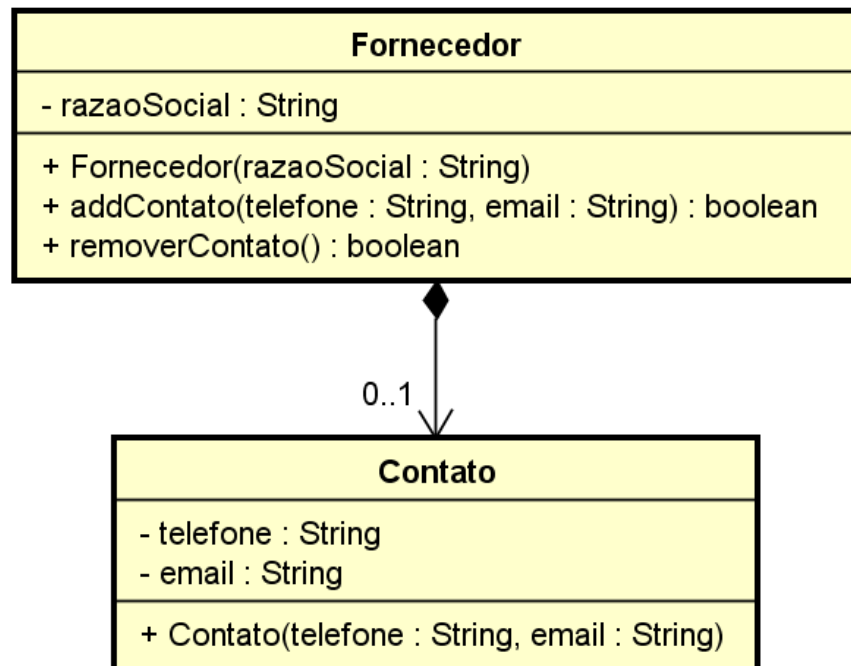


Na multiplicidade 0..1, o “todo” pode nascer sem possuir nenhuma parte.

- ▶ 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” depende do tempo da classe “todo”.



# Composição: Multiplicidade 0..1



## Implementação

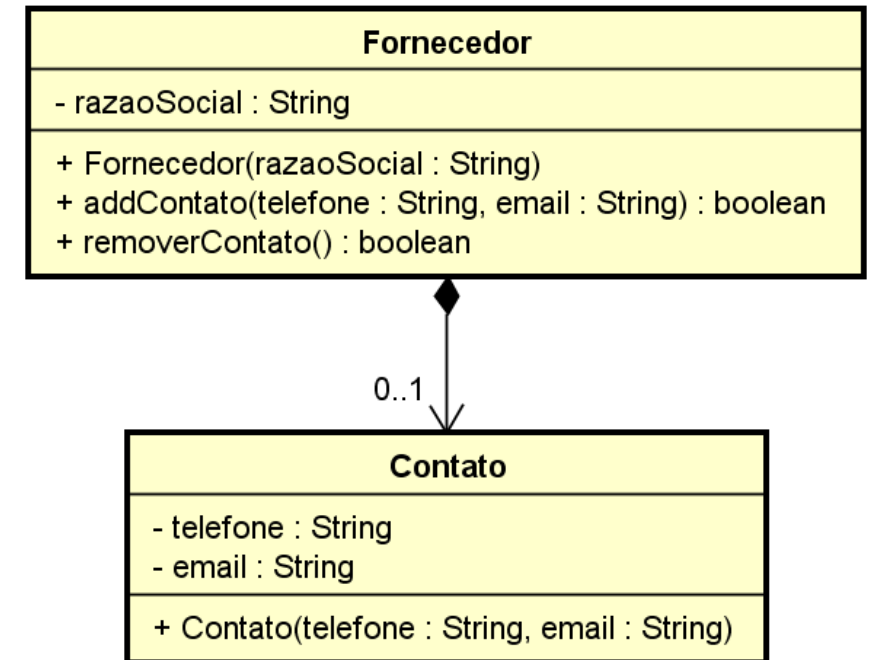
- ▶ Um Contato compõe um Fornecedor.
  - ▶ O Fornecedor pode ter 0 ou 1 Contato;
  - ▶ O vínculo se dará no método `addContato()`;
- ▶ Primeiro programe as partes, depois o relacionamento.
  - ▶ Crie o Contato somente dentro do método do vínculo.

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

# Composição: Multiplicidade 0..1

## Implementando a Classe Contato

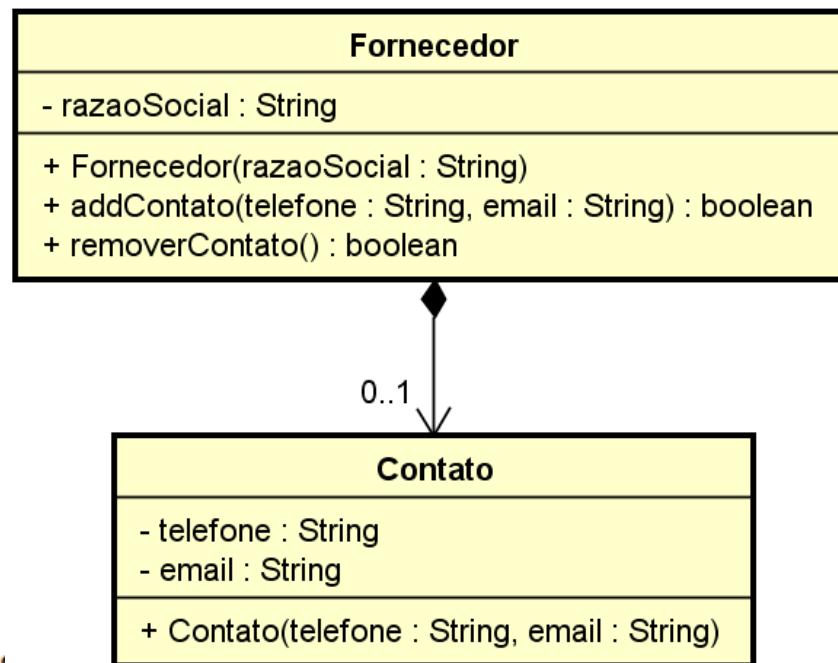
```
1 package br.com.renanrodrigues.composicao;
2
3 public class Contato {
4
5     private String telefone;
6     private String email;
7
8     public Contato(String telefone, String email) {
9         this.telefone = telefone;
10        this.email = email;
11    }
12
13    public String getTelefone() {
14        return telefone;
15    }
16
17    public String getEmail() {
18        return email;
19    }
20
21    @Override
22    public String toString() {
23        return "Contato [telefone=" + telefone + ", email=" + email + "]";
24    }
25 }
```



# Composição: Multiplicidade 0..1

## Implementando a Classe Fornecedor

```
1 package br.com.renanrodrigues.composicao;
2
3 public class Fornecedor {
4
5     private String razaoSocial;
6     private Contato contato;
7
8     public Fornecedor(String razaoSocial) {
9         this.razaoSocial = razaoSocial;
10    }
11
12    public boolean addContato(String telefone, String email) {
13        boolean sucesso = false;
14
15        if (this.contato == null) {
16            contato = new Contato(telefone, email);
17            return true;
18        }
19
20        return sucesso;
21    }
22 }
```



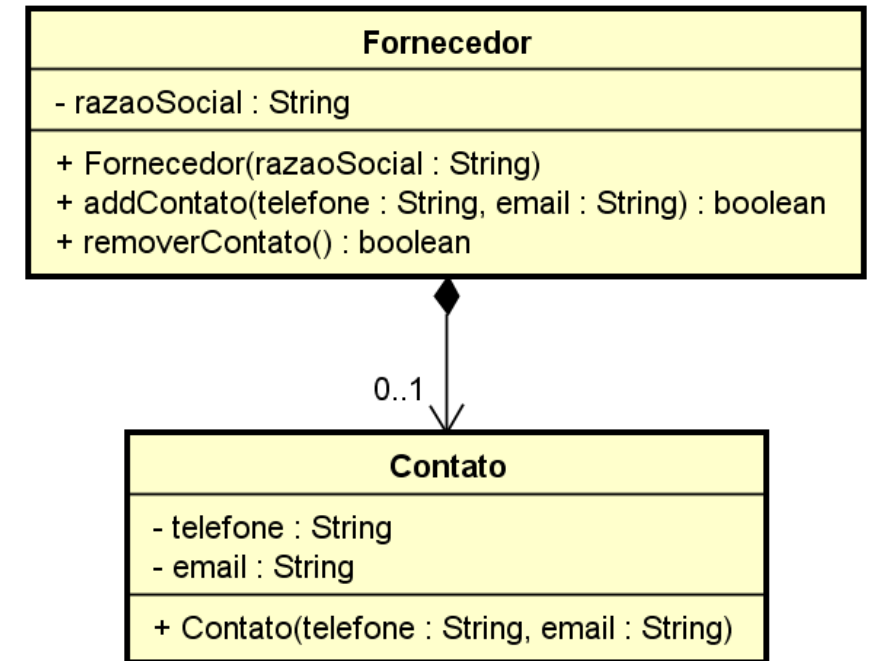


# Composição: Multiplicidade 0..1

## Implementando a Classe Fornecedor

```
23 public boolean removerContato() {  
24     boolean sucesso = false;  
25  
26     if (contato != null) {  
27         contato = null;  
28         sucesso = true;  
29     }  
30  
31     return sucesso;  
32 }
```

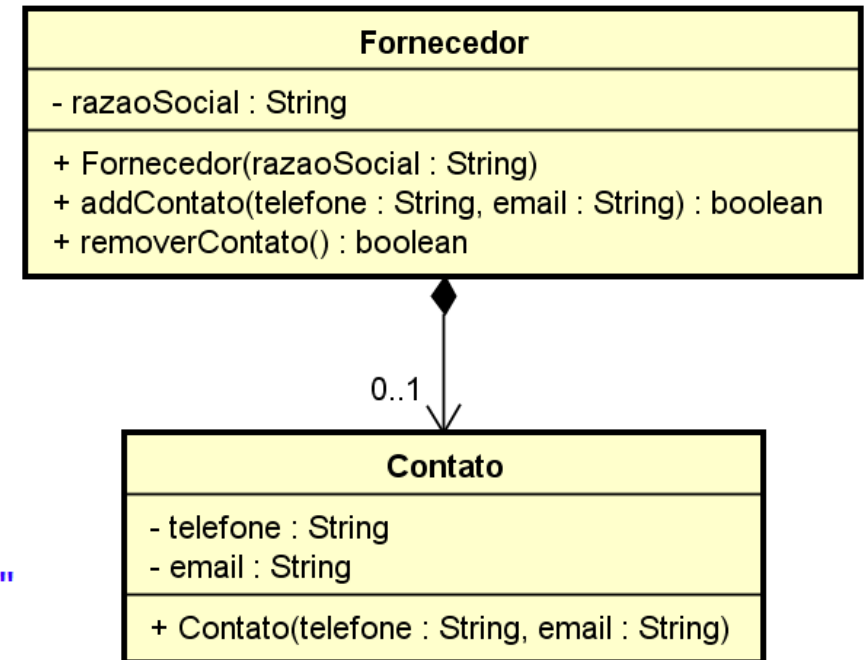
CONTINUA >>



# Composição: Multiplicidade 0..1

## Implementando a Classe Fornecedor

```
34 public String getRazaoSocial() {  
35     return razaoSocial;  
36 }  
37  
38 public Contato getContato() {  
39     return contato;  
40 }  
41  
42 @Override  
43 public String toString() {  
44     return "Fornecedor [razaoSocial=" + razaoSocial + ", contato=" +  
45         contato + "]  
46 }  
47  
48 }
```



# Composição: Multiplicidade 0..1

## Programa Principal

```
1 package br.com.renanrodrigues.composicao;
2
3 public class TesteFornecedor {
4
5     public static void main(String[] args) {
6
7         Fornecedor f = new Fornecedor("Distribuidora ABC");
8         System.out.println(f);
9
10        f.addContato("2345-6789", "abc@abc.com");
11        f.addContato("0000-0000", "kkk@kkk.com");
12        System.out.println(f);
13
14        f.removerContato();
15        f.addContato("9876-5432", "contato@abc.com");
16        System.out.println(f);
17    }
18 }
```

Saída do Programa

```
Fornecedor [razaoSocial=Distribuidora ABC, contato=null]
Fornecedor [razaoSocial=Distribuidora ABC, contato=Contato [telefone=2345-6789, email=abc@abc.com]]
Fornecedor [razaoSocial=Distribuidora ABC, contato=Contato [telefone=9876-5432, email=contato@abc.com]]
```

# Composição: Multiplicidade 1

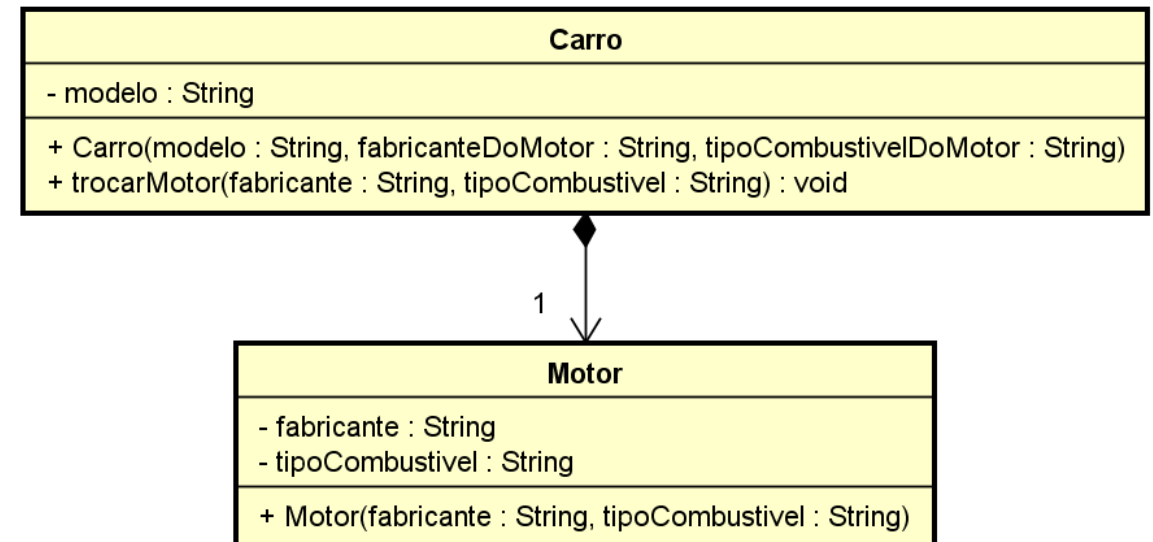
## Exemplo: Um Carro e seu Motor

O Carro possui um Motor. Se o Carro for destruído, o Motor também é destruído.

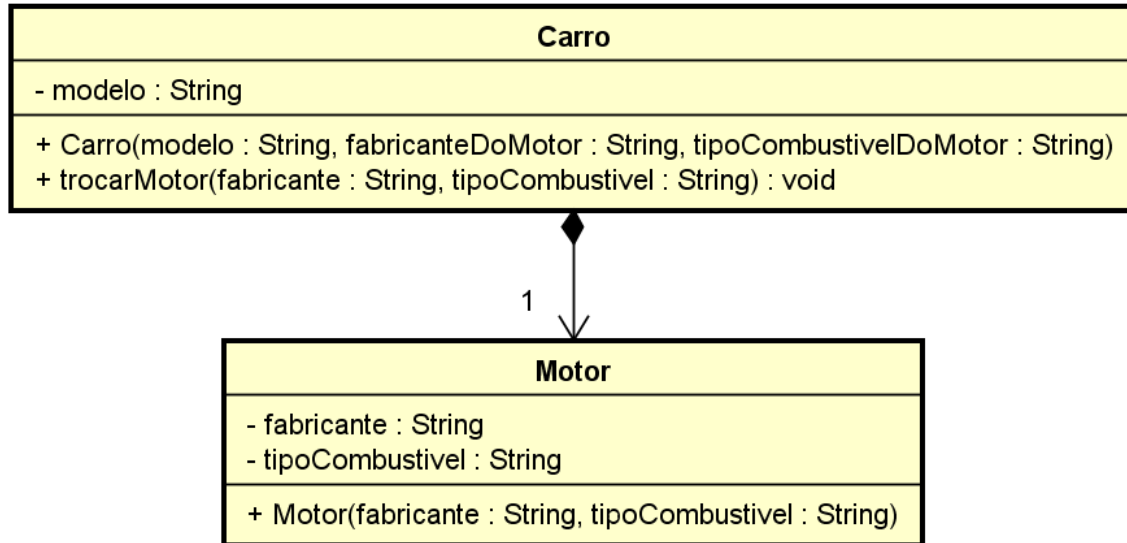


Na multiplicidade 1, o “todo” DEVE nascer possuindo uma parte.

- ▶ Assim sendo, neste caso, a “parte” deve ser criada no momento da criação do “todo”;
- ▶ Ao longo de seu ciclo de vida, uma “parte” pode ser substituída, mas nunca removida.
- ▶ Tempo de vida da classe "parte" depende do tempo da classe "todo".



# Composição: Multiplicidade 1



## Implementação

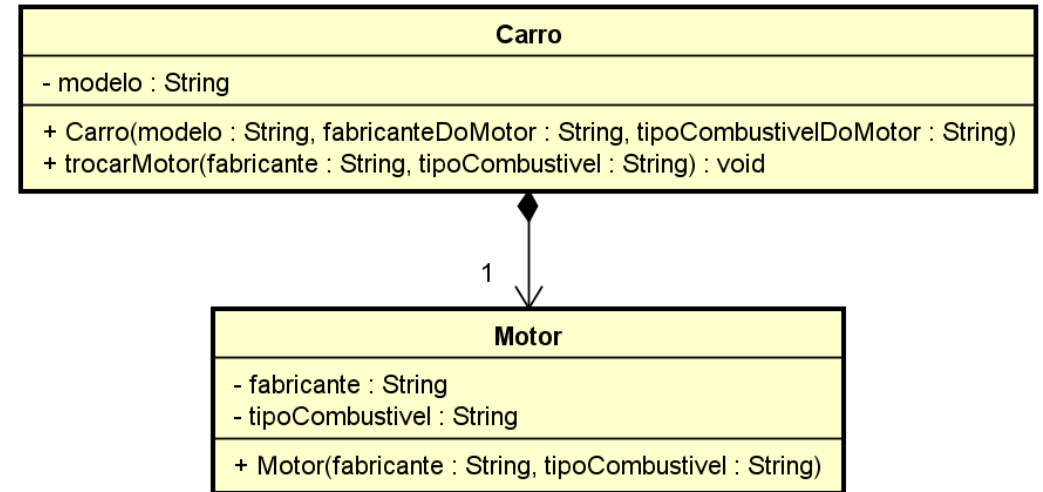
- ▶ Um Motor compõe um Carro.
  - ▶ O Carro deve ter um Motor;
  - ▶ O vínculo se dará construtor.
- ▶ Primeiro programe as partes, depois o relacionamento.
  - ▶ Crie o Motor somente dentro do construtor do Carro.

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

# Composição: Multiplicidade 1

## Implementando a Classe Motor

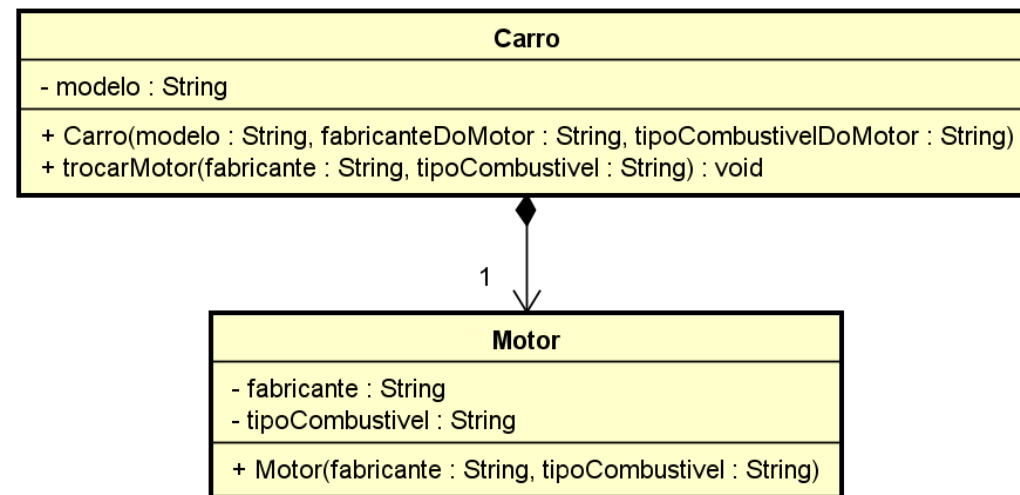
```
1 package br.com.renanrodrigues.composicao;
2
3 public class Motor {
4
5     private String fabricante;
6     private String tipoCombustivel;
7
8     public Motor(String fabricante, String tipoCombustivel)
9         this.fabricante = fabricante;
10        this.tipoCombustivel = tipoCombustivel;
11    }
12
13    public String getFabricante() {
14        return fabricante;
15    }
16
17    public String getTipoCombustivel() {
18        return tipoCombustivel;
19    }
20
21    @Override
22    public String toString() {
23        return "Motor [fabricante=" + fabricante + ", tipoCombustivel="
24            + tipoCombustivel + "];"
25    }
26 }
```



# Composição: Multiplicidade 1

## Implementando a Classe Carro

```
1 package br.com.renanrodrigues.composicao;
2
3 public class Carro {
4
5     private String modelo;
6     private Motor motor;
7
8     public Carro(String modelo, String fabricanteDoMotor, String tipoDoCombustivelDoMotor) {
9         this.modelo = modelo;
10        this.motor = new Motor(fabricanteDoMotor, tipoDoCombustivelDoMotor);
11    }
12
13    public void trocarMotor(String fabricante, String tipoDoCombustivel) {
14        this.motor = new Motor(fabricante, tipoDoCombustivel);
15    }
16 }
```

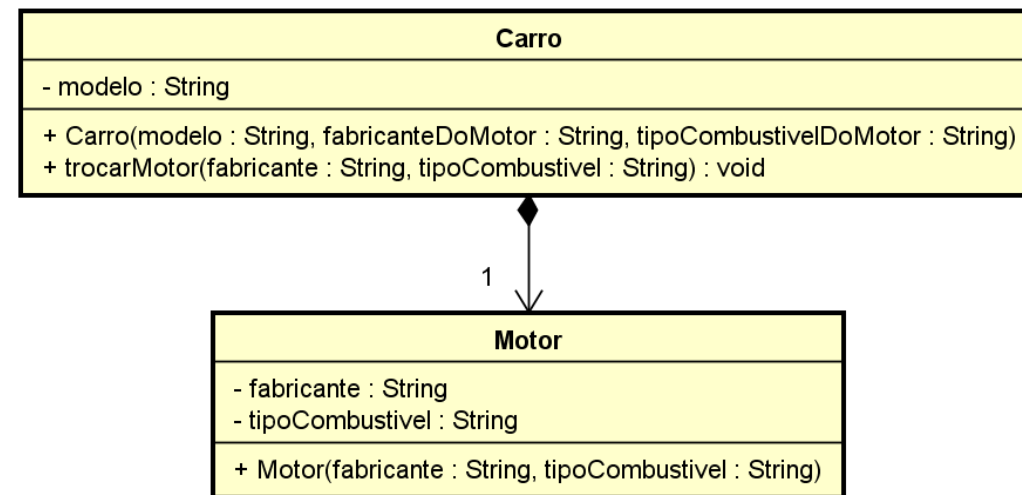


CONTINUA >>

# Composição: Multiplicidade 1

## Implementando a Classe Carro

```
17 public String getModelo() {  
18     return modelo;  
19 }  
20  
21 public Motor getMotor() {  
22     return motor;  
23 }  
24  
25 @Override  
26 public String toString() {  
27     return "Carro [modelo=" + modelo + ", motor=" + motor + "];"  
28 }  
29 }
```

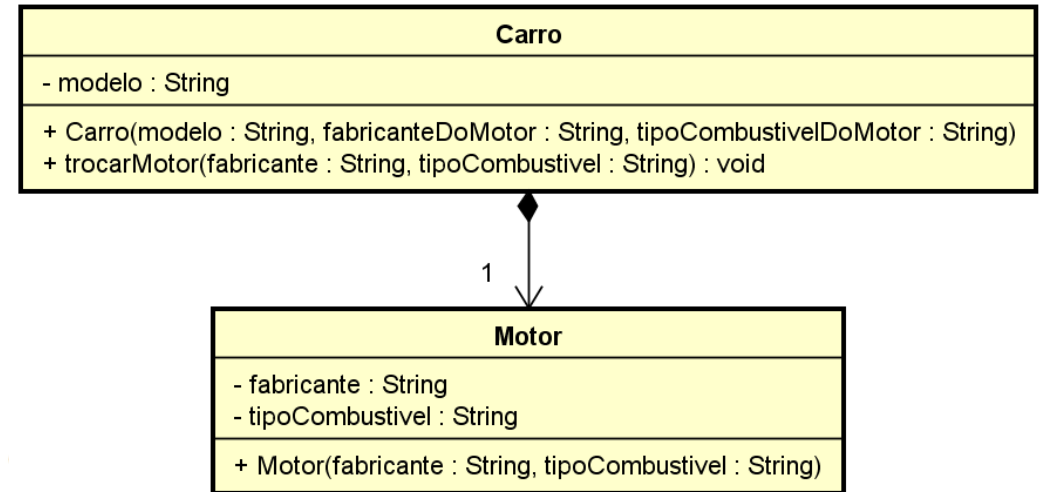




# Composição: Multiplicidade 1

## Programa Principal

```
1 package br.com.renanrodrigues.composicao;
2
3 public class TesteCarro {
4
5     public static void main(String[] args)
6
7         Carro c = new Carro("Civic", "Honda", "A/G");
8         System.out.println(c.toString());
9
10        c.trocarMotor("Ferrari", "G");
11        System.out.println(c.toString());
12
13    }
14 }
```



Saída do Programa

```
Carro [modelo=Civic, motor=Motor [fabricante=Honda, tipoCombustivel=A/G]]
Carro [modelo=Civic, motor=Motor [fabricante=Ferrari, tipoCombustivel=G]]
```

# Composição: Multiplicidade 0..N

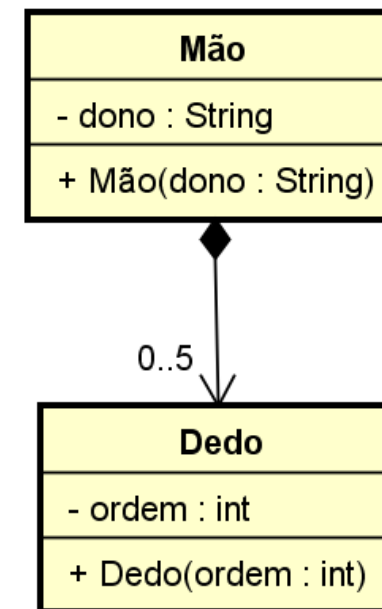
## Exemplo: Uma Mão e seus Dedos

Uma Mão possui no máximo 5 Dedos. Se a Mão for removida, o Dedo também é removido.

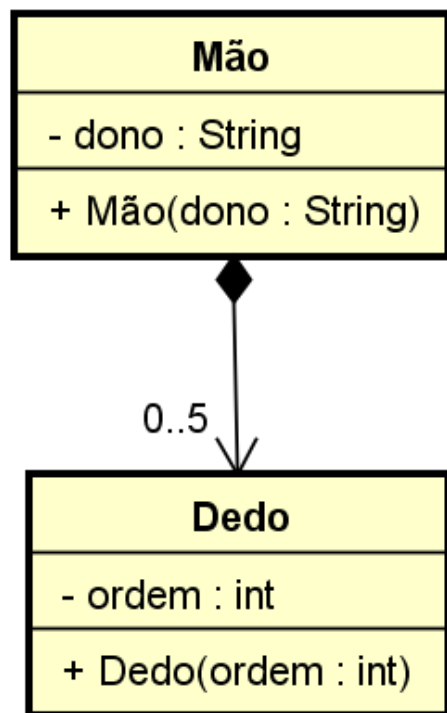


Na multiplicidade 0..\*, o “todo” pode nascer sem possuir nenhuma parte.

- ▶ Assim sendo, neste caso, a “parte” deve ser criada no momento da criação do “todo”;
- ▶ Ao longo de seu ciclo de vida, N “partes” podem compor o “todo”, com o “todo” sabendo quais “partes” estarão se relacionando com ele;



# Composição: Multiplicidade 0..N



## Implementação

- ▶ Um Dedo compõe uma Mão.
  - ▶ A Mão pode ter 0 ou 5 Dedos;
  - ▶ O vínculo se dará no método `addDedo()`.
- ▶ Primeiro programe as partes, depois o relacionamento.
  - ▶ Crie o Dedo somente dentro do método de vínculo.

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

# Composição: Multiplicidade 0..N

## Implementando a Classe Dedo

```
1 package br.com.renanrodrigues.composicao;
```

```
2
```

```
3 public class Dedo {
```

```
4
```

```
5     private int ordem;
```

```
6
```

```
7     public Dedo(int ordem) {
```

```
8         this.ordem = ordem;
```

```
9     }
```

```
10
```

```
11     public int getOrdem() {
```

```
12         return ordem;
```

```
13     }
```

```
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         Dedo other = (Dedo) obj;
```

```
24         if (ordem != other.ordem)
```

```
25             return false;
```

```
26         return true;
```

```
27     }
```

```
28
```

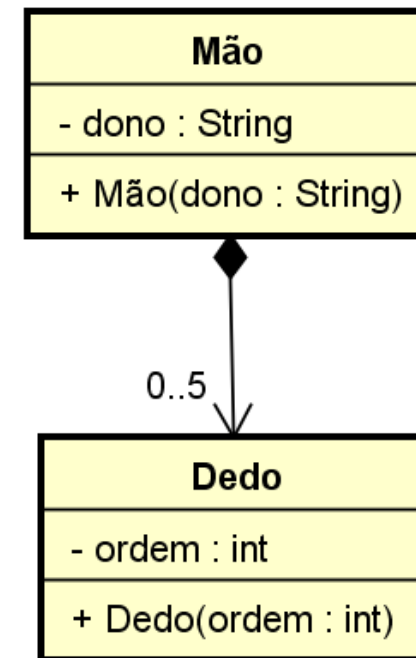
```
29     @Override
```

```
30     public String toString() {
```

```
31         return "Dedo [ordem=" + ordem + "];
```

```
32     }
```

```
33 }
```

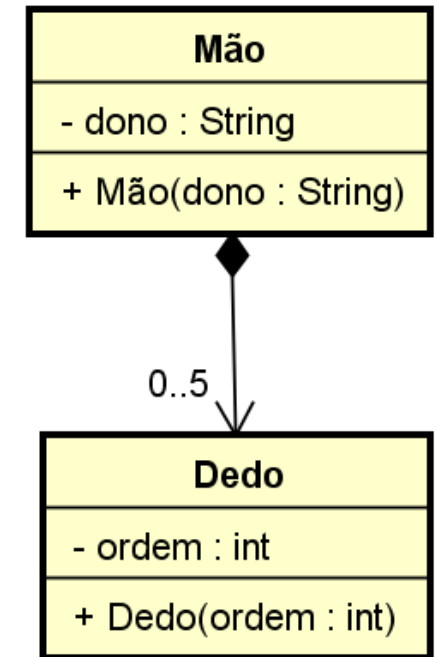


# Composição: Multiplicidade 0..N

## Implementando a Classe Mão

```
1 package br.com.renanrodrigues.composicao;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Mao {
7
8     private String dono;
9     private List<Dedo> listaDedo = new ArrayList<Dedo>();
10
11     public Mao(String dono) {
12         this.dono = dono;
13     }
14 }
```

CONTINUA >>

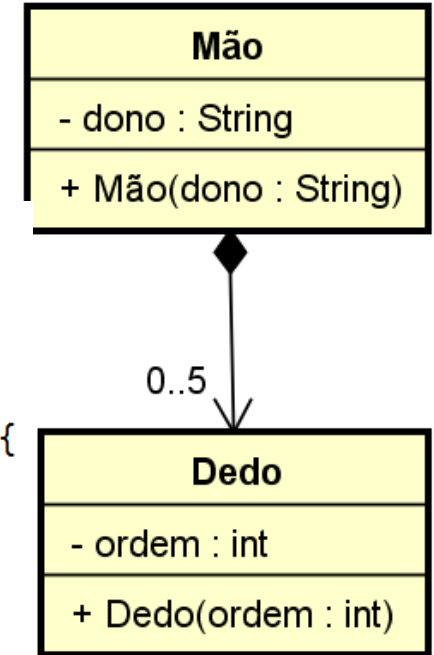


# Composição: Multiplicidade 0..N

## Implementando a Classe Mão

```
15 public boolean addDedo(int ordem) {  
16     boolean sucesso = false;  
17  
18     Dedo dedo = new Dedo(ordem);  
19  
20     if ( (ordem >= 1 && ordem <=5) && !listaDedo.contains(dedo) && listaDedo.size() < 4 ) {  
21         listaDedo.add(dedo);  
22         sucesso = true;  
23     }  
24     return sucesso;  
25 }
```

```
27 public boolean removerDedo(int ordem) {  
28     boolean sucesso = false;  
29  
30     Dedo dedo = new Dedo(ordem);  
31  
32     if (listaDedo.size() > 0 && listaDedo.contains(dedo)) {  
33         listaDedo.remove(dedo);  
34         sucesso = true;  
35     }  
36  
37     return sucesso;  
38 }
```

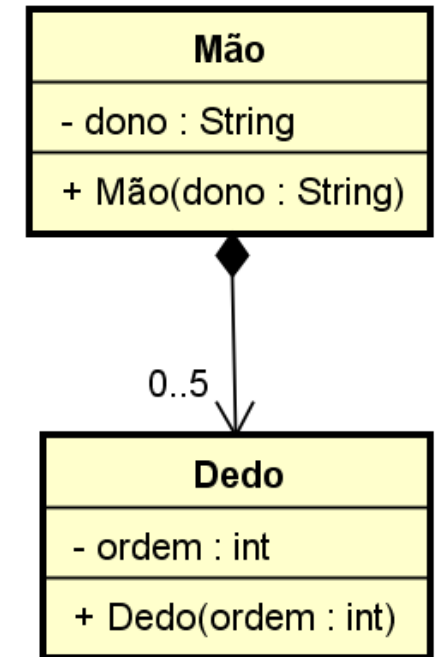


CONTINUA >>

# Composição: Multiplicidade 0..N

## Implementando a Classe Mão

```
40 public String getDono() {  
41     return dono;  
42 }  
43  
44 public List<Dedo> getListado() {  
45     return listaDedo;  
46 }  
47  
48 @Override  
49 public String toString() {  
50     return "Mao [dono=" + dono + ", listaDedo=" + listaDedo + "];"  
51 }  
52 }
```



CONTINUA >>

# Composição: Multiplicidade 0..N

## Programa Principal

```
1 package br.com.renanrodrigues.composicao;
2
3 public class TesteMao {
4
5     public static void main(String[] args) {
6
7         Mao m = new Mao("Renan");
8         System.out.println(m);
9
10        m.addDedo(1);
11        m.addDedo(2);
12        m.addDedo(2);
13        m.addDedo(6);
14        m.addDedo(3);
15        System.out.println(m);
16
17        m.removerDedo(2);
18        System.out.println(m);
19
20        m.addDedo(2);
21        System.out.println(m);
22    }
23 }
```

### Saída do Programa

```
Mao [dono=Renan, listaDedo=[]]
Mao [dono=Renan, listaDedo=[Dedo [ordem=1], Dedo [ordem=2], Dedo [ordem=3]]]
Mao [dono=Renan, listaDedo=[Dedo [ordem=1], Dedo [ordem=3]]]
Mao [dono=Renan, listaDedo=[Dedo [ordem=1], Dedo [ordem=3], Dedo [ordem=2]]]
```



# Composição: Multiplicidade 0..\*

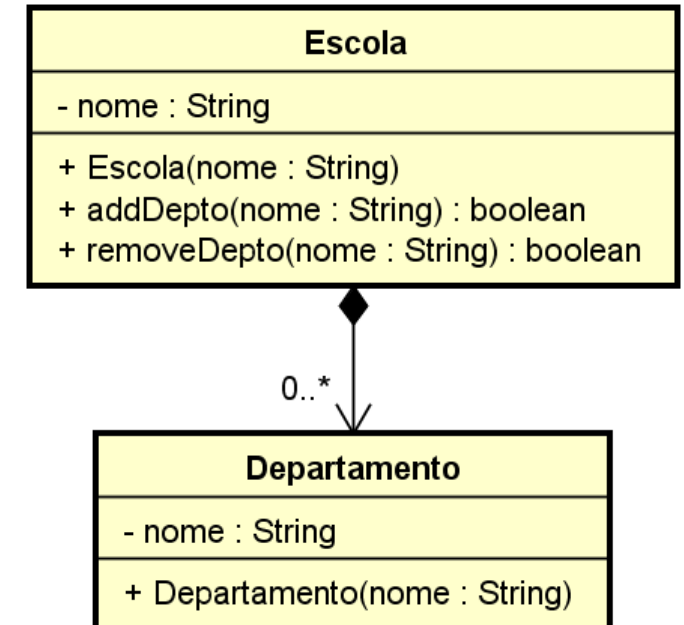
## Exemplo: Uma Escola e seus Departamentos

Uma Escola possui vários Departamentos. Se a Escola for destruída, o Departamento também é destruído.

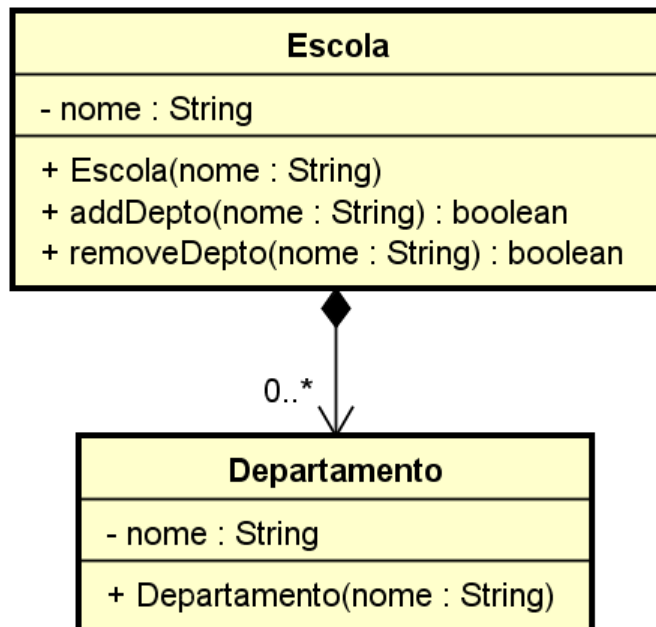


Na multiplicidade 0..\*, o “todo” pode nascer sem possuir nenhuma parte.

- ▶ Ao longo de seu ciclo de vida, N “partes” podem compor o “todo”, com o “todo” sabendo quais “partes” estarão se relacionando com ele;
- ▶ Tempo de vida da classe "parte" depende do tempo da classe "todo".



# Composição: Multiplicidade 0..\*



## Implementação

- ▶ Um Departamento compõe uma Escola.
  - ▶ A Escola pode ter vários Departamentos.
  - ▶ O vínculo se dará no método `addDepartamento()`.
- ▶ Primeiro programe as partes, depois o relacionamento.
  - ▶ Crie o Departamento somente dentro do método de vínculo.

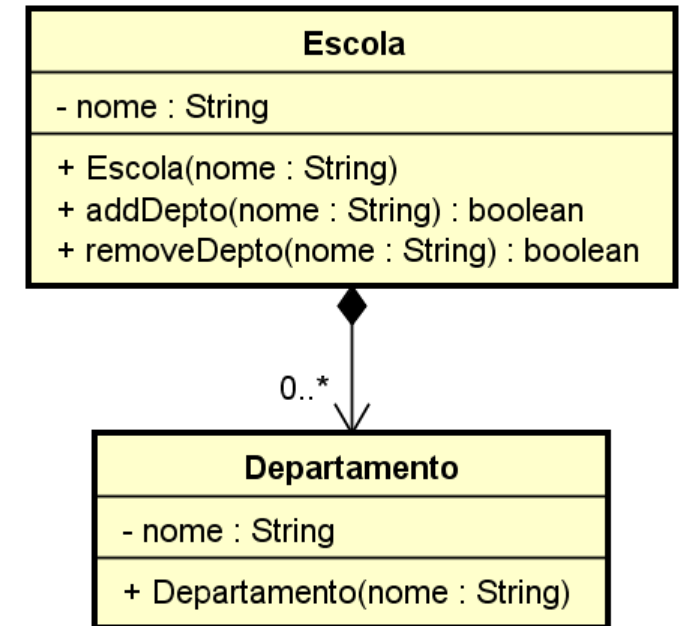
**É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.**

# Composição: Multiplicidade 0..\*

## Implementando a Classe Departamento

```
1 package br.com.renanrodrigues.composicao;
2
3 public class Departamento {
4
5     private String nome;
6
7     public Departamento(String nome) {
8         this.nome = nome;
9     }
10
11     public String getNome() {
12         return nome;
13     }
14 }
```

```
15
16 @Override
17 public boolean equals(Object
18     if (this == obj)
19         return true;
20     if (obj == null)
21         return false;
22     if (getClass() != obj.getClass())
23         return false;
24     Departamento other = (Departamento) obj;
25     if (nome == null) {
26         if (other.nome != null)
27             return false;
28     } else if (!nome.equals(other.nome))
29         return false;
30     return true;
31 }
32
33 @Override
34 public String toString() {
35     return "Departamento [nome=" + nome + "]";
36 }
```

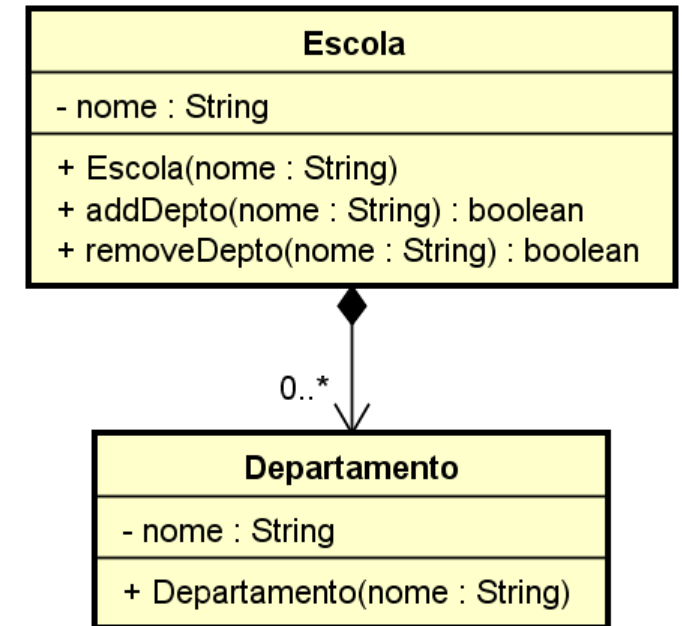


# Composição: Multiplicidade 0..\*

## Implementando a Classe Escola

```
1 package br.com.renanrodrigues.composicao;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Escola {
7
8     private String nome;
9     private List<Departamento> listaDepto = new ArrayList<Departamento>();
10
11     public Escola(String nome) {
12         this.nome = nome;
13     }
14 }
```

CONTINUA >>

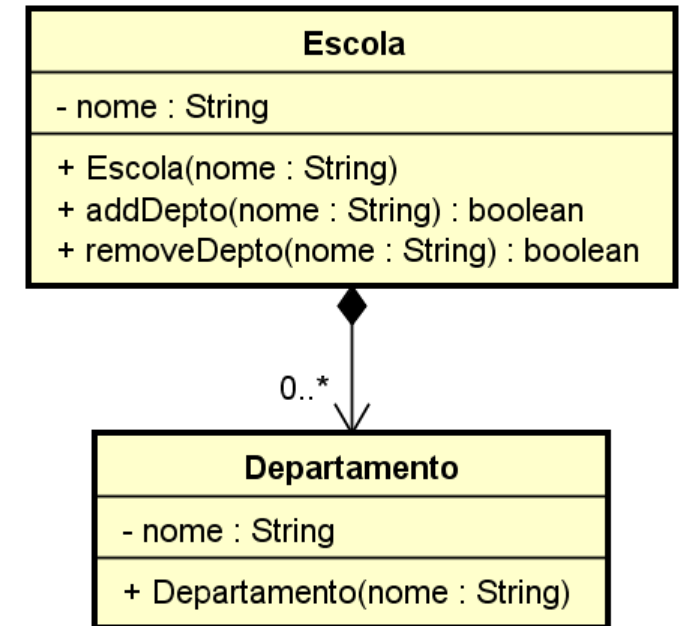


# Composição: Multiplicidade 0..\*

## Implementando a Classe Escola

```
15 public boolean addDepto(String nome) {  
16     boolean sucesso = false;  
17  
18     Departamento depto = new Departamento(nome);  
19  
20     if (!listaDepto.contains(depto)) {  
21         listaDepto.add(depto);  
22         sucesso = true;  
23     }  
24     return sucesso;  
25 }
```

```
27 public boolean removeDepto(String nome) {  
28     boolean sucesso = false;  
29  
30     Departamento depto = new Departamento(nome);  
31  
32     if (listaDepto.size() > 0 && listaDepto.contains(depto)) {  
33         listaDepto.remove(depto);  
34         sucesso = true;  
35     }  
36  
37     return sucesso;  
38 }
```



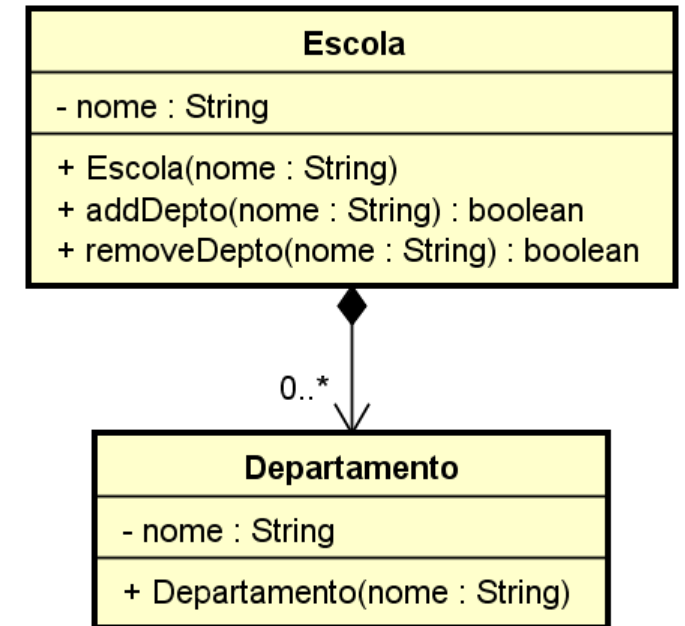
CONTINUA >>

# Composição: Multiplicidade 0..\*

## Implementando a Classe Escola

```
40 public String getNome() {  
41     return nome;  
42 }  
43  
44 public List<Departamento> getListaDepto() {  
45     return listaDepto;  
46 }  
47  
48 @Override  
49 public String toString() {  
50     return "Escola [nome=" + nome + ", listaDepto=" + listaDepto + "];  
51 }  
52 }
```

CONTINUA >>



# Composição: Multiplicidade 0..\*

## Programa Principal

```
1 package br.com.renanrodrigues.composicao;
2
3 public class TesteEscola {
4
5     public static void main(String[] args) {
6
7         Escola e = new Escola("IFG");
8         System.out.println(e);
9
10        e.addDepto("Depto I");
11        e.addDepto("Depto II");
12        e.addDepto("Depto III");
13        e.addDepto("Depto IV");
14        System.out.println(e);
15
16        e.removeDepto("Depto II");
17        e.removeDepto("Depto III");
18        System.out.println(e);
19
20    }
21 }
```

### Saída do Programa

```
Escola [nome=IFG, listaDepto=[]]
Escola [nome=IFG, listaDepto=[Departamento [nome=Depto I], Departamento [nome=Depto II],
                                Departamento [nome=Depto III],
                                Departamento [nome=Depto IV]]]
Escola [nome=IFG, listaDepto=[Departamento [nome=Depto I], Departamento [nome=Depto IV]]]
```



# Composição: Multiplicidade 1..\*

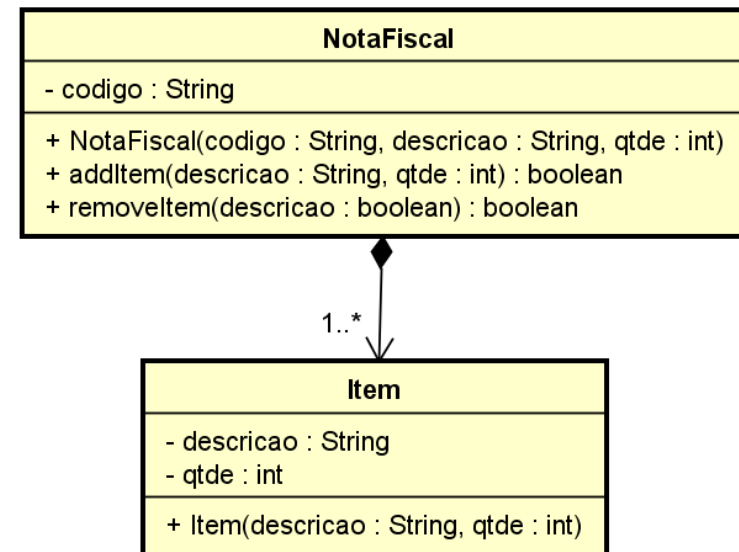
## Exemplo: Uma Nota Fiscal e seus Itens

Uma Nota Fiscal possui um ou mais Itens. Se a Nota Fiscal for destruída, os Itens também são destruídos.



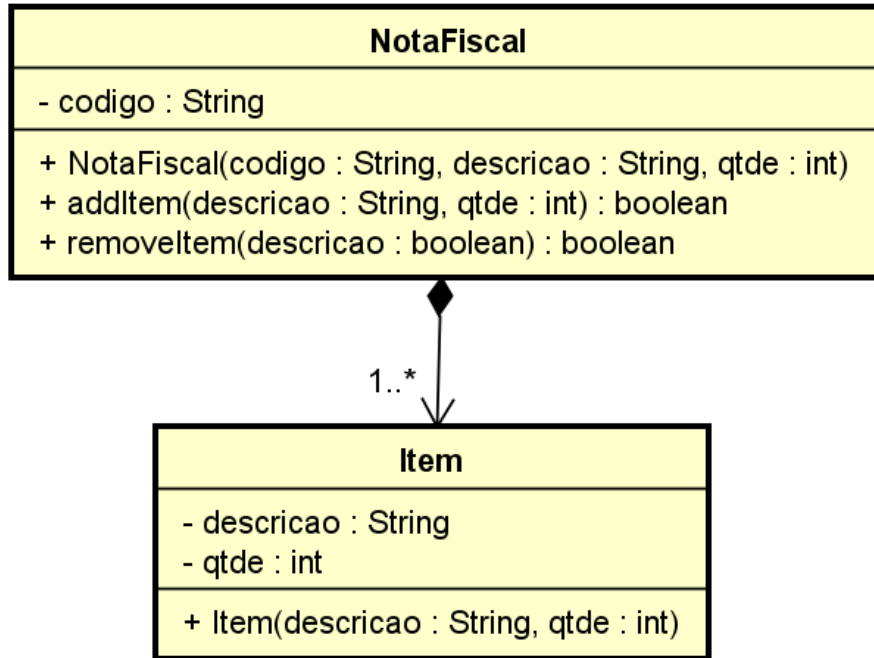
**Na multiplicidade 1..\*, o “todo” DEVE nascer possuindo uma parte.**

- ▶ Ao longo de seu ciclo de vida, muitas “partes” podem compor o “todo”, com o “todo” sabendo quais “partes” estarão se relacionando com ele.
- ▶ Tempo de vida da classe "parte" depende do tempo da classe "todo".

[illegible]



# Composição: Multiplicidade 1..\*



## Implementação

- ▶ Um Item compõe uma Nota Fiscal.
  - ▶ A Nota Fiscal pode ter um ou mais itens.
  - ▶ O vínculo se dará no método `addItem()`.
- ▶ Primeiro programe as partes, depois o relacionamento.
  - ▶ Crie o Item somente dentro construtor e método de vínculo.

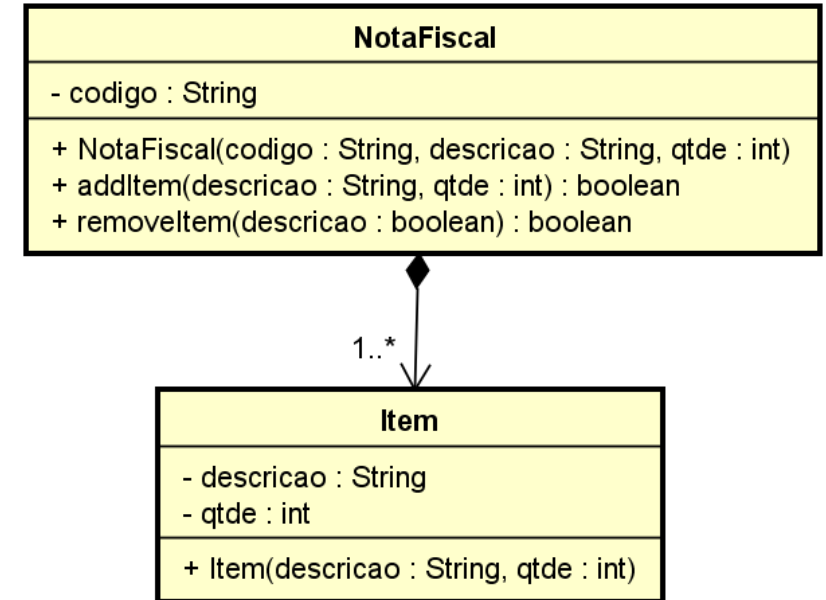
**É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.**

# Composição: Multiplicidade 1..\*

## Implementando a Classe Item

```
1 package br.com.renanrodrigues.composicao;
2
3 public class Item {
4
5     private String descricao;
6     private int qtde;
7
8     public Item(String descricao, int qtde) {
9         this.descricao = descricao;
10        this.qtde = qtde;
11    }
12
13    public Item(String descricao) {
14        this.descricao = descricao;
15    }
16
17    public String getDescricao() {
18        return descricao;
19    }
20
21    public int getQtde() {
22        return qtde;
23    }
```

CONTINUA >>

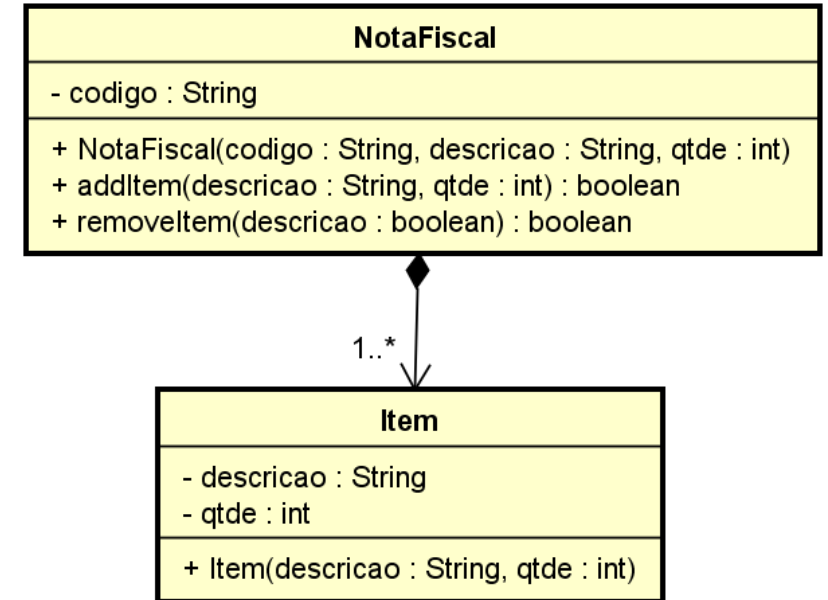


# Composição: Multiplicidade 1..\*

## Implementando a Classe Item

```
25 @Override
26 public boolean equals(Object obj) {
27     if (this == obj)
28         return true;
29     if (obj == null)
30         return false;
31     if (getClass() != obj.getClass())
32         return false;
33     Item other = (Item) obj;
34     if (descricao == null) {
35         if (other.descricao != null)
36             return false;
37     } else if (!descricao.equals(other.descricao))
38         return false;
39     return true;
40 }

@Override
public String toString() {
    return "Item [descricao=" + descricao + ", qtde=" + qtde + "]";
}
```

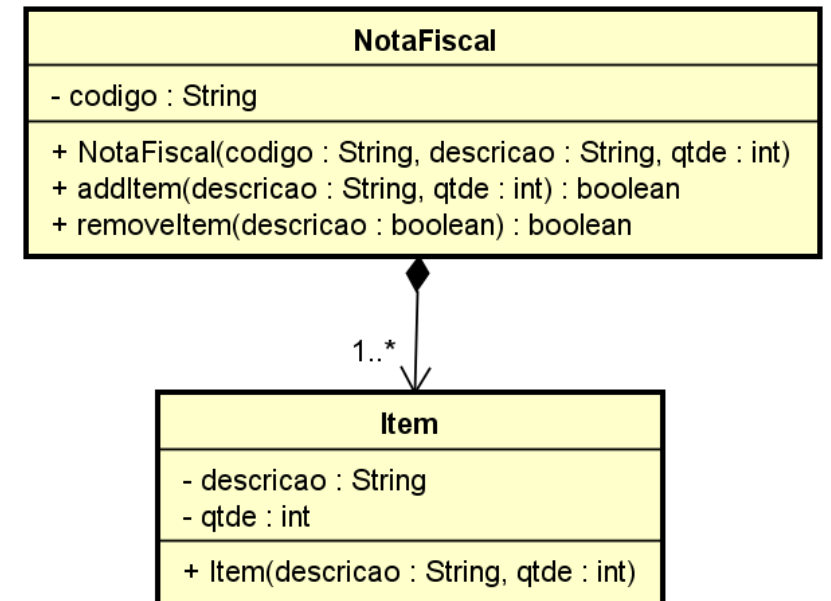


CONTINUA >>

# Composição: Multiplicidade 1..\*

## Implementando a Classe NotaFiscal

```
1 package br.com.renanrodrigues.composicao;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class NotaFiscal {
7
8     private String codigo;
9     private List<Item> listaItem = new ArrayList<Item>();
10
11     public NotaFiscal(String codigo, String descricao, int qtde) {
12         this.codigo = codigo;
13
14         this.addItem(descricao, qtde);
15     }
16 }
```



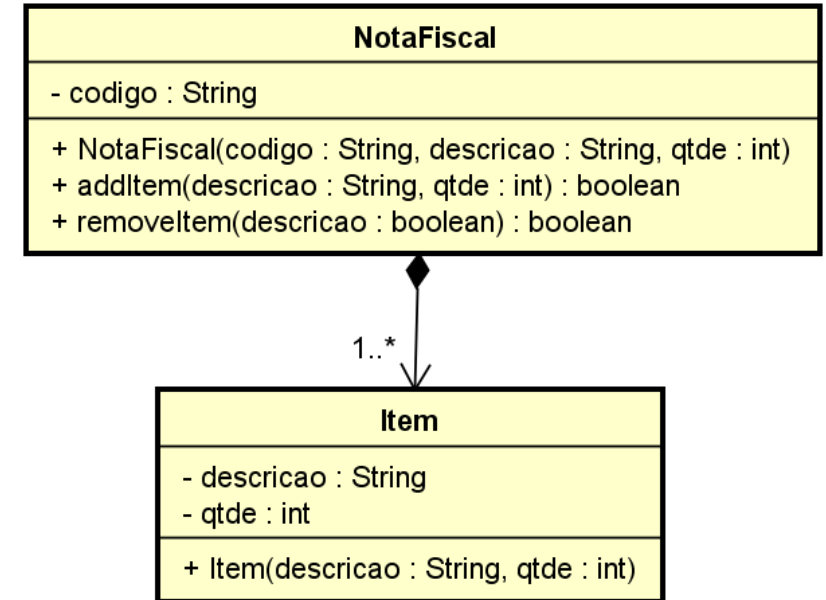
CONTINUA >>

# Composição: Multiplicidade 1..\*

## Implementando a Classe NotaFiscal

```
17 public boolean addItem(String descricao, int qtde) {  
18     boolean sucesso = false;  
19  
20     Item i = new Item(descricao, qtde);  
21  
22     if (!listaItem.contains(i)) {  
23         listaItem.add(i);  
24         sucesso = true;  
25     }  
26     return sucesso;  
27 }
```

```
29 public boolean removeItem(String descricao) {  
30     boolean sucesso = false;  
31  
32     Item i = new Item(descricao);  
33  
34     if (listaItem.size() > 1 && listaItem.contains(i)) {  
35         listaItem.remove(i);  
36         sucesso = true;  
37     }  
38  
39     return sucesso;  
40 }
```

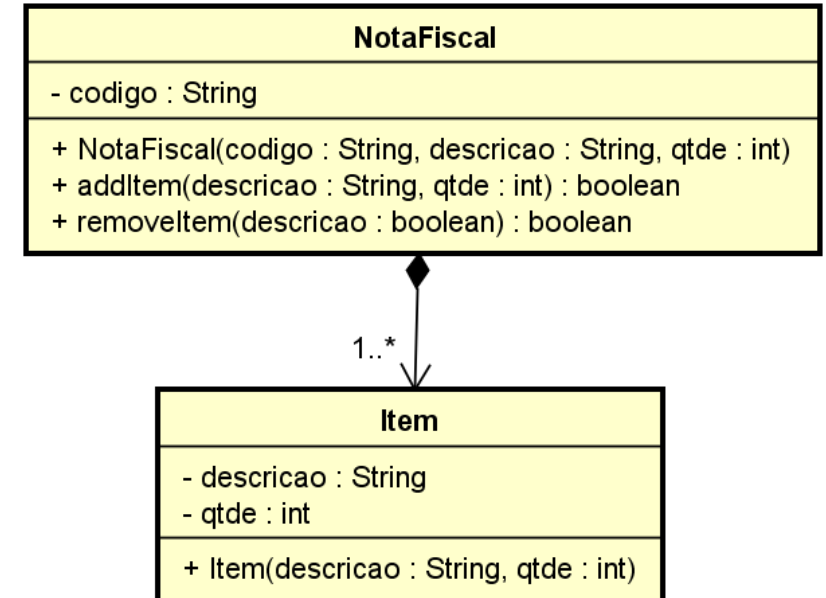


CONTINUA >>

# Composição: Multiplicidade 1..\*

## Implementando a Classe NotaFiscal

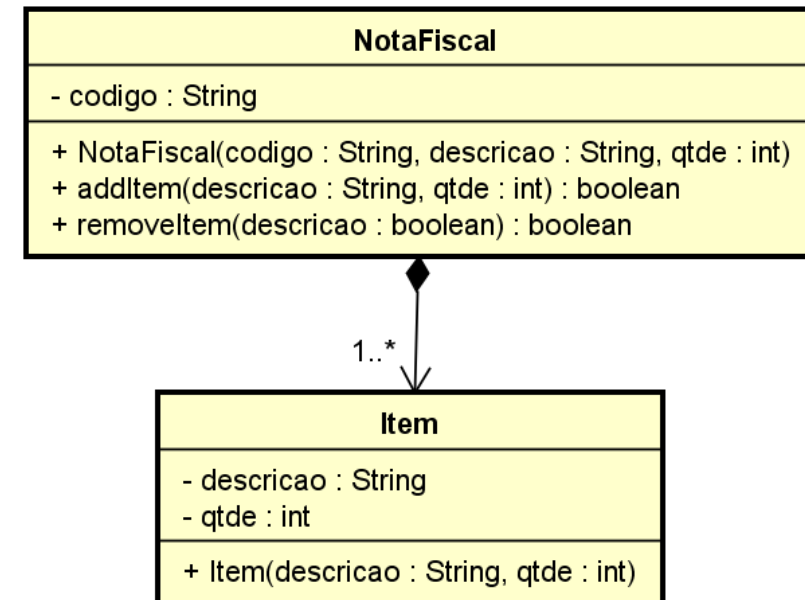
```
42 public String getCodigo() {  
43     return codigo;  
44 }  
45  
46 public List<Item> getListaItem() {  
47     return listaItem;  
48 }  
49  
50 @Override  
51 public String toString() {  
52     return "NotaFiscal [codigo=" + codigo + ", listaItem=" + listaItem  
53         + " ]";  
54 }  
55 }
```



# Composição: Multiplicidade 1..\*

## Programa Principal

```
1 package br.com.renanrodrigues.composicao;
2
3 public class TesteNotaFiscal {
4
5     public static void main(String[] args) {
6
7         NotaFiscal nf = new NotaFiscal("123", "Caneta", 5);
8         System.out.println(nf);
9
10        nf.addItem("Caderno", 3);
11        System.out.println(nf);
12
13        nf.removeItem("Caneta");
14        System.out.println(nf);
15
16        nf.removeItem("Caderno");
17        System.out.println(nf);
18    }
19 }
```



## Saída do Programa

```
NotaFiscal [codigo=123, listaItem=[Item [descricao=Caneta, qtde=5]]]
NotaFiscal [codigo=123, listaItem=[Item [descricao=Caneta, qtde=5], Item [descricao=Caderno, qtde=3]]]
NotaFiscal [codigo=123, listaItem=[Item [descricao=Caderno, qtde=3]]]
NotaFiscal [codigo=123, listaItem=[Item [descricao=Caderno, qtde=3]]]
NotaFiscal [codigo=123, listaItem=[Item [descricao=Régua, qtde=15]]]
NotaFiscal [codigo=123, listaItem=[Item [descricao=Régua, qtde=15], Item [descricao=Borracha, qtde=8]]]
```