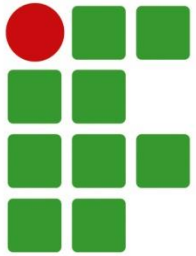


INSTITUTO FEDERAL
Goiás

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

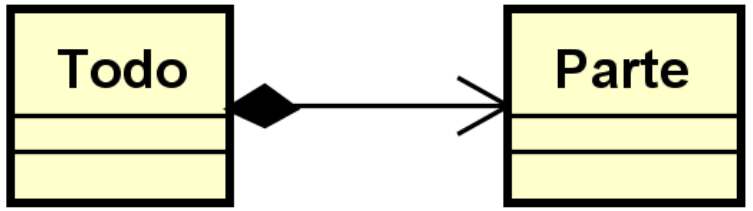
Indicada para representar um relacionamento entre “parte” e “todo”, onde o “todo” é formado por partes.

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 destruirmos o objeto todo as partes permanecem na memória, por terem sido criadas fora do escopo da classe todo.

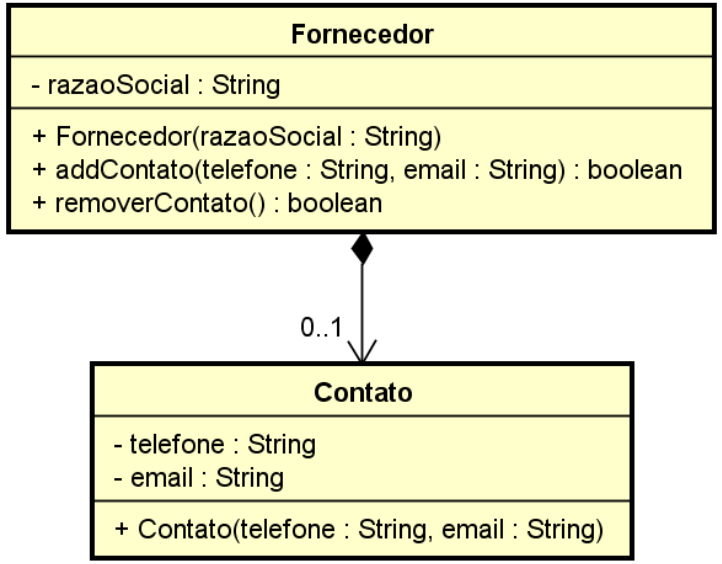
Composição: Multiplicidade 0..1

Exemplo: Um Fornecedor e seu Contato

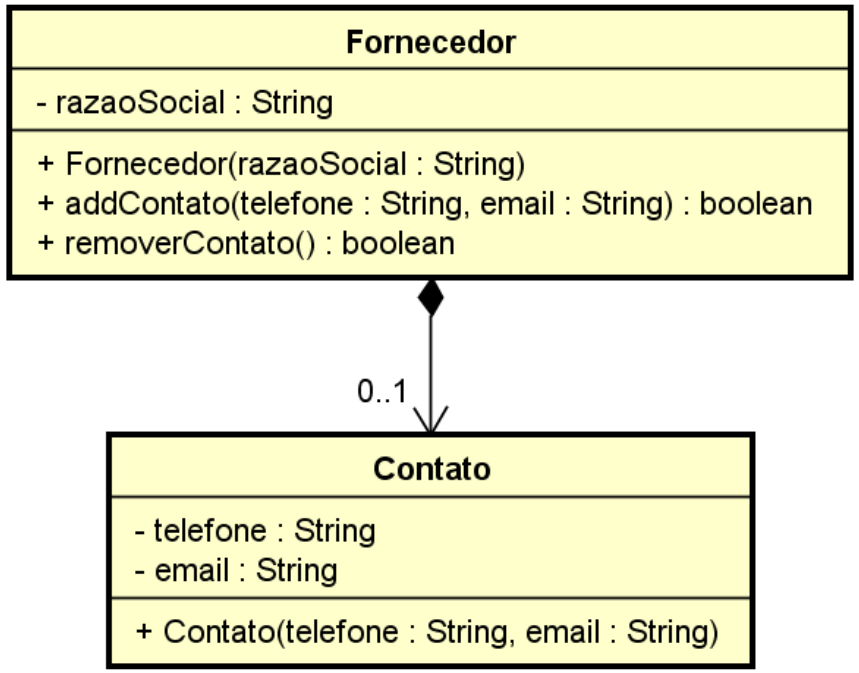
O Fornecedor pode ter um Contato. Se o Fornecedor for removido, o Contato também deve ser removido.

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

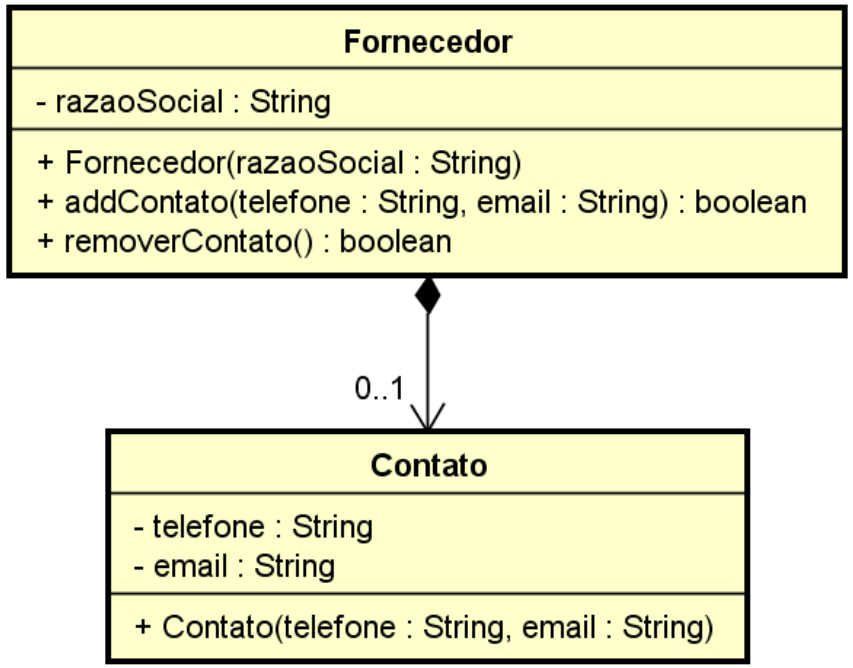
- Um Contato compõe um Fornecedor.
  - O Fornecedor pode ter 0 ou 1 Contato;
  - O vínculo se dará no método addContato();
- Primeiro programe as partes, depois o relacionamento.
  - Crie o Contato somente dentro do método do vínculo.

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

Composição: Multiplicidade 0..1

Implementando a Classe Contato

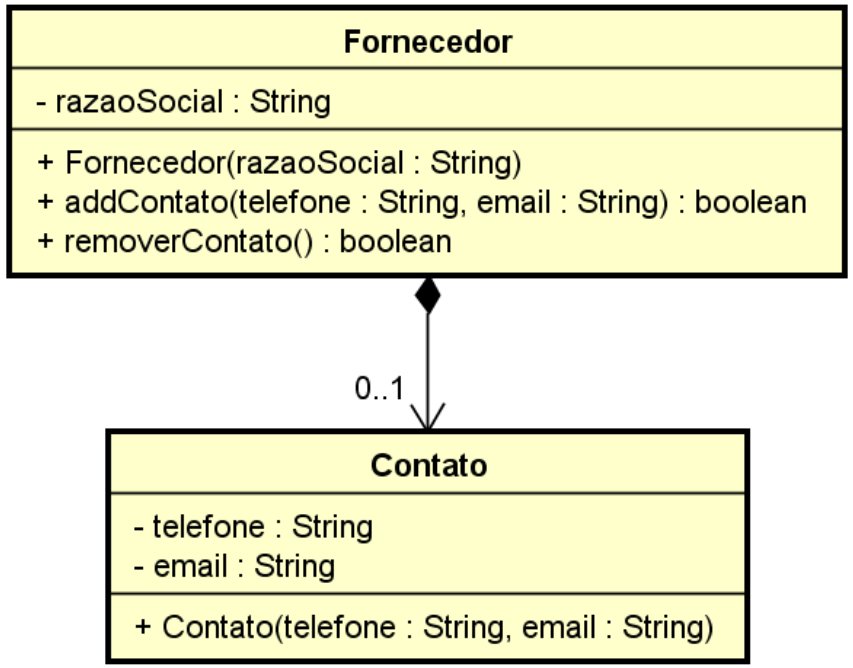
```java
package br.com.renanrodrigues.composicao;

public class Contato {

    private String telefone;
    private String email;

    public Contato(String telefone, String email) {
        this.telefone = telefone;
        this.email = email;
    }

    public String getTelefone() {
        return telefone;
    }

    public String getEmail() {
        return email;
    }

    @Override
    public String toString() {
        return "Contato [telefone=" + telefone + ", email=" + email + "]";
    }
}
```



Composição: Multiplicidade 0..1

Implementando a Classe Fornecedor

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

public class Fornecedor {

    private String razaoSocial;
    private Contato contato;

    public Fornecedor(String razaoSocial) {
        this.razaoSocial = razaoSocial;
    }

    public boolean addContato(String telefone, String email) {
        boolean sucesso = false;

        if (this.contato == null) {
            contato = new Contato(telefone, email);
            return true;
        }

        return sucesso;
    }
```



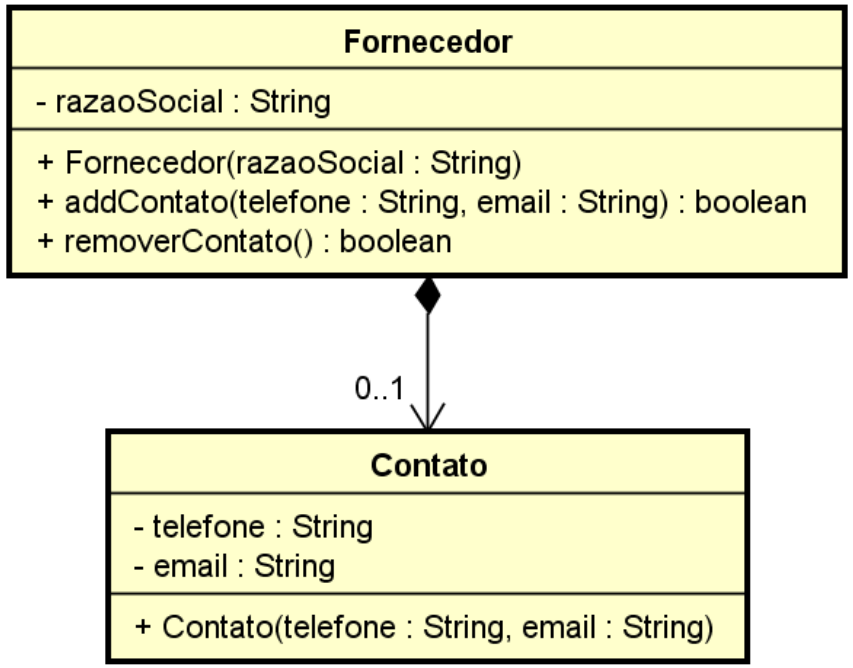
CONTINUA >>

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    }
```

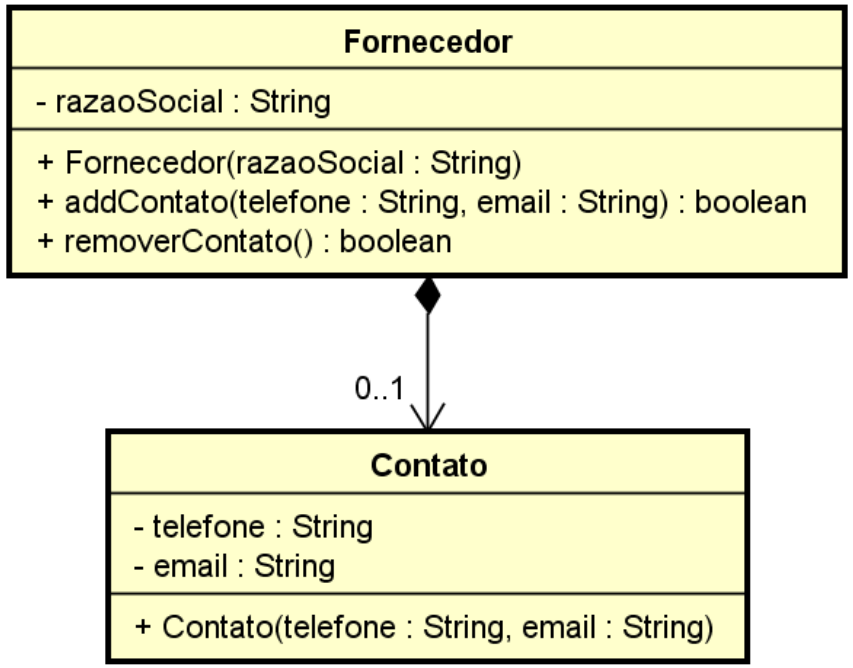
CONTINUA >>



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]]
```

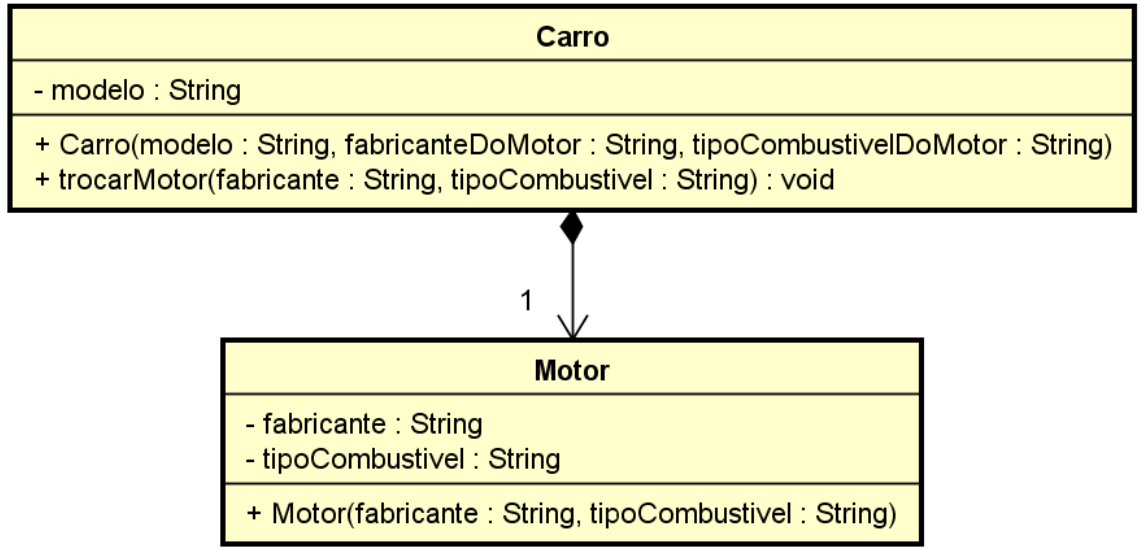
Composição: Multiplicidade 1

Exemplo: Um Carro e seu Motor

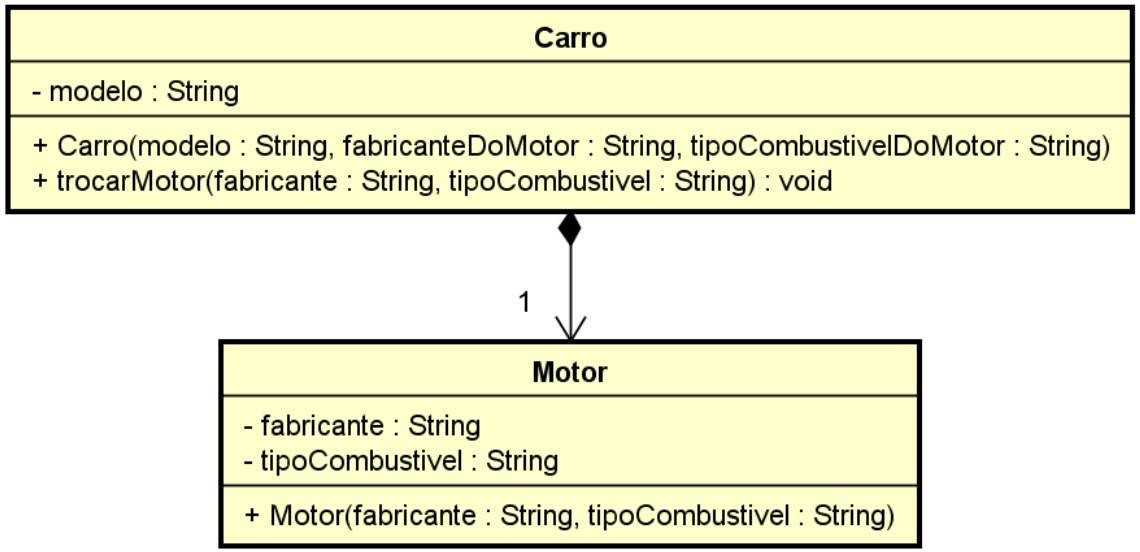
O Carro possui um Motor. Se o Carro for destruído, o Motor também é destruído.

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

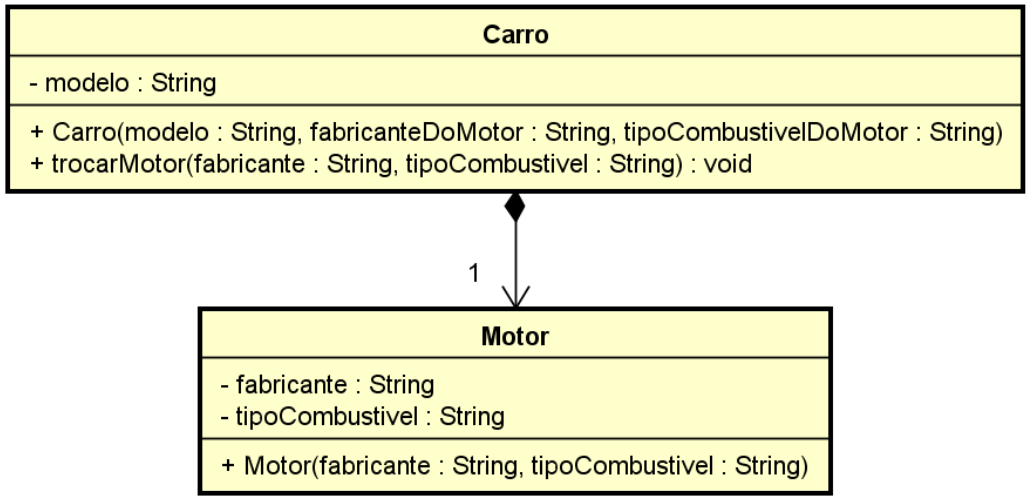
- Um Motor compõe um Carro.
  - O Carro deve ter um Motor;
  - O vínculo se dará construtor.
- Primeiro programe as partes, depois o relacionamento.
  - Crie o Motor somente dentro do construtor do Carro.

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

Composição: Multiplicidade 1

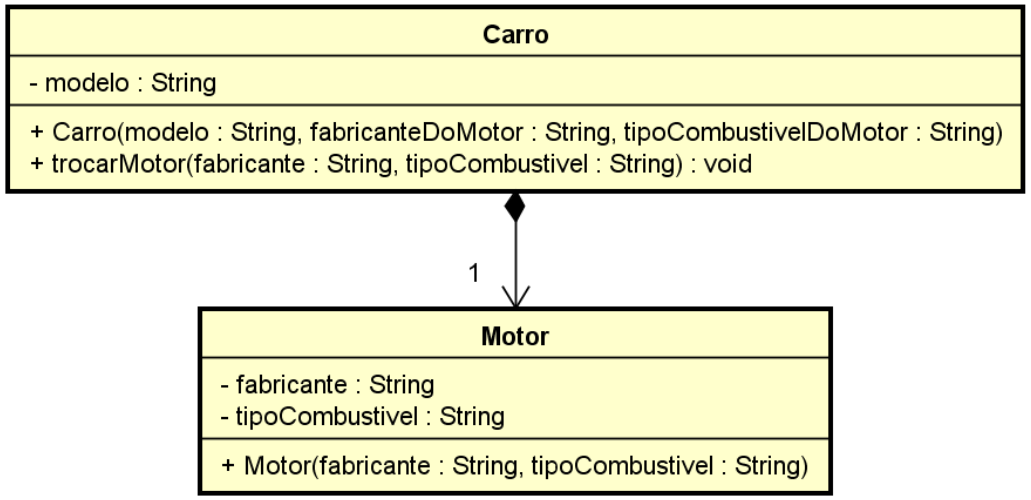
Implementando a Classe Motor

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

public class Motor {

    private String fabricante;
    private String tipoCombustivel;

    public Motor(String fabricante, String tipoCombustivel)
        this.fabricante = fabricante;
        this.tipoCombustivel = tipoCombustivel;
    }

    public String getFabricante() {
        return fabricante;
    }

    public String getTipoCombustivel() {
        return tipoCombustivel;
    }

    @Override
    public String toString() {
        return "Motor [fabricante=" + fabricante + ", tipoCombustivel="
                + tipoCombustivel + "]";
    }
}
```



Composição: Multiplicidade 1

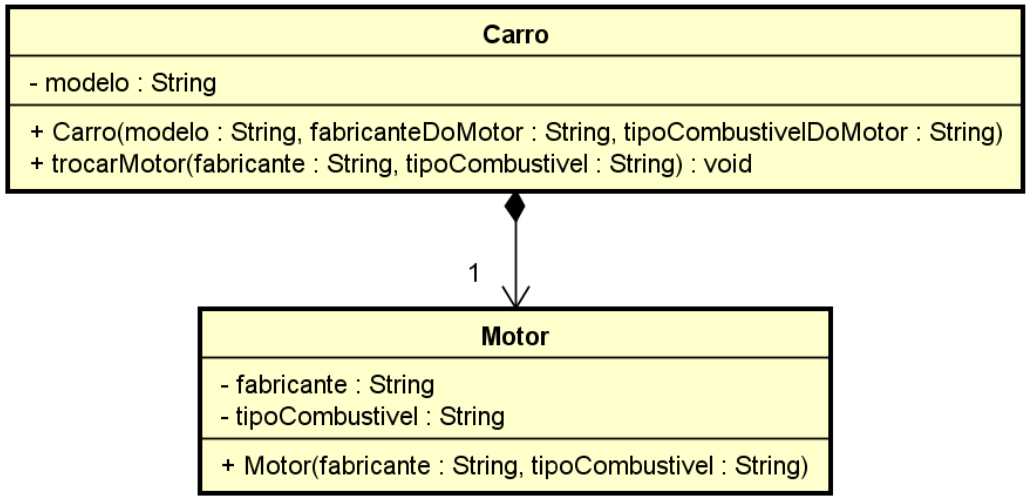
Implementando a Classe Carro




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

public class Carro {

    private String modelo;
    private Motor motor;

    public Carro(String modelo, String fabricanteDoMotor, String tipoDoCombustivelDoMotor) {
        this.modelo = modelo;
        this.motor = new Motor(fabricanteDoMotor, tipoDoCombustivelDoMotor);
    }

    public void trocarMotor(String fabricante, String tipoDoCombustivel) {
        this.motor = new Motor(fabricante, tipoDoCombustivel);
    }

```

CONTINUA >>

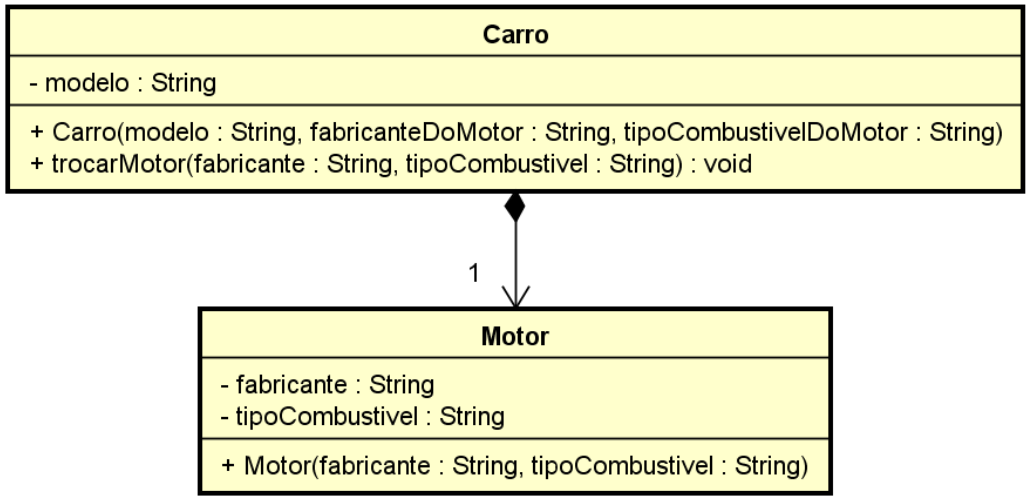
Composição: Multiplicidade 1
Implementando a Classe Carro




```
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




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

public class TesteCarro {

    public static void main(String[] args) {

        Carro c = new Carro("Civic", "Honda", "A/G");
        System.out.println(c.toString());

        c.trocarMotor("Ferrari", "G");
        System.out.println(c.toString());

    }
}
```

Saída do Programa

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

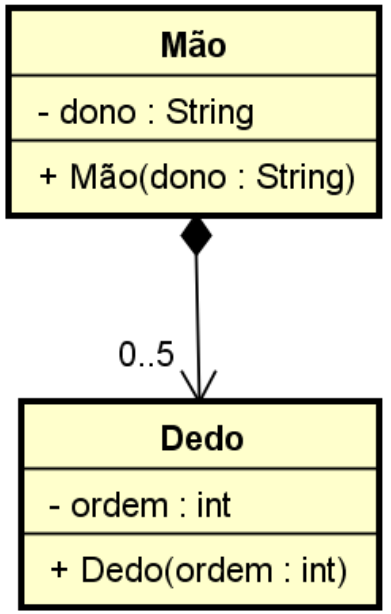
Composição: Multiplicidade 0..N

Exemplo: Uma Mão e seus Dedos

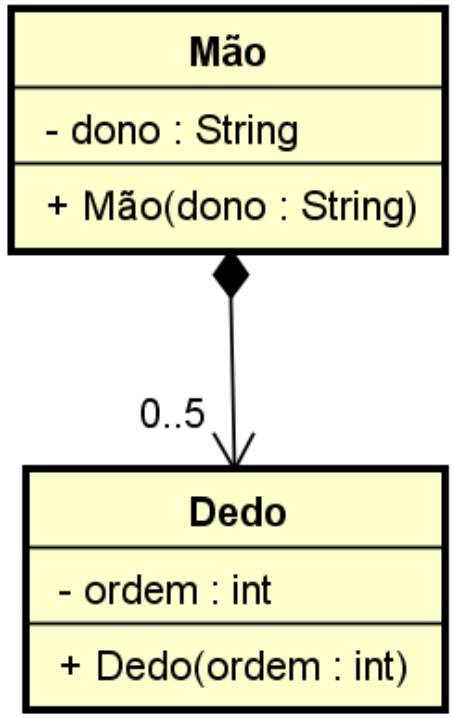
Uma Mão possui no máximo 5 Dedos. Se a Mão for removida, o Dedo também é removido.

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

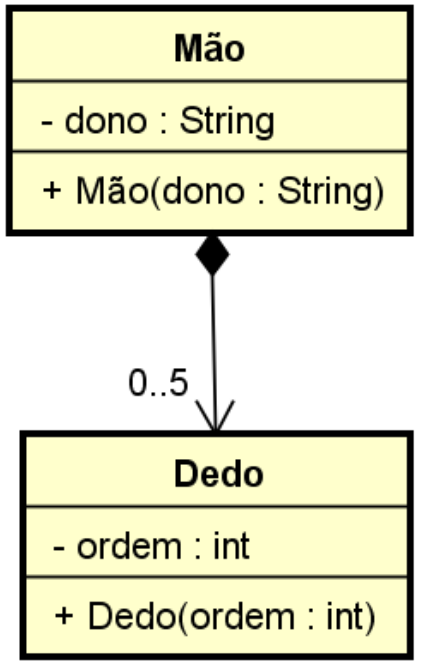
- Um Dedo compõe uma Mão.
  - A Mão pode ter 0 ou 5 Dedos;
  - O vínculo se dará no método addDedo().
- Primeiro programe as partes, depois o relacionamento.
  - Crie o Dedo somente dentro do método de vínculo.

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

Composição: Multiplicidade 0..N

Implementando a Classe Dedo

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

public class Dedo {

    private int ordem;

    public Dedo(int ordem) {
        this.ordem = ordem;
    }

    public int getOrdem() {
        return ordem;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Dedo other = (Dedo) obj;
        if (ordem != other.ordem)
            return false;
        return true;
    }

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

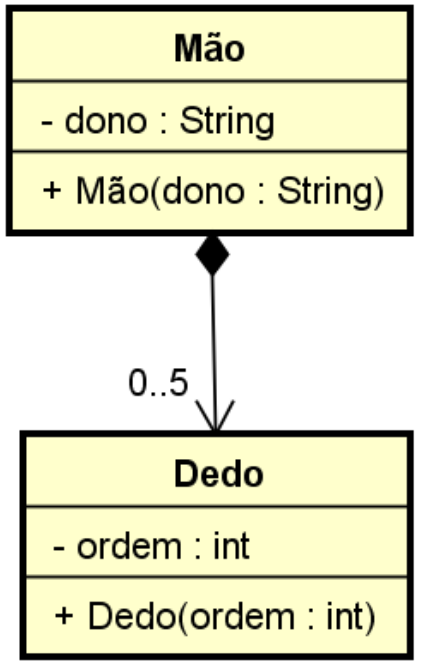


Composição: Multiplicidade 0..N

Implementando a Classe Mão

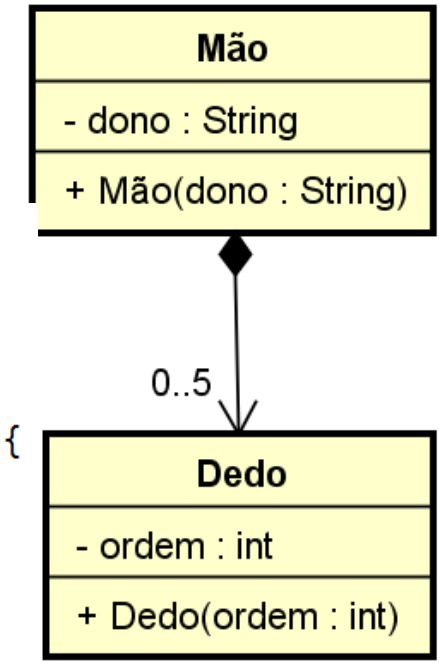
```
1  package br.com.renanrodrigues.composicao;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  public class Mao {
7
8      private String dono;
9      private List<Dedo> listaDedo = new ArrayList<Dedo>();
10
11     public Mao(String dono) {
12         this.dono = dono;
13     }
```

CONTINUA >>



Composição: Multiplicidade 0..N

Implementando a Classe Mão




```java
public boolean addDedo(int ordem) {
    boolean sucesso = false;

    Dedo dedo = new Dedo(ordem);

    if ( (ordem >= 1 && ordem <=5) && !listaDedo.contains(dedo) && listaDedo.size() < 4 ) {
        listaDedo.add(dedo);
        sucesso = true;
    }
    return sucesso;
}
```

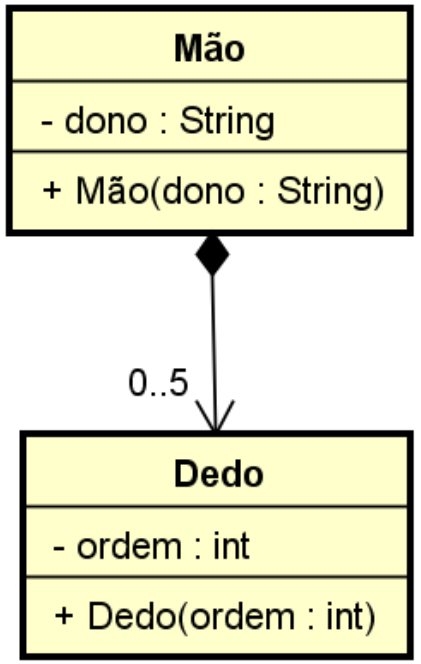
```java
public boolean removerDedo(int ordem) {
    boolean sucesso = false;

    Dedo dedo = new Dedo(ordem);

    if (listaDedo.size() > 0 && listaDedo.contains(dedo)) {
        listaDedo.remove(dedo);
        sucesso = true;
    }

    return sucesso;
}
```

CONTINUA >>

Composição: Multiplicidade 0..N

Implementando a Classe Mão

```
40      public String getDono() {
41          return dono;
42      }
43
44      public List<Dedo> getListaDedo() {
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

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

public class TesteMao {

    public static void main(String[] args) {

        Mao m = new Mao("Renan");
        System.out.println(m);

        m.addDedo(1);
        m.addDedo(2);
        m.addDedo(2);
        m.addDedo(6);
        m.addDedo(3);
        System.out.println(m);

        m.removerDedo(2);
        System.out.println(m);

        m.addDedo(2);
        System.out.println(m);
    }
}
```

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]]]
```

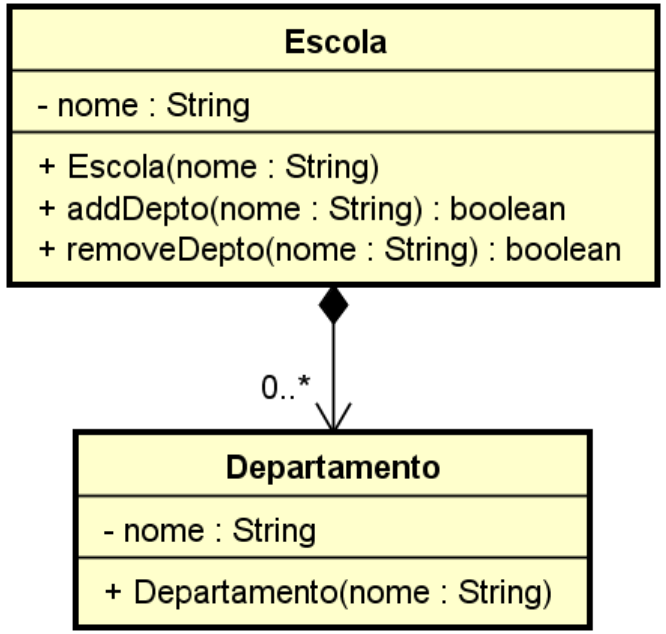
Composição: Multiplicidade 0..*

Exemplo: Uma Escola e seus Departamentos

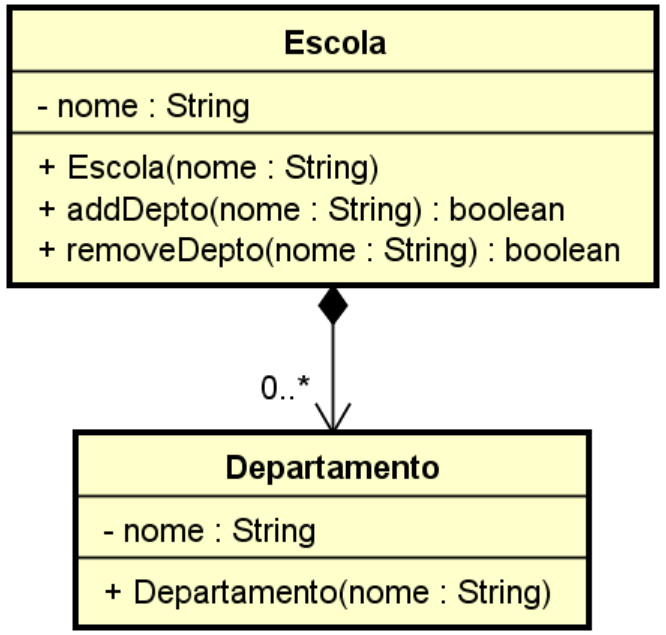
Uma Escola possui vários Departamentos. Se a Escola for destruída, o Departamento também é destruído.

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..*



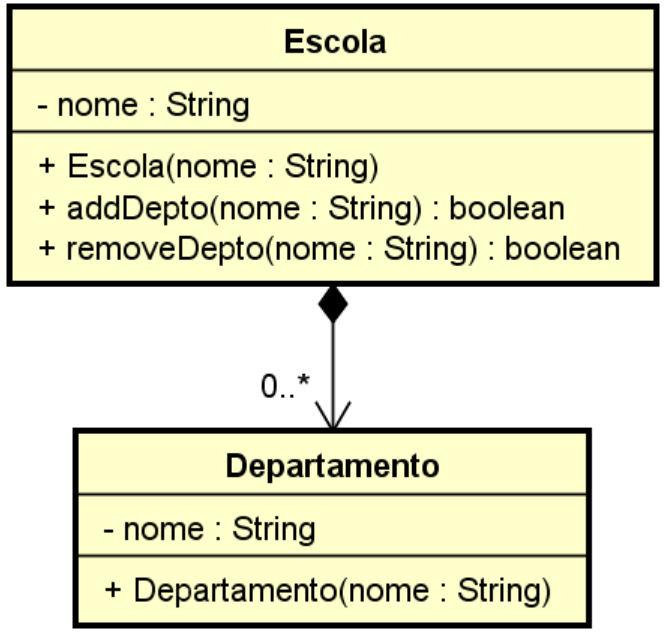
Implementação

- 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..*

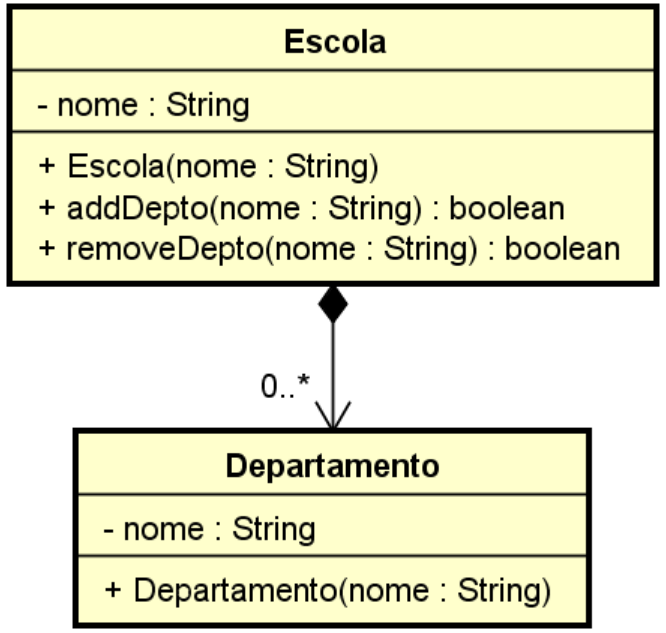
Implementando a Classe Departamento




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

public class Departamento {

    private String nome;

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

    public String getNome() {
        return nome;
    }

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

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

Composição: Multiplicidade 0..*

Implementando a Classe Escola

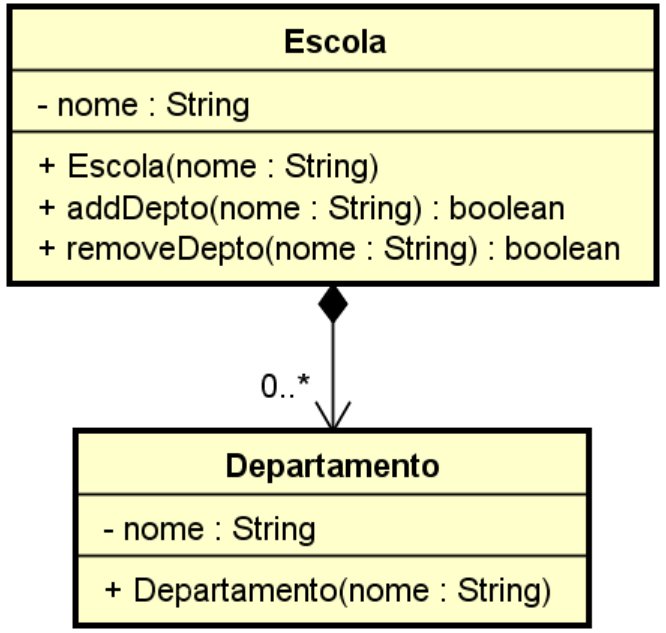



```java
package br.com.renanrodrigues.composicao;

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

public class Escola {

    private String nome;
    private List<Departamento> listaDepto = new ArrayList<Departamento>();

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

CONTINUA >>

Composição: Multiplicidade 0..*

Implementando a Classe Escola




```java
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    }
```

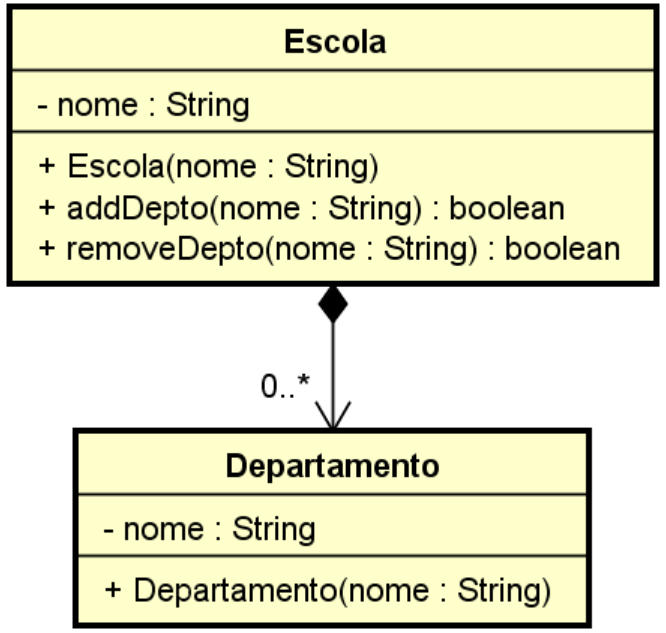
```java
27    public boolean removeDepto(String nome) {
28        boolean sucesso = false;
29
30        Departamento depto = new Departamento(nome);
31
32        if (listaDepto.size() > 0 && listaDepto.contains(depto)) {
33            listaDepto.remove(depto);
34            sucesso = true;
35       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

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

public class TesteEscola {

    public static void main(String[] args) {

        Escola e = new Escola("IFG");
        System.out.println(e);

        e.addDepto("Depto I");
        e.addDepto("Depto II");
        e.addDepto("Depto III");
        e.addDepto("Depto IV");
        System.out.println(e);

        e.removeDepto("Depto II");
        e.removeDepto("Depto III");
        System.out.println(e);

    }
}
```

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 Nata Fiscal for destruída, os Itens também são destruídos.

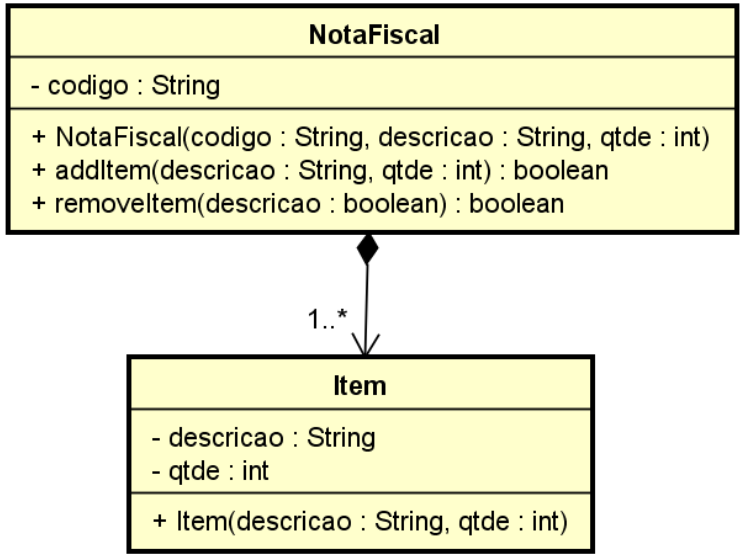
NOME
Endereço completo
CNPJ
Inscrição Estadual
MEI - dispensado
Destinatário
End.
CNPJ/CPF
NOTA FISCAL DE VENDA A CONSUMIDOR – MOD 2 Série D
Data de emissão ...../...../.....
nº NNN

QUANT	Discriminação mercadorias	Preço unit	Total

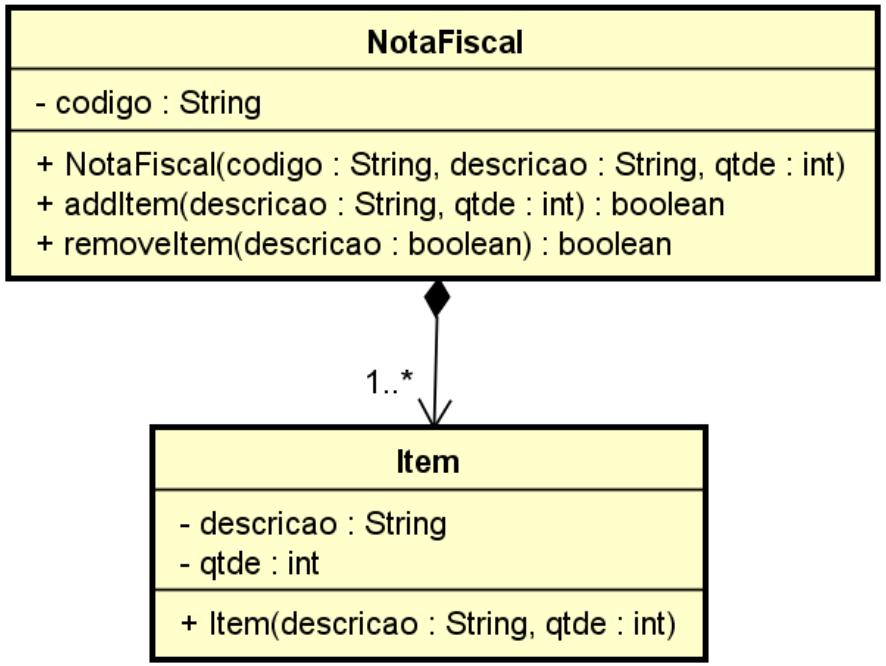
Nome, endereço, inscrição estadual e CNPJ do impressor da nota, data e quantidade de impressão, número de ordem da primeira e última nota impressa e respectiva série e número da Autorização de Impressão de Documentos Fiscais-AIDF

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".



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.

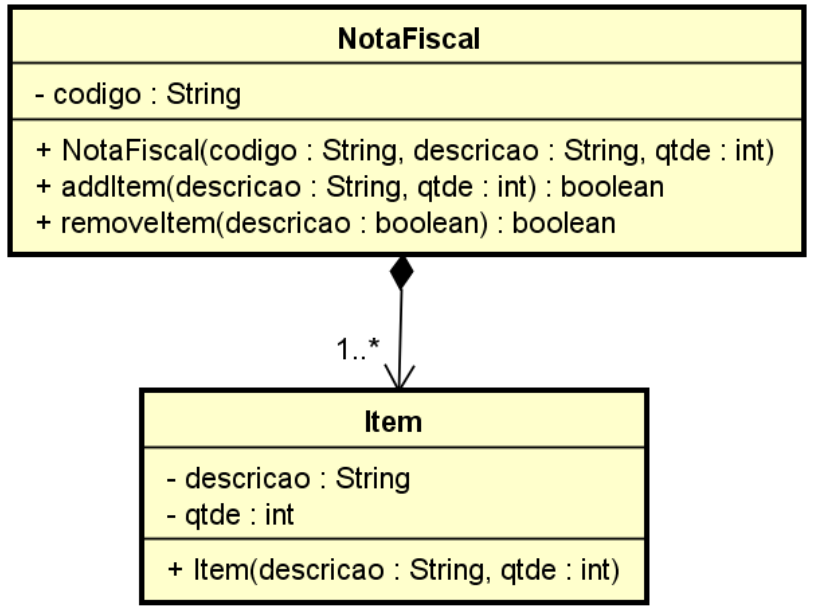
É de responsabilidade do desenvolvedor prover métodos para vínculo, substituição e/ou remoção da parte.

Composição: Multiplicidade 1..*

Implementando a Classe Item

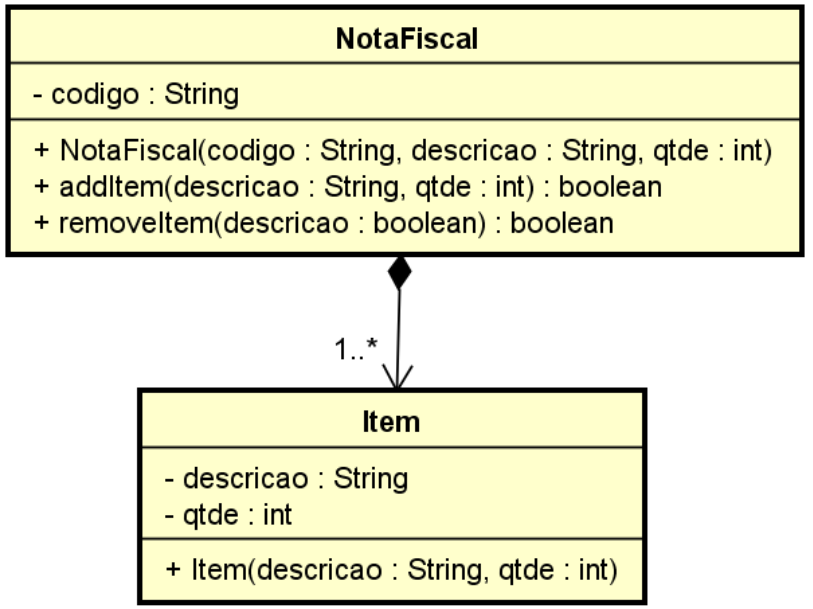
```java
package br.com.renanrodrigues.composicao;

public class Item {

    private String descricao;
    private int qtde;

    public Item(String descricao, int qtde) {
        this.descricao = descricao;
        this.qtde = qtde;
    }

    public Item(String descricao) {
        this.descricao = descricao;
    }

    public String getDescricao() {
        return descricao;
    }

    public int getQtde() {
        return qtde;
    }
}
```

CONTINUA >>



Composição: Multiplicidade 1..*

Implementando a Classe Item



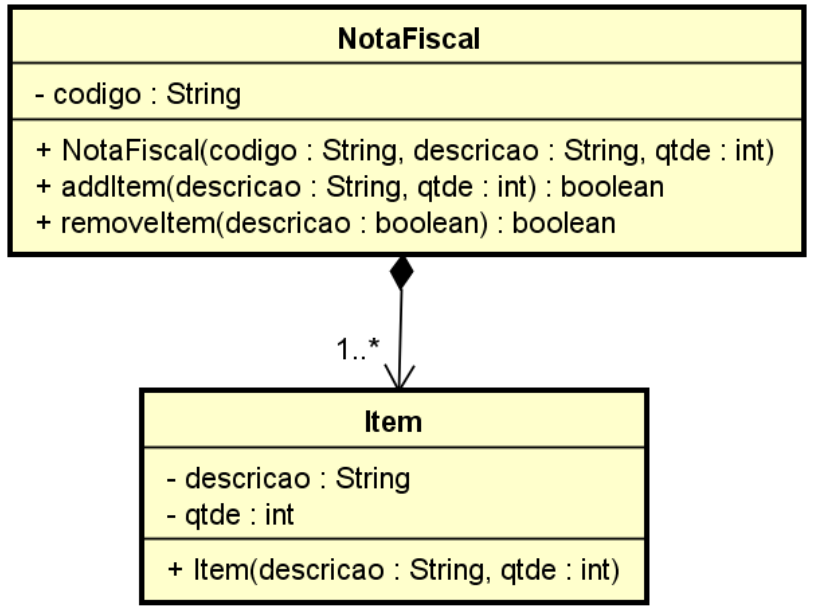

```
25      @Override
26      public boolean equals(Object obj) {
27          if (this == obj)
28              return true;
29          if (obj == null)
30              return false;
31          if (getClass() != obj.getClass())
32              return false;
33          Item other = (Item) obj;
34          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

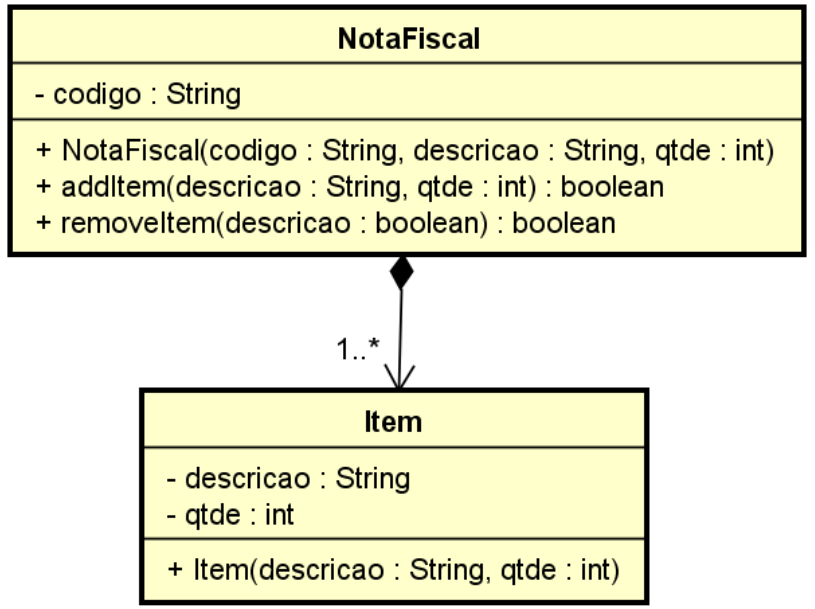



```java
package br.com.renanrodrigues.composicao;

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

public class NotaFiscal {

    private String codigo;
    private List<Item> listaItem = new ArrayList<Item>();

    public NotaFiscal(String codigo, String descricao, int qtde) {
        this.codigo = codigo;

        this.addItem(descricao, qtde);
    }
```

CONTINUA >>

Composição: Multiplicidade 1..*

Implementando a Classe NotaFiscal

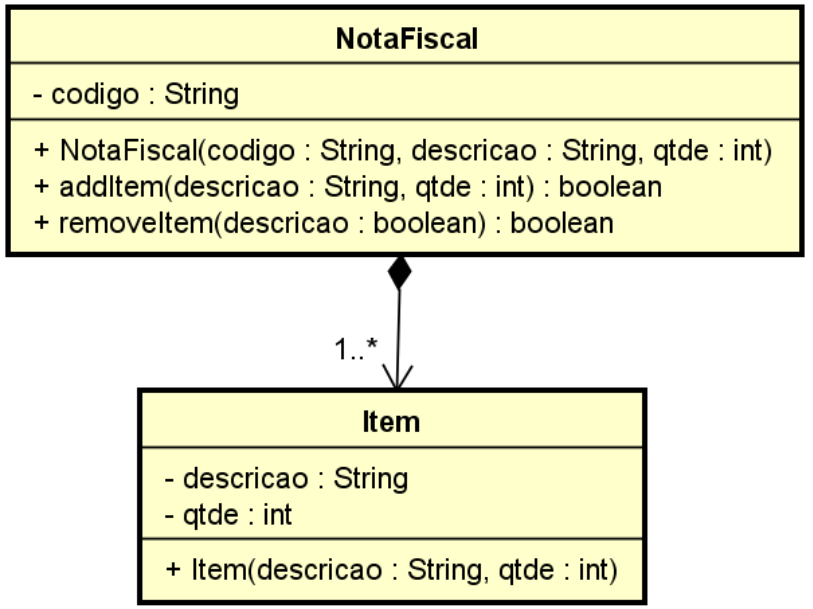



```java
public boolean addItem(String descricao, int qtde) {
    boolean sucesso = false;

    Item i = new Item(descricao, qtde);

    if (!listaItem.contains(i)) {
        listaItem.add(i);
        sucesso = true;
    }
    return sucesso;
}
```

```java
public boolean removeItem(String descricao) {
    boolean sucesso = false;

    Item i = new Item(descricao);

    if (listaItem.size() > 1 && listaItem.contains(i)) {
        listaItem.remove(i);
        sucesso = true;
    }

    return sucesso;
}
```

CONTINUA >>

Composição: Multiplicidade 1..*
Implementando a Classe NotaFiscal

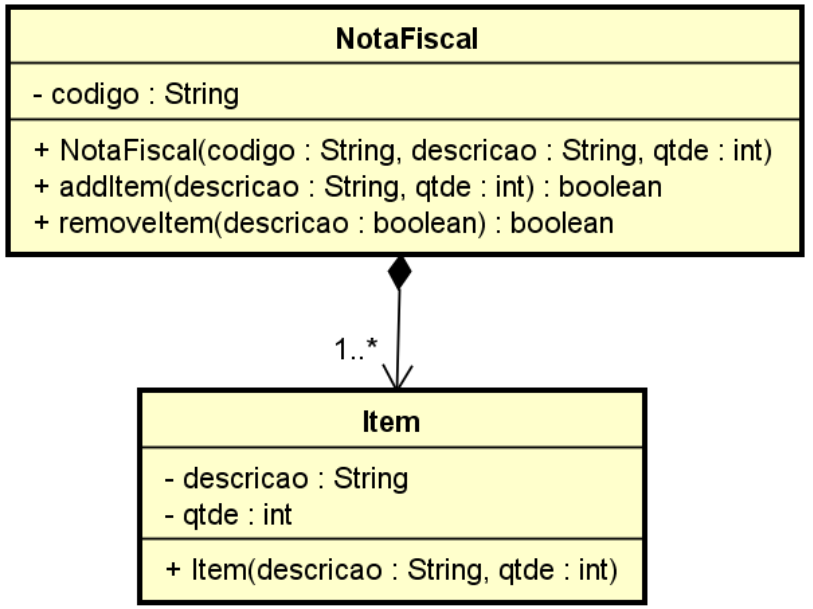



```
42      public String getCodigo() {
43          return codigo;
44      }
45
46      public List<Item> getListaItem() {
47          return listaItem;
48      }
49
50      @Override
51      public String toString() {
52          return "NotaFiscal [codigo=" + codigo + ", listaItem=" + listaItem
53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



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]]]
```