

INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
SÃO PAULO  
Campus São Carlos

*Análise e Desenvolvimento de Sistemas*

*Programação Orientada a Objetos (POOS3)*

# Associação entre Classes



**Pablo Dalbem**

**[dalbem@ifsp.edu.br](mailto:dalbem@ifsp.edu.br)**

# Conteúdo

Associação entre classes

Multiplicidade

Navegabilidade

Classe associativa

Associação reflexiva

Exercício

# Recapitulando...

Em POO, os componentes do problema são chamados de **objetos**.

Os objetos possuem características próprias (**atributos**) e comportamentos específicos (**métodos**).

Objetos que compartilham dos mesmos atributos e métodos são agrupados em classes.

A **classe** atua como um tipo de dados, um molde. Ela define quais atributos e métodos os objetos possuem, bem como a maneira pela qual eles podem ser acessados.

# Introdução

Os atributos das classes não são apenas do tipo de dados primitivo.

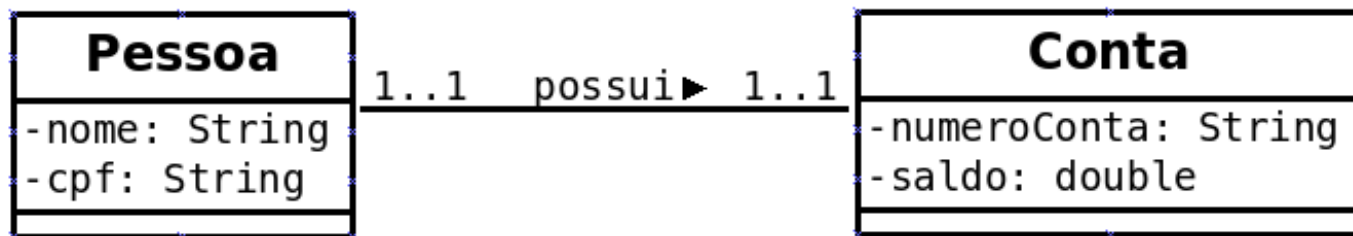
Os atributos podem ser do tipo de outras classes, como aquelas da API Java ou aquelas criadas pelo desenvolvedor.

Quando isso ocorre, dizemos que há uma **associação entre classes**.

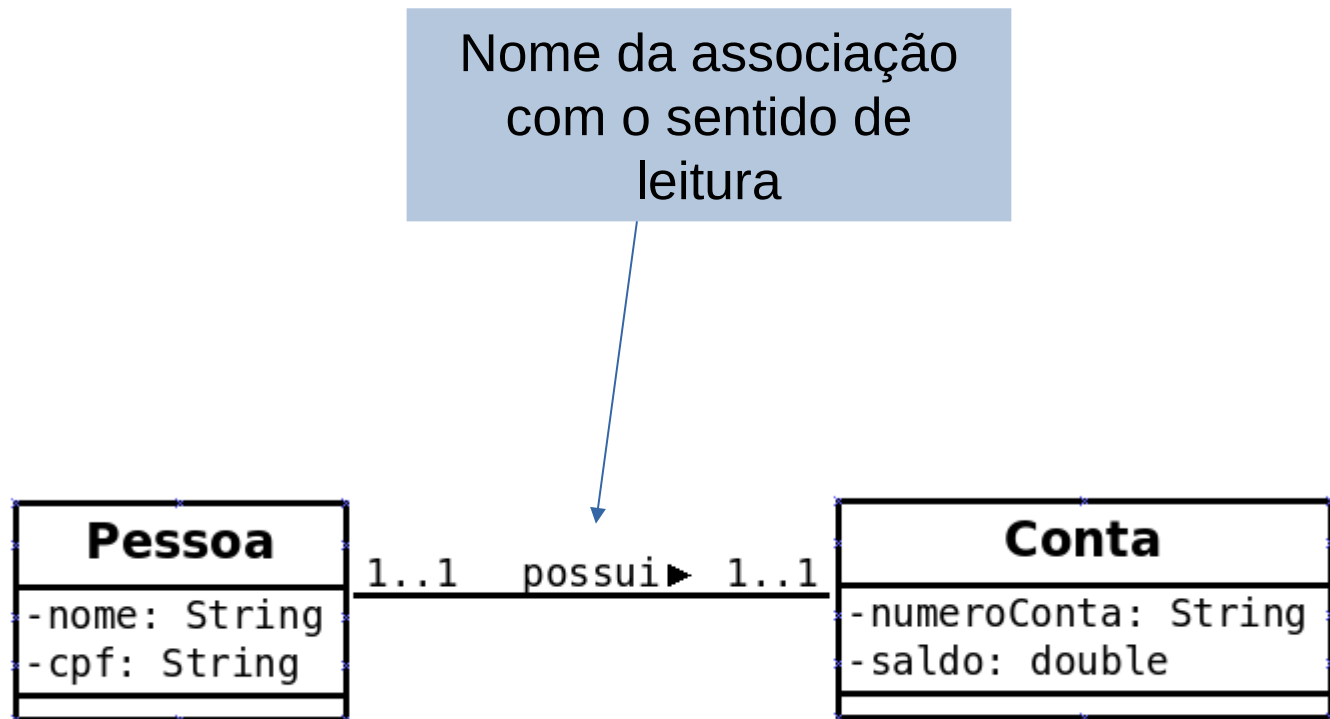
\*Não iremos abordar aqui os conceitos de agregação e composição. Eles serão vistos na disciplina DOO. A diferença é conceitual, semântica. Praticamente, na implementação, não há diferença entre eles.

# Associação

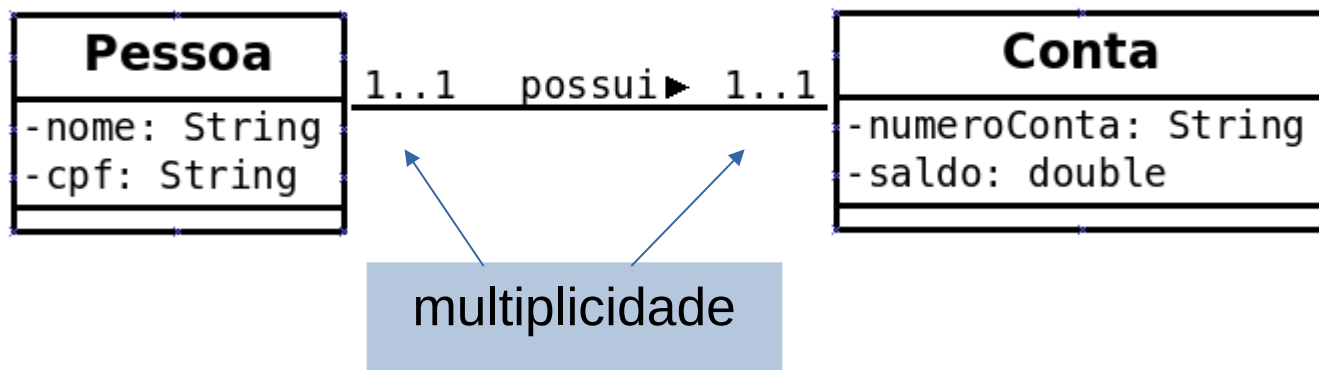
A linha sólida indica que o objeto Pessoa conhece o objeto Conta e vice-versa.



# Associação



# Associação



0..1	Zero ou um.
1..1	Um e somente um.
0..*	Zero ou muitos.
*	Muitos.
1..*	No mínimo um ou muitos.
3..5	Mínimo de três e máximo de cinco.

# Implementação

```
public class Pessoa {
    private String nome;
    private String cpf;
    private Conta conta;

    public Pessoa(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }

    public Conta getConta() {
        return conta;
    }

    public void setConta(Conta conta) {
        this.conta = conta;
    }

    //Outros getters e setters..
}
```

```
public class Conta {
    private String numeroConta;
    private double saldo;
    private Pessoa proprietario;

    public Conta(String numeroConta, double
saldo) {
        this.numeroConta = numeroConta;
        this.saldo = saldo;
    }

    public Pessoa getProprietario() {
        return proprietario;
    }

    public void setProprietario(Pessoa
                                proprietario) {
        this.proprietario = proprietario;
    }

    // Outros getters e setters
}
```

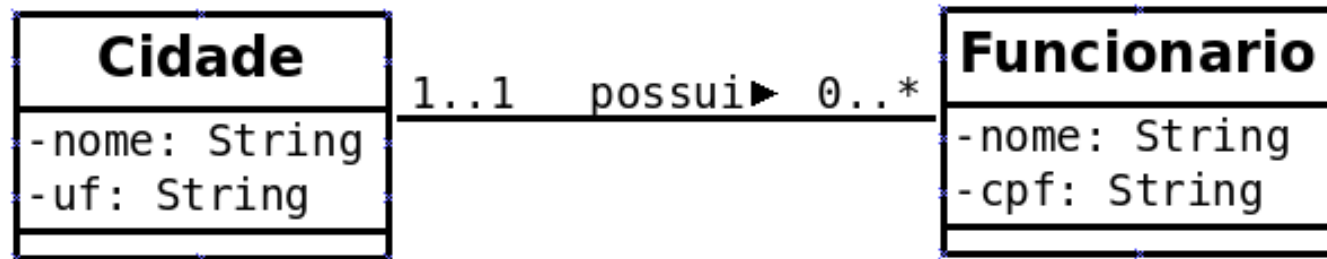


# Implementação

```
public class Main {  
    public static void main (String[] args){  
  
        Pessoa pessoa = new Pessoa("Pablo", "11122233344");  
        Conta conta = new Conta("1010", 1000.00);  
  
        //associação entre pessoa e conta  
        pessoa.setConta(conta);  
        conta.setProprietario(pessoa);  
  
        System.out.println(pessoa.getConta().getSaldo());  
        System.out.println(conta.getProprietario().getNome());  
    }  
}
```

# Associação

Como ficaria a implementação com a seguinte multiplicidade?



# Classe Funcionario

```
public class Funcionario {  
    private String nome;  
    private String cpf;  
    private Cidade cidade;  
  
    public Cidade getCidade() {  
        return cidade;  
    }  
  
    public void setCidade(Cidade cidade) {  
        this.cidade = cidade;  
    }  
  
    //Outros setters e getters.  
    //Construtores  
}
```

# Classe Cidade

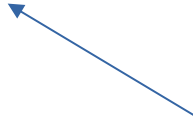
```
public class Cidade {  
    private String nome;  
    private String uf;  
    private Funcionario[] funcionarios;  
  
    public Funcionario[] getFuncionarios() {  
        return funcionarios;  
    }  
  
    public void setFuncionarios (Funcionario[] funcionarios) {  
        this.funcionarios = funcionarios;  
    }  
  
    // Restante da classe.  
}
```

Implementada com alocação estática de memória. Em breve, veremos com alocação dinâmica.

# Classe Main – Exemplo 1

```
public class Main {  
    public static void main (String[] args){  
        Cidade cidade = new Cidade("São Carlos", "SP");  
  
        Funcionario func1 = new Funcionario("Pablo", "12345678910");  
        Funcionario func2 = new Funcionario("Maria", "99988877766");  
  
        func1.setCidade(cidade);  
        func2.setCidade(cidade);  
  
        cidade.setFuncionarios(new Funcionario[]{func1,func2});  
    }  
}
```

O método  
setFuncionarios()  
recebe um array  
como argumento. É  
possível passar o  
array desta forma



# Classe Main – Exemplo 2

```
public class Main {  
    public static void main (String[] args){  
        Cidade cidade = new Cidade("São Carlos", "SP");  
  
        Funcionario func1 = new Funcionario("Pablo", "12345678910");  
        Funcionario func2 = new Funcionario("Maria", "99988877766");  
  
        func1.setCidade(cidade);  
        func2.setCidade(cidade);  
  
        Funcionario[] funcionarios = new Funcionario[10];  
        Funcionarios[0] = func1;  
        Funcionarios[1] = func2;  
        cidade.setFuncionarios(funcionarios);  
  
    }  
}
```

Outra forma de  
passar o array

# Classe Main – Exemplo 3

```
public class Main {  
    public static void main (String[] args){  
        Cidade cidade = new Cidade("São Carlos", "SP");  
  
        Funcionario func1 = new Funcionario("Pablo", "12345678910");  
        Funcionario func2 = new Funcionario("Maria", "99988877766");  
  
        func1.setCidade(cidade);  
        func2.setCidade(cidade);  
  
        cidade.setFuncionarios(new Funcionario[]{func1,func2});  
  
        Funcionario[] listaFuncionarios = cidade.getFuncionarios();  
  
        for (int i=0;i<listaFuncionarios.length;i++)  
            if (listaFuncionarios[i]!=null) {  
                System.out.println(listaFuncionarios[i].getNome());  
                System.out.println(listaFuncionarios[i].getCpf());  
            }  
    }  
}
```

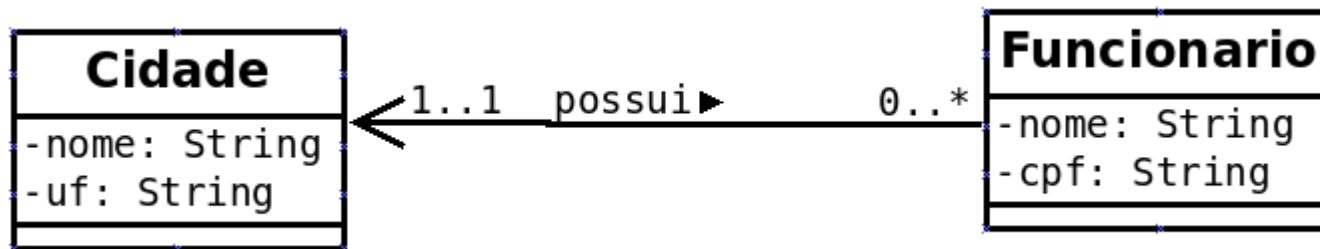
Listando os funcionários de uma cidade específica. Também é possível utilizar o **for** aprimorado!!!

# Navegabilidade

Restringe o sentido da associação.

Indicada por uma seta no fim da associação.

No exemplo, o objeto Funcionario conhece o objeto Cidade. Porém, o objeto Cidade não precisa conhecer seus objetos da classe Funcionario.





# Implementação

```
public class Cidade {  
    private String nome;  
    private String uf;
```

```
    // setters e getters.
```

```
    // Construtores
```

```
}
```

```
public class Funcionario {  
    private String nome;  
    private String cpf;  
    private Cidade cidade;
```

```
    public Cidade getCidade() {  
        return cidade;  
    }
```

```
    public void setCidade(Cidade cidade) {  
        this.cidade = cidade;  
    }
```

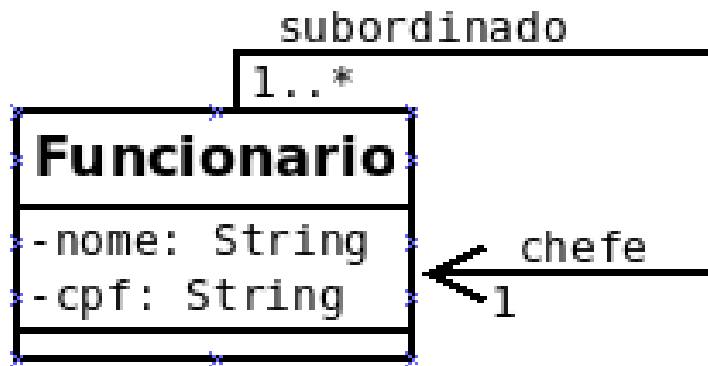
```
    //Outros setters e getters.
```

```
    //Construtores
```

```
}
```

# Associação Reflexiva

Quando um objeto se relaciona com outro objeto da mesma classe.

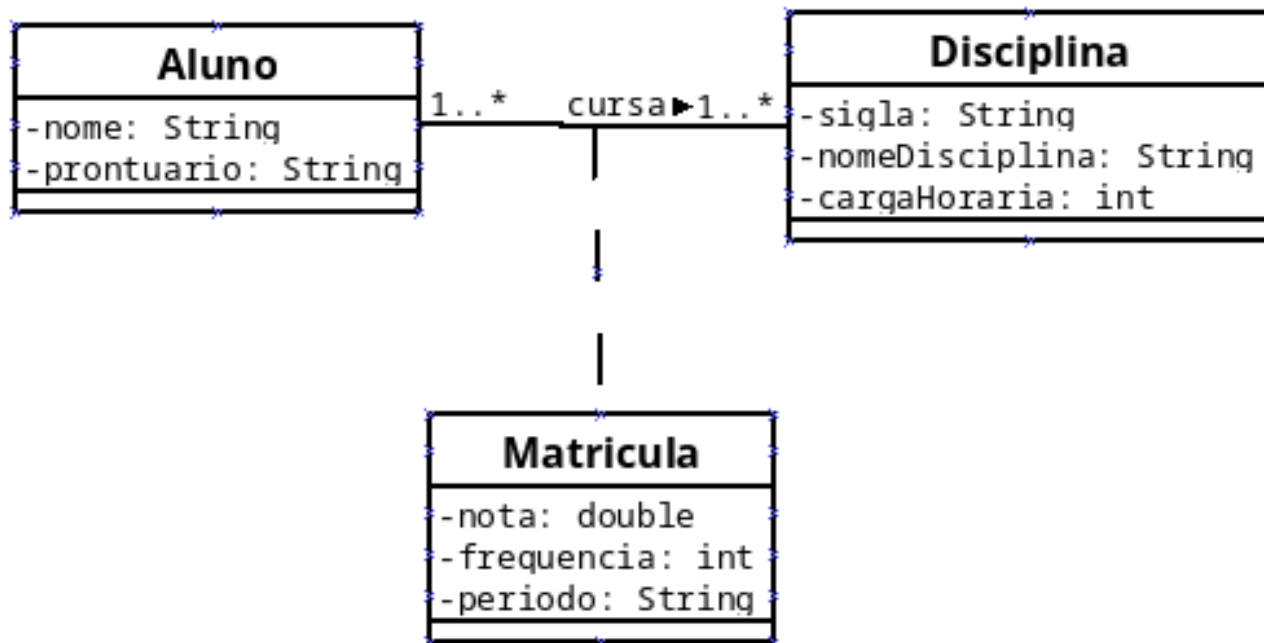


```
public class Funcionario {
    private String nome;
    private String cpf;
    private Funcionario chefia;

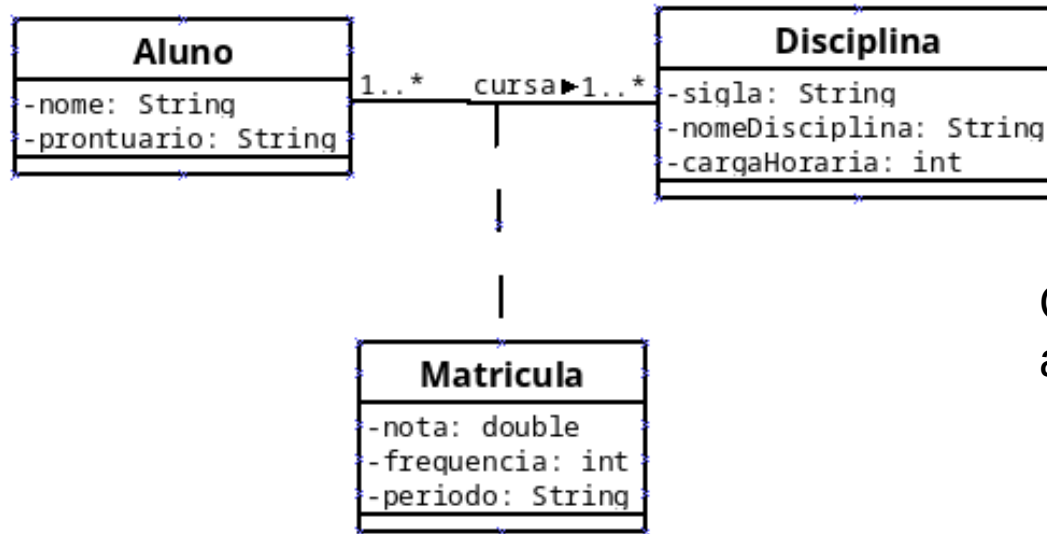
    //restante da classe
}
```

# Classe associativa

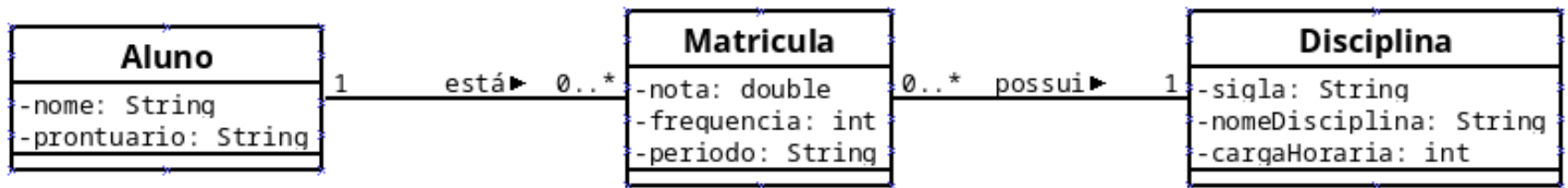
Ocorre quando a multiplicidade da associação é “Muitos para Muitos” e há atributos desta associação.



A classe associativa pode, muitas vezes, ser representada como associação binária



Classe  
associativa



Associação binária

# Implementação

```
public class Aluno {  
    private String nome;  
    private String prontuario;  
    private Matricula[] matriculas;  
    ...  
}
```

```
public class Disciplina {  
    private String sigla;  
    private String nomeDisciplina;  
    private int cargaHoraria;  
    private Matricula[] matriculas;  
    ...  
}
```

```
public class Matricula {  
    private Aluno aluno;  
    private Disciplina disciplina;  
    private double nota;  
    private int frequencia;  
    private String periodo;  
    ...  
}
```

**Acesse o exemplo completo em:**

[https://github.com/pdalbem/POO\\_2023\\_1/tree/main/Associa%C3%A7%C3%A3o%20entre%20Classes/ExemploFaculdade](https://github.com/pdalbem/POO_2023_1/tree/main/Associa%C3%A7%C3%A3o%20entre%20Classes/ExemploFaculdade)

Vamos testar com as seguintes informações:

nome	prontuário
João	SC12345
Maria	SC11223
José	SC98765

sigla	nomeDisciplina	CH
POOS3	Programação Orientada a Objetos	80
ESDS3	Estrutura de Dados	80
ESWS3	Engenharia de Software	80

aluno	disciplina	nota	freq.	período
João	Programação Orientada a Objetos	9	70	2022.2
Maria	Programação Orientada a Objetos	10	90	2022.2
Maria	Estrutura de Dados	8	100	2022.2
José	Estrutura de Dados	9	100	2022.2
José	Engenharia de Software	10	90	2022.2

```
public class Main {  
    public static void main(String[] args) {  
        Aluno a1 = new Aluno("João", "SC12345");  
        Aluno a2 = new Aluno("Maria", "SC11223");  
        Aluno a3 = new Aluno("José", "SC98765");  
  
        Disciplina d1 = new Disciplina("POOS3", "Programação Orientada a Objetos", 80);  
        Disciplina d2 = new Disciplina("ESDS3", "Estrutura de Dados", 80);  
        Disciplina d3 = new Disciplina("ESWS3", "Engenharia de Software", 80);  
  
        Matricula m1 = new Matricula(a1, d1, 9, 70, "2022.1");  
        Matricula m2 = new Matricula(a2, d1, 10, 90, "2022.1");  
        Matricula m3 = new Matricula(a2, d2, 8, 100, "2022.1");  
        Matricula m4 = new Matricula(a3, d2, 9, 100, "2022.1");  
        Matricula m5 = new Matricula(a3, d3, 10, 90, "2022.1");  
  
        a1.setMatriculas(new Matricula[]{m1});  
        a2.setMatriculas(new Matricula[]{m2, m3});  
        a3.setMatriculas(new Matricula[]{m4, m5});  
  
        d1.setMatriculas(new Matricula[]{m1, m2});  
        d2.setMatriculas(new Matricula[]{m3, m4});  
        d3.setMatriculas(new Matricula[]{m5});  
    }  
}
```

## Continuação...

```
//Saber as disciplinas em que José está matriculado
for (Matricula m : a3.getMatriculas()) {
    System.out.println(m.getDisciplina().getNomeDisciplina());
}

//Saber quem está cursando POO
for (Matricula m : d1.getMatriculas())
    System.out.println(m.getAluno().getNome());
}
}
```



# Considerações sobre arrays

Array armazena uma coleção de elementos do mesmo tipo.

Possui tamanho fixo, tendo que ser declarado na criação.

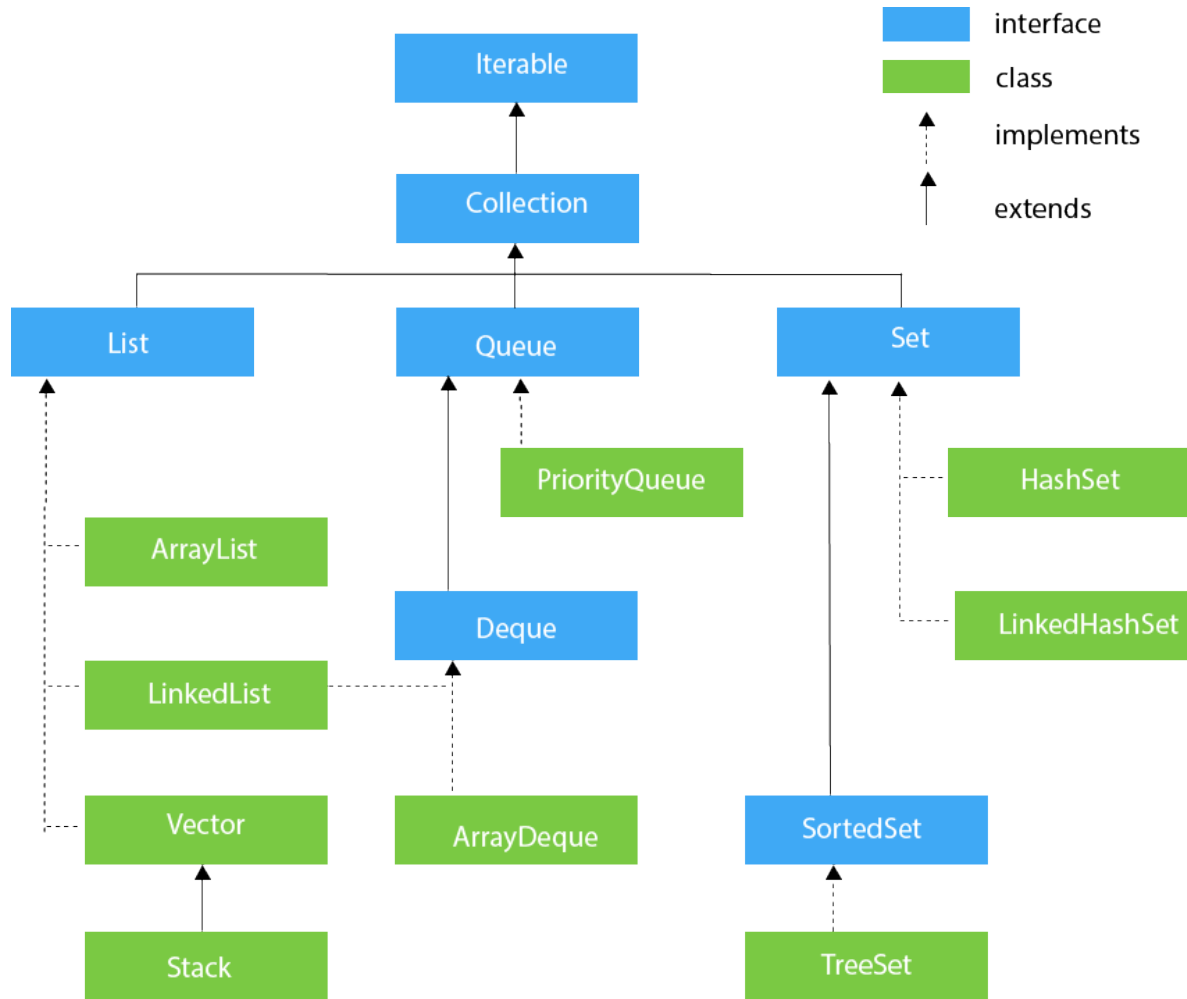
- É preciso saber de antemão o número de elementos
- Problema em superestimar ou subestimar o tamanho

Remoção de elementos pode deixar espaço vazio no array.

- Desperdício e rápido esgotamento de espaço
- Para novas inserções: deslocamento ou procura por posições disponíveis

# API Collection

*Framework* para manipulação de coleções de dados



# List

## List

Armazena um conjunto ordenado de elementos, podendo ter valores repetidos. Não é necessário definir sua capacidade de antemão. Seu tamanho é variável, aumentando conforme demanda.

Os principais métodos definidos por List são:

`add(E elemento)`: adiciona o elemento no final da lista;

`add(int index, E elemento)`: adiciona o elemento na posição `index`;

`get(int index)`: retorna o objeto na posição especificada no índice;

`indexOf(E elemento)`: retorna o índice no qual o elemento está alocado;

`remove(E elemento)`: remove o elemento da lista;

`remove(int index)`: remove o elemento na posição especificada pelo índice;

`contains(E elemento)`: retorna se o elemento está ou não na lista;

`clear()`: remove todos os elementos da lista;

`size()`: retorna o número de elementos da lista;

# ArrayList

## ArrayList

Implementação de List. Amplamente utilizada

Exemplo de lista **genérica** com ArrayList

```
List lista = new ArrayList<>();  
//OU: ArrayList lista = new ArrayList<>();  
lista.add("Java");  
lista.add("C");  
lista.add(0,"Python");  
lista.add(10);  
lista.add(5.5);  
System.out.println(lista.size()); // mostra 5  
lista.remove("Python");  
System.out.println(lista.size()); // mostra 4  
for (Object o: lista)  
    System.out.println(o);
```

# ArrayList

Exemplo de lista **de Strings** com ArrayList

```
List<String> lista = new ArrayList<>>();  
//OU: ArrayList<String> lista = new ArrayList<>();
```

```
lista.add("Java");  
lista.add("C");  
lista.add(0, "Python");  
// lista.add(10); Erro, não é String
```

```
for (String s: lista)  
    System.out.println(s);
```

# ArrayList

## Exemplo de lista **de Livros** com ArrayList

```
Livro livro1 = new Livro();
```

```
Livro livro2 = new Livro();
```

```
...
```

```
List<Livro> lista = new ArrayList<>();
```

```
lista.add(livro1);
```

```
lista.add(livro2);
```

*// É possível declarar a lista, passando objetos livro1 e livro2*

```
List<Livro> lista2 = new ArrayList<>(List.of(livro1,livro2));
```

*// Percorrendo a lista com o for aprimorado*

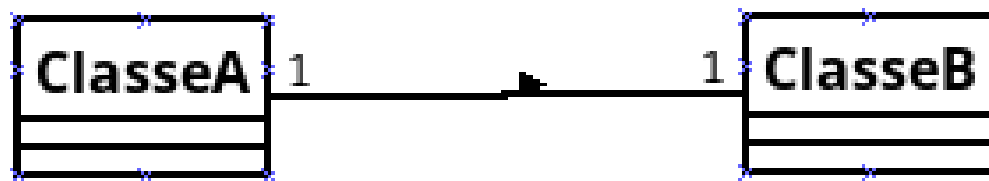
```
for (Livro livro : lista)
```

```
    System.out.println(livro.getTitulo());
```

# Resumo

Navegabilidade bidirecional (ambas as classes se enxergam)

Multiplicidade 1 para 1



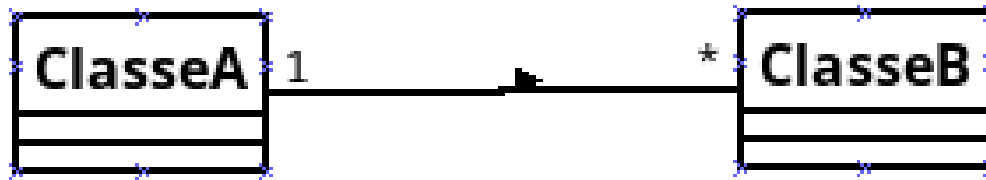
```
public class ClasseA {  
    private ClasseB classeB;  
    ...  
}
```

```
public class ClasseB {  
    private ClasseA classeA;  
    ...  
}
```

# Resumo

Navegabilidade bidirecional (ambas as classes se enxergam)

Multiplicidade 1 para muitos



```
public class ClasseA {
    private List<ClasseB> classeB = new ArrayList<>();
    ...
}
```

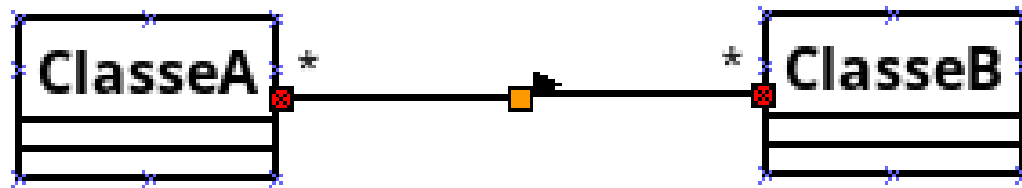
```
public class ClasseB {
    private ClasseA classeA;
    ...
}
```



# Resumo

Navegabilidade bidirecional (ambas as classes se enxergam)

Multiplicidade muitos para muitos



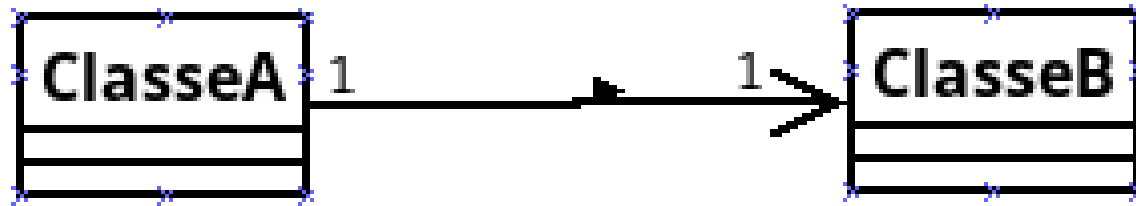
```
public class ClasseA {  
    private List<ClasseB> classeB = new ArrayList<>();  
    ...  
}
```

```
public class ClasseB {  
    private List<ClasseA> classeA = new ArrayList<>();  
    ...  
}
```

# Resumo

Navegabilidade unilateral (ClasseA enxerga ClasseB. Porém, ClasseB NÃO enxerga ClasseA)

Multiplicidade um para um



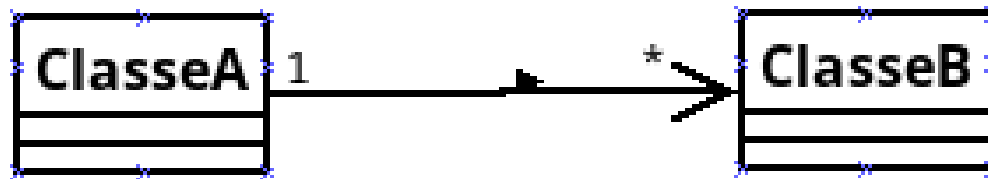
```
public class ClasseA {  
    private ClasseB classeB;  
    ...  
}
```

```
public class ClasseB {  
    ...  
}
```

# Resumo

Navegabilidade unilateral (ClasseA enxerga ClasseB. Porém, ClasseB NÃO enxerga ClasseA)

Multiplicidade um para muitos



```
public class ClasseA {  
    private List<ClasseB> classeB = new ArrayList<>();  
    ...  
}
```

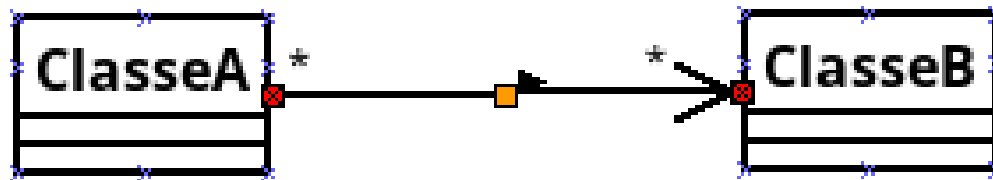
```
public class ClasseB {  
    ...  
}
```

# Resumo

Navegabilidade unilateral (ClasseA enxerga ClasseB. Porém, ClasseB NÃO enxerga ClasseA)

Multiplicidade muitos para muitos

REPARE que a implementação é igual ao do slide anterior. Afinal, a navegabilidade é unilateral de A para B.



```
public class ClasseA {
    private List<ClasseB> classeB = new ArrayList<>();
    ...
}
```

```
public class ClasseB {
    ...
}
```

# Exercício I

Implemente em Java a seguinte especificação para um player de músicas:

É preciso armazenar as seguintes informações sobre as músicas: título, artista, duração e estilo musical.

O player permite a criação de playlists, as quais devem possuir um nome. Uma playlist pode conter várias músicas e uma mesma música pode estar em diversas playlists.

O usuário pode querer saber a duração de uma playlist (soma dos tempos das músicas que a compõem).

O programa deve permitir mostrar os dados das músicas. Não é necessário saber em quais playlists elas estão. Similarmente, o programa deve mostrar os dados da playlist, com seu nome e nome das músicas participantes.

# Exercício I

## DICA

Para armazenar o tempo de duração das músicas, utilize a classe **LocalTime** e seus métodos **plusHours**, **plusMinutes** e **plusSeconds**

<https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html>

## Exemplo:

//Objeto tempo representando 2 minutos e 30 segundos.

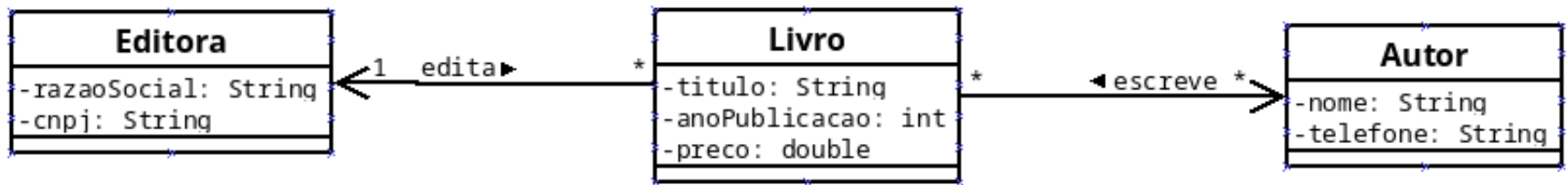
```
LocalTime tempo = LocalTime.of(0,2,30);
```

//Adicionando 3 minutos. Agora, tempo vale 5 minutos e 30 segundos

```
tempo.plusMinutes(3);
```

# Exercício II

Implemente a associação entre classes usando ArrayList.  
Atenção para a multiplicidade e navegabilidade.

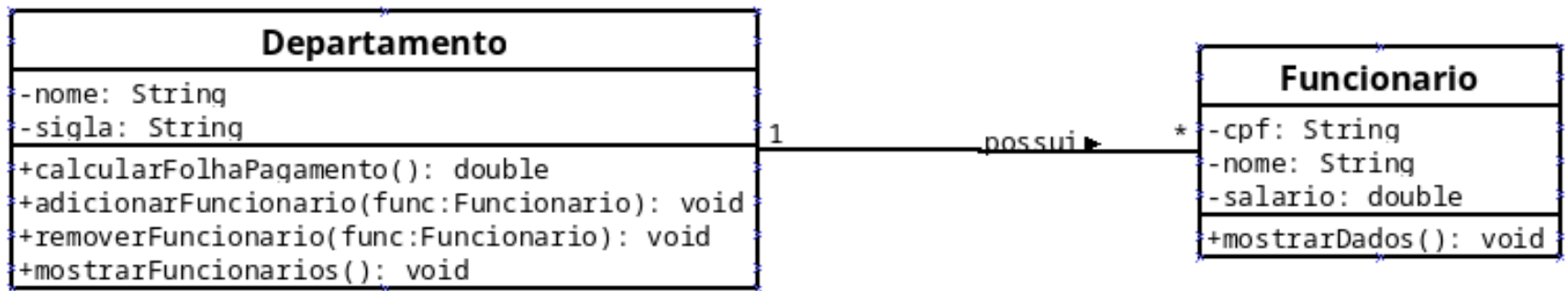


Na classe Principal:

- mostre os nomes dos autores de um determinado livro

# Exercício III

Implemente a associação entre classes usando ArrayList.  
Atenção para a multiplicidade e navegabilidade.



O método ***CalcularFolhaPagamento()*** deve retornar quanto o departamento gasta com salários de seus funcionários.