

04. Herança de Classes

Prof. Alexandre Krohn



Roteiro

- Herança de Classes
- Classes Abstratas
- Sobreescrita de Métodos
- Método toString()
- Exercícios



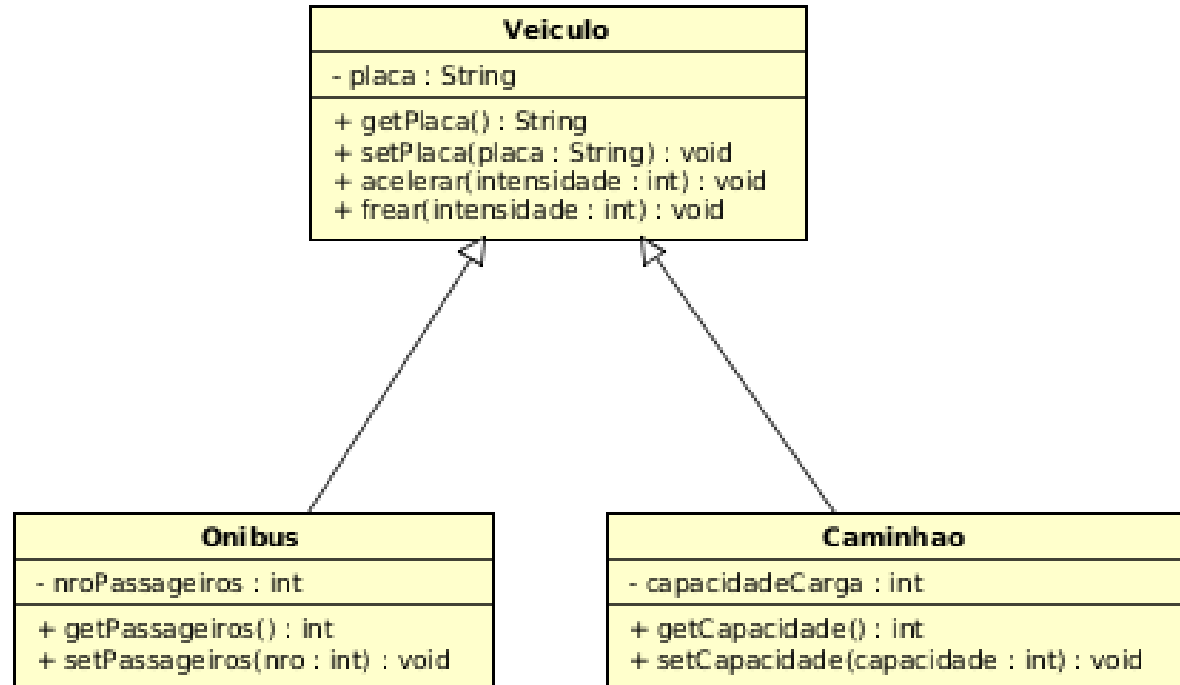
Herança (Recapitulando...)

- **Herança** é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos, através de "heranças".
- Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.



Herança

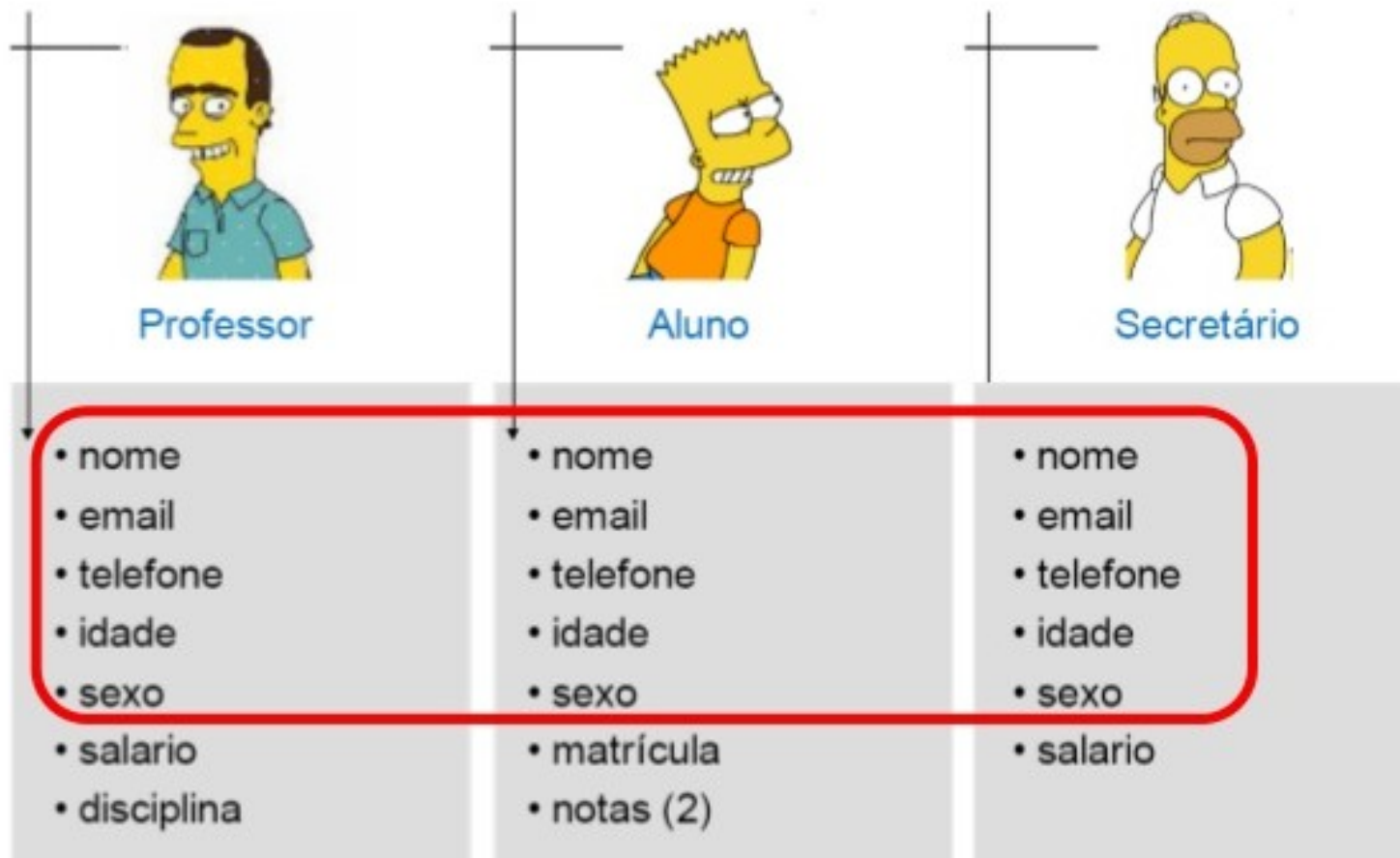
- A ideia de **herança** é facilitar a programação. Uma **classe A** deve **herdar** de uma **classe B** quando podemos dizer que **A é um B**.



- **Ônibus é um Veículo**
- **Caminhão é um Veículo**

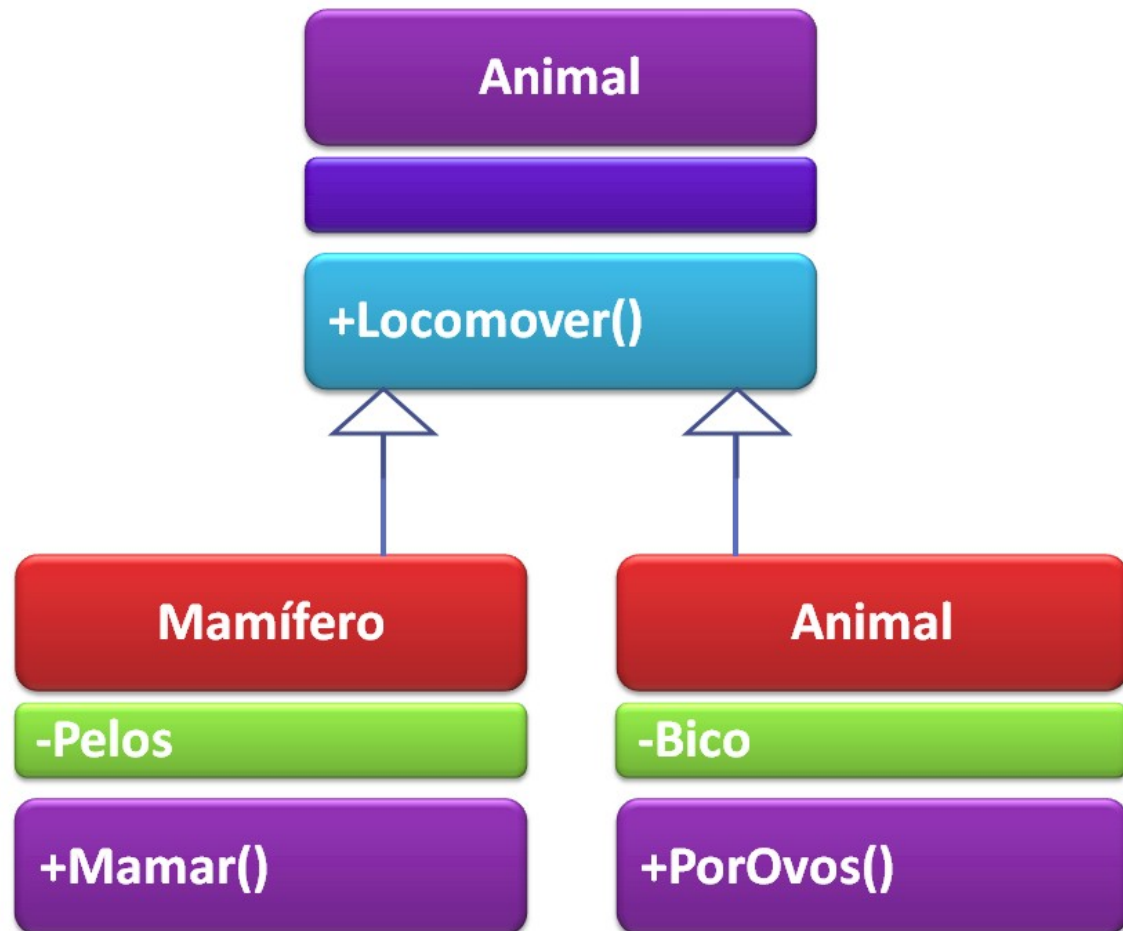
Herança

- Dados (atributos) comuns



Herança

- Comportamentos (métodos) comuns





Herança

- Reduz a quantidade de código escrito
- Diminui o tamanho dos códigos-fonte
- Distribui a complexidade da aplicação



Herança : Exemplo

- Vamos supor um sistema que trabalhe com dois tipos de pessoas:
 - Pessoas Físicas : possuem CPF
 - Pessoas Jurídicas : possuem CNPJ
 - Todas possuem nome e telefone

Herança : Exemplo

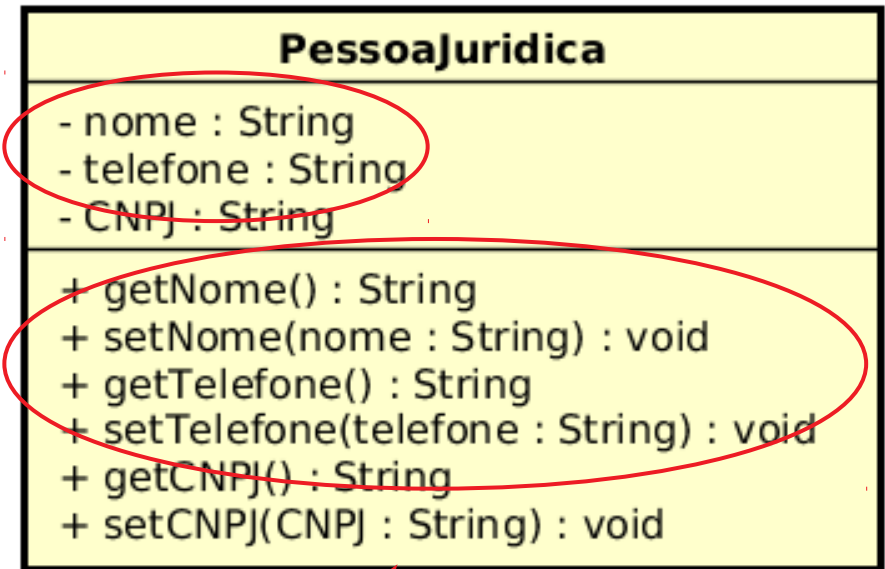
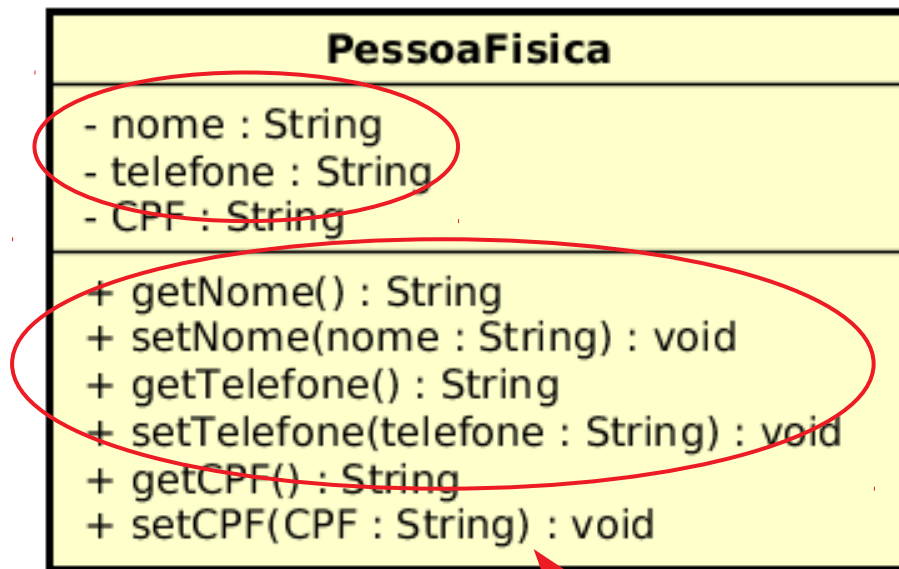
- Precisamos criar duas classes:

PessoaFisica
- nome : String - telefone : String - CPF : String
+ getNome() : String + setNome(nome : String) : void + getTelefone() : String + setTelefone(telefone : String) : void + getCPF() : String + setCPF(CPF : String) : void

PessoaJuridica
- nome : String - telefone : String - CNPJ : String
+ getNome() : String + setNome(nome : String) : void + getTelefone() : String + setTelefone(telefone : String) : void + getCNPJ() : String + setCNPJ(CNPJ : String) : void

Herança : Exemplo

- Precisamos criar duas classes:



Analizando, vemos muita
repetição aqui

```
public class PessoaFisica {  
  
    private String nome;  
    private String telefone;  
    private String CPF;  
  
    public PessoaFisica() {  
  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getTelefone() {  
        return telefone;  
    }  
  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
  
    public String getCPF() {  
        return CPF;  
    }  
  
    public void setCPF(String cPF) {  
        CPF = cPF;  
    }  
  
}
```

```
public class PessoaJuridica {  
  
    private String nome;  
    private String telefone;  
    private String CNPJ;  
  
    public PessoaJuridica() {  
  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getTelefone() {  
        return telefone;  
    }  
  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
  
    public String getCNPJ() {  
        return CNPJ;  
    }  
  
    public void setCNPJ(String cNPJ) {  
        CNPJ = cNPJ;  
    }  
  
}
```

```
public class PessoaFisica {
```

```
    private String nome;  
    private String telefone;  
    private String CPF;
```

```
    public PessoaFisica() {  
    }  
}
```

```
    public String getNome() {  
        return nome;  
    }  
}
```

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

```
    public String getTelefone() {  
        return telefone;  
    }  
}
```

```
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
}
```

```
    public String getCPF() {  
        return CPF;  
    }  
}
```

```
    public void setCPF(String cPF) {  
        CPF = cPF;  
    }  
}
```

```
}
```

```
public class PessoaJuridica {
```

```
    private String nome;  
    private String telefone;  
    private String CNPJ;
```

```
    public PessoaJuridica() {  
    }  
}
```

```
    public String getNome() {  
        return nome;  
    }  
}
```

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

```
    public String getTelefone() {  
        return telefone;  
    }  
}
```

```
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
}
```


```
    public String getCNPJ() {  
        return CNPJ;  
    }  
}
```

```
    public void setCNPJ(String cNPJ) {  
        CNPJ = cNPJ;  
    }  
}
```

```
}
```

Diferenças

```
graph TD; D[Diferenças] --> CPF[CPF]; D --> CNPJ[CNPJ]; D --> getCPF[getCPF()]; D --> setCPF[setCPF()];
```



Herança ; Exemplo

- Há muito código-fonte duplicado.
- Isso **não** é bom!

Herança : Exemplo

- Então, criaremos uma classe com o que é comum a essas duas!

PessoaFisica
- nome : String - telefone : String - CPF : String
+ getNome() : String + setNome(nome : String) : void + getTelefone() : String + setTelefone(telefone : String) : void + getCPF() : String + setCPF(CPF : String) : void

PessoaJuridica
- nome : String - telefone : String - CNPJ : String
+ getNome() : String + setNome(nome : String) : void + getTelefone() : String + setTelefone(telefone : String) : void + getCNPJ() : String + setCNPJ(CNPJ : String) : void

Herança : Exemplo

- Surge a classe **Pessoa**

Pessoa
- nome : String - telefone : String
+ getNome() : String + setNome(nome : String) : void + getTelefone() : String + setTelefone(telefone : String) : void

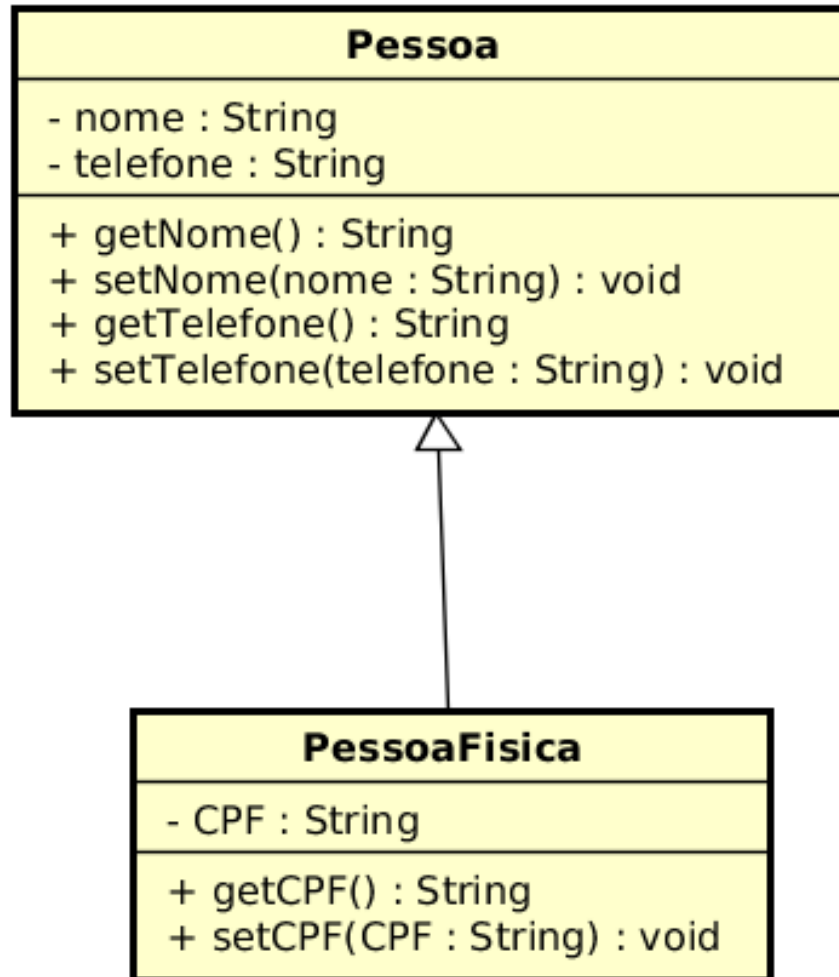
Classe Pessoa

```
public class Pessoa {  
  
    private String nome;  
    private String telefone;  
  
    public Pessoa() {  
  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getTelefone() {  
        return telefone;  
    }  
  
    public void setTelefone(String telefone) {  
        this.telefone = telefone;  
    }  
  
}
```

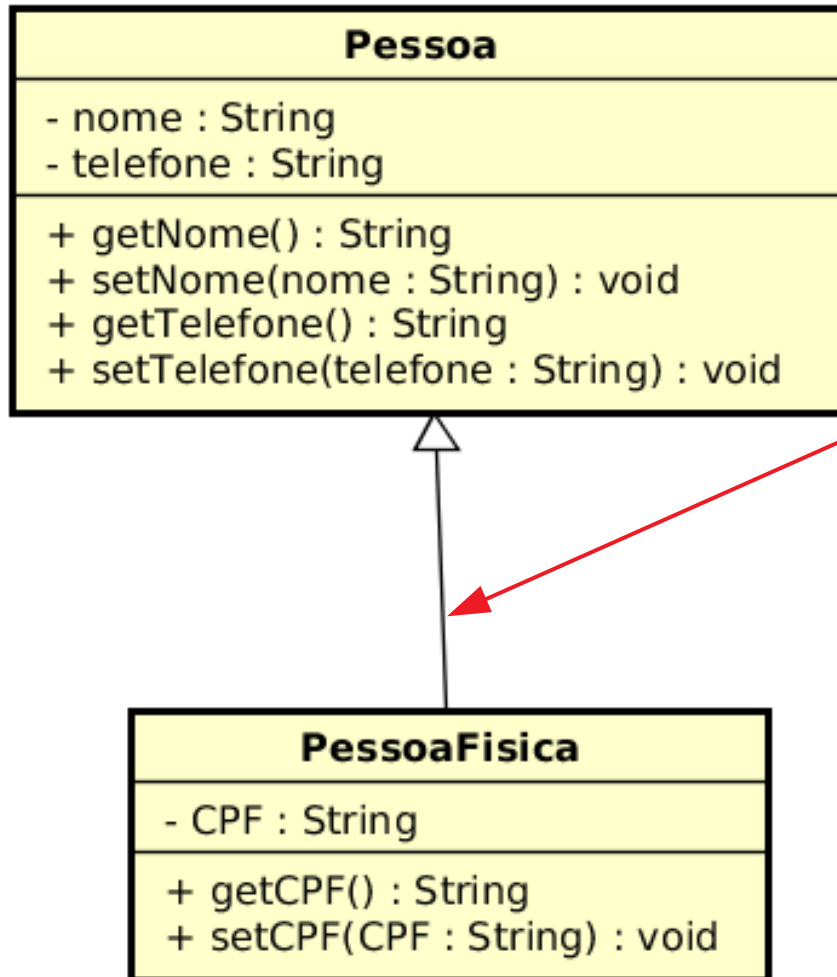
PessoaFisica herda de Pessoa

- Depois, dizemos que classe PessoaFisica herdará de pessoa, **acrescentando** o atributo e os métodos para manipular CPF

PessoaFisica herda de Pessoa



PessoaFisica herda de Pessoa



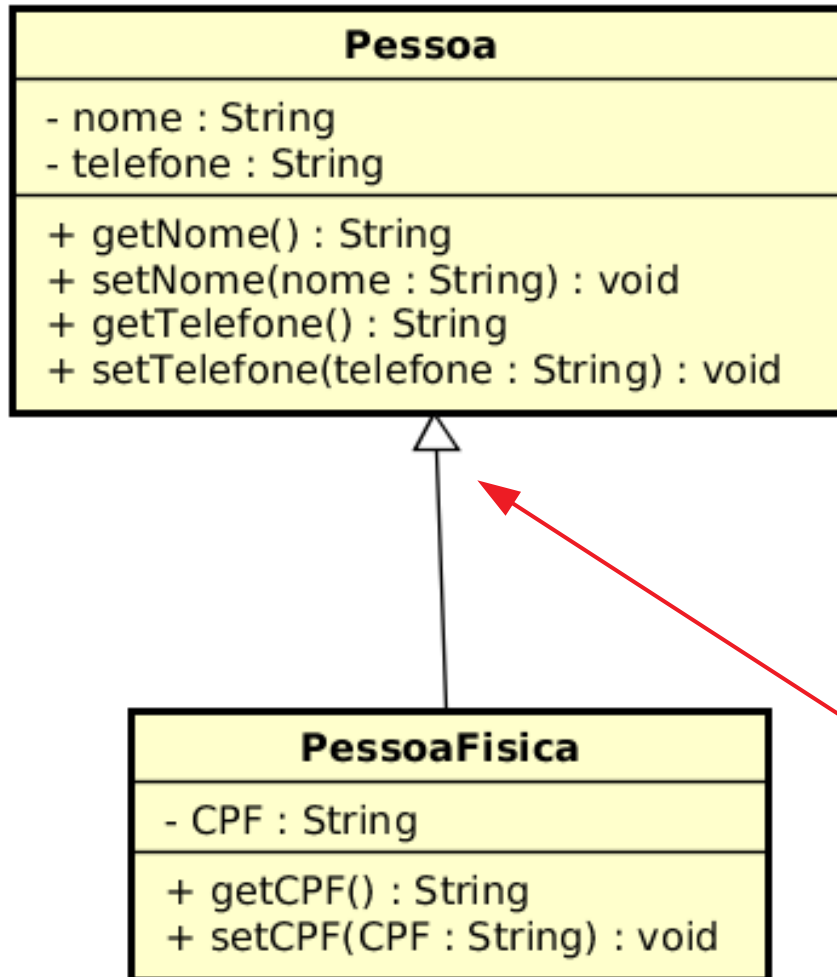
A flecha com um triângulo fechado deve ser lida como

PessoaFisica herda de Pessoa

ou

PessoaFisica é uma Pessoa

PessoaFisica herda de Pessoa



A flecha com um triângulo fechado deve ser lida como

PessoaFisica herda de Pessoa

ou

PessoaFisica é uma Pessoa


O sentido da flecha no diagrama é sempre da classe que herda para a que dá a herança

Dizemos que é de **filho** para **pai**

Herança em Java

- Para definirmos que uma classe herda de outra, em java, utilizamos a palavra-chave ***extends***

```
public class PessoaFisica extends Pessoa{  
  
}
```



Herança em Java

- Por herança, uma classe herda todas os atributos e métodos públicos (*public*) da classe “pai”, bem como os métodos protegidos (*protected*).

Pessoa Física

```
public class PessoaFisica extends Pessoa{  
    private String CPF;  
  
    public PessoaFisica() {  
  
    }  
  
    public String getCPF() {  
        return CPF;  
    }  
  
    public void setCPF(String cPF) {  
        CPF = cPF;  
    }  
}
```


Pessoa Física

```
public class PessoaFisica extends Pessoa{
```

```
    private String CPF;
```

```
    public PessoaFisica() {
```

```
    }
```

```
    public String getCPF() {
```

```
        return CPF;
```

```
    }
```

```
    public void setCPF(String CPF) {
```

```
        CPF = CPF;
```

```
    }
```

```
}
```

O termo **extends** informa que essa classe herda de outra

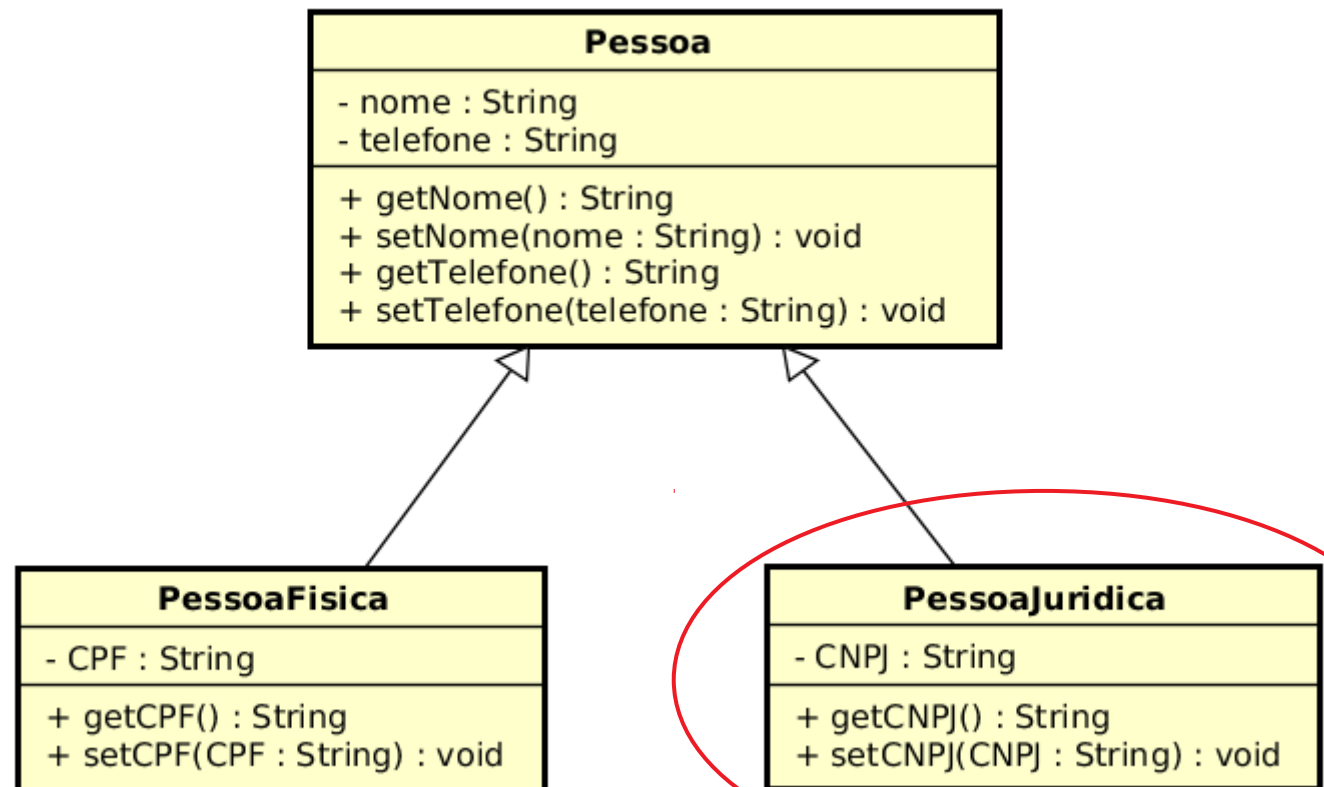
Pessoa Física

```
public class PessoaFisica extends Pessoa{  
    private String CPF;  
  
    public PessoaFisica() {  
    }  
  
    public String getCPF() {  
        return CPF;  
    }  
  
    public void setCPF(String CPF) {  
        CPF = CPF;  
    }  
}
```

Depois declara-se **SOMENTE** os atributos e métodos que diferenciam essa classe da classe “pai”

Pessoa Jurídica

- Repete-se o processo com a classe PessoaJuridica



Pessoa Jurídica

```
public class PessoaJuridica extends Pessoa{  
    private String CNPJ;  
    public PessoaJuridica() {  
    }  
    public String getCNPJ() {  
        return CNPJ;  
    }  
    public void setCNPJ(String cNPJ) {  
        CNPJ = cNPJ;  
    }  
}
```



Superclasse

- A classe Pessoa nesse exemplo é **base para a herança** das outras classes.
- Chamamos essas classes base de **superclasses**.
- A classe **Pessoa** será a **superclasse** da classe **PessoaFisica** e da classe **PessoaJuridica**



Subclasse

- A classe que herda de outra (especializa esta) é chamada de **subclasse**
- **PessoaFisica e PessoaJuridica** são **subclasses** de **Pessoa**

Uso das classes

- Instancia-se e usa-se as classes normalmente

```
PessoaFisica f1 = new PessoaFisica();
```

```
PessoaJuridica j1 = new PessoaJuridica();
```

```
f1.setNome("João da Silva");
```

```
f1.setTelefone("989899811");
```

```
f1.setCPF("123.456.789-11");
```

```
j1.setNome("Marcopolo Carrocerias");
```

```
j1.setTelefone("30259987");
```

```
j1.setCNPJ("15.183.297/0001-45");
```

Uso das classes

- Instancia-se e usa-se as classes normalmente

```
PessoaFisica f1 = new PessoaFisica();
```

```
PessoaJuridica j1 = new PessoaJuridica();
```

```
f1.setNome("João da Silva");
```

```
f1.setTelefone("989899811");
```

```
f1.setCPF("123.456.789-11");
```

```
j1.setNome("Marcopolo Carrocerias");
```

```
j1.setTelefone("30259987");
```

```
j1.setCNPJ("15.183.297/0001-45");
```

Tentar usar o método setCNPJ(..) no objeto f1 vai gerar erro, pois o mesmo não existe na classe PessoaFisica

O mesmo ocorre se tentarmos chamar setCPF(...) na classe PessoaJuridica

Os outros métodos vieram da classe Pessoa através de herança



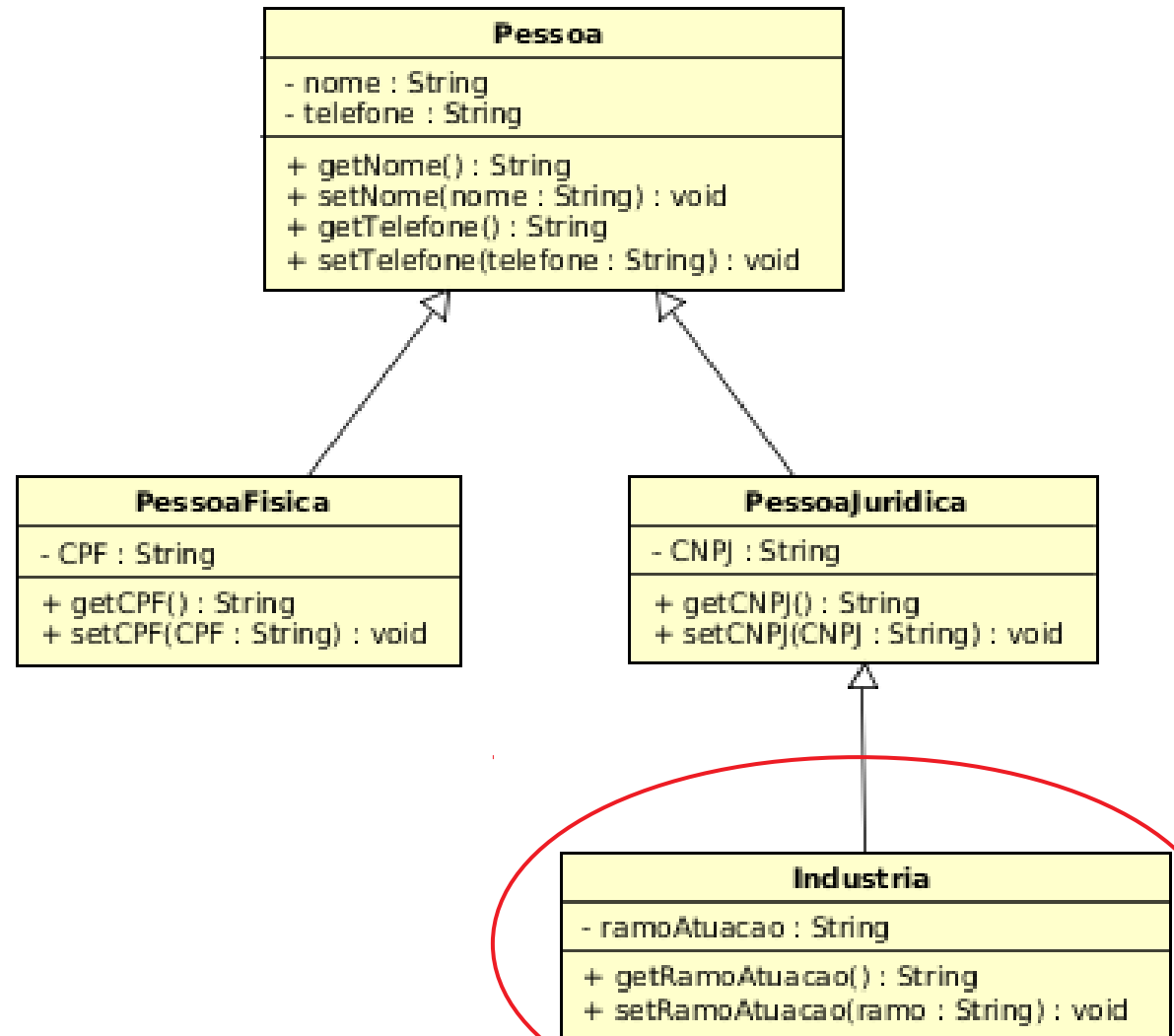
Herança de herança

- Não há limite para as heranças.
- Pode-se herdar de uma classe que herda de outra. Nesse caso, herda-se tudo que a classe anterior disponibiliza

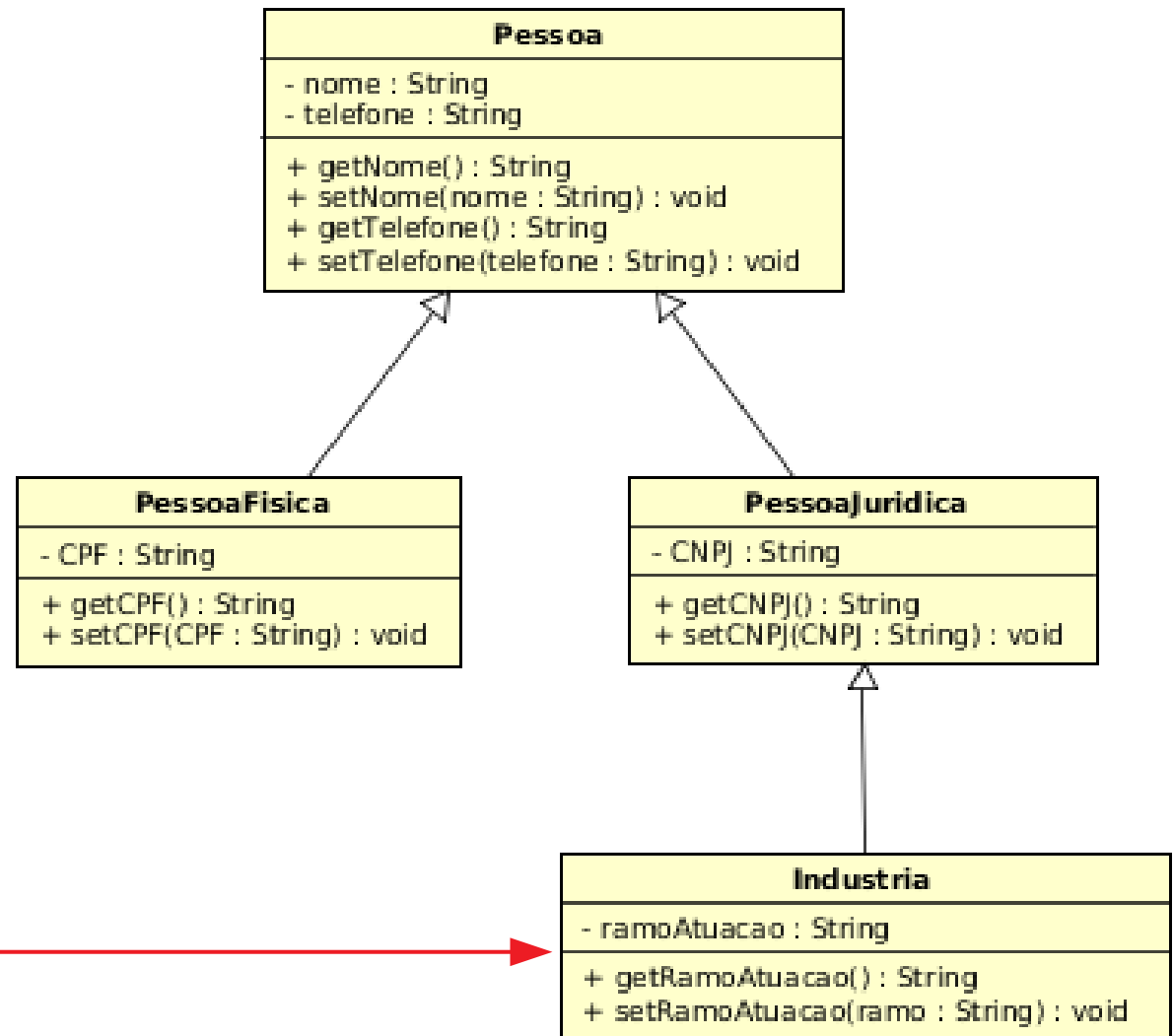
Herança de herança : Exemplo

- Vamos criar uma classe Industria
 - Indústria é uma pessoa jurídica que apresenta um ramo de atuação

Herança de herança : Exemplo



Herança de herança : Exemplo



A classe **Industria** herda as propriedades e os métodos de **PessoaJuridica** e de **Pessoa**

Herança de herança : Exemplo

```
public class Industria extends PessoaJuridica {  
    private String ramoAtividade;  
  
    public Industria() {  
  
    }  
  
    public String getRamoAtividade() {  
        return ramoAtividade;  
    }  
  
    public void setRamoAtividade(String ramoAtividade) {  
        this.ramoAtividade = ramoAtividade;  
    }  
}
```

A herança é sempre declarada como sendo da classe mais **próxima**

Herança de herança : uso

```
Industria i1 = new Industria();
```

```
i1.setNome("Da nona!");
```

```
i1.setTelefone("987677645");
```

```
i1.setCNPJ("55.436.243/0001-69");
```

```
i1.setRamoAtividade("Alimentos");
```

Classe Pessoa

- A classe Pessoa ainda pode ser utilizada por si só, se necessário:

```
Pessoa p = new Pessoa();  
p.setNome("Zézinho");  
p.setTelefone("122119811");
```



Roteiro

- Herança de Classes
- **Classes Abstratas**
- Sobreescrita de Métodos
- Método toString()
- Exercícios



Classes Abstratas

- **Classes abstratas** são classes que **não podem ser instanciadas**
- São uteis quando a superclasse não existe por si só.
 - No exemplo, imagine que só possam existir pessoas físicas ou jurídicas!

Classes Abstratas

- Define-se uma classe abstrata com a palavra-chave ***abstract***

```
public abstract class Pessoa {  
    private String nome;  
    private String telefone;  
  
    public Pessoa() {  
          
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

Classes Abstratas

- Define-se uma classe abstrata com a palavra-chave ***abstract***

```
public abstract class Pessoa {  
  
    private String nome;  
    private String telefone;  
  
    public Pessoa() {  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

Classes abstratas **não** podem ser instanciadas.

Chamar a operação **new** em uma classe abstrata gera erro

Classes Abstratas

- Erro na instanciação:

```
21 Pessoa p = new Pessoa();  
22 p.setNome("Zézinha");  
23 p.setTelefone("122119811");  
24
```

Cannot instantiate the type Pessoa



Roteiro

- Herança de Classes
- Classes Abstratas
- Sobreescrita de Métodos
- Método toString()
- Exercícios



Sobreescrita de Métodos

- Se uma subclasse possui um método com o mesmo nome e tipo de parâmetros da superclasse, o método que será chamado será o seu, e não o herdado
- Isso é chamado **sobreescrita de métodos**



Sobreescrita de Métodos

- Perceba que esse método especializa um comportamento do método da superclasse, realizando a mesma operação, porém de forma diferente

Sobreescrita de Métodos

```
public class Industria extends PessoaJuridica {  
    private String ramoAtividade;  
  
    public Industria() {}  
  
    public String getRamoAtividade() {}  
  
    public void setRamoAtividade(String ramoAtividade) {}  
  
    @Override  
    public void setNome(String nome) {  
        super.setNome("Indústria : " + nome);  
    }  
}
```

Sobreescrita de Métodos

```
public class Industria extends PessoaJuridica {  
    private String ramoAtividade;  
  
    public Industria() {}  
  
    public String getRamoAtividade() {}  
  
    public void setRamoAtividade(String ramoAtividade) {}  
  
    @Override  
    public void setNome(String nome) {  
        super.setNome("Indústria : " + nome);  
    }  
}
```

O método setNome() foi especializado.

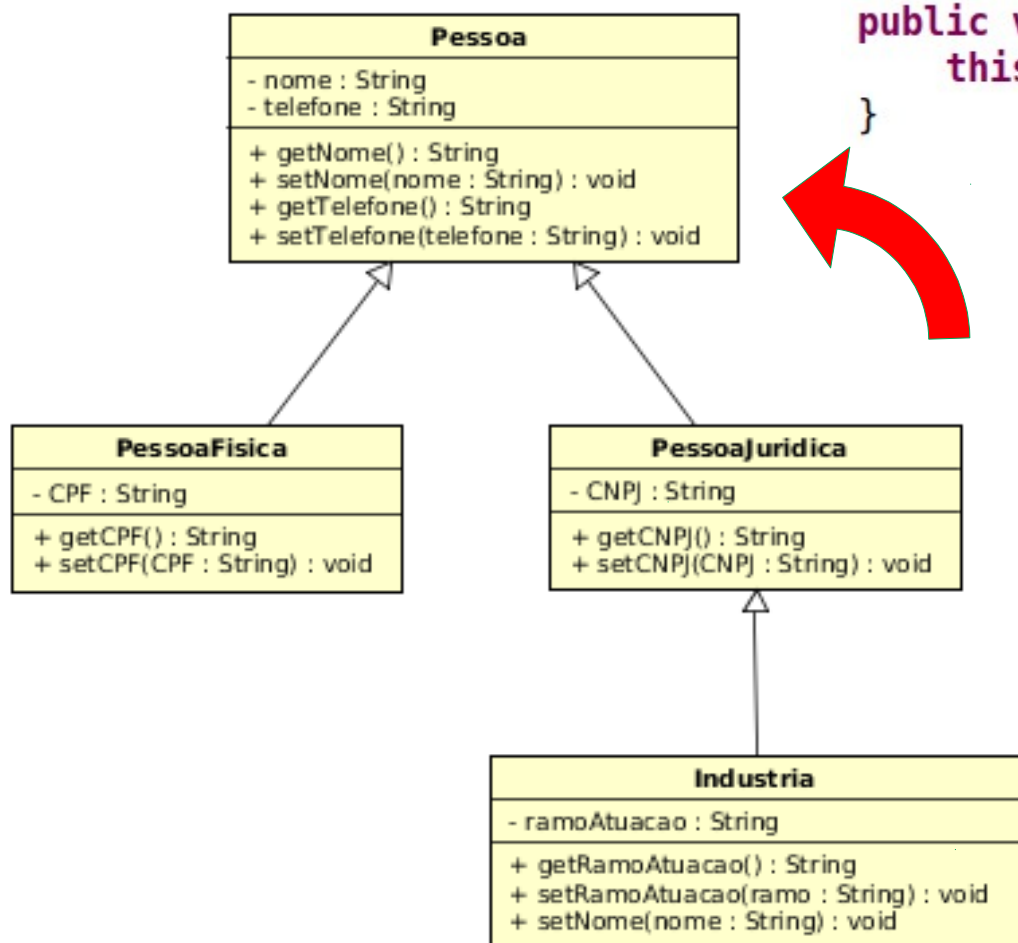
Assume um novo comportamento na classe Industria



Sobreescrita : detalhes

- **@Override** : Serve para sinalizar que um método está substituindo o comportamento de seu ancestral
- **super** : Sempre que precisamos invocar o método ancestral, chamamos **super()**

Sobreescrita : super()



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

Ao chamar um método em uma classe, Java verifica a existência do mesmo na classe.

Se for encontrado, executa, caso contrário, vai subindo na hierarquia até encontrar e então executa.

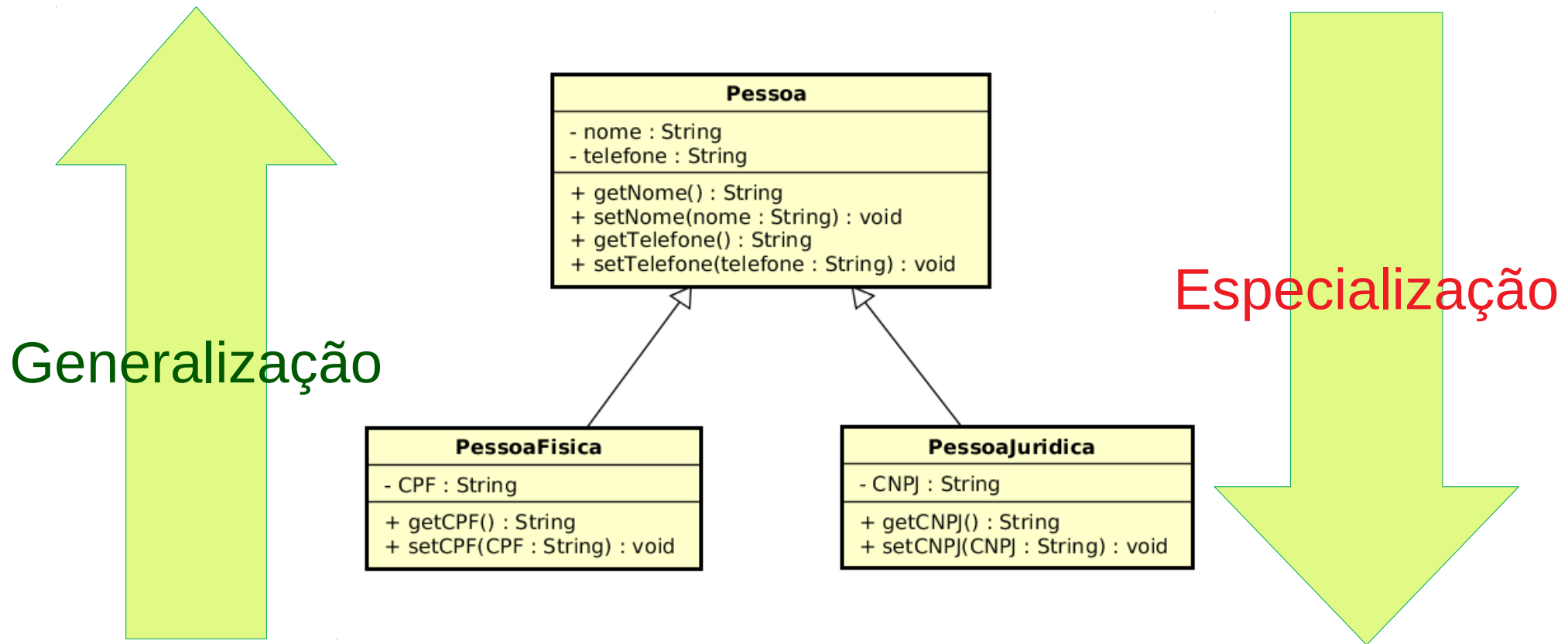
```
@Override  
public void setNome(String nome) {  
    super.setNome("Indústria : " + nome);  
}
```



Herança : Finalizando

- O uso de herança acaba por tornar o código mais simples e direto
- Há duas maneiras de “descobrir” heranças em Programação Orientada a Objetos:
 - Especializando
 - Generalizando

Generalização / Especialização





Herança : Generalização

- Depois de definir as classes que nossa aplicação precisará, procura-se por aspectos comuns nessas classes e verifica-se se faz sentido criar uma superclasse para conter esses aspectos.
- Pergunte se as duas classes poderiam ter um pai comum.
 - Ex.: Todas as pessoas que participam de uma aplicação : Funcionários, Clientes, Gerentes, etc...



Herança : Especialização

- Quando se sabe de antemão que haverá uma família de classes com informações bastante semelhantes, começa-se criando a superclasse, e depois as classes mais específicas.
 - Ex.: Peças em uma linha de montagem



Roteiro

- Herança de Classes
- Classes Abstratas
- Sobreescrita de Métodos
- Método toString()
- Exercícios



Método *toString()*

- Em Java, **todas** as classes herdam de uma superclasse chamada ***Object***
- A classe ***Object*** define um método chamado ***toString()***, que gera uma representação textual da instância do objeto em uso.

Método *toString()*

- Ex.:

```
Industria i1 = new Industria();  
  
i1.setNome("Da nona!");  
i1.setTelefone("987677645");  
i1.setCNPJ("55.436.243/0001-69");  
i1.setRamoAtividade("Alimentos");  
  
System.out.println(i1);
```



Resultado

Industria@4d7e1886

Método *toString()*

- Ex.:

```
Industria i1 = new Industria();  
  
i1.setNome("Da nona!");  
i1.setTelefone("987677645");  
i1.setCNPJ("55.436.243/0001-69");  
i1.setRamoAtividade("Alimentos");  
  
System.out.println(i1);
```

A resposta padrão do método **toString** é composta pelo nome da classe “arroba” seu código hash, que é o identificador único daquele objeto na memória:

Industria@4d7e1886

Resultado

Industria@4d7e1886

Método *toString()* : Sobreescreita

- Podemos sobrescrever o método *toString()* para que o mesmo retorne uma resposta mais adequada para nossas necessidades

Método *toString()* : Sobreescreita

```
public class Industria extends PessoaJuridica {  
  
    @Override  
    public String toString() {  
        return getNome() + "\nCNPJ : " + getCNPJ() +  
            "\nTelefone : " + getTelefone() +  
            "\nRamo : " + getRamoAtividade();  
    }  
}
```


Método *toString()* sobreescrito

- Ex.:

```
Industria i1 = new Industria();
```

```
i1.setNome("Da nona!");
```

```
i1.setTelefone("987677645");
```

```
i1.setCNPJ("55.436.243/0001-69");
```

```
i1.setRamoAtividade("Alimentos");
```

```
System.out.println(i1);
```

```
Indústria :Da nona!  
CNPJ : 55.436.243/0001-69  
Telefone : 987677645  
Ramo : Alimentos
```



Resultado



Dúvidas?



Próximos passos



- Sobrecarga de métodos



Atividades

- Execute as atividades presentes no documento

04.Lista.de.Exercícios.POO.pdf