



Universidade Estadual da Paraíba  
Campus VII – Gov. Antônio Mariz  
CCEA – Centro de Ciências Exatas e Sociais Aplicadas  
Licenciatura em Computação  
Java - Noturno

## Exercício Prático 5 – Herança

Nome: \_\_\_\_\_

# Introdução

Enquanto programamos em Java, há a necessidade de trabalharmos com várias classes. Muitas vezes, classes diferentes tem características comuns, então, ao invés de criarmos uma nova classe com todas essas características usamos as características de um objeto ou classe já existente.

Ou seja, herança é, na verdade, uma classe derivada de outra classe.

Para simplificar de uma forma mais direta, vejamos:

Vamos imaginar que exista uma classe chamada Pessoa, e nela estão definidos os seguintes atributos: Nome (String), idade (int).

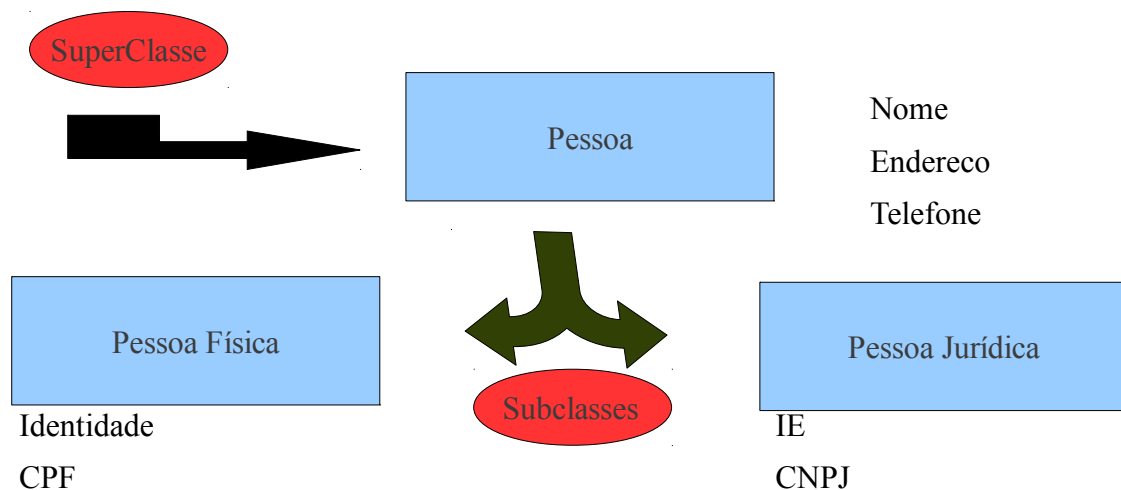
Se levarmos em conta a classe PessoaFisica que estamos usando de exemplo até agora, podemos dizer que PessoaFisica deriva de Pessoa. Ou seja, a classe PessoaFisica possui todas as características da classe Pessoa, além de ter suas próprias características.

# Extends e Super

Para fazermos uma classe herdar as características de uma outra, usamos a palavra reservada **extends** logo após a definição do nome da classe. Dessa forma:

```
public class NomeDaClasseASerCriada extends NomeDaClasseASerHerdada
```

**Importante:** Java permite que uma classe herde apenas as características de uma única classe, ou seja, não pode haver heranças múltiplas. Porém, é permitido heranças em cadeias, por exemplo: se a classe PessoaFisica herda a classe Pessoa, se por ventura fizéssemos uma classe PessoaFisicaSimples essa classe poderia derivar da classe PessoaFisica que por sua vez deriva da classe pessoa, onde conseqüentemente iria possuir todas as características das classes acima de sua hierarquia.



A **classe pai (ou superclasse)** Pessoa e suas respectivas **classes filhas (ou subclasses)**, que são PF, PJ. PF é filha da classe Pessoa e PJ também é filha de Pessoa e não tem subclasses.

Podemos ver também que a classe pessoa tem os **atributos** nome, telefone e endereço. A classe PF tem os atributos RG e CPF. A classe PJ tem os atributos CNPJ e IE. Como PF e PJ são subclasses de Pessoas, elas herdam da classe Pessoas os atributos nome, telefone e endereço. As classes Amigos e Parentes, por serem subclasses de PF, **herdam os atributos** RG e CPF, além disso, também herdam os atributos nome, telefone e endereço, já que sua classe pai é uma classe filha de Pessoa.

Por esta razão podemos dizer que a **hierarquia em Java** ocorre de cima para baixo. Uma superclasse (ou classe pai) pode ter várias subclasses (ou classes filhas) que, por sua vez, podem ter suas próprias subclasses. Mas o oposto não acontece. Não é possível para uma classe filha ter duas ou mais classes pais.

Para melhor entendimento usaremos o esquema da primeira imagem para criarmos um **programinha simples** que irá **receber os dados de pessoas digitados pelo usuário**.

## Código

Temos abaixo **código do programa** que receberá os **dados digitados pelo usuário**. Os comandos que ainda não foram explicados serão explicados antes da criação do código de cada classe. Gostaria de ressaltar que o foco aqui é a **demonstração da herança**. Por essa razão os comandos aqui são explicados apenas para facilitar o entendimento de como o código funciona.

### Pessoa.java

Aqui a classe pai é criada com seus **atributos e métodos construtores**. O primeiro método já **define o valor dos atributos** enquanto o segundo método **recebe os valores por parâmetro (como já foi explicado anteriormente)**.

```

public class Pessoa {
    String nome;
    String telefone;
}
  
```

```

String endereco;

public Pessoa() {
    nome = "";
    telefone = "";
    endereco = "";
}
public Pessoa(String nome, String telefone, String endereco) {
    this.nome = nome;
    this.telefone = telefone;
    this.endereco = endereco;
}
}

```

## PF.java

Aqui é criada a classe PF, que é filha da classe Pessoa. Por essa razão é utilizada a cláusula *extends* na criação da classe (para mostrar que **a classe PF herda atributos da classe Pessoa**). Da mesma forma que na classe Pessoa, na PF foram criados os métodos construtores. A diferença é que dessa vez utilizamos o comando *super()*, que serve para **chamar o construtor da classe pai**. É importante notar que o método construtor usado para passagem de parâmetros não tem apenas os seus atributos que já foram definidos na criação, mas também tem os atributos da classe Pessoa: public PF (String nome, String telefone, String endereco, String RG, String CPF) Na verdade tanto faz o nome que será usado. Para evitar confusões usei nomes iguais aos dos atributos: *nome*, *telefone* e *endereco*. Poderia também usar *n*, *t* e *e* que, da mesma forma, os valores seriam passados, já que o *super()* funciona de forma a enviar os parâmetros na ordem em que se encontram. **No comando *super()* foram definidos os atributos que serão enviados para a classe pai**. É justamente aqui que o *super()* faz a diferença, na **passagem por parâmetros**.

```

public class PF extends Pessoa {
    String RG;
    String CPF;

    public PF() {
        super();
        RG = "";
        CPF = "";
    }
    public PF(String nome, String telefone, String endereco,
String RG, String CPF) {
        super (nome, telefone, endereco);
        this.RG = RG;
        this.CPF = CPF;
    }
}

```

## PJ.java

```

public class PJ extends Pessoa {
    String CNPJ;
    String IE;
}

```

```

public PJ() {
    super();
    CNPJ = "";
    IE = "";
}
public PJ(String nome, String telefone, String endereco,
String CNPJ, String IE) {
    super (nome, telefone, endereco);
    this.CNPJ = CNPJ;
    this.IE = IE;
}
}

```

## CadastraPessoa.java

Este é o **método principal** (*main*), que é o **primeiro código a ser executado**. Por ser um código um pouco maior e uma classe diferente das outras, deixarei os comentários dentro do código para facilitar o entendimento e a visualização.

```

import javax.swing.JOptionPane; //import necessário para usar o JOptionPane

public class CadastraPessoa {

    public static void main(String[] args) {

        Pessoa objPessoa = new Pessoa(); //Declaração e instanciação do objeto
da classe Pessoa.

        //O comando abaixo declara uma variável local de tipo inteira (int)
chamada 'opcao' que guardo um número digitado pelo usuário. Para isso é usado o
comando JOptionPane.showInputDialog que serve justamente para pedir um valor
para o usuário. Como o JOptionPane trabalha com String e a variável opção é
inteira, foi necessário usar o comando Integer.parseInt para transformar em
inteiro o valor que foi recebido como String. Os '\n' utilizados servem para
colocar quebras de linha.
        int opcao = Integer.parseInt(JOptionPane.showInputDialog("Digite a
opção: \n\n1. Pessoa física\n2. Pessoa Jurídica"));

        //Os comandos abaixo declaram variáveis de tipo String e guardam o dado
digitado pelo usuário no JOptionPane. Como as variáveis usadas abaixo são de
tipo String, não é necessário usar o Integer.parseInt
        String nome = JOptionPane.showInputDialog("Digite o nome");
        String telefone = JOptionPane.showInputDialog("Digite o telefone");
        String endereco = JOptionPane.showInputDialog("Digite o endereço");

        //Abaixo temos o comando if que checa se o valor dentro da variável
'opcao' é 1. Se for será executado o código abaixo. Se não for, não será
executado o bloco e outro comando é executado (neste caso, outro if que checa se
a o valor em 'opcao' é 2.
        if (opcao == 1) {
            PF objPF = new PF();

            String RG = JOptionPane.showInputDialog("Digite o RG");
            String CPF = JOptionPane.showInputDialog("Digite o CPF");

```

```

opcao = Integer.parseInt(JOptionPane.showInputDialog("Digite a
opção:\n\n1. Amigos\n2. Parentes"));
if (opcao == 1) {
    String blog = JOptionPane.showInputDialog("Digite o blog");

    //Por fim, abaixo é declarado e instanciado o objeto de classe
Amigos. Dessa vez os valores são passados por parâmetros. Como podemos notar, os
códigos anteriores pediram os dados para os usuários e armazenaram nas variáveis
nome, telefone, endereco, RG, CPF e blog. Agora esses dados estão sendo passados
por parâmetros já na instanciação do objeto. Aqui notamos como funciona a
hierarquia em Java e também como o comando super() trabalha, já que primeiro são
passados os parâmetros para os atributos da classe Pessoa (nome, telefone,
endereco), depois para a classe PF (RG, CPF) e, por fim, para a classe Amigos
(blog)

        Amigos objAmigos = new
Amigos(nome, telefone, endereco, RG, CPF, blog);
    }
    if (opcao == 2) {
        String email = JOptionPane.showInputDialog("Digite o e-mail");
        Parentes objParentes = new
Parentes(nome, telefone, endereco, RG, CPF, email);
    }

    if (opcao == 2) {
        String CNPJ = JOptionPane.showInputDialog("Digite o CNPJ");
        String IE = JOptionPane.showInputDialog("Digite a Inscrição
Estadual");
        PJ objPJ = new PJ(nome, telefone, endereco, CNPJ, IE);
    }
}
}
}

```

## Exercícios

1. Crie mais duas classes que herdem a Classe Pessoa e modifique a classe principal para que está possa trabalhar também com essas novas classes.

Comando	Execução
<b>ls</b>	lista os arquivos de um determinado diretório
<b>pwd</b>	mostra o diretório atual
<b>mkdir &lt;nome&gt;</b>	cria um diretório com o <nome> passado como parâmetros
<b>cd &lt;nome&gt;</b>	direciona o terminal para o diretório com o <nome>
<b>javac &lt;nome&gt;.java</b>	compila um arquivo com o nome passado e gera o bytecode
<b>java &lt;nome&gt;</b>	executa o bytecode gerado pelo processo de compilação
<b>javadoc &lt;nome&gt;.java</b>	gera o javadoc do arquivo <nome>.java