

## Aula 5: Herança

### Apresentação

---

Entenderemos o que é herança e como podemos aplicá-la na criação de novas classes com base nas existentes. Veremos que sua utilização permite o reaproveitamento do código, gerando economia de tempo durante o processo de desenvolvimento da aplicação, bem como facilidade de manutenção do programa.

Compreenderemos como acessar membros da superclasse com super a partir de uma subclasse, e exploraremos o conceito de polimorfismo de sobrescrita.

### Objetivos

---

- Analisar o conceito de herança;
- Aplicar o conceito de herança na construção e análise de programas em Java.

# Herança



Fonte: [Unsplash](#)

A programação orientada a objetos tem como um dos principais pilares o reaproveitamento de código. Reaproveitar o código significa menos esforço em seu desenvolvimento e mais facilidade na manutenção do sistema. Ao evitarmos a redundância de código, fica mais fácil gerar alterações, uma vez que não precisaremos modificá-lo em vários locais diferentes.

## Atenção

A herança é um conceito muito importante que possibilita identificar duas ou mais classes que possuam semelhanças. Estas podem ser definidas através de uma hierarquia, em que os membros comuns às duas ou mais classes passam para uma nova classe, conhecida como Superclasse ou classe “mãe”.

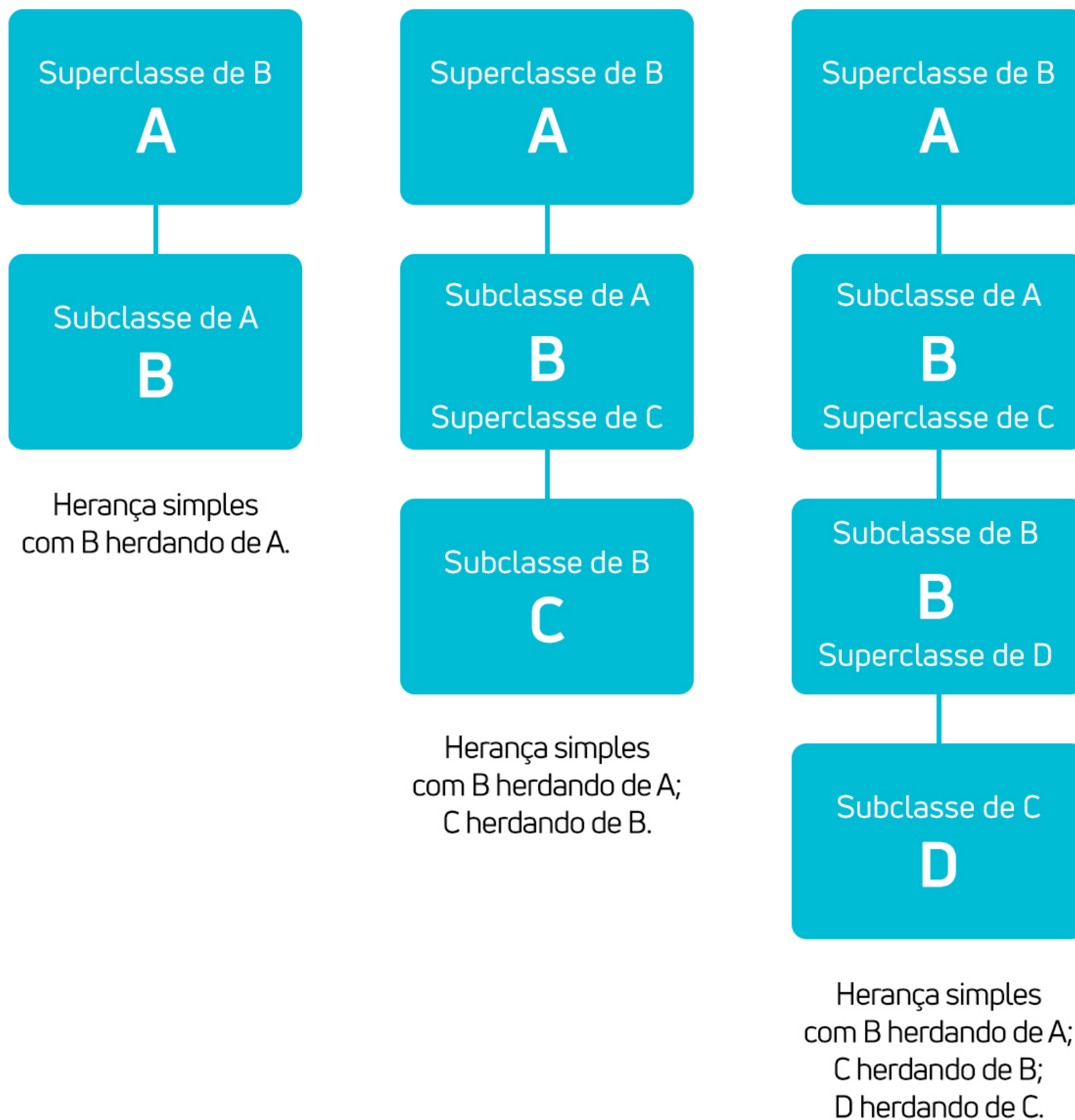
Já as classes originais permanecerão apenas com os membros não comuns, sendo denominadas Subclasses ou classes “filhas”. **Ao aplicar este conceito, podemos trabalhar com uma hierarquia entre as classes, em que as de maior hierarquia aglutinam os membros comuns e as de menor hierarquia possuem apenas membros distintos entre elas.**

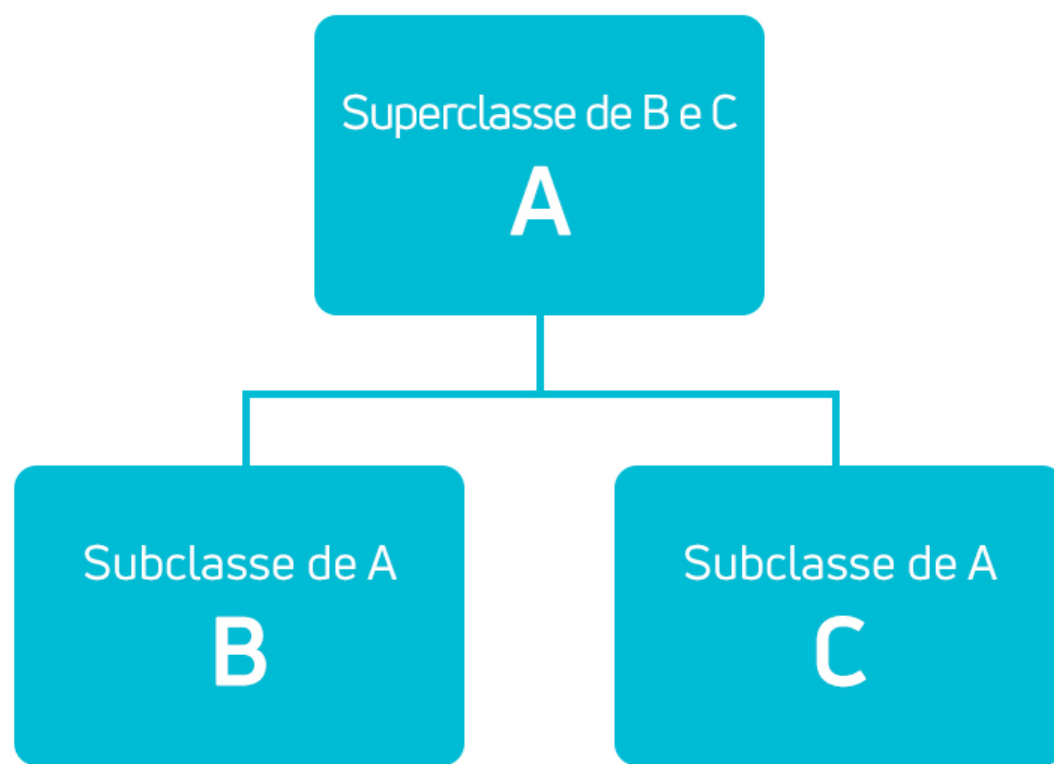
**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online



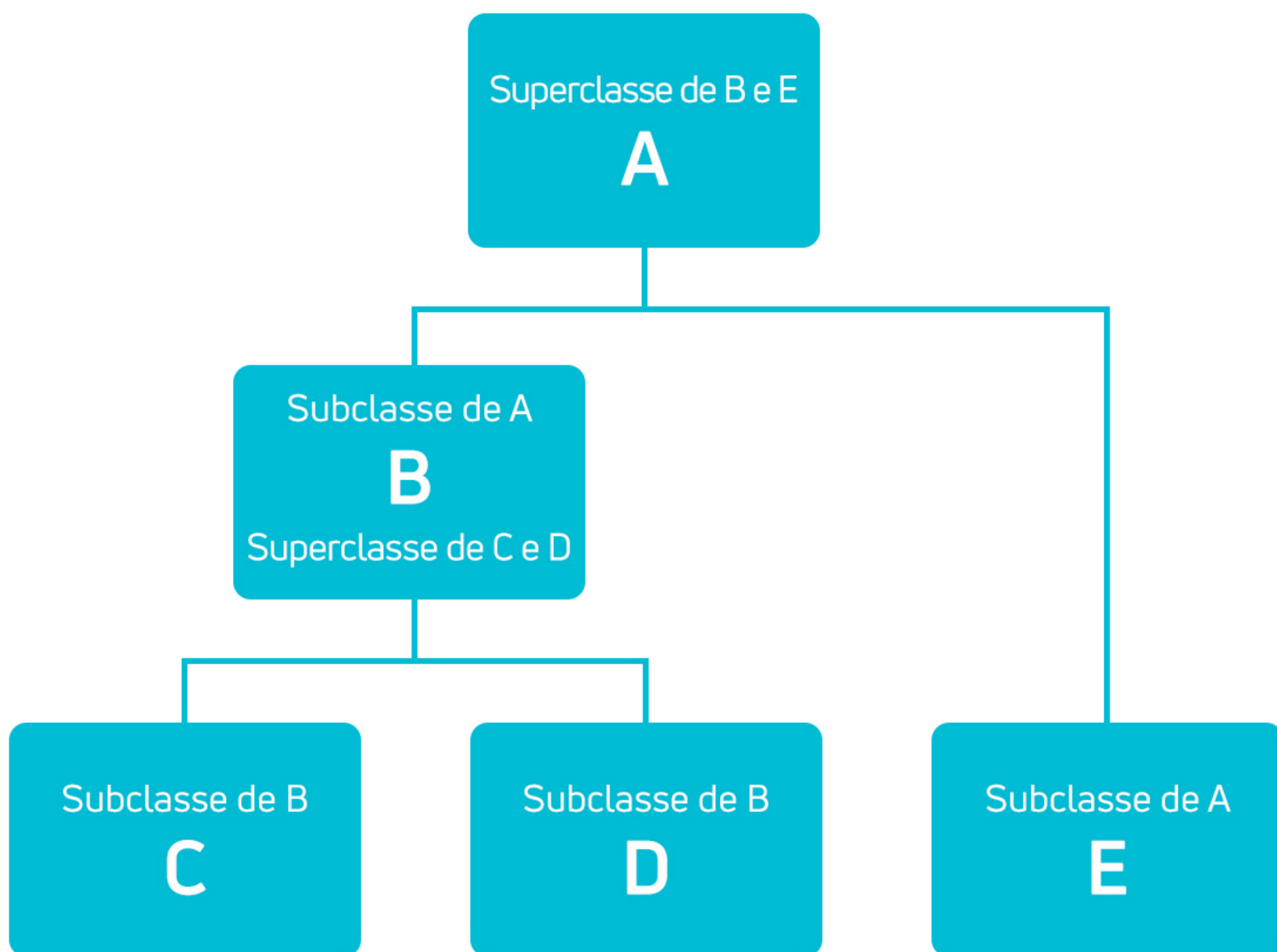


Em Java, temos apenas a implementação da herança simples. A herança simples se caracteriza por cada classe herdar sempre de apenas uma outra classe por vez. Devemos observar que, mesmo que tenhamos uma sequência de classes herdando, em que uma herda da outra, ainda assim, temos a herança simples, que pode ser observada nos exemplos das figuras a seguir:

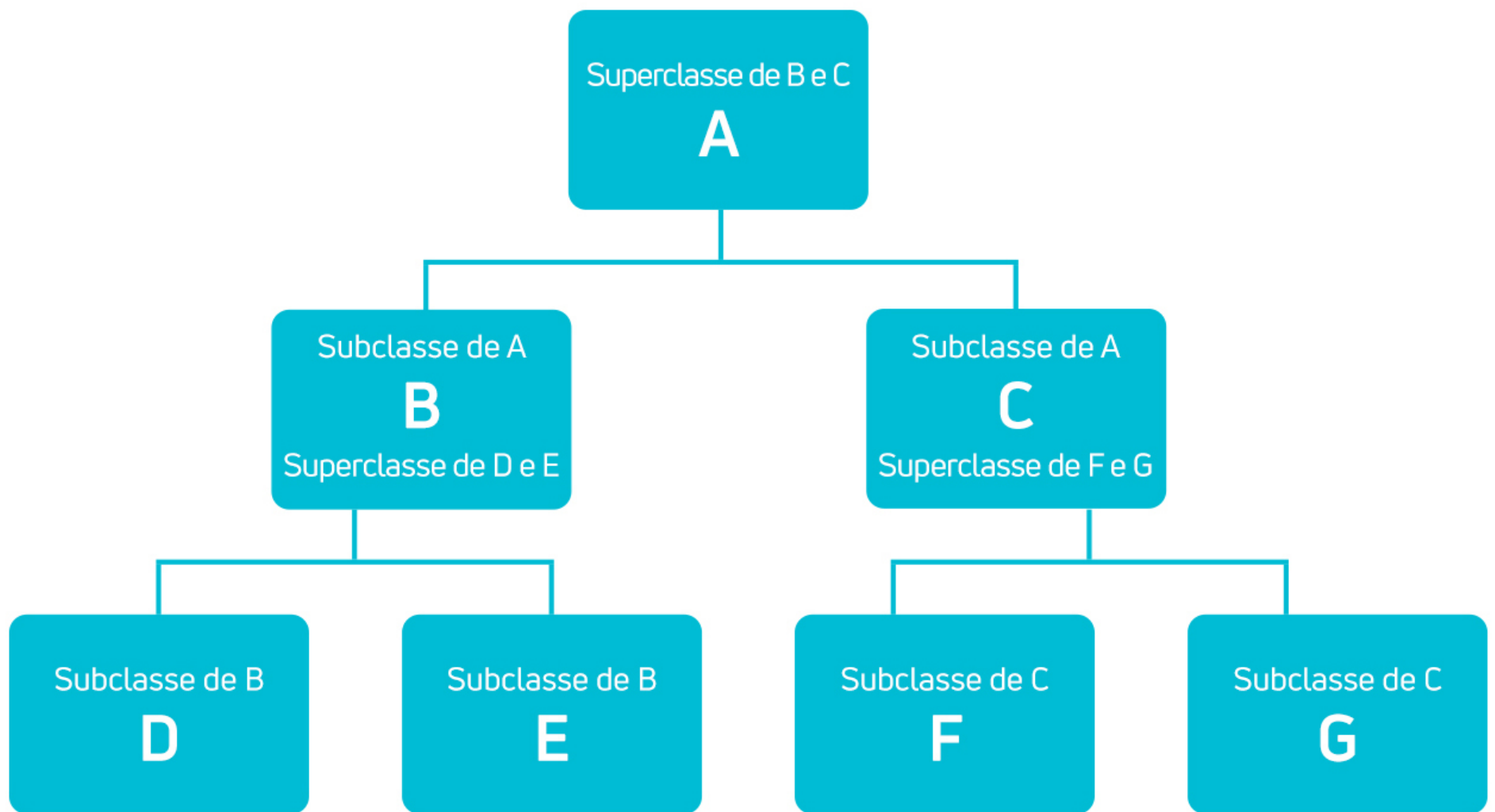




Herança simples com B herdando de A; C herdando de A.



Herança simples com B herdando de A; C e D herdando de B; E herdando de A.



Herança simples com B e C herdando de A; D e E herdando de B; F e G herdando de C.

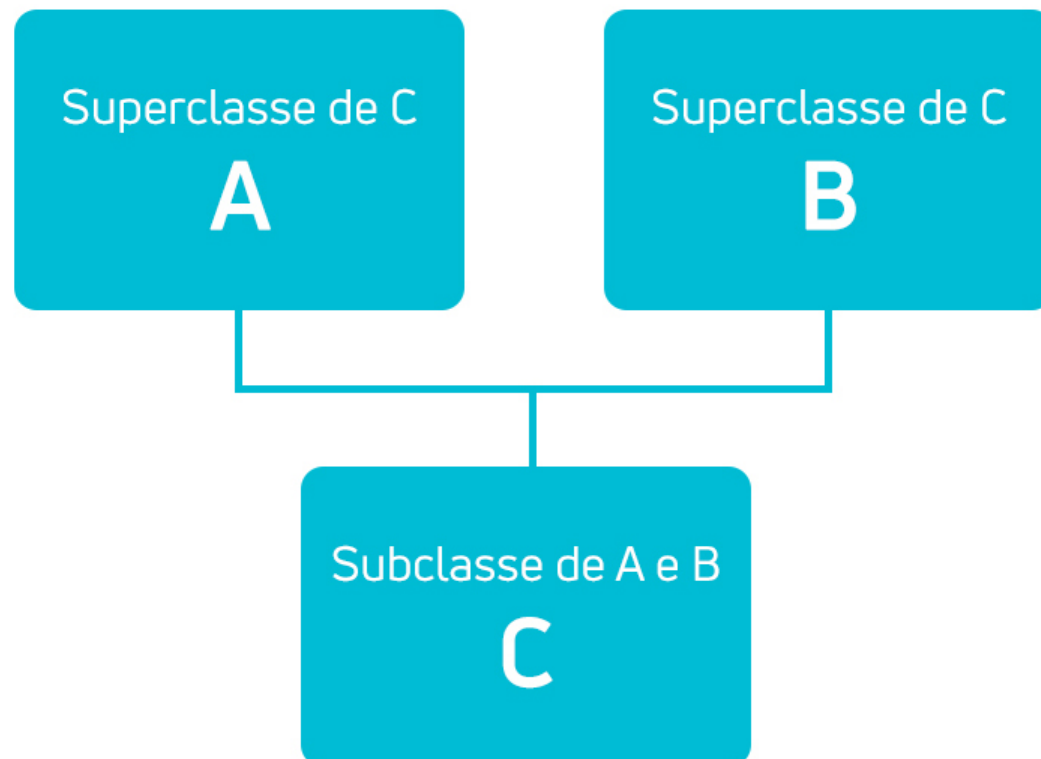
### Nota:

Em todos os casos, temos sempre a herança simples, pois cada classe sempre herda apenas da sua classe superior, mesmo se incluirmos novos níveis.

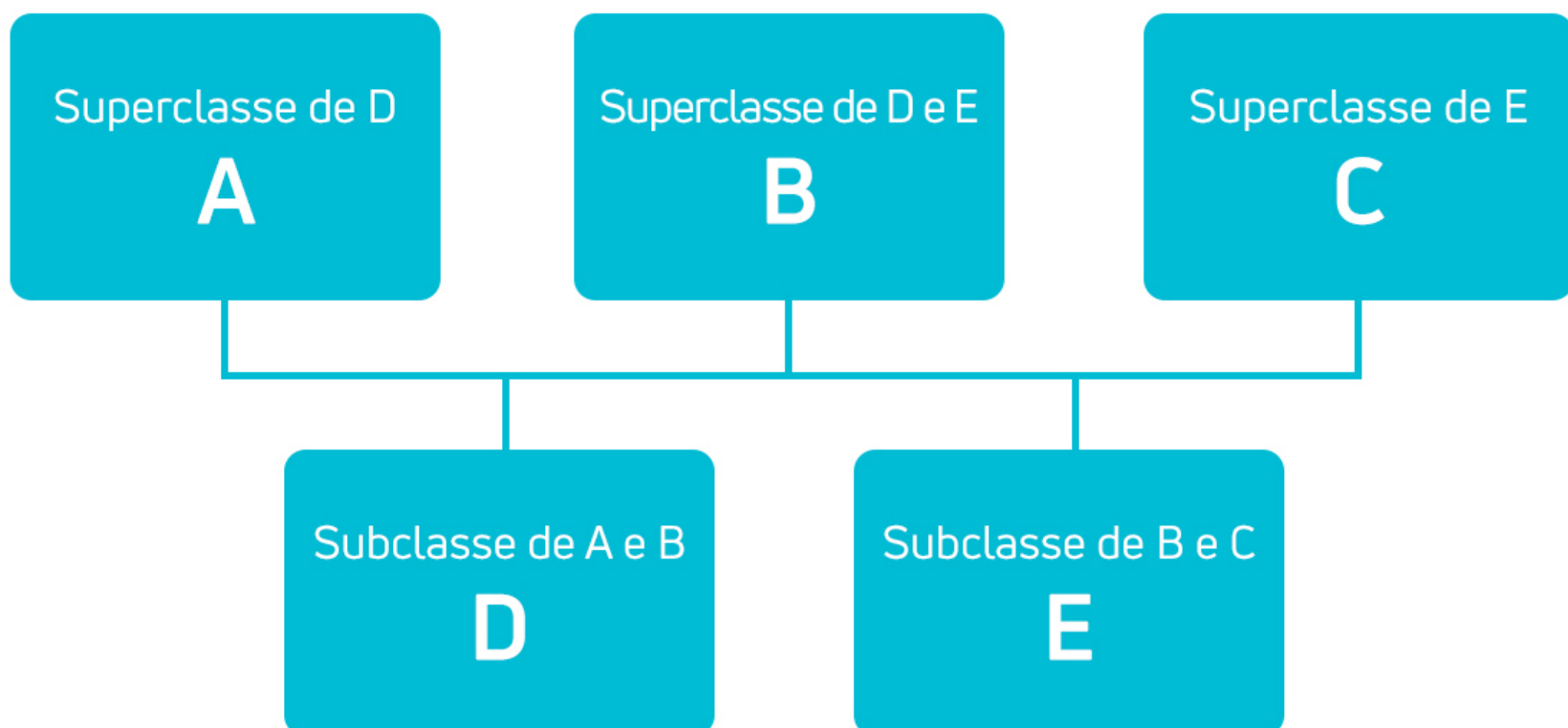


A herança múltipla se caracteriza quando uma mesma classe herda de duas ou mais classes ao mesmo tempo. Java não permite a implementação da herança múltipla; mesmo este sendo um conceito da programação orientada a objetos, algumas linguagens de programação não implementam este conceito. A linguagem C permite a implementação de herança múltipla, mas Java e C#, por exemplo, não permitem esta implementação.

A herança múltipla pode ser observada nos exemplos das figuras a seguir:



Herança múltipla com C herdando ao mesmo tempo de A e B.



Herança múltipla com D herdando ao mesmo tempo de A e B;  
E herdando ao mesmo tempo de B e C.



# Herança em Java

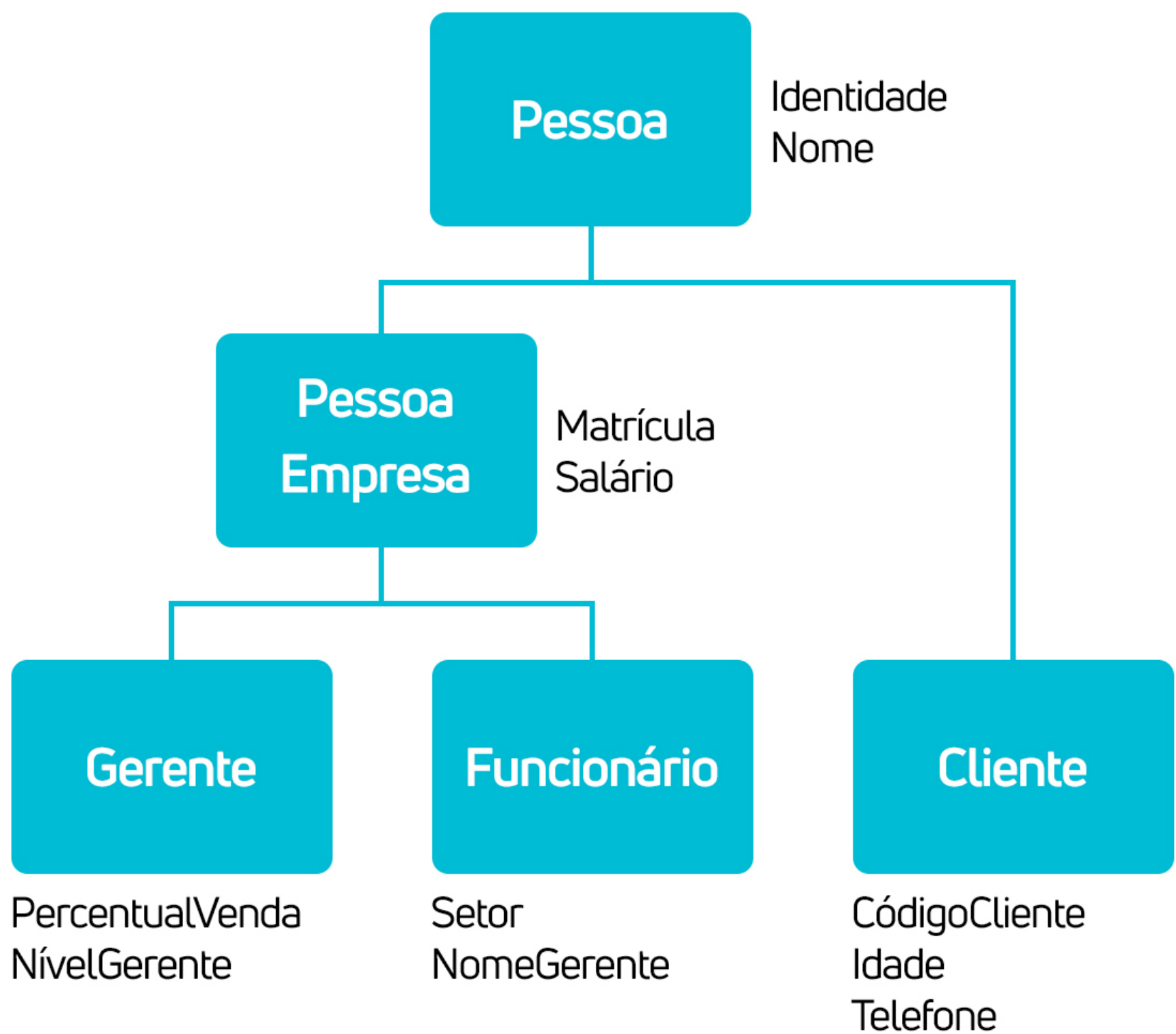
Vejamos um exemplo:

Sejam as três classes a seguir referentes a uma empresa:

Classe	Atributos	Atributos
Gerente	Identidade: texto Nome: texto Matrícula: texto Salário: real PercentualVenda: real NívelGerente: inteiro	<ul style="list-style-type: none"><li>• Setters [para todos os atributos]</li><li>• Getters [para todos os atributos]</li><li>• Construtores [mínimo 5]</li><li>• Imprimir [para exibir os atributos]</li><li>• EntradaDados [para todos os atributos]</li><li>• Cadastrar [atribui a todos os atributos]</li></ul>
Funcionário	Identidade: texto Nome: texto Matrícula: texto Salário: real Setor: texto NomeGerente: texto	<ul style="list-style-type: none"><li>• Setters [para todos os atributos]</li><li>• Getters [para todos os atributos]</li><li>• Construtores [mínimo 5]</li><li>• Imprimir [para exibir os atributos]</li><li>• EntradaDados [para todos os atributos]</li><li>• Cadastrar [atribui a todos os atributos]</li></ul>
Cliente	Identidade: texto Nome: texto CódigoCliente: texto Idade: inteiro Telefone: texto	<ul style="list-style-type: none"><li>• Setters [para todos os atributos]</li><li>• Getters [para todos os atributos]</li><li>• Construtores [mínimo 5]</li><li>• Imprimir [para exibir os atributos]</li><li>• EntradaDados [para todos os atributos]</li><li>• Cadastrar [atribui a todos os atributos]</li></ul>

Ao analisar as classes, podemos afirmar que teremos que repetir a declaração dos atributos Identidade e Nome em todas elas, bem como os métodos *Setters* e *Getters* destes atributos em todas as três classes, sem contar que ainda teremos que repetir parte da codificação dos métodos cadastrar, imprimir e entradaDados.

Neste caso, podemos aplicar os conceitos de herança e teremos a seguinte estrutura para atender ao problema descrito acima:




- Como os atributos Identidade e Nome são comuns às três classes, eles ficarão na Superclasse de maior hierarquia;
- Como os atributos Matrícula e Salário pertencem somente às classes Gerente e Funcionário, teremos uma classe intermediária chamada PessoaEmpresa, que herdará da classe Pessoa, mas esta também será uma Superclasse para as classes Gerente e Funcionário;
- A classe Cliente herdará da classe Pessoa;
- As classes Gerente e Funcionário herdarão da classe PessoaEmpresa.

Desta forma, não teremos redundância de códigos, escreveremos menos linhas e teremos maior facilidade na manutenção das classes da nossa biblioteca de classes.

Vamos analisar agora as diferenças entre as versões das nossas classes com e sem a aplicação da herança:

 Classe Gerente sem o uso do conceito de herança:

 Clique no botão acima.

## Classe Gerente sem o uso do conceito de herança:

```
import java.util.Scanner;
public class Gerente {
    String identidade, nome, matricula;
    double salario, percentualVenda;
    int nivelGerente;
    public String getIdentidade() {
        return identidade;
    }
    public void setIdentidade(String id) {
        if (!id.isEmpty()) {
            identidade = id;
        }
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String no) {
        if (!no.isEmpty()) {
            nome = no;
        }
    }
    public String getMatricula() {
        return matricula;
    }
    public void setMatricula(String ma) {
        if (!ma.isEmpty()) {
            matricula = ma;
        }
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double sa) {
        if (sa >= 0) {
            salario = sa;
        }
    }
    public double getPercentualVenda() {
        return percentualVenda;
    }
    public void setPercentualVenda(double pv) {
        if (pv >= 0) {
            percentualVenda = pv;
        }
    }
    public int getNivelGerente() {
        return nivelGerente;
    }
    public void setNivelGerente(int ng) {
        if (ng >= 0) {
            nivelGerente = ng;
        }
    }
    public Gerente() { }
    public Gerente(String id) {
```

```

        setIdentidade(id);
    }

    public Gerente(double sa) {
        setSalario(sa);
    }

    public Gerente(String id, double sa) {
        setIdentidade(id);
        setSalario(sa);
    }

    public Gerente(double sa, String id) {
        setIdentidade(id);
        setSalario(sa);
    }

    public Gerente(String id, String no, String ma, double sa, double pv, int ng) {
        setIdentidade(id);
        setNome(no);
        setMatricula(ma);
        setSalario(sa);
        setPercentualVenda(pv);
        setNivelGerente(ng);
    }

    public void cadastrar(String id, String no, String ma, double sa, double pv, int ng) {
        setIdentidade(id);
        setNome(no);
        setMatricula(ma);
        setSalario(sa);
        setPercentualVenda(pv);
        setNivelGerente(ng);
    }

    public void entradaDados() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Identidade :");
        setIdentidade(entrada.nextLine());
        System.out.println("Nome :");
        setNome(entrada.nextLine());
        System.out.println("Matrícula :");
        setMatricula(entrada.nextLine());
        System.out.println("Salário :");
        setSalario(Double.parseDouble(entrada.nextLine()));
        System.out.println("Percentual de Venda:");
        setPercentualVenda(Double.parseDouble(entrada.nextLine()));
        System.out.println("Nível Gerencia :");
        setNivelGerente(Integer.parseInt(entrada.nextLine()));
        entrada.close();
    }

    public void imprimir() {
        System.out.println("Identidade :" + getIdentidade());
        System.out.println("Nome :" + getNome());
        System.out.println("Matrícula :" + getMatricula());
        System.out.println("Salário :" + getSalario());
        System.out.println("Percentual de Venda:" + getPercentualVenda());
        System.out.println("Nível Gerencia :" + getNivelGerente());
    }
}

```

Total de 109 linhas de código.

Classe Funcionário sem o uso do conceito de herança:

```
import java.util.Scanner;
public class Funcionario {
    String identidade, nome, matricula, setor, nomeGerente;
    double salario;
    public String getIdentidade() {
        return identidade;
    }
    public void setIdentidade(String id) {
        if (!id.isEmpty()) {
            identidade = id;
        }
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String no) {
        if (!no.isEmpty()) {
            nome = no;
        }
    }
    public String getMatricula() {
        return matricula;
    }
    public void setMatricula(String ma) {
        if (!ma.isEmpty()) {
            matricula = ma;
        }
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double sa) {
        if (sa >= 0) {
            salario = sa;
        }
    }
    public String getSetor() {
        return setor;
    }
    public void setSetor(String se) {
        if (!se.isEmpty()) {
            setor = se;
        }
    }
    public String getNomeGerente() {
        return nomeGerente;
    }
    public void setNomeGerente(String ng) {
        if (!ng.isEmpty()) {
            nomeGerente = ng;
        }
    }
    public Funcionario() { }
    public Funcionario(String id) {
        setIdentidade(id);
    }
    public Funcionario(double sa) {
        setSalario(sa);
    }
}
```

```

}

public Funcionario(String id, double sa) {
    setIdentidade(id);
    setSalario(sa);
}

public Funcionario(double sa, String id) {
    setIdentidade(id);
    setSalario(sa);
}

public Funcionario(String id, String no, String ma, double sa, String se, String ng) {
    setIdentidade(id);
    setNome(no);
    setMatricula(ma);
    setSalario(sa);
    setSetor(se);
    setNomeGerente(ng);
}

public void cadastrar(String id, String no, String ma, double sa, String se, String ng) {
    setIdentidade(id);
    setNome(no);
    setMatricula(ma);
    setSalario(sa);
    setSetor(se);
    setNomeGerente(ng);
}

public void entradaDados() {
    Scanner entrada = new Scanner(System.in);
    System.out.println("Identidade :");
    setIdentidade(entrada.nextLine());
    System.out.println("Nome :");
    setNome(entrada.nextLine());
    System.out.println("Matrícula :");
    setMatricula(entrada.nextLine());
    System.out.println("Salário :");
    setSalario(Double.parseDouble(entrada.nextLine()));
    System.out.println("Setor :");
    setSetor(entrada.nextLine());
    System.out.println("Nome Gerente:");
    setNomeGerente(entrada.nextLine());
}

public void imprimir() {
    System.out.println("Identidade :" + getIdentidade());
    System.out.println("Nome :" + getNome());
    System.out.println("Matrícula :" + getMatricula());
    System.out.println("Salário :" + getSalario());
    System.out.println("Setor :" + getSetor());
    System.out.println("Nome Gerente:" + getNomeGerente());
}
}

```

Total de 110 linhas de código.

Classe Cliente sem o uso do conceito de herança:

```

import java.util.Scanner;

public class Cliente {
    String identidade, nome, codigoCliente, telefone;

```

```
int idade;
public String getIdentidade() {
    return identidade;
}
public void setIdentidade(String id) {
    if (!id.isEmpty()) {
        identidade = id;
    }
}
public String getNome() {
    return nome;
}
public void setNome(String no) {
    if (!no.isEmpty()) {
        nome = no;
    }
}
public String getCodigoCliente() {
    return codigoCliente;
}
public void setCodigoCliente(String cc) {
    if (!cc.isEmpty()) {
        codigoCliente = cc;
    }
}
public String getTelefone() {
    return telefone;
}
public void setTelefone(String tf) {
    if (!tf.isEmpty()) {
        telefone = tf;
    }
}
public int getIdade() {
    return idade;
}
public void setIdade(int id) {
    if (id >= 0) {
        idade = id;
    }
}
public Cliente() { }
public Cliente(String id) {
    setIdentidade(id);
}
public Cliente(int id) {
    setIdade(id);
}
public Cliente(String id, int ida) {
    setIdentidade(id);
    setIdade(ida);
}
public Cliente(int ida, String id) {
    setIdentidade(id);
    setIdade(ida);
}
public Cliente(String id, String no, String cc, String tf, int ida) {
    setIdentidade(id);
    setNome(no);
    setCodigoCliente(cc);
```



```

        setTelefone(tf);
        setIdade(ida);
    }

    public void cadastrar(String id, String no, String cc, String tf, int ida) {
        setIdentidade(id);
        setNome(no);
        setCodigoCliente(cc);
        setTelefone(tf);
        setIdade(ida);
    }

    public void entradaDados() {
        Scanner entrada = new Scanner(System.in);
        System.out.println("Identidade :");
        setIdentidade(entrada.nextLine());
        System.out.println("Nome :");
        setNome(entrada.nextLine());
        System.out.println("Código do Cliente :");
        setCodigoCliente(entrada.nextLine());
        System.out.println("Telefone :");
        setTelefone(entrada.nextLine());
        System.out.println("Idade :");
        setIdade(Integer.parseInt(entrada.nextLine()));
        entrada.close();
    }

    public void imprimir() {
        System.out.println("Identidade :" + getIdentidade());
        System.out.println("Nome :" + getNome());
        System.out.println("Código do Cliente :" + getCodigoCliente());
        System.out.println("Telefone :" + getTelefone());
        System.out.println("Idade :" + getIdade());
    }
}

```

Total de 96 linhas de código.

Todos os códigos marcados na cor vermelha estão duplicados nas três classes. Os códigos marcados na cor verde estão duplicados nas classes Gerente e Funcionário.

**Foram necessárias, para criar as três classes, um total de (109 + 110 + 96 =) 315 linhas de código.**

Vamos a algumas perguntas:

**Se for necessário incluir um novo atributo com o CPF em todas as classes?**

Resposta: Teremos que alterar todas as classes, dificultando a manutenção.

**Se for necessário um novo atributo para armazenar a data de admissão para Gerentes e Funcionários?**


Resposta: Será necessário alterar as classes Gerente e Funcionário.

**Se for necessário incluir a data da primeira compra do cliente?**

Resposta: Será necessário alterar apenas a classe Cliente.

Vamos guardar estas perguntas e repeti-las após a aplicação da herança.

 Aplicando os conceitos de herança

 Clique no botão acima.

## Aplicando os conceitos de herança:

Classe Pessoa com o uso do conceito de herança (SuperClasse):

```
import java.util.Scanner;
public class Pessoa {
    String identidade, nome;
    public String getIdentidade() {
        return identidade;
    }
    public void setIdentidade(String id) {
        if (!id.isEmpty()) {
            identidade = id;
        }
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String no) {
        if (!no.isEmpty()) {
            nome = no;
        }
    }
}
public Pessoa() { }
public Pessoa(String id) {
    setIdentidade(id);
}
public Pessoa(String id, String no) {
    setIdentidade(id);
    setNome(no);
} // Só podemos criar 3 construtores para a classe Pessoa
public void cadastrar(String id, String no) {
    setIdentidade(id);
    setNome(no);
}
public void entradaDados() {
    Scanner entrada = new Scanner(System.in);
    System.out.println("Identidade :");
    setIdentidade(entrada.nextLine());
    System.out.println("Nome :");
    setNome(entrada.nextLine());
    entrada.close();
}
public void imprimir() {
    System.out.println("Identidade :" + getIdentidade());
    System.out.println("Nome :" + getNome());
}
}
```

Total de 44 linhas de código.

Classe PessoaEmpresa com o uso do conceito de herança (SubClasse de Pessoa):

```
import java.util.Scanner;
public class PessoaEmpresa extends Pessoa {
    String matricula;
    double salario;
    public String getMatricula() {
        return matricula;
    }
    public void setMatricula(String ma) {
        if (!ma.isEmpty()) {
            matricula = ma;
        }
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double sa) {
        if (sa >= 0) {
            salario = sa;
        }
    }
    public PessoaEmpresa() { }
    public PessoaEmpresa(String id) {
        super(id);
    }
    public PessoaEmpresa(double sa) {
        setSalario(sa);
    }
    public PessoaEmpresa(String id, double sa) {
        super(id);
        setSalario(sa);
    }
    public PessoaEmpresa(double sa, String id) {
        super(id);
        setSalario(sa);
    }
    public PessoaEmpresa(String id, String no, String ma, double sa) {
        super(id, no);
        setMatricula(ma);
        setSalario(sa);
    }
    public void cadastrar(String id, String no, String ma, double sa) {
        super.cadastrar(id, no);
        setMatricula(ma);
        setSalario(sa);
    }
    public void entradaDados() {
        Scanner entrada = new Scanner(System.in);
        super.entradaDados();
        System.out.println("Matricula :");
        setMatricula(entrada.nextLine());
        System.out.println("Salário :");
        setSalario(Double.parseDouble(entrada.nextLine()));
        entrada.close();
    }
    public void imprimir() {
        super.imprimir();
        System.out.println("Matricula :" + getMatricula());
        System.out.println("Salário :" + getSalario());
    }
}
```

Classe Gerente com o uso do conceito de herança (SubClasse de PessoaEmpresa):

```
import java.util.Scanner;
public class Gerente extends PessoaEmpresa {
    double percentualVenda;
    int nivelGerente;
    public double getPercentualVenda() {
        return percentualVenda;
    }
    public void setPercentualVenda(double pv) {
        if (pv >= 0) {
            percentualVenda = pv;
        }
    }
    public int getNivelGerente() {
        return nivelGerente;
    }
    public void setNivelGerente(int ng) {
        if (ng >= 0) {
            nivelGerente = ng;
        }
    }
    public Gerente() { }
    public Gerente(String id) {
        super(id);
    }
    public Gerente(double sa) {
        super(sa);
    }
    public Gerente(String id, double sa) {
        super(id, sa);
    }
    public Gerente(double sa, String id) {
        super(id, sa);
    }
    public Gerente(String id, String no, String ma, double sa, double pv, int ng) {
        super(id, no, ma, sa);
        setPercentualVenda(pv);
        setNivelGerente(ng);
    }
    public void cadastrar(String id, String no, String ma, double sa, double pv, int ng) {
        super.cadastrar(id, no, ma, sa);
        setPercentualVenda(pv);
        setNivelGerente(ng);
    }
    public void entradaDados() {
        Scanner entrada = new Scanner(System.in);
        super.entradaDados();
        System.out.println("Percentual de Venda:");
        setPercentualVenda(Double.parseDouble(entrada.nextLine()));
        System.out.println("Nível Gerencia :");
        setNivelGerente(Integer.parseInt(entrada.nextLine()));
        entrada.close();
    }
    public void imprimir() {
```

```

        super.imprimir();
        System.out.println("Percentual de Venda:" + getPercentualVenda());
        System.out.println("Nível Gerencia :" + getNivelGerente());
    }
}

```

Total de 58 linhas de código.

Classe Funcionário com o uso do conceito de herança (SubClasse de PessoaEmpresa):

```

import java.util.Scanner;
public class Funcionario extends PessoaEmpresa {
    String setor, nomeGerente;
    public String getSetor() {
        return setor;
    }
    public void setSetor(String se) {
        if (!se.isEmpty()) {
            setor = se;
        }
    }
    public String getNomeGerente() {
        return nomeGerente;
    }
    public void setNomeGerente(String ng) {
        if (!ng.isEmpty()) {
            nomeGerente = ng;
        }
    }
    public Funcionario() { }
    public Funcionario(String id) {
        super(id);
    }
    public Funcionario(double sa) {
        super(sa);
    }
    public Funcionario(String id, double sa) {
        super(id, sa);
    }
    public Funcionario(double sa, String id) {
        super(id, sa);
    }
    public Funcionario(String id, String no, String ma, double sa, String se, String ng) {
        super(id, no, ma, sa);
        setSetor(se);
        setNomeGerente(ng);
    }
    public void cadastrar(String id, String no, String ma, double sa, String se, String ng) {
        super.cadastrar(id, no, ma, sa);
        setSetor(se);
        setNomeGerente(ng);
    }
    public void entradaDados() {
        Scanner entrada = new Scanner(System.in);
        super.entradaDados();
        System.out.println("Setor :");
        setSetor(entrada.nextLine());
    }
}

```

```

        System.out.println("Nome Gerente:");
        setNomeGerente(entrada.nextLine());
    }

    public void imprimir() {
        super.imprimir();
        System.out.println("Setor :" + getSetor());
        System.out.println("Nome Gerente:" + getNomeGerente());
    }
}

```

Total de 56 linhas de código.

Classe Cliente com o uso do conceito de herança (SubClasse de Pessoa):

```

import java.util.Scanner;
public class Cliente extends Pessoa {
    String codigoCliente, telefone;
    int idade;
    public String getCodigoCliente() {
        return codigoCliente;
    }
    public void setCodigoCliente(String cc) {
        if (!cc.isEmpty()) {
            codigoCliente = cc;
        }
    }
    public String getTelefone() {
        return telefone;
    }
    public void setTelefone(String tf) {
        if (!tf.isEmpty()) {
            telefone = tf;
        }
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int id) {
        if (id >= 0) {
            idade = id;
        }
    }
    public Cliente() { }
    public Cliente(String id) {
        super(id);
    }
    public Cliente(int id) {
        setIdade(id);
    }
    public Cliente(String id, int ida) {
        super(id);
        setIdade(ida);
    }
    public Cliente(int ida, String id) {
        super(id);
        setIdade(ida);
    }
}

```



```
public Cliente(String id, String no, String cc, String tf, int ida) {
    super(id, no);
    setCodigoCliente(cc);
    setTelefone(tf);
    setIdade(ida);
}

public void cadastrar(String id, String no, String cc, String tf, int ida) {
    super.cadastrar(id, no);
    setCodigoCliente(cc);
    setTelefone(tf);
    setIdade(ida);
}

public void entradaDados() {
    Scanner entrada = new Scanner(System.in);
    super.entradaDados();
    System.out.println("Código do Cliente :");
    setCodigoCliente(entrada.nextLine());
    System.out.println("Telefone :");
    setTelefone(entrada.nextLine());
    System.out.println("Idade :");
    setIdade(Integer.parseInt(entrada.nextLine()));
    entrada.close();
}

public void imprimir() {
    super.imprimir();
    System.out.println("Código do Cliente :" + getCodigoCliente());
    System.out.println("Telefone :" + getTelefone());
    System.out.println("Idade :" + getIdade());
}
}
```

Total de 73 linhas de código.

Foram necessárias, para criar as três classes, um total de (44 + 60 + 58 + 56 + 73 =) 291 linhas de código e sem repetição de código. Como a versão inicial tem 315 linhas de código, economizamos 24 linhas inicialmente, mas o mais importante é a manutenção, como veremos a seguir. Vamos responder novamente às três perguntas feitas anteriormente:

### Se for necessário incluir um novo atributo com o CPF em todas as classes?

Resposta: Teremos que alterar apenas a classe Pessoa, uma vez que todas as demais classes irão herdar qualquer atualização nesta classe.

### Se for necessário um novo atributo para armazenar a data de admissão para Gerentes e Funcionários?

Resposta: Será necessário alterar apenas a classe PessoaEmpresa, já que as classes Gerente e Funcionário herdarão desta classe.

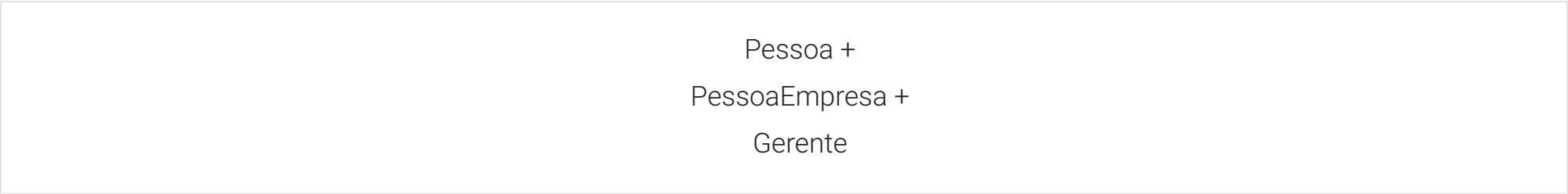
### Se for necessário incluir a data da primeira compra do cliente?

Resposta: Será necessário alterar apenas a classe Cliente.

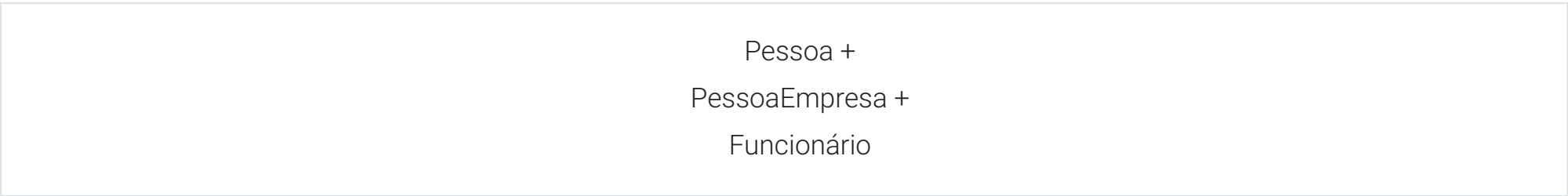
Atenção

Após analisarmos as duas soluções, chegamos à conclusão de que, ao utilizar a herança, não só evitamos a redundância de códigos (repetição) como facilitamos a manutenção, uma vez que, para realizar qualquer atualização, deveremos sempre alterar apenas uma das classes.

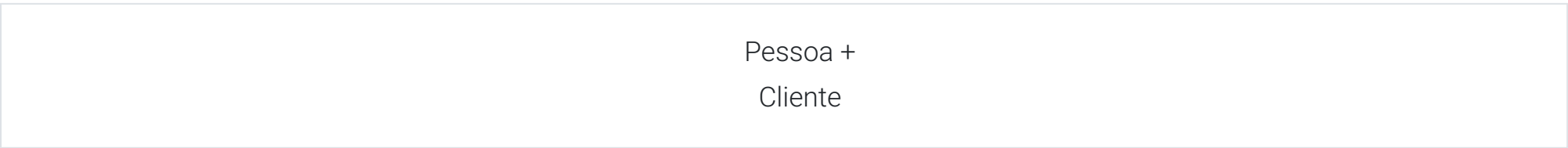
No final, a classe Gerente é composta por todos os membros de Pessoa, PessoaEmpresa e Gerente, uma vez que Gerente estende PessoaEmpresa, que por sua vez estende a classe Pessoa:



O mesmo ocorre com a classe Funcionário, composta por todos os membros de Pessoa, PessoaEmpresa e Funcionário, uma vez que Funcionário estende PessoaEmpresa, que por sua vez estende a classe Pessoa:



Já a classe Cliente é composta por todos os membros de Pessoa e Cliente, uma vez que Cliente estende a classe Pessoa:



# Herança de métodos construtores

---

Vamos analisar os métodos construtores da classe Cliente:

```
public Cliente() { }
public Cliente(String id) {
    super(id);
}
public Cliente(int id) {
    setIdade(id);
}
public Cliente(String id, int ida) {
    super(id);
    setIdade(ida);
}
public Cliente(int ida, String id) {
    super(id);
    setIdade(ida);
}
public Cliente(String id, String no, String cc, String tf, int ida) {
    super(id, no);
    setCodigoCliente(cc);
    setTelefone(tf);
    setIdade(ida);
}
```

Alguns destes métodos repassam os parâmetros recebidos para a Superclasse, através da palavra reservada `super`. Esta instrução diz ao compilador que o(s) parâmetro(s) será(ão) repassado(s) a um método construtor com a mesma assinatura na SuperClasse.

Assim, o método:

```
public Cliente(String id) {
    super(id);
}
```

---

`super` → Repassará o parâmetro `id` para o método:

---

```
public Pessoa(String id) {
    setIdentidade(id);
}
```

O método:

```
public Cliente( String id, int ida ) {
    super ( id );
    setIdade( ida );
}
```

---

`id` → Repassará o parâmetro `id` para o mesmo método construtor usado no anterior:

---

```
public Pessoa( String id ) {
    setIdentidade ( id );
}
```

```
}
```

Por final o método:

```
public Cliente(String id, String no, String cc, String tf, int ida) {  
    super(id, no);  
    setCodigoCliente(cc);  
    setTelefone(tf);  
    setIdade(ida);  
}
```

---

**id** → Repassará os parâmetros **id** e **no** para o método construtor da SuperClasse com a assinatura equivalente a estes dois parâmetros, que será o método:

---

```
public Pessoa(String id, String no) {  
    setIdentidade(id);  
    setNome(no);  
}
```

Todo método construtor de uma SubClasse deve referenciar um construtor da SuperClasse, isso quer dizer que para o construtor vazio: `public Cliente() {}`, será obrigatório que exista um construtor vazio na SuperClasse: `public Pessoa() {}`.

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

# Sobrescrita de métodos

---

Métodos de uma SuperClasse podem ser sobrescritos em suas subclasses, implicando que um método descrito na Superclasse poderá ser substituído na Subclasse. Para isso, é importante observar que estes métodos devem possuir as mesmas assinaturas. Caso contrário, será usado o conceito de Sobrecarga e não de Sobrescrita.

Vamos analisar o método imprimir da Superclasse Pessoa:

```
public void imprimir() {  
    System.out.println("Identidade :" + getIdentidade());  
    System.out.println("Nome :" + getNome());  
}
```

Ele não é suficiente para atender às necessidades da classe Cliente, então poderíamos tê-lo substituído por:

```
public void imprimir() {  
    System.out.println("Identidade :" + getIdentidade());  
    System.out.println("Nome :" + getNome());  
    System.out.println("Código do Cliente :" + getCodigoCliente());  
    System.out.println("Telefone :" + getTelefone());  
    System.out.println("Idade :" + getIdade());  
}
```

Esta substituição permitiria que o método imprimir fosse substituído na Subclasse por um método mais completo que atendesse a sua necessidade.

Note que as assinaturas dos métodos, tanto na Superclasse Pessoa como na Subclasse Cliente, são idênticas: imprimir(); desta forma, houve uma Sobrescrita (*Override*) e não uma Sobrecarga (*Overhead*).

Como uma primeira forma de atender a demanda da classe Cliente, o método imprimir nesta nova versão seria suficiente, mas podemos melhorar nossa solução, observe que parte do código já existe no método imprimir da Superclasse Pessoa, ocorrendo uma redundância de código nas instruções destacadas na cor vermelha:

```
public void imprimir() {  
    System.out.println("Identidade :" + getIdentidade());  
    System.out.println("Nome :" + getNome());  
    System.out.println("Código do Cliente :" + getCodigoCliente());  
    System.out.println("Telefone :" + getTelefone());  
    System.out.println("Idade :" + getIdade());  
}
```

Note que as instruções na cor vermelha já existem no método imprimir da Superclasse e, assim, podemos reaproveitar os códigos da Superclasse ao chamar o método imprimir da Superclasse na Subclasse:

```
public void imprimir() {  
    super.imprimir();  
    System.out.println("Código do Cliente :" + getCodigoCliente());  
    System.out.println("Telefone :" + getTelefone());  
    System.out.println("Idade :" + getIdade());  
}
```

Ao reaproveitar o método imprimir da Superclasse, temos dois ganhos muito importantes:

- 01** Não haverá redundância de códigos, as instruções do método imprimir da Superclasse serão reaproveitadas na Subclasse;
- 02** 2. Caso haja a necessidade de incluir um novo atributo no método imprimir, só precisaremos realizar a alteração em apenas uma classe, pois, se for um atributo específico da classe Cliente, só precisaremos incluir a instrução no método da classe Cliente. Caso contrário, se o atributo for comum às demais classes, a instrução deverá ser incluída apenas na Superclasse Pessoa.

**A herança é um conceito importantíssimo da programação orientada a Objetos, permitindo que reaproveitemos membros Superclasse, que serão herdados pelas Subclasses, evitando redundância de códigos, além de facilitar a manutenção das nossas classes, uma vez que qualquer necessidade de mudança implicará na alteração de apenas uma classe.**

**Atenção!** Aqui existe uma videoaula, acesso pelo conteúdo online

## Atividades

---

1) [INAZ do Pará - 2019 - CORE-SP - Analista de T.I]: “O desenvolvimento de software é extremamente amplo. Nesse mercado, existem diversas linguagens de programação, que seguem diferentes paradigmas. Um desses paradigmas é a Orientação a Objetos, que atualmente é o mais difundido entre todos. Isso acontece porque se trata de um padrão que tem evoluído muito, principalmente em questões voltadas para segurança e reaproveitamento de código, o que é muito importante no desenvolvimento de qualquer aplicação moderna”.

Disponível em [aqui](#). Acesso em: 17.11.2018

Considere o programa abaixo escrito na linguagem Java:

```
public class veiculo {}  
public class carro extends veiculo {}  
public class aviao extends veiculo {}
```

Qual a afirmativa correta?

- a) A classe veículo é subclasse da classe avião.
  - b) A classe avião é subclasse da classe carro.
  - c) A classe veículo é superclasse das classes carro e avião.
  - d) As classes carro e avião são superclasses da classe veículo.
  - e) As classes veículo e carro são subclasses da classe máquinas.
-

2) [PosComp 2015]: Considere o seguinte código desenvolvido em Java.

```
public class Animal {
    int numeroPatas;
    public void fale() { };
}
public class Cao extends Animal {
    public void fale() {
        System.out.println("au au");
    }
}
public class Gato extends Animal {
    public void fale() {
        System.out.println("miau");
    }
}
public class GatoPersa extends Gato {
    public void fale() {
        System.out.println("miauuuu");
    }
}
public class Tigre extends Gato {
    public void fale() {
        super.fale();
        System.out.println("rrrrrr");
    }
}
public class Principal {
    public static void main(String[] args) {
        Gato gato = new GatoPersa();
        gato.fale();
        Cao cao = new Cao();
        cao.fale();
        Tigre tigre = new Tigre();
        tigre.fale();
    }
}
```

- |            |             |         |         |         |
|------------|-------------|---------|---------|---------|
| a) miauuuu | b) miauuuuu | c) miau | d) miau | e) miau |
| au au      | au au       | au au   | au au   | au au   |
| miau       | rrrrrr      | miau    | rrrrrr  | miau    |
| rrrrrr     |             | miau    |         | rrrrrr  |

Notas

Referências

DEITEL, Paul. **Java**: como programar (Biblioteca Virtual). 10. ed. São Paulo: Pearson, 2017.

FURGERI, Sérgio. **Java 8** – ensino didático: desenvolvimento e implementação de aplicações. São Paulo: Érica, 2015.



- 
- Agregação;
  - Particionamento.

## Explore mais

---

Leia o texto:

- [Entendendo e Aplicando Herança em Java.](#)