Herança
LAB - 7

Herança, Encapsulamento e Polimorfismo são um dos conceitos mais importantes em **Programação Orientada a Objetos**. Basicamente herança permite o reuso de código, já que uma **subclasse** (classe filha) pode usar as propriedades e métodos definidos na superclasse (classe pai).

Duração prevista: 60 minutos

Exercícios

Exercício 1: Criar uma classe pai e sua classe filha.

Exercício 2: Invocar construtores com palavra chave super e this.

Exercício 3: Utilizando o Polimorfismo

Exercício 4: Utilizando a palavra reservada final em classes e métodos

Herança

1 – Crie a classe **Pessoa** conforme exemplo abaixo.
Está será a superclasse para todos os tipos de pessoas que teremos em nosso projeto.


```
public class Pessoa {

	private String nome;
	private String telefone;
	private String endereco;

	public Pessoa() {}

	public Pessoa( String nome ) {

		this.nome = nome;
	}
	//get e set
	public void ImprimeNome() {

		System.out.println("O nome da pessoa é : " + nome);

	}
}
```

2 - Crie a classe **Cliente** conforme o exemplo abaixo.
Aqui estamos empregando o conceito de Herança, dizemos que um cliente é uma pessoa.
A classe Cliente herdará as propriedades e métodos definidos de sua superclasse Pessoa, porém, a classe Cliente está definindo alguns atributos próprios.

```
public class Cliente extends Pessoa{

	private String cpf;
	private String rg;

	public Cliente() {}

	public Cliente(String cpf) {

		this.cpf = cpf;

	}

	//get e set

}
```

3 - Crie a classe **TesteHeranca** conforme abaixo, compile e execute o programa.

```
public class TesteHeranca {

	public static void main(String[] args) {


		Pessoa pessoa = new Pessoa();
		pessoa.setNome("José da Silva");
		pessoa.ImprimeNome();


		Cliente cliente = new Cliente();
		cliente.setNome("Maria");
		cliente.setCpf("004.536.781-55");
		cliente.setTelefone("(62)9974-4750");
		cliente.ImprimeNome();

	}
}
```

Invocando o construtor da super classe

1 – Na classe **Cliente**, adicione o trecho de código abaixo.
Observe que neste construtor estamos invocando explicitamente o construtor sobrecarregado da classe **Pessoa** através do método super(), e estamos passando por parâmetro a variável nome.

```
// construtores
public Cliente(String nome, String cpf ) {

	super(nome);
	this.cpf = cpf;
}
```

2 – Na classe **Cliente** sobrescreva o método **ImprimeNome**, porém, adicione os atributos CPF e ENDERECO para serem impressos juntamente com o nome do cliente.

```
@Override
public void ImprimeNome() {

        System.out.println(" Nome do cliente: " + getNome() +
" Nº do CPF: " + cpf +
" Endereço :" + getEndereco());

}
```

Polimorfismo

1 – Na classe **TesteHeranca** adicione o trecho de código conforme abaixo.
Vamos utilizar o **Polimorfismo** na instanciação do objeto e na chamada de método.

```
Pessoa pessoaCliente = new Cliente("Ze", "777.777.777-77");
pessoaCliente.ImprimeNome();

//Erro de compilação
Cliente clientePessoa = (Cliente) pessoa;
pessoaCliente1.ImprimeNome();
```

java.lang.ClassCastException: Pessoa cannot be cast to Cliente.

Erro em tempo de execução que ocorre quando tentamos fazer um cast (coerção - conversão explícita) de uma classe para outra classe. A classe Pessoa não é do tipo Cliente.

Palavra chave final em classes e métodos

1 - Modifique a classe **Cliente**, adicione a palavra reservada **final** em sua declaração.

```
final public class Cliente extends Pessoa {...}
```

2 - Crie a classe **ClienteEspecial** conforme exemplo abaixo.

```
public class ClienteEspecial extends Cliente{...}
```

Você perceberá que a classe **ClienteEspecial** não compilará porque está tentando herdar de uma classe que foi atribuída como **final**. Se você retirar a declaração final da classe **Cliente,** a classe **ClienteEspecial** compilará sem erros.

Por favor, exclua a classe **ClienteEspecial** do projeto, este exercício é só para mostrar que não é possível herdar de uma classe atribuída como **final**.

3 – Agora iremos testar o comportamento de um método declarado como **final**. Na classe **Cliente**, para não permitir **override** (sobrescrita) do método **ImprimeNome(),** altere-o usando a palavra reservada **final** na definição do método.

```
@Override
public final void ImprimeNome() {

System.out.println(" Nome do cliente: " + getNome() + " Nº do CPF: "
                + cpf + " Endereço :" + getEndereco());
        }
```

4 - Crie a classe **ClienteTeste** conforme exemplo abaixo.

```
public class ClienteTeste extends Cliente {

        public ClienteTeste(String nome, String cpf){
                super(nome, cpf);
        }

@Override
public void ImprimeNome() {

        System.out.println(" Nome do cliente: " + getNome() + " Nº do CPF: "
                        + getCpf() + " Endereço :" + getEndereco());
}
}
```

Você perceberá que a classe **ClienteTeste** não compilará porque você está tentando sobrescrever um método que foi declarado na superclasse como **final**. Para que a classe **ClienteTeste** compile corretamente você pode remover a definição de final do método na superclasse **Cliente** ou apagar o método sobrescrito na **subclasse ClienteTeste**.

Mas esta classe deve ser excluída do projeto porque este exercício foi só para mostrar que não é possível reescrever um **método** declarado como **final**.