Lab 7 – Herança

**Herança** é um dos conceitos mais importantes em **Programação Orientada a Objeto** juntamente com **Encapsulamento** e **Polimorfismo**. Basicamente herança permite o reuso de código, já que uma **sub-classe** (classe filha) pode usar as propriedade e métodos definidos na super classe (classe pai).

O objetivo deste laboratório é permitir que você entenda os conceitos de herança em Java. Estaremos reutilizando muito dos exercícios anteriores, portanto classes citadas neste laboratório cuja listagem não esteja presente aqui é porque você já possui o código como resultado de outros laboratórios.

Sugerimos que estes exemplos sejam feitos com uso da IDE Eclipse.

## Duração prevista: 90 minutos

***Exercícios***

**Exercício 1**: Construindo programas que criam classes com herança (30 minutos) **Exercício 2**: Invocando construtores com palavra chave super (20 minutos) **Exercício 3**: Type Casting (20 minutos)

**Exercício 4**: Palavra chave final em classes e métodos (20 minutos)

# *Exercício 1 - Construindo programas que criam classes através*

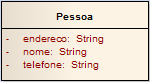
1. Neste exercício vamos construir uma classe **Pessoa.java,** onde Pessoa será a classe pai das demais classes que irá herdar suas características.

Figura 6.1 – Diagrama UML classe Pessoa

**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 e' : " + nome);

}

}

Listagem 6.1 – classe Pessoa.

1. Crie a classe **Cliente.java** abaixo, perceba que ela **extends** da classe **Pessoa.java**, isso porque

ele é uma **Pessoa**, mas possui algumas características distintas.

**public class** Cliente **extends** Pessoa{

**private** String cpf; **private** String rg; **public** Cliente() {

}

**public** Cliente(String cpf) {

**this**.cpf = cpf;

}

//get e set

}

Listagem 6.2 – classe Cliente herdando características de Pessoa.

1. Altere a classe **TesteHeranca.java** conforme abaixo, compile e execute o programa, veja o resultados.

**public class** TesteHeranca {

**public static void** main(String[] args) {

// cria uma pessoa

Pessoa pessoa = **new** Pessoa(); pessoa.setNome("Fulano de tal");

// Mostra toString() metodo sobrescrito na classe Cliente pessoa.ImprimeNome();

// cria um cliente

Cliente cliente = **new** Cliente();

cliente.setNome("Fulano de tal"); cliente.setCpf("999.999.999-99");

// Mostra toString() método sobrescrito de Pessoa dentro da classe Cliente cliente.ImprimeNome();

// método de acesso a atributo presente no objeto Pessoa cliente.setTelefone("(99)9999-9999");

// Mostra toString() metodo sobrescrito de Pessoa dentro da classe Cliente cliente.ImprimeNome();

}

}

Listagem 6.3 – classe TesteHeranca

# *Exercício 2 - Invocando construtores com palavra chave super*

1. Modifique a **Listagem-6.2**, adicionando o trecho de código abaixo para sobrepor o construtor da

superclasse **Pessoa.java** em **Cliente.java**, observe o uso do método **super()** para referenciar um construtor da classe **Pessoa.java** a ser invocado.

// construtores

**public** Cliente(String nome, String cpf ) {

**super**(nome);

**this**.cpf = cpf;

}

Listagem 6.4 – Usando super em cadeia de contrutores.

1. Modifique a **Listagem-6.2**, faça **override** do método **ImprimeNome()** de modo que deixe claro qual é instancia de Pessoa.

@Override

**public void** ImprimeNome() {

System.*out*.println("Nome do cliente e' : " + getNome() + " Nº CPF: " + cpf + " Seu endereco :" + getEndereco());

}

Listagem 6.5 – Override método transferir() classe ContaOuro.java

# *Exercício 3 - Type Casting e Polimorfismo*

1. Modifique a classe **TesteHeranca.java** conforme **Listagem-6.6** para testar a classe

**Polimorfismo** em tempo de execução e verificar a referência de instância de uma variável.

// referencia subclasse Cliente, upper casting

Pessoa pessoaCliente = **new** Cliente("Ze", "777.777.777-77");

// polimorfismo no tempo de execucao pessoaCliente.ImprimeNome();

Cliente pessoaCliente1 = (Cliente) pessoa; //Erro de compilação Pessoa não e tipo de Cliente pessoaCliente1.ImprimeNome();

Listagem 6.6 – TesteHeranca.java.

# *Exercício 4 - Palavra chave final em classes e métodos*

1. Modifique **Listagem-6.2** para não mais permitir herança da classe **Cliente.java**, usando palavra chave **final**.

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

1. Crie e compile nova classe **ClienteEspecial.java** abaixo para testar se não será mais permitido uma classe herdar da classe **Cliente.java**, consequentemente não sobrepondo o seu comportamento.

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

Você irá ver que a classe não compilará porque esta tentando herdar de uma classe que foi atribuída como **final**. Se você retirar a declaração final da classe **Cliente.java** a classe compilará sem erros, mas o correto é apagar esta classe **ClienteEspecial.java** do projeto, este exercício é só para mostrar que não é possível herdar de uma classe atribuída como **final**.

1. Modifique sua classe **Cliente.java** resultante do Exercício 2 item 2, para não permitir **override** do método **ImprimeNome()** usando palavra chave **final** na definição do método.

@Overridewww.3way.com.br

**public final void** ImprimeNome() {

System.*out*.println("Nome do cliente e' : " + getNome() + " Nº CPF: " + cpf + " Seu endereco :" + getEndereco());

}

Listagem 6.7 – método final

1. Crie e tente compilar classe **ClienteDosSonhos.java** abaixo.

**public class** ClienteDosSonhos **extends** Cliente {

**public** ClienteDosSonhos ( String nome, **int** cpf ) {

**super**(nome, cpf);

}

**public boolean** ImprimeNome() {

System.*out*.println("Esse e' o cliente dos sonhos. Seu nome e' : " + getNome() + " Nº CPF: " + cpf + " Seu endereco :" + getEndereco());

}

*}*

Listagem 6.8 – método final não pode ser override

Você irá perceber que a classe não compilará porque você esta tentando sobrepor um método que foi declarado na super classe como **final**, para que a classe compile corretamente você pode remover a definição de final do método na super classe ou apagar o método escrito na **sub-classe**. 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**.