

# LTP 1 – Java Básico

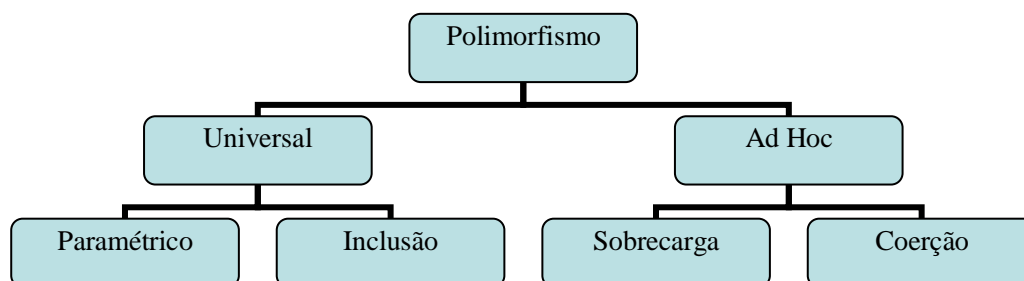
## Aula 9

### Tópicos

- 1 - Polimorfismo
  - 1.1 - Polimorfismo ad hoc por sobrecarga
  - 1.2 - Polimorfismo ad hoc por coerção
  - 1.3 - Polimorfismo universal paramétrico
  - 1.4 - Polimorfismo universal por inclusão sem redefinição de métodos
  - 1.5 - Polimorfismo universal por inclusão com redefinição de métodos
- 2 - Reescrevendo um método herdado
  - 2.1 - Invocando o método reescrito – usando super
- 3 - Exercícios
  - 3.1 - Conta Bancária
    - 3.1.1 Solução
  - 3.2 - Adicionando um novo método à Conta Bancária
    - 3.2.1 Solução
  - 3.3 - Subclasses da Conta
    - 3.3.1 Solução
  - 3.4 - Crie o método main
    - 3.4.1 Solução

### 1 - Polimorfismo

Polimorfismo é a capacidade de um objeto poder ser referenciado de várias formas. Apresentamos a seguir os tipos existentes.



#### 1.1 - Polimorfismo ad hoc por sobrecarga

Mudança dos parâmetros e, adicionalmente, do tipo de retorno do método, na mesma classe ou de classe base para classe derivada.

##### Exemplo:

```
public class NewClass {  
    int numMatricula;  
    int CPF;  
    public void aluno (int nMatr) {  
        numMatricula = nMatr;  
    }  
    public void aluno (int nMatr, int CPF){  
        numMatricula = nMatr;  
        this.CPF = CPF;  
    }  
}
```

## 1.2 - Polimorfismo ad hoc por coerção

A linguagem tem um mapeamento interno entre tipos. Forma limitada de polimorfismo. Se num particular contexto o tipo requerido é diferente do tipo dado, a linguagem verifica se existe uma coerção (i.e. conversão de tipos).

Em Java são executadas as seguintes conversões implicitamente :

- byte para short, int, long, float ou double
- short para int, long, float ou double
- char para int, long, float ou double
- int para long, float ou double
- long para float ou double
- float para double

### Exemplo:

```
package projetojava;
public class NewClass {
    public static void main(String args[ ]){
        ExemploCoercao obj = new ExemploCoercao();
        float x=5.0f;
        double y=2;
        int z = (int)y;
        int w = (int)x;
        System.out.println("w = "+w);
        System.out.println("z = "+z);
        obj.divideValores(w,z);
    }
}
```

---

```
package projetojava;
public class ExemploCoercao {
    public void divideValores(int a,int b){
        System.out.println(a+" / "+b+" = "+(a/b));
    }
}
```

## 1.3 - Polimorfismo universal paramétrico

Java não oferece um mecanismo generalizado para definição de métodos paramétricos. Um exemplo desse tipo de polimorfismo em Java, porém, é a definição de um array. O tipo array é um tipo pré-definido de Java, assim como os tipos int e char. Um objeto de tipo array possui um conjunto de operações características, por exemplo:

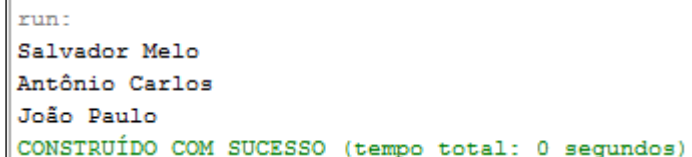
```
public class NewClass {
    public static void main(String args[]){
        int x[]= new int [11];
        System.out.println(x.length);
    }
}
```

O tipo array é declarado através do símbolo [ ], int funciona como parâmetro para a construção do array.

### 1.4 - Polimorfismo universal por inclusão sem redefinição de métodos

Polimorfismo de Inclusão é o estilo de polimorfismo encontrado em todas as linguagens orientadas a objetos. Ele está relacionado com a existência da hierarquia de generalização / especialização e com o conceito de subtipo. A noção de subtipo implica que elementos do subconjunto também pertencem ao superconjunto.

```
public class Carta extends Documento {  
  
}  
public class Telegrama extends Documento {  
  
}  
public class Documento {  
    public void mostra(String nome){  
        System.out.println(nome);  
    }  
}  
public class NewClass {  
    public static void main(String args[]){  
        Documento obj = new Documento();  
        obj.mostra("Salvador Melo");  
        obj = new Carta();  
        obj.mostra("Antônio Carlos");  
        obj = new Telegrama();  
        obj.mostra("João Paulo");  
    }  
}
```



```
run:  
Salvador Melo  
Antônio Carlos  
João Paulo  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

#### Outro exemplo:

```
public class Faculdade{  
    protected String nomeFaculdade = "Projeção";  
    public String getNomeFaculdade(){  
        return nomeFaculdade;  
    }  
}  
public class Curso extends Faculdade{  
    protected String nomeCurso = "Sistemas de Informação";  
    public String getNomeCurso(){  
        return nomeCurso;  
    }  
}  
public class Exemplo {  
    public static void main(String args[]){  
        Curso aux = new Curso();  
        Faculdade obj = aux;  
        System.out.println(obj.getNomeFaculdade());  
        System.out.println(aux.getNomeCurso());  
    }  
}
```

---

}

```
run:
Projeção
Sistemas de Informação
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## 1.5 - Polimorfismo universal por métodos com redefinição de métodos

Se uma classe herda um método que “não lhe serve” pode redefinir esse método. Ou seja, preserva a assinatura do método, mas muda a implementação da classe base para classe derivada.

### Exemplo:

```
public class IPVA {
    protected final double vlrBase = 1.50;
    protected int qtdCavalos;
    public double calcularIPVA () {
        return vlrBase * qtdCavalos;
    }
}

public class IPVACaminhao extends IPVA {
    private int qtdEixos;
    public double calcularIPVA() {
        return vlrBase * qtdCavalos * qtdEixos;
    }
}
```

---

Considere que no exemplo anterior vimos que todo Diretor é um Funcionario, pois é uma extensão deste. Lembrando que uma variável do tipo Funcionario é uma referência para um Funcionario, nunca o objeto em si.

```
class Funcionario {
    protected double salarioFuncionario;
    public double getBonificacaoFuncionario() {
        return this.salarioFuncionario * 0.10;
    }
    public void setSalarioFuncionario(double salarioFuncionario){
        this.salarioFuncionario = salarioFuncionario;
    }
}

class Diretor extends Funcionario {
    public double getBonificacaoFuncionario() {
        return this.salarioFuncionario * 0.10 + 1000;
    }
}

public class Principal {
    public static void main(String args[]){
        Diretor diretor = new Diretor();
        Funcionario funcionario = diretor;
        funcionario.setSalarioFuncionario(5000.0);
        System.out.println(funcionario.getBonificacaoFuncionario());
    }
}
```

---

## 2 - Reescrevendo um método herdado

Imagine uma estrutura de empresa poderíamos ter uma classe `Funcionario` e outra `Diretor`. Em todo momento que criarmos um objeto do tipo `Diretor`, este objeto possuirá também os atributos definidos na classe `Funcionario`, pois um `Diretor` é um `Funcionario`. Considere a seguinte situação, todo fim de ano, os funcionários da empresa recebem uma bonificação: os funcionários comuns recebem 10% do valor do salário e os gerentes 20%.

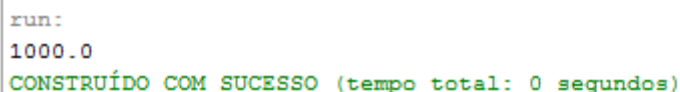
```
public class Funcionario {
    protected String nomeFuncionario;
    protected String cpfFuncionario;
    protected double salarioFuncionario;
    public double getBonificacaoFuncionario() {
        return this.salarioFuncionario * 0.10;
    }
    public void setSalarioFuncionario(double salarioFuncionario){
        this.salarioFuncionario = salarioFuncionario;
    }
}
```

Alguns funcionários são diretores e outros não. Porém, os diretores guardam outras informações, além das mesmas informações de um funcionário comum.

```
class Diretor extends Funcionario {
    int senha;
    int numeroDeFuncionariosGerenciados;
    public double getBonificacaoFuncionario() {
        return this.salarioFuncionario * 0.20;
    }
}
```

E na classe `Principal` teríamos:

```
public class Principal {
    public static void main(String args[]){
        Diretor obj = new Diretor();
        obj.setSalarioFuncionario(5000.0);
        System.out.println(obj.getBonificacaoFuncionario());
    }
}
```



```
run:
1000.0
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Ou seja, No Java, quando herdamos um método, podemos alterar seu comportamento. Podemos **reescrever** (reescrever, sobrescrever, *override*) este método.

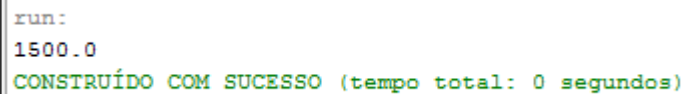
### 2.1 - Invocando o método reescrito – usando *super*

Depois de reescrito, não podemos mais chamar o método antigo que fora herdado da classe mãe. Imagine que para calcular a bonificação de um `Diretor` devemos fazer igual ao cálculo de um `Funcionario`, porém adicionando R\$ 1000. Poderíamos fazer assim:

```
class Diretor extends Funcionario {  
    public double getBonificacaoFuncionario() {  
        return this.salarioFuncionario * 0.10 + 1000;  
    }  
}
```

O problema que poderia ocorreria quando o `getBonificacaoFuncionario` mudar, precisaremos mudar o método da classe `Diretor` para acompanhar a nova bonificação. Para evitar isso, o `getBonificacaoFuncionario` do `Diretor` pode chamar o método da classe `Funcionario` utilizando a palavra chave `super`.

```
class Diretor extends Funcionario {  
    public double getBonificacaoFuncionario() {  
        return super.getBonificacaoFuncionario() + 1000;  
    }  
}
```



```
run:  
1500.0  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

## 3 - Exercícios

### 3.1 - Conta Bancária

Crie uma classe `Conta`, que possua um saldo os métodos para pegar saldo, depositar e sacar.

- Crie a classe `Conta`
- Adicione o atributo `saldo`
- Crie os métodos `getSaldo()`, `deposita(double)` e `saca(double)`

### 3.2 - Adicionando um novo método à Conta Bancária

- Adicione um método na classe `Conta`, que atualiza o saldo dessa conta de acordo com uma taxa percentual fornecida.

### 3.3 - Subclasses da Conta

Crie duas subclasses da classe `Conta`: `ContaCorrente` e `ContaPoupanca`. Ambas terão o método `atualiza` reescrito: A `ContaCorrente` deve atualizar-se com o dobro da taxa e a `ContaPoupanca` deve atualizar-se com o triplo da taxa.

Além disso, a `ContaCorrente` deve reescrever o método `deposita`, a fim de retirar uma taxa bancária de dez centavos de cada depósito.

- Crie as classes `ContaCorrente` e `ContaPoupanca`. Ambas são filhas da classe `Conta`
- Reescreva o método `atualiza` na classe `ContaCorrente`, seguindo o enunciado
  - Repare que, para acessar o atributo `saldo` herdado da classe `Conta`, **você vai precisar trocar o modificador de visibilidade de `saldo` para `protected`.**
- Reescreva o método `atualiza` na classe `ContaPoupanca`, seguindo o enunciado
- Na classe `ContaCorrente`, reescreva o método `deposita` para descontar a taxa bancária de dez centavos

### 3.4 - Crie o método main

Crie uma classe com método `main` e instancie essas classes, atualize-as e veja o resultado.

- Instancie um objeto para a classe `Conta`
- Instancie um objeto para a classe `ContaCorrente`

- Instancie um objeto para a classe ContaPoupanca
- Chame os métodos deposita de cada uma das classes, passando o valor 1000 para cada uma, por exemplo
- Chame os métodos atualiza de cada uma das classes, passando o valor 0.01 para cada uma, por exemplo
- Mostre o saldo a partir de cada uma das classes.