

TI-220 Java Orientado a Objetos

ANTONIO CARVALHO - TREINAMENTOS


A solid blue horizontal bar spanning the width of the slide, located at the bottom.

Polimorfismo

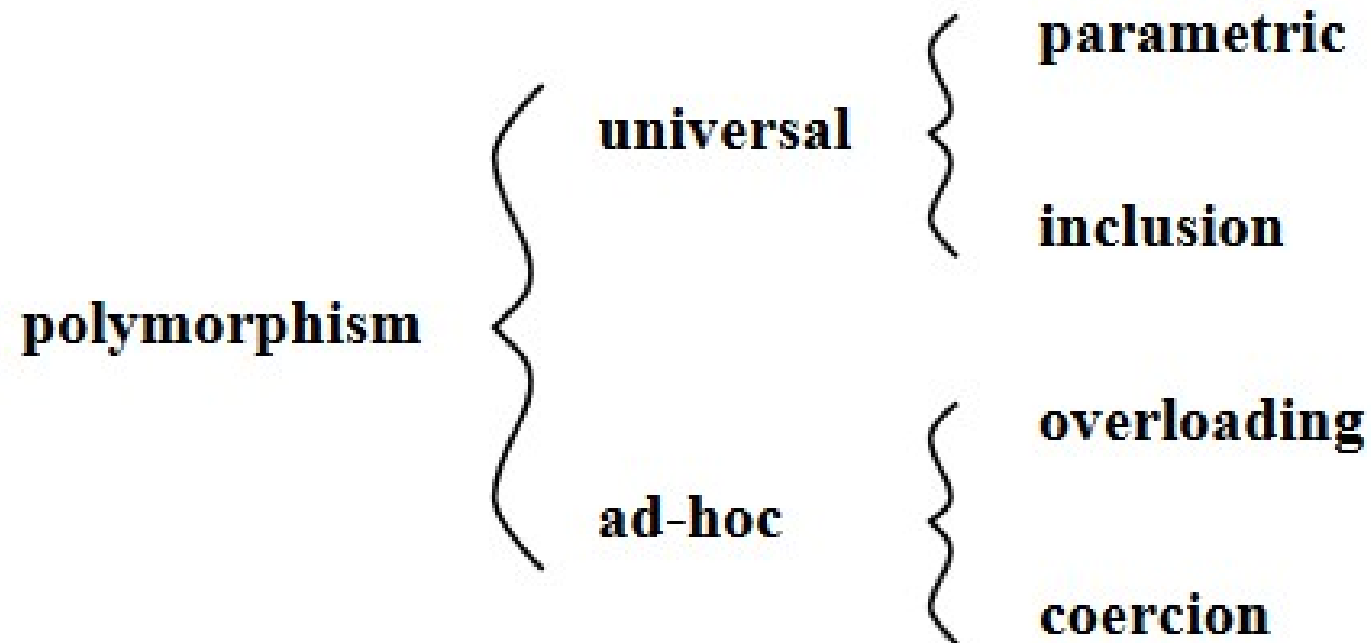
Definição

Polimorfismo, do grego: muitas formas.

Polimorfismo pode ser definido como:

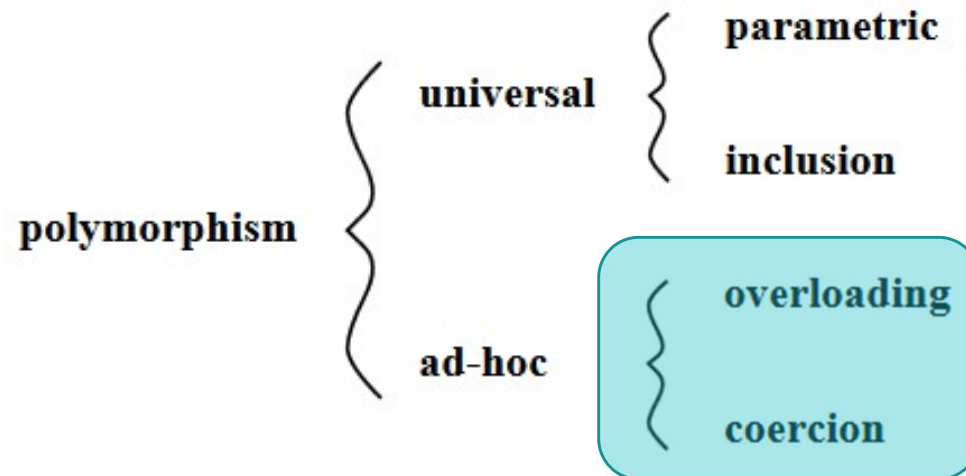
- Capacidade de um operador executar a ação apropriada dependendo do tipo do operando.
 - Capacidade de uma referência de se associar a instâncias de diversos tipos
- 

Tipo de polimorfismo



Ad-hoc

É o tipo de polimorfismo que ocorre um número limitado de vezes, sendo produzido em tempo de compilação.



Ad-hoc - Coerção

A coerção ocorre quando um tipo é transformado em outro tipo devido a operação utilizada.

No Java uma variável de um tipo pode ser transformada em outro tipo para viabilizar uma operação.

Exemplos:

```
int a = 10;  
String c = "número : " + a;
```

```
int a = 10;  
float b = 5.7f;  
float c = a + b;
```

No exemplo da esquerda a variável **(a)** é convertida para String para realizar a concatenação, e no exemplo da direita a variável **(a)** é convertida para **float**

Ad-hoc - Sobrecarga (Overload)

A sobrecarga permite a criação de métodos com o mesmo nome porém com parâmetros diferentes, de modo que serão invocados conforme o tipo e a quantidade de parâmetros informados.

Ad-hoc - Sobrecarga (Overload)

Exemplo :

```
int somaNumeros( int numero1, int numero2 ) {  
    System.out.println("Método 1");  
    int soma = numero1 + numero2;  
    return (soma)  
}
```

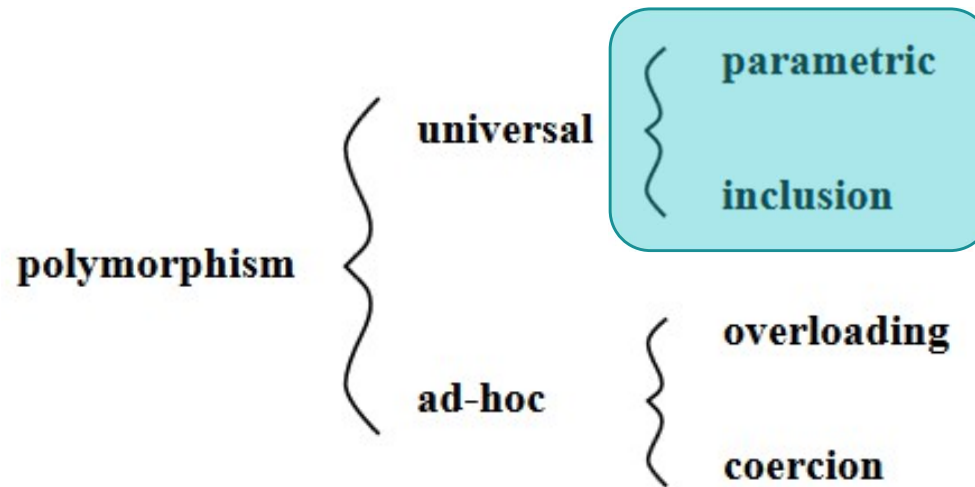
```
int somaNumeros( int numero1, int numero2, int numero3 ) {  
    System.out.println("Método 2");  
    int soma = numero1 + numero2 + numero3;  
    return (soma)  
}
```


Ad-hoc - Sobrecarga (Overload)

Ao ser invocado o método somaNumeros com dois números inteiros, então será invocado o **método 1**, caso a quantidade de parâmetros seja 3, então o método invocado será o **método 2**

Universal

Neste tipo de polimorfismo o código a ser executado é definido em tempo de execução, através de uma ligação tardia.



Universal

Como exemplo vamos tomar uma classe **A** que herda de **B**, sendo que a classe **B** tem um método **foo()**.

Ao invocar o método **foo()** de uma instância da classe **A**, a JVM verifica em tempo de execução se a classe **A** possui uma versão própria do método **foo()**, executando-o.


Caso contrário o método **foo()** da classe **B** será executado.



Inclusion - Sobrescrita (Overriding)

A sobrescrita permite que uma subclasse modifique o comportamento herdado de uma super classe.

Dessa forma uma subclasse (classe mais específica) pode conter um comportamento diferenciado daquele que foi herdado.

A solid blue horizontal bar spanning the width of the slide, located at the bottom.

Inclusion - Sobrescrita (Overriding)

Exemplo de sobrescrita:

```
public class Pessoa {  
    public void comer() {  
        System.out.println("Pessoa comendo");  
    }  
    public void andar( int qtdPassos) {  
        System.out.println("Pessoa andando " + qtdPassos + " passos");  
    }  
}
```

```
public class Atleta extends Pessoa {  
    public void comer() {  
        System.out.println("Atleta comendo");  
    }  
    public void andar( int qtdPassos ) {  
        System.out.println("Atleta andando " + qtdPassos +  
            " passos rapidamente");  
    }  
}
```

Inclusion - Sobrescrita (Overriding)

No exemplo anterior a classe **Atleta** herda da classe **Pessoa** e modifica os comportamentos **comer** e **andar**, de forma que o Atleta execute estes comportamentos de maneira diferente.

O comportamento é submetido ao polimorfismo e será acessado conforme o tipo da instância.

```
public class Campo {  
    public static void main(String[] args) {  
        Pessoa p1 = new Pessoa();  
        p1.comer(); // Mostrará Pessoa comendo  
        Atleta a = new Atleta();  
        a.comer(); // Mostrará Atleta comendo  
        Pessoa p2 = new Atleta();  
        p2.comer(); // Mostrará Atleta comendo  
        // Porque o método comer é invocado conforme a instância  
    }  
}
```

Inclusion - Sobrescrita (Overriding)

A sobrescrita tem algumas regras são elas:

- O tipo de retorno pode ser o mesmo tipo ou um subtipo dele
- Não pode haver redução de visibilidade
- Não pode lançar mais ou menos exceptions que o método que está sendo sobrescrito, como exemplo um método que lança **NullPointerException** não pode ser sobrescrito e lançar outra exception como **IOException**.
 - Não é porque o método está sendo sobrescrito que ele apresentará menos riscos à excessões.
- Métodos marcados com **final** ou **static** não podem ser sobrescritos
- Um método somente poderá ser sobrescrito, caso a subclasse consiga acessá-lo.

Inclusion - Sobrescrita (Overriding)

Exemplo das regras aplicadas

```
public class Pessoa {  
    public void comer() {  
        System.out.println("Pessoa comendo");  
    }  
    public void andar( int qtdPassos) {  
        System.out.println("Pessoa andando " + qtdPassos + " passos");  
    }  
    private void pensar() {  
        System.out.println("Pessoa pensando");  
    }  
}
```

```
public class Atleta extends Pessoa {  
    private void comer() { // Não compila pois reduz a visibilidade  
        System.out.println("Atleta comendo");  
    }  
    public void andar( int qtdPassos ) throws IOException {  
        // Não compila pois aumenta a quantidade de exceptions  
        System.out.println("Atleta andando " + qtdPassos +  
            " passos rapidamente");  
    }  
    private void pensar() {  
        // Não é sobrescrita pois o método pensar não é visível nesta classe  
        System.out.println("Atleta pensando");  
    }  
}
```


Parametric - Generics

A parametrização contrasta com o ad-hoc pois neste último existe um conjunto de métodos criados para trabalhar com tipos diferentes.

Já o paramétrico permite a criação de um único método genérico que pode ser utilizado para diversos tipos.

Exemplos:

```
// AD-HOC
class Mat {
    double soma(int a, int b) {
        return a + b;
    }
    double soma(float a, float b) {
        return a + b;
    }
}
```

```
// Parametric
class Mat<T extends Number> {
    double soma(T a, T b) {
        return a.doubleValue() + b.doubleValue();
    }
}
```

Dúvidas

