

Aula 10 – Polimorfismo

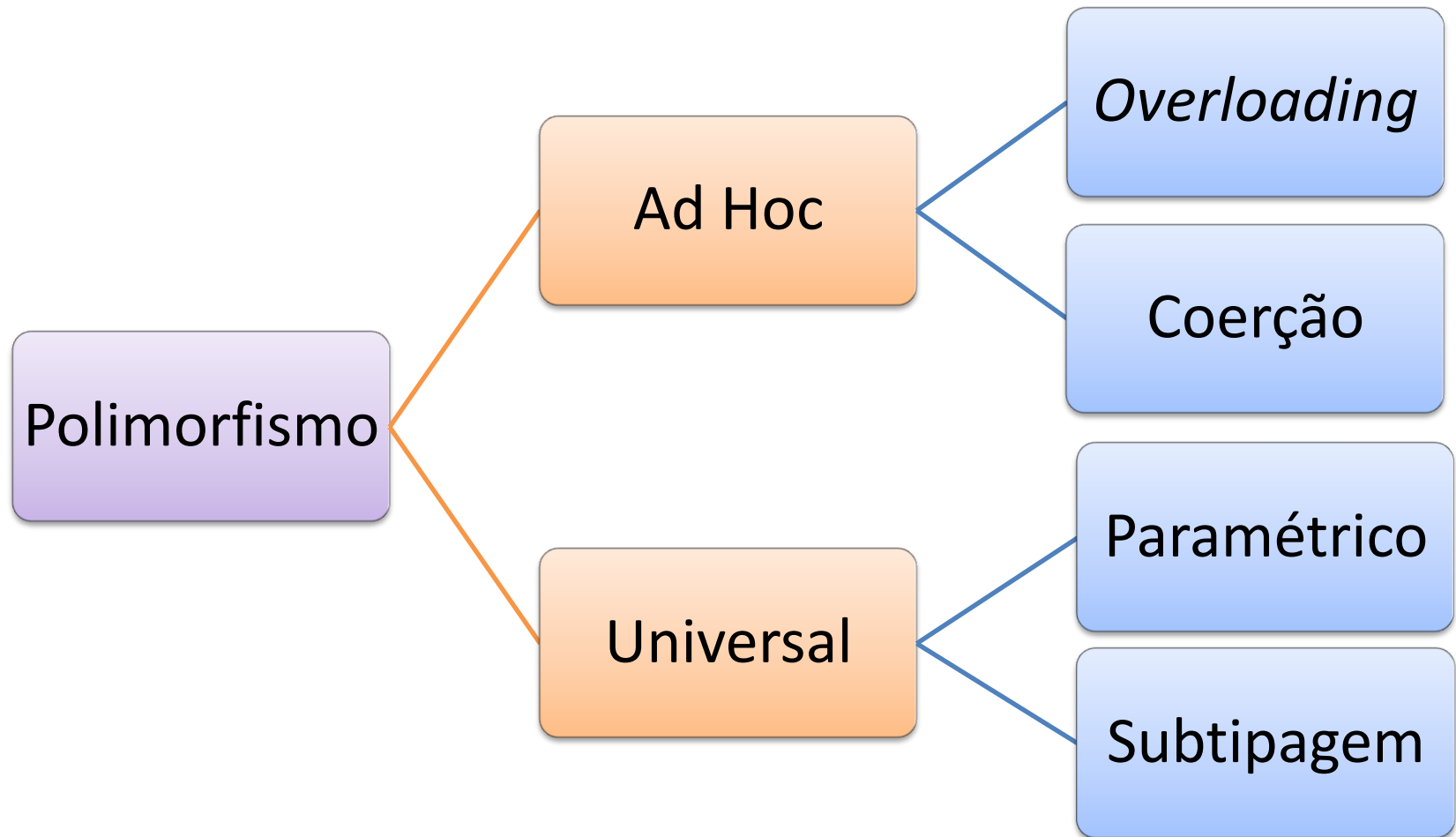
Prof. Jefersson Alex dos Santos

Baseado em: <http://homepages.dcc.ufmg.br/~fernando/classes/dcc052/ementa/lesson8/>

Polimorfismo

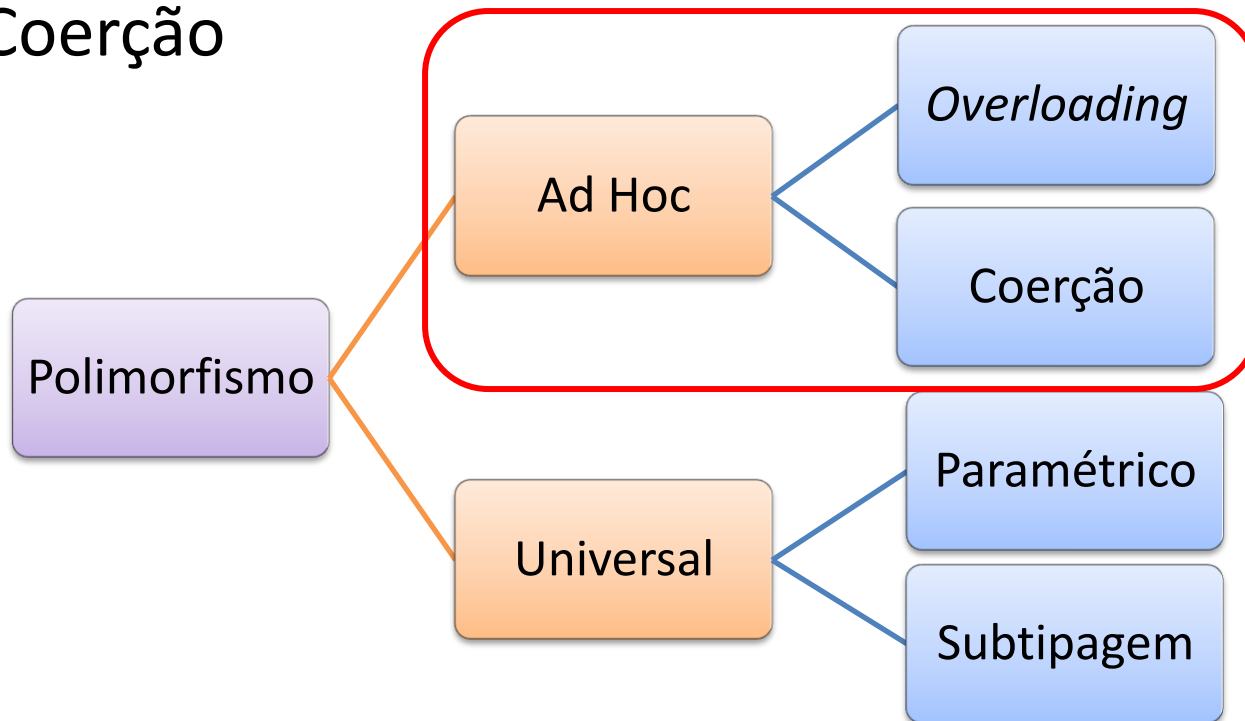
- Definição: É a capacidade de um objeto em decidir que método aplicar a si mesmo
- Termo originário do grego: "muitas formas" (*poli* = muitas, *morphos* = formas).
- Métodos/Funções:
 - São polimórficos se têm pelo menos dois tipos diferentes

Polimorfismo



Polimorfismo Ad-hoc

- Número finito de variações
 - Sobrecarga (*overloading*)
 - Coerção



Polimorfismo:

Sobrecarga (*overloading*)

- É resolvido estaticamente, em tempo de compilação
- Um exemplo?

Polimorfismo:

Sobrecarga (*overloading*)

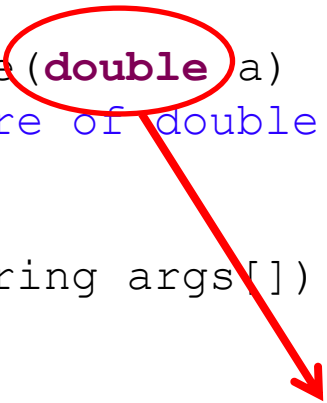
- Um exemplo?

```
public class Square {  
    public static int square(int a) {  
        System.out.println("Square of int: " + a);  
        return a * a;  
    }  
    public static double square(double a) {  
        System.out.println("Square of double: " + a);  
        return a * a;  
    }  
    public static void main(String args[]) {  
        square(1);  
        square(1.0);  
    }  
}
```

Polimorfismo: Sobrecarga (*overloading*)

- Um exemplo?

```
public class Square {  
    public static int square(int a) {  
        System.out.println("Square of int: " + a);  
        return a * a;  
    }  
    public static double square(double a) {  
        System.out.println("Square of double: " + a);  
        return a * a;  
    }  
    public static void main(String args[]) {  
        square(1);  
        square(1.0);  
    }  
}
```



E se trocarmos por **int** ?

Polimorfismo: Coerção de Tipo

- Quais chamadas são válidas?

```
public class Coercao {  
    public static void f(double x) {  
        System.out.println(x);  
    }  
    public static void main(String args[]) {  
        f(3.1416);  
        f((byte)1);  
        f((short)2);  
        f('a');  
        f(3);  
        f(4L);  
        f(5.6F);  
    }  
}
```


Resumo: Polimorfismo Ad-hoc

- Sobrecarga (*overloading*)
 - Utiliza os tipos para escolher a definição
- Coerção
 - Utiliza a definição para escolher o tipo de conversão

Resumo: Polimorfismo Ad-hoc

- Que função é chamada abaixo?

```
public class Square {  
  
    public static double square(double a) {  
        System.out.println("Square of double: " + a);  
        return a * a;  
    }  
  
    public static int square(int a) {  
        System.out.println("Square of int: " + a);  
        return a * a;  
    }  
  
    public static void main(String args[]) {  
        square(1);  
        square(1.0);  
        square('a');  
    }  
}
```

Resumo: Polimorfismo Ad-hoc

TIPO	TAMANHO
boolean	1 bit
byte	1 byte
short	2 bytes
char	2 bytes
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes

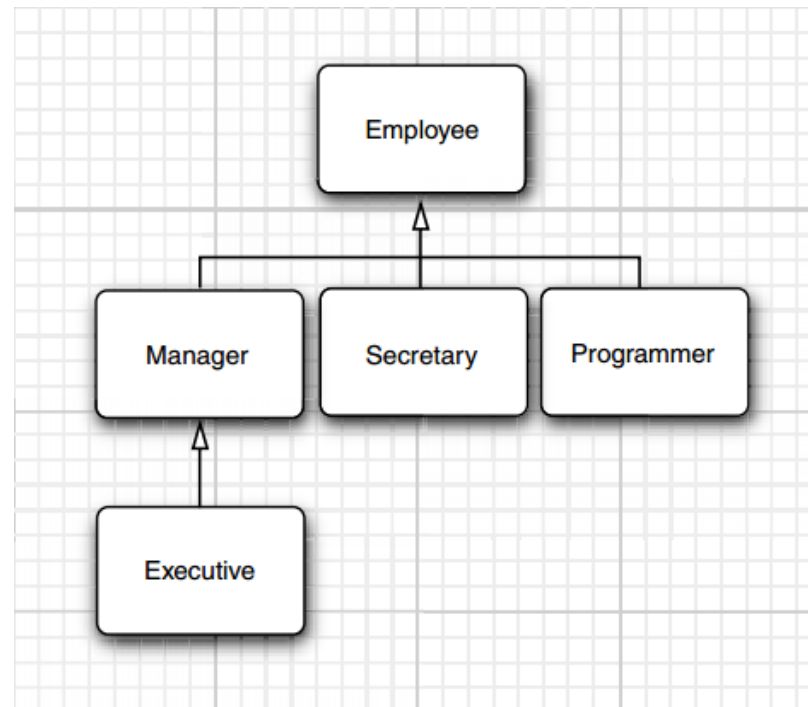
Resumo: Polimorfismo Ad-hoc

- E nesse caso?

```
public class Conflito {  
  
    public static void sum(int a, int b) {  
        System.out.println("Sum of int: " + (a + b));  
    }  
  
    public static void sum(double a, double b) {  
        System.out.println("Sum of double: " + (a + b));  
    }  
  
    public static void main(String args[]) {  
        sum(1, 2);  
        sum(1.1, 2.2);  
        sum(1, 2.2);  
        sum((int)1.1, (int)2.2);  
    }  
}
```

Polimorfismo Universal

- O que acontece quando a chamada de um método é feita a objetos de vários tipos de hierarquia?



Polimorfismo Universal

- O que acontece quando a chamada de um método é feita a objetos de vários tipos de hierarquia?
 - A subclasse verifica se ela tem ou não um método com esse nome e com os mesmos parâmetros
 - Senão tiver, a classe progenitora tornasse pelo responsável pelo processamento da mensagem

Polimorfismo Universal

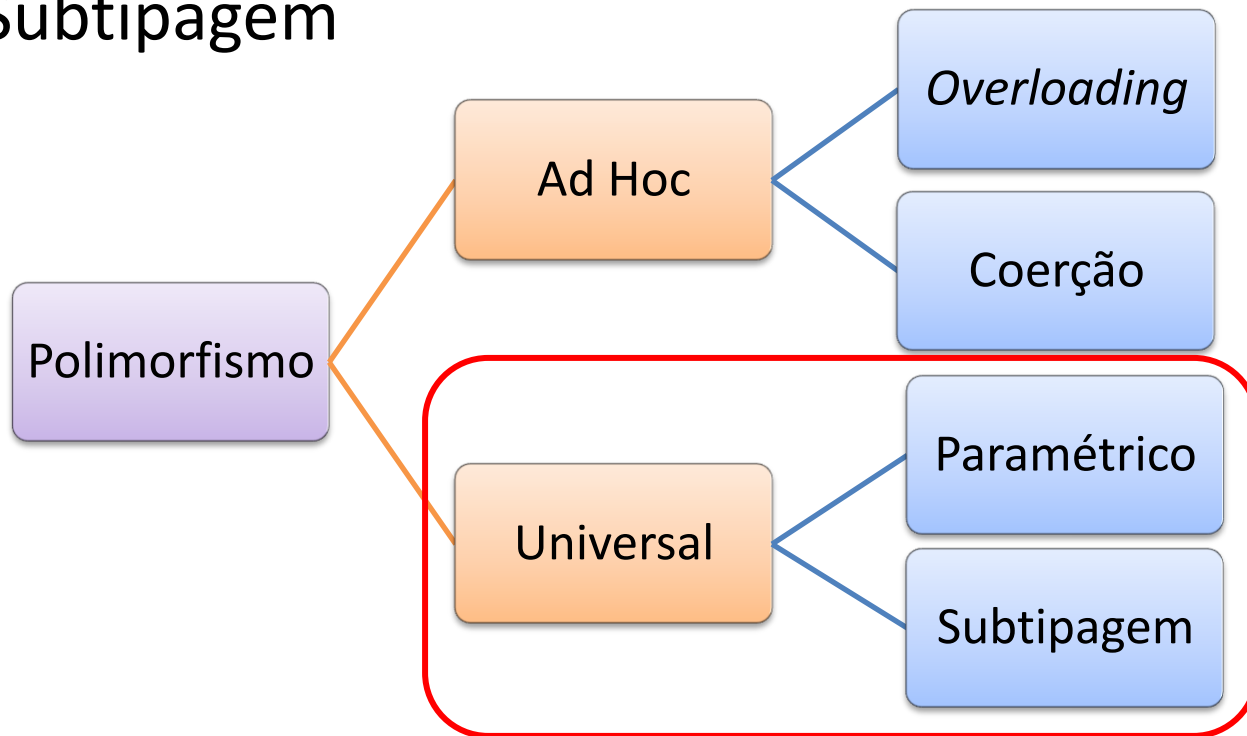
- Ligação tardia (*late binding*)
 - É a chave para o funcionamento
 - O compilador não gera o código em tempo de compilação
 - Ligação dinâmica:
 - Cada vez que se aplica um método a um objeto, o compilador gera código para calcular que método chamar

Polimorfismo Universal

- Princípio da substituição de Liskov:
 - se S é um subtipo de T , então os objetos do tipo T podem ser substituídos pelos objetos de tipo S sem que seja necessário alterar as propriedades deste programa

Polimorfismo Universal

- Número infinito de variações
 - Paramétrico
 - Subtipagem



Polimorfismo Paramétrico

- É uma forma de se tornar uma linguagem mais expressiva
- Mantém toda sua tipagem estática segura
- Foi introduzido em Java 1.5, como uma forma de reuso.
 - Este tipo de polimorfismo é chamado de *generics*.
 - São usados de forma similar a *templates* em C++

Polimorfismo Paramétrico

- Qual o problema desse código?

```
import java.util.List;
import java.util.LinkedList;

public class BadExample {
    public static void main(String args[]) {
        List myIntList = new LinkedList();
        myIntList.add(new Integer(0));
        Integer x = (Integer) myIntList.iterator().next();
    }
}
```

Note: BadExample.java uses unchecked or unsafe operations.

Note: Recompile with -Xlint:unchecked for details.

Polimorfismo Paramétrico

- Como melhorar?

```
import java.util.List;
import java.util.LinkedList;

public class BadExample {
    public static void main(String args[]) {
        List myIntList = new LinkedList();
        myIntList.add(new Integer(0));
        Integer x = (Integer) myIntList.iterator().next();
    }
}
```

Polimorfismo Paramétrico

- Como melhorar?

```
import java.util.List;
import java.util.LinkedList;

public class BadExample {
    public static void main(String args[]) {
        List<Integer> myIntList = new LinkedList<Integer>();
        myIntList.add(new Integer(0));
        Integer x = (Integer) myIntList.iterator().next();
    }
}
```

Definindo uma Classe Genérica

```
public interface Cont<E> {  
    E get();  
    void set(E e);  
}  
  
public class ContImpl<E> implements Cont<E> {  
    private E e;  
    public E get() { return e; }  
    public void set(E e) { this.e = e; }  
    public static void main(String args[]) {  
        ContImpl<Integer> c = new ContImpl<Integer>();  
        c.set(2);  
        System.out.println(c.get());  
    }  
}
```

Referências

- C. S. HORSTMANN, G. CORNELL. Core Java 2: Volume I – Fundamentos.
- DEITEL, H. M., DEITEL, P. J. Java: Como Programar. Pearson Brasil, 2005.
- BARNES, J. David, KÖLLING, Michael. Programação Orientada a Objetos com Java. Pearson, 2004.