

Herança e Sobreposição

Prof. Ms. Peter Jandl Junior

J12B

Linguagem de Programação Orientada a Objetos

Ciência da Computação - UNIP – Jundiaí

POO::herança & sobreposição

- Os conceitos essenciais da POO - Programação Orientada a Objetos:
 - Sobrecarga e Sobreposição,
 - Herança e
 - Polimorfismo.
- A rigor, o polimorfismo proporciona todos os mecanismos- chave da Orientação a Objetos.

Herança e Hierarquias

- Mecanismo que possibilita compartilhamento dinâmico de código.

Sobreposição

- Mecanismo que possibilita substituição de operações em subclasses.

Aplicação

- Como a herança e a sobrecarga podem ser explorados.

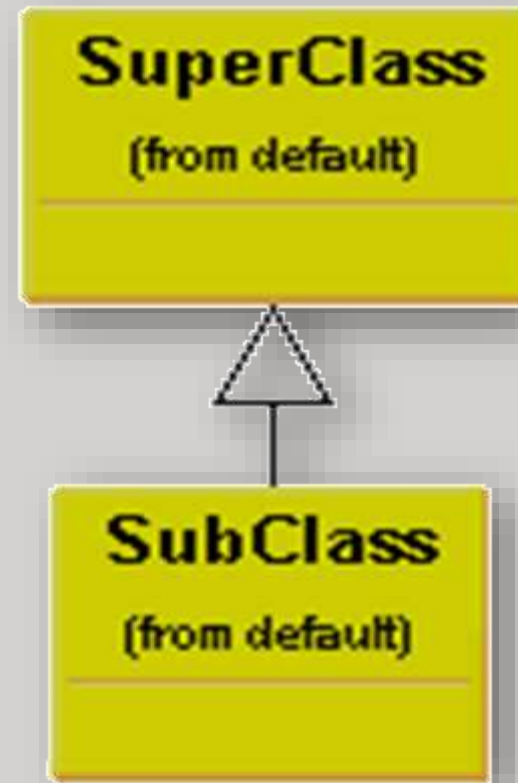
POO::herança

Herança

- ❑ Característica muito importante da Orientação a Objetos.
- ❑ Permite a construção de tipos de dados (classes) baseada em outros tipos já existentes, onde:
 - ❑ as características ancestrais são compartilhadas pelos descendentes e
 - ❑ novas características são adicionadas nestes.
- ❑ Possibilita a *especialização* das classes.

Herança

- A classe de origem é chamada de **classe base** ou **classe-pai** ou **superclasse**.
- A nova classe criada é chamada de **classe derivada** ou **classe-filha** ou **subclasse**.



Herança

```
// SuperClass.java
public class SuperClass {
    :
}

// SubClass.java
public class SubClass extends SuperClass {
    :
}
```

Herança: acessibilidade de membros

Especificador	Acessibilidade de membros			
	Implementação Superclasse	Instâncias Superclasse	Implementação SubClasse	Instâncias SubClasse
private	sim	não	não	não
protected	sim	não	sim	não
public	sim	sim	sim	sim

Membros protegidos constituem uma espécie de herança de programador, isto é, só está disponível para *implementações* de subclasses.

Herança

□ Por que usamos?

Porque simplifica e flexibiliza o projeto do software.

□ Para que usamos?

Para permitir reusabilidade do código e do projeto

□ Quando usamos?

Quando percebemos a existência de uma relação “é um” (*is a*)

Herança:aplicações

Extensão

- A Herança permite a criação de novas subclasses que ampliam as operações e atributos existentes na superclasse.

Restrição

- A Herança também permite que as funcionalidades da superclasse sejam alteradas em subclasses.

A **Herança** é um mecanismo de *especialização*, ou seja, permite a construção de novas classes derivadas que possuem características especiais em relação a classe base.

Hierarquias de Classes

O uso da herança para construção de famílias de classes



Herança::organização de hierarquias

- A construção de hierarquias envolve três etapas:
 1. Identificação das classes hierarquicamente relacionadas (relação "is a/é um").
 2. Fatoração dos elementos comuns nos níveis mais altos possíveis da hierarquia.
 3. Inclusão das especificidades (atributos e operações próprios) nos níveis mais baixos da hierarquia.

Herança::organização de hierarquias

- Hierarquias de classes são tipicamente representadas por árvores de classes onde:
 - raiz é posicionada acima, folhas para baixo (árvore invertida);
 - elementos mais próximos da raiz são ditos de hierarquia mais alta;
 - elementos mais próximos da folhas são ditos de hierarquia mais baixa;
 - o grau hierárquico se refere, portanto, ao número de filhos (classes derivadas) possíveis;
 - o grau hierárquico também se refere as possibilidades de generalização existentes (polimorfismo).

Herança & Hierarquias::um exemplo

- É comum a utilização de conversores de moedas e outras unidades em aplicações.
- Conversores lineares, os mais comuns, podem ser expressos como:
 - $\text{saida} = \text{entrada} * \text{kProp} + \text{kLin} = \text{entrada} * \alpha + \beta$
 - Onde: α é constante de proporcionalidade (kProp) e β é a constante linear (kLin).
- Um método, prototipado como segue, poderia realizar as conversões:
`double converter(double entrada);`

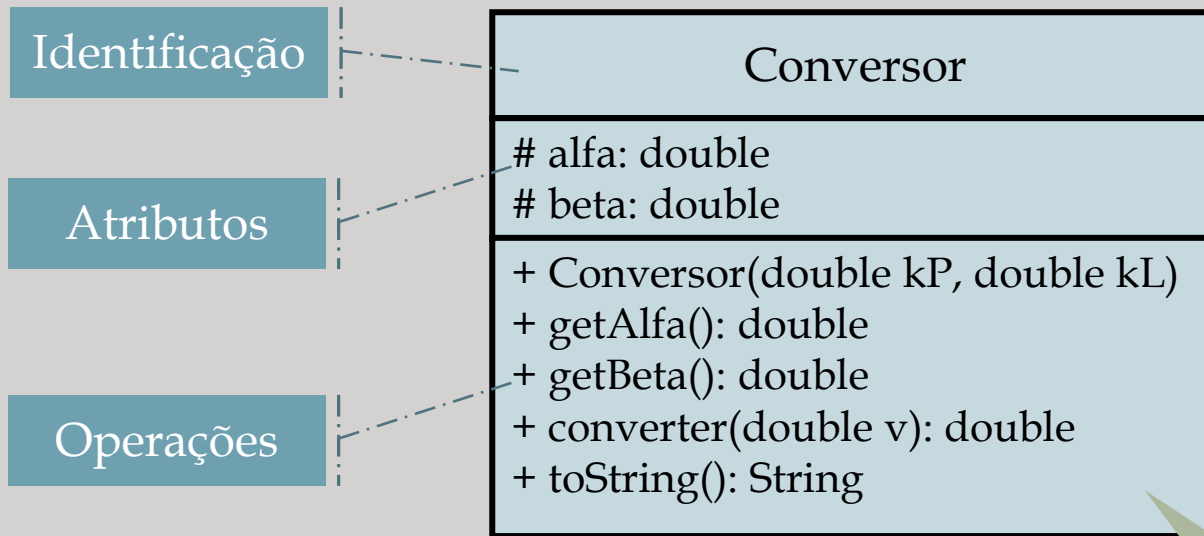
Herança & Hierarquias: um exemplo

```
public class Conversor {  
    // constantes  
    protected double alfa, beta;  
    // construtor  
    public Conversor (double alfa, double beta) {  
        this.alfa = alfa; this.beta = beta;  
    }  
    // métodos de acesso  
    public double getAlfa ( ) { return alfa; }  
    public double getBeta ( ) { return beta; }  
    // método de mutação  
    public double converter (double valor) {  
        return valor * alfa + beta;  
    }  
    // método de informação  
    public String toString ( ) {  
        return "Conversor[alfa=" + alfa + ", beta=" + beta + "];"  
    }  
}
```

Conversor::exemplos de uso

- Criar objetos Conversor exige 2 parâmetros tipo double
- `Conversor objeto = new Conversor (1.5 , 0.33);`
- `1.5 <-- objeto.getAlfa()`
- `0.33 <-- objeto.getBeta()`
- `? <-- objeto.converter(1.0)`
- `1,83 <-- objeto.converter(1.0)`
- `15.33 <-- objeto.converter(10.0)`
- `? <-- objeto.toString()`
- `"Conversor[alfa=1.5, beta=0.33]" <-- objeto.toString()`

Herança & Hierarquias: um exemplo



Representação de
classe UML

Herança & Hierarquias::um exemplo

- Um conversor de medidas em centímetros para polegadas pode ser definido e usado assim:

```
// instancia conversor com valores adequados
Conversor c2p = new Conversor(0.3937, 0.0);
double cm = 15;                // medida em cm
// conversão cm para pol
double pol = c2p.converter(cm);
// /Exibe: 15cm --> 5.9055pol
System.out.println(cm + "cm --> " + pol + "pol");
```

Herança & Hierarquias::um exemplo

- Outros conversores podem ser definidos com uso direto da classe **Conversor** e a correta parametrização de seu único construtor.
- O uso da herança permite aproveitar melhor as características da classe **Conversor**, ao mesmo tempo que facilita seu uso.
- Subclasses de **Conversor** poderiam encapsular mais facilmente a parametrização de conversões comuns (cm→pol, pol→cm, Celsius→Kelvin ou Kelvin→Celsius).

Herança & Hierarquias::um exemplo

CmPol.java

```
public class CmPol
    extends Conversor {

    public CmPol ( ) {

        // aciona o construtor
        // da superclasse
        super(0.3937, 0.0);

    }

}
```

PolCm.java

```
public class PolCm
    extends Conversor {

    public PolCm ( ) {

        // aciona o construtor
        // da superclasse
        super(2.54, 0.0);

    }

}
```

Representação de
classe UML

Herança & Hierarquias: um exemplo

Classe

Conversor

alfa: double
beta: double

+ Conversor(double kP, double kL)
+ getAlfa(): double
+ getBeta(): double
+ converter(double v): double
+ toString(): String

Representação de
classe UML

Subclasse

CmPol

+ CmPol()

Herança

PolCm

+ PolCm ()

Herança & Hierarquias::um exemplo

CelsiusKelvin.java

```
public class CelsiusKelvin
    extends Conversor {

    public CelsiusKelvin ( ) {

        // aciona o construtor
        // da superclasse
        super(1.0, 273.0);

    }

}
```

KelvinCelsius.java

```
public class KelvinCelsius
    extends Conversor {

    public KelvinCelsius ( ) {

        // aciona o construtor
        // da superclasse
        super(1.0, -273.0);

    }

}
```

Herança & Hierarquias::um exemplo

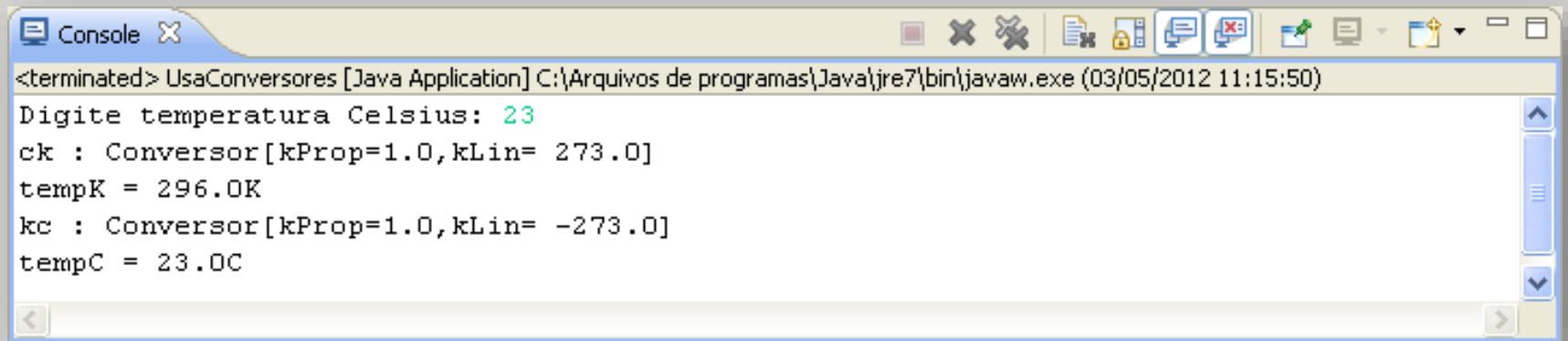
- A classe **UsaConversores**, que segue, realiza uma entrada, cria objetos conversores e efetua algumas conversões, exibindo os resultados.
- Os conversores são obtidos a partir dos construtores *default* disponíveis nas respectivas classes, os quais encapsulam as constantes necessárias, ou seja, sem necessidade de qualquer parametrização.
- A conversão sempre é realizada pelo método **converter(double)**, disponível na superclasse, que realiza a operação conforme os parâmetros específicos de cada conversor.

Herança & Hierarquias: um exemplo

```
UsaConversores.java X
1 import java.util.Scanner;
2
3 public class UsaConversores {
4     public static void main(String arg[]) {
5         Scanner sc = new Scanner(System.in);
6         // prepara console
7         System.out.print("Digite temperatura Celsius: ");
8         double tempC = sc.nextDouble();
9         // lê temp Celsius
10        // cria conversor Celsius p/ Kelvin e exibe info
11        CelsiusKelvin ck = new CelsiusKelvin();
12        System.out.println("ck : " + ck);
13        // uso implícito de toString()
14        double tempK = ck.converter(tempC);
15        // conversão C-->K
16        System.out.println("tempK = " + tempK + "K"); // resultado
17        // cria conversor Kelvin p/ Celsius e exibe info
18        KelvinCelsius kc = new KelvinCelsius();
19        System.out.println("kc : " + kc); // uso implícito de toString()
20        tempC = kc.converter(tempK); // conversão K-->C
21        System.out.println("tempC = " + tempC + "C"); // resultado
22    }
23 }
24
```

(C) 1999-2020 Jandl 09/02/2021 23

Herança & Hierarquias::um exemplo



```
<terminated> UsaConversores [Java Application] C:\Arquivos de programas\Java\jre7\bin\javaw.exe (03/05/2012 11:15:50)
Digite temperatura Celsius: 23
ck : Conversor[kProp=1.0,kLin= 273.0]
tempK = 296.0K
kc : Conversor[kProp=1.0,kLin= -273.0]
tempC = 23.0C
```

- Classe **UsaConversores** constitui a interface de uma aplicação dos conversores **CelsiusKelvin** e **KelvinCelsius**, estas derivadas da classe **Conversor**.

POO::sobreposição

Sobreposição de métodos

- A sobreposição (ou substituição) de métodos consiste na implementação de método na subclasse com a mesma assinatura de outro existente na superclasse.
- Permite dotar a subclasse com implementação distinta da superclasse, mas mantendo sua interface (o que facilita seu uso por meio do polimorfismo).
- Também é conhecida como *method override*.

Method overload é a sobrecarga de métodos *intraclasse*.

Não confundir com *method overload*.

Method override é a sobreposição de métodos *interclasses*.

Sobreposição::um exemplo

- Apesar das conversões lineares serem muito comuns, existem outras conversões e modelos úteis em outros domínios, por exemplo:
 - Modelo hiperbólico $f(x) = 1 / (\alpha + \beta x)$
 - Modelo quadrático $f(x) = \alpha * x^2 + \beta x + \gamma$
- O conversor linear desenvolvido (classe Conversor) é apenas parcialmente adequado.

Sobreposição::um exemplo

```
// Hiperbolico.java
public class Hiperbolico
    extends Conversor {
    public Hiperbolico ( double alfa, double beta) {
        // aciona construtor da superclasse
        super(alfa, beta);
    }
    @Override
    public double converter(double valor) {
        return 1 / (valor*alfa + beta);
    }
}
```

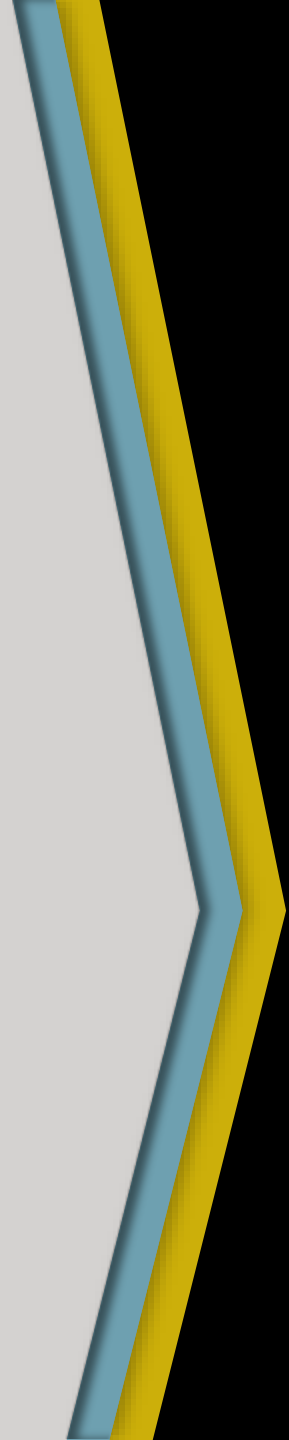
Substituição do método
converter(double)

Sobreposição::um exemplo

- Uso do conversor hiperbólico:

```
public class TesteHiperbolico {  
    public static void main(String[] a) {  
        Hiperbolico hiper = new Hiperbolico(1.3, -0.5);  
        System.out.printf(" | ----x---- | -----f(x)----- | ", v, res);  
        for (double v = -5.0; v <= 5.0 ; v += 0.5) {  
            double res = hiper.converter(v);  
            System.out.printf(" | %9.4f | %20.4f | \n", v, res);  
        }  
    }  
}
```

Exercícios Propostos & Recomendações de Estudo



Exercícios Propostos

Entrega HOJE, até o final da aula, por envio via chat do Zoom.

1. Implemente todas as classes presentes nos slides. Sugere-se que todas façam parte do pacote **lpoo**.

2. Implemente e teste dois novos conversores como feito nos slides:

- a) Celsius para Fahrenheit
- b) Fahrenheit para Celsius

Duas classes separadas para os conversores e uma terceira, com main, para o teste.

3. Implemente e teste um novo modelo de conversão quadrática empregando a sobreposição.

Uma classe para o conversor e outra, com main, para o teste.

Recomendações de Estudo

- **Java – Guia do Programador**,
3ª Edição,
P. JANDL Jr,
Novatec, 2015.
- **Java 6- Guia de Consulta Rápida**,
P. JANDL Jr,
Novatec, 2008.
- **Java 5- Guia de Consulta Rápida**,
P. JANDL Jr,
Novatec, 2006.
- **Introdução ao Java**,
P. JANDL Jr,
Berkeley, 2002.

