

# PROGRAMAÇÃO ORIENTADA A OBJETOS

## HERANÇA



6/6/2008

Programação Orientada a Objetos  
Prof. Ademar Schmitz, M.Sc.

1



## AGENDA

1. Conceito de Herança.
2. Mecânica da Herança.
3. Objetivos da Herança.
4. Conversão de Tipos.
5. Como evitar a Herança.
6. Dicas para o Projeto de Herança.

6/6/2008

Programação Orientada a Objetos  
Prof. Ademar Schmitz, M.Sc.

2



## O QUE É HERANÇA?

- **Encapsulamento** permite:
  - Escrever objetos definidos e independentes;
  - Permite que um objeto utilize outro;
- **Herança** permite:
  - Criar uma classe a partir da definição de outra classe.

6/6/2008

Programação Orientada a Objetos  
Prof. Ademar Schmitz, M.Sc.

3



## O QUE É HERANÇA?

- Mecanismo que permite basear uma nova classe na definição de uma classe previamente existente;
- Quando uma classe **herda** de outra, todos os métodos e atributos que aparecem na interface da classe previamente existente apareceram automaticamente na interface da nova classe;
- A herança permite à classe que está sendo herdada redefinir qualquer comportamento que não goste.

6/6/2008

Programação Orientada a Objetos  
Prof. Ademar Schmitz, M.Sc.

4



## O QUE É HERANÇA?

- Com identificar o tipo de relacionamento?
  - “É um”: Herança
  - “Tem um”: Associação
  - “Usa um”: Dependência

6/6/2008

Programação Orientada a Objetos  
Prof. Ademar Schmitz, M.Sc.

5




## O QUE É HERANÇA?

- “**É um**”: descreve o relacionamento em que a classe que esta herdando é do mesmo tipo da classe herdada;
- “**Tem um**”: descreve o relacionamento em que uma classe contém uma instância de outra classe;
- “**Usa um**”: descreve o relacionamento em que uma classe depende de outra classe.

6/6/2008

Programação Orientada a Objetos  
Prof. Ademar Schmitz, M.Sc.


6



## HERANÇA

- No Java, a palavra-chave **extends** indica que está sendo criada uma nova classe que deriva de uma classe existente;
- A classe existente é chamada de **superclasse**, **classe base** ou **classe progenitora**;
- A nova classe é chamada de **subclasse**, **classe derivada** ou **classe filha**;
- As **subclasses** tem mais funcionalidades que suas **superclasses**.


6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 7



## HERANÇA

- A palavra-chave **super** sempre se refere a uma superclasse;
- Se o construtor da subclasse não chamar um construtor da superclasse explicitamente, então a superclasse usa seu **construtor padrão** (sem argumentos);
- Se a superclasse não tiver construtor padrão e o construtor da subclasse não chamar nenhum outro construtor da superclasse explicitamente, então o compilador Java vai informar um **erro**.


6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 8



## MECÂNICA DA HERANÇA

- Quando uma classe herda de outra, ela herda implementação, comportamentos e atributos;
- Uma classe pode ter três tipos de métodos e atributos:
  - Sobrepostos**: a nova classe herda o método ou atributo, mas define uma nova "definição" para eles;
  - Novos**: A nova classe adiciona um método ou atributo completamente novo;
  - Recurativos**: a nova classe simplesmente herda um método ou atributo da progenitora;


6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 9



## SOBREPOSIÇÃO DE MÉTODOS

- É o processo de uma filha pegar um método que aparece na progenitora e reescrevê-lo para mudar o seu comportamento;
- Também conhecido como:
  - Sobrescrita**;
  - Overriding**;

6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 10




## class Pontos

```

class Pontos
{
    class PontoUnidimensional
    {
        - coordenada_x: double
        + getCoordenada_x(): double
        + setCoordenada_x(double): void
        + toString(): String
    }
    class PontoBidimensional
    {
        - coordenada_y: double
        + getCoordenada_y(): double
        + setCoordenada_y(double): void
        + toString(): String
    }
    class PontoTridimensional
    {
        - coordenada_z: double
        + getCoordenada_z(): double
        + setCoordenada_z(double): void
        + toString(): String
    }
}
        
```



6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 11



## SOBREPOSIÇÃO DE MÉTODOS E ATRIBUTOS

- Neste caso ocorre a sobreposição do método **toString()**;
- Diz-se que a subclasse **sobrescreve** a método definido pela superclasse;
- Chamamos este processo de **sobrescrita**.



6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 12

## MÉTODOS E ATRIBUTOS NOVOS

- É um método ou atributo que aparece na filha, mas não aparece na progenitora:
- Atributos novos:
  - `coordenada_y` em `PontoBidimensional`.
  - `coordenada_z` em `PontoTridimensional`.
- Métodos novos:
  - `getCoordenada_y` em `PontoBidimensional`.
  - `setCoordenada_y` em `PontoBidimensional`.
  - `getCoordenada_z` em `PontoTridimensional`.
  - `setCoordenada_z` em `PontoTridimensional`.



6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 13

## MÉTODOS E ATRIBUTOS RECURSIVOS

- É um método ou atributo que aparece na progenitora mas não aparece na filha;
- Atributos recursivos:
  - `coordenada_x` em `PontoUnidimensional`.
- Métodos recursivos:
  - `getCoordenada_x` em `PontoUnidimensional`.
  - `setCoordenada_x` em `PontoUnidimensional`.



6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 14

## OBJETIVOS DA HERANÇA

- Reutilização de implementação:
  - Obtida através de atributos e métodos recursivos;
- Diferença:
  - Obtida através de atributos e métodos sobrescritos e novos.
- Substituição de tipo:
  - Obtida através da capacidade utilizar subtipos no lugar de supertipos.



6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 15

## HERANÇA PARA SUBSTITUIÇÃO DE TIPO

- Permite que se descreva relacionamentos com capacidade de substituição;
- A capacidade de substituição aumenta a capacidade de reutilização.

6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 16

## HERANÇA PARA SUBSTITUIÇÃO DE TIPO

```

class Pontos
{
    class PontoUnidimensional
    {
        - coordenada_x: double
        + getCoordenada_x(): double
        + setCoordenada_x(double): void
        + toString(): String
    }



    class PontoBidimensional
    {
        - coordenada_y: double
        + getCoordenada_y(): double
        + setCoordenada_y(double): void
        + toString(): String
    }

    class PontoTridimensional
    {
        - coordenada_z: double
        + getCoordenada_z(): double
        + setCoordenada_z(double): void
        + toString(): String
    }

    class Linha
    {
        + ponto1: PontoBidimensional
        + ponto2: PontoBidimensional
        + getDistancia(): double
        + getPontoMedio(): PontoBidimensional
    }
}

```

6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 17

## HERANÇA

```

class Diagrama de Classes
{
    class ItemDeVenda
    {
        - quantidade: int
        - desconto: double
        + totalItemSemDesconto(): double
        + totalItemComDesconto(): double
    }

    class Produto
    {
        - codigo: int
        - descricao: String
        - preco: double
        - qtdEstoque: int
        + incrementaEstoque(int): int
        + decrementaEstoque(int): int
    }

    class Livro
    {
        - autor: String
        - isbn: String
    }



    class Cd
    {
        - duracao: int
        - numeroFaixas: int
    }

    class Dvd
    {
        - duracao: int
    }

    ItemDeVenda o1 --> "0..*" Produto : -produto
    Produto <|-- Livro
    Produto <|-- Cd
    Produto <|-- Dvd
}

```



6/6/2008
 Programação Orientada a Objetos  
 Prof. Ademar Schmitz, M.Sc.
 18

## HIERARQUIA DE HERANÇAS

- A herança não está limitada a derivar apenas uma camada de classes;
- A coleção de todas as classes que derivam de um ancestral em comum é chamada de **hierarquia de heranças**;
- O caminho de uma classe particular até o seu ancestral na hierarquia das heranças é chamado de **seqüência de heranças** (ou descendência).



6/6/2008
Programação Orientada a Objetos  
Prof. Ademair Schmitz, M.Sc.
19

## CHAMADAS AO SUPER

- Chamadas ao **super** em **construtores**:
  - O construtor de uma subclasse sempre deve invocar o construtor da sua superclasse como sua primeira instrução;
  - Se o código fonte não incluir esta chamada, o Java inserirá a chamada automaticamente.



6/6/2008
Programação Orientada a Objetos  
Prof. Ademair Schmitz, M.Sc.
20

## CHAMADAS AO SUPER

- Chamadas ao super em **métodos**:
  - Ao contrário das chamadas super nos construtores, o nome da superclasse é declarado explicitamente;
  - Ao contrário à regra para chamadas super nos construtores, a chamada nos métodos pode ocorrer em qualquer lugar dentro desse método;
  - Ao contrário às chamadas super nos construtores, nenhuma chamada super automática é gerada e nenhuma chamada super é requerida.



6/6/2008
Programação Orientada a Objetos  
Prof. Ademair Schmitz, M.Sc.
21

## SUBCLASSES E SUBTIPOS

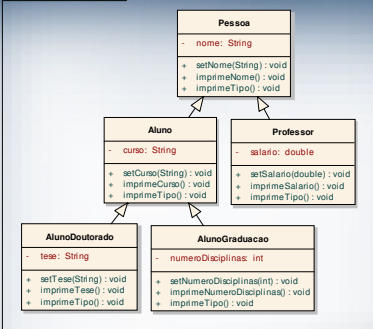
- O tipo definido por uma definição de **subclasse** é um **subtipo** do tipo definido pela superclasse correspondente;
- Variáveis podem armazenar objetos do seu tipo declarado ou de qualquer subtipo de seu tipo declarado;
- Os objetos de **subtipo** podem ser utilizados onde quer que os objetos de um **supertipo** sejam esperados.

6/6/2008
Programação Orientada a Objetos  
Prof. Ademair Schmitz, M.Sc.
22






## SUBCLASSES E SUBTIPOS

class Diagrama de Classes



6/6/2008
Programação Orientada a Objetos  
Prof. Ademair Schmitz, M.Sc.
23





## CONVERSÃO DE TIPOS


- Embora um objeto de **subclasse** seja um objeto de **superclasse**, os dois objetos são diferentes;
- Objetos de **subclasses** podem ser tratados como se fossem objetos de **superclasses**;
- Entretanto, o oposto não é verdadeiro:
  - A subclasse pode ter membros adicionais.

6/6/2008
Programação Orientada a Objetos  
Prof. Ademair Schmitz, M.Sc.
24






## CONVERSÃO DE TIPOS




- A conversão de um tipo de referência em outro tipo de referência é chamado de **casting**;
- A conversão de tipos é permitida apenas dentro em um relacionamento superclasse (supertipo) / subclasse (subtipo);
- Temos dois tipos de **casting**:
  - Upcasting (widening)
  - Downcasting (narrowing)

6/6/2008 Programação Orientada a Objetos Prof. Ademar Schmitz, M.Sc. 25




## CONVERSÃO DE TIPOS




- **Upcasting**
  - Significa converter uma referência subtipo para uma referência supertipo;
  - Sempre é permitida;
  - Realizada implicitamente.

6/6/2008 Programação Orientada a Objetos Prof. Ademar Schmitz, M.Sc. 26




## CONVERSÃO DE TIPOS




- **Downcasting**
  - Significa converter uma referência supertipo em uma referência subtipo;
  - Deve ser realizada explicitamente;
  - É realizada em tempo de execução.

6/6/2008 Programação Orientada a Objetos Prof. Ademar Schmitz, M.Sc. 27




## COMO EVITAR HERANÇA E SOBRESCRITA




- As classes que não podem ser **classes-mãe** são chamadas de **classes finais**;
- Usa-se o modificador **final** na definição da classe para indicar isto;

6/6/2008 Programação Orientada a Objetos Prof. Ademar Schmitz, M.Sc. 28




## COMO EVITAR HERANÇA E SOBRESCRITA




- Pode-se também fazer **final** um método específico de uma classe;
- Nenhuma subclasse poderá sobrepor ou substituir esse método;
- Todos os métodos de uma classe final são automaticamente “finais”.

6/6/2008 Programação Orientada a Objetos Prof. Ademar Schmitz, M.Sc. 29



## RECOMENDAÇÕES DE PROJETO DE HERANÇA



1. Coloque operações e campos comuns na superclasse;
2. Use herança para modelar uma relação de “é um tipo de”;
3. Não use herança a menos que todos os métodos herdados façam sentido.
4. Use polimorfismo, não informação de tipo.

6/6/2008 Programação Orientada a Objetos Prof. Ademar Schmitz, M.Sc. 30