HERANÇA EM JAVA

Técnicas de Programação

Jose Macedo

Herança e Polimorfismo

- Uma classe pode herdar propriedades e métodos de outra classe
- Em Java, podemos ter herança de código ou herança de interface
- Herança de Interface: herda todos métodos declarados que não sejam privados
- Herança de Código: herda métodos com implementações e campos que não são privados

Herança em Java

- Quando uma classe A herda de B, diz-se que A é a sub-classe e estende B, a superclasse
- Uma classe Java estende apenas uma outra classe- a essa restrição damos o nome de herança simples
- Para criar uma sub-classe, usamos a palavra reservada *extends*

Exemplo de Herança

- Suponhamos que existe a classe Ponto.

```
class Ponto {

  float x, y;
  Ponto () {
     x=0; y=0;
  }

   Ponto(float x1, float y1) {
        x = x1; y = y1;
    }

    void setXY (float x1, float y1) {
        x = x1; y = y1;
    }

    void mover(float dx, float dy) {
        x = x + dx; y = y + dy;
    } } }
```

Exemplo de Herança

- Podemos criar uma classe PontoColorido a partir da classe Ponto.

```
class PontoColorido extends Ponto {
  int cor;
  void setCor(int c) {
        cor = c;
  }
}
```

Herança de Código

- A classe PontoColorido herda a interface e o código da classe Ponto. Ou seja, PontoColorido passa a ter tanto os campos quanto os métodos (com suas implementações) de Point.

```
PontoColorido p1 = new PontoColorido();
p1.setXY(2,1);
p1.setCor(0);
p1.mover(1,0);
```

- Porém o ideal seria definir um construtor para PontoColorido similar ao construtor de Ponto:

```
class PontoColorido extends Ponto {
  int cor;
  PontoColorido(float x1, float y1, int c)
  {
        x = x1; y = y1
        cor = c;
     }
}
```

Palavra reservada “super”

- Usada para referenciar o construtor da superclasse.
- PontoColorido precisa iniciar sua parte Ponto antes de iniciar sua parte estendida
- No caso de não haver chamada explícita, a linguagem Java chama o construtor padrão da super classe (super()), como foi o caso do exemplo anterior

Uso de "*super(...)*"

- Podemos modificar a classe PontoColorido para chamar o metodo construtor da superclasse.

```
class PontoColorido extends Ponto {
  int cor;
  PontoColorido(float x, float y, int c) {
        super(x,y);
        cor = c;
     }
}
```

Herança de Código

- Com o uso de super utilisamos o codigo definido no metodo construtor da superclasse.
- Podemos criar agora um PontoColorido usando o novo construtor de PontoColorido:

```
PontoColorido p1 = new PontoColorido(1,2,0);
p1.mover(1,0);
```

Modelo de Herança

- Java adota o modelo de árvore
- A classe **Object** é a raiz da hierarquia de classes à qual todas as classes existentes pertencem
- Quando não declaramos que uma classe estende outra, ela, implicitamente, estende **Object**

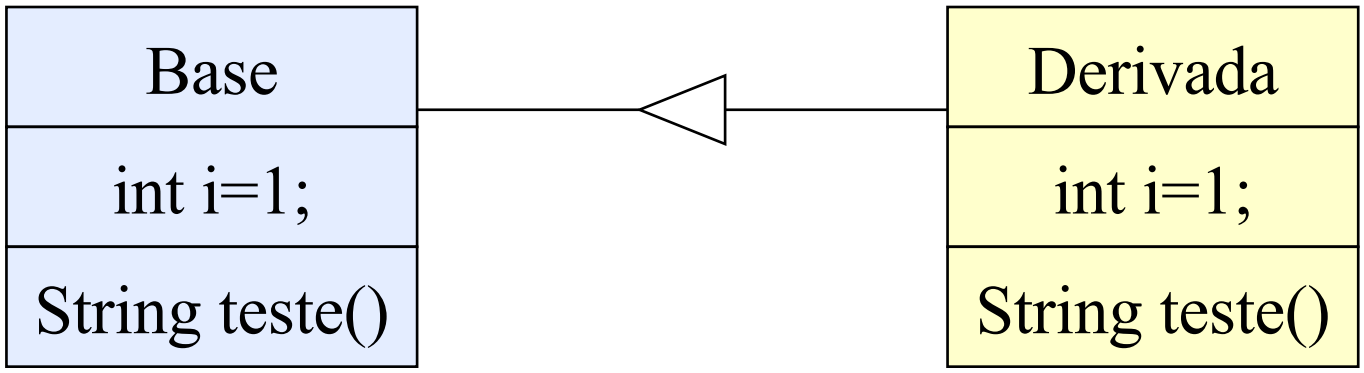
Especialização x Extensão

- Uma classe pode herdar de outra para especializá-la redefinindo métodos, sem ampliar sua interface
- Uma classe pode herdar de outra para estendê-la declarando novos métodos e, dessa forma, apliando sua interface.
- Ou as duas coisas podem acontecer simultaneamente

Sombreamento de propriedades e métodos


```
class Base {
        int i = 1;
        String teste() {
                return "Base";
        }
}
```

```
class Derivada extends Base {
    int i = 2;
    String teste() {
            return "Derivada";
    }
}
```



Qual serão os valores finais de x, y, z, xstr, ystr e zstr?


```
class Heranca {
    public static void main (String arg[]) {
        Base base = new Base();
        Derivada derivada = new Derivada();
        Base baseDerivada = new Derivada();
        int x = base.i;
        int y = derivada.i;
        int z = baseDerivada.i;
        String xstr = base.teste();
        String ystr = derivada.teste();
        String zstr = baseDerivada.teste();
}
```

Resultado

Se incluirmos:

```
System.out.println("Teste de Herança");
System.out.println("Propriedades => "+x+" - "+y+" - "+z);
System.out.println("Metodos      => "+xString+" - "+yString+" - "+zString);
```

A saída será:

Teste de Herança

Propriedades => 1 - 2 - 1

Metodos   => Base - Derivada - Derivada

Sombreamento de propriedades e métodos


```
class Super {
    Super() {
       System.out.println("Construtor    Super"); }

    void teste() {
       System.out.println("Metodo   Superclasse");}
}
```

```
class Subclasse extends Super {
    Subclasse() {
        System.out.println("Construtor Subclasse");  }

    void teste() {
       super.teste();
       System.out.println("Metodo Subclasse"); }
    }
```

O que será impresso ?


```
class HerancaTeste {

    public static void main (String arg[])
  {
          System.out.println("*** TESTE
  ***");
          Subclasse sub = new Subclasse();
          sub.teste();
    }

}
```

O resultado será ?


```
*** TESTE ***
Contrutor Super
Contrutor Subclasse
Metodo Superclasse
Metodo Subclasse
```

Métodos Constantes

- Métodos cujas implementações não podem ser redefinidas nas sub-classes.
- Objetivo é evitar o processamento do Late Binding
- Devemos usar o modificador final

```
public class A{
  public final int f ( ) {
    ...
    }
}
```

Classes Constantes

- Uma classe constante não pode ser estendida.

```
public final class A{
   ...
}
```

Conversão de Tipo

- Podemos usar uma versão mais especializada quando precisamos de um objeto de certo tipo mas o contrário não é verdade. Por isso, se precisarmos fazer a conversão de volta ao tipo mais especializado, teremos que fazê-lo explicitamente.

Type Casting

- A conversão explícita de um objeto de um tipo para outro é chamada de type casting.

```
Ponto pt = new PontoColorido(0,0,1);
PontoColorido px = (PontoColorido)pt;
pt = new Ponto();
px = (PontoColorido)pt; //ERRO
pt = new PontoColorido(0,0,0);
px = pt; //ERRO
```

Classes Abstratas

- Ao criarmos uma classe para ser estendida. Às vezes codificamos alguns métodos para os quais não sabemos dar uma implementação, ou seja, um método que só sub-classes saberão implementar
- Uma classe deste tipo não pode ser instanciada pois sua funcionalidade está incompleta. Tal classe é dita *abstrata*.

Classes Abstratas em Java

- Java suporta o conceito de classes abstratas: podemos declarar uma classe abstrata usando o modificador **abstract**
- Métodos podem ser declarados abstratos para que suas implementações sejam adiadas para as sub-classes. Da mesma forma, usamos o modificador abstract e omitimos a implementação desse método.

Classes Abstratas


```
public abstract class Figura  {
    Ponto pOrigem;
     public abstract void desenhar();
     public abstract void apagar();
     public abstract float calcularArea();
     public void mover(Ponto p) {
         origem.mover(pOrigem.x,pOrigem.y);
     }
}
```