Programação OO em Java

Profa Andréa Schwertner Charão

DELC/CT/UFSM

Sumário

- Classes abstratas
- Interfaces
- Tipos genéricos

Classes abstratas

- São classes que **não podem** ser instanciadas, porque representam entidades "incompletas"
- Possuem métodos abstratos que devem ser sobrescritos nas classes derivadas

```
abstract class Bicho
{
  protected String nome;
  public Bicho(String nome)
  {
    this.nome = nome;
  }
  abstract public String som();
}
```

mensagem do compilador:

```
Bicho.java: Bicho is abstract; cannot be instantiated
    Bicho b = new Bicho();
              ^
1 error
```

Exemplo

```
abstract class Bicho
{
  protected String nome;
  public Bicho(String nome)
  {
    this.nome = nome;
  }
  abstract public String som();
}

class Cachorro extends Bicho
{
  public Cachorro(String nome)
  {
    super(nome);
  }
  public String som()
  {
    return "Au Au";
  }
}

class Gato extends Bicho
{
  public Gato(String nome)
  {
    super(nome);
  }
  public String som()
  {
    return "Miau";
  }
}
```

Método **abstrato** som()

Método **concreto** som()

Cria array de referências para Bichos

```
class BichoApp
{
  public static void main(String[] args)
  {
    Bicho[] bs = new Bicho[2];
    bs[0] = new Cachorro("Scooby");
    bs[1] = new Gato("Garfield");
    for (int i = 0; i < bs.length; i++)
      System.out.println(bs[i].som());
  }
}
```

Classes abstratas

Erro: classe não abstrata com método abstrato

```
class Bicho
{
  protected String nome;
  public Bicho(String nome)
  {
    this.nome = nome;
  }
  abstract public String som();
}
```

- Métodos abstratos só podem ser declarados em classes abstratas
- Em geral, classes abstratas também possuem métodos concretos
- Se uma classe só tem métodos abstratos, é melhor declará-la como **interface**

mensagem do compilador:

```
Bicho.java: Bicho is not abstract and does not override abstract method som() in Bicho
class Bicho
^
1 error
```

Java na Desciclopédia :-)

Fonte:
http://desciclopedia.org/wiki/Java_(linguagem_de_programação)

Interfaces

- São um tipo de encapsulamento contendo principalmente métodos
- Definem um conjunto de métodos (comportamento) que devem ser implementados em classes que herdam a interface

```
interface Matricial
{
  public void transpoe();
  public void inverte();
}
```

```
interface Runnable
{
  public void run();
}
```

Implementando interfaces

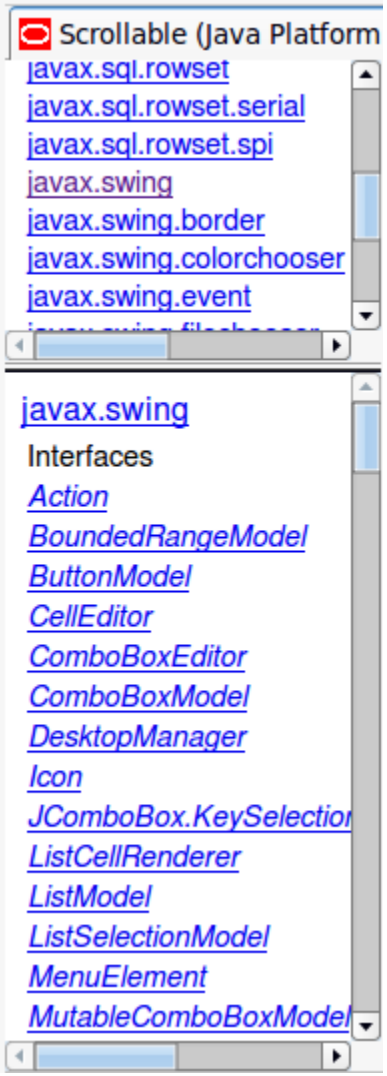
- Usar a palavra-chave **implements**

```
class MatrizEsparsa
   implements Matricial
{
   public void transpoe()
   { ... }
   public void inverte()
   { ... }
}
```

```
class Worker
   implements Runnable
{
   public void run()
   { ... }
}
```

Implementando interfaces

- Classes que implementam uma mesma interface garantem que têm um comportamento comum
- A plataforma Java tem diversas interfaces pré-definidas (ActionListener, Scrollable, Runnable, etc.)



Mais sobre interfaces

- Java suporta "herança múltipla" de interfaces, mas não de classes

```
class A {...}
interface B {...}
interface B {...}

class X extends A
implements B,C
{...}
```

Mais sobre interfaces

- Atributos declarados em interfaces são implicitamente `public static final` (constantes)
- Métodos declarados em interfaces são implicitamente `public abstract`

Veja mais sobre interfaces em:
http://download.oracle.com/javase/tutorial/java/concepts/interface.html

Tipos genéricos

- Classes genéricas definidas em função de algum parâmetro (tipos parametrizáveis)
- Polimorfismo paramétrico

```
class Box<T> {

    private T t; // T significa "Type"

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
}
```

Usando tipos genéricos

- Para usar o tipo, define-se o parâmetro específico

```
class BoxApp {

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        Integer i = integerBox.get();
        System.out.println(i);

        Box<String> stringBox = new Box<String>();
        stringBox.add("Hello");
        String s = stringBox.get();
        System.out.println(s);

    }
}
```

Métodos genéricos

- Métodos também podem ser genéricos
- Podem ou não pertencer a classes genéricas

```
public class Box<T> {

    private T t;

    public void add(T t) {

    public T get() {

    public <U> void inspect(U u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }

    public static void main(String[] args) {
}
```

Mais sobre genéricos em Java

- Genéricos podem ser aninhados (ex.: List<Box<String>>)
- **Convenção**: usar maiúscula para indicar um nome de parâmetro
  - T: Type (primeiro parâmetro de tipo)
  - S, U, V, etc.: segundo, terceiro, etc. parâmetros de tipo
  - E: Element (usado em coleções)

```
class Box<T,S,U>
{ ... }
```

Veja mais em:
http://download.oracle.com/javase/tutorial/java/generics/index.html