

# **Programação OO em Java**

Profa Andréa Schwertner Charão  
DLSC/CT/UFSM

# Sumário

- Classes abstratas
- Interfaces
- Tipos genéricos
- Coleções
- Introdução GUI

# 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();
}
```

Método **abstrato**  
som()

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

Método **concreto**  
som()

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

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

- 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**

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

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 :-)



**Classes  
abstratas :-)**

Fonte:

[http://desciclopedia.org/wiki/Java\\_\(linguagem\\_de\\_programação\)](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**

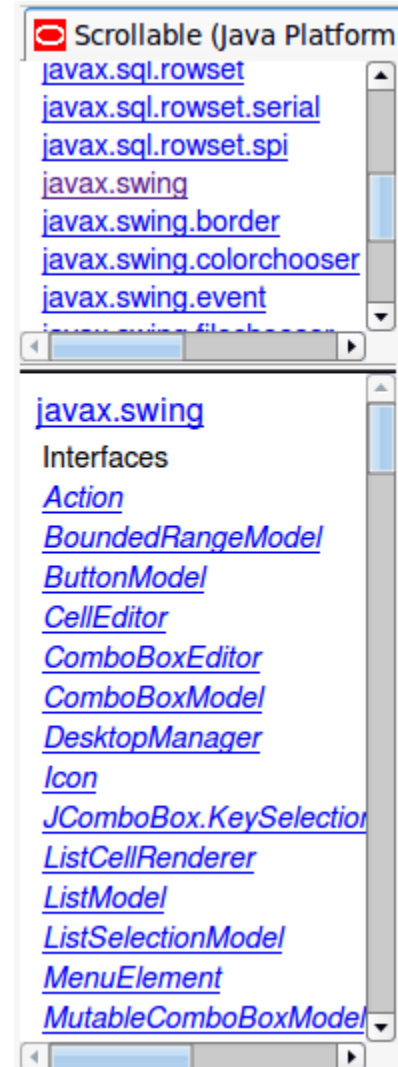
```
class MatrizEsparsa
    implements Matricial
{
    public void transpoe()
    { ... }
    public void inverte()
    { ... }
}
```

```
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);  
    }  
}
```

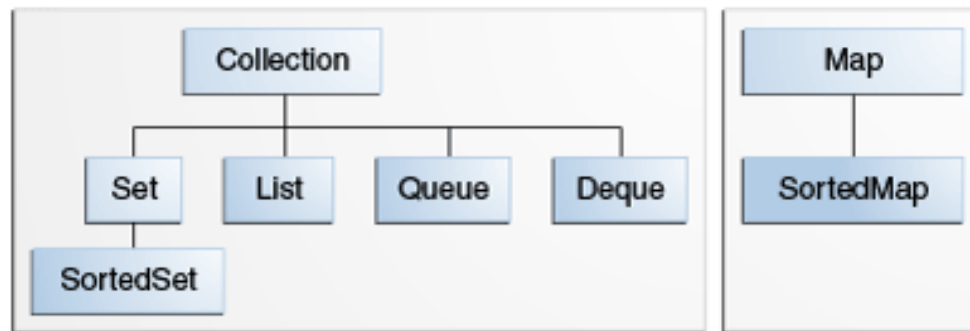
Veja mais em:

<http://download.oracle.com/javase/tutorial/java/generics/index.html>

# Collections em Java

- Um framework com **estruturas de dados** e **algoritmos** reutilizáveis, disponíveis em `java.util`.
- Componentes
  - **Interfaces**: definem como as estruturas podem ser manipuladas (ex.: `List`)
  - **Implementações**: definem estruturas concretas (ex.: `ArrayList`, `LinkedList`)
  - **Algoritmos**: métodos estáticos que se aplicam a diferentes coleções

# Collections Interface



Fonte:

<http://download.oracle.com/javase/tutorial/collections/interfaces/index.html>

# Exemplo de implementação: ArrayList

- Representa uma lista que pode ser acessada por índices (0 a size()-1)
- Implementa métodos da **interface List**:
  - add(E e): adiciona elemento
  - size(): número de elementos da lista
  - clear(): remove todos os elementos
  - isEmpty(): verifica se lista é vazia
  - remove(Object o): remove elemento
  - remove(int index): remove elemento
  - etc.



# Exemplo

```
import java.util.*;
class ArrayListExemplo {
    public static void main(String args[]) {
        // cria objeto
        ArrayList<String> sl = new ArrayList<String>();
        System.out.println("Tamanho inicial: " + sl.size());
        // adiciona elementos
        sl.add("Fulano");
        sl.add("Beltrano");
        sl.add("Sicrano");
        sl.add("Fulana");
        System.out.println("Novo tamanho: " + sl.size());
        // remove elementos
        sl.remove("Beltrano");
        sl.remove(0);
        System.out.println("Conteudo: " + sl);
    }
}
```

saída:

Tamanho inicial: 0  
Novo tamanho: 4  
Conteudo: [Sicrano, Fulana]

# Percorrendo a lista

- Laço **for** tradicional, com índice

```
for (int i = 0; i < sl.size(); i++) {  
    String elem = sl.get(i);  
    System.out.println(elem);  
}
```

# Percorrendo a lista com for-each

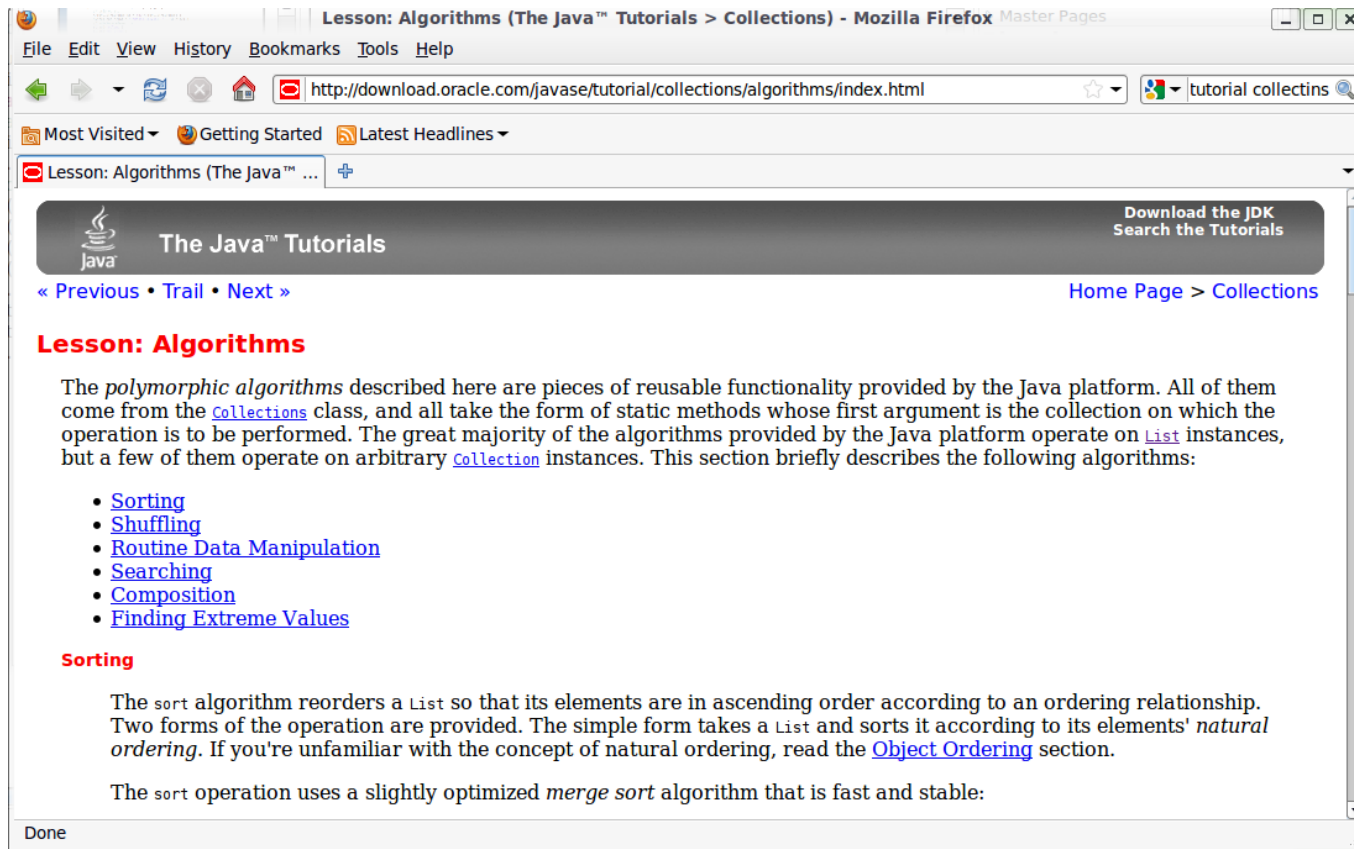
- Laço **for** alternativo (**for-each**)
- Inspirado na programação funcional
- Pode ser usado com arrays ou classes Collection

The diagram illustrates the components of the `for (String elem : sl) { ... }` loop. Three arrows point from descriptive text to parts of the code: 'tipo dos elementos na coleção' points to 'String', 'referência para elemento da coleção' points to 'elem', and 'referência para a coleção' points to 'sl'.

```
for (String elem : sl) {  
    System.out.println(elem);  
}
```

# Algoritmos

- Ordenação, busca, embaralhamento, etc.



# Algoritmos: sort

```
import java.util.*;

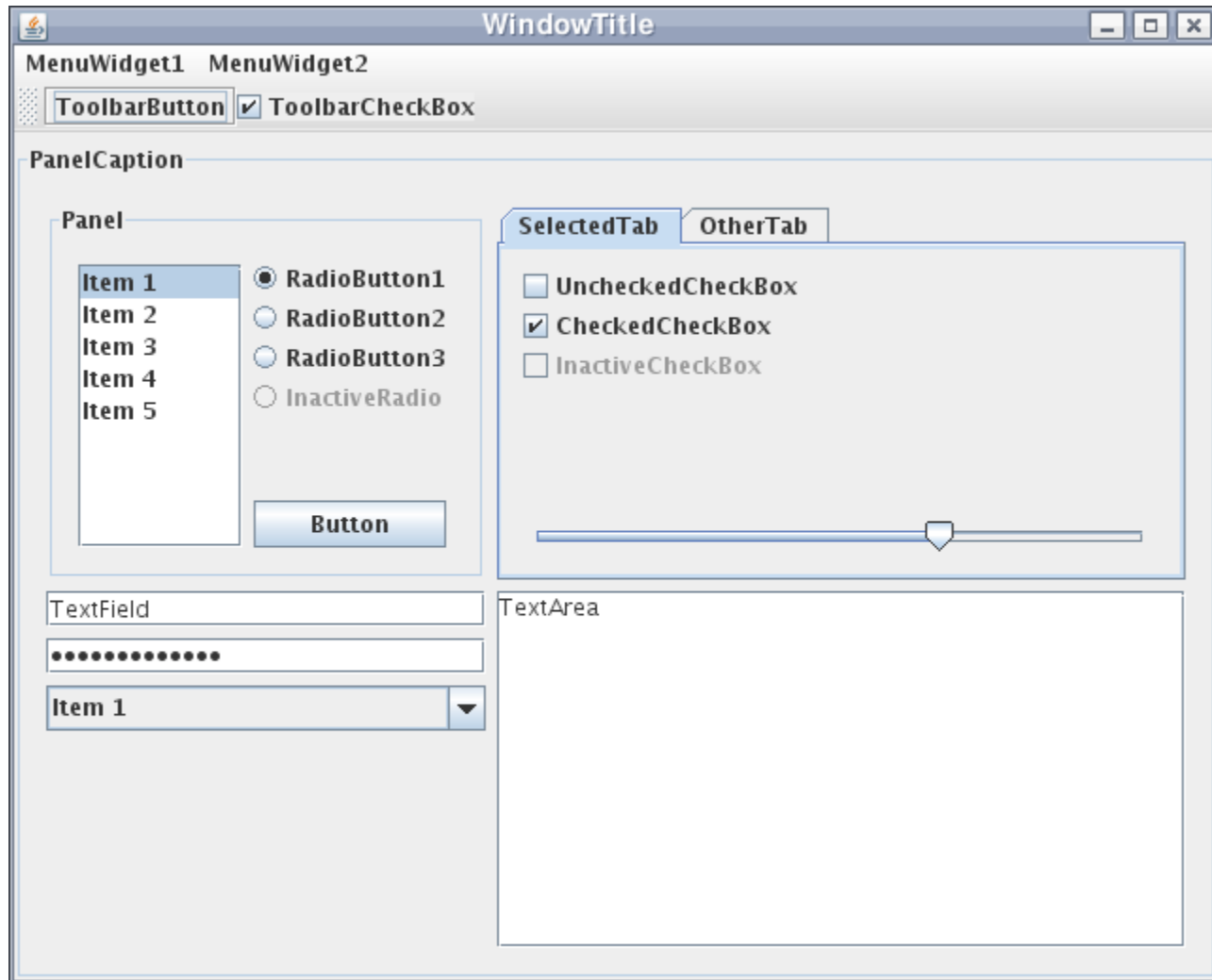
class Sort {
    public static void main(String[] args) {
        String[] array = {"cadabra", "abra"};
        List<String> list = Arrays.asList(array);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

# Algoritmos: shuffle

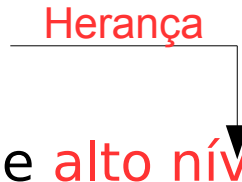
```
import java.util.*;

class Shuffle {
    public static void main(String[] args) {
        String[] array = {"a", "b", "c", "d", "e"};
        List<String> list = Arrays.asList(array);
        Collections.shuffle(list);
        System.out.println(list);
    }
}
```

# Graphical User Interfaces (GUIs)

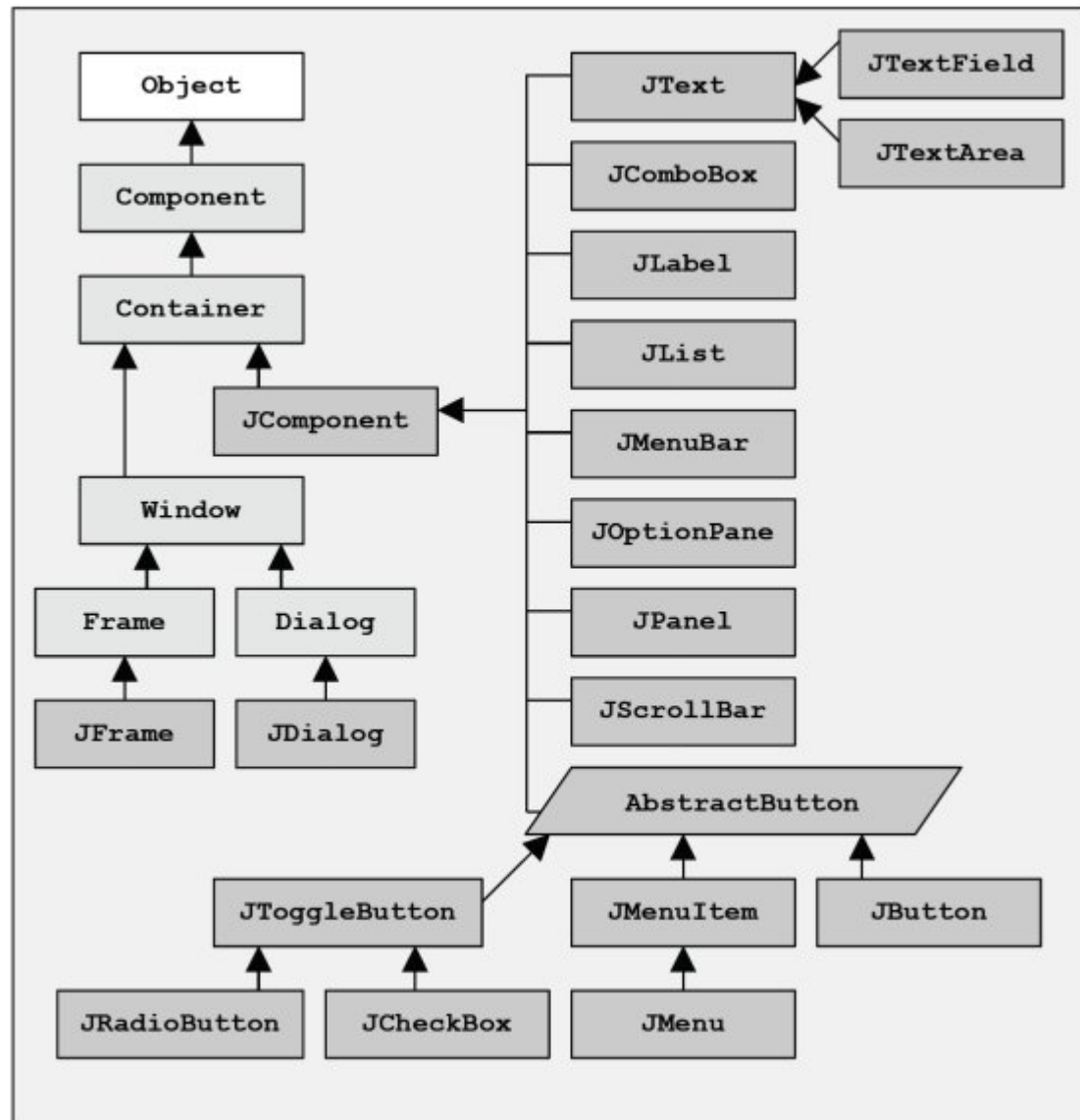


# Graphical User Interfaces (GUIs)

- Oferecem portabilidade
  - Utilizadas tanto em Desktop quanto Web
  - Pacotes de classes da plataforma java:
    - **java.awt.\***: elementos básicos
    - **javax.swing.\***: componentes de alto nível
    - **javafx.\***: interfaces “ricas em efeitos” (Java SE 8)
- 
- ```
graph TD; A["java.awt.*: elementos básicos"] -- Herança --> B["javax.swing.*: componentes de alto nível"]
```



# Hierarquia de classes

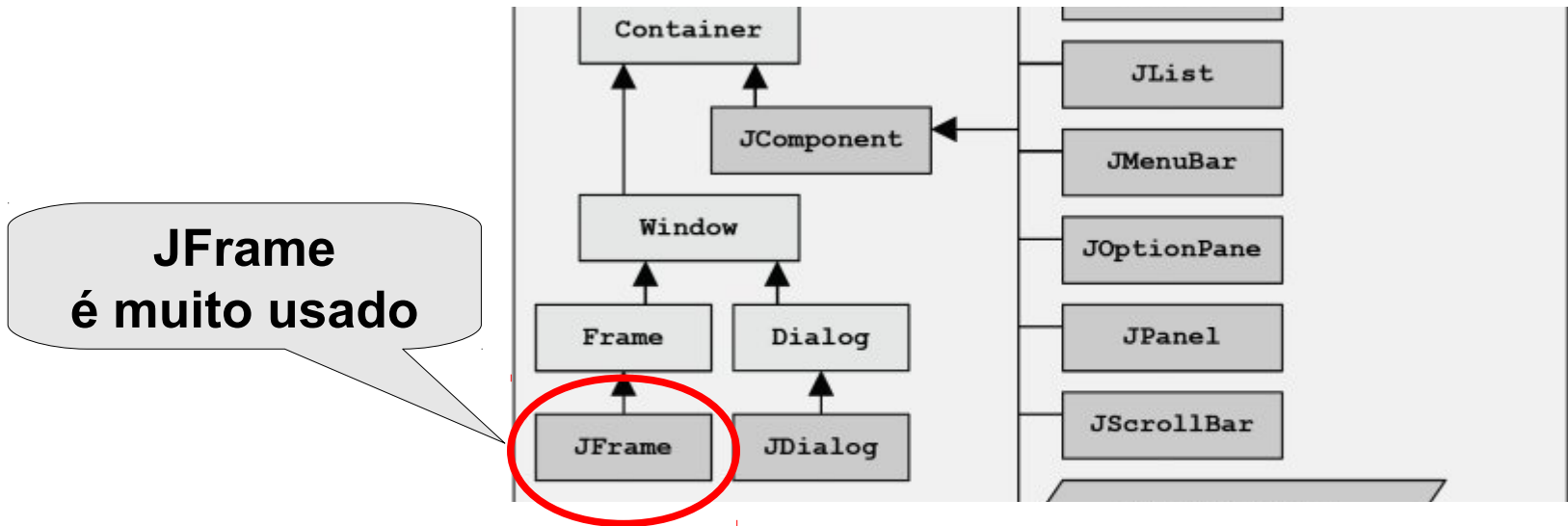


Fonte:

JavaTech, an Introduction to Scientific and Technical Computing with Java  
 Por Clark S. Lindsey, Johnny S. Tolliver, Thomas Lindblad

# Categorias de classes

- **Containers** (ex: JFrame, JDialog, JPanel): são componentes que contêm outros objetos
- Em geral, qualquer GUI usa no mínimo uma classe desta categoria

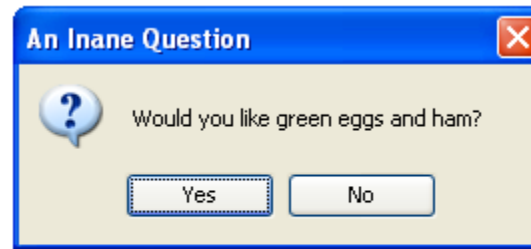


# Categorias de classes

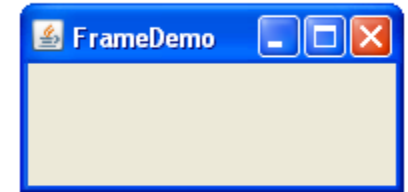
## ■ Alguns **containers**:

- JFrame
- JDialog

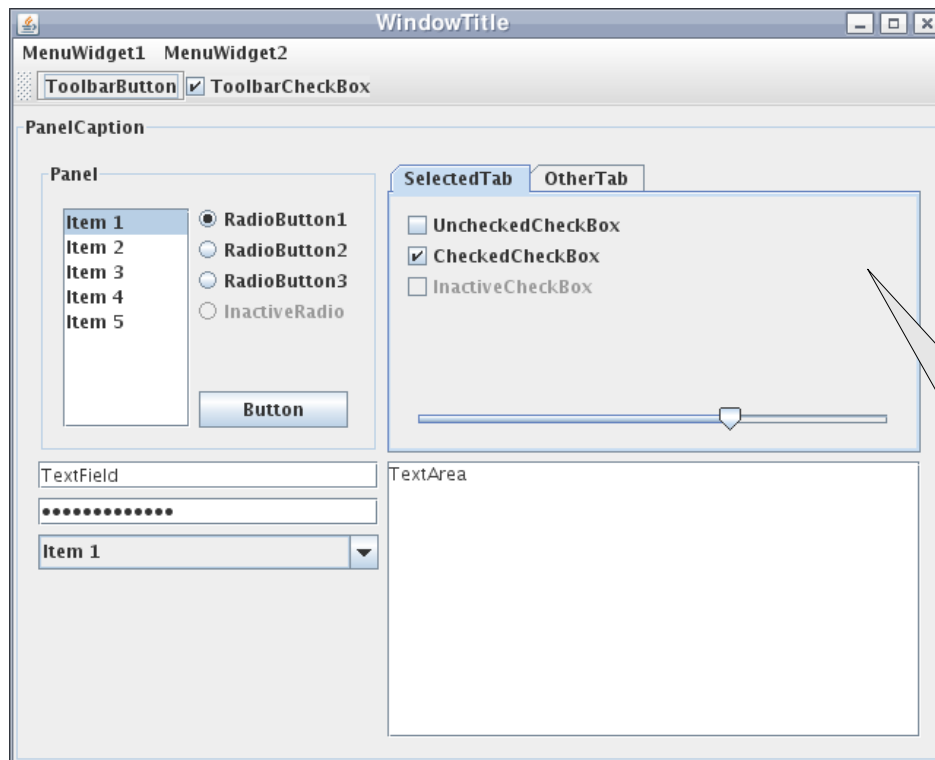
JFrame  
vazio



JDialog



JFrame



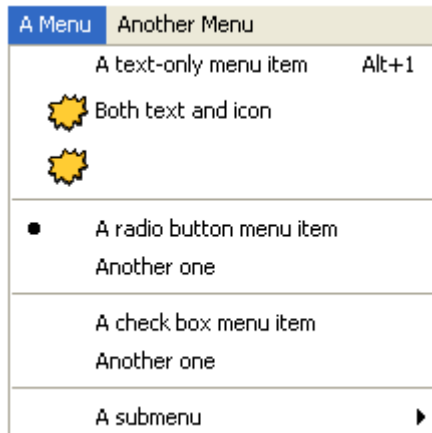
JFrame  
contendo  
outros  
objetos

# Categorias de classes

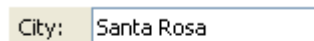
- **Componentes interativos básicos** (ex: JButton, JMenu, etc.): são componentes para entrada/saída



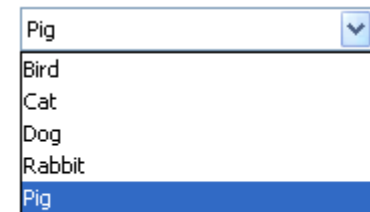
[JButton](#)



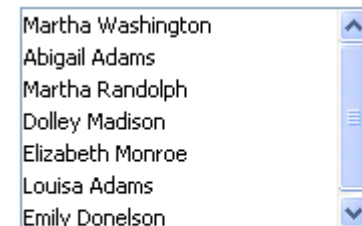
[JCheckBox](#)



[JTextField](#)



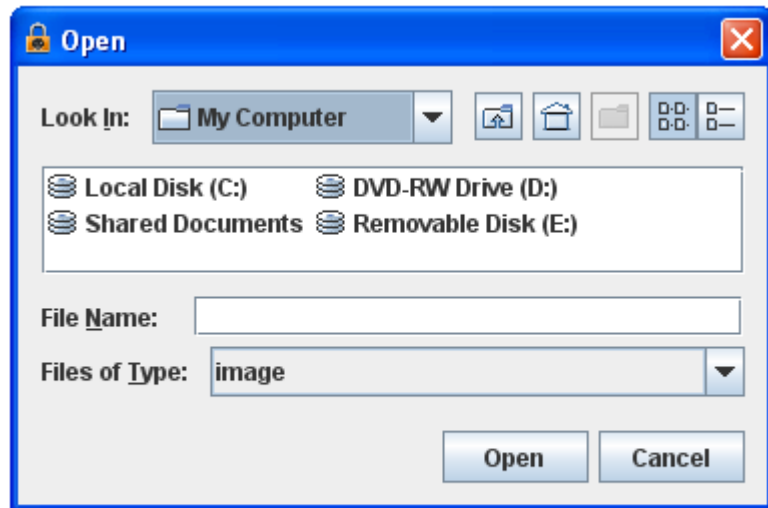
[JComboBox](#)



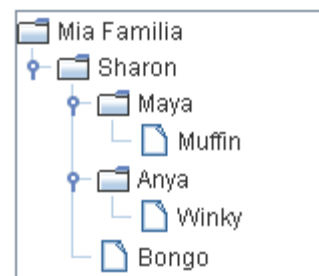
[JList](#)

# Categorias de classes

- **Componentes de alto nível** (ex: JFileChooser, JTable, etc.): são componentes "sofisticados" para interação com o usuário



JFileChooser



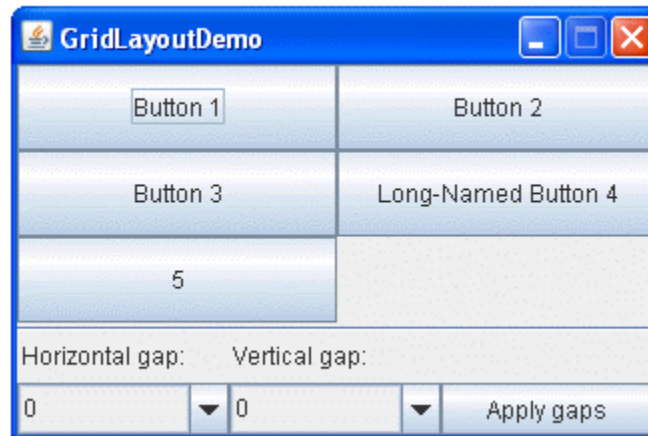
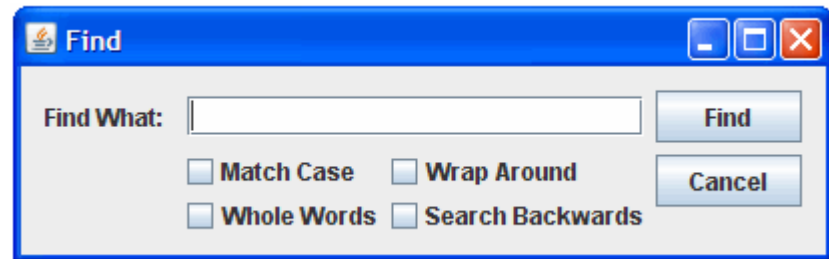
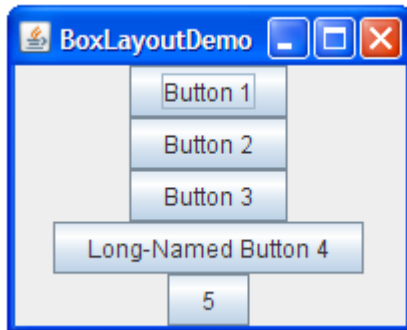
JTree

| Host             | User               | Password    | Last Modified |
|------------------|--------------------|-------------|---------------|
| Biocca Games     | Freddy             | !#asf6Awwzb | Mar 16, 2006  |
| zabble           | ichabod            | Tazb!34\$fZ | Mar 6, 2006   |
| Sun Developer    | fraz@hotmail.co... | AasV541!fbZ | Feb 22, 2006  |
| Heirloom Seeds   | shams@gmail....    | bkz[ADF78!  | Jul 29, 2005  |
| Pacific Zoo Shop | seal@hotmail.c...  | vbAf1 24%z  | Feb 22, 2006  |

JTable

# Categorias de classes

- **Layout Managers:** são classes que ajudam a organizar componentes em um container



Fonte:

<http://download.oracle.com/javase/tutorial/uiswing/layout/visual.html>

# Interação

- **Eventos/ações:** são classes que representam a interação propriamente dita (ver interface ActionListener)

```
class ArrayListGUI extends JFrame {  
    ...  
    buttonLimpar = new javax.swing.JButton();  
    buttonLimpar.setText("Limpar");  
    buttonLimpar.addActionListener(new java.awt.event.ActionListener() {  
        public void actionPerformed(java.awt.event.ActionEvent evt) {  
            buttonLimparActionPerformed(evt);  
        }  
    });  
    ...  
    private void buttonLimparActionPerformed(java.awt.event.ActionEvent evt) {  
        textAno.setText("");  
        comboSemestre.setSelectedIndex(0);  
        textNome.setText("");  
        textNota.setText("");  
    }  
}
```

