



# Interface Gráfica e Banco de Dados em Java

## Componentes GUI – Parte III

# Licença para uso e distribuição

Este material está disponível para uso não-comercial e pode ser derivado e/ou distribuído, desde que utilizando uma licença equivalente.



Atribuição-Uso Não-Comercial-  
Compartilhamento pela mesma  
licença, versão 2.5

<http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt>

Você pode copiar, distribuir, exibir e executar a obra, além de criar obras derivadas, sob as seguintes condições: (a) você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante; (b) você não pode utilizar esta obra com finalidades comerciais; (c) Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

# Objetivos desta parte

- Explicar a arquitetura de modelos separados do Swing, apresentando as classes de modelo;
- Mostrar em detalhes o funcionamento dos dois componentes mais complexos do Swing:
  - Árvores (JTree);
  - Tabelas (JTable).

Grupo de Usuários de Java do Estado do Espírito Santo

# Arquitetura de sistemas

- Deve ser escolhida quando implementamos um sistema;
- Exemplo – arquitetura de uma única camada:



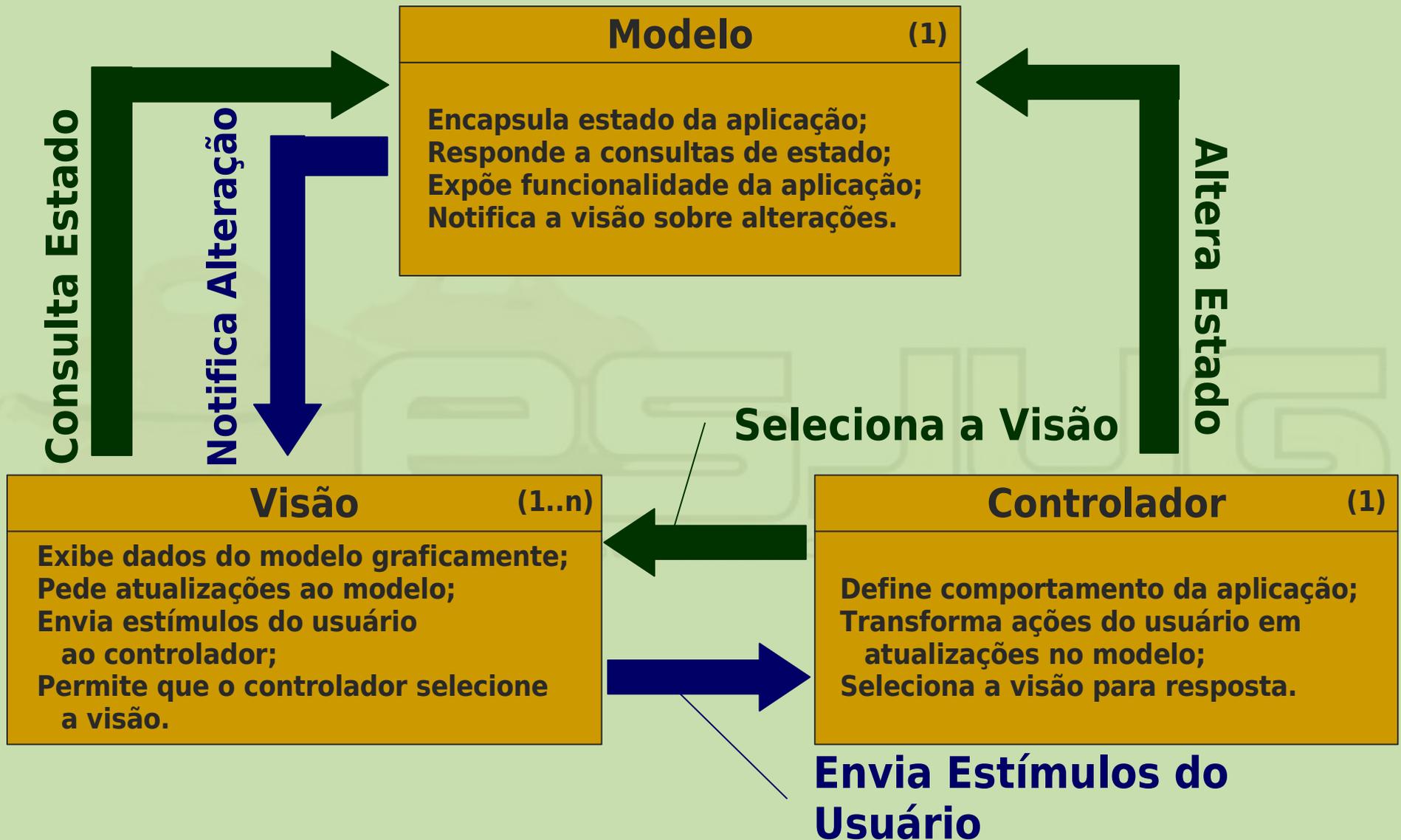
# Uma camada x N camadas

- Sem complicações para desenvolver;
- Difícil manutenção:
  - Código desorganizado;
  - Difícil depuração;
  - Alterações em qualquer camada afetam todas as outras.
- Difícil reutilização.
- Maior complexidade no desenvolvimento;
- Manutenção mais simples:
  - Código organizado;
  - Depuração isolada de camadas;
  - Alterações numa camada não afetam outras.
- Facilita o reuso.

# Model-View-Controller (MVC)

- Desenvolvido pelo Xerox PARC para o Smalltalk, em 1978;
- Objetivo: mapear entrada-processamento-saída em GUIs para OO: controle-modelo-visão;
- Usado para:
  - Criação de componentes GUI reutilizáveis;
  - Estruturação da aplicações *Web* (padrão de projeto *Front Controller*).

# Estrutura do MVC



Legenda: **Eventos** Chamada de Métodos

# Separable Model Architecture (SMA)

- O Swing utiliza a abordagem MVC para seus componentes GUI;
- No entanto, a visão e o controlador situam-se no mesmo componente;
- Muitos consideram, portanto, que não é MVC;
- Então, dá-se o nome de Separable Model Architecture – Arquitetura de Modelo Separável.

# SMA no Swing

## JSlider



### Controller

3) Atualiza seu estado, consultando os valores.



### View

### DefaultBoundedRangeModel

- minimum : int  
- maximum : int  
- value : int

+ getMinimum() : int  
+ setMinimum() : int  
+ getMaximum() : int  
+ setMaximum() : int  
+ getValue() : int  
+ setValue() : int

### Model

2) Notifica visão sobre a alteração

1) Aplicação altera dados no modelo.

# Os modelos (pacote javax.swing)

- Modelos são especificados por interfaces:
  - `ButtonModel`, `ComboBoxModel`, `ListModel`, ...
- A maioria dos componentes Swing possui modelos padrão (*default*):
  - `DefaultButtonModel`, `DefaultComboBoxModel`, `DefaultListModel`, ...
- Na maioria dos casos, não precisamos utilizar os modelos;
- Em outros casos, precisamos usá-los para qualquer coisa que fuja do trivial.

# JList e ListModel

- Uma lista JList possui um modelo ListModel;
- Criação automática do modelo:

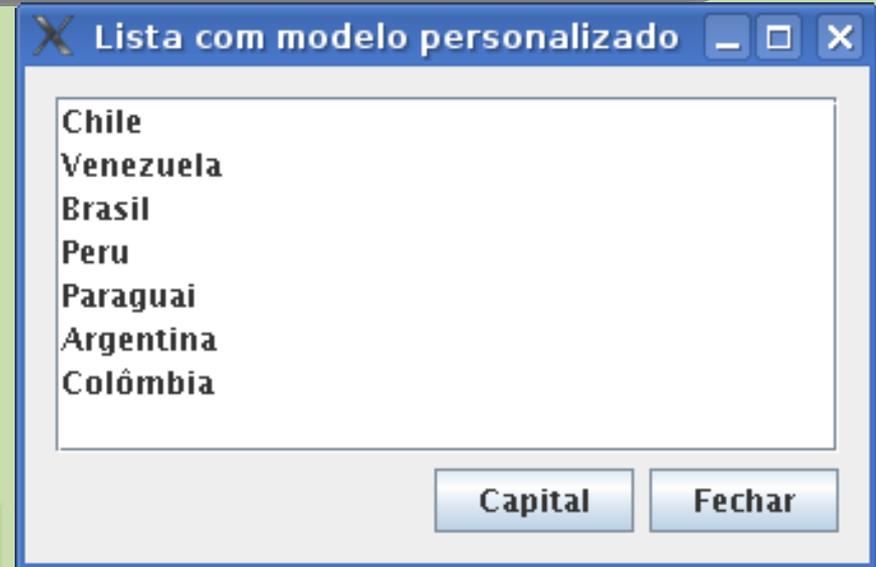
```
String[] cores = new String[] { "Preto",  
"Branco", "Azul", "Vermelho" };  
  
// Construtor cria modelo default a partir  
// dos elementos do vetor:  
JList listaCores = new JList(cores);  
  
// Obtém o modelo criado por padrão:  
ListModel modelo = listaCores.getModel();
```

# Criando nossos próprios modelos

- Podemos criar nossos próprios modelos:
  - Instanciando `DefaultListModel`;
  - Criando nossas próprias classes que implementem `ListModel`;
  - Criando nossas próprias classes estendendo `AbstractListModel`.
- Atribuimos o modelo à lista:
  - Via construtor: `JList l = new JList(modelo);`
  - Via método: `l.setModel(modelo).`

# DefaultListModel

- Construa a interface exibida ao lado;
- Crie uma classe que represente uma cidade, registrando seu nome;
- Crie uma classe que represente um país, registrando seu nome e capital;
- Adicione vários países à `JList` por meio de um `DefaultListModel`;
  - Use o método `addElement()`.



# Implementando ListModel

- Faça novamente o mesmo exercício, trocando o `DefaultListModel` por uma classe sua;
- Tal classe deve implementar `ListModel`;
- Métodos importantes:
  - `getElementAt(i)`: retorna o elemento do índice `i`;
  - `getSize()`: retorna o tamanho da lista.
- Possui dois outros métodos:
  - `addDataListener()`, `removeDataListener()`;
  - Um ouvinte de dados deve ser notificado sempre que ocorrem mudanças no modelo.

# Estendendo `AbstractListModel`

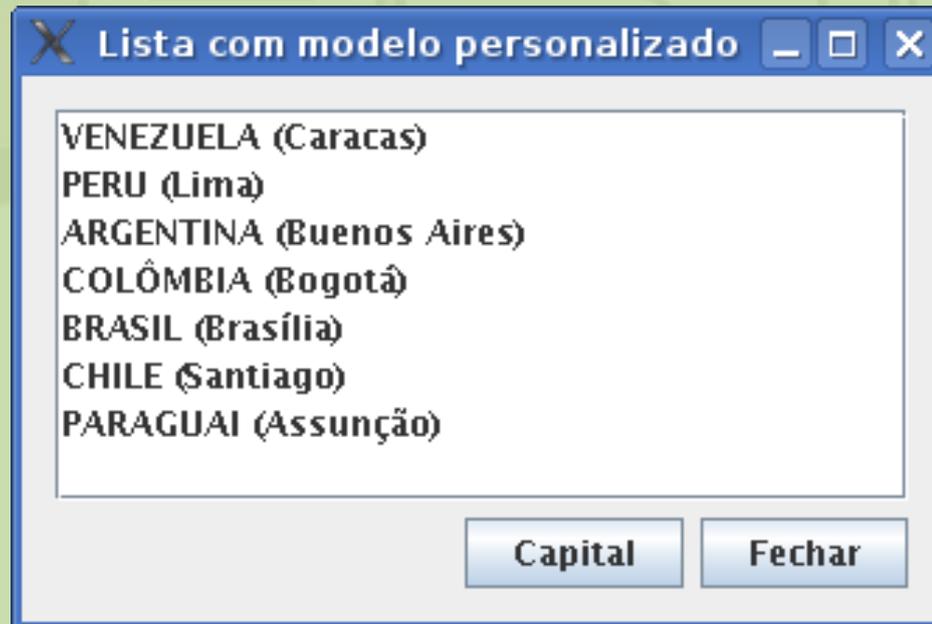
- Faça novamente o mesmo exercício, desta vez sua classe deve estender `AbstractListModel`;
- A classe abstrata provê implementação padrão para os métodos relacionados aos ouvintes;
- Faltam implementar os métodos principais:
  - `getElementAt(i)`: retorna o elemento do índice `i`;
  - `getSize()`: retorna o tamanho da lista.

## Desvinculando aparência e objeto

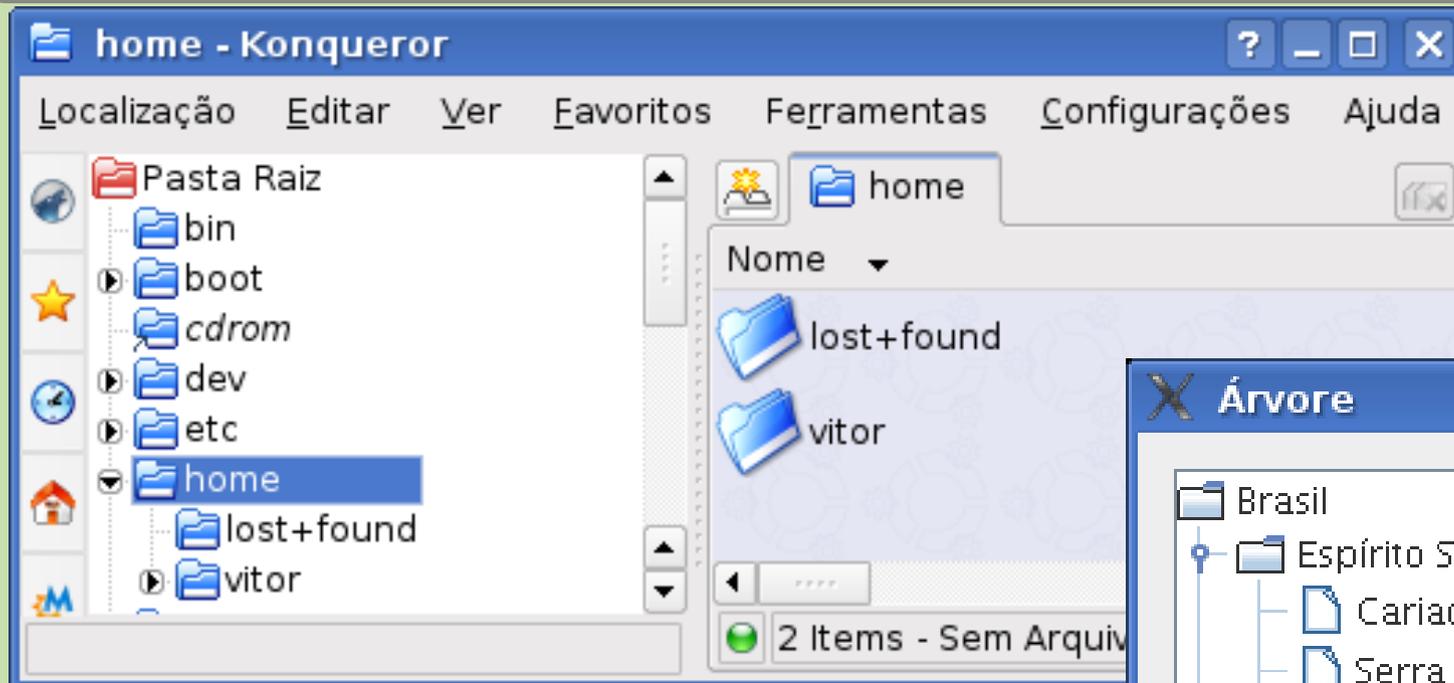
- O método `getElementAt(i)` retorna o elemento a ser exibido na posição `i`;
- Seu retorno é `Object`. Será chamado `toString()` para exibi-lo na lista;
- Também é usado para retornar os objetos que iremos manipular (ex.: exibir a capital);
  - Direto ou via `getSelectedValues()`.

# Desvinculando aparência e objeto

- Podemos desvincular o que é exibido do que é retornado para manipulação:
  - `getElementAt()` retornaria uma `String` personalizada;
  - `getPaisAt()` retornaria o país.

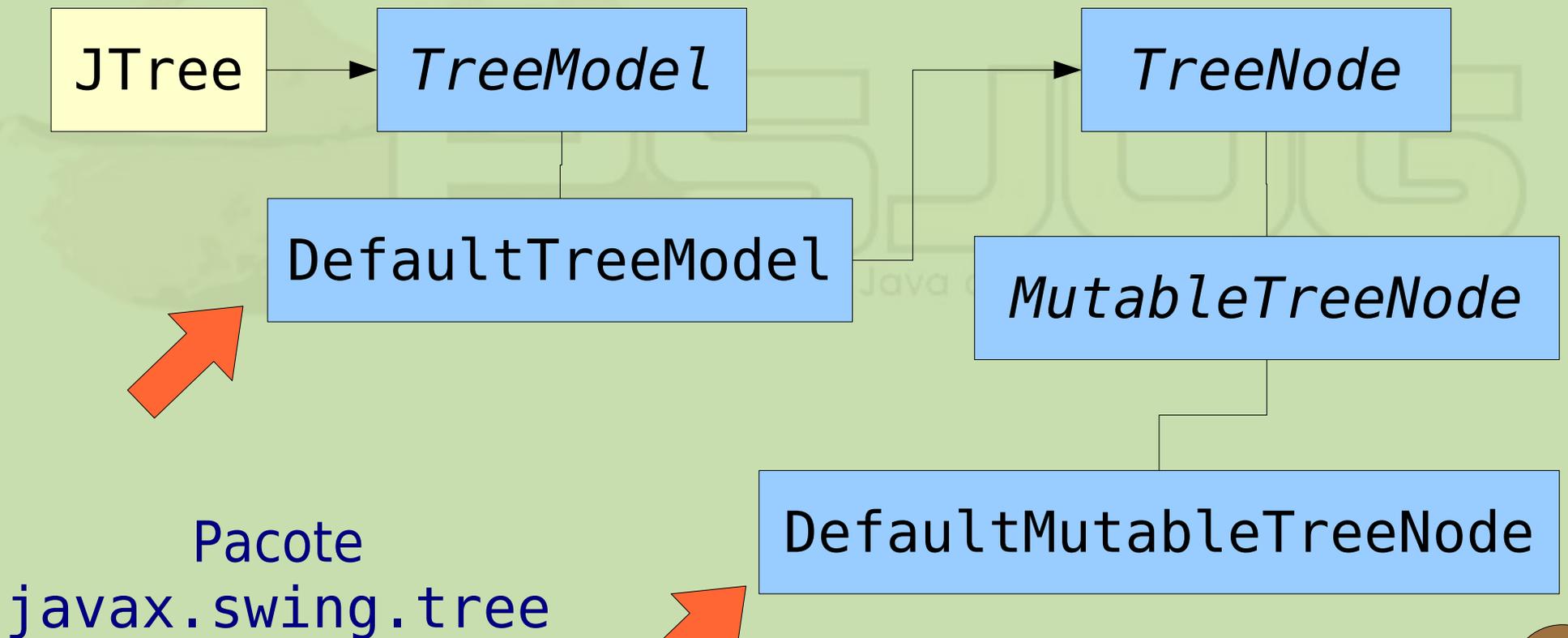


# Árvores!



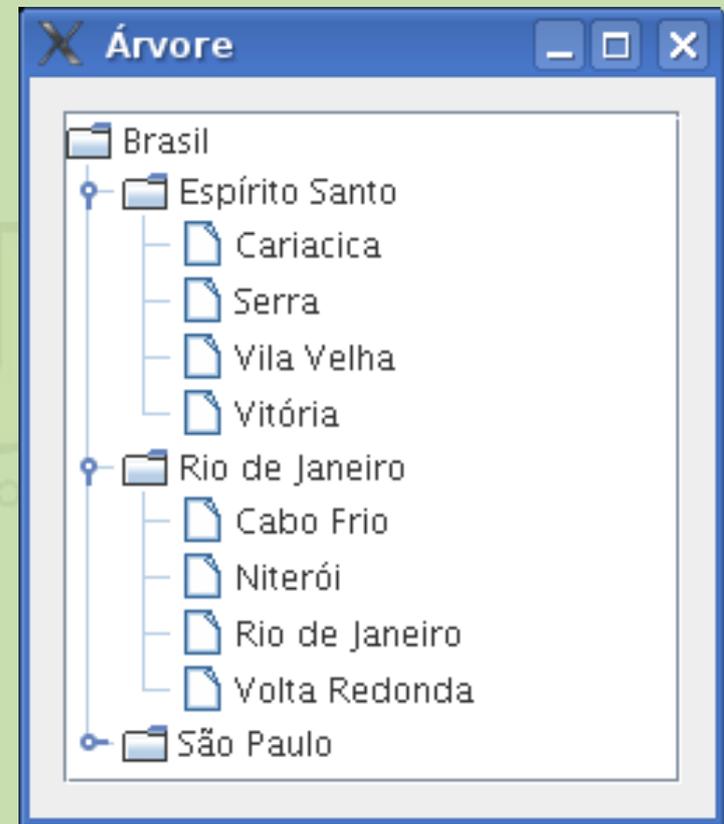
# JTree

- Instâncias de `javax.swing.JTree` representam árvores em interfaces gráficas;
- Construídas a partir de seus modelos:



# Árvore simples

- Crie uma interface similar à exibida abaixo:
- Use `DefaultMutableTreeNode`:
  - Use a classe `Pais` dada;
  - Passe o conteúdo do nó no construtor;
  - Adicione um filho a um nó com o método `add()`.
- Crie um `DefaultTreeModel`:
  - Passe o nó raiz como parâmetro do construtor;
  - Use `setModel()` no objeto `JTree`.



# JTree: algumas propriedades

- JTree:
  - `showsRootHandle`: indica se o item da raiz deve mostrar uma alça de abrir / fechar;
  - `rootVisible`: indica se o item da raiz deve ser mostrado.
- `TreeNode`:
  - `isLeaf()`: indica se o nó é folha (não tem filhos);
  - `getParent()`: nó pai;
  - `getAllowsChildren()`: se permite adicionar filhos;
  - `getChildCount()`, `getChildAt()`: filhos.

# Caminhos da árvore

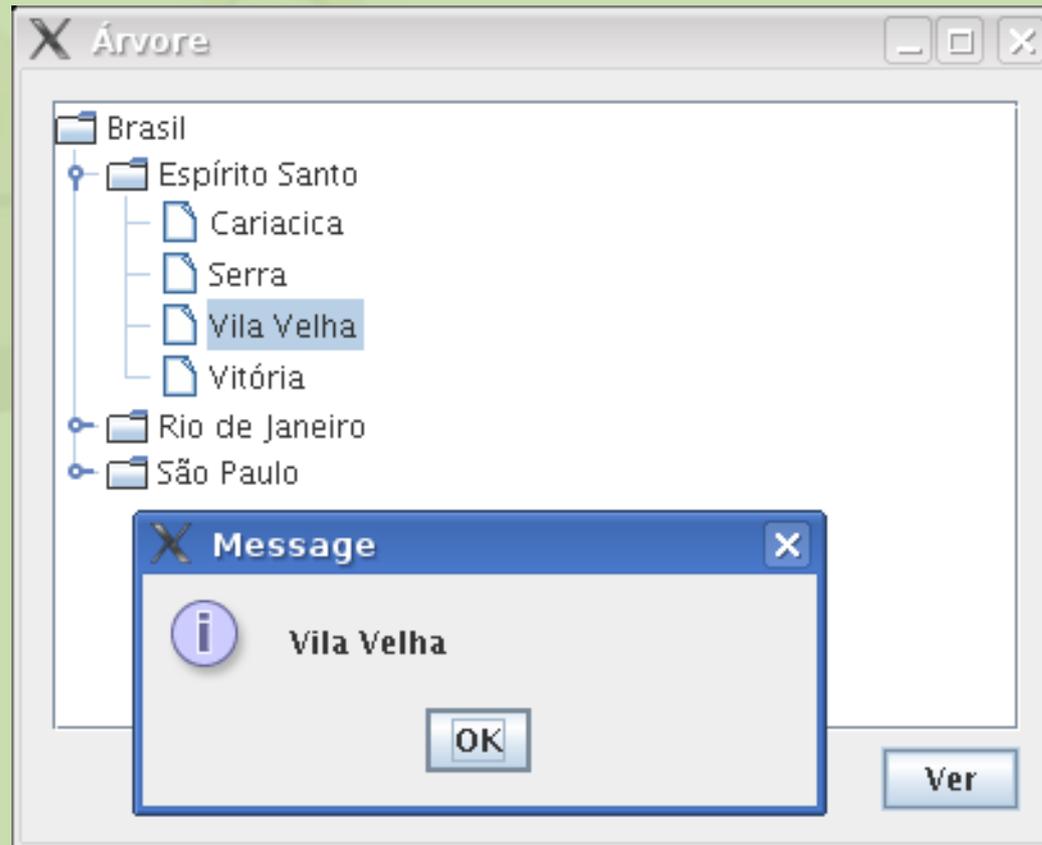
- O método `getSelectionPath()` de `JTree` retorna o caminho de nós do nó selecionado;
- Um caminho de nós inclui todos os nós da raiz até o nó em questão;
- Instância de `javax.swing.tree.TreePath`:
  - `getLastPathComponent()`: último nó do caminho;
  - `getPath()`: todos os nós do caminho;
  - `getPathComponent(i)`: nó do índice `i`;
  - `getPathCount()`: número de nós do caminho.

# Modos de seleção

- Assim como listas, árvores possuem diferentes modos de seleção (`TreeSelectionMode`):
  - `SINGLE_TREE_SELECTION`;
  - `CONTIGUOUS_TREE_SELECTION`;
  - `DISCONTIGUOUS_TREE_SELECTION` (padrão).
- Quando usamos seleção múltipla, temos que obter todos os caminhos selecionados:
  - `TreePath[] getSelectedPaths()`.

# Experimente:

- Adicione um botão “Ver” à GUI;
- Ao clicar no botão, exiba uma mensagem (JOptionPane) com o nome do país, estado ou cidade.



# Inserindo nós dinamicamente

- Antes de montarmos o modelo, criamos uma estrutura de nós com `add()`;
- Depois da árvore pronta, não devemos usar mais o método `add()` do nó;
- Ao invés disso, usaremos um método do modelo – `insertNodeInto(novoNo, noPai, pos)`:
  - `novoNo`: o nó que será inserido;
  - `noPai`: o nó abaixo do qual será inserido;
  - `pos`: posição onde o novo nó será inserido. Para colocar no final, informe `noPai.getChildCount()`.

## Experimente:

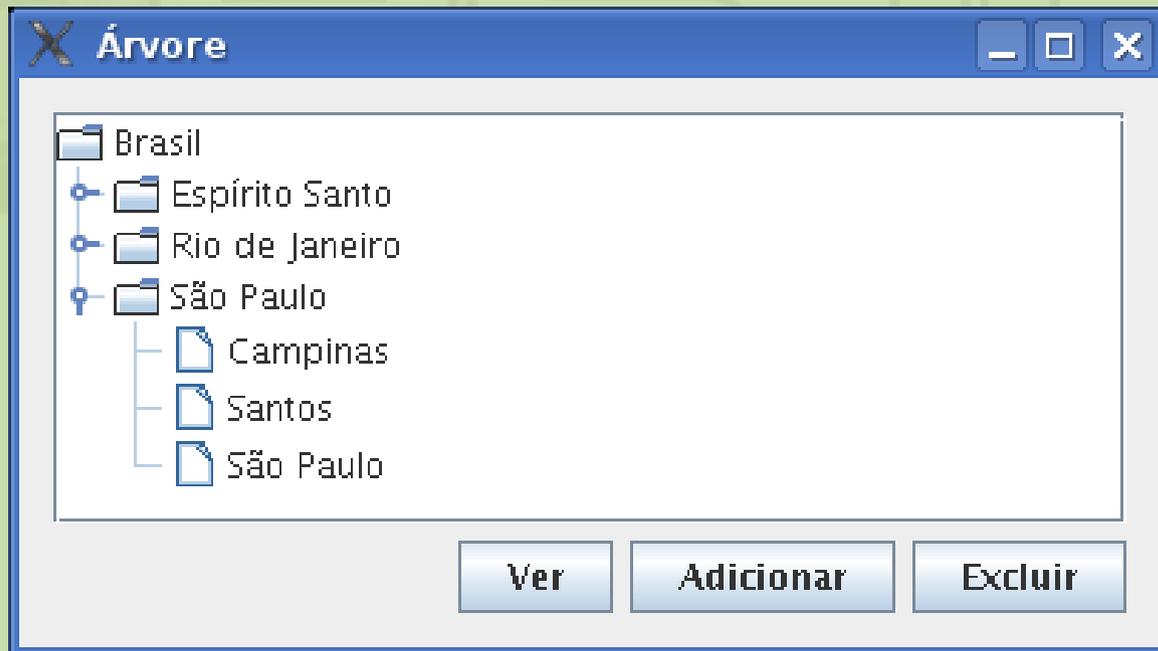
- Coloque um botão “Adicionar” na GUI;
- Ao clicar no botão, adicione um novo nó:
  - Se um país estiver selecionado, adicionar um novo estado a ele;
  - Se um estado estiver selecionado, adicionar uma nova cidade a ele;
  - Se uma cidade estiver selecionada, adicionar uma nova cidade do mesmo estado.
- Atalho para obter o nó selecionado:  
`tree.getLastSelectedPathComponent()`.

## Detalhes de interface

- Quando um novo nó é adicionado, se seu pai estiver fechado ele não aparece;
- O mesmo acontece se ele encontra-se fora do campo de visão num painel de rolagem;
- Dois métodos podem ser usados para corrigir estes problemas:
  - `arvore.makeVisible(caminho);`
  - `arvore.scrollPathToVisible(caminho).`

# Removendo nós dinamicamente

- Outro método do modelo serve para remover um nó – `removeNodeFromParent (no)`;
- Experimente:
  - Adicione um botão “Excluir” que remova um nó de estado ou de cidade da árvore.

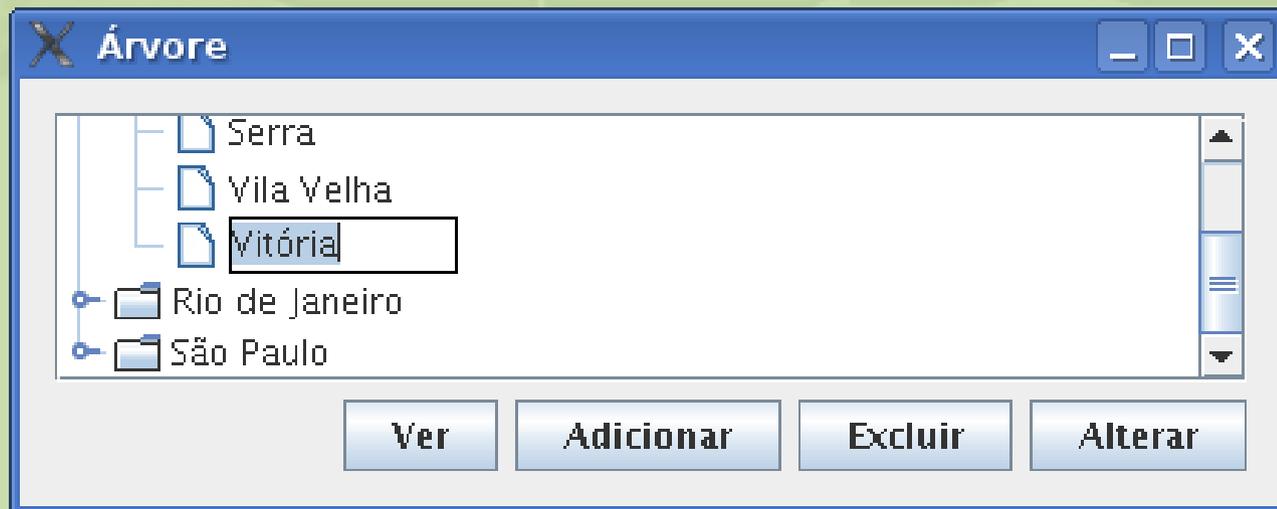


# Alterando nós dinamicamente

- Para alterar um nó, basta que alteremos seu objeto de usuário: `setUserObject(obj);`
- No entanto, alterações devem ser notificadas para que a interface se redesenhe:
  - Use o método do modelo: `nodeChanged(no);`
- Experimente:
  - Adicione um botão “Alterar”;
  - Pergunte o novo nome para o país, estado ou cidade;
  - Atualize o nó.

# Árvores editáveis

- Quando a árvore contém somente strings, podemos fazer com que seja editável:
  - `arvore.setEditable(true)`.
- Desta forma, o usuário pode efetuar um duplo-clique no nó e editá-lo diretamente;
- O objeto adicionado será uma nova `String`!



# Enumerações de nós

- Para navegar por todos os nós de uma árvore ou sub-árvore, podemos obter uma enumeração:
- Na classe `DefaultMutableTreeNode`:
  - `breadthFirstEnumeration()`: em amplitude;
  - `depthFirstEnumeration()`: em profundidade;
  - `postorderEnumeration()`: em pós-ordem;
  - `preorderEnumeration()`: em pré-ordem.
- Retornam um `java.util.Enumeration`. Os métodos `hasMoreElements()` e `nextElements()` manipulam a enumeração.

# Renderizador de células

- Quando a árvore exibe seus nós, delega a tarefa a um renderizador de células;
- Trocando o renderizador, podemos, por exemplo, alterar os ícones dos nós:

```
DefaultTreeCellRenderer renderer;  
renderer = new DefaultTreeCellRenderer();  
  
renderer.setLeafIcon(icon1);  
renderer.setClosedIcon(icon2);  
renderer.setOpenIcon(icon3);  
  
tree.setCellRenderer(renderer);
```

# Personalizando para cada nó

- Adicionando o renderizador *default* para toda a árvore altera-a uniformemente;
- Como fazer para alterar alguns nós específicos (ex.: ícone diferente para a raiz)?
- Criamos nosso próprio renderizador de células, estendendo o *default* e sobrescrevendo:  
`getTreeCellRendererComponent()`
- Assim como em `paintComponent()`, devemos chamar o método na superclasse primeiro.

# Eventos de árvore

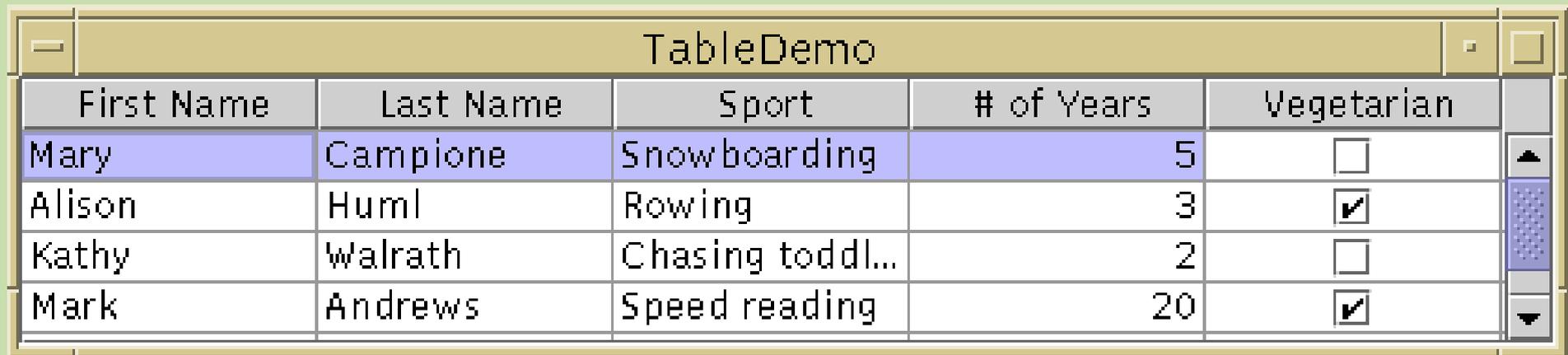
- `TreeSelectionListener`:
  - `valueChanged()`: mudou o nó selecionado.
- `TreeExpansionListener`:
  - `treeCollapsed()`: um nó foi contraído;
  - `treeExpanded()`: um nó foi expandido.
- `TreeWillExpandListener`:
  - Idem, acionado antes da expansão ou contração;
  - Pode lançar `ExpandVetoException` para vetar a ação.

# Experimente:

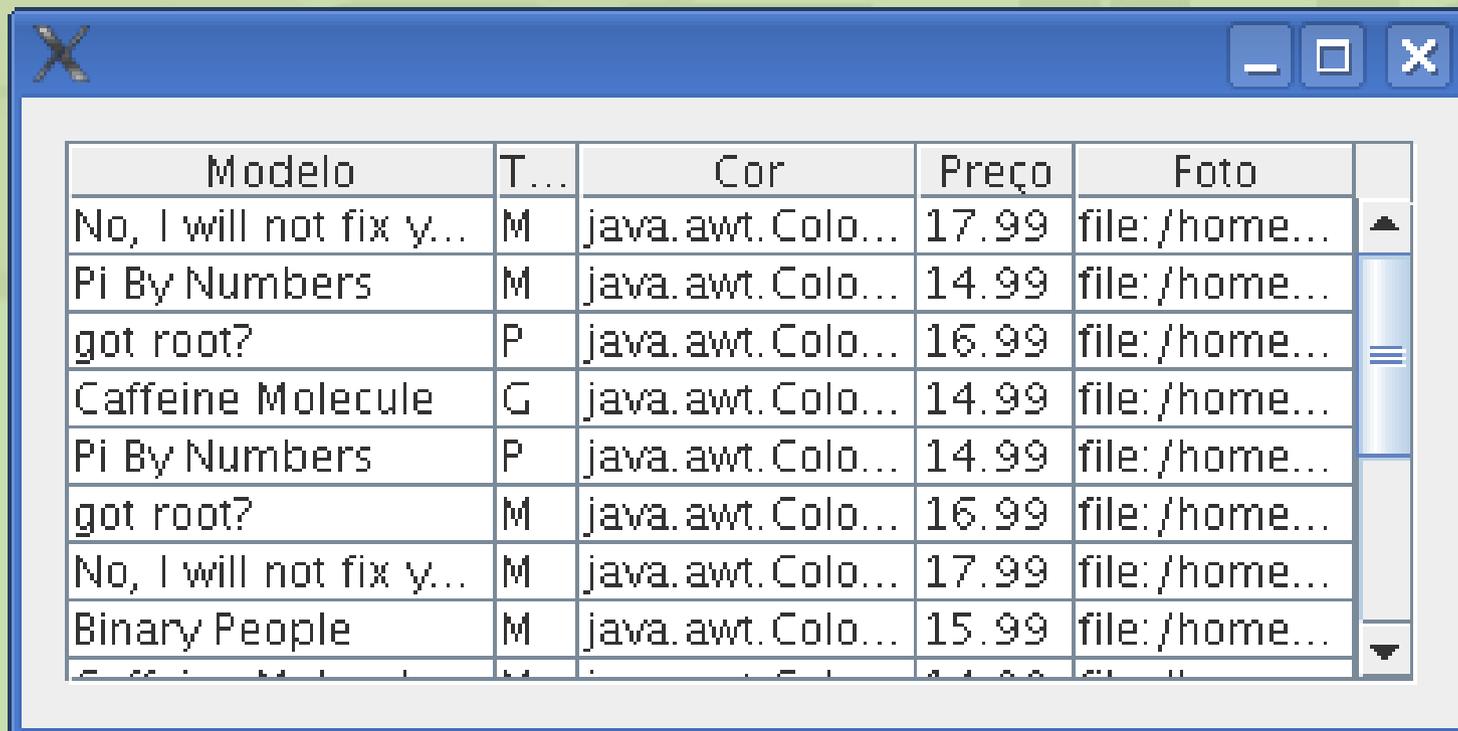
- Adicione um rótulo à direita da árvore;
- Ao selecionar um nó, procure uma imagem JPEG com o mesmo nome em /resources/fotos;
- Se encontrar, exiba a imagem no rótulo.



# Tabelas!



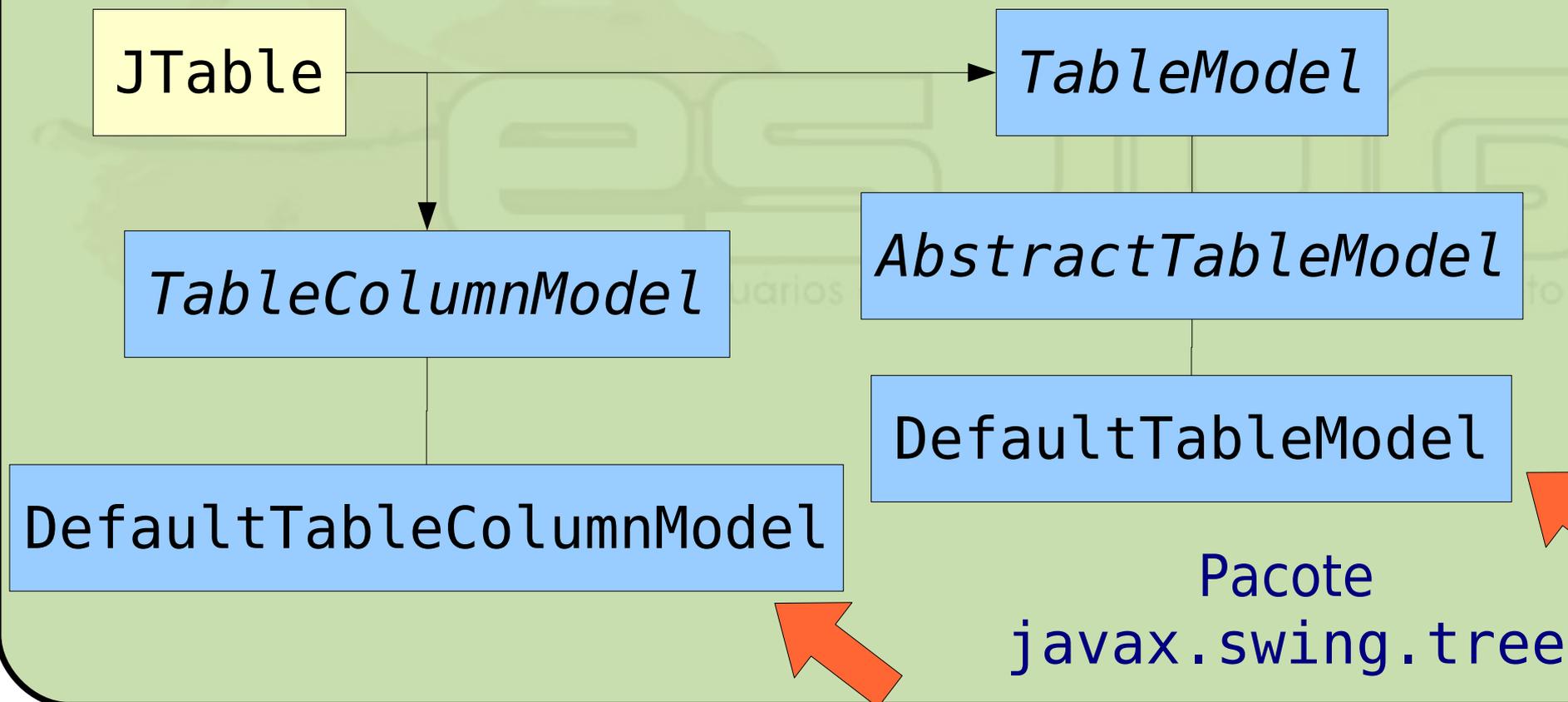
First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	<input type="checkbox"/>
Alison	Huml	Rowing	3	<input checked="" type="checkbox"/>
Kathy	Walrath	Chasing toddl...	2	<input type="checkbox"/>
Mark	Andrews	Speed reading	20	<input checked="" type="checkbox"/>



Modelo	T...	Cor	Preço	Foto
No, I will not fix y...	M	java.awt.Colo...	17.99	file:/home...
Pi By Numbers	M	java.awt.Colo...	14.99	file:/home...
got root?	P	java.awt.Colo...	16.99	file:/home...
Caffeine Molecule	G	java.awt.Colo...	14.99	file:/home...
Pi By Numbers	P	java.awt.Colo...	14.99	file:/home...
got root?	M	java.awt.Colo...	16.99	file:/home...
No, I will not fix y...	M	java.awt.Colo...	17.99	file:/home...
Binary People	M	java.awt.Colo...	15.99	file:/home...

# JTable

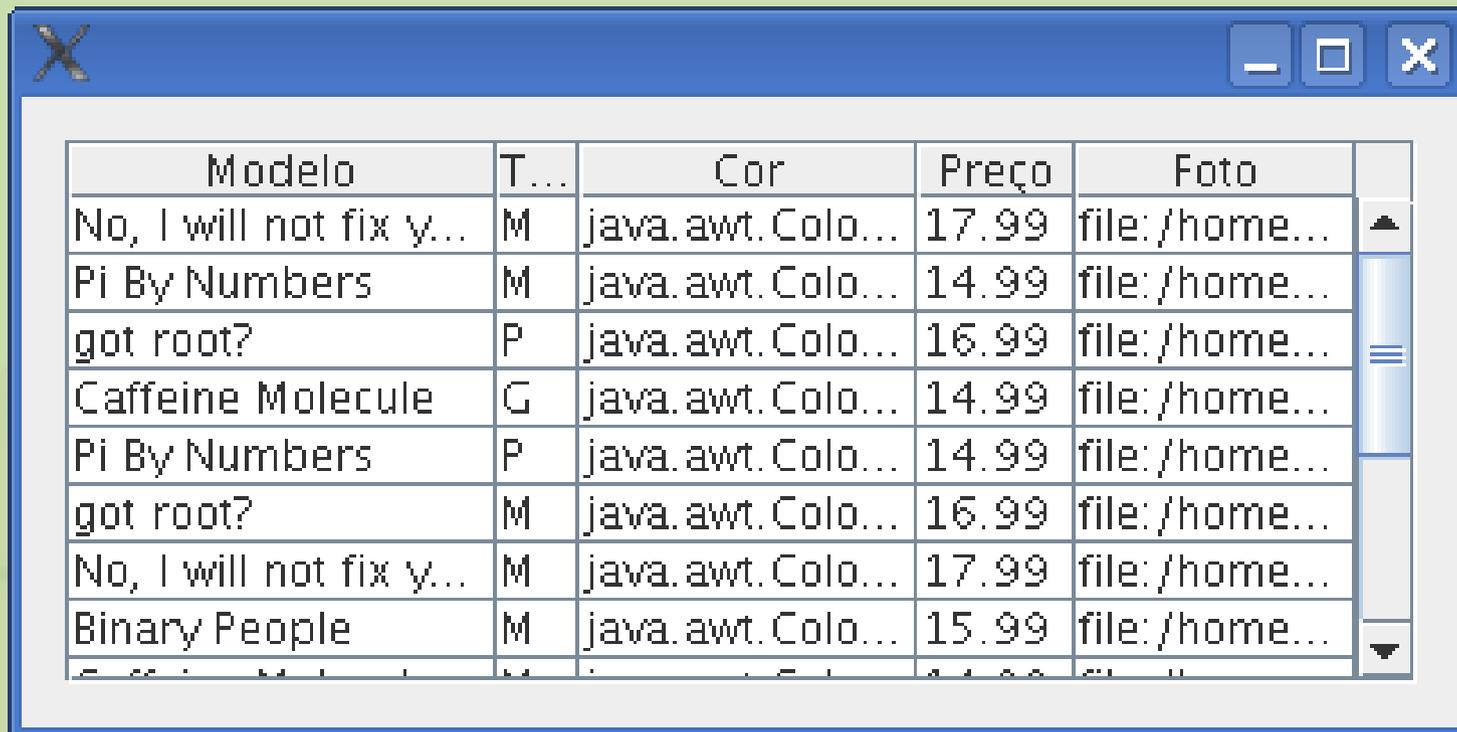
- Instâncias de `javax.swing.JTable` representam tabelas em interfaces gráficas;
- Construídas a partir de seus modelos:



# Tabela simples

- Use `DefaultTableModel`:
  - Coloque os dados a serem exibidos na tabela numa matriz de objetos;
  - Coloque os nomes das colunas num vetor de strings;
  - Passe os dois no construtor do modelo;
  - Atribua à tabela: `tabela.setModel()`.
- Experimente:
  - Use a lista de produtos da classe `Camisa` dada;
  - Converta os atributos das camisas em colunas da matriz.

# Características



Modelo	T...	Cor	Preço	Foto
No, I will not fix y...	M	java.awt.Colo...	17.99	file:/home...
Pi By Numbers	M	java.awt.Colo...	14.99	file:/home...
got root?	P	java.awt.Colo...	16.99	file:/home...
Caffeine Molecule	G	java.awt.Colo...	14.99	file:/home...
Pi By Numbers	P	java.awt.Colo...	14.99	file:/home...
got root?	M	java.awt.Colo...	16.99	file:/home...
No, I will not fix y...	M	java.awt.Colo...	17.99	file:/home...
Binary People	M	java.awt.Colo...	15.99	file:/home...

- Redimensionamento e movimento das colunas;
- Edição das células.

# Configurações básicas

- `cellSelectionEnabled`: se pode selecionar uma única célula (ao invés da linha toda);
- `dragEnabled`: se pode arrastar as colunas;
- `setRowHeight`: altura da linha;
- `showGrid`: se deve mostrar as linhas internas;
- `getColumnModel().getColumn(C)` obtém a coluna de índice C:
  - `setMinWidth()`, `setPreferredWidth()`, `setMaxWidth()`, `setResizable()`, etc.

# Implementando seu próprio modelo

- A melhor prática é implementar seu próprio modelo, e não usar o *default*;
- Deve implementar `TableModel`;
- Melhor estender `AbstractTableModel`;
  - `getRowCount()`: número de linhas;
  - `getColumnCount()`: número de colunas;
  - `getValueAt(L, C)`: valor apresentado na linha L e coluna C;
  - `getColumnName(C)`: título da coluna C.
- Atribua o modelo à tabela (`setModel()`).

# Experimente:

- Crie uma subclasse de `AbstractTableModel`;
  - Deve receber a coleção de camisas, transformá-la numa lista e armazená-la como atributo;
  - `getRowCount()` retorna o tamanho da lista;
  - `getColumnCount()` retorna 5;
  - `getValueAt(L, C)` retorna um dos atributos da camisa do índice L, dependendo do valor de C:
    - Coluna 0 = modelo; coluna 1 = tamanho; coluna 2 = cor; coluna 3 = preço; coluna 4 = foto.
  - `getColumnName(C)` retorna o nome da coluna dependendo do valor de C (idem acima).

# Formatando a saída

- Implementando um modelo personalizado, separamos os dados da apresentação;
- No método `getValueAt()` podemos formatar os dados das células:
  - Ex.: formatar o preço do produto com um formatador de valores monetários.
- O método `getColumnClass(C)` retorna a classe da coluna `C`;
  - Valores são exibidos de forma padronizada para algumas classes (ex.: `ImageIcon` e `Boolean`).
- Experimente!

# Renderizador de células

- Indicar a classe faz com que o Swing escolha renderizadores apropriados;
- Porém algumas classes não possuem renderizadores padronizados:
  - Ex.: `java.awt.Color`.
- Para estas classes, podemos criar renderizadores personalizados;
  - Implemente a interface `TableCellRenderer`;
  - Retorne um `Component` via `getTableCellRendererComponent()`.

## Experimente:

```
class ColorTableCellRenderer implements
TableCellRenderer {
    private JPanel panel = new JPanel();

    public Component
    getTableCellRendererComponent(JTable
    table, Object value, boolean isSelected,
    boolean hasFocus, int row, int column)
    {
        panel.setBackground((Color)value);
        return panel;
    }
}
```

# Ordenando a tabela

- A ordem dos elementos na tabela depende da ordem dos objetos na lista do modelo;
- Experimente:
  - Adicione um evento de mouse, detectando cliques no cabeçalho da tabela (`getTableHeader()`);
  - Ordene a tabela de acordo com o valor daquela coluna (use comparadores).

# Código-fonte

- Evento de mouse (classe SortListener):

```
public void mouseClicked(MouseEvent event) {  
    int coluna =  
        table.getTableHeader().columnAtPoint(event.getPoint());  
    int indice = table.convertColumnIndexToModel(coluna);  
    modelo.ordena(indice);  
}
```

- Adição do evento:

```
MouseListener listener = new SortListener();  
table.getTableHeader().addMouseListener(listener);
```

- Ordenação e atualização da tabela:

```
Comparator<Camisa> comparador = null;  
/* Cria o comparador de acordo com a coluna. */  
Collections.sort(produtos, comparador);  
fireTableDataChanged();
```

# Edição de células

- Método `isCellEditable(L, C)` de `TableModel`:
  - Indica se a célula na linha L e coluna C pode ser editada;
  - Deve ser sobrescrito por nosso modelo.
- Experimente:
  - Faça com que as células da coluna de preço sejam editáveis, mas não as demais;
  - Rode a aplicação e dê um duplo-clique na célula de preço que quiser editar;
  - Note que o valor não permanece...

# Efetivando a alteração

- Quando uma célula é alterada, a árvore chama o método `setValueAt(valor, L, C)` no modelo:
  - `valor`: valor digitado na célula;
  - `L` e `C`: linha e coluna.
- Experimente:
  - Implemente a mudança de preço da camisa no modelo personalizado;
  - Use um formatador para converter string em double.

# Editores customizados

- Quando editamos um campo, um editor padrão (ex.: caixa de texto para strings) é utilizado;
- Podemos indicar um outro componente como editor via `TableCellEditor`:
  - Criando nossa própria implementação;
  - Usando `DefaultCellEditor` para indicar uma checkbox, campo texto ou combo box como editor.

## Experimente:

- Coloque um editor customizado para tamanho: uma combo box com os tamanhos P, M e G;

```
JComboBox combo =  
    new JComboBox(new Character[] { 'P', 'M', 'G' });  
TableCellEditor editor = new DefaultCellEditor(combo);  
TableModel tcm = table.getColumnModel();  
TableColumn col = tcm.getColumnModel(1);  
col.setCellEditor(editor);
```

- Não se esqueça de:
  - Liberar a edição em `isEditable()`;
  - Efetivar a edição em `setValueAt()`.

# Há muito mais para experimentar...

- JTable é o componente mais complexo do Swing;
- É possível fazer:
  - Mudar o modo de seleção da tabela;
  - Detectar seleções do usuário;
  - Incluir e remover linhas da tabela;
  - Especificar dicas de ferramenta (*tool tip*) para células e cabeçalho;
  - Etc.
- Estude!

# Conclusões

- Vimos nesta parte do curso:
  - A arquitetura de modelo separável do Swing;
  - Os componentes JTree e JTable.
- Vimos neste curso:
  - Como criar janelas, painéis e preenchê-los com outros componentes usando *layout managers*;
  - Como funcionam vários componentes Swing;
  - Como utilizar a IDE NetBeans para criar GUIs;
  - Como tratar eventos diversos dos componentes.