# Unidade 5

Esta aula será destinada a apresentar ferramentas do Java Swing, que irá tornar nossos programas em Java mais visualmente atraentes. Temos diversas telas exemplificadas, mas a livre criatividade é bem-vinda e trará aplicações mais bonitas e funcionais. Atenção especial deve ser dada nas inserções de códigos e os erros aparentes, que as vezes devem ser ignorados, mas na maioria delas devem ser corrigidos.

## Aula 1 - Definindo as Views

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Manipular e fazer bom uso do interface builder
* Criar e utilizar telas no aplicativo

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 20 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 50 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Protótipo das views;
* Diagrama de classes das Views;
* Criando as telas.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 1.1 - Protótipo das Views

É interessante apresentar uma atividade para prototipagem do programa proposto. Lembrando que, como é um protótipo, não se preocupe com os detalhes e não gaste muito tempo nesta tarefa.

#### - Tela principal

A tela principal será onde teremos uma lista de contatos em uma tabela com a opção de ser direcionado a uma tela para efetuar um novo cadastro, mostrar seus dados, exclui-los ou ser direcionado para uma tela de edição de contato.

#### - Tela de cadastro

Aqui estarão os campos de textos para preenchimento de dados, com a opção de salvá-los ou cancelar o cadastro.

#### - Tela de edição

Nesta tela estarão os campos de texto para edição dos dados cadastrados, com a opção de salvar ou cancelar a edição, é um reaproveitamento da tela de cadastro com os dados do contato já preenchidos e com um novo título.

### 1.2 - Diagrama de classes das Views

Você pode citar algumas classes mais comuns utilizadas no pacote Swing, como JPanel, JFrame e JTable.

#### 1.2.1 - Tela principal

Os elementos nesta tela serão um botão para Cadastrar, outro para Editar e outro para Excluir um contato. Terá uma tabela onde serão mostrado o nome dos contatos cadastrados e Rótulos com detalhes de Nome, Email, Endereço e Grupo. No diagrama de classe da DashboardView (Figura 5.2) temos todos os elementos da tela e seus métodos.

#### 1.2.2 - Tela de cadastro e edição

Nesta tela temos os Campos de Texto com Nome, Email, Endereço e Grupo (que será uma caixa suspensa) para edição e cadastro e terá um botão para Salvar e outro para Cancelar.

#### 1.2.3 - Relacionamento entre as telas

Aqui é necessário que o aluno compreenda a proposta do trabalho. Relacione nosso protótipo com conceitos de diagrama de classes aprendidos na aula anterior.

### 1.3 - Criando nossas telas

#### 1.3.1 - Criando o projeto

Para facilitar e identificar com mais agilidade os possíveis erros citados na aula e outros decorrentes dos alunos, utilizar a mesma nomenclatura citada no texto.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 1 > Exemplos > 1.3.1** no seu Netbeans.

#### 1.3.2 - Criando a primeira tela

A tela será uma JDialog, que é uma tela que possui seu próprio método que faz com que a tela seja executada independentemente de outros serviços. Crie um JDialog e o renomeie, de preferência, como RegisterOrEditContactView.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 1 > Exemplos > 1.3.2** no seu Netbeans.

#### 1.3.3 - Entendendo o Interface Builder

Educador, apresente as janelas do Interface Builder e introduza a paleta de elementos.

#### 1.3.4 – Paleta de elementos

Estes são os componentes que você conseguirá compor qualquer tipo de aplicação.

##### 1.3.4.1 – Jlabel

O JLabel é um rótulo ou um texto simples que exibe informações na tela, não interagível.

##### 1.3.4.2 – JtextField

O JTextField é um campo que tem o intuito de receber um texto digitado pelo usuário e possui o atributo text.

##### 1.3.4.3 - JCombobox:

JComBox é uma caixa de combinação que reúne valores definidos para o usuário selecionar.

##### 1.3.4.4 – Jbutton

O JButton é um botão que recebe o evento de clique.

##### 1.3.4.5 – JPanel

JPanel é um componente que agrupa outros componentes.

##### 1.3.4.6 – JDialog

Uma JDialog é uma janela de diálogo ou subjanela independente que tem o papel de mostrar um aviso temporário, que sobrepõe a janela principal da aplicação Swing.

##### 1.3.4. – JFrame

Esta é a classe principal da aplicação e só pode haver um JFrame sendo executado por vez, ao contrário do JDialog.

##### 1.3.4.7 – JTable

O JTable é utilizado para visualizar dados em forma de grade, com cabeçalho, colunas e linhas e é um dos componentes mais complexos do pacote Java Swing e seu trabalho é dividido em Model, View e Controller.

##### 1.3.4.8 – JMenuBar

JMenuBar é uma barra de menu que geralmente vemos no topo dos programas.

##### 1.3.4.9 – JMenu

JMenu é uma janela pop-up contendo itens de menu (JMenuItem) que é exibido quando o usuário seleciona um item no JMenuBar.

##### 1.3.4.10 – JMenuItem

**E**sta classe é um botão posicionado em uma lista.

#### 1.3.4 - Inserindo elementos na tela

**P**ara adicionar os elementos requisitados, basta clicar e arrastar o componente para sua tela de projeto.

#### 1.3.5 – Modificando as propriedades

**P**odemos encontrar diversas opções de customização para nossas telas, porém são limitadas. Caso queira soltar a imaginação e ter opções ilimitadas de customização de tela, pesquise um pouco sobre criação de GUI.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 1 > Exemplos > 1.3.5** no seu Netbeans.

### 1.4 – Resumo

Descobrimos que o Java pode ser um pouco mais charmoso. Utilizando a biblioteca Swing podemos criar telas de nível profissional e deixar nossos sistemas um pouco mais atraente. Vimos que a fase de prototipagem e mockups podem esclarecer a mente do desenvolvedor na hora de entender o propósito de um software e, por fim, aprendemos a compor telas utilizando o Interface Builder do Netbeans

### 1.5 - Exercícios

|  |  |
| --- | --- |
| **1.** | **Qual é o intuito dos protótipos?** |
|  | Resposta: Validar de uma maneira rápida como os itens de interface se organizarão na tela. |
| **2.** | **O que são os Mockups?** |
|  | Resposta: São desenhos da tela exatamente como ela deverá ser. |
| **3.** | **Qual é o nível de detalhamento de um protótipo em comparação com os Mockups?** |
|  | Resposta: Ao contrário da fase de protótipo, estes desenhos devem ser fiéis ao produto final, para que o desenvolvedor replique o conceito desenhado na aplicação. |
| **4.** | **O que deve ser levado em consideração antes de diagramarmos as classes das views?** |
|  | Resposta: Antes de começar a diagramar, devemos levar em conta a ferramenta a ser utilizada para criação das Views. |
| **5.** | **O que é o Swing? Existem outras bibliotecas com o mesmo intuito?** |
|  | Resposta: Swing é uma biblioteca Java poderosa para criação de telas e componentes gráficos. Existem diversas outras bibliotecas com o mesmo intuito. |
| **6.** | **O que é um Interface Builder?** |
|  | Resposta: É uma ferramenta do Netbeans que permite a criação de telas com o Java Swing com simples arrastar e colar. |
| **7.** | **Qual é a diferença entre o JFrame e o JDialog?** |
|  | Resposta: A característica principal do JDialog é que quando ele é executado, ele sempre sobrepõe outras telas com a presença de uma nova janela. O JFrame apenas atualiza a si mesmo e apresenta os novos componentes sem a necessidade de apresentar uma nova janela por cima. |
| **8.** | **O que são Layouts? Quais são os tipos de Layouts em Java Swing? Explique a diferença de cada um deles.** |
|  | Resposta: As partes do Swing responsáveis por dispor os elementos na tela são os Layout Managers.  **FlowLayout**  É o mais simples e o padrão de todos os JPanels. Organiza os componentes um ao lado do outro em linha, da esquerda para a direita, usando o tamanho que você definiu ou, se não houver definição, seus tamanhos preferenciais. Quando a linha fica cheia, uma nova linha é criada.  **BoxLayout**  Organiza os componentes sequencialmente pelo eixo X ou eixo Y (indicamos isso no construtor) usando os tamanhos preferenciais de cada componente.  **GridLayout**  Organiza os componentes em um grid (tabela) com várias linhas e colunas (podemos definir no construtor). Os componentes são colocados um por célula e com tamanho que ocupe a célula toda.  **GridBagLayout**  É o mais complexo layout e é baseado no GridLayout. A idéia também é representar a tela como um grid com linhas e colunas. Mas no GridBagLayout podemos posicionar elementos ocupando várias células em qualquer direção, o que permite layouts mais customizados, mas também causa um custo muito alto de manutenção. A definição de onde deve ser colocado cada componente é feita através de restrições (GridBagConstraints) passadas ao método add.  **BorderLayout**  Divide o container em cinco regiões: Norte, Sul, Leste, Oeste e Centro. Ao adicionar um componente, indicamos a região onde queremos adicioná-lo. Na hora de renderizar, o BorderLayout primeiro coloca os componentes do Norte e Sul em seus tamanhos preferenciais; depois, coloca os do Leste e Oeste também nos tamanhos preferenciais; por último, coloca o componente do Centro ocupando todo o restante do espaço. |

Para mais informações sobre layouts, confira a documentação Java em: <https://docs.oracle.com/javase/tutorial/uiswing/layout/index.html> .

### 1.6 – TDP

#### Tarefa

Agora que temos os protótipos que definem a estrutura das Views.

E você está chegando ao fim deste projeto!

Você deverá criar suas próprias telas. Use cores, estilos e tamanhos que desejar. Apenas não deixe de seguir a estrutura que os protótipos sugerem. Não se preocupe com os dados por enquanto. Crie dados fictícios e apresente-os na tela.

### SUGESTÃO:

Veja a solução do TDP da aula 3 desta unidade. Nela é apresentado o passo-a-passo para construção de cada Tela/Funcionalidade.

## Aula 2 – Finalizando as Views e os Models

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Aplicar os conhecimentos aprendidos nas aulas anteriores para adicionar funcionalidade às telas criadas.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 20 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 50 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* A tela principal;
* Codificando nossos Models.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 2.1 - A tela principal

Dúvidas constantes a respeito da localização das funções irão surgir. Também a respeito do visual, alguns alunos menos detalhistas não se atentam ao básico proposto e estes merecem maior atenção, pois em uma aplicação é indispensável uma boa apresentação visual.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 2 > Exemplos > 2.1** no seu Netbeans.

### 2.2 - Codificando nossos Models

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 2 > Exemplos > 2.2** no seu Netbeans.

#### 2.2.1 - Enum Group

Um enum é uma enumeração. É como se déssemos apelidos a certos números, simplesmente para não manipularmos eles diretamente. É mais fácil trabalhar com palavras do que com números. A própria matemática é assim.

Para mais detalhes acesse: <http://www.devmedia.com.br/tipos-enum-no-java/25729> .

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 2 > Exemplos > 2.2.1** no seu Netbeans.

#### 2.2.2 - Classe Contact

Como já dito, tome cuidado com os alertas e erros que o Netbeans acusar pelo caminho. O Netbeans é muito bem-intencionado em solucionar seus problemas, mas nem sempre as soluções apresentadas por ele são o melhor caminho. Na maioria das vezes, o melhor é ignorar os alertas do tipo “warning” e em caso de erros de compilação, não concerte até que todo o código seja inserido e importe tudo que for solicitado.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 2 > Exemplos > 2.2.2** no seu Netbeans.

### 2.3 – Resumo

Nesta aula continuamos a criação das Views e aprendemos a configurar uma JTable. Também consolidamos nossos conhecimentos sobre os enums. Eles serão bastante utilizados na sua vida de programador. Por fim codificamos a camada Model e aprendemos como comparar objetos utilizando o método equals().

### 2.4 – Exercícios

**1. O que é um JPanel?**

Resposta: É um componente gráfico que permite agrupar outros elementos como se fossem parte de uma camada.

**2. Como realizar uma ação quando clicarmos em um JButton ?**

Resposta: Cada JButton possui um método que é autogerado pelo Netbeans quando o inserimos no nosso interface builder. Estes métodos são chamados quando o botão recebe evento de clique.

**3. Explique qual é a funcionalidade do método equals() .**

Resposta: equals() É um método da classe Object necessário para que se faça a comparação de igualdade entre dois objetos. É possível sobreescrevê-lo e criar o próprio tratamento que decide a igualdade entre dois objetos.

**4. Para que serve a Classe UUID?**

Resposta: Para obtenção de um número aleatório único.

**5. O que deve ser modificado em um enum para que ele atribua valores do tipo String?**

Resposta: Devemos criar um atributo do tipo String, um construtor que atribui valor a esta String e implementar o método toString() para obter este valor do tipo String .

**6. Por que devemos especificar o tipo de dado de uma coluna de uma JTable?**

Resposta: Se o tipo do objeto não for especificado, um número hexadecimal que representa o endereço de memória seria mostrado, ou seja, um código estranho sem serventia para o seria mostrado.

### 2.5 – TDP

#### 2.5.1 - A criação das telas

#### Tarefa

Estamos chegando ao FIM!

Continue a criação das Views como você fez na aula anterior. Crie as Views representadas nos protótipos acima. Lembre-se que você está livre para usar o estilo que quiser na criação das View, apenas tente seguir a estrutura proposta nos protótipos.

### SUGESTÃO:

Veja a solução do TDP da aula 3 desta unidade. Nela é apresentado o passo a passo para construção de cada Tela/Funcionalidade

## Aula 3 - Controllers

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Relacionar as Views com os Controllers utilizando métodos próprios dos componentes aqui ensinados

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 20 minutos de aula expositiva;
* 20 minutos para tirar as dúvidas dos alunos;
* 50 minutos para desenvolver as atividades propostas para a turma.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* A camada Controller;
* Interligando as Views aos Models;
* Look & Feel.

### PONTOS IMPORTANTES

Para garantir ao aluno um aprendizado significativo, resgate os principais conceitos de cada tópico abordado na aula. Neste momento, é importante que você utilize dicas e sugestões para reforçar os temas trabalhados, oferecendo também exemplos que possibilitem a relação entre teoria e prática.

### 3.1 - A Camada Controller

Atente**-**se ao código mostrado e sempre faça as devidas substituições quando solicitado. A camada controller será responsável por executar os métodos das classes de nosso programa.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.1** no seu Netbeans.

#### 3.1.1 - Preparando lista para a Tabela

Esta é uma etapa crítica para o procedimento prático em que na maioria das vezes o aluno pode se perder ou se esquecer de conceitos aprendidos. Sempre importe as bibliotecas, pacotes e classes, complemente os códigos quando é solicitado no texto e quando necessário retome as aulas anteriores em busca de procedimentos adequados.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.1** no seu Netbeans.

### 3.2 - Interligando as Views com os Models

Esta etapa serve para interligar nossos componentes da View, como os botões, tabela e combo box, aos nossos métodos, ou seja, aqui faremos os botões funcionar, nossa combo box mostrar uma lista de grupos e nossas tabelas mostrar nossos cadastros, e as demais funcionalidades.

#### 3.2.1 - Integrando o Combobox

Nesta etapa, para que o combo box apresente os valores que inserimos no nosso código anteriormente, é só utilizar o código personalizado sugerido no editor de modelo da combo box.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.1** no seu Netbeans.

#### 3.2.2 - Fechando a tela

Com duplo clique no botão cancelar você será direcionado ao campo do código destinado ao método cancelar do botão. Incremente com o código sugerido para que a janela feche e não continue alocando memória.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.2** no seu Netbeans.

#### 3.2.3 - Salvando um novo contato

É comum que o aluno se confunda com os botões e troque os códigos por engano. Sempre tente executar o aplicativo e procurar os possíveis erros e continue ponderando as sugestões de correção tentadoras do Netbeans.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.3** no seu Netbeans.

#### 3.2.4 - Integrando a tabela

Esteja atento a cada passo do aluno e oriente-o a implementar os seus códigos nos pacotes corretos, dentro das classes corretas, aplicando os métodos corretamente, idêntico aos do teto. Como a implementação da tabela possui muitos passos, o aluno pode se perder.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.4** no seu Netbeans.

#### 3.2.5 - Exibindo os detalhes de um contato

Aqui algumas correções sugestionadas pelo Netbeans podem arruinar o programa. Tome cuidado.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.5** no seu Netbeans.

#### 3.2.6 - Deletando um contato

Aqui basta incrementar o método que dá a função ao botão deletar, sugestionada no livro do aluno.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.6** no seu Netbeans.

#### 3.2.7 - Interligando as telas

Mesma coisa do item anterior, mas incrementando os métodos do botão editar e cadastrar.

**Material de Apoio:** Para conferir o exemplo deste tópico abra o projeto **Material de Apoio POO > Unidade 5 > Aula 3 > Exemplos > 3.2.7** no seu Netbeans.

### 3.3 - Look & Feel

Como seção final e extra do módulo de POO, o Look & Feel é apresentado. Look & Feel é o conjunto de cores e a aparência dos componentes do Java Swing. Nesta seção é ensinado como trocar o Look & Feel entre os disponíveis por padrão no Netbeans, então atente-se para que o aluno não modifique os códigos autogerados pelo Java Swing.

### 3.4 – Resumo

Nesta aula você consolidou seus conhecimentos em Java. Vimos como de fato, funciona uma arquitetura MVC, aprendemos a criar adaptações em nossas classes para servir como modelo em nossos componentes de View. E também conseguimos criar eventos em botões e interceptar eventos interessantes, de agora em diante iremos trabalhar outros aspectos para o desenvolvimento de Aplicativos.

Nossas aulas sobre Swing e Java terminam por aqui. Há muito mais sobre Swing para aprender envolvendo outros tipos de aplicações, continue se aprimorando!

### 3.5 – Exercícios

**1. O que é um Controller?**

Resposta: É uma camada que controla a apresentação dos dados na camada view. É o intermediador entre a camada model e a camada view.

**2. Sobre os enums, o que é necessário ser feito para que um JCombobox receba-o como Model?**

Resposta: Para que um JCombobox receba um Model basta implementá-lo na janela de propriedades, na opção model, em seguida insera um código personalizado que resgata constantes para os enums criados por meio do método Groups.values().

**3. Você utilizou um ArrayList. Pesquise e responda. O que é um ArrayList? Para que serve?**

Resposta: O ArrayList é uma classe usada para representar um array em orientação a objetos. Em Java temos uma classe chamada List, que também é usada para manipular listas, e o ArrayList é uma especialização desta classe com alguns métodos a mais. Na documentação oficial do Java podemos ver uma explicação completa e neste site (<http://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298>), podemos explorar algumas formas de manipulação de ArrayLists.

**4. Qual método é necessário para que uma view se torne visível? E qual é o método para destruir uma view?**

Resposta: Para que a view se torne visível o método necessário é setVisible(true). Para destruir a view usa-se dispose()

**5. O que é um Listener? Cite um exemplo.**

Resposta: Para interceptar o evento de seleção da tabela, usamos o que chamamos de Listener.

Um listener é utilizado para monitorar alguma atividade e avisar quando um evento esperado acontece. Este é um assunto mais avançado em Java. Se quiser, recomende o site: <https://docs.oracle.com/javase/tutorial/uiswing/events/> .

**6. O que é Look&Feel?**

Resposta: O Swing possui alguns temas disponíveis para a nossa aplicação. Estes temas são chamados de Look & Feel.

### 3.6 – TDP

Esta é a atividade final onde colocará à prova todo conhecimento adquirido no curso.

#### 3.6.1 - Controllers

Após ter feitos as Views, você precisa dar vida a elas, interligando-as com os Models, e persistindo os dados salvos.

Primeiro você deverá interligar as Views com os Models através dos Controllers. Muito similar ao aprendido em aula.

#### Tarefa

Chegamos a última etapa!

Como se sente? Realizado com certeza, veja onde chegou!

Para finalizarmos o projeto, nessa etapa final você deverá finalizar as funcionalidades de cadastrar, editar, visualizar e deletar, para isso utilize Controllers para realizar a interação entre View e Models

#### Fim

E após tudo isso como está o seu projeto?!

Agora chegou o grande momento de apresentar a sua turma e ao seu educador! Parabéns! Você está craque em Java!!

### SUGESTÃO:

# TDP Unidade 5

Estamos na etapa final. Este trabalho prático é um pouco maior do que os exemplos dados em aula, então é importantíssimo que façamos o máximo de proveito de Orientação a Objetos que aprendemos em aula. Será fácil se perder em meio a tantas telas, mas com bom uso de OO, isto não será problema.

Até então fizemos interpretamos o problema, modelamos um diagrama de classe e implementamos nossas classes Model.

Nossa camada Model não está totalmente pronta, na verdade só estará pronta no final do projeto, pois somos livres para adaptá-la da melhor maneira conforme o programa aumenta, veremos isto no decorrer do projeto.

Vamos implementar de uma ordem diferente dos exemplos dados em aula. Vamos utilizar o método de desenvolvimento incremental.

## Desenvolvimento Incremental

Desenvolvimento Incremental é uma estratégia de planejamento estagiado em que várias partes do sistema são desenvolvidas em paralelo, e integradas quando completas. Neste caso não serão desenvolvidas em paralelo, mas estas partes do sistema serão desenvolvidas de forma que se tornem funcionais sem o projeto como um todo. Por exemplo, primeiro vamos criar a funcionalidade de cadastrar um filme. Primeiro criaríamos os models (mas já estão criados), depois devemos criar as Views respectivas desta funcionalidade, e por fim os controllers, com o intuito de gerar uma parte do sistema funcional.

É claro que cada funcionalidade gera uma dependência até o projeto esteja finalizado, por exemplo, para cadastrar filmes precisamos adicionar atores, equipe e produtoras, que por sua vez precisam das respectivas funcionalidades de cadastro. Então a ordem da implementação das funcionalidades será conforme suas dependências. Você irá entender. Agora mãos na massa; vamos criar nossa tela de cadastro de filmes.

## 1 - Cadastro de filmes

Está é uma das telas principais do nosso app, e com ela existe muitas dependências. Como já criamos todos os Models, vamos criar nossas Views, e alterar os models conforme for necessário.

### 1.1 - Criando a View

Antes de criar a View vamos relembrar o protótipo correspondente à esta tela.

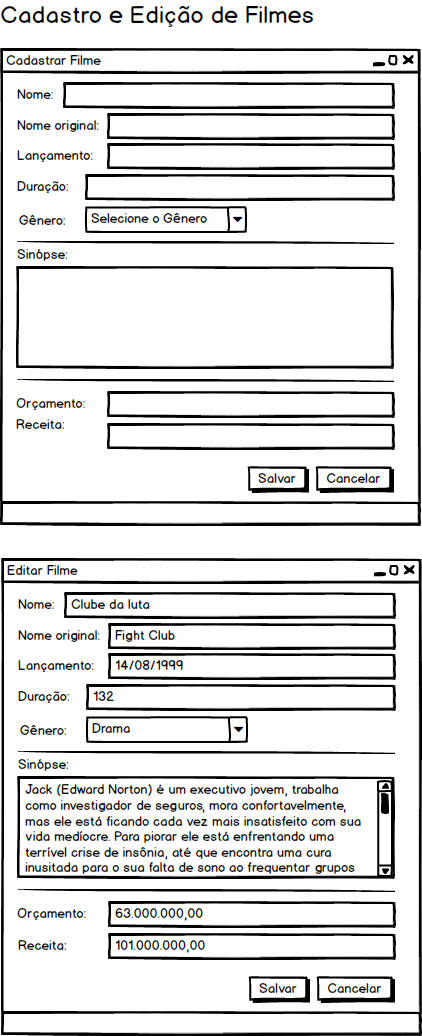


Figura 5. – Cadastrar e Editar filmes

Crie então um JDialog com o nome AddOrEditMovieView e altere a propriedade title para “Adicionar Filmes”. Os nomes das variáveis foram escolhidos para facilitar a identificação do tipo dos componentes, por exemplo: JLabels contém o sufixo Label, JButtons contém o sufixo Button, o JTextArea contém o sufixo TextArea.

Aqui o procedimento é o mesmo ensinado em aula. Insira os componentes, ajuste o tamanho do JDialog e nomeie as variáveis conforme a figura abaixo:

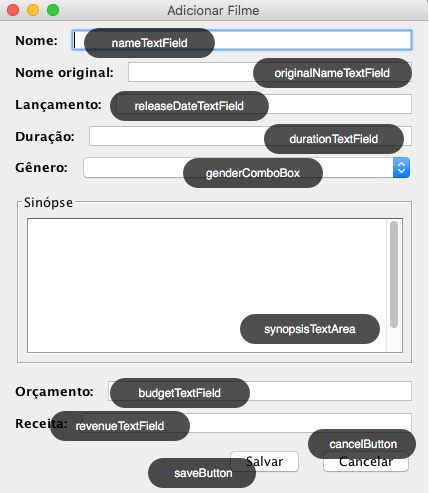


Figura 5. - Adicionar Filme

Lembre-se que o aluno poderá criar seu próprio estilo de tela, mas deverá respeitar a estrutura proposta no protótipo. Seguindo este exemplo, sua tela ficará parecida com esta:

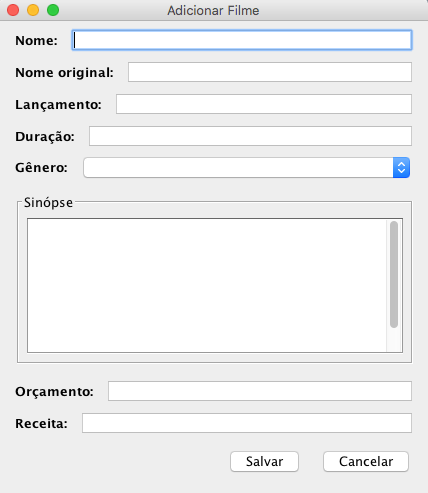


Figura 5. - Adicionar Filme

### 1.2 - Criando os Controllers

Seguindo o exemplo dado em aula, precisamos criar uma classe que gerencie os filmes, com ações de cadastrar, editar e deletar.

#### 1.2.1 - Crie uma implementação genérica de AbstractTableModel

Na fase de models criamos a interface TableModel que terá a utilidade de prover os dados que irão na primeira e segunda coluna das tabelas. Haverá muitas tabelas, e não precisamos de implementar o método AbstractTableModel para cada tabela que desejamos utilizar. Já que padronizamos que todas as tabelas terão uma ou duas colunas, basta utilizar o poder da OO e generalizar uma classe que implemente esta interface (AbstractTableModel) para todas as outras tabelas. Vamos lá:

**1 –** Crie uma classe e nomeie-a TableModelImplementation.

**2 –** Nesta classe, estenda a classe AbstractTableModel.

**3 –** Implemente todos os métodos abstratos, como visto em aula.

**4 –** Não inicialize o atributo cols

**5 –** Crie o método updateRows(), como criado em aula, mas deixe-o flexível para manipular linhas com uma e duas colunas. Veja como o método fica e repare nos métodos getFirstColumn() e getSecondColumn():

**public** **void** **updateRows()** **{**

rows **=** **new** ArrayList**();**

**if** **(**list**.**isEmpty**())** **{**

**return;**

**}**

**for(**TableModel object**:** list**)** **{**

**if(**object**.**getSecondColumn**()** **!=** **null)** **{**

rows**.**add**(new** Object**[]** **{**object**.**getFirstColumn**(),** object**.**getSecondColumn**()});**

**}else{**

rows**.**add**(new** Object**[]** **{**object**.**getFirstColumn**()});**

**}**

**}**

**}**

Aí está o motivo de criarmos a interface TableModel. Lembra que implementamos estes métodos (getFirstColumn() e getSecondColumn()) em cada model? Isto significa que dependendo do tipo de dado que o array list é composto, exibirá um dado diferente. Estamos aos poucos mudando a estratégia vista em aula.

Confira a classe completa no material de apoio desta unidade.

#### 1.2.2 - Crie um gerenciador de listas genérico

Neste projeto teremos diversas listas (filmes, atores, pessoas, países, gêneros, etc), então para que possamos economizar código e fazer o máximo de reuso. Lembra que implementamos a interface TableModel nos Models? Pois é, ela foi criada pensando em exibir estes models na tabela.

Agora você verá o quão a OO é poderosa. Siga os seguintes passos:

**1 -** Crie um pacote chamado Controllers. É nele que você salvará todos os controllers.

**2 -** Crie uma classe chamada GenericTableController. Esta classe será muito parecida com a classe ContactsController criada em aula.

**3 -** Adicione o atributo list sendo um ArrayList. Este ArrayList receberá objetos que conformam com a interface TableModel que criamos.

**public** **abstract** **class** **GenericTableController** **{**

**protected** ArrayList**<**TableModel**>** list**;**

**}**

**4 -** Adicione um atributo para guardarmos nosso objeto selecionado. Ele também conformará a interface TableModel.

**protected** TableModel selectedObject**;**

Nosso Controller não poderá conter atributos e métodos estáticos, pois ele será utilizado para mais de uma tabela, isto é, precisaremos de instancias para cada tabela. Ao mesmo tempo que precisamos instanciar cada Controller, também precisamos manter os dados na lista, impedindo que estes sejam perdidos quando seu uso for suspenso. Então neste Controller teremos um atributo TableModelImplementation, para implicar que todas as tabelas que utilizarem (ou subclasses que estenderem) a classe GenericTableController, instancie um novo TableModelImplementation (já que é esta classe que possui a lista onde os dados serão mantidos).

**5 –** Crie o atributo tableModel do tipo TableModelImplementation:

**protected** TableModelImplementation tableModel **=** **null;**

**6 -** Insira os métodos getter e setter para o atributo selectedObject.

**public** **abstract** **class** **GenericTableController** **{**

*/*/ ...

**public** TableModel **getSelectedObject()** **{**

**return** selectedObject**;**

**}**

**public** **void** **setSelectedObject(**TableModel selectedObject**)** **{**

**this.**selectedObject **=** selectedObject**;**

**}**

**}**

Observe o comentário “…” . Ele indica que existe código ali que omitimos. Vamos utilizar esta notação para omitir código e deixar o passo-a-passo mais limpo.

**7 –** Implemente o método updateRows():

**protected** **final** **void** **updateRows()** **{**

tableModel**.**list **=** list**;**

tableModel**.**updateRows**();**

**}**

**8 -** Agora, como o exemplo dado em aula, teremos métodos de controle da lista: o save, delete e o update. Veja na listagem abaixo:

**public** **abstract** **class** **GenericTableController** **{**

*/*/ ...

**public** **void** **save(**TableModel newObject**)** **{**

**if(**list **==** **null)** **{**

list **=** **new** ArrayList**<>();**

**}**

list**.**add**(**newObject**);**

updateRows**();**

doAdditionalUpdated**();**

**}**

**public** **void** **edit(**TableModel objectToBeEdited**)** **{**

**for(**TableModel object**:** list**)** **{**

**if(**objectToBeEdited**.**equals**(**object**))** **{**

list**.**set**(**list**.**indexOf**(**object**),** objectToBeEdited**);**

**break;**

**}**

**}**

updateRows**();**

doAdditionalUpdated**();**

**}**

**public** **void** **delete(**TableModel objectToBeDeleted**)** **{**

**for(**TableModel object**:** list**)** **{**

**if(**objectToBeDeleted**.**equals**(**object**))** **{**

list**.**remove**(**object**);**

**break;**

**}**

**}**

updateRows**();**

doAdditionalUpdated**();**

**}**

**}**

**Obs.:** Repare que escondemos novamente o código antigo utilizando o comentário “…”, apenas para manter o exemplo limpo.

O básico para nosso controller está pronto. Veja como ele deve estar neste ponto:

**package** Controllers**;**

**import** Models.TableModel**;**

**import** java.util.ArrayList**;**

**public** **abstract** **class** **GenericTableController** **{**

**protected** ArrayList**<**TableModel**>** list**;**

**protected** TableModelImplementation tableModel **=** **null;**

**protected** TableModel selectedObject**;**

**public** **GenericTableController()** **{**

list **=** **new** ArrayList**<>();**

tableModel **=** **new** TableModelImplementation**();**

tableModel**.**list **=** list**;**

updateRows**();**

**}**

**public** **void** **save(**TableModel newObject**)** **{**

**if(**list **==** **null)** **{**

list **=** **new** ArrayList**<>();**

**}**

list**.**add**(**newObject**);**

updateRows**();**

doAdditionalUpdated**();**

**}**

**public** **void** **edit(**TableModel objectToBeEdited**)** **{**

**for(**TableModel object**:** list**)** **{**

**if(**objectToBeEdited**.**equals**(**object**))** **{**

list**.**set**(**list**.**indexOf**(**object**),** objectToBeEdited**);**

**break;**

**}**

**}**

updateRows**();**

doAdditionalUpdated**();**

**}**

**public** **void** **delete(**TableModel objectToBeDeleted**)** **{**

**for(**TableModel object**:** list**)** **{**

**if(**objectToBeDeleted**.**equals**(**object**))** **{**

list**.**remove**(**object**);**

**break;**

**}**

**}**

updateRows**();**

doAdditionalUpdated**();**

**}**

**protected** **abstract** **void** **doAdditionalUpdated();**

**protected** **final** **void** **updateRows()** **{**

tableModel**.**list **=** list**;**

tableModel**.**updateRows**();**

**}**

**public** TableModel **getSelectedObject()** **{**

**return** selectedObject**;**

**}**

**public** **void** **setSelectedObject(**TableModel selectedObject**)** **{**

**this.**selectedObject **=** selectedObject**;**

**}**

**public** ArrayList**<**TableModel**>** **getList()** **{**

**return** list**;**

**}**

**public** **void** **setList(**ArrayList**<**TableModel**>** list**)** **{**

tableModel**.**list **=** list**;**

**this.**list **=** list**;**

updateRows**();**

**}**

**public** TableModelImplementation **getTableModel()** **{**

TableModelImplementation newModel **=** **new** TableModelImplementation**();**

newModel**.**list **=** list**;**

newModel**.**setRows**(**tableModel**.**getRows**());**

newModel**.**setCols**(**tableModel**.**getCols**());**

**return** newModel**;**

**}**

**}**

##### 1.2.2.1. MovieController

Agora que criamos o controller genérico vamos extendê-lo para criar um controller de filmes.

**1 -** No pacote Controllers, crie uma classe com o nome MoviesController que extenda da classe GenericTableController:

**public** **class** MoviesController **extends** GenericTableController {

}

**2 -** Adicione um construtor para que possamos fazer as devidas inicializações:

**private** **MoviesController()** **{**

**super();**

*// Adicionamos dois filmes para teste*

list**.**add**(new** Movie**(**"Teste"**,** "Nome original Teste"**,** "Sinopse Teste"**,** "11/11/1111"**,** **new** Gender**(**"Ação"**),** "R$ 1,00"**,** "R$ 2,00"**,** "130"**););**

list**.**add**(new** Movie**(**"Teste 2"**,** "Nome original Teste 2"**,** "Sinopse Teste"**,** "11/11/1112"**,** **new** Gender**(**"Comédia"**),** "R$ 1,00"**,** "R$ 2,00"**,** "130"**));**

tableModel**.**list **=** list**;**

*// Determinamos os rótulos das duas colunas*

tableModel**.**setCols**(new** String**[]** **{**"Nome"**,** "Ano"**});**

updateRows**();**

**}**

Veja aqui a inicialização do atributo cols que adiamos no GenericTableController. Inicializamos aqui pois este será o controller para a lista de filmes e neste ponto já sabemos que a tabela respectiva deste controller será a tabela de filmes e que ela possuirá a coluna “Nome” e “Ano”.

O teste destas classes ficará para quando implementarmos a tela Dashboard.

No exemplo dado em aula criamos diversos métodos estáticos e o atributo list deixamos como estático também. No caso deste TDP, não podemos deixa-los estático pois, como dito acima, precisaremos de dados diferentes para diferentes tabelas, então utilizaremos a técnica Singleton.

##### 1.2.2.2. Padrão de projeto Singleton

Se você estudou o padrão Singleton, como pedido na Unidade 3, você já deve saber. O Singleton é uma técnica que força que uma determinada classe possua uma instância única, instância esta que será utilizada durante toda a execução do software. A instância é guardada em um atributo estático de mesmo tipo da classe. Vamos implementar um Singleton na MoviesController.

**1 –** Em MoviesController, crie um atributo estático para armazenar a instância única e compartilhada.

**private** **static** MoviesController sharedInstance **=** **null;**

**2 –** Para evitar que a classe seja instanciada, marque o construtor como privado.

**private** **MoviesController()** **{**

*// ...*

**}**

3 – Crie um getter para sharedInstance que garanta que a mesma só será instanciada se a mesma for nula:

**public** **static** MoviesController **getSharedInstance()** **{**

**if(**sharedInstance **==** **null)** **{**

sharedInstance **=** **new** MoviesController**();**

**}**

**return** sharedInstance**;**

**}**

Nosso singleton está implementado. Para utilizar esta instância única, ao invés de criar uma nova instancia utilizando o construtor new, utilize o seguinte padrão:

MoviesController instance = MoviesController.**getSharedInstance();**

A partir daí basta realizar a chamada dos métodos em instance normalmente que a instância compartilhada irá armazenar todas as modificações.

#### 1.2.3 - Integrando a view com o controller (cadastrando filmes)

Agora vamos fazer de fato o cadastro de filmes implementando a integração.

Para dar andamento a esta tela, vamos resolver nossa primeira dependência: a lista de gêneros.

Faremos a implementação da classe AbstractListModel que é similar ao nosso conhecido AbstractTableModel, a diferença é que a primeira não lida com colunas.

Também vamos implementar a interface ComboBoxModel para que nosso controller seja compatível com o nosso JCombobox.

Lembre-se que teremos muitos JCombobox no nosso app, então iremos implementar uma classe genérica para isto também.

##### 1.2.3.1. Criando um model para combobox genérico

Será o mesmo procedimento da criação do nosso GenericTableController, siga os passos:

**1 -** Crie uma classe com o nome GenericListController extendendo a classe AbstractListModel e implemente a interface ComboBoxModel como o trecho a seguir:

**import** javax.swing.AbstractListModel**;**

**import** javax.swing.ComboBoxModel**;**

**public** **abstract** **class** **GenericListController** **extends** AbstractListModel **implements** ComboBoxModel **{**

**}**

**2 -** Crie um atributo do tipo TableModel para que possamos armazenar a linha selecionada do combobox:

**import** javax.swing.AbstractListModel**;**

**import** javax.swing.ComboBoxModel**;**

**public** **abstract** **class** **GenericListController** **extends** AbstractListModel **implements** ComboBoxModel **{**

**static** TableModel selection **=** **null;**

**}**

**3 -** Agora vamos criar o atributo responsável por guardar os valores do combobox:

**public** **abstract** **class** **GenericListController** **extends** AbstractListModel **implements** ComboBoxModel **{**

**static** TableModel selection **=** **null;**

**static** TableModel**[]** values**;**

**}**

**4 -** Veja que existe um warning pedindo que os métodos da interface e classe mãe sejam implementados. Então assim o faça e adicione os seguintes tratamentos.

@Override

**public** Object **getElementAt(int** index**)** **{**

**return** values**[**index**];**

**}**

@Override

**public** **int** **getSize()** **{**

**return** values**.**length**;**

**}**

@Override

**public** **void** **setSelectedItem(**Object anItem**)** **{**

selection **=** **(**TableModel**)** anItem**;**

**}**

@Override

**public** Object **getSelectedItem()** **{**

**return** selection**;**

**}**

**5 –** Crie um método de conversão para conseguirmos utilizar um ArrayList (que é mais fácil de ser manipulado) ao invés de um array primitivo:

**protected** **abstract** TableModel **getObjectFromString(**String string**);**

**public** **static** **void** **setValuesFromObject(**ArrayList**<**TableModel**>** array**)** **{**

TableModel**[]** aux **=** **new** TableModel**[**array**.**size**()];**

Iterator**<**TableModel**>** iterator **=** array**.**iterator**();**

**for** **(int** i **=** 0**;** i **<** aux**.**length**;** i**++)** **{**

aux**[**i**]** **=** iterator**.**next**();**

**}**

values **=** aux**;**

**}**

Nosso gerenciador de lista está pronto. Vamos criar nosso gerenciador de Gêneros.

**6 -** Crie um arquivo chamado de GendersController que estenda a classe GenericListController. Nele iremos inicializar nossa lista de valores com uma lista predefinida de gêneros. Veja o resultado final deste controller:

**import** Models.Gender**;**

**import** Models.TableModel**;**

**public** **class** **GendersController** **extends** GenericListController **{**

**public** **GendersController()** **{**

init**();**

**}**

**private** **static** **void** **init()** **{**

selection **=** **new** Gender**(**"Ação"**);**

values **=** **new** TableModel**[]** **{**

selection**,**

**new** Gender**(**"Animação"**),**

**new** Gender**(**"Aventura"**),**

**new** Gender**(**"Chanchada"**),**

**new** Gender**(**"Cinema catástrofe"**),**

**new** Gender**(**"Comédia"**),**

**new** Gender**(**"Comédia romântica"**),**

**new** Gender**(**"Comédia dramática"**),**

**new** Gender**(**"Comédia de ação"**),**

**new** Gender**(**"Cult"**),**

**new** Gender**(**"Dança"**),**

**new** Gender**(**"Documentários"**),**

**new** Gender**(**"Drama"**),**

**new** Gender**(**"Espionagem"**),**

**new** Gender**(**"Fantasia"**),**

**new** Gender**(**"Faroeste (ou western)"**),**

**new** Gender**(**"Ficção científica"**),**

**new** Gender**(**"Franchise/Séries"**),**

**new** Gender**(**"Guerra"**),**

**new** Gender**(**"Machinima"**),**

**new** Gender**(**"Masala"**),**

**new** Gender**(**"Musical"**),**

**new** Gender**(**"Filme noir"**),**

**new** Gender**(**"Policial"**),**

**new** Gender**(**"Romance"**),**

**new** Gender**(**"Suspense"**),**

**new** Gender**(**"Terror (ou horror)"**),**

**new** Gender**(**"Trash"**)**

**};**

**}**

*// Verifica se o gênero já está contido na lista*

**public** **static** **boolean** **isValidGender(**Gender gender**)** **{**

init**();**

**for(**TableModel object**:** values**)** **{**

Gender validGender **=** **(**Gender**)**object**;**

**if(**gender**.**equals**(**validGender**))** **{**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

@Override

**protected** TableModel **getObjectFromString(**String string**)** **{**

init**();**

Gender gender **=** **new** Gender**(**string**);**

**if(**isValidGender**(**gender**))** **{**

**return** gender**;**

**}**

**return** **null;**

**}**

**}**

Deixamos o gênero “Ação” como selecionado do nosso combobox. Você pode adicionar mais gêneros que julgar necessário

**Obs.:** A notação de array acima (TableModel[] { ... }) é uma atribuição literal de array. Este é um **array primitivo** simples e não possui todos os recursos de um ArrayList.

**6 -** O último passo para terminar nosso combobox de gênero é ligá-lo ao controller criado.

**6.1 -** Para isto no arquivo AddOrEditMovieView selecione o combobox genderComboBox e visite seu painel de propriedades.

**6.2 -** Localize a propriedade **model** e clique no botão “…” para abrir sua tela de configuração de model. Selecione a opção **Código personalizado**.

**6.3 -** Agora vamos atribuir nosso GendersController ao genderComboBox. Adicione o seguinte código no campo que se revelou:

**new** GendersController()

As configurações deverão estar como a figura abaixo:

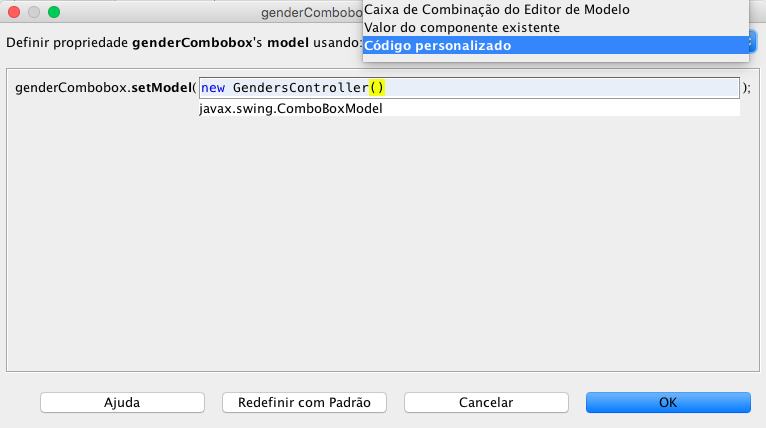


Figura 5. - Model do combobox de gênero

**6.4 -** Agora basta entrar no modo código-fonte do arquivo AddOrEditMovieView e importar GendersControllers à classe. (Use Cmd+Shift+I, se você estiver em um Mac, ou Ctrl+Shift+I caso esteja em um computador Windows).

Agora basta executar o arquivo AddOrEditMovieView e ver que nosso combobox foi preenchido.

##### 1.2.3.2. Configurando o JTextArea

Nosso JTextArea deverá ter as seguintes configurações:

* Barra de rolagem vertical
* Quebra de texto ao final do container

Para isso siga os seguintes passos:

**1 -** Basta selecionarmos nosso synopsisTextArea e, no painel de propriedades, deixe marcado a propriedade lineWrap, como na imagem a seguir:



Figura 5. - Propriedade lineWrap selecionada

É só isso que precisamos fazer, a medida que digitarmos a barra vertical irá aparecer.

##### 1.2.3.3. Fechando a tela

Neste passo iremos aproveitar os exemplos dados em aula. Ainda no arquivo AddOrEditMovieView no modo **Projeto**, e dê duplo clique no botão “Cancelar”. Você irá entrar na implementação do evento de clique do botão. Adicione o método setVisible(false) e o dispose(). Veja como ficará:

**private** **void** **cancelButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

setVisible**(false);**

dispose**();**

**}**

Execute o arquivo e teste clicando no botão **Cancelar**.

##### 1.2.3.4. Máscaras para TextField

Este conteúdo é adicional ao livro do aluno, então é importante que você, educador, mostre esta parte em especial aos alunos.

Na nossa View temos campos numéricos (duration), campo de data (releaseDate) e campos monetários (budget e revenue). Devemos restringir o tipo da entrada nestes campos para que o usuário não coloque algo diferente do que estamos esperando (tempo negativo em duration, palavras em releaseDate e nos campos monetários, etc). Para isto vamos criar máscaras aos campos de texto. Começaremos com o durationTextField, que é o mais simples,

Para o durationTextField devemos restringir a entrada para que receba apenas números inteiros positivos, pois este campo receberá os minutos de duração de um filme.

**1** - Remova o durationTextField e no lugar dele insira um JFormattedTextField, chame-o de durationFormattedTextField.

**2 -** Selecione o durationFormattedTextField e no painel de propriedades clique no botão “…” da propriedade formatterFactory. A seguinte tela se abrirá:

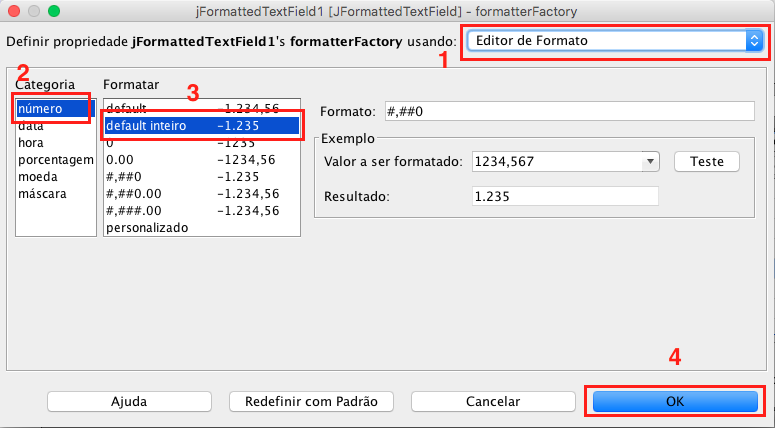


Figura 5. - Formatação de duração

**2.1 -** Selecione **Editor de formato** (1), categoria **número** (2) e no grupo **Formatar** selecione **default inteiro** (3). Clique em **OK** para aplicar a alteração (4).

Agora vamos formatar os campos monetários.

**3** - Remova os campos budgetTextField e revenueTextField, no lugar dele insira isJFormattedTextField, chame-os de budgetFormattedTextField e revenueFormattedTextField respectivamente.

**4 -** Selecione o budgetFormattedTextField e no painel de propriedades clique no botão “…” da propriedade formatterFactory. A siga os passos de acordo com a imagem:

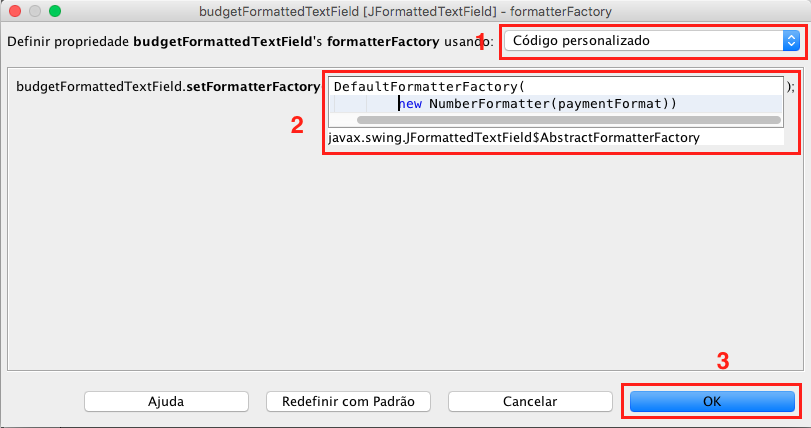


Figura 5. - Formatação monetária

**4.1 -** Selecione **Código personalizado** (1), cole o código abaixo (2) e clique em **OK** para aplicar a alteração (3).

new DefaultFormatterFactory(new NumberFormatter(paymentFormat))

**5 -** Repita o procedimento 4 e 4.1 para revenueFormattedTextField.

**6 -** Para ajudar o usuário, altere a propriedade text de ambos campos com o valor R$ 0,00.

Se inserirmos algum valor inválido o campo ficará em branco no momento que seu foco for perdido.

**7 -** Agora vamos formatar o campo de data. Remova o releaseDateTextField e no lugar dele insira um JFormattedTextField, chame-o de releaseDateFormattedTextField.

**8 -** Selecione o releaseDateFormattedTextField e no painel de propriedades clique no botão “…” da propriedade formatterFactory. A siga os passos de acordo com a imagem:

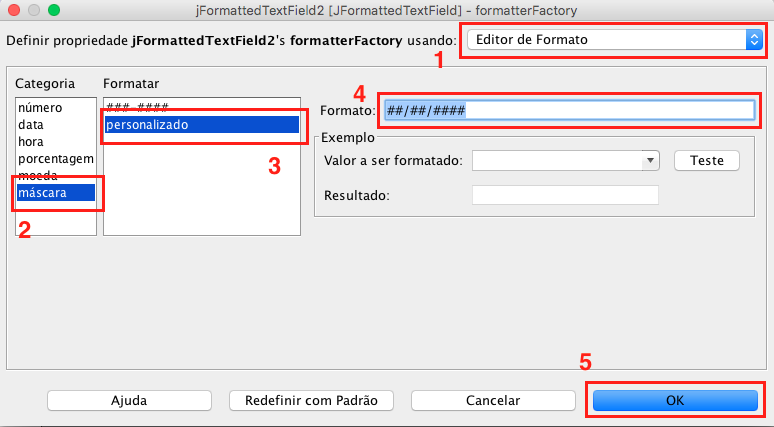


Figura 5. - Formatação de data

**8.1 -** Selecione **Editor de formato** (1), categoria **máscara** (2) e no grupo **Formatar** selecione **personalizar**(3). No campo **Formato** insira a seguinte máscara de data: ##/##/#### (4). Clique em **OK** para aplicar a alteração (5).

Ufa! Campos Formatado.

##### 1.2.3.5. Salvando um filme na lista

Vamos salvar um filme. Seguiremos os mesmos passos dados em aula, com o aditivo de que iremos fazer validação dos campos.

**1** - No arquivo AddOrEditMovieView em modo **Código-Fonte** adicione um atributo do tipo MovieController.

MoviesController controller **=** **null;**

**2 –** No método construtor, inicialize o atributo Controller:

controller **=** MoviesController**.**getSharedInstance**();**

**1** - No arquivo AddOrEditMovieView em modo Projeto dê duplo clique no botão salvar e adicione o seguinte código:

String name **=** nameTextField**.**getText**();**

String originalName **=** originalNameTextField**.**getText**();**

String releaseDate **=** releaseDateFormattedTextField**.**getText**();**

String duration **=** durationFormattedTextField**.**getText**();**

Gender gender **=** **(**Gender**)** genderCombobox**.**getSelectedItem**();**

String synopsis **=** synopsisTextArea**.**getText**();**

String revenue **=** revenueFormattedTextField**.**getText**();**

String budget **=** budgetFormattedTextField**.**getText**();**

Isto irá pegar os valores contidos nos campos da View.

**2** - Agora vamos fazer as validações dos campos. Adicione o seguinte código:

*// ...*

**if** **(**name**.**isEmpty**()** **||**

originalName**.**isEmpty**()** **||**

releaseDate**.**isEmpty**()** **||**

duration**.**isEmpty**()** **||**

gender **==** **null** **||**

synopsis**.**isEmpty**()** **||**

revenue**.**isEmpty**()** **||**

budget**.**isEmpty**())** **{**

JOptionPane**.**showMessageDialog**(null,** "Existem campos vazios que devem ser preenchidos. \nTodos os campos são obrigatórios."**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}**

**else** **if** **(**Long**.**parseLong**(**duration**)** **<** 1**)** **{**

JOptionPane**.**showMessageDialog**(null,** "Duração do filme inválida. A duração não pode ser negativa"**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else** **if(**releaseDate**.**equals**(**" / / "**)){**

JOptionPane**.**showMessageDialog**(null,** "Lançamento inválido. A data deve estar no formato dd/MM/yyyy"**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else{**

Movie movie **=** **new** Movie**(**name**,** originalName**,** synopsis**,** releaseDate**,** gender**,** budget**,** revenue**,** duration**);**

controller**.**save**(**movie**);**

JOptionPane**.**showMessageDialog**(null,** "Filme adicionado com sucesso. \nEdite-o para adicionar Elenco, Equipe e Produtoras"**,** "Sucesso"**,** DISPOSE\_ON\_CLOSE**);**

setVisible**(false);**

dispose**();**

**}**

O método completo ficará assim:

**private** **void** **saveButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

String name **=** nameTextField**.**getText**();**

String originalName **=** originalNameTextField**.**getText**();**

String releaseDate **=** releaseDateFormattedTextField**.**getText**();**

String duration **=** durationFormattedTextField**.**getText**();**

Gender gender **=** **(**Gender**)** genderCombobox**.**getSelectedItem**();**

String synopsis **=** synopsisTextArea**.**getText**();**

String revenue **=** revenueFormattedTextField**.**getText**();**

String budget **=** budgetFormattedTextField**.**getText**();**

*//Checando campos obrigatórios*

**if** **(**name**.**isEmpty**()** **||**

originalName**.**isEmpty**()** **||**

releaseDate**.**isEmpty**()** **||**

duration**.**isEmpty**()** **||**

gender **==** **null** **||**

synopsis**.**isEmpty**()** **||**

revenue**.**isEmpty**()** **||**

budget**.**isEmpty**())** **{**

JOptionPane**.**showMessageDialog**(null,** "Existem campos vazios que devem ser preenchidos. \nTodos os campos são obrigatórios."**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}**

**else** **if** **(**Long**.**parseLong**(**duration**)** **<** 1**)** **{**

JOptionPane**.**showMessageDialog**(null,** "Duração do filme inválida. A duração não pode ser negativa"**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else** **if(**releaseDate**.**equals**(**" / / "**)){**

JOptionPane**.**showMessageDialog**(null,** "Lançamento inválido. A data deve estar no formato dd/MM/yyyy"**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else{**

Movie movie **=** **new** Movie**(**name**,** originalName**,** synopsis**,** releaseDate**,** gender**,** budget**,** revenue**,** duration**);**

controller**.**save**(**movie**);**

JOptionPane**.**showMessageDialog**(null,** "Filme adicionado com sucesso. \nEdite-o para adicionar Elenco, Equipe e Produtoras"**,** "Sucesso"**,** DISPOSE\_ON\_CLOSE**);**

setVisible**(false);**

dispose**();**

**}**

**}**

Terminamos a funcionalidade de cadastro de filmes. Agora seguiremos com a criação da DashboardView para testar o cadastro.

##### 1.2.3.6. Editando um filme da lista

Agora que criamos a estrutura para salvar, vamos criar a estrutura para editar.

**1** - Crie o método fillFields():

**private** **void** fillForm**()** **{**

selectedMovie **=** **(**Movie**)** controller**.**getSelectedObject**();**

nameTextField**.**setText**(**selectedMovie**.**getName**());**

originalNameTextField**.**setText**(**selectedMovie**.**getOriginalName**());**

releaseDateFormattedTextField**.**setText**(**selectedMovie**.**getFormattedReleaseDate**());**

durationFormattedTextField**.**setText**(**selectedMovie**.**getFormattedDuration**());**

genderCombobox**.**setSelectedItem**(**selectedMovie**.**getGender**().**getName**());**

synopsisTextArea**.**setText**(**selectedMovie**.**getSynopsis**());**

revenueFormattedTextField**.**setText**(**selectedMovie**.**getFormattedRevenue**());**

budgetFormattedTextField**.**setText**(**selectedMovie**.**getFormattedBudget**());**

**}**

**2** - Invoque o método fillFields() no final do método construtor e faça as devidas alterações como visto em aula:

**public** **AddOrEditMovieView(**java**.**awt**.**Frame parent**,** **boolean** modal**)** **{**

**super(**parent**,** modal**);**

controller **=** MoviesController**.**getSharedInstance**();**

paymentFormat **=** NumberFormat**.**getCurrencyInstance**();**

initComponents**();**

**if(**controller**.**getSelectedObject**()** **!=** **null)** **{**

setTitle**(**"Editar filme"**);**

fillForm**();**

**}**

**}**

**2** - Faça as devidas alterações no método do **botão salvar**, como visto em aula. O método deverá ficar assim:

**private** **void** **saveButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

String name **=** nameTextField**.**getText**();**

String originalName **=** originalNameTextField**.**getText**();**

String releaseDate **=** releaseDateFormattedTextField**.**getText**();**

String duration **=** durationFormattedTextField**.**getText**();**

Gender gender **=** **(**Gender**)** genderCombobox**.**getSelectedItem**();**

String synopsis **=** synopsisTextArea**.**getText**();**

String revenue **=** revenueFormattedTextField**.**getText**();**

String budget **=** budgetFormattedTextField**.**getText**();**

*//Checando campos obrigatórios*

**if** **(**name**.**isEmpty**()** **||**

originalName**.**isEmpty**()** **||**

releaseDate**.**isEmpty**()** **||**

duration**.**isEmpty**()** **||**

gender **==** **null** **||**

synopsis**.**isEmpty**()** **||**

revenue**.**isEmpty**()** **||**

budget**.**isEmpty**())** **{**

JOptionPane**.**showMessageDialog**(null,** "Existem campos vazios que devem ser preenchidos. \nTodos os campos são obrigatórios."**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}**

**else** **if** **(**Long**.**parseLong**(**duration**)** **<** 1**)** **{**

JOptionPane**.**showMessageDialog**(null,** "Duração do filme inválida. A duração não pode ser negativa"**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else** **if(**releaseDate**.**equals**(**" / / "**)){**

JOptionPane**.**showMessageDialog**(null,** "Lançamento inválido. A data deve estar no formato dd/MM/yyyy"**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else{**

**if(**selectedMovie **==** **null)** **{**

Movie movie **=** **new** Movie**(**name**,** originalName**,** synopsis**,** releaseDate**,** gender**,** budget**,** revenue**,** duration**);**

controller**.**save**(**movie**);**

JOptionPane**.**showMessageDialog**(null,** "Filme adicionado com sucesso. \nEdite-o para adicionar Elenco, Equipe e Produtoras"**,** "Sucesso"**,** DISPOSE\_ON\_CLOSE**);**

**}else{**

selectedMovie**.**setName**(**name**);**

selectedMovie**.**setOriginalName**(**originalName**);**

selectedMovie**.**setReleaseDate**(**releaseDate**);**

selectedMovie**.**setDuration**(**duration**);**

selectedMovie**.**setGender**(**gender**);**

selectedMovie**.**setBudget**(**budget**);**

selectedMovie**.**setRevenue**(**revenue**);**

selectedMovie**.**setSynopsis**(**synopsis**);**

controller**.**edit**(**selectedMovie**);**

**}**

setVisible**(false);**

dispose**();**

**}**

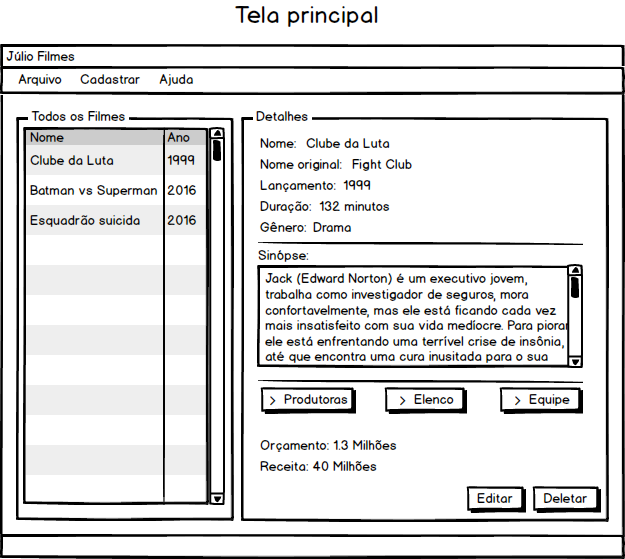
**}**

## 2 - Tela principal - DashboardView

### 2.1 - Criando a View

Antes de criar a View vamos lembrar o seu protótipo:

Figura 5. - Protótipo da Tela Principal



Crie então um JFrame com o nome DashboardView. Os nomes das variáveis foram escolhidos para facilitar a identificação do tipo dos componentes, por exemplo: JLabels contém o sufixo Label, JButtons contém o sufixo Button, o JTextArea contém o sufixo TextArea.

Aqui o procedimento é o mesmo dado na tela anterior. Insira os componentes, ajuste o tamanho do JFrame e nomeie as variáveis conforme a figura abaixo:

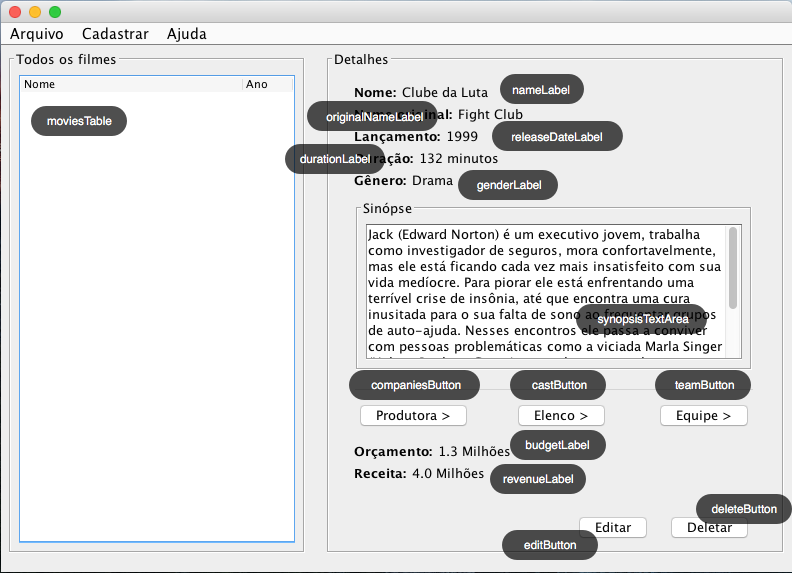
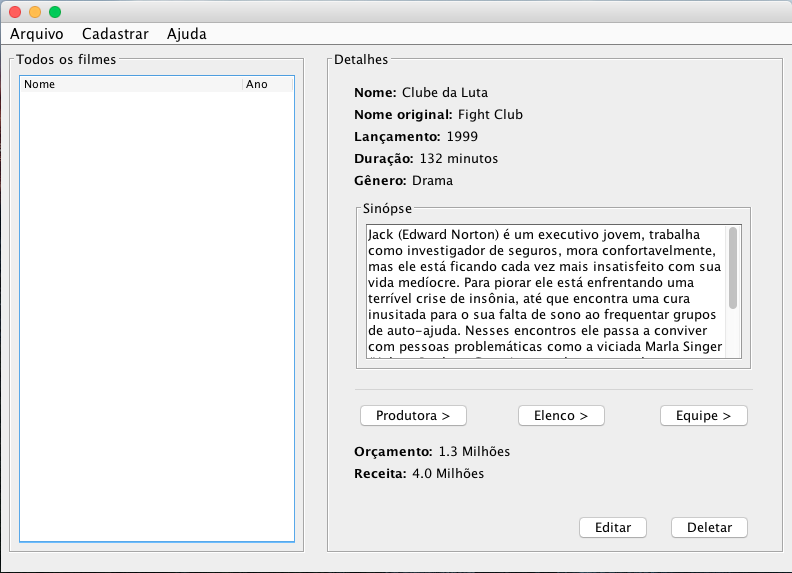


Figura 5. - Variáveis da Tela Principal

Lembre-se que o aluno poderá criar seu próprio estilo de tela, mas deverá respeitar a estrutura proposta no protótipo. Seguindo este exemplo, sua tela ficará parecida com esta:

Figura 5. - Mockup da Tela Principal



#### 2.1.2 - Criando os Controllers

Aqui teremos apenas as seguintes tarefas:

* Fazer a ligação entre as telas e exibir os itens na tabela
* Exibir os detalhes do item selecionado na tabela

Vamos começar ligando as telas DashboardView a AddOrEditMovieView.

Primeiro tenha em mente a estrutura de um menu:

Uma barra de menu JMenuBar possui menus JMenu, que por sua vez possui itens JMenuItem.

No JMenu **Cadastro** insira um JMenuItem com nome de variável registerMoviesMenuItem e altere seu text para “Filmes”. Ficará como na imagem abaixo:

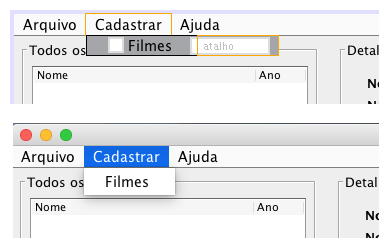


Figura 5. - JMenuItem filmes

##### 2.1.2.1. Fazendo a ligação entre as telas

Primeiro vamos instanciar uma variável controller do tipo MoviesController como feito em AddOrEditMoviesView.

**1 –** Crie um atributo do tipo MoviesController:

MoviesController controller **=** **null;**

**2 –** Inicialize-o no método construtor de DashboardView

**public** **DashboardView()** **{}**

initComponents**();**

*// ...*

controller **=** MoviesController**.**getSharedInstance**();**

**}**

Repetindo o procedimento de abrir uma tela ensinado em aula, dê duplo clique no botão Filmes do menu Cadastro e insira o seguinte código ao método deste botão:

**private** **void** **registerMoviesMenuItemActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{***//GEN-FIRST:event\_registerMoviesMenuItemActionPerformed*

controller**.**setSelectedObject**(null);**

AddOrEditMovieView addOrEditMovieView **=** **new** AddOrEditMovieView**(this,** rootPaneCheckingEnabled**);**

addOrEditMovieView**.**setVisible**(true);**

addOrEditMovieView**.**addWindowListener**(new** WindowAdapter**()** **{**

@Override

**public** **void** **windowClosed(**WindowEvent e**)** **{**

moviesTable**.**setModel**(**controller**.**getTableModel**());**

**}**

**});**

**}**

Isto bastará para mostrar os novos filmes adicionados, mas caso você queira deixar filmes pré-adicionados, como no exemplo dado em aula, **insira** a seguinte linha de código no método construtor da DashboardView abaixo da chamada do método initComponents():

moviesTable**.**setModel**(**controller**.**getTableModel**());**

Agora você pode testar a inserção de novos filmes.

##### 2.1.2.2. Exibindo os detalhes do item selecionado na tabela

Mais uma vez vamos seguir o exemplo dado em aula.

**1** - Na DashboardView em **modo Código-fonte** crie um método com o nome updatePanel() :

**public** **void** **updatePanel()** **{**

**int** selectedRow **=** moviesTable**.**getSelectedRow**();**

**if** **(**selectedRow **>=** 0 **&&** **!**controller**.**getList**().**isEmpty**())** **{**

Movie movie **=** **(**Movie**)** controller**.**getList**().**get**(**selectedRow**);**

controller**.**setSelectedObject**(**movie**);**

nameLabel**.**setText**(**movie**.**getName**());**

originalNameLabel**.**setText**(**movie**.**getOriginalName**());**

releaseDateLabel**.**setText**(**movie**.**getFormattedReleaseDate**());**

durationLabel**.**setText**(**movie**.**getFormattedDuration**());**

synopsisTextArea**.**setText**(**movie**.**getSynopsis**());**

genderLabel**.**setText**(**movie**.**getGender**().**getName**());**

budgetLabel**.**setText**(**movie**.**getFormattedBudget**());**

revenueLabel**.**setText**(**movie**.**getFormattedRevenue**());**

**}** **else** **{**

nameLabel**.**setText**(**""**);**

originalNameLabel**.**setText**(**""**);**

releaseDateLabel**.**setText**(**""**);**

durationLabel**.**setText**(**""**);**

synopsisTextArea**.**setText**(**""**);**

genderLabel**.**setText**(**""**);**

budgetLabel**.**setText**(**""**);**

revenueLabel**.**setText**(**""**);**

**}**

**}**

Este método irá preencher ou limpar as Labels de detalhes do filme.

**2** - Insira o **listener** de seleção da tabela e a chamada de updatePanel() no **método construtor**:

**public** **DashboardView()** **{**

initComponents**();**

*// Adiciona atores para nível de teste*

ActorsController**.**setActorsList**();**

controller **=** MoviesController**.**getSharedInstance**();**

moviesTable**.**setModel**(**controller**.**getTableModel**());**

updatePanel**();**

moviesTable**.**getSelectionModel**().**addListSelectionListener**(new** ListSelectionListener**()** **{**

@Override

**public** **void** **valueChanged(**ListSelectionEvent event**)** **{**

**if** **(**controller**.**getList**().**size**()** **>** 0**)** **{**

editButton**.**setEnabled**(true);**

deleteButton**.**setEnabled**(true);**

castButton**.**setEnabled**(true);**

teamButton**.**setEnabled**(true);**

companiesButton**.**setEnabled**(true);**

**}** **else** **{**

editButton**.**setEnabled**(false);**

deleteButton**.**setEnabled**(false);**

castButton**.**setEnabled**(false);**

teamButton**.**setEnabled**(false);**

companiesButton**.**setEnabled**(false);**

**}**

updatePanel**();**

**}**

**});**

**}**

##### 2.1.2.3. Deletando um filme

Para **deletar um filme** basta adicionar o seguinte código no método do botão deleteButton:

**private** **void** **deleteButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**int** selectedRow **=** moviesTable**.**getSelectedRow**();**

**if** **(**selectedRow **>=** 0**)** **{**

Movie movie **=** **(**Movie**)** controller**.**getList**().**get**(**selectedRow**);**

controller**.**delete**(**movie**);**

**}**

moviesTable**.**setModel**(**controller**.**getTableModel**());**

controller**.**setSelectedObject**(null);**

updatePanel**();**

**if** **(**controller**.**getList**().**size**()** **>** 0**)** **{**

editButton**.**setEnabled**(true);**

deleteButton**.**setEnabled**(true);**

castButton**.**setEnabled**(true);**

teamButton**.**setEnabled**(true);**

companiesButton**.**setEnabled**(true);**

**}** **else** **{**

editButton**.**setEnabled**(false);**

deleteButton**.**setEnabled**(false);**

castButton**.**setEnabled**(false);**

teamButton**.**setEnabled**(false);**

companiesButton**.**setEnabled**(false);**

**}**

**}**

##### 2.1.2.4. Editando um filme

Também é simples. Basta seguir o exemplo dado em aula e fazer as devidas alterações. O método do **botão editar** fica da seguinte maneira:

**private** **void** **editButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**int** selectedIndex **=** moviesTable**.**getSelectedRow**();**

**if** **(**selectedIndex **>=** 0**)** **{**

Movie movie **=** **(**Movie**)** controller**.**getList**().**get**(**selectedIndex**);**

AddOrEditMovieView addOrEditMovieView **=** **new** AddOrEditMovieView**(this,** rootPaneCheckingEnabled**);**

controller**.**setSelectedObject**(**movie**);**

addOrEditMovieView**.**setVisible**(true);**

addOrEditMovieView**.**addWindowListener**(new** WindowAdapter**()** **{**

@Override

**public** **void** **windowClosed(**WindowEvent e**)** **{**

moviesTable**.**setModel**(**controller**.**getTableModel**());**

**}**

**});**

**}**

**}**

## 3 – Cadastro de Pessoas

Na nossa arquitetura classificamos como TeamMember as pessoas que exercem atividade profissional sobre o filme. Para adicionarmos um elenco e uma equipe ao filme, primeiro devemos cadastrar estas pessoas.

### 3.1 – Criando a View

Vamos relembrar do mockup desta View:

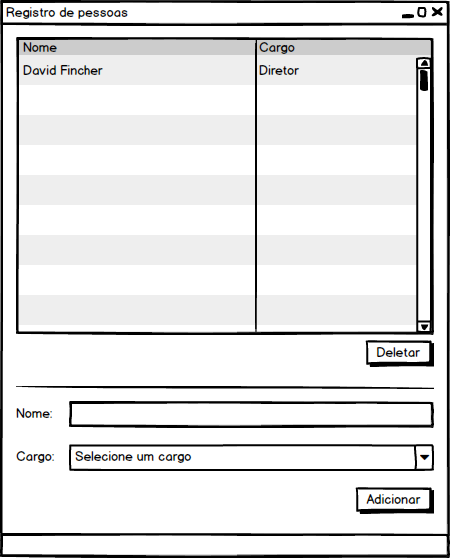


Figura 5. - Tela cadastro de pessoas

Nas aulas 1 e 2 desta unidade foi pedido a criação das Views, então você deverá ter uma tela semelhante a esta, que nomeamos de PeopleView:

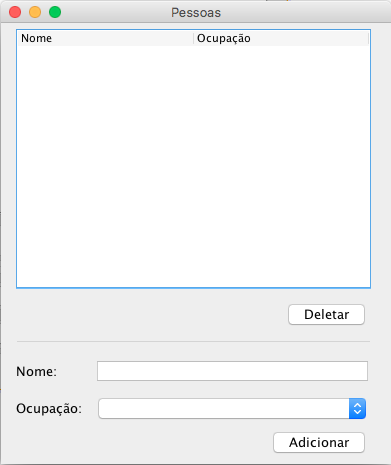


Figura 5. - Mockup tela Pessoas

Antes de começarmos os controllers, mude o nome das variáveis para corresponder a imagem abaixo:

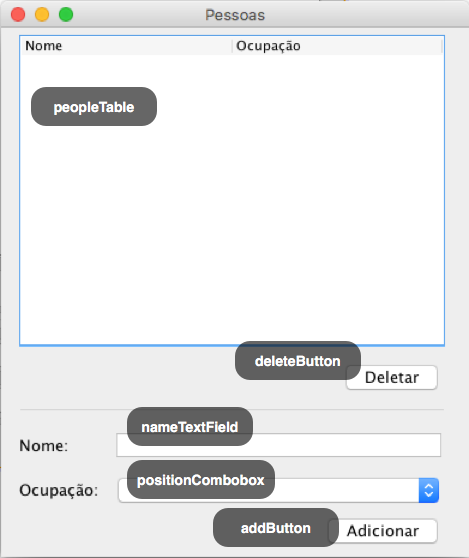


Figura 5. - Variáveis de cadastro de pessoas

Repare que temos uma seção abaixo do botão **Deletar.**  Esta seção é equivalente à tela auxiliar que se abria para cadastrar/editar no exemplo da agenda de contatos. Faremos o cadastro, deleção e exibição na tabela tudo aqui nesta mesma View.

### 3.2 – Criando os controllers

Este Controller será aos moldes do MoviesController. Executaremos passos semelhantes:

**1 –** Crie um arquivo chamado PeopleController que extenda de GenericTableController.

**public** **class** **PeopleController** **extends** GenericTableController **{**

**}**

**2 –** Transforme a classe em Singleton:

**public** **class** **PeopleController** **extends** GenericTableController **{**

**private** **static** PeopleController sharedInstance **=** **null;**

**public** **static** PeopleController **getSharedInstance()** **{**

**if(**sharedInstance **==** **null)** **{**

sharedInstance **=** **new** PeopleController**();**

**}**

**return** sharedInstance**;**

**}**

**}**

**3 –** Implemente os métodos abstratos. Neste caso não precisaremos definir nenhum comportamento a ele:

@Override

**protected** **void** **doAdditionalUpdated()** **{** **}**

**4 –** Implemente o método construtor (adicionamos algumas pessoas para teste).

**private** **PeopleController()** **{**

**super();**

list**.**add**(new** TeamMember**(**"Leonardo Di Caprio"**,** Position**.**Actor**));**

list**.**add**(new** TeamMember**(**"Scarlett Johansson"**,** Position**.**Actor**));**

list**.**add**(new** TeamMember**(**"Sofia Coppola"**,** Position**.**Writer**));**

list**.**add**(new** TeamMember**(**"Will Smith"**,** Position**.**Actor**));**

list**.**add**(new** TeamMember**(**"Steven Spielberg"**,** Position**.**Director**));**

tableModel**.**list **=** list**;**

tableModel**.**setCols**(new** String**[]** **{**"Nome"**});**

updateRows**();**

**}**

**5 –** Futuramente você precisará utilizar este Controller para obter todos os não atores na tela de equipe do filme. Então crie o seguinte método:

**public** Object**[]** **getNonActorComboboxModel()** **{**

ArrayList**<**TableModel**>** people **=** **new** ArrayList**<>();**

**for(**TableModel object**:** list**)** **{**

TeamMember person **=** **(**TeamMember**)** object**;**

**if(**person**.**getPosition**()** **!=** Position**.**Actor**)** **{**

people**.**add**(**person**);**

**}**

**}**

**return** people**.**toArray**();**

**}**

Este retorna um array primitivo pois será utilizado em um combobox na tela de Equipe do filme.

Veja a classe completa no material de apoio.

#### 3.2.1 – Interligando a View ao Controller

Mesmo procedimento para o AddOrEditMovieView.

**1 –** Na tela DashboardView insira mais um JMenuItem, agora com o nome **Pessoas**, no menu **Cadastrar**:

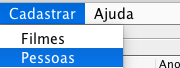


Figura 5. - Menu Pessoas

**2 –** Dê duplo clique no JMenuItem Pessoas para inserir uma ação nele. Ele deverá abrir a tela PeopleView:

**private** **void** **personMenuItemActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)**

PeopleView peopleView **=** **new** PeopleView**(this,** rootPaneCheckingEnabled**);**

peopleView**.**setVisible**(true);**

**}**

**3 –** Na PeopleView clique na positionCombobox e configure seu model no painel de propriedades:



Figura 5. - Utilizando o enum Position para popular o positionCombobox

Não esqueça de entrar no modo **Código-fonte**  e fazer todos os imports que a IDE pede.

**4 –** Crie e inicialize um atributo do tipo PeopleController:

**public** **class** **PeopleView** **extends** javax**.**swing**.**JDialog **{**

PeopleController controller **=** **null;**

**public** **PeopleView(**java**.**awt**.**Frame parent**,** **boolean** modal**)** **{**

**super(**parent**,** modal**);**

initComponents**();**

controller **=** PeopleController**.**getSharedInstance**();**

**}**

*// ...*

**}**

**5 –** Inicialize o model da tabela também.

**public** **class** **PeopleView** **extends** javax**.**swing**.**JDialog **{**

*// ...*

**public** **PeopleView(**java**.**awt**.**Frame parent**,** **boolean** modal**)** **{**

*// ...*

peopleTable**.**setModel**(**controller**.**getTableModel**());**

**}**

*// ...*

**}**

**6 –** Adicione o listener de seleção da tabela. O método construtor ficará assim:

**public** **PeopleView(**java**.**awt**.**Frame parent**,** **boolean** modal**)** **{**

**super(**parent**,** modal**);**

initComponents**();**

controller **=** PeopleController**.**getSharedInstance**();**

peopleTable**.**setModel**(**controller**.**getTableModel**());**

deleteButton**.**setEnabled**(false);**

peopleTable**.**getSelectionModel**().**addListSelectionListener**(new** ListSelectionListener**()** **{**

@Override

**public** **void** **valueChanged(**ListSelectionEvent event**)** **{**

**int** selectedRow **=** peopleTable**.**getSelectedRow**();**

**if(**selectedRow **>=** 0**)** **{**

deleteButton**.**setEnabled**(true);**

**}**

**}**

**});**

**}**

**7 -**  Adicione os métodos para deletar e salvar uma nova pessoa (nos botões **Adicionar** e **Deletar**, respectivamente):

**private** **void** **deleteButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**int** selectedRow **=** peopleTable**.**getSelectedRow**();**

**if** **(**selectedRow **>=** 0**)** **{**

TeamMember person **=** **(**TeamMember**)** controller**.**getList**().**get**(**selectedRow**);**

controller**.**delete**(**person**);**

**}**

peopleTable**.**setModel**(**controller**.**getTableModel**());**

controller**.**setSelectedObject**(null);**

deleteButton**.**setEnabled**(false);**

**}**

**private** **void** **saveButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**if(**nameTextField**.**getText**().**isEmpty**())** **{**

JOptionPane**.**showMessageDialog**(null,** "Informe o nome da pessoa."**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else{**

TeamMember person **=** **new** TeamMember**(**nameTextField**.**getText**());**

person**.**setPosition**((**Position**)** positionCombobox**.**getSelectedItem**());**

controller**.**save**(**person**);**

JOptionPane**.**showMessageDialog**(null,** "Pessoa adicionada com sucesso"**,** "Sucesso"**,** DISPOSE\_ON\_CLOSE**);**

peopleTable**.**setModel**(**controller**.**getTableModel**());**

**}**

**}**

Nosso Controller de pessoas está pronto. Este Controller guarda todas as pessoas, mas como faremos para separar Atores e Membros da equipe de cada filme? Siga para próxima etapa.

## 4 – Cadastro de Elenco

Cadastro de elenco e de equipe se assemelham. Ambos terão que utilizar as pessoas cadastradas em PeopleController.

### 4.1 – Criando a View

Primeiro relembre o protótipo da tela de cadastro de elenco:

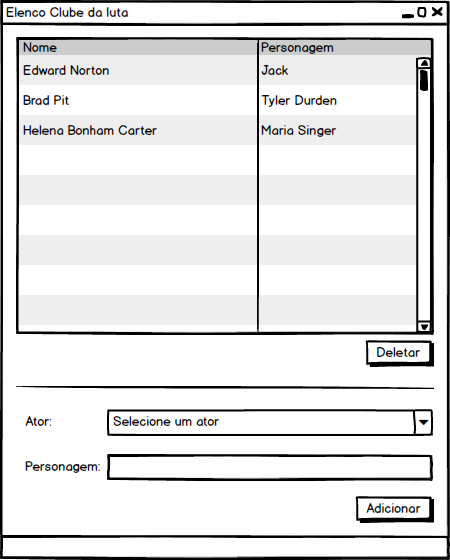


Figura 5. - Protótipo Elenco

As criações das Views foram pedidas nas aulas anteriores, então sua tela deverá se assemelhar com esta:

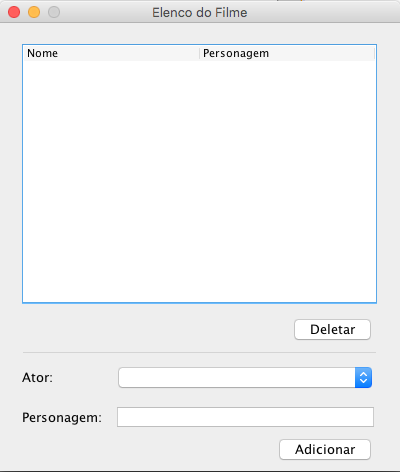


Figura 5. - Mockup tela Elenco

**1 –** Renomeie suas variáveis para corresponder com a imagem abaixo:

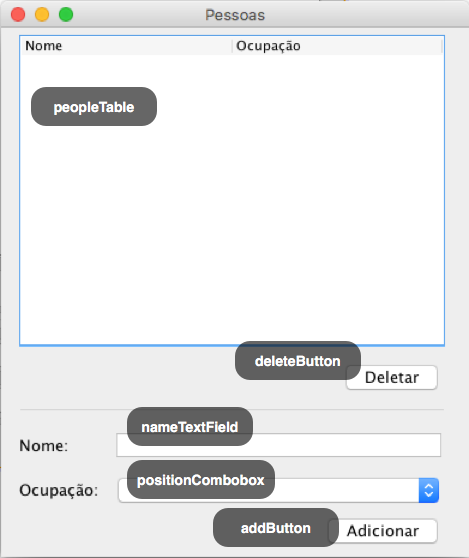


Figura 5. - Variáveis da tela de Elenco

### 4.2 – Criando os controllers

Este Controller será aos moldes do MoviesController. Executaremos passos semelhantes:

**1 –** Crie um arquivo chamado CastController que extenda de GenericTableController.

**public** **class** **CastController** **extends** GenericTableController **{**

**}**

**2 –** Transforme a classe em Singleton:

**public** **class** **CastController** **extends** GenericTableController **{**

**private** **static** CastController sharedInstance **=** **null;**

**public** **static** CastController **getSharedInstance()** **{**

**if(**sharedInstance **==** **null)** **{**

sharedInstance **=** **new** CastController **();**

**}**

**return** sharedInstance**;**

**}**

**}**

**3 –** Crie uma variável do tipo MoviesController para acessarmos o objeto selecionado e manipular sua lista do elenco:

**private** MoviesController moviesController **=** **null;**

**4 –** Implemente os métodos abstratos. Este irá atualizar tanto a lista do MoviesController (setCast()) quanto a lista da tabela:

@Override

**protected** **void** doAdditionalUpdated**()** **{**

**((**Movie**)** moviesController**.**getSelectedObject**()).**setCast**(**list**);**

list **=** **((**Movie**)** moviesController**.**getSelectedObject**()).**getCast**();**

tableModel**.**list **=** list**;**

updateRows**();**

**}**

**5 –** Implemente o método construtor. O intuito deste Controller é manipular a lista do elenco do filme selecionado:

**private** CastController**()** **{**

**super();**

moviesController **=** MoviesController**.**getSharedInstance**();**

list **=** **((**Movie**)** moviesController**.**getSelectedObject**()).**getCast**();**

tableModel**.**list **=** list**;**

tableModel**.**setCols**(new** String**[]** **{**"Nome"**,** "Personagem"**});**

updateRows**();**

**}**

Veja a classe completa no material de apoio.

Estes passos foram responsáveis por exibir o elenco na tabela da CastView. Agora temos que exibir a lista de atores no actorCombobox da CastView.

#### 4.2.1 – Controller de atores

Iremos criar um Controller para atores que estenderá de GenericListController para ser exibido no combobox. Repare que ele é dinâmico, ou seja, não possui um número fixo de atores. A única coisa que precisaremos fazer é definir que seus **values** serão atores provenientes do PeopleController.

**1 -**  Crie uma classe com nome ActorsController que estenda de GenericListController:

**public** **class** **ActorsController** **extends** GenericListController **{**

**}**

**2 –** Como o values deste Controller será dinâmico, vamos criar um ArrayList para manipular esta lista com mais facilidade:

**public** **class** **ActorsController** **extends** GenericListController **{**

**public** **static** ArrayList**<**TableModel**>** list **=** **new** ArrayList**<>();**

**}**

**3 –** Crie um método de inicialização para esta lista (setter):

**public** **static** **void** **setActorsList()** **{**

ArrayList**<**TableModel**>** personList **=** PeopleController**.**getSharedInstance**().**list**;**

list **=** **new** ArrayList**<>();**

**for(**TableModel object**:** personList**)** **{**

TeamMember person **=** **(**TeamMember**)** object**;**

**if(**person**.**getPosition**().**equals**(**Position**.**Actor**)){**

list**.**add**(**person**);**

**}**

**}**

**if(!**list**.**isEmpty**())** **{**

selection **=** list**.**get**(**0**);**

**}**

setValuesFromObject**(**list**);**

**}**

Este será utilizado na tela CastView para verificar se já existe ator cadastrado, por isto é estático:

**4 –** Invoque o método acima no construtor:

**public** **ActorsController()** **{**

setActorsList**();**

**}**

**5 –** Implemente os métodos abstratos:

@Override

**public** TableModel **getObjectFromString(**String string**)** **{**

**for(**TableModel object**:** list**)** **{**

TeamMember actor **=** **(**TeamMember**)** object**;**

**if(**actor**.**getName**().**equals**(**string**))** **{**

**return** actor**;**

**}**

**}**

**return** **null;**

**}**

Veja a classe completa no material de apoio.

Agora podemos exibir os atores no actorCombobox.

#### 4.2.1 – Interligando as Views e Controllers

Primeiro vamos invocar a tela de elenco:

**1 –** Em DashboardView, dê duplo clique no botão castButton e insira o seguinte código:

**private** **void** **castButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**if** **(**ActorsController**.**list**.**isEmpty**())** **{**

JOptionPane**.**showMessageDialog**(null,** "Não há nenhum ator registrado. Registre ao menos um ator antes de adicionar o elenco."**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}** **else** **{**

CastView castView **=** **new** CastView**(this,** rootPaneCheckingEnabled**);**

castView**.**setVisible**(true);**

**}**

**}**

**2 –** Inicialize o ActorsController no construtor:

**public** **DashboardView()** **{**

initComponents**();**

ActorsController**.**setActorsList**();**

*// ...*

**}**

Veja a classe DashboardView completa no material de apoio.

**3 –** Em CastView, configure o model de actorCombobox para receber os valores de ActorsController:

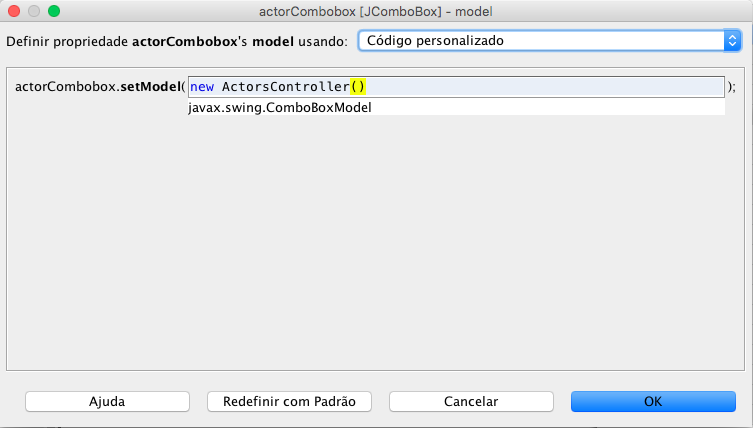


Figura 5. - Configurando o model de actorCombobox

Não se esqueça de importar tudo que a IDE pede.

**4 –** Crie um atributo para CastController, inicialize-o e inicialize o model da tabela:

**public** **class** **CastView** **extends** javax**.**swing**.**JDialog **{**

CastController controller **=** **null;**

**public** **CastView(**java**.**awt**.**Frame parent**,** **boolean** modal**)** **{**

**super(**parent**,** modal**);**

initComponents**();**

Movie movie **=** **(**Movie**)**MoviesController**.**getSharedInstance**().**getSelectedObject**();**

controller **=** CastController**.**getSharedInstance**();**

controller**.**setList**(**movie**.**getCast**());**

castTable**.**setModel**(**controller**.**getTableModel**());**

**}**

}

Esta inicialização garante que você utilize a lista cast do filme selecionado.

**5 –** Adicione os listeners de seleção da tabela:

**public** **CastView(**java**.**awt**.**Frame parent**,** **boolean** modal**)** **{**

*// ...*

deleteButton**.**setEnabled**(false);**

castTable**.**getSelectionModel**().**addListSelectionListener**(new** ListSelectionListener**()** **{**

@Override

**public** **void** **valueChanged(**ListSelectionEvent event**)** **{**

**int** selectedRow **=** castTable**.**getSelectedRow**();**

**if(**selectedRow **>=** 0**)** **{**

deleteButton**.**setEnabled**(true);**

**}**

**}**

**})****;**

**}**

**6 -**  Por fim implemente os métodos do botão **Adicionar** e  **Deletar.**

**private** **void** **deleteButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**int** selectedRow **=** castTable**.**getSelectedRow**();**

**if** **(**selectedRow **>=** 0**)** **{**

CastMember cast **=** **(**CastMember**)** controller**.**getList**().**get**(**selectedRow**);**

controller**.**delete**(**cast**);**

**}**

castTable**.**setModel**(**controller**.**getTableModel**());**

controller**.**setSelectedObject**(null);**

deleteButton**.**setEnabled**(false);**

**}**

**private** **void** **addButtonActionPerformed(**java**.**awt**.**event**.**ActionEvent evt**)** **{**

**if(**characterTextField**.**getText**().**isEmpty**())** **{**

JOptionPane**.**showMessageDialog**(null,** "Especifique o nome do personagem."**,** "Atenção"**,** DISPOSE\_ON\_CLOSE**);**

**}else{**

TeamMember person **=** **(**TeamMember**)** actorCombobox**.**getSelectedItem**();**

Actor actor **=** **new** Actor**(**person**.**getName**());**

CastMember cast **=** **new** CastMember**(**actor**.**getName**(),** characterTextField**.**getText**());**

controller**.**save**(**cast**);**

JOptionPane**.**showMessageDialog**(null,** "Personagem adicionado com sucesso."**,** "Sucesso"**,** DISPOSE\_ON\_CLOSE**);**

castTable**.**setModel**(**controller**.**getTableModel**());**

**}**

**}**

**Veja o código completo no material de apoio.**

## 5 - Para onde partir daqui

A partir daqui não teremos novidades. As tarefas se tornarão cada vez mais repetitivas. É uma grande chance de você, educador, treinar o seu domínio sobre a linguagem. Disponibilizaremos o aplicativo pronto e funcionando para que você tome como referência, mas é de grande importância que você finalize por você mesmo, pois desta maneira você conseguira depurar e resolver os erros de programação. Isto é importante pois os alunos aparecerão com muitos problemas e erros no código, e se você não tiver um domínio maior da aplicação e não estiver habituado a pesquisar e depurar, não conseguirá ajudar o aluno. Este final de projeto é uma boa oportunidade para treinar isto. Para ajudar, mostraremos uma ordem de execução para que você possa se guiar e não se perder no meio de tantos arquivos.

1. Tela de cadastro de Produtoras
   1. Mesmo procedimento do cadastro de Pessoas
   2. Crie o Controller de Produtoras e Países antes
2. Controller de Produtoras
   1. Mesmo procedimento do Controller de Pessoas.
   2. Crie o Controller de Países antes=
3. Controller de Países
   1. Mesmo procedimento do Controller de Gêneros
4. Tela de cadastro de equipe no filme
   1. Mesmo procedimento da tela de Elenco
   2. Crie o Controller de Equipe primeiro
5. Controller de Equipe
   1. Mesmo procedimento do Controller de Elenco
6. Tela de cadastro de produtoras no filme
   1. Mesmo procedimento da tela de Elenco
   2. Crie o Controller de Produtoras primeiro
7. Controller de Produtoras
   1. Mesmo procedimento do Controller de Elenco

Utilize os Mockups e protótipos para basear-se.