# Unidade 5

Nesta aula será apresentada as ferramentas do Java Swing, que tornará nossos programas em Java visualmente atraentes. Você terá diversas telas exemplificadas e a livre criatividade é bem-vinda e trará aplicações bonitas e funcionais. Deve-se ter atenção especial nas inserções de códigos e os erros aparentes, alguns serão ignorados, já outros, em sua maioria, deverão 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 da 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

Educador, é interessante apresentar uma atividade para prototipagem do programa proposto. Lembre-se que por se tratar de um protótipo, não necessita utilizar muito tempo nesta tarefa e prender-se aos detalhes.

#### - Tela principal

Explique que na tela principal constará uma tabela com a lista de contatos, podendo ser direcionado a uma nova tela em que poderá ser realizado um novo cadastro, visualização de dados, exclusão e edição de cadastro.

#### - Tela de cadastro

Fale que nesta tela constarão os campos de textos para preenchimento de dados do cadastro, com a opção de salvar ou cancelar.

#### - Tela de edição

Nesta tela constarão os campos de texto para edição dos dados cadastrados, com a opção de salvar ou cancelar a edição. Esta tela 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

Educador, cite algumas classes mais comuns utilizadas no pacote Swing como as classes JPanel, JFrame e JTable.

#### 1.2.1 - Tela principal

Educador, nesta tela constará botões para Cadastrar, Editar e Excluir um contato. O nome dos contatos cadastrados e Rótulos com detalhes de Nome, Email, Endereço e Grupo serão visualizados em uma tabela. 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 constará uma caixa suspensa com campos de texto para edição e cadastro e, as opções salvar e cancelar.

#### 1.2.3 - Relacionamento entre as telas

Educador, é importante 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

Com o objetivo de facilitar e identificar com agilidade os possíveis erros citados na aula e outros decorrentes dos alunos, utilize 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

Educador, a primeira tela criada será uma JDialog, ou seja, possui seu próprio método, fazendo com que ela seja executada independentemente de outros serviços. Crie um JDialog e o renomeie 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

Apresente as janelas do Interface Builder e introduza a paleta de elementos.

#### 1.3.4 – Paleta de elementos

Explique que a paleta de elementos são os componentes que o aluno conseguirá compor qualquer tipo de aplicação.

##### 1.3.4.1 – Jlabel

Ensine ao aluno que o JLabel é um rótulo ou um texto simples que exibe informações na tela, não interagível.

##### 1.3.4.2 – JtextField

Já 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:

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

##### 1.3.4.4 – Jbutton

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

##### 1.3.4.5 – JPanel

Ensine ao aluno que JPanel é um componente que agrupa outros componentes.

##### 1.3.4.6 – JDialog

Educador, esclareça que 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

Explique que a JFrameé a classe principal da aplicação e só pode haver uma JFrame sendo executado por vez, ao contrário do JDialog.

##### 1.3.4.7 – JTable

Informe que 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, sendoseu trabalho dividido em Model, View e Controller.

##### 1.3.4.8 – JMenuBar

Ensine ao aluno que JMenuBar é uma barra de menu que geralmente se encontra no topo dos programas.

##### 1.3.4.9 – JMenu

Educardo, informe ao aluno que 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, por fim, a classe JMenultemé um botão posicionado em uma lista.

#### 1.3.4 - Inserindo elementos na tela

Educador, ensine que **p**ara adicionar os elementos requisitados, basta clicar e arrastar o componente para sua tela de projeto.

#### 1.3.5 – Modificando as propriedades

**Informe ao aluno que é possível** encontrar diversas opções de customização para telas, no entanto, elas 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.

### 1.5 - Exercícios

### 

### **As perguntas têm por objetivo fixar os principais conceitos abordados durante a aula. Para isso, viabilize o tempo necessário para o aluno responder às perguntas, acompanhando-os nas dúvidas. Para correção das perguntas, considere as respostas a seguir:**

|  |  |
| --- | --- |
| **1.** | **Qual é o intuito dos protótipos?** |
|  | Resposta: Os protótipos têm como intuito validar de maneira rápida como os itens de interface se organizarão na tela. |
| **2.** | **O que são os Mockups?** |
|  | Resposta: Mockups 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,quando 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: Layout são as partes do Swing responsáveis por dispor os elementos na tela, sendo eles os Layout Managers. Os tipos de layouts e suas diferenças podem ser:  **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 ideia 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. |

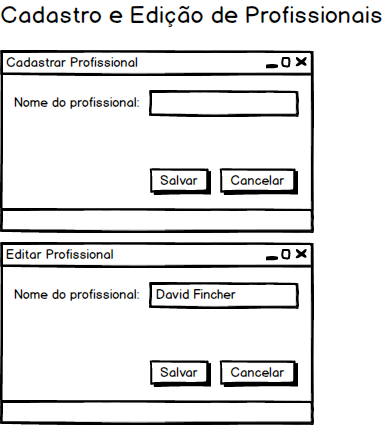
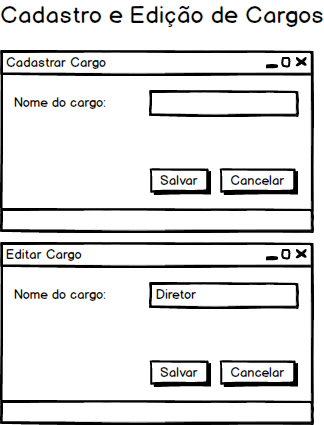
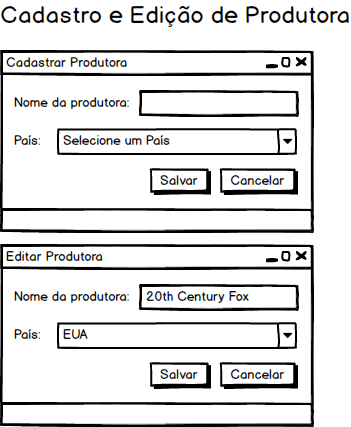
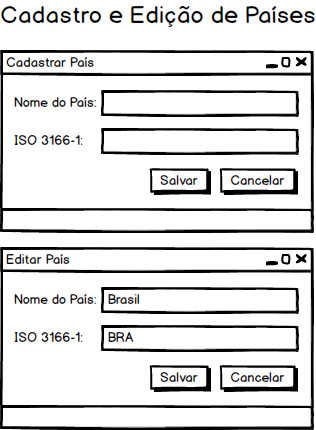
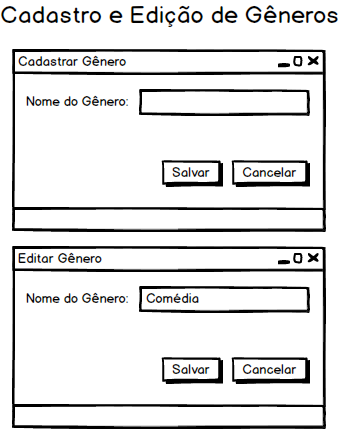
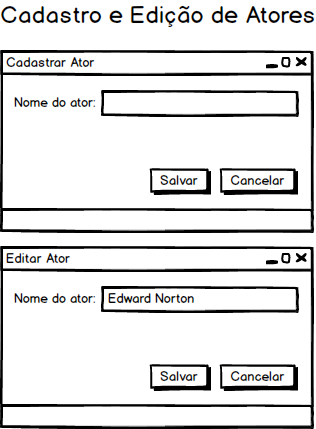
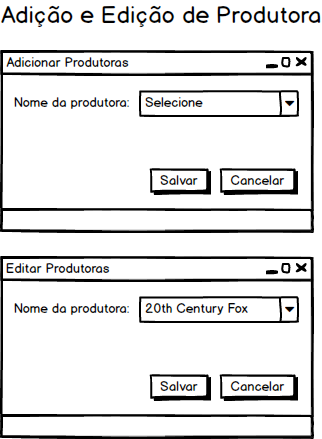
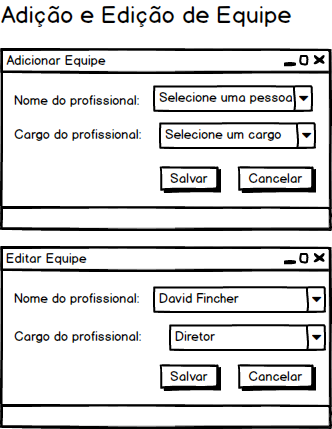
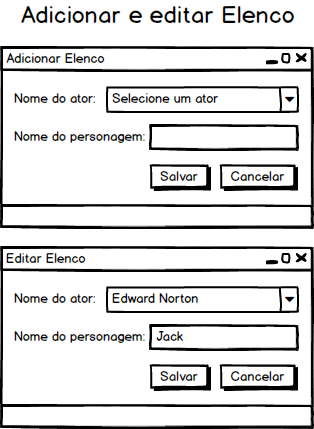
Para mais informações sobre layouts, confira a documentação Java em: <https://docs.oracle.com/javase/tutorial/uiswing/layout/index.html> .

### 1.6 – TDP

Ainda considerando a teoria relacionada à prática, é necessário que você seja um orientador do processo de desenvolvimento desta atividade, garantindo que o aluno tenha o conhecimento necessário para tornar-se um profissional qualificado para o mercado de trabalho.

**Seus protótipos abriram um leque de inspiração para Mariana. Com isso ela teve diversas ideias e um protótipo foi aprovado. Veja a primeira parte de protótipos:**

### 



#### Tarefa

**Agora que você tem os protótipos que definem a estrutura das views e 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.**

Agora que aluno tem os protótipos que definem a estrutura das views, ele deverá criar suas próprias telas. Peça para que ele utilize 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. O passo a passo para a construção de cada tela/funcionalidade e a solução do TDP constam na aula três dessa unidade.

### CONTEÚDOS COMPLEMENTARES

Você, Educador, deve agregar novos conhecimentos sobre os assuntos desta aula. Conheça algumas sugestões de conteúdo:

* Leia a 8a edição (2010) da obra **Java – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall.  Este livro apresenta: aulas desde básicas até avançadas sobre programação Java. Esta é a unidade final do curso. Supomos que o aluno não terá mais dificuldades com a linguagem Java, então este livro será um bom cookbook para você, educador, e o aluno consultar soluções mais sofisticadas para os problemas desta aula.
* Leia a 1a edição (2006) da obra **Swing: A Beginners Guide**, do autor Gilleanes T. A. Guedes, publicado pela editora McGraw-Hill. Este livro contém: material complementar para desenvolvimento em Swing. Este livro poderá servir como referência na execução dos TDPs
* Para mais informações sobre layouts, confira a documentação Java em: <https://docs.oracle.com/javase/tutorial/uiswing/layout/index.html> .

### REFERÊNCIAS

* GUEDES, GILLEANES T. A. **Swing: A Beginners Guide.** McGraw-Hill. 1a Edição, 2006.

### Conheça todas as referências utilizadas pelo autor para elaboração dessa aula no Livro do Aluno:

* NETBEANS. **Swing**. Disponível em: <https://netbeans.org/features/java-on-client/swing\_pt\_BR.html>. Acessado em 13 dec. 2016.
* ORACLE. **Lesson: Laying Out Components Within a Container.** Disponível em: <https://docs.oracle.com/javase/tutorial/uiswing/layout/index.html>. Acesso em: 15 dec. 2016.

## 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 e visual surgirão, alguns alunos são menos detalhistas não se atentam ao básico proposto e necessitam de uma 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

Dado o modelo de dados em diagrama, este tópico apresentará sua implementação. Uma dica para implementar os models diagramados é sempre tê-los em mãos e dominar sua leitura. Então, é importante que você saiba interpretar com domínio os diagramas de classes, como ensinado na unidade passada.

**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. Educador, explique ao aluno que enumeração é como se déssemos apelidos a certos números, simplesmente para não os manipular diretamente, sendo 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 é o melhor caminho. Na maioria das vezes, o melhor é ignorar os alertas do tipo “warning” e em caso de erros de compilação, não conserte 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.

### 2.4 – Exercícios

### 

### **As perguntas têm por objetivo fixar os principais conceitos abordados durante a aula. Para isso, viabilize o tempo necessário para o aluno responder às perguntas, acompanhando-os nas dúvidas. Para correção das perguntas, considere as respostas a seguir:**

**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 sobrescrevê-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?**

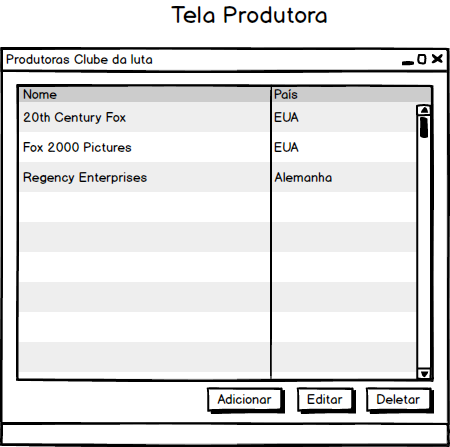
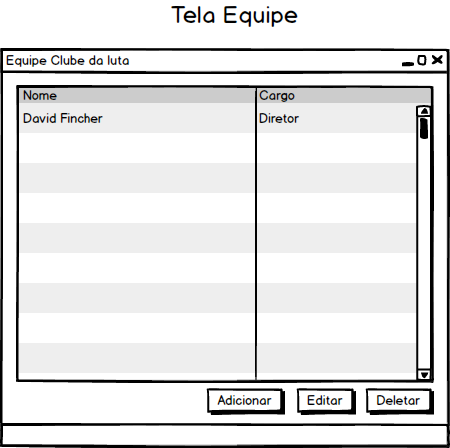
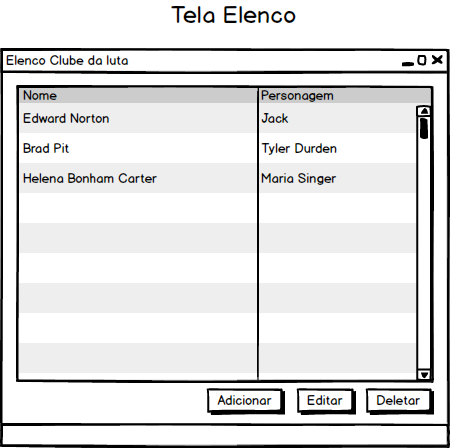
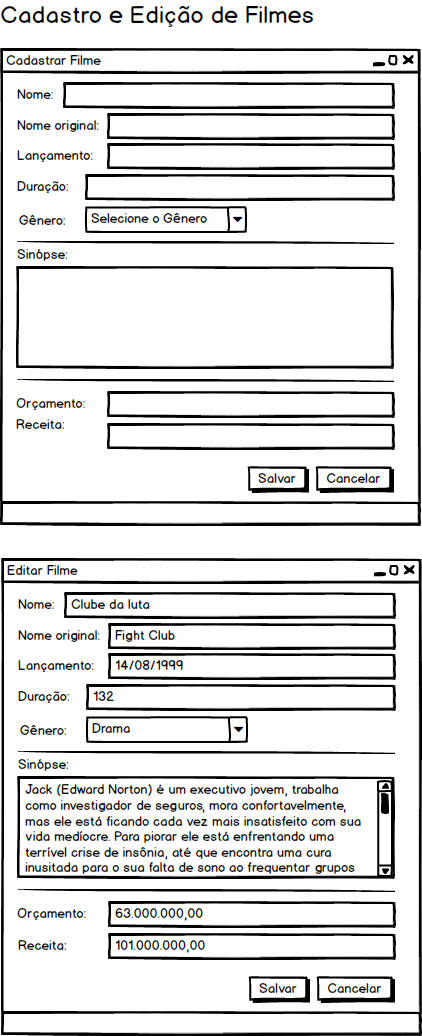
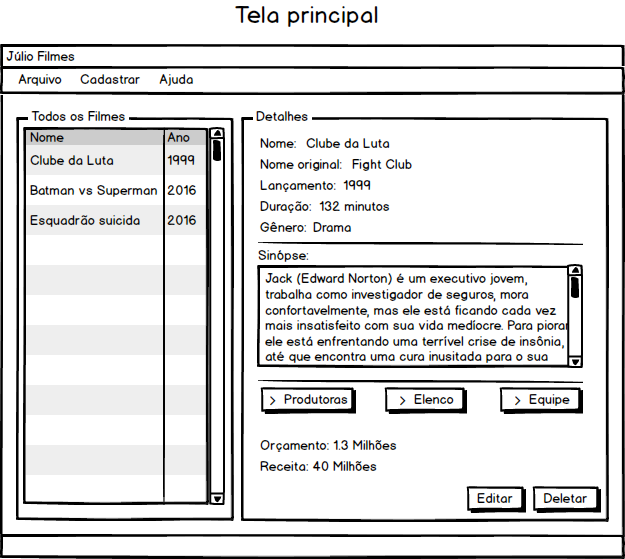
Resposta: Se o tipo do objeto não for especificado, um número hexadecimal que representa o endereço de memória seria mostrado, ou seja, um código estranho sem serventia para o seria mostrado.

### 2.5 – TDP

Ainda considerando a teoria relacionada à prática, é necessário que você seja um orientador do processo de desenvolvimento desta atividade, garantindo que o aluno tenha o conhecimento necessário para tornar-se um profissional qualificado para o mercado de trabalho.

#### 2.5.1 - A criação das telas

**Pode ser um pouco cansativa a criação das telas. Agora sobraram as telas principais do aplicativo. São as mais desafiadoras. Veja a segunda parte de protótipos:**



**Agora você e seu amigo deverão analisar bem essa segunda demanda de protótipos e dar vida a eles também.**

#### Tarefa

**O módulo de POO está chegando ao FIM!**

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

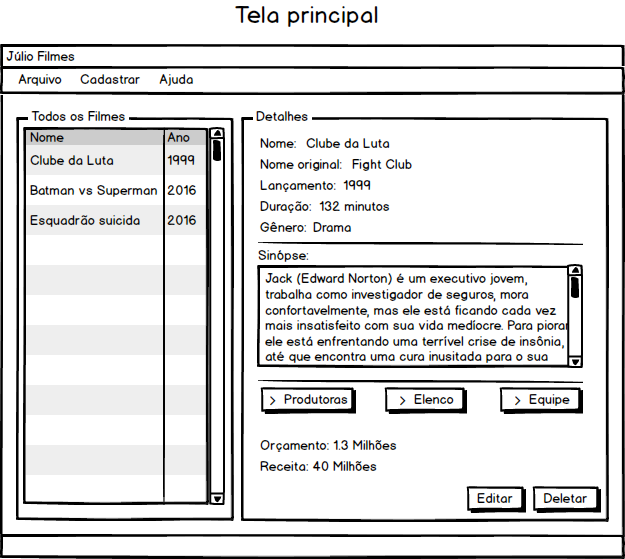
### SUGESTÃO:

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

## 1 - Tela principal - DashboardView

Antes de você criar a View, relembre o seu protótipo:

Figura 5. 9- Protótipo da Tela Principal



Crie 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 e o JTextArea contém o sufixo TextArea.

Esse 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:

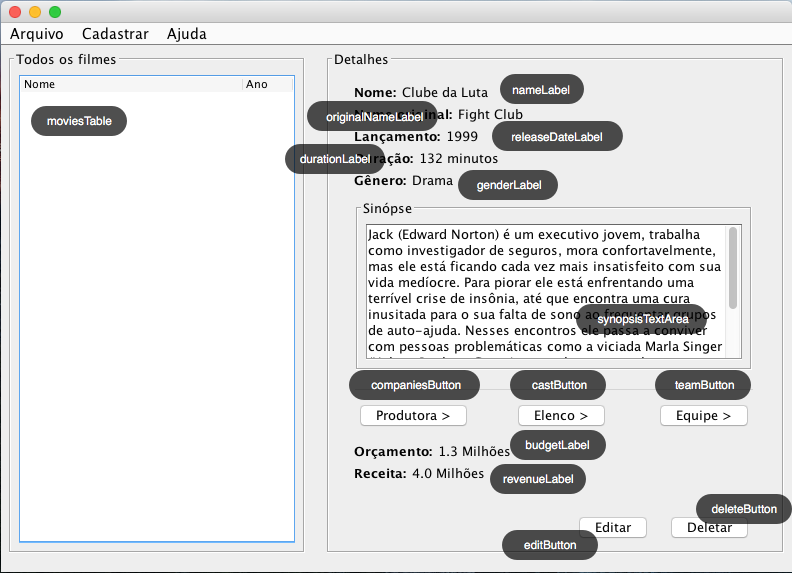
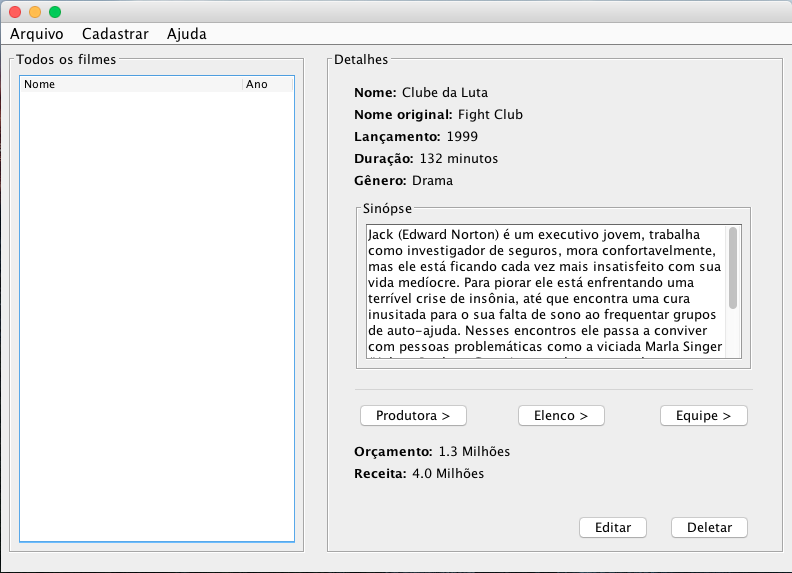


Figura 5. 5- Variáveis da Tela Principal

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. 11- Mockup da Tela Principal



## 2 - Cadastro de filmes

Esta é uma das telas principais do nosso app e com ela existe muitas dependências. Como já foi criado todos os Models, criare as Views e altere os models conforme for necessário

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

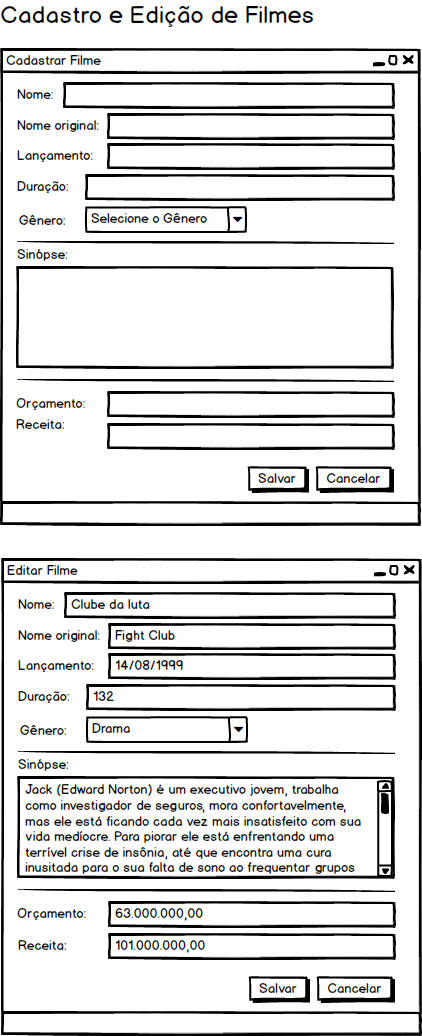


Figura 5. 4 – Cadastrar e Editar filmes

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:

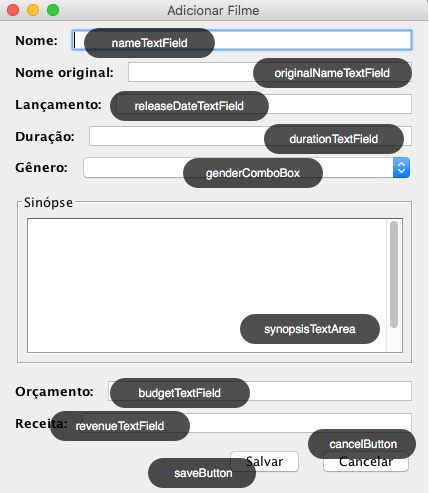


Figura 5. 5 - Adicionar Filme

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:

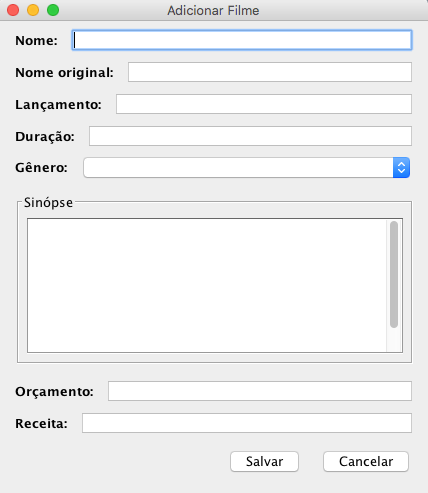


Figura 5. 6 - Adicionar Filme

## 3 – Cadastro de Elenco

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

Primeiro relembre o protótipo da tela de cadastro de elenco:

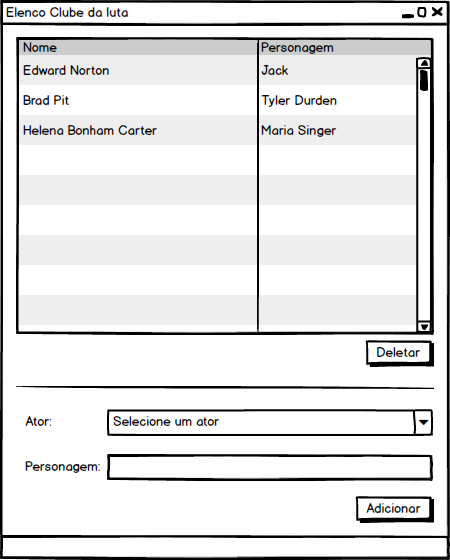


Figura 5. 7 - Protótipo Elenco

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

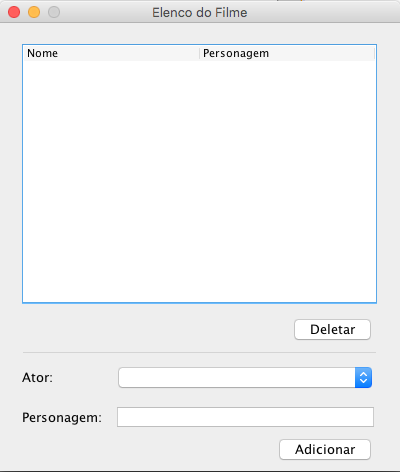


Figura 5. 8 - Mockup tela Elenco

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

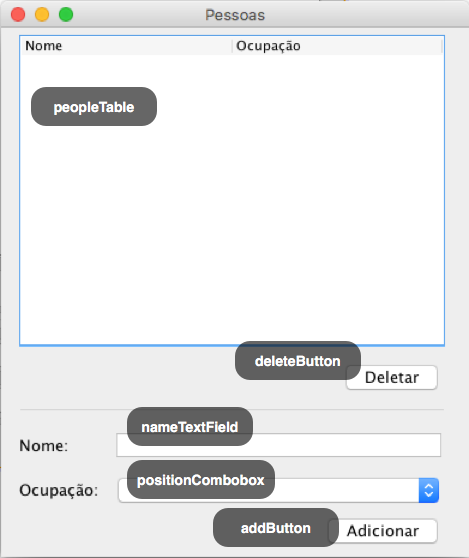


Figura 5. 9 - Variáveis da tela de Elenco

### COTEÚDOS COMPLEMENTARES

Você, Educador, deve agregar novos conhecimentos sobre os assuntos desta aula. Conheça algumas sugestões de conteúdo:

Leia a 8a edição (2010) da obra **Java – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall. Este livro apresenta: aulas desde básicas até avançadas sobre programação Java. Esta é a unidade final do curso. Supomos que o aluno não terá mais dificuldades com a linguagem Java, então este livro será um bom cookbook para você, educador, e o aluno consultar soluções mais sofisticadas para os problemas desta aula.

Leia a 1a edição (2006) da obra **Swing: A Beginners Guide**, do autor Gilleanes T. A. Guedes, publicado pela editora McGraw-Hill. Este livro contém: material complementar para desenvolvimento em Swing. Este livro poderá servir como referência na execução dos TDPs.

### REFERÊNCIAS

* GUEDES, GILLEANES T. A. **Swing: A Beginners Guide.** McGraw-Hill. 1a Edição, 2006.

### Conheça todas as referências utilizadas pelo autor para elaboração dessa aula no Livro do Aluno:

* NETBEANS. **Swing**. Disponível em: <https://netbeans.org/features/java-on-client/swing\_pt\_BR.html>. Acessado em 13 dec. 2016.
* DEVMEDIA. **Tipos Enum no Java.** Disponível em: <http://www.devmedia.com.br/tipos-enum-no-java/25729>. Acesso em: 15 dec. 2016.

# 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.

### 3.5 – Exercícios

### **As perguntas têm por objetivo fixar os principais conceitos abordados durante a aula. Para isso, viabilize o tempo necessário para o aluno responder às perguntas, acompanhando-os nas dúvidas. Para correção das perguntas, considere as respostas a seguir:**

**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.

**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. Ele é utilizado para monitorar alguma atividade e avisar quando um evento esperado acontece.

**6. O que é Look&Feel?**

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

### 3.6 – TDP

Ainda considerando a teoria relacionada à prática, é necessário que você seja um orientador do processo de desenvolvimento desta atividade, garantindo que o aluno tenha o conhecimento necessário para tornar-se um profissional qualificado para o mercado de trabalho.

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

#### 3.6.1 - Controllers

**Após ter feito 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 por meio dos Controllers. Isso é muito similar ao aprendido em aula.**

#### Tarefa

**Você chegou à última etapa!**

**Como se sente? Espera-se que realizado, pois veja aonde chegou!**

**Para finalizar o projeto, nesta etapa final você deverá finalizar as funcionalidades de cadastrar, editar, visualizar e deletar. Para isso utilize os Controllers para realizar a interação entre Views e Models.**

#### FALTA INSTRUÇÃO AO INSTRUTOR

#### Tarefa

**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!!**

Educador, esta é a etapa final, este trabalho prático é um pouco maior do que os exemplos dados em sala de aula, então é importante que se faça o máximo de proveito de Orientação a Objetos aprendido em aula. Será fácil se perder em meio a tantas telas, mas com bom uso de Orientação a Objetos, isto não será problema.

Até então interpretamos o problema, modelamos um diagrama de classe e implementamos nossas classes Model. A camada Model não está totalmente pronta, na verdade só estará pronta no final do projeto, pois a adaptação é livre podendo realizá-la da melhor maneira conforme o programa aumenta, veremos isto no decorrer do projeto. Implemente de uma ordem diferente dos exemplos dados em aula, utilizando 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 que 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. Agora vamos criar nossa tela de cadastro de filmes.

## 1 - Cadastro de filmes

Esta é uma das telas principais do nosso app e com ela existe muitas dependências. Como já foi criado todos os Models, criare as Views e altere os models conforme for necessário.

### 1.1 - Criando a View

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

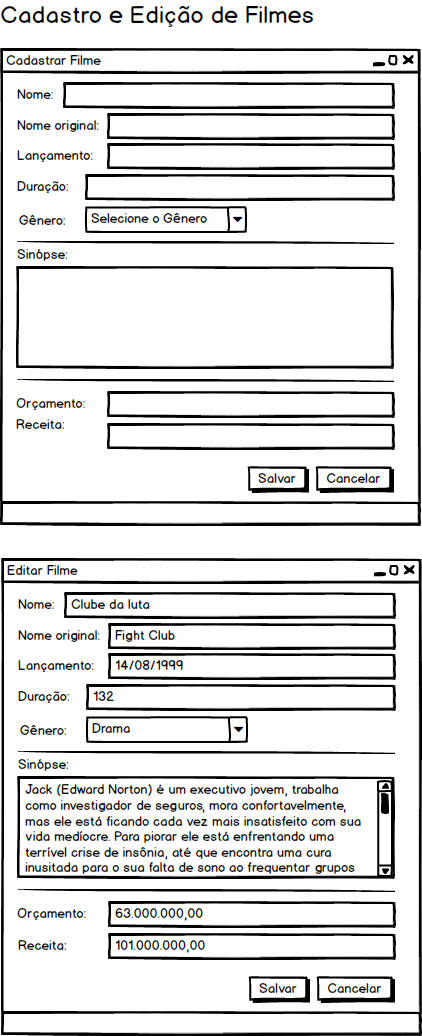


Figura 5. – Cadastrar e Editar filmes

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:

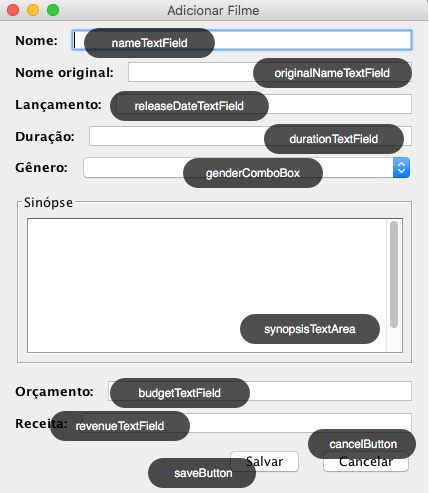


Figura 5. - Adicionar Filme

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:

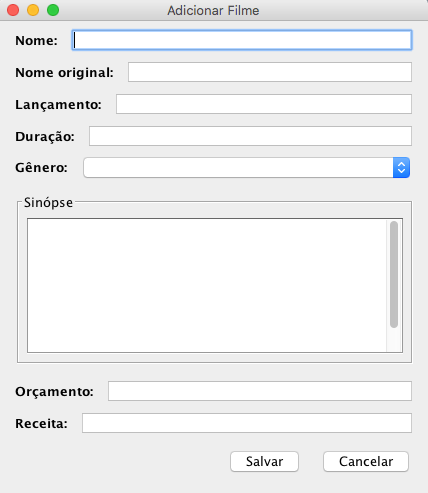


Figura 5. - Adicionar Filme

### 1.2 - Criando os Controllers

Seguindo o exemplo dado em aula é necessário 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 models foi criado 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 é necessário implementar o método AbstractTableModel para cada tabela que se deseja utilizar. Já que foi padronizado que todas as tabelas terão uma ou duas colunas, basta utilizar a Orientação a Objetos 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. Você se recorda que foi implementado os 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. Aos poucos você está 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 como filmes, atores, pessoas, países, gêneros, etc, para que se possa economizar código e fazer o máximo de reuso. Você se recorda que foi implementado a interface TableModel nos Models? Pois é, ela foi criada pensando em exibir estes models na tabela. 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**;**

O Controller não poderá conter atributos e métodos estáticos, pois ele será utilizado para mais de uma tabela, isto é, precisa-se de uma instância para cada tabela. Ao mesmo tempo que é necessário instanciar cada Controller, também é preciso manter os dados na lista, impedindo que estes sejam perdidos quando seu uso for suspenso. Nesse Controller teremos um atributo TableModelImplementation, para implicar que todas as tabelas que utilizarem a classe GenericTableController ou subclasses que estenderem, 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 um código que omitimos. Utilize 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, você terá 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 conseguir utilizar um ArrayList, de fácil manipulação, 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**;**

**}**

Ogerenciador de lista está pronto. Agora crie o gerenciador de Gêneros.

**6 -** Crie um arquivo chamado de GendersController que estenda a classe GenericListController. Nele você inicializará a 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;**

**}**

**}**

Deixe 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 isso 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 atribua o GendersController ao genderComboBox. Adicione o seguinte código no campo que se revelou:

**new** GendersController()

As configurações deverão estar como a figura abaixo:

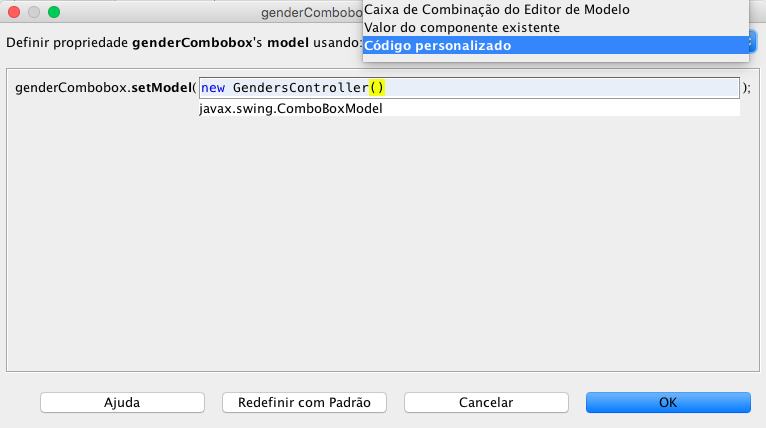


Figura 5. - Model do combobox de gênero

**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 o combobox foi preenchido.

##### 1.2.3.2. Configurando o JTextArea

O JTextArea deverá ter as seguintes configurações:

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

Para isso siga os seguintes passos:

**1 -** Selecione o synopsisTextArea e no painel de propriedades deixe marcado a propriedade lineWrap. Como na imagem a seguir:



Figura 5. - Propriedade lineWrap selecionada

A medida que você for digitando a barra vertical aparecerá.

##### 1.2.3.3. Fechando a tela

Nesse passo aproveite os exemplos dados em aula. Ainda no arquivo AddOrEditMovieView no modo **Projeto**, dê duplo clique no botão “Cancelar”. Você entrará 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

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

Na View você terá campos numéricos (duration), campo de data (releaseDate) e campos monetários (budget e revenue). Deve-se restringir o tipo da entrada nestes campos para que o usuário não coloque algo diferente do que estamos esperando, como tempo negativo em duration, palavras em releaseDate e nos campos monetários, etc. Para isso crie máscaras aos campos de texto. Comece com o durationTextField que é o mais simples.

Para o durationTextField deve-se 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:

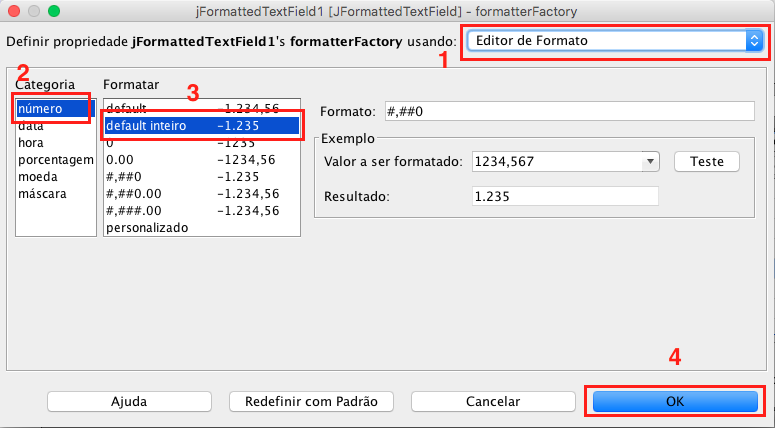


Figura 5. - Formatação de duração

**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 formate 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. Siga os passos de acordo com a imagem:

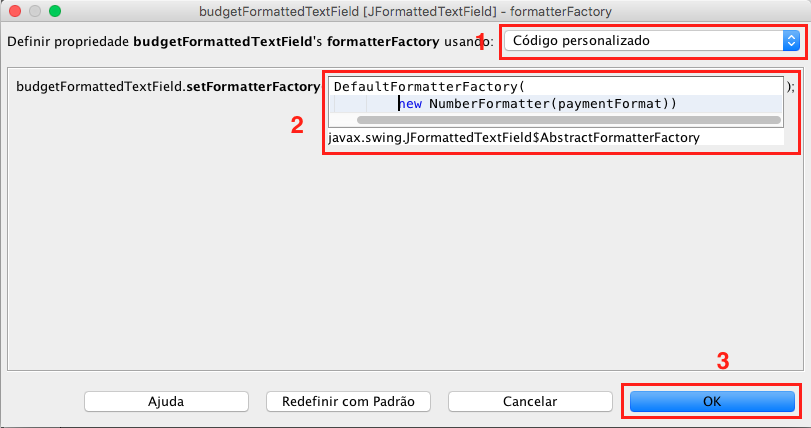


Figura 5. - Formatação monetária

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

**7 -** Agora formate 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. Siga os passos de acordo com a imagem:

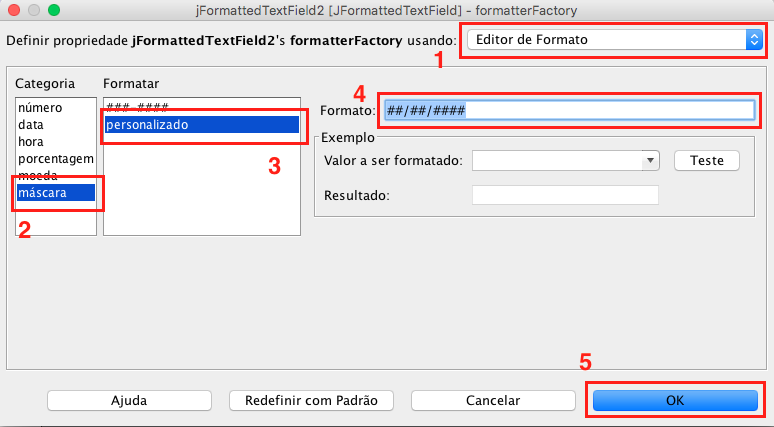


Figura 5. - Formatação de data

**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). Campos Formatados!

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

Para salvar um filme, você seguirá os mesmos passos dados em aula, adicionando a 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**();**

Este processo irá aderir 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**();**

**}**

**}**

A funcionalidade de cadastro de filmes foi finalizada, o próximo passo é 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. Veja como o método ficará:

**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**();**

**}**

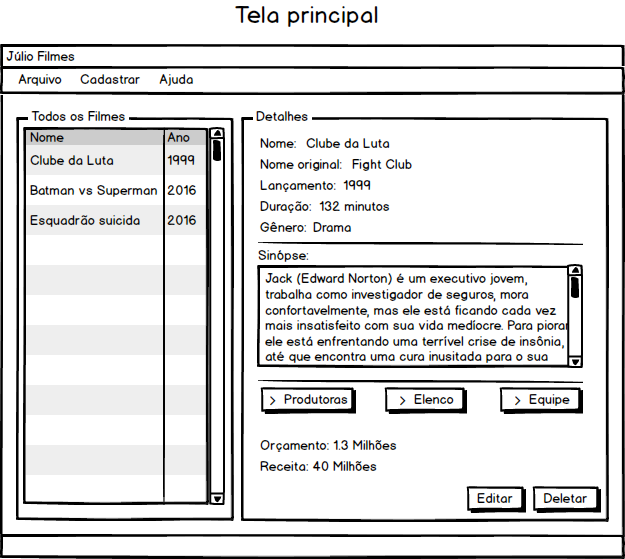
**}**

## 2 - Tela principal - DashboardView

### 2.1 - Criando a View

Antes de você criar a View, relembre o seu protótipo:

Figura 5. - Protótipo da Tela Principal



Crie 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 e o JTextArea contém o sufixo TextArea.

Esse 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:

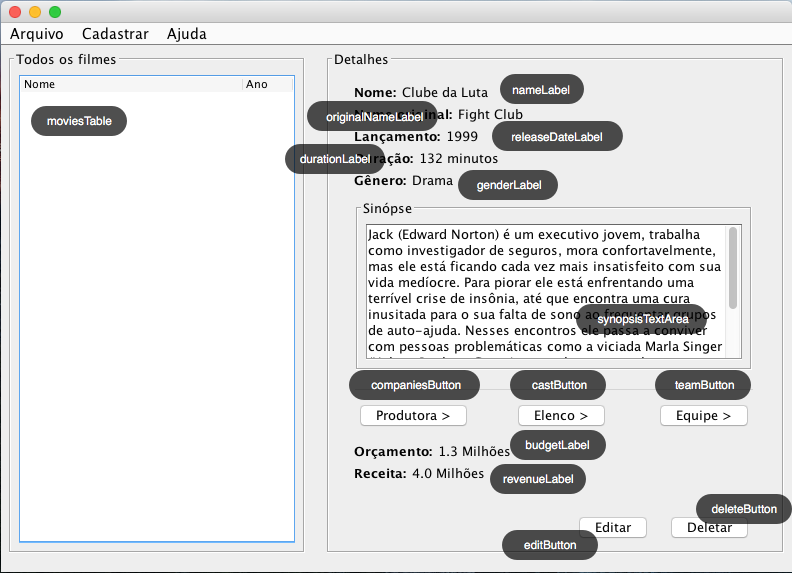
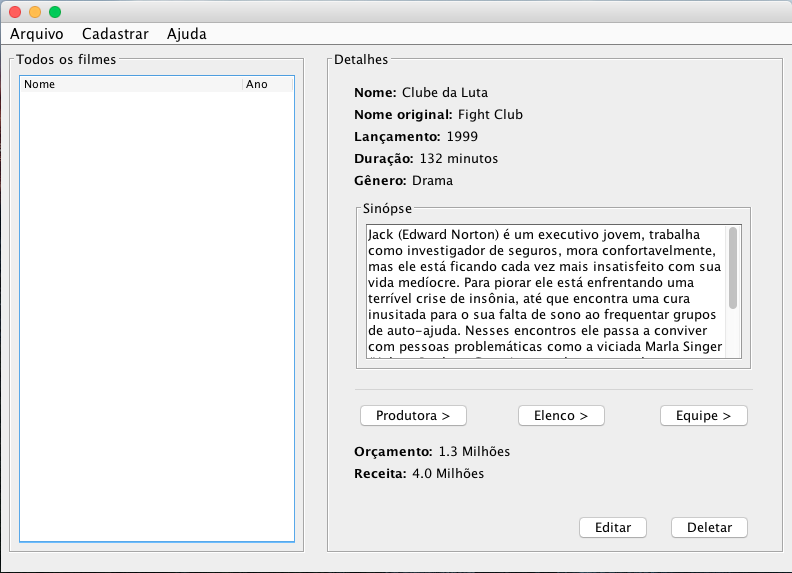


Figura 5. - Variáveis da Tela Principal

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

Nessa etapa você terá as seguintes tarefas:

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

Comece 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:

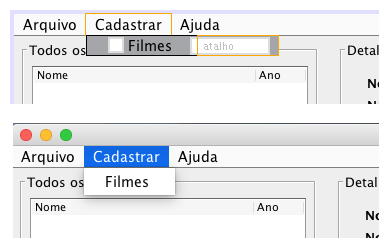


Figura 5. - JMenuItem filmes

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

Primeiro instancie 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**());**

**}**

**});**

**}**

Isso 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**(**""**);**

**}**

**}**

Esse método preencherá ou limpará 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

Para editar um filme 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 arquitetura classificamos como TeamMember as pessoas que exercem atividade profissional sobre o filme. Para adicionar um elenco e uma equipe ao filme, primeiro você deve cadastrar estas pessoas.

### 3.1 – Criando a View

Vamos relembrar do mockup desta View:

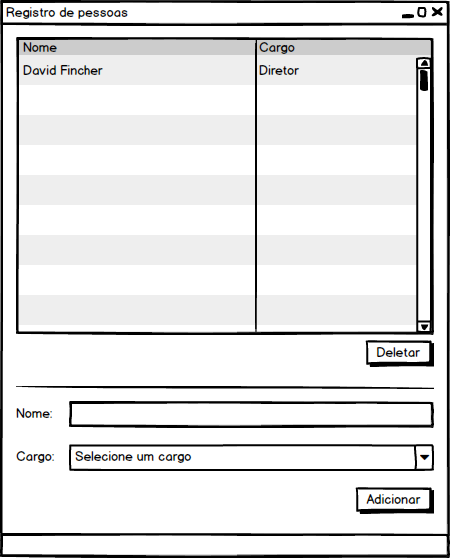


Figura 5. - Tela cadastro de pessoas

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

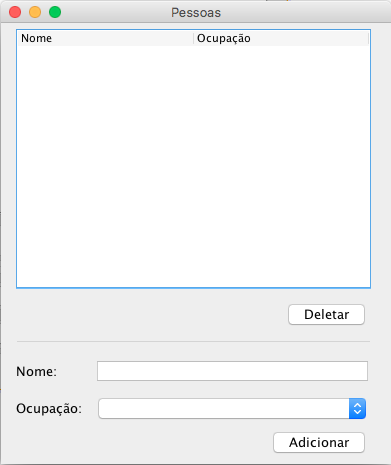


Figura 5. - Mockup tela Pessoas

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

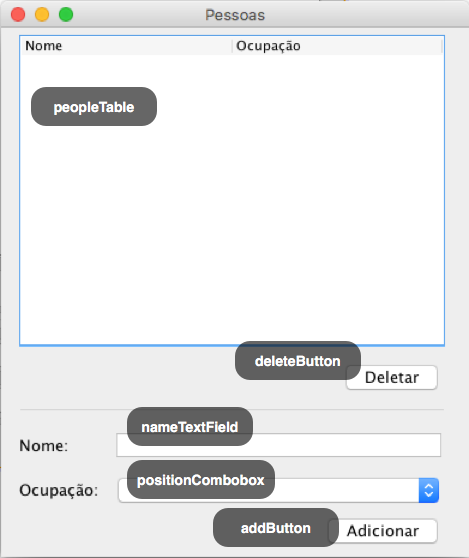


Figura 5. - Variáveis de cadastro de pessoas

Repare que temos uma seção abaixo do botão **Deletar.**  Essa seção é equivalente à tela auxiliar que se abria para cadastrar/editar no exemplo da agenda de contatos. Na tabela você fará o cadastro, deleção e exibição dos dados na nesta mesma View.

### 3.2 – Criando os controllers

Esse Controller será aos moldes do MoviesController. Executaremos passos semelhantes ao já realizado. Veja a seguir:

**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. Nesse caso não precisará 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**();**

**}**

Esse método 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**:

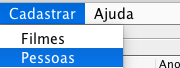


Figura 5. - Menu Pessoas

**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 o 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**());**

**}**

**}**

O Controller de pessoas está pronto. Ele 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:

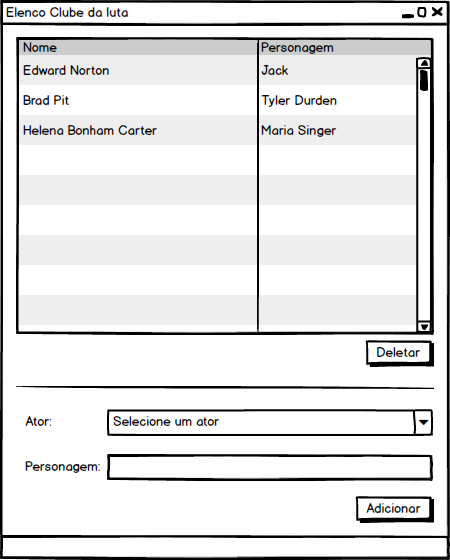


Figura 5. - Protótipo Elenco

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

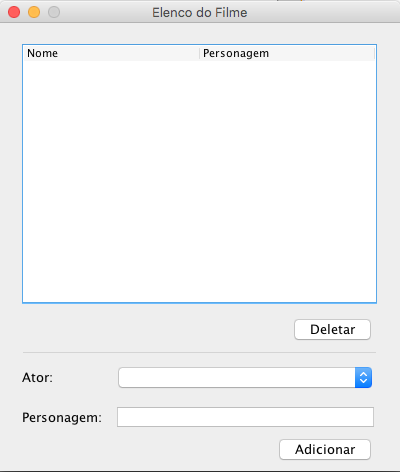


Figura 5. - Mockup tela Elenco

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

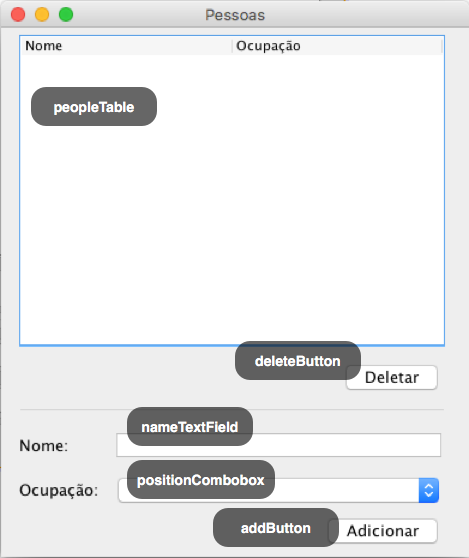


Figura 5. - Variáveis da tela de Elenco

### 4.2 – Criando os controllers

Este Controller será aos moldes do MoviesController. Siga os seguintes passos:

**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 atualizará 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 você terá que exibir a lista de atores no actorCombobox da CastView.

#### 4.2.1 – Controller de atores

Crie 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 você precisará é definir que os **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, crie 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 método será utilizado na tela CastView para verificar se já existe ator cadastrado, por isso 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 você pode exibir os atores no actorCombobox.

#### 4.2.1 – Interligando as Views e Controllers

Primeiro invocaque 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:

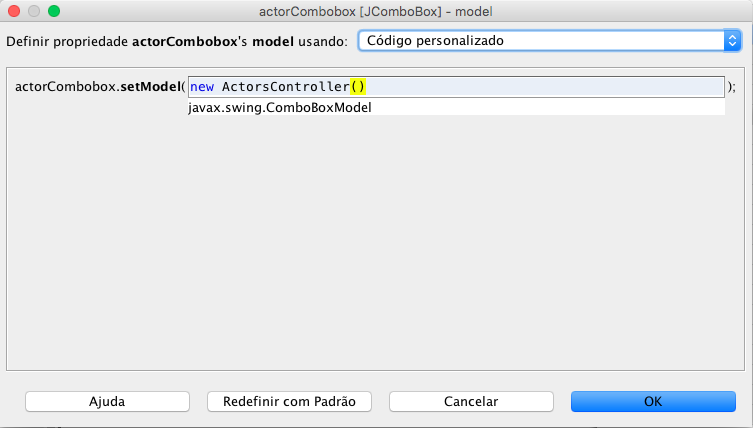


Figura 5. - Configurando o model de actorCombobox

Não se esqueça de importar tudo que a IDE pede.

**4 –** Crie um atributo para CastController, inicialize-o e faça o mesmo com 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.**

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 conseguirá depurar e resolver os erros de programação. Isso 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 ajudá-lo. Esse final de projeto é uma boa oportunidade para treinar! 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.

### CONTEÚDOS COMPLEMENTARES

Você, Educador, deve agregar novos conhecimentos sobre os assuntos desta aula. Conheça algumas sugestões de conteúdo:

* Leia a 8a edição (2010) da obra **Java – Como Programar,** do autor Deitel, publicada pela editora Prentice Hall. Este livro apresenta: aulas desde básicas até avançadas sobre programação Java. Esta é a unidade final do curso. Supomos que o aluno não terá mais dificuldades com a linguagem Java, então este livro será um bom cookbook para você, educador, e o aluno consultar soluções mais sofisticadas para os problemas desta aula.
* Leia a 1a edição (2006) da obra **Swing: A Beginners Guide**, do autor Gilleanes T. A. Guedes, publicado pela editora McGraw-Hill. Este livro contém: material complementar para desenvolvimento em Swing. Este livro poderá servir como referência na execução dos TDPs.

### REFERÊNCIAS

* GUEDES, GILLEANES T. A. **Swing: A Beginners Guide.** McGraw-Hill. 1a Edição, 2006.

### Conheça todas as referências utilizadas pelo autor para elaboração dessa aula no Livro do Aluno:

* NETBEANS. **Swing**. Disponível em: <https://netbeans.org/features/java-on-client/swing\_pt\_BR.html>. Acessado em 13 dec. 2016.