LPIII - JavaServer Faces

1 - Construindo Aplicações Web em alto nível com JavaServer Faces

Este é o primeiro de quatro tutoriais, a partir dos quais você aprenderá a utilizar JavaServer Faces para construir aplicações web. Com o uso dessa tecnologia, você poderá observar um grande desacoplamento entre o layout das páginas dinâmicas no seu projeto web e o código Java utilizado para processar e armazenar a informação fornecida pelos usuários através dos formulários exibidos nas páginas dinâmicas.

Componentes gráficos, são disponibilizados em várias bibliotecas, para a construção das páginas dinâmicas a partir do uso de tags em arquivos no formato xhtml. A implementação com esses componentes é realizada em um nível muito mais alto, do que as implementações de componentes do pacote javax.swing utilizados para a criação de aplicações desktop. As tags que representam cada componente gráfico estão associadas a parâmetros que suportam customizações da utilização do componente. Adicionalmente, você pode definir estilos de formatação a partir de arquivos css (cascading style sheets), para padronizar a apresentação de suas páginas dinâmicas. Você poderá especificar elementos que afetam o formato dos documentos (ex: fontes, espaçamento, margens) em arquivos css a parte da estrutura do documento (ex: headers, corpo, formulários, links) implementado em arquivos xhtml, nos quais as classes de estilo dos arquivos css são simplesmente referenciadas.

O código Java é armazenado em arquivos Java, denominados Beans, nos quais você poderá implementar variáveis para armazenar temporariamente os dados informados pelos usuários da aplicação web, bem como, prover o métodos para processar esses dados ou armazená-los em um banco de dados.

O controle das threads necessárias para tratar cada usuário no servidor de aplicações, e a comunicação entre o navegador e servidor de aplicações, são transparentes para o desenvolvedor da aplicação web.

Neste tutorial você aprenderá, a partir da ilustração de dois exemplos, os conceitos básicos da utilização de JavaServer Faces. Nos próximos três tutoriais, você aprenderá a utilizar o modelo objeto-relacional, que suporta a denotação dos objetos da sua aplicação, a partir da qual o banco de dados é gerado automaticamente, juntamente com os métodos básicos para inserção, remoção, alteração e consulta de objetos em um banco de dados relacional. Com a utilização do modelo objeto-relacional, a persistência de objetos nos banco de dados relacional é transparente para o desenvolvedor de aplicações web, que passa a implementar um código capaz de salvar e recuperar objetos, sem a necessidade de implementar comandos SQL para acessar registros em tabelas em bancos de dados relacionais.

Na próxima seção você aprenderá o passo a passo da instalação e configuração: (a) do IDE (Integrated Development Environment) NetBeans para desenvolver e gerar o executável da sua aplicação web; (b) do servidor de aplicações WidFly para suportar a execução da sua aplicação web em um navegador; e (c) do framework EclipseLink para suportar o modelo objeto-relacional.

As duas aplicações ilustradas neste tutorial, foram desenvolvidas especificamente para este tutorial, com base em aplicações ilustradas nos capítulos 29 e 30 do livro:

- Paul Deitel; Harvey Deitel. Java How to Program. Ninth edition; 2012.

2 - Instalando e Configurando o Ambiente de Desenvolvimento de Aplicações Web

Em geral, novas versões do IDE e do servidor de aplicações invariavelmente resultam em alterações de configuração que, no caso de aplicações web que dependem da instalação e configuração de vários softwares distintos que precisam trabalhar em conjunto, podem tomar um tempo substancial para corrigir se você erra alguma informação ou omite algum passo.

Para evitar transtornos, serão utilizadas versões, de cada um desses softwares, que comprovadamente funcionam com as configurações que você irá aprender nesse passo a passo. Seja muito cuidadoso ao seguir este passo a passo. Um pequeno erro em qualquer um desses passos poderá resultar em dificuldades significativas para localizar o erro e corrigí-lo.

2.1 - Instalando e Configurando Ambiente para Aplicação Web sem utilizar Banco de Dados

Na seção 3 deste tutorial será ilustrada uma aplicação web, denominada ClubeCinema, que não utiliza banco de dados. Siga os passos de 1 a 6 para instalar e configurar o ambiente para executá-la.

Passo 1 : Instalação do SGBD (Sistema de Gerenciamento de Banco de Dados) MySQL

- dados de instalação
  - nome de usuário (username) : root
  - senha (password) : admin
- você pode utilizar o par nome de usuário e senha de sua escolha
  - desde que os informe corretamente para
    - criar e visualizar o banco de dados a partir do IDE NetBeans
    - configurar fontes de dados no console do servidor de aplicações WildFly

Passo 2 : instalar o JDK e o IDE NetBeans

- instale o JDK jdk-8u102-windows-x64.exe
- instale o NetBeans netbeans-8.2-javaee-windows.exe
  - não selecione nenhum dos servidores de aplicação oferecidos
    - posteriormente será configurado para utilizar o servidor aplicações WildFly

Passo 3 : disponibilizar o servidor de aplicações WildFly

- dowload da versão 10 do WildFly no site http://wildfly.org/
  - descompacte e armazene em um diretório de sua escolha (ex: WildFly10) no drive C
    - com exceção do diretório C:\Arquivos de Programas
- para ativar o WildFly utilizando o IP da máquina na qual está executando
  - defina o script WildFly10\iniciar-servidor.bat
    cd bin
    standalone.bat --server-config=standalone-full.xml -b 0.0.0.0

Passo 4 : altere a porta de execução do WildFly de 9990 para 9991
- no arquivo WildFly10\standalone\configuration\standalone-full.xml
  - <socket-binding name="management-http" interface="management" port="${jboss.management.http.port:9991}"/>
- no arquivo WildFly10\domain\configuration\host.xml
  - <socket interface="management" port="${jboss.management.http.port:9991}"/>

Passo 5 : configure o WildFly como o servidor de aplicações do NetBeans
- NetBeans : Ferramentas → Servidores → Adicionar Servidor
  - selecione o WildFly
  - localize o diretório do WildFly
  - altere a porta para 9991

Passo 6 : distribuia e execute a aplicação web
- distribua uma versão executável da sua aplicação web
  - ative o servidor de aplicações WildFly (script iniciar-servidor.bat)
    - aguarde a inicialização do WildFly
      - até visualizar uma mensagem finalizando com o texto
        - services are lazy, passive or on-demand
  - gere a versão executável da aplicação web
    - NetBeans : Executar → Limpar e Construir Projeto
  - distribua o executável de seu projeto no servidor de aplicações
    - copiando o arquivo com extensão war (Web application Archive)
      - gerado no subdiretório dist (distribution) do seu projeto
        - para o diretório WildFly10\standalone\deployments
    - aguarde o status : ClubeCinema.war.deployed
- abra o navegador e execute o seu projeto
  - na porta 8080 do processador local
    - localhost:/8080/ClubeCinema
- para qualquer alteração no seu projeto
  - gere novamente o arquivo war
  - e repita o processo de distribuição e execução

2.2 - Completando Ambiente para Aplicação Web com Banco de Dados via SQL

Na seção 4 deste tutorial, será ilustrada uma aplicação web, denominada ClubeCinemaDB, que utiliza banco de dados com acesso a partir de comandos SQL. Para executá-la você precisará complementar o Ambiente, seguindo os passos 6 e 7.

Passo 6 : disponibilize o conector MySQL
- download do conector MySQL : mysql-connector-java-5.1.39-bin.jar
- distribua no diretório C:\WildFly 10\standalone\deployments

Passo 7 : criar fonte de dados para acesso ao banco de dados

- crie o banco de dados filmes
  - na aba Serviços do NetBeans
    - conecte o Servidor MySQL e crie o banco de dados filmes
    - execute script (seção 4) para criar a tabela filmes
- ative o WildFly e execute seu console no navegador : localhost:/9991
  - na primeira vez, será necessário cadastrar um nome de usuário e uma senha
  - para criar um novo usuário, execute: C:\WildFly 10\bin\add-user.bat
    - What type of user do you wish to add?
      a) Management User (mgmt-users.properties)
      b) Application User (application-users.properties)
      (a): a
    - Is this new user going to be used for one AS process to connect to another AS process?
      e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
      yes/no? yes
- crie e configure a fonte de dados para a aplicação ClubeCinemaDB no console do WildFly
  - Configuration → Subsystems → Datasources → Non-XA → Add → MySQLDatasource
    - Datasource Attributes
      - Name : ClubeCinemaDB**DS**
      - JNDI Name : java:/ClubeCinemaDB**DS**
    - JDBC Driver
      - DetectedDrive
        - mysql-connector-java-5.1.39-bin.jar_com.mysql.jdbc.Driver_5_1
    - ConnectionSettings
      - ConnectionURL : jdbc:mysql://localhost:3306/**filmes**
      - Username : root
      - Password : admin
    - TestConnection
      - botão TestConnection
        - para checar se a fonte de dados se conecta com o MySQL
    - RunTime
      - StandaloneServer
        - para reiniciar o servidor de aplicações e concluir a criação da fonte de dados
- distribua a aplicação no WildFly e execute no navegador

2.3 - Completando Ambiente para Aplicação Web com Modelo Objeto-Relacional

Nos próximos três tutoriais, para utilizar o modelo objeto-relacional, você precisará adicionar o suporte ao framework EclipseLink, dado que o WildFly utiliza o framework Hibernate, como implementação padrão do Java Persistence API. Para executar a aplicação que será desenvolvida incrementalmente no tutoriais de 2 a 4, siga o passo 8.

Passo 8 : adicionando o suporte do framework EclipseLink ao WildFly

- dowload da versão 2.6 do EclipseLink no site http://eclipse.org/
- descompacte e armazene em um diretório de sua escolha (ex: EclipseLink) no drive C
  - com exceção do diretório C:\Arquivos de Programas
- copie EclipseLink\jlib\eclipselink.jar
  - para a pasta WildFly10\modules\system\layers\base\org\eclipse\persistence\main\
- crie uma biblioteca EclipseLink no NetBeans
  - NetBeans : Ferramentas → Bibliotecas → Nova Biblioteca
    - Nome de Biblioteca : EclipseLink
    - adicionar JAR/Pasta
      - EclipseLink/jlib/eclipselink.jar
      - EclipseLink/jlib/jpa/javax.persistence_2.1.1.v201509150925.jar
      - EclipseLink/jlib/jpa/org.eclipse.persistence.jpa.modelgen_2.6.4.v20160829-44060b6
- adicione a biblioteca EclipseLink ao projeto
  - NetBeans : projeto ClubleCinemaDB
    - Bibliotecas → Adicionar Biblioteca : EclipseLink
- adicionar referências ao EclipseLink no arquivo WildFly10\modules\system\layers\base\org\eclipse\persistence\main\module.xml
  - adicionar na tag <resources> : mais uma tag <resource-root>

```
<resource-root path="eclipselink.jar">
    <filter>
        <exclude path="javax/**" />
    </filter>
</resource-root>
```

  - adicionar na tag <dependencies > : mais uma tag <module>

```
<module name="javax.ws.rs.api"/>
```

- no console do WildFly configurar a implementação do EclipseLink
  - Configuration → System Properties → Add
    - Name : eclipselink.archive.factory
    - Value : org.jipijapa.eclipselink.JBossArchiveFactoryImpl
- reiniciar o WildFly

Para executar a aplicação, desenvolvida de forma incremental nos tutorias de 2 a 4, distribua a aplicação no WildFly e execute no navegador.

3 - Implementando uma Aplicação Web sem utilizar Banco de Dados

A primeira aplicação web, ilustrada neste tutorial, suporta o cadastro de filmes e a pesquisa de filmes cadastrados por um diretor selecionado. Essa aplicação web será intitulada ClubeCinema, em referência a amigos interessados em compartilhar informações sobre filmes preferidos.

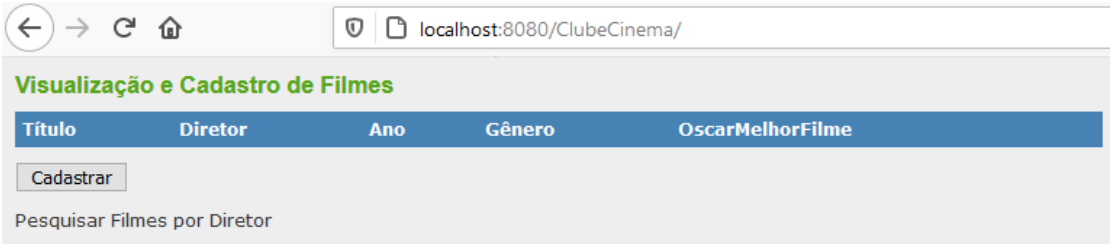
Para criar uma aplicação web no NetBeans:

- NetBeans : Arquivo → Novo Projeto
  - Escolher Projeto
    - Categorias : selecione Java Web
    - Projetos : selecione Aplicação Web
  - Nome e Localização
    - Nome do Projeto : ClubeCinema
    - Projetos : selecione o diretório no qual vai armazenar o seu projeto.
  - Servidor e Definições
    - Servidor : selecione WildFly Application Server
    - Versão do Java EE : selecione Java EE 7 Web
  - Frameworks
    - assinale JavaServer Faces

Será criada a primeira página dinâmica do projeto ClubeCinema, com o nome padrão index.

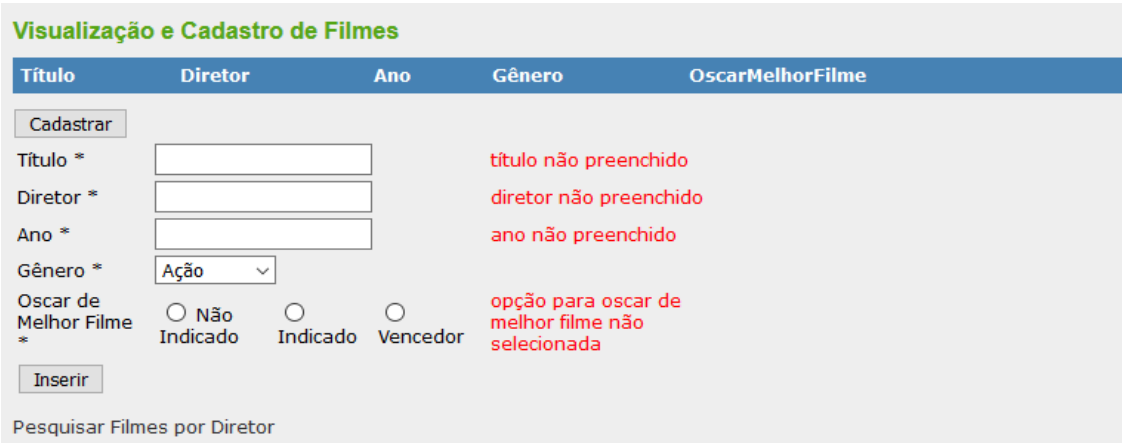
A seguir, serão ilustrados: (a) as saídas geradas na execução da aplicação web; (b) as páginas dinâmicas; (c) os arquivos de estilo; (d) a classe entidade; e (e) a classe Bean.

Ilustração inicial da página de Visualização e Cadastro de Filmes.



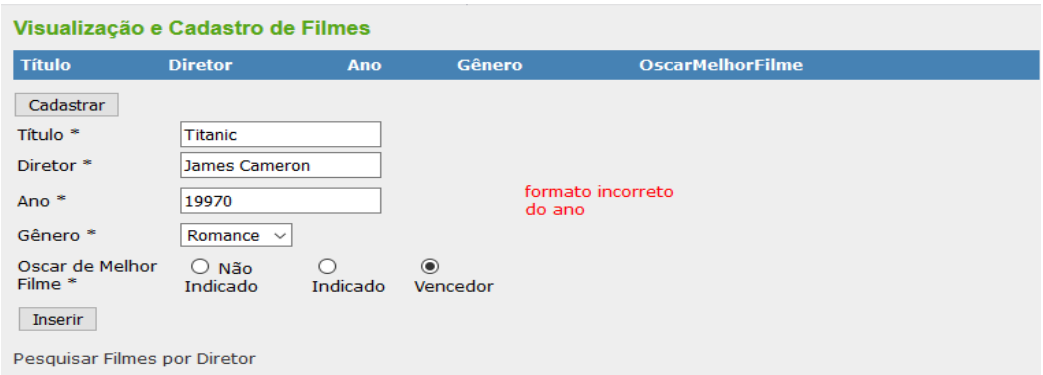
Cadastrando o primeiro filme na página de Visualização e Cadastro de Filmes

- sem preencher as informações obrigatórias



Cadastrando o primeiro filme página de Visualização e Cadastro de Filmes

- sem preencher corretamente a informação de ano



Cadastrando o quarto filme na página de Visualização e Cadastro de Filmes.

Na página de Pesquisa de Filmes por Diretor

- selecionando um diretor e pesquisando seus filmes cadastrados

Página dinâmica de Visualização e Cadastro de Filmes

- importação de três bibliotecas de componentes
- cabeçalho (head) com título
- importação de arquivos de estilo (outputStylesheet)
  - definidos em Páginas Web/resources/css

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <f:facet name="first">
            <meta content='text/html; charset=UTF-8' http-equiv="Content-Type"/>
            <title>Cadastro de Filmes</title>
        </f:facet>
        <h:outputStylesheet name="style.css" library="css"/>
        <h:outputStylesheet name="default.css" library="css"/>
        <h:outputStylesheet name="overrides.css" library="css"/>
    </h:head>
```

Página dinâmica de Visualização e Cadastro de Filmes

- corpo (body) com colunas iniciais da tabela (dataTable) de visualização dos filmes
  - chamada de método de leitura somente com o nome da referência
  - tabela utiliza classes de estilos definidas em arquivo css

```
    </h:head>
    <h:body>
        <h3>Cadastro de Filmes</h3>
        <h:dataTable value="#{clubeCinemaBean.filmes}" var="filme"
                     rowClasses="oddRows,evenRows" headerClass="header"
                     styleClass="table" cellpadding="5" cellspacing="0">
            <h:column>
                <f:facet name="header">Título</f:facet>
                    #{filme.título}
            </h:column>
            <h:column>
                <f:facet name="header">Diretor</f:facet>
                    #{filme.diretor}
            </h:column>
```

Página dinâmica de Visualização e Cadastro de Filmes

- botão de comando (commandButton) Cadastrar
  - chamada do nome completo do método acompanhado de parênteses
- painel de grade (panelGrid) com dois primeiros campos do cadastro
  - grade com 3 colunas
    - rótulo (outputText)
    - leitura de informação (inputText)
      - chamada de métodos de leitura somente com o nome da referência e do atributo
    - mensagens de informação não preenchida (message)
      - texto da mensagem (requiredMessage)

```
</h:dataTable>
<h:form>
    <h:commandButton value="Cadastrar" action="#{clubeCinemaBean.reset()}"/>
    <h:panelGrid columns="3" style="height:100px;width:500px;"
                 rendered="#{clubeCinemaBean.cadastrar}">
        <h:outputText value="Título *"/>
        <h:inputText id="tituloInputText" required="true"
                     value="#{clubeCinemaBean.filme.titulo}"
                     requiredMessage="título não preenchido"/>
        <h:message for="tituloInputText" styleClass="mensagem_erro"/>
        <h:outputText value="Diretor *"/>
        <h:inputText id="diretorInputText" required="true"
                       value="#{clubeCinemaBean.filme.diretor}"
                       requiredMessage="diretor não preenchido"/>
        <h:message for="diretorInputText" styleClass="mensagem_erro"/>
```

Página dinâmica de Visualização e Cadastro de Filmes

- informação do cadastro com formato determinado por expressão regular (validateRegex)
  - mensagem de informação com formato incorreto (message)
    - texto da mensagem (validatorMessage)

```
<h:outputText value="Ano *"/>
<h:inputText id="anoInputText" required="true"
             value="#{clubeCinemaBean.filme.ano}"
             requiredMessage="ano não preenchido"
             validatorMessage="formato incorreto do ano">
    <f:validateRegex pattern="\d{4}"/>
</h:inputText>
<h:message for="anoInputText" styleClass="mensagem_erro"/>
```

Página dinâmica de Visualização e Cadastro de Filmes

- informação do cadastro com primeiras opções (selectItem) de componente do tipo ComboBox (selectOneMenu)

```
<h:outputText value="Gênero *"/>
<h:selectOneMenu id="gêneroSelectOneMenu" required="true"
                 value="#{clubeCinemaBean.filme.gênero}"
                 requiredMessage="gênero não selecionado">
    <f:selectItem itemValue="ação" itemLabel="Ação" />
    <f:selectItem itemValue="aventura" itemLabel="Aventura" />
    <f:selectItem itemValue="comédia" itemLabel="Comédia" />
    <f:selectItem itemValue="drama" itemLabel="Drama" />
```

Página dinâmica de Visualização e Cadastro de Filmes

- informação do cadastro com componente do tipo botões de radio (selectOneRadio)

```
    <h:selectOneRadio id="oscar_melhor_filmeSelectOneRadio" required="true"
                      value="#{clubeCinemaBean.filme.oscarMelhorFilme}"
                      requiredMessage="opção para oscar de melhor filme não selecionada">
        <f:selectItem itemValue="não indicado" itemLabel="Não Indicado"/>
        <f:selectItem itemValue="indicado" itemLabel="Indicado"/>
        <f:selectItem itemValue="vencedor" itemLabel="Vencedor"/>
    </h:selectOneRadio>
    <h:message for="oscar_melhor_filmeSelectOneRadio" styleClass="mensagem_erro"/>
</h:panelGrid>
```

Página dinâmica de Visualização e Cadastro de Filmes

- botão de comando (commandButton) para inserir filme
- link (outputLink) para renderização da página de Pesquisa de Filmes por Diretor

```
        </h:panelGrid>
        <h:panelGrid rendered="#{clubeCinemaBean.cadastrar}">
            <h:commandButton value="Inserir" action="#{clubeCinemaBean.cadastrarFilme()}"/>
        </h:panelGrid>
        <p><h:outputLink value="pesquisa.xhtml">Pesquisar Filmes por Diretor</h:outputLink></p>
    </h:form>
</h:body>
</html>
```

Página dinâmica de Pesquisa de Filmes por Diretor

- componente do tipo ComboBox (selectOneMenu) para seleção do diretor
- botão de comando (commandButton) Pesquisar

```
<h:body>
    <h3>Pesquise os Filmes de um Diretor</h3>
    <h:form>
        <h:panelGrid columns="3" style="height:50px;width:300px;">
            <h:outputText value="Selecione um diretor *"/>
            <h:selectOneMenu id="diretorSelectOneMenu" required="true"
                             value="#{clubeCinemaBean.diretorSelecionado}"
                             requiredMessage="diretor não selecionado" style="width:100px">
                <f:selectItems value="#{clubeCinemaBean.getDiretoresItens()}"/>
            </h:selectOneMenu>
            <h:message for="diretorSelectOneMenu" styleClass="mensagem_erro"/>
        </h:panelGrid>
        <h:commandButton value="Pesquisar" action="#{clubeCinemaBean.pesquisarFilme()}"/>
    </h:form>
```

Página dinâmica de Pesquisa de Filmes por Diretor

- componente de saída formatado (outputFormat) para informar o diretor selecionado
- lista não ordenada (unorded list : ul) com repetição (repeat) de itens de lista (list itens : li) em formato de bullets
- link (outputLink) para renderização da página de Visualização e Cadastro de Filmes

```
        </h:form>
        <h:panelGrid rendered="#{clubeCinemaBean.pesquisar}">
            <h:form>
                <h:outputFormat value="Filmes dirigidos por {0}" styleClass="mensagem">
                    <f:param value="#{clubeCinemaBean.diretorSelecionado}"/>
                </h:outputFormat>
            </h:form>
            <ul>
                <ui:repeat value="#{clubeCinemaBean.getInfoFilmesDiretor()}" var="info_filme">
                    <li>#{info_filme}</li>
                </ui:repeat>
            </ul>
        </h:panelGrid>
        <p><h:outputLink value="index.xhtml">Visualizar e Cadastrar Filmes</h:outputLink></p>
    </h:body>
</html>
```

Arquivo de estilo : style.css

```
.mensagem_erro { color:red; }
.mensagem { color:blue; }
.header { background-color: steelblue; color: white; text-align: left; }
.oddRows { padding-left: 5px; background-color: lightcyan; }
.evenRows { padding-left: 5px; background-color: lightskyblue; }
.table { width: 750px; }
```

Arquivo de estilo : default.css

```
body { background-color: #eeeeee; font-size: 12px; color: #000000; margin: 10px;
    font-family: Verdana, "Verdana CE",  Arial, "Arial CE", "Lucida Grande CE",
                 lucida, "Helvetica CE", sans-serif; }
h1 { font-size:  24px; font-weight: bold; margin: 0px; padding: 0px; color: #55A616;
    font-family: Arial, "Arial CE", "Lucida Grande CE", lucida, "Helvetica CE", sans-serif; }
h2 { font-family: Arial, "Arial CE", "Lucida Grande CE", lucida, "Helvetica CE", sans-serif;
    font-size:  20px; font-weight: bold; margin: 0 0 10px 0; padding: 0px; color: #55A616; }
h3 { font-family: Arial, "Arial CE", "Lucida Grande CE", lucida, "Helvetica CE", sans-serif;
    font-size:  16px; font-weight: bold; margin: 0 0 10px 0; padding: 0px; color: #55A616; }
a:link, a:visited { color: #333; font-size: 12px; text-decoration: none; }
a:link:hover, a:visited:hover { color: #333; font-size: 12px; text-decoration : underline; }
.panelGridClass td { vertical-align: top; }
input.ipt-large { width:100%; }
.btnPadrao { margin-top: 10px; }
```

Arquivo de estilo : overrides.css

```
.ui-widget,
.ui-widget .ui-widget,
.ui-widget .ui-wizard-step-title,
.ui-widget td.label, .ui-widget td.value, .ui-widget label,
table { font-size: 12px !important; }
.ui-panel { margin-bottom: 10px; }
```

Entidade Filme da aplicação web ClubeCinema

- atributos
- construtor

```
package entidades;

public class Filme {

    private String título;
    private String diretor;
    private String ano;
    private String gênero;
    private String oscarMelhorFilme;

    public Filme() { }
```

Entidade Filme da aplicação web ClubeCinema

- método toString

```
public String toString(boolean sem_diretor) {
    if (sem_diretor) return ano + " - " + título + " - " + oscarMelhorFilme;
    else return ano + " - " + título + " - " + diretor + " - " + oscarMelhorFilme;
}
```

Bean da aplicação web ClubeCinema.

- importações
- nome do objeto criado automaticamente (@Named)
- escopo de persistência do objeto da classe Bean (@SessionScoped)
  - persistência em todas as páginas da sessão

```
package beans;

import entidades.Filme;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import javax.annotation.PostConstruct;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;
import javax.faces.model.SelectItem;

@Named(value = "clubeCinemaBean")
@SessionScoped
public class ClubeCinemaBean implements Serializable {
```

Bean da aplicação web ClubeCinema.

- atributos e referências
- inicialização (@PostConstruct)

```
private ArrayList<Filme> filmes;
private Filme filme;
private String diretorSelecionado;
private boolean cadastrar;
private boolean pesquisar;


@PostConstruct
public void init() {
    filmes = new ArrayList();
    filme = new Filme();
    cadastrar = false;
    pesquisar = false;
}
```

Bean da aplicação web ClubeCinema.

- métodos de leitura e alteração do atributo diretorSelecionado

```
public String getDiretorSelecionado() {
    return diretorSelecionado;
}

public void setDiretorSelecionado(String diretorSelecionado) {
    this.diretorSelecionado = diretorSelecionado;
}
```

Bean da aplicação web ClubeCinema.

- cadastrar um filme
- pesquisar filme
- inicializar objeto e flags (variáveis do tipo boolean) para próximo cadastro

```
public String cadastrarFilme(){
    filmes.add(filme);
    reset();
    return "index";
}

public void pesquisarFilme() {
    pesquisar = true;
    cadastrar = false;
}

public void reset(){
    filme = new Filme();
    cadastrar = true;
    pesquisar = false;
}
```

Bean da aplicação web ClubeCinema

- criar itens de lista de diretores para utilização em component do tipo ComboBox (selectOneMenu)
  - nomes dos diretores em ordem alfabética crescente

```
public ArrayList<SelectItem> getDiretoresItens() {
    Collection<String> diretores = getDiretores();
    ArrayList<SelectItem> itens = new ArrayList();
    for(String diretor : getDiretores()) {
        boolean inserido = false;
        for (int n = 0; n < itens.size(); n++) {
            if (diretor.compareTo(itens.get(n).getLabel()) > 0) continue;
            itens.add(n, new SelectItem(diretor, diretor));
            inserido = true;
            break;
        }
        if (!inserido) itens.add(new SelectItem(diretor, diretor));
    }
    return itens;
}
```

Bean da aplicação web ClubeCinema.

- obter lista de nomes de diretores dos filmes cadastrados
  - sem duplicar nomes

```
private Collection<String> getDiretores() {
    HashMap<String, String> diretores = new HashMap();
    for (Filme filme : filmes) {
        String diretor = filme.getDiretor();
        if (diretores.get(diretor) == null) diretores.put(diretor, diretor);
    }
    return diretores.values();
}
```

Bean da aplicação web ClubeCinema.

- criar lista com informações sobre filmes de um diretor selecionado previamente
  - ano do filme em ordem numérica decrescente

```
public ArrayList<String> getInfoFilmesDiretor() {
    ArrayList<String> info_filmes_diretor = new ArrayList();
    for (Filme filme : filmes) {
        String diretor = filme.getDiretor();
        if (diretor.equals(diretorSelecionado)) {
            boolean inserido = false;
            String ano_filme = filme.getAno();
            for (int n = 0; n < info_filmes_diretor.size(); n++) {
                String ano = info_filmes_diretor.get(n).split(" - ")[0];
                if (ano_filme.compareTo(ano) < 0) continue;
                info_filmes_diretor.add(n, filme.toString(true));
                inserido = true;
                break;
            }
            if (!inserido) info_filmes_diretor.add(filme.toString(true));
        }
    }
    return info_filmes_diretor;
}
```

4 - Alterando a Aplicação Web para utilizar Banco de Dados

A segunda aplicação web, ilustrada neste tutorial, utiliza banco de dados para suportar a mesmas funcionalidades da aplicação ClubeCinema. Essa aplicação web será intitulada ClubeCinemaDB (DB : Data Base).

Na aplicação ClubeCinema foi necessária a utilização do escopo de persistência SessionScoped, para o que o objeto da classe Bean persistisse enquanto o usuário alternava o uso das páginas da sessão para cadastrar e pesquisar filmes.

Na aplicação ClubeCinemaDB a persistência será feita na tabela filmes do banco de dados filmes no SGBD (Sistema de Gerenciamento de Banco de Dados) MySQL. Neste caso, pode ser utilizado o escopo de persistência RequestScope, no qual o objeto da classe Bean persiste somente em uma dada página.

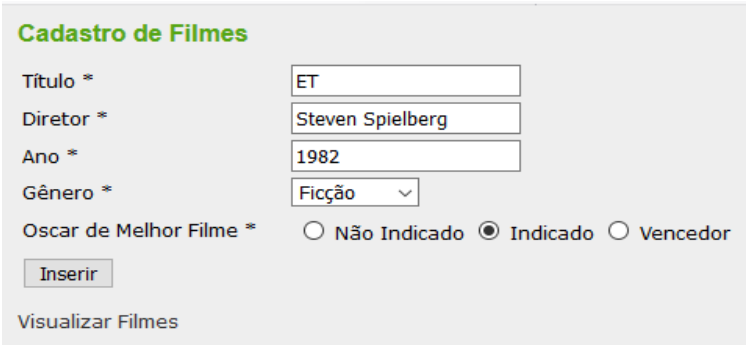
Um efeito colateral da utilização deste escopo mais restrito é a necessidade de separar a página de visualização e da página de cadastro. Quando o usuário cadastra, o filme é armazenado no banco de dados. Quando o usuário visualiza os filmes na tabela, os filmes são previamente consultados no banco de dados. Quando o usuário pesquisa filmes de um dado diretor, a lista de diretores é gerada a partir de uma consulta no banco de dados, bem como, a lista de filmes dirigidos pelo diretor selecionado.

A utilização de um escopo com persistência mais restrita é recomendável, sempre que possível, pois diminui o tempo no qual um objeto precisa ser mantido no servidor de aplicações. Lembre-se que o servidor de aplicações, em uma aplicação web, atende vários usuários em paralelo.

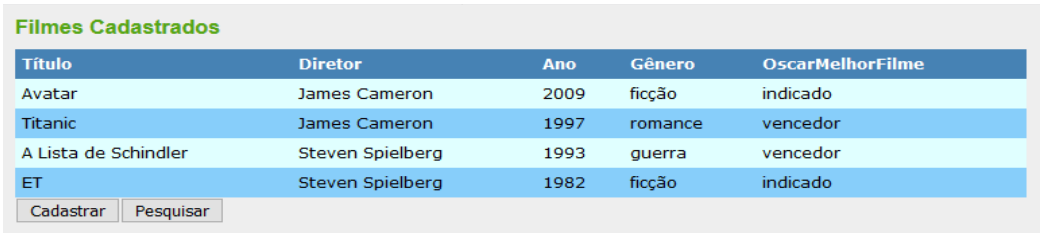
Pelo mesmo motivo, é importante otimizar a utilização de recursos no SGBD. Observe nos métodos que serão utilizados para executar um comando SQL no banco de dados, que após a execução de um dado comando, e utilização de seus resultados no caso de consultas, os recursos do banco (lista de consultas, comando e conexão com o banco) são liberados (closed). Na realidade, para otimizar o tempo de processamento, o comando de liberação de conexão com o banco não resulta na destruição da conexão. É utilizado um pool de conexões (uma fila de conexões disponíveis) . Cada vez que uma conexão é liberada, ela volta a ficar disponível no pool, para evitar a perda de tempo de processamento associada à criação e destruição de uma conexão a cada nova demanda.

A página dinâmica de Pesquisa de Filmes por Diretor não foi alterada. Portando serão ilustradas somente as páginas: Visualização de Filmes Cadastrados e Cadastro de Filmes. As mensagens de erro no preenchimento do cadastro de um filme também não foram alteradas e, portanto, não serão ilustradas.

Cadastrando o quarto filme na página de Cadastro de Filmes.



Visualizando os filmes cadastrados na página de Visualização de Filmes.



Página dinâmica de Visualização de Filmes Cadastrados

- botão de comando (commandButton) para cadastrar filme
  - ação resulta na renderização da página cadastro
- botão de comando (commandButton) para pesquisar filmes por diretor
  - idem da página pesquisa

```
        </h:dataTable>
        <h:form>
            <h:commandButton value="Cadastrar" action="cadastro"/>
            <h:commandButton value="Pesquisar" action="pesquisa"/>
        </h:form>
    </h:body>
</html>
```

Página dinâmica de Cadastro de Filmes

- botão de comando (commandButton) para inserir filme no banco de dados e renderizar a página de Visualização de Filmes
- link (outputLink) para renderizar a página de visualização de Filmes Cadastrados
  - este link é utilizado para que o usuário tenha a opção de retornar à visualização de filmes sem ter cadastrado nenhum filme adicional

```
            <h:panelGrid>
                <h:commandButton value="Inserir" action="#{clubeCinemaDBBean.cadastrarFilme()}"/>
            </h:panelGrid>
            <p><h:outputLink value="index.xhtml">Visualizar Filmes</h:outputLink></p>
        </h:form>
    </h:body>
</html>
```

Os arquivos de estilo css não foram alterados.

A única alteração na entidade Filme é criar um construtor adicional para inicializar todos os atributos da classe Filme.

Bean da aplicação web ClubeCinemaDB

- importações
- nome do objeto criado automaticamente (@Named)
- escopo de persistência do objeto da classe Bean (@RequestScoped)
  - persistência somente na página atual é compensada pela utilização do banco de dados

```
package beans;

import entidades.Filme;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.annotation.PostConstruct;
import javax.annotation.Resource;
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;
import javax.faces.model.SelectItem;
import javax.sql.DataSource;

@Named(value = "clubeCinemaDBBean")
@RequestScoped
public class ClubeCinemaDBBean {
```

Bean da aplicação web ClubeCinemaDB

- anotação da fonte de dados (@Resource)
  - a fonte de dados é criada no console do servidor de aplicações
- atributos e referências
  - inclusão de lista de mensagens de erro
    - foi prevista para armazenar as mensagens de erro
      - mas não está sendo utilizado um componente gráfico para mostrar as mensagens de erro
- inicialização (@PostConstruct)

```
@Resource(lookup = "java:/ClubeCinemaDBDS")
private DataSource filmeDataSource;

private ArrayList<Filme> filmes;
private Filme filme;
private String diretorSelecionado;
private boolean cadastrar;
private boolean pesquisar;
private ArrayList<String> mensagensErro;

@PostConstruct
public void init() {
    filmes = new ArrayList();
    filme = new Filme();
    mensagensErro = new ArrayList();
    cadastrar = false;
    pesquisar = false;
}
```

Bean da aplicação web ClubeCinemaDB

- obter filmes
  - alterado para chamar método que atualiza o atributo filmes
    - a partir dos registros de filmes lidos no banco de dados

```
public ArrayList<Filme> getFilmes() throws SQLException {
    getFilmesDB();
    return filmes;
}
```

Bean da aplicação web ClubeCinemaDB

- cadastrar um filme
  - é alterado para inserir o filme no banco de dados

```
public String cadastrarFilme() throws SQLException {
    saveFilmeDB();
    reset();
    return "index";
}
```

Os métodos pesquisarFilme e reset não são alterados.

A seguir, serão ilustrados os métodos criados para o acesso ao banco de dados.

Bean da aplicação web ClubeCinemaDB

- método para criar uma conexão no banco de dados

```
public Connection getConnectionDB() {
    Connection conexão = null;
    if (filmeDataSource == null) {
        mensagensErro.add("DataSource não acessível");
        return null;
    }
    try { conexão = filmeDataSource.getConnection(); }
    catch (SQLException exception) {mensagensErro.add(exception.getMessage()); }
    return conexão;
}
```

Bean da aplicação web ClubeCinemaDB

- método para salvar as informações de um filme no banco de dados

```
public void saveFilmeDB() throws SQLException {
    Connection conexão = getConnectionDB();
    if (conexão == null) return;
    PreparedStatement comando = null;
    try {
        comando = conexão.prepareStatement
            ("INSERT INTO Filmes (Titulo, Diretor, Ano, Genero, OscarMelhorFilme)"
                + "VALUES (?, ?, ?, ?, ?)");
        comando.setString(1, filme.getTítulo());
        comando.setString(2, filme.getDiretor());
        comando.setString(3, filme.getAno());
        comando.setString(4, filme.getGênero());
        comando.setString(5, filme.getOscarMelhorFilme());
        comando.executeUpdate();
        comando.close();
    } catch (SQLException exception) {
        if (comando != null) comando.close();
        mensagensErro.add(exception.getMessage());
    }
    conexão.close();
    return;
}
```

Bean da aplicação web ClubeCinemaDB

- consultar os filmes do banco de dados
- atualizar a lista de filmes da classe Bean
- e retornar o string da página a ser renderizada

```
public String getFilmesDB() throws SQLException {
    String próxima_página = "";
    ArrayList<Filme> filmes = new ArrayList();
    Connection conexão = getConnectionDB();
    if (conexão == null) return "";
    PreparedStatement comando = null;
    ResultSet consultas = null;
    try {
        comando = conexão.prepareStatement
            ("SELECT Titulo, Diretor, Ano, Genero, OscarMelhorFilme FROM Filmes ORDER BY Ano DESC");
        consultas = comando.executeQuery();
        while (consultas.next()) {
            Filme filme = new Filme(consultas.getString("Titulo"), consultas.getString("Diretor"),
                consultas.getString("Ano"), consultas.getString("Genero"),
                consultas.getString("OscarMelhorFilme"));
            filmes.add(filme);
        }
        consultas.close();
        comando.close();
        próxima_página = "index";
    } catch (SQLException exception) {
        if (consultas != null) consultas.close();
        if (comando != null) comando.close();
        mensagensErro.add(exception.getMessage());
    }
    conexão.close();
    this.filmes = filmes;
    return próxima_página;
}
```

Bean da aplicação web ClubeCinemaDB

- para obter os diretores associados aos filmes cadastrados
  - sem duplicar nomes na lista de diretores retornados

```
public ArrayList<String> getDiretoresDB() throws SQLException {
    ArrayList<String> diretores = new ArrayList();
    Connection conexão = getConnectionDB();
    if (conexão == null) return diretores;
    PreparedStatement comando = null;
    ResultSet consultas = null;
    try {
        comando = conexão.prepareStatement("SELECT Diretor FROM Filmes ORDER BY Diretor");
        consultas = comando.executeQuery();
        while (consultas.next()) {
            String diretor = consultas.getString("Diretor");
            int total_diretores = diretores.size();
            if ((total_diretores == 0) || (!diretor.equals(diretores.get(total_diretores - 1))))
                diretores.add(diretor);
        }
        consultas.close();
        comando.close();
    } catch (SQLException exception) {
        if (consultas != null) consultas.close();
        if (comando != null) comando.close();
        mensagensErro.add(exception.getMessage());
    }
    conexão.close();
    return diretores;
}
```

Bean da aplicação web ClubeCinemaDB

- gerar itens da lista de diretores
  - alterado somente para utilizar a leitura de diretores a partir do banco de dados

```
public ArrayList<SelectItem> getDiretoresItens() throws SQLException {
    ArrayList<SelectItem> itens = new ArrayList();
    for (String diretor : getDiretoresDB()) {
        boolean inserido = false;
        for (int n = 0; n < itens.size(); n++) {
            if (diretor.compareTo(itens.get(n).getLabel()) > 0) continue;
            itens.add(n, new SelectItem(diretor, diretor));
            inserido = true;
            break;
        }
        if (!inserido) itens.add(new SelectItem(diretor, diretor));
    }
    return itens;
}
```

Bean da aplicação web ClubeCinema.

- criar lista com informações sobre filmes de diretor escolhido
  - alterado para ler os filmes cadastrados do banco de dados

```
public ArrayList<String> getInfoFilmesDiretor() throws SQLException {
    getFilmesDiretorDB();
    ArrayList<String> info_filmes_diretor = new ArrayList();
    for (Filme filme : filmes) {
        String diretor = filme.getDiretor();
        if (diretor.equals(diretorSelecionado)) {
            boolean inserido = false;
            String ano_filme = filme.getAno();
            for (int n = 0; n < info_filmes_diretor.size(); n++) {
                String ano = info_filmes_diretor.get(n).split(" - ")[0];
                if (ano_filme.compareTo(ano) < 0) continue;
                info_filmes_diretor.add(n, filme.toString(true));
                inserido = true;
                break;
            }
            if (!inserido) info_filmes_diretor.add(filme.toString(true));
        }
    }
    return info_filmes_diretor;
}
```

Bean da aplicação web ClubeCinemaDB

- gerar itens da lista de filmes para um dado diretor
  - em ordem decrescente de ano

```
public void getFilmesDiretorDB() throws SQLException {
    ArrayList<Filme> filmes = new ArrayList();
    Connection conexão = getConnectionDB();
    if (conexão == null) return;
    PreparedStatement comando = null;
    ResultSet consultas = null;
    try {
        comando = conexão.prepareStatement("SELECT * FROM Filmes WHERE Diretor = ? ORDER BY Ano DESC");
        comando.setString(1, diretorSelecionado);
        consultas = comando.executeQuery();
        while (consultas.next()) {
            Filme filme = new Filme(consultas.getString("Titulo"), consultas.getString("Diretor"),
                consultas.getString("Ano"), consultas.getString("Genero"),
                consultas.getString("OscarMelhorFilme"));
            filmes.add(filme);
        }
        consultas.close();
        comando.close();
        this.filmes = filmes;
    } catch (SQLException exception) {
        if (consultas != null) consultas.close();
        if (comando != null) comando.close();
        mensagensErro.add(exception.getMessage());
    }
    conexão.close();
    return;
}
```

5 - Exercícios para Fixação dos Conceitos Aprendidos

Estude detalhadamente todo o tutorial e revise os conceitos aprendidos, antes de iniciar a implementação desses exercícios. Se aparecer alguma dúvida na realização de algum exercício, deixe o exercício de lado e estude novamente as seções do material associadas a sua dúvida. Então, retome seu exercício sem consultar o material pontualmente para resolvê-lo.

Exercício 1

Implemente a aplicação web Biblioteca para cadastrar livros em um banco de dados relacional, utilizando comandos SQL, com as seguintes páginas dinâmicas:

a) página para visualizar os livros cadastrados;

b) página para cadastrar um dado livro no banco de dados;

c) e páginas para pesquisar livros cadastrados de acordo com os seguintes filtros: ano mínimo de publicação, prefixo do título e assunto do livro.

Exercício 2

Implemente uma aplicação de cadastro e pesquisa de sua escolha, sem utilizar banco de dados, com pelo menos três filtros de pesquisa.