LPIII - Modelo Objeto-Relacional

1 - Criando a Arquitetura da Aplicação Web no NetBeans

Os tutoriais desta disciplina estão sendo escritos em uma linguagem mais informal, para tornar a leitura mais amigável. Para facilitar a leitura quando um parágrafo ou trecho de código não couber no final da página, será transferido para a próxima página, portanto, não estranhe espaços em branco ao final de uma página.

Inicialmente você irá construir a arquitetura, baseada no modelo objeto-relacional, que será utilizada na aplicação web ClubeCinemaOR (OR : Object-Relational). Siga cuidadosamente os passos, descritos nessa seção. Lembre-se que qualquer descuido na execução desses passos poderá resultar um perda substancial de tempo para identificar e corrigir o erro cometido.

Passo 1 : Criação do Projeto e Configurações Iniciais

- 1A - Criar o projeto como uma aplicação web no NetBeans
- 1B - Adicionar bibliotecas ao projeto
- 1C - Criar o banco de dados no NetBeans
- 1D - Criar a fonte de dados no console do WildFly

1A - Criar o projeto como 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 : ClubeCinemaOR
    - 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
    - Selecione os frameworks que você deseja utilizar em sua aplicação Web.
      - assinale JavaServer Faces
    - Configuração Java ServerFaces
      - Componentes
        - Selecione pacotes de componentes JSF para configurar a aplicação Web.
          - assinale PrimeFaces

1B - Adicionar bibliotecas ao projeto

- criar uma pasta lib do projeto e fazer o download, armazenando na pasta lib, das seguintes bibliotecas:
  - commons-filesupload-1.3.jar
  - south-street-1.0.10.jar (tema utilizado nas páginas dinâmicas)
  - primesfaces-4.0.jar (nova biblioteca de componentes gráficos)
    - remover primesfaces-5.0.jar
- no projeto : Bibliotecas
  - Adicionar JAR/pasta
    - adicionar as três bibliotecas da pasta lib
  - Adicionar Biblioteca
    - EclipseLink

1C - Criar o banco de dados no NetBeans

- banco de dados : cinema
- no modelo objeto-relacional
  - não é necessário criar as tabelas do banco com script SQL
    - as entidades do projeto serão denotadas com informações
      - a partir das quais o banco de dados será criado automaticamente
  - lembre-se de remover o banco de dados e criá-lo novamente
    - após qualquer alteração realizada em classes entidade

1D - Criar a fonte de dados no console do WildFly (Tutorial 1 : Passo 7)

- ative o servidor de aplicações WildFly
- utilize o console do WildFly para criar a fonte de dados
  - nome da fonte de dados : ClubeCinemaORDS
    - nome do projeto acrescido do sufixo DS (Data Source)
      - utilizar esse nome em Datasource Attributes : Name e JNDI Name
  - nome do banco de dados : cinema
    - utilizar esse nome em ConnectionSettings : ConnectionURL

Passo 2 : Criar a primeira Entidade e a Interface Comum

Classes de entidade serão persistidas no banco de dados. Para que isso ocorra no modelo objeto-relacional, serão associadas a um unidade de persistência.

No projeto ClubeCinemaOR

- 2A - Criar entidade entities.Diretor
- 2B - Criar a interface comum
- 2C - Entidade Diretor deve implementar a interface PersistentEntity

2A - Criar entidade entities/Diretor : projeto ClubeCinemaOR → Novo → Outros

- Escolher Tipo de Arquivo
  - Categorias : Persistência
  - Tipo de Arquivo : Classe da Entidade
- Nome e Localização
  - Nome da Classe : Diretor
  - Pacote: entities
  - Tipo de Chave Primária : Long
  - Criar Unidade de Persistência
- Provedor e Banco de Dados
  - Nome da Unidade de Persistência : ClubeCinemaORPU
    - nome do projeto com sufixo PU (Persistence Unit)
  - Provedor de Persistência: EclipseLink
  - Origem de Dados : selecionar opção java:/ClubeCinemaORDS
  - Utilizar APIs de Transação Java
  - Estratégia de Geração de Tabela: Criar

2B - Definir interface comum

- criar interface entities.PersistentEntity
  - com assinaturas dos métodos de leitura e alteração do identificador da entidade

```
package entities;

public interface PersistentEntity {
    public Long getId();
    public void setId(Long id);
}
```

2C - Entidade Diretor deve implementar a interface PersistentEntity

```
@Entity
public class Diretor implements Serializable, PersistentEntity {
```

Passo 3 : Criando as Classe EJB (Enterprise JavaBeans)

As classes EJB suportam os métodos utilizados para persistência de objetos das classes entidades na base de dados. Além de classes EJB comuns (EntityService e EntityFinder) , que serão criadas no pacote services, cada classe entidade terá uma classe EJB correspondente.

No projeto ClubeCinemaOR

- 3A - Criar classes EJB para a entidade Diretor no pacote services
- 3B - Adaptar classe gerada AbstractFacade para EntityService
- 3C - Adaptar classe gerada DiretorFacade para DiretorService
- 3D - Criar a classe services.EntityFinder

3A - Criar classes EJB para a entidade Diretor : projeto ClubeCinemaOR → Novo → Outros

- Escolher Tipo de Arquivo
  - Categorias : Enterprise JavaBeans
  - Tipo de Arquivo : Beans de Sessão para Classes de Entidade
- Classes de Entidade
  - selecionar entities.Diretor em Classes de Entidade Disponíveis
  - e adicionar em Classes de Entidade Selecionadas
- Beans de sessão gerados
  - Pacote : services

3B - Adaptar classe gerada AbstractFacade para EntityService

- renomear (refatorando) AbstractFacade para EntityService
- adaptar a classe EntityService conforme código abaixo

Classe EntityService

- importações
- definição como classe abstrata cujo tipo genérico T estende PersistentEntity
- definir referência em do tipo EntityManager denotando como contexto de persistência (@PersistenceContext)
  - nome da unidade de persistência : nome do projeto com sufixo PU (Persistent Unit)
- construtor

```
package services;

import entities.PersistentEntity;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

public abstract class EntityService<T extends PersistentEntity> {
    @PersistenceContext(unitName = "ClubeCinemaORPU")
    protected EntityManager em;

    private Class<T> entityClass;

    public EntityService(Class<T> entityClass) { this.entityClass = entityClass; }
```

Classe EntityService

- suporta os métodos de persistência de objetos das classes entidade no banco de dados
  - classe entidade é passada como parâmetro genérico T

```
public void create(T entity) { em.persist(entity); }
public void edit(T entity) { em.merge(entity); }
public void remove(T entity) { em.remove(em.merge(entity)); }
public T find(Object id) { return em.find(entityClass, id); }

public List<T> getAll() {
    CriteriaQuery query = em.getCriteriaBuilder().createQuery();
    query.select(query.from(entityClass));
    return em.createQuery(query).getResultList();
}
public long getCount() {
    CriteriaQuery query = em.getCriteriaBuilder().createQuery();
    Root<T> root = query.from(entityClass);
    query.select(em.getCriteriaBuilder().count(root));
    return (Long) em.createQuery(query).getSingleResult();
}
protected T attach(T entity) { return find(entity.getId()); }
```

3C - Adaptar classe gerada DiretorFacade para DiretorService

- renomear (refatorando) DiretorFacade para DiretorService
- adaptar a classe DiretorService conforme código abaixo

```
package services;

import entities.Diretor;
import javax.ejb.Stateless;

@Stateless
public class DiretorService extends EntityService<Diretor> {
    public DiretorService() { super(Diretor.class); }
}
```

3D - Criar a classe services.EntityFinder

- conforme código abaixo

```
package services;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class EntityFinder {
    @PersistenceContext(unitName = "ClubeCinemaORPU")
    private EntityManager em;

    public Object find(Class type, Object id) { return em.find(type, id); }
}
```

Passo 4 : Criar a classe util/RequestParameters

A classe util/RequestParameters é gerada manualmente, conforme o código abaixo, para capturar os parâmetros passados na URL. Evita a utilização de código Java ServerFaces nos Beans e a repetição de código em cada Bean.

```
package util;

import java.io.Serializable;
import java.util.Map;
import javax.annotation.PostConstruct;
import javax.faces.context.FacesContext;

public class RequestParameters implements Serializable {

    private Map<String, String> params;

    @PostConstruct
    public void init() {
        FacesContext faces_context = FacesContext.getCurrentInstance();
        params = faces_context.getExternalContext().getRequestParameterMap();
    }

    public String get(String param) { return params.get(param); }
}
```

Passo 5 – Criar classe DiretorBean associada a entidade Diretor

Da mesma forma que a classes EJB, cada classe entidade terá uma classe Bean associada. O objeto da classe Bean é referenciado pelas páginas dinâmicas para obter informações ou armazenar informações (antes de persisti-las na base de dados). A classe Bean suporta os métodos que invocam os métodos de persistência de objetos na respectiva classe EJB.

Classe DiretorBean

- importações
- definição com escopo ViewScoped

```
package beans;

import entities.Diretor;
import java.io.Serializable;
import util.RequestParameters;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.faces.view.ViewScoped;
import javax.inject.Inject;
import services.DiretorService;

@Named(value = "diretorBean")
@ViewScoped
public class DiretorBean implements Serializable {
```

Classe DiretorBean

- classe EJB associada
- injeção de objeto da classe RequestParameters
- referência ao objeto da classe entidade : value (mesmo nome para todas as classes Bean)
- inicialização

```
@EJB
private DiretorService diretorService;

@Inject
private RequestParameters parameters;

private Diretor value;

@PostConstruct
public void init() {
    String id = parameters.get("id");
    if (id == null) value = new Diretor();
    else value = diretorService.find(Long.valueOf(id));
}
```

Classe DiretorBean

- leitura e alteração da referência ao objeto da entidade (value)
- reset : criação de objeto para receber informações do formulário de cadastro
- inserir : invocado pelo comando Cadastrar da página dinâmica
  - esse método vai crescer ao longo desse tutorial

```
public Diretor getValue() { return value; }
public void setValue(Diretor value) { this.value = value; }

public void reset () { value = new Diretor(); }
public void inserir() { reset(); }
```

Classe DiretorBean

- métodos que utilizam métodos EJB para persistência de objetos no banco de dados

```
public List<Diretor> getAll() { return diretorService.getAll(); }

public String save() {
    diretorService.create(value);
    reset();
    return null;
}

public String update() {
    diretorService.edit(value);
    return null;
}

public String delete() {
    diretorService.remove(value);
    return null;
}
```

Passo 6 – Criação de classes de conversão de Object para String e vice-versa

- 6A - Criação da classe genérica de conversão : converters/EntityConverter
- 6B - Criação da classe de conversão converters/DiretorConverter associada à entidade Diretor

6A - Criação da classe genérica de conversão : converters/EntityConverter

Classe converters/EntityConverter

- importações
- definição como classe abstrata
  - com tipo genérico T implementando a interface Converter
- referência genérica entityClass
- construtor

```
package converters;

import entities.PersistentEntity;
import services.EntityFinder;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public abstract class EntityConverter<T> implements Converter {
    EntityFinder entityFinder = lookupEntityFinder();

    private Class<T> entityClass;

    public EntityConverter(Class<T> entityClass) { this.entityClass = entityClass; }
```

Classe converters/EntityConverter

- método de busca

```
private EntityFinder lookupEntityFinder() {
    try {
        Context c = new InitialContext();
        return (EntityFinder) c.lookup("java:global/ClubeCinemaOR/EntityFinder!services.EntityFinder");
    } catch (NamingException ne) {
        Logger.getLogger(getClass().getName()).log(Level.SEVERE, "exception caught", ne);
        throw new RuntimeException(ne);
    }
}
```

Classe converters/EntityConverter

- métodos de conversão para Object e para String

```
@Override
public Object getAsObject(FacesContext context, UIComponent component,
                          String value) {
    try {
        Long id = Long.valueOf(value);
        return entityFinder.find(entityClass, id);
    }
    catch (NumberFormatException e) { throw new ConverterException("formato de ID invalido", e); }
}

@Override
public String getAsString(FacesContext context, UIComponent component, Object value) {
    return ((PersistentEntity) value).getId().toString();
}
```

6B - Criação da classe de conversão converters/DiretorConverter associada à entidade Diretor

```
package converters;

import entities.Diretor;
import javax.faces.convert.FacesConverter;
import javax.inject.Named;

@FacesConverter(forClass = Diretor.class)
@Named
public class DiretorConverter extends EntityConverter<Diretor> {

    public DiretorConverter() {
        super(Diretor.class);
    }
}
```

Passo 7 – Criação e Adaptação de arquivos de configuração

- 7A - Criar e adaptar o arquivo de configuração beans.xml
- 7B - Adaptar o arquivo de configuração web.xml
- 7C - Criar o arquivo de configuração persistence.xml

7A - Criar e adaptar o arquivo de configuração beans.xml

- projeto ClubeCinemaOR → Novo → Outros
  - Escolher Tipo de Arquivo
    - Categorias : Injeção de Dependência e Contextos
    - Tipo de Arquivo : beans.xml (Arquivo de Configuração CDI)
  - Nome e Localização
    - não alterar nenhum campo
- adaptar o arquivo beans.xml
  - alterar bean-discovery-mode : de annotated para all
    - para utilizar injeção de dependência em todas as classes
      - e não apenas aquelas nas quais foi utilizado @Named

7B - Adaptar o arquivo de configuração web.xml para:

- alterar página inicial do sistema

```
<welcome-file-list>
  <welcome-file>index.jsf</welcome-file>
</welcome-file-list>
```

- alterar estágio do projeto para evitar warnings de desenvolvimento

```
<context-param>
  <param-name>javax.faces.PROJECT_STAGE</param-name>
  <param-value>Production</param-value>
</context-param>
```

- acrescentar tag url-pattern para alterar extensão de xhtml para jsf na URL

```
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
  <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

- acrescentar tag context-param para definir tema de estilos do PrimeFaces

```
<context-param>
  <param-name>primefaces.THEME</param-name>
  <param-value>south-street</param-value>
</context-param>
```

- acrescentar tag context-param para interpretar string vazio como null

```
<context-param>
  <param-name>
   javax.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL
  </param-name>
  <param-value>true</param-value>
</context-param>
```

7C - Criar o arquivo de configuração persistence.xml

- projeto ClubeCinemaOR → Novo → Outros
  - Escolher Tipo de Arquivo
    - Categorias : Persistência
    - Tipo de Arquivo : Unidade de Persistência
  - Provedor e Banco de Dados
    - Nome da Unidade de Persistência : ClubeCinemaORPU
    - Provedor de Persistência : selecionar EclipseLink (JPA 2.1)
    - Origem de Dados : selecionar java:/ClubeCinemaORDS
    - assinalar Utilizar APIs de Transação Java
    - Estratégia de Geração de Tabela : selecionar Criar

Passo 8 - Criação dos Arquivos de Estilo

- 8A - Criar a pasta resources no diretório Páginas Web
- 8B - Inserir os arquivos de estilo do projeto ClubeCinema (Tutorial 1 : seção 3):
  - default.css --- overrides.css

Passo 9 - Adaptar as Páginas Dinâmicas criadas automaticamente

- 9A - Adaptar a página welcomePrimeFaces.xhtml para template.xhtml
- 9B - Adaptar a página index.xhtml

9A - Adaptar a página welcomePrimeFaces.xhtml para template.xhtml

- renomer a página (refatorando)
- acrescentar na tag head as folhas de estilo de saída outputStylesheet (Tutorial 1 - seção 3)
  - default.css --- overrides.css
- alterar conteúdo da tag title para : Clube de Amigos do Cinema
- interna à tag body e externa à tag layout
  - acrescentar tag div e tag form (interna à tag div) e acrescentar parâmetros à tag layout

```
<h:body>
  <div style="margin: 0 auto;">
    <h:form id="form">
      <p:layout fullPage="true" style="text-align:left; min-width:1020px;
                max-width:1020px; min-height:600px">
```

- redefinir layout da região norte

```
<p:layoutUnit position="north" size="50" style="text-align:center;">
  <h1>Clube de Amigos do Cinema</h1>
</p:layoutUnit>
```

- redefinir layout da região sul utilizando conteúdo em negrito

```
<p:layoutUnit position="south" size="50" style="text-align:center;">
  <br/><strong>LPIII - Aplicação Web</strong>
</p:layoutUnit>
```

- redefinir layout da região oeste

```
<p:layoutUnit position="west" size="186">
  <p:menu>
    <p:menuitem value="Página Inicial" url="/index.jsf"></p:menuitem>
  </p:menu>
  <p:menu>
    <p:submenu label="Amigos">
      <p:menuitem value="Cadastrar Amigos" url="/index .jsf"/>
    </p:submenu>
    <p:submenu label="Filmes">
      <p:menuitem value="Cadastrar Diretores" url="/cadastrarDiretores.jsf"/>
      <p:menuitem value="Cadastrar Atores" url="/index .jsf"/>
      <p:menuitem value="Cadastrar Filmes" url="/index .jsf"/>
    </p:submenu>
    <p:submenu label="Comentários">
      <p:menuitem value="Cadastrar Comentários" url="/index .jsf"/>
    </p:submenu>
  </p:menu>
</p:layoutUnit>
```

- redefinir layout da região central

```
<p:layoutUnit position="center" style="padding: 10px;>
    <ui:insert  name="conteudo"></ui:insert>
</p:layoutUnit>
```

9B - Alterar a página index.xhtml

- criar pasta resources/images em Páginas Web
- inserir imagem cinema.jpg em resources/images
  - imagem obtida no site
    - https://catracalivre.com.br/agenda/cinema-ao-ar-livre-shell-open-air-sp/
- composição com a página de template (composition)
  - conteúdo a ser inserido na área central da página de template (define)
- usar o seguinte conteúdo em index.xhtml

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:p="http://primefaces.org/ui"
                template="/template.xhtml">
  <ui:define name="conteudo">
    <p:graphicImage name="cinema.jpg" library="images"/>
  </ui:define>
</ui:composition>
```

Segue a ilustração da página inicial da aplicação web ClubeCinemaOR:

Passo 10 - Incluir atributos e métodos associados na entidade Diretor

- incluir atributos : String nome --- boolean ganhadorOscar
  - e métodos de leitura e alteração

Passo 11 - Criar a página dinâmica cadastrarDiretores.xhtml

Página dinâmica cadastrarDiretores

- composição com a página de template (composition)
  - conteúdo a ser inserido na área central da página de template (define)
- painéis para encapsular tabela de visualização de diretores (panel -- outputPanel)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:p="http://primefaces.org/ui"
                template="/template.xhtml">
    <ui:define name="conteudo">
        <h2>Cadastrar Diretores</h2>
        <p:panel header="Diretores Cadastrados">
            <p:outputPanel id="diretor">
```

Página dinâmica cadastrarDiretores

- tabela (dataTable) de visualização de diretores cadastrados

```
<p:outputPanel id="diretor">
    <p:dataTable id="dataTable" var="diretor"
                 value="#{diretorBean.all}"
                 widgetVar="tabelaDiretores"
                 emptyMessage="Nenhum registro encontrado">
        <p:column headerText="Nome">
            <h:outputText value="#{diretor.nome}" />
        </p:column>
        <p:column headerText="Ganhador de Oscar">
            <h:outputText value="#{diretor.ganhadorOscar}" />
        </p:column>
    </p:dataTable>
```

Página dinâmica cadastrarDiretores

- painel (panel) para encapsular botão de comando (commandButton) Cadastrar
  - para renderizar os campos do formulário para cadastrar um diretor

```
            </p:dataTable>
            <p:panel>
                <p:commandButton value="Cadastrar"
                                 actionListener="#{diretorBean.inserir()}"
                                 oncomplete="panelDiretor.show();"
                                 update=":form:displayDiretor" />
            </p:panel>
        </p:outputPanel>
    </p:panel>
```

Página dinâmica cadastrarDiretores

- painel (panel) encapsulando painel de grade (panelGrid)
  - encapsulando os campos do formulário de cadastro
    - campo de texto (inputText)
    - componente para assinalar opção do tipo boolean (selectBooleanCheckbox)
  - label (outputText) e mensagem de erro (message) associados a cada componente de entrada de dados do formulário

```
</p:panel>
<p:panel header="Dados do Diretor" id="dadosDiretor" widgetVar="panelDiretor"
        visible="false" closable="true" style="margin-top:10px;">
    <p:messages id="erroFilme"/>
    <h:panelGrid id="displayDiretor" columns="1" styleClass="grid">
        <p:panel>
            <h:panelGrid columns="3" columnClasses="label, value" styleClass="grid">
                <h:outputText value="Nome *" />
                <p:inputText id="nomeInputText" required="false" label="Nome"
                             value="#{diretorBean.value.nome}" styleClass="ipt-large" />
                <p:message for="nomeInputText" />
                <h:outputText value="Ganhador de Oscar: *" />
                <p:selectBooleanCheckbox id="ganhadorOscarSelectBooleanCheckbox"
                                         value="#{diretorBean.value.ganhadorOscar}" />
                <p:message for="ganhadorOscarSelectBooleanCheckbox"/>
            </h:panelGrid>
        </p:panel>
```

Página dinâmica cadastrarDiretores

- painel de grupo (panelGroup) botão de comando (commandButton) Inserir para persistir diretor no banco de dados

```
                </p:panel>
                <h:panelGroup>
                    <p:commandButton value="Inserir" action="#{diretorBean.save()}"
                                     ajax="false" icon="ui-icon-circle-plus" />
                </h:panelGroup>
            </h:panelGrid>
        </p:panel>
    </ui:define>
</ui:composition>
```

Segue a ilustração da página cadastrarDiretores:

- diretor sendo cadastrado

Segue a ilustração da página cadastrarDiretores:

- diretor inserido

2 - Acrescentar Funcionalidades de Consulta, Alteração e Remoção ao Cadastro

No código ilustrado na seção anterior, o acionamento do comando Cadastrar resulta na renderização: dos campos do formulário a serem preenchidos, e do comando Inserir para incluir os dados do diretor no banco de dados.

Nesta seção, serão descritos os passos necessários para acrescentar funcionalidades ao cadastro de diretores. Será acrescentado o comando Consultar, associado a cada diretor visualizado na tabela de diretores. O acionamento do comando Consultar resulta na renderização: dos campos do formulário com as informações do diretor recuperadas no banco de dados, e dos comandos Alterar e Remover.

Essa melhoria no cadastro de diretores, deverá ser utilizada como referência para a entrega da versão da aplicação web, associada a este tutorial. Após detalhar os passos para orientar a implementação solicitada, serão ilustradas somente as saídas decorrentes, omitindo o código da implementação. A seguir serão descritos os passos necessários para enriquecer o cadastro de diretores.

Acrescentar coluna de consulta na tabela de visualização de diretores

- na classe DiretorBean, acrescentar :
  - método : void consultar (Diretor value) { setValue(value); }
- na página cadastrarDiretores
  - inserir uma coluna na tabela de diretores denominada Ação
  - associar a essa ação o botão de comando Consultar
    - invocando a chamada do método consultar da classe DiretorBean
      - passando diretor como parâmetro do método consultar

Incluir comandos de alteração e remoção de um diretor selecionado

- na página cadastrarDiretores
  - incluir no painel Dados do Diretor o comando Alterar
    - invocando o método update da classe DiretorBean
  - idem para o comando Remover
    - invocando o comando delete da classe DiretorBean

Separar a renderização do comando de inserção da renderização dos comandos de alteração e remoção

- na classe DiretorBean
  - criar a variável : boolean consultar
    - e seu par de métodos de leitura e alteração
  - para sinalizar uma consulta
    - incluir a ativação da variável consultar no método consultar
    - incluir a desativação da variável consultar no método inserir
- criar dois grupos de paineis com panelGroup
  - o primeiro grupo com o comando Inserir
    - a ser renderizado quando a variável consultar não estiver ativa
  - e o segundo grupo com os comandos Alterar e Remover
    - a serem renderizados quando a variável consultar estiver ativa

Segue a ilustração da página cadastrarDiretores:

- funcionalidade cadastrar

Segue a ilustração da página cadastrarDiretores:

- funcionalidade consultar

Segue a ilustração da página cadastrarDiretores:

- funcionalidade alterar : alterando o atributo ganhadorOscar

Segue a ilustração da página cadastrarDiretores:

- funcionalidade remover

3 - Exercícios para Fixação dos Conceitos Aprendidos

Exercício 1
Implemente a aplicação web Biblioteca para cadastrar livros em um banco de dados relacional, utilizando o modelo objeto-relacional, com as seguintes páginas dinâmicas:
a) página inicial da aplicação;
b) página para visualizar e cadastrar, suportanto: visualização dos livros cadastrados, cadastro de livro, e consulta de um livro cadastrado (com opções de alteração ou remoção do livro consultado).

Exercício 2
Implemente uma aplicação de cadastro de sua escolha, utilizando o modelo objeto-relacional.