Tópicos Especiais em Análise e Desenvolvimento de Sistemas

Prof. Rafael Odon (rafael.alencar@prof.una.br)

ORIENTAÇÕES PARA ENTREGA:

- O roteiro deverá ser feito continuando **com a mesma dupla** dos roteiros de JPA e ou **individual** para quem decidiu não fazer em dupla.
- Enviar até o início da aula do dia 23/05/2014 o código e as respostas de **todas as perguntas** desse roteiro para o e-mail do professor (rafael.alencar@prof.una.br) com o título:
  - `Roteiro JSF 03 – Nome do 1º Aluno / Nome do 2º Aluno`
- Lembre-se: **todas as informações** para responder as perguntas são dadas em **sala de aula**. Preste atenção na aula e faça notas pessoais!
- As **respostas** devem ser pessoais e **escritas com suas palavras**. Não serão aceitas respostas longas e/ou **copiadas** de forma literal do material de referência ou Internet.

Java Server Faces

Roteiro de Aula 03

Esse Roteiro se baseia no uso dos softwares:

- Netbeans 7.4 para JAVA EE
- GlassFish 4
- Java SDK 1.7
- Java EE 6
- Java DB (Apache Derby)

Fase 1: Desenvolvendo um CRUD de Bean Complexo com JSF + JPA

No roteiro anterior criamos nossos primeiros CRUDs, ainda simples, abordando as entidades que possuíam apenas as propriedades ID e Nome. Agora, vamos abordar uma entidade um pouco mais complexa do nosso sistema. Vamos construir a funcionalidade **Manter Modelo** por se tratar de uma entidade que se relaciona com Fabricante e possui além do nome um campo não textual (ano).

1. Crie a classe **ModeloMB** no pacote **sysfrota.ui.managedbean** com o seguinte código:

```
package sysfrota.ui.managedbean;

import sysfrota.persistence.dao.ModeloDAO;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import sysfrota.entidades.Fabricante;
import sysfrota.entidades.Modelo;
import sysfrota.persistence.dao.FabricanteDAO;

@ManagedBean
@SessionScoped
public class ModeloMB{

    private static final String PAGINA_EDICAO = "/modeloEdit.xhtml";
    private static final String PAGINA_LISTAGEM = "/modeloList.xhtml";

    Modelo modelo;
    List<Modelo> listaModelos;
    ModeloDAO modeloDAO = new ModeloDAO();
    FabricanteDAO fabricanteDAO = new FabricanteDAO();
    List<Fabricante> listaFabricantes;

    public ModeloMB() {
```

```
    }

    @PostConstruct
    private void atualizar() {
        modelo = new Modelo();
        listaModelos = modeloDAO.listarTodos();
        listaFabricantes = fabricanteDAO.listarTodos();
    }

    public String editar(Modelo modelo) {
        this.modelo = modeloDAO.carregar(modelo);
        return PAGINA_EDICAO;
    }

    public String criar() {
        this.modelo = new Modelo();
        return PAGINA_EDICAO;
    }

    public String remover(Modelo c) {
        modeloDAO.remover(c);
        irParaListagem();
    }

    public String salvar() {
        modeloDAO.salvar(modelo);
        irParaListagem();
    }

    public String cancelar() {
        irParaListagem();
    }

    public String irParaListagem() {
        atualizar();
        return PAGINA_LISTAGEM;
    }

    //getters & setters
    public List<Modelo> getListaModelos() {
        return listaModelos;
    }

    public List<Fabricante> getListaFabricantes() {
        return listaFabricantes;
    }

    public Modelo getModelo() {
        return modelo;
    }

    public void setModelo(Modelo modelo) {
        this.modelo = modelo;
    }
}
```

1. O formato geral de um ManagedBean de CRUD você já conhece do roteiro anterior. Mas observe agora que esse MB, além de ter uma lista de Modelos (entidade que está sendo mantida), **também trabalha com uma lista da Fabricantes.**
   Essa lista será utilizada para montar uma Caixa de Seleção (Combo Box) com os Fabricantes disponíveis.

2. Crie o arquivo **modeloList.xhtml** com o código abaixo:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:f="http://java.sun.com/jsf/core"
                template="/template.xhtml">
    <ui:define name="corpo">
        <h:form>
```

```
            <h:commandButton action="#{modeloMB.criar()}" value="Novo modelo" />
            <h2>Lista de modelos</h2>
            <h:dataTable value="#{modeloMB.listaModelos}" var="m" border="1">
                <h:column>
                    <f:facet name="header">Id</f:facet>
                    #{m.id}
                </h:column>
                <h:column>
                    <f:facet name="header">Nome</f:facet>
                    #{m.nome}
                </h:column>
                <h:column>
                    <f:facet name="header">Ano</f:facet>
                    #{m.ano}
                </h:column>
                <h:column>
                    <f:facet name="header">Fabricante</f:facet>
                    #{m.fabricante.nome}
                </h:column>
                <h:column>
                    <f:facet name="header">Opções</f:facet>
                    <h:commandButton action="#{modeloMB.editar(m)}" value="Editar" />
                    <h:commandButton action="#{modeloMB.remover(m)}" value="Remover" />
                </h:column>
            </h:dataTable>
        </h:form>
    </ui:define>
</ui:composition>
```

3. Observe que essa tela de listagem de modelos não apresenta nenhuma novidade em relação à tela de listagem de um CRUD simples que já conhecemos.

4. Crie o arquivo **modeloEdit.xhtml** com o código abaixo:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:f="http://java.sun.com/jsf/core"
                template="/template.xhtml">
    <ui:define name="corpo">
        <h2>Editar modelo</h2>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="Nome"/>
                <h:inputText value="#{modeloMB.modelo.nome}" />

                <h:outputLabel value="Ano"/>
                <h:inputText value="#{modeloMB.modelo.ano}" maxlength="4" label="Ano">
                    <f:convertNumber type="number" integerOnly="true" maxIntegerDigits="4" groupingUsed="false"/>
                </h:inputText>

                <h:outputLabel value="Fabricante"/>
                <h:selectOneMenu value="#{modeloMB.modelo.fabricante}" converter="fabricanteConverter"
                        label="Fabricante" >
                    <f:selectItem noSelectionOption="true" itemLabel="Selecione..." />
                    <f:selectItems value="#{modeloMB.listaFabricantes}" var="f" itemValue="#{f}"
                        itemLabel="#{f.nome}"/>
                </h:selectOneMenu>
            </h:panelGrid>
            <h:commandButton action="#{modeloMB.salvar}" value="Salvar" />
            <h:commandButton action="#{modeloMB.cancelar}" value="Cancelar" immediate="true" />
        </h:form>
    </ui:define>
</ui:composition>
```

5. Observe as novidades dessa tela de edição. Começamos com **um campo de texto ligado à propriedade ano do Modelo**. Essa propriedade é de um tipo numérico. Para tanto foi usado nele **uma facet de validação de conversão chamada <f:convertNumber />** que diz que o

tipo do campo é numérico, apenas inteiros, que o máximo de dígitos aceitos são 4 e que não é usado separador de milhar.

6. Observe também que o relacionamento do Modelo com os Fabricantes é visualizado através de uma caixa de seleção definida pelo componente **<h:selectOneMenu />**. O atributo value desse componente está apontando para o fabricante do modelo do nosso MB, ou seja, tanto o valor selecionado dessa caixa equivalerá ao fabricante armazenando em modelo, quanto a seleção feita nessa caixa irá modificar o fabricante do modelo editado.

7. Em seguida é definido o **uso de um converter**, que será discutido mais adiante.

8. Perceba que interno ao componente **<h:selectOneMenu />**, temos a possibilidade de definir os elementos que compõem a caixa de seleção. Primeiramente definimos **um item de seleção de valor vazio** que pode ser selecionado se o Modelo não tiver nenhum fabricante.

9. Ainda dentro do **<h:selectOneMenu>** definimos também que o restantes dos itens da caixa de seleção serão obtidas da nossa lita de fabricantes do MB através da facet **<f:selectItems />.** Observe que assim como o <h:dataTable>, nela ocorrerá uma iteração em elementos, e podemos dar um apelido para a variável que representa esses elementos para utilizar no valor e na descrição dos items de seleção.

10. Não execute a aplicação ainda! Algumas etapas serão necessárias para que tudo funcione.

11. Crie a classe FabricanteConverter no pacote sysfrota.ui.converters:

```
package sysfrota.ui.converter;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.FacesConverter;
import javax.inject.Named;
import sysfrota.entidades.Fabricante;
import sysfrota.persistence.dao.FabricanteDAO;

@FacesConverter(value="fabricanteConverter")
public class FabricanteConverter implements Converter {

    FabricanteDAO fabricanteDAO = new FabricanteDAO();

    @Override
    public Fabricante getAsObject(FacesContext context, UIComponent component, String value) {
        try{
            Fabricante f = fabricanteDAO.carregarPeloId(Long.valueOf(value));
            return f;
        }catch(Exception e){
            return null;
        }
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        if (value != null) {
            return ((Fabricante) value).getId().toString();
        } else {
            return null;
        }
    }
}
```

12. Um converter é um recurso do JSF utilizado para converter Strings para Objetos e Objetos para Strings. Isso é necessário pois como você deve se lembrar do estudo de Fundamentos e Desenvolvimento Web, todos os parâmetros enviados de um formulário HTML chegam para o servidor como String. Dessa forma, quando um determinado parâmetro representa um objeto Java, é preciso criar uma classe que entenda como fazer essa conversão. Eis aí o papel dos Convertes do JSF.

13. Observe que o nosso Converter **implementa a interface javax.faces.convert.Converter.**

14. Observe ainda que nosso Converter **está anotado com @FacesConverter** onde é dito seu apelido para ser referenciado via EL.

15. A interface Converter do JSF pede que sejam implementados 2 métodos: **getAsString (obter como String)** – responsável por transformar um objeto em um texto que o represente, no nosso caso, retornando o id do fabricante; e **getAsObject (obter como objeto)** – reponsável por pegar um texto que representa um objeto e encontrar o objeto respectivo, no nosso acaso, feito através de uma busca no banco de dados pelo fabricante de Id informado.

16. Adicione o método **equals** abaixo na classe **Fabricante**:

```
...
    @Override
    public boolean equals(Object o) {
        if(o == null || !(o instanceof Fabricante)){
            return false;
        }
        Fabricante other = (Fabricante) o;
        return this.getId().equals(other.getId());
    }
...
```

17. O método **equals** é um método muito importante inicialmente definido na classe Object do Java. Lembre-se que todos as classes do Java naturalmente herdam de Object, sendo assim, todas podem sobrescrever o equals original.

O equals é utilizado para comparar instâncias e deve retornar verdadeiro apenas se ambas a instâncias são significativamente iguais, mesmo sendo objetos diferentes na memória.

Ao utilizar caixas de seleção no JSF, após a obtenção de um objeto do Converter, é feito um equals em cada item da listagem afim de validar se o objeto obtido realmente estava no domínio da lista utilizada. Não implementar o equals da entidade utilizada na caixa de seleção levaria a um erro de validação.

18. Crie um novo link no menu no seu template para dar acesso à página **fabricanteList.xhtml** através do método **#{fabricanteMB.irParaListagem()}**

19. Limpe e construa sua aplicação e execute-a! Acesse a funcionalidade de Manter Fabricantes e tente criar alguns deles, assim como editá-los e removê-los a partir da listagem.

Fase 2: Melhore o CRUD criado!

1. Observe que o CRUD criado na Fase 1 não possui tratamento de exceção e envio de mensagens de sucesso/falha adicionadas ao contexto de mensagens do JSF, assim como foi exemplificado no CRUD Simples do Roteiro 01. Melhore o código para contemplar tais mensagens.

2. Inspirado na solução do Roteiro 01, faça o campo nome, ano e fabricante serem requeridos na tela de edição de Modelo.

RESPONDA:

I. Que componente pode ser utilizado juntamente com um <h:inputText /> para validar se o tipo é numérico? Dê exemplos de variações desse componente para diferentes tipos de valores numéricos.
II. Pesquise e descubra outro componentes parecidos com os da questão I, mas que valida campos do tipo Data. Dê exemplos de variações desse componente para diferentes tipos de datas.
III. Explique a utilização do componente <h:selectOneMenu>. Como podemos informar a propriedade para a qual o valor selecionado será mapeado? Como podemos informar a lista de itens da caixa de seleção? Como podemos informar um iten de valor vazio?

IV. Qual o papel do Converter no JSF? Como implementar um Converter? Em que situação ele foi necessário? Dê um exemplo.
V. Qual a importância do método equals no Java e em que situação ele foi necessário no JSF?

Fase 3: Faça o CRUD de Carro!

1. A partir do aprendizado obtido no Roteiro 01, 02 e 03, crie uma versão mais básica do CRUD de Carro contendo os campos: Placa, Modelo, Cor, Ano, Chassi e Quilometragem.

2. Inicialmente liste todos os Modelos sem filtrar por Fabricante.

3. Lembre-se de criar um Converter para Modelo e de implementar o método equals da classe para que o SelectOneMenu funcione corretamente.

3. Inclua validação nativa do JSF nos campos numéricos (Ano e Quilometragem).

4. Todos os campos devem ser requeridos, exceto Quilometragem.

5. No JSF, é possível criar v*alidators* (validadores) personalizados para serem executados durante a fase de validação do ciclo de vida do JSF. Vamos criar um! Crie no Managed Bean de carro o seguinte método:

```
...
    public void validarAno(FacesContext context, UIComponent validate,Object value) {
        Long ano = (Long) value;
        if (ano > Calendar.getInstance().get(Calendar.YEAR)) {
            throw new ValidatorException(
               new FacesMessage(FacesMessage.SEVERITY_ERROR,"Ano não pode ser superior ao ano corrente.",null));
        }
    }
...
```

6. No XHTML do formulário de edição de Carro, referencie o novo método criado através da propriedade validator do input de Ano: `validator="#{carroMB.validarAno}"`

7. Observe que um *validator* deve receber certos parâmetros, dentre eles **o objeto que guarda o valor digitado no campo** que o acionou. No código do *validator* é possível verificar se o valor é valido segundo alguma lógica. Caso não seja, basta lançar uma **ValidatorException contendo uma mensagem JSF** de erro correspondente.

8. O *validator* criado acima verifica se o ano digitado não é superior ao ano corrente. Teste sua aplicação e veja se funcoinou.

RESPONDA:

I. Ao invés de criar métodos validadores em um ManagedBeans específico, é possível criar uma classes validadora para ser reutilizadas em diversos pontos do sistema. Pesquisa e descubra como criar uma classe validadora no JSF. Dê um exemplo.
II. Recorra ao material didático da disciplina ou à Internet e responda, considerando o ciclo de vida de uma requisição no JSF:
   a) Em que fase ocorre a execução dos Converters?
   b) Em que fase ocorre a execução dos Validators? O que deve acontecer antes dessa fase?
   c) Em que fase o Managed Bean recebe as alterações feitas no formulário pelo usuário? O que deve acontecer antes dessa fase?
   d) Em que fase o método indicado pelo link ou botão clicado pelo usuário é executado? O que deve acontecer antes dessa fase?

Fase 4: Adicione o combo de Fabricante ao Crud de Carro

Ao invés de listar todos os Modelos no formulário de criação de Carro, vamos fazer com que o usuário possa filtrar os modelos pelos Fabricantes, diminuindo o espaço de busca. Para isso, vamos utilizar Ajax.

1. Modifique seu XHTML de edição de Carro para ter um novo <h:selectOneMenu /> com a lista de fabricantes.

```
...
<h:outputLabel value="Fabricante" for="fabricante"/>
<h:selectOneMenu id="fabricante" value="#{carroMB.fabricante}" converter="fabricanteConverter">
    <f:selectItem itemLabel="Selecione..." noSelectionOption="true" />
    <f:selectItems value="#{carroMB.listaFabricantes}" var="f" itemLabel="#{f.nome}" itemValue="#{f}"/>
    <f:ajax listener="#{carroMB.atualizarModelo}" execute="fabricante" render="modelo" />
</h:selectOneMenu>

<h:outputLabel value="Modelo" for="modelo"/>
<h:selectOneMenu id="modelo" value="#{carroMB.carro.modelo}" converter="modeloConverter"
    disabled="#{empty carroMB.listaModelos}">
    <f:selectItem itemLabel="Selecione..." noSelectionOption="true" />
    <f:selectItems value="#{carroMB.listaModelos}" var="m" itemLabel="#{m.nome} #{m.ano}" itemValue="#{m}"/>
</h:selectOneMenu>
...
```

2. Observe o <h:selectOneMenu /> de Fabricante. Nele foi utilizado o **converter de Fabricante** (já utilizado no Crud de Modelo). O fabricante selecionado nesse combo irá ser armazenado **na propriedade fabricante do Managed Bean de Carro**, que, naturalmente é do tipo Fabricante. Crie essa propriedade.

```
...
   public class CarroMB {
       ...
       Fabricante fabricante;
       ...
       public Fabricante getFabricante() { ... }
       public void setFabricante(Fabricante fabricante) { ... }
       ...
   }
...
```

3. Observe também que o <h:selectOneMenu /> de Fabricante traz os valores de **uma lista de Fabricantes existente no seu Managed Bean**. Crie essa propriedade e carregue essa lista na inicialização do seu Managed Bean e também antes de editar/criar um carro.

4. Observe a introdução da **tag <f:ajax /> dentro do <h:selectOneMenu />** de fabricante. Essa tag irá **executar o método indicado no seu atributo *listener* sem fazer refresh** na página, através da tecnologia Ajax. Crie esse método de modo que ele carregue a lista de modelos a partir da **propriedade fabricante** criada anteriormente.

5. Na tag **<f:ajax />** também foi configurado na **propriedade *execute*** o id dos campos que irão participar da submissão Ajax. Já na **propriedade *render*** configura-se o id campos que deverão ser atualizados quando a submissão/reposta ajax estiver completa. Sendo assim, no código acima, o valor do **campo fabricante** será enviado, e ao obter a reposta, o **campo modelo** será atualizado.

6. Observe também que o campo modelo só fica habilitado quando a lista de modelos não está vazia, comportamento configurado na **propriedade disabled**.

7. Certifique que todas as propriedades e métodos foram criados para que esse trecho de código funcione. Execute sua apliação e teste tentando criar um novo carro.

8. Atualize seu método de editar do Managed Bean de Carro para que ao carregar o carro editado a **propriedade fabricante** seja atualizada corretamente. Execute sua apliação e teste tentando editar um novo carro.

RESPONDA:

I. O que significa a sigla Ajax?
II. Qual a diferença entre uma requisição Ajax e uma requisição comum?

III. Em que situações é vantajoso utilizar Ajax?
IV. Que tag da JSF permite adicionar comportamento Ajax aos componentes? Qual o namespace dessa tag? O que deve ser informado nas propriedades **listener**, **execute** e **render** dessa tag?

Fase 5: Incremente sua exclusão com verificação de integridade referencial

Um problema recorrente quando criamos aplicativos com persistência em bancos de dados relacionais é o tratamento da integridade referencial no Banco de Dados. Ao excluir registros pode haver referencias para ele. Uma maneira de resolver isso é configurando o comportamento de Remoção em Cascata, que pode ser feito via anotações JPA. No entanto, nem sempre esse é o comportamento desejado.

9. Adicione o seguinte método ao DAO de Fabricante:

```
...
   public boolean possuiReferencias(Fabricante fabricante){
      Query query = em.createQuery("SELECT COUNT(m) FROM Modelo m WHERE m.fabricante = :f");
      query.setParameter("f", fabricante);
      Long qtdReferencias = (Long) query.getSingleResult();
      return qtdReferencias > 0;
   }
...
```

10. Observe que o método verifica quantos modelos existem para um determinado fabricante pois existe um relacionamento @ManyToOne de Modelo para Fabricante.

11. Agora é a sua vez de colocar a mão na massa! No seu Managed Bean de Fabricante, verifique antes de remover um Fabricante se ele possui referências, utilizando para isso o novo método adicionado ao DAO. Se houver referências, não execute a remoção e exiba uma mensagem JSF de erro adequada.

12. Aplique também essa solução para verificar integridade referencial nos CRUD de Modelo e Característica.

Fase 6: Adicione relacionamento com característica

Após as melhorias que fizemos com esse Roteiro já temos o CRUD de Carro. No entanto ele ainda não se relaciona com as características. Que tal esse **desafio**?

1. Pesquisa sobre a utilização do **componente <h:selectManyCheckbox />**, que permite a criação de lista de caixas de marcação (checkboxes). Descubra como utilizá-lo.

> **Atenção:** esse componente trabalha de forma muito parecida com o *<h:selectOneMenu />*. No entanto, você precisará de duas listas: uma dos objetos listados cada um como um checkbox e outra que irá guardar os objetos marcados. Não esqueça de observar o método *equals* desses objetos!

2. Incremente o seu CarroMB para manter uma lista de características além da lista de fabricantes e da lista de modelos.

3. Incremente o seu formulário de criação de Carro para apresentar um <h:selectManyCheckbox /> que permite criar um carro com as características selecionadas. Verifique no banco se estão sendo persistidas corretamente as características desejadas.

4. Garanta também que o componente <h:selectManyCheckbox /> funcionará corretamente para edição de Carros já existentes, trazendo pré-marcadas as características dos carros já salvas.