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 23/04/14 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 01 – 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 01 – 11/04/2014

Esse Roteiro se baseia no uso dos softwares:

- Netbeans 7.4.1 Java EE
- Java DB (Apache Derby)
- Java Development Kit (JDK) 1.7
- Java EE 6

- Se o seu NetBeans não for a versão Java EE (Java Enterprise Edition – usado para aplicações Web), remova-o, baixe e instale essa versão:
  - https://netbeans.org/downloads/index.html (clique em Download na segunda coluna, Java EE)

  ou

- Se o seu NetBeans não for a versão Java EE, tente acessar: **Ferramentas > Plugins > Plugins Disponíveis**, marque as opções:
  - *PrimeFaces*
  - *JSF*
  - *Java EE Base*
  - *Glassfish Server 3*
- Clique em **Instalar**, clique em **Próximo**, aguarde a instalação e em seguida **reinicie** o Netbeans.

Fase 1: Configurando a aplicação

1. Crie um novo projeto **Java Web** no Netbeans de nome **RoteiroJSF01**
   - Acessar o menu Arquivo > Novo Projeto > Categoria **Java Web** > Aplicação Web. Clicar em Próximo.
   - No campo nome do projeto, digitar **RoteiroJSF01.** Clicar em Próximo.

- Escolher o servidor **GlassFish 4.x**. Clicar em Próximo.
- Escolher a versão do Java EE: **JAVA EE 6** (ou algo próximo na versão 6)
- Deixar as outras opções como padrão.
- Na tela de escolha de frameworks, marcar a opção **JavaServer Faces**. Abas serão habilitadas na parte de baixo da janela. Na **Bibliotecas** escolher a **biblioteca JSF 2.1 ou JSF 2.2**, a que estiver disponível na versão 2. Na aba **Configurações**, definir como padrão de URL do JSF Servlet o texto: *.jsf
- Clica em **Finalizar**

 Caso o servidor GlassFish não esteja instalado, tente uma das opções:

- Baixe e instale pelo próprio Netbeans no momento da primeira utilização;
- Baixe do site do GlassFish o ZIP da ultima versão estável Full Platform, extraia para alguma pasta do seu computador e adicione ao Netbeans manualmente:
  - http://download.java.net/glassfish/4.0/release/glassfish-4.0.zip

2. Com o projeto criado, clique com o botão direito nele, expanda o item Bibliotecas, e expanda o item GlassFish. Você verá que uma śerie de bibliotecas java (arquivos .jar) estão sendo fornecidas pelo próprio Servidor de Aplicação GlassFish, dentre eles o JSF (javax.faces.jar).
3. **Edite o arquivo Páginas Web > WEB-INF/web.xml.** Perceba que nesse arquivo XML constam diversas configurações sobre a aplicação, inclusive algumas relativas ao JSF e à pagina inicial da aplicação.
   O **web.xml** é um arquivo extremamente importante em aplicações Java Web. Ele é conhecido como **Descritor de Implantação** (*Deployment Descriptor*), e é utilizado pelo Servidor de Aplicação (no nosso caso o GlassFish) no momento para configurar a aplicação em questão corretamente no momento em que ela é inicializada.
4. Substitua o conteúdo do arquivo **Páginas Web > index.xhtml** pelo o seguinte:

```
<?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">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Bem vindo</title>
    </h:head>
    <h:body>
        <h:outputText value="Bem vindo!"/>
    </h:body>
</html>
```

5. Com o botão direito no projeto peça para **Limpar e Construir** e em seguida peça para **Executar**. Aguarde a implantação ser feita e o Glassfish ser iniciado.
6. Acesse no seu navegador o endereço http://localhost:8080/RoteiroJSF01/
7. Verifique se foi apresentada uma página em branco com o texto “Bem vindo!”.
8. Peça para ver o código-fonte dessa página apresentada no navegador. Compare e veja se o HTML gerado foi parecido com o 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">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
        <title>Bem vindo</title></head><body>Bem vindo!</body>
</html>
```

Fase 2: Criando sua primeira tela em XHTML

1. Crie um XHTML chamada sobre.xhtml, configurado para o JSF:
    - Botão direito no projeto, Novo > Página JSF. Caso a opção não apareça, tente: Novo > Outros > JavaServer Faces > Página JSF.
    - Na tela seguinte, no campo **Nome do Arquivo** digite apenas **sobre** (perceba que a extensão .xhtml é colocada automaticamente, não é preciso digitá-la)
    - Clique em Finalizar. Verifique se aparece o arquivo **sobre.xhtml** abaixo de Páginas Web no seu projeto.
    - **DICA:** Após criar esse arquivo, certifique-se de que o conteúdo do **web.xml** não foi alterado.
2. Edite o arquivo **sobre.xhtml** criado.
3. Perceba que por se tratar de um XHTML, ele é também um XML versão 1.0.
4. Perceba que a tag raiz <html /> declara o uso de um **namespace** de prefixo **h** através do texto `xmlns:h="http://java.sun.com/jsf/html"`. Esse namespace faz com que além das tags do HTML convencionais tenhamos também disponíveis para uso um outro conjunto de tags, nesse caso tags para facilitar escrita de HTML com JSF.
5. Vamos explorar o namespace **h**. Ainda no arquivo **sobre.xhtml**, digite dentro do corpo (entre as tags `<h:body />`) apenas `<h:` e pressione CTRL+ESPAÇO para ativar o aucomplete do Netbeans. Verifique na lista de autocomplementos as tags que esse namespace trouxe. Tente imaginar para que elas servem baseado em seus nomes.
6. Continue digitando o restante da tag de modo a escrever no arquivo a tag <h:outputText />
7. Adicione a essa tag a propriedade **value** com valor **Sobre**:
   `<h:outputText value="Sobre" />`
8. Com o botão direito no projeto peça para **Limpar e Construir** e em seguida peça para **Executar**. Aguarde a implantação ser feita e o Glassfish ser iniciado.
9. Acesse no seu navegador o endereço http://localhost:8080/RoteiroJSF01/sobre.jsf
    - Perceba que não foi acessado o documento sobre.xhtml, e sim um endereço simbólico **sobre.jsf**.
    - O controlador do JSF foi configurado para interceptar todas as chamadas de endereços no padrão ***.jsf** e então atuar entregando a versão HTML da tela descrita no XHTML.
    - Na prática esse controlador é um Servlet chamado *FacesServlet*.
    - Você se lembra que na disciplina de Fundamentos e Desenvolvimento WEB chegamos a construir vários Servlets, um para cada página da aplicação? Cada um respondia num padrão de URL específico.
    - A primeira facilidade que muitos frameworks MVC trazem é prover um controlador único que elimina a necessidade da criação de Servlets.

10. Peça para exibir o código-fonte da página criada e perceba qual foi a relação entre o componente <h:outputText /> e o HTML gerado.

Fase 3: Navegando entre telas

1. Volte a editar a pagina **index.xhml** e insira os seguintes componentes no corpo da página, abaixo do texto de boas vindo já criado:

```
<h:form>
    <h:commandLink action="sobre.xhtml" value="Sobre o projeto..." />
</h:form>
```

2. Visite a página inicial novamente no navegador e verifique se foi criado um link com o texto "Sobre o projeto..." que ao ser clicado direciona para a página sobre.jsf

3. Volte na página anterior, a que contém o link, e peça para exibir o código-fonte. Perceba que dessa vez o componente **<h:commandLink />** foi responsável por fazer bem mais do que simplesmente escrever um texto na tela.

4. Perceba também que foi preciso incluir um **<h:form />** ao redor do link. Você consegue imaginar por que isso foi necessário?

   - Componentes de tela JSF trazem consigo vários aspectos embutidos tais como comportamentos do lado do cliente com HTML, CSS e JavaScript e também comportamentos do lado do servidor através de classes Java.

   - Os detalhes por detrás de um componente na maioria dos casos são desconhecidos por quem implementa, daí precisamos nos preocupar apenas com questões específicas da lógica do negócio do sistema que está sendo desenvolvido.

5. Inspirado no link que você acabou de criar da página **index** para a página **sobre**, crie o link contrário, ou seja, um texto "Voltar" na página **sobre** que ao ser clicado direciona para a página inicial.

6. Relembre um pouco do HTML e crie alguns elementos visuais nas suas páginas **sobre** e **index**. Lembre-se: as tags do HTML não morreram! Utilize **<h1>, <h2>, <h3>** para fazer títulos e subtítulos. Utilize **<p>** para parágrafos. Utilize **<hr>** para fazer linhas horizontais. Tente verificar se as tags que você está utilizando podem ser substituídas por componentes do namespace **h** e teste sua utilização.

RESPONDA:

I. Para que serve o arquivo **web.xml**? Onde ele fica localizado?
II. O que é um XHTML? Qual a sua vantagem sobre o HTML? Por que o JSF o utiliza?
III. Por que não acessamos os arquivos XHTML diretamente, mas sim endereços terminados em **.jsf**?
IV. É correto dizer que quando escrevemos um XHTML no JSF estamos escrevendo a página vista pelo usuário? Qual a relação entre a página vista pelo usuário na web e o XHTML associado?
V. No contexto do XHTML, o que é um namespace? Dê um exemplo do universo do JSF.
VI. Escolha 3 tags do namespace **h** do JSF e explique que componentes elas representam, e que HTML será gerado ao utilizá-la.
VII. Qual a vantagem de utilizar um componente JSF no XHTML ao invés de escrever seu código equivalente em HTML diretamente?
VIII. É correto dizer que ao utilizar JSF **não** estamos utilizando Java Servlets? Justifique sua resposta.

Fase 4: Seu primeiro ManagedBean (MB)

1. Crie uma classe chamada SobreMB com o seguinte código:

```
import javax.faces.bean.ManagedBean;

@ManagedBean
public class SobreMB{

    private String nome = "Rafael Odon";
    private String email = "rafael.alencar@prof.una.br";

    public void teste(){
        System.out.println("Você clicou em teste!");
    }

    public String voltar(){
        return "index.html";
    }

    public String getEmail() {
        return email;
    }

    public String getNome() {
        return nome;
    }

}
```

2. Personalize a classe com valores corretos para a variável **nome** e **e-mail** para representar seu dados.

    - Essa classe é o que chamamos de **Managed Bean** (vulgo MB), que é nada mais que um JavaBean para representar a parte Java de uma tela JSF, ou, no contexto do MVC, o modelo por detrás de uma visualização.

    - As propriedades e métodos desse JavaBean são acessíveis em qualquer XHTML através da **Linguagem de Expressão (EL)**, o que nos permite fazer uma cola entre o que o usuário vê e o modelo orientado a objeto do Java.

    - Dessa forma, se precisarmos de um texto dinâmico em uma página ou da execução de alguma lógica a partir de uma ação do usuário, os nossos MB deverão conter todas as propriedades e métodos necessários para isso acontecer.

3. Modifique o código da sua pagina sobre.xhtml para ficar da seguinte forma:

```
<?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">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h1><h:outputText value="Sobre" /></h1>
        <br/>
        Autor: <h:outputText value="#{sobreMB.nome}" />
        <br/>
        Email: <h:outputText value="#{sobreMB.email}" />
        <h:form>
            <h:commandButton action="#{sobreMB.teste()}" value="Teste" />
            <br/>
            <h:commandLink action="#{sobreMB.voltar()}" value="Voltar" />
        </h:form>
    </h:body>
</html>
```

4. No XHTML acima, perceba que **os valores dos <h:outputText /> agora são propriedades de um objeto chamado sobreMB.**

    - Ao utilizar a anotação ***@javax.faces.bean.ManagedBean*** na classe que representa um MB, elas são automaticamente disponibilizados para serem acessados via EL.

    - Se não for configurado nada, o nome de um MB na EL é o nome de sua classe com a primeira letra minúscula. Ex: *SobreMB → sobreMB*

5. Observe que a EL do XHTML do JSF tem a sintaxe: tralha, abre chaves, expressão, fecha chaves. Ex: *#{sobreMB.nome}*

    - Antes da classe SobreMB ser um Managed Bean, ela é um JavaBean.

    - Todas as propriedades que possuem um método *get* respectivo, estrão disponíveis para serem acessadas via EL diretamente através do operador ponto.

    - No exemplo acima, como criamos os métodos **getNome()** e **getEmail()**, podemos acessar via EL as propriedades **#{sobreMB.nome}** e **#{sobreMB.email}**.

6. Perceba que o **método teste() da classe SobreMB foi mencionado como ação do botão com texto Teste.**

7. Perceba ainda que o link que tínhamos antes para voltar para a página inicio foi modificado. A página de retorno advém do **resultado da execução do método #{sobreMB.voltar()}**, que no caso da nossa implementação, retorna justamente o texto “index.html”.

8. Limpe e construa seu projeto e em seguida execute-o. Acesse a página **sobre.jsf** novamente e teste o comportamento programado.

9. Verifique se seu **nome** e seu **e-mail** foram exibidos.

10. Clique no botão **Teste** e veja se no console do GlassFish no Netbeans houve a saída esperada.

11. Clique no link **Voltar** e veja se o retorno do método direcionou para a página correta.

    - Os ManagedBeans e suas interações com os XHTML abrem novas portas!

    - Perceba que as possibilidade de acessar propriedades de uma classe Java, ou de executar seus métodos ao clicar em links e botões da tela trazem várias oportunidades de implementação.

    - Para que essa interação do MB com o XHTML seja possível, uma instância da classe SobreMB precisa existir na memória quando a página é visitada. Quem gerencia essa relação é o JSF.

    - O controlador do JSF além de fazer a mágica de converter nossos XHTML em telas HTML, também é capaz de compreender que um clique em um botão nesse HTML deverá executar um determinado método de uma classe java.

Fase 5: Olá Mundo ao estilo JSF

1. Crie uma classe chamada **OlaMB** com o seguinte código:

```
import javax.faces.bean.ManagedBean;

@ManagedBean
public class OlaMB {

    private String nome;

    public void digaOla() {
        //faça nada
    }

    public String getNome() {
```

```
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

2. Crie uma o arquivo **ola.xhtml** com o seguinte código:

```
<?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">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Olá</title>
    </h:head>
    <h:body>
        <h1>Olá #{olaMB.nome}!</h1>
        <h:messages />
        <h:form>
            Nome: <h:inputText value="#{olaMB.nome}" />
            <h:commandButton action="#{olaMB.digaOla()}" value="Diga olá" />
        </h:form>
    </h:body>
</html>
```

3. Limpe e construa seu projeto e em seguida execute-o. Verifique o funcionamento do código criado acessando a página **ola.jsf**.

4. Discuta em sala com o professor e os colegas algumas novidades apresentadas pelo código apresentado, e as possibilidades trazidas.

5. Modifique o ManagedBean para que o nome exibido fique todo em maiúsculo.

Fase 6: Somador ao estilo JSF

1. Crie uma classe chamada **SomadorMB** com o seguinte código:

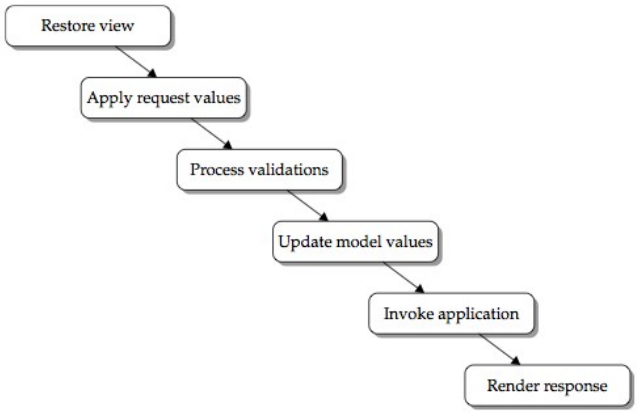
```
import javax.faces.bean.ManagedBean;

@ManagedBean
public class SomadorMB {

    private Integer termoA=0;
    private Integer termoB=0;
    private Integer soma=0;

    public void somar(){
        soma = termoA + termoB;
    }

    public Integer getSoma() {
        return soma;
    }

    public Integer getTermoA() {
        return termoA;
    }

    public Integer getTermoB() {
        return termoB;
    }

    public void setTermoA(Integer termoA) {
        this.termoA = termoA;
    }

    public void setTermoB(Integer termoB) {
        this.termoB = termoB;
    }
}
```

2. Crie uma o arquivo **somador.xhtml** com o seguinte código:

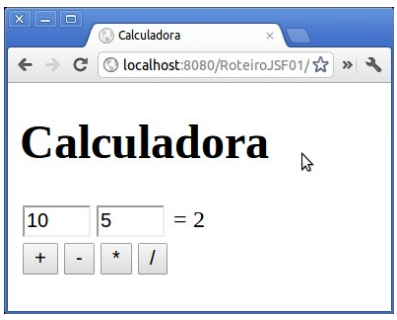
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <title>Somador</title>
    </h:head>
    <h:body>
        <h1>Somador</h1>
        <h:messages />
        <h:form>
            <h:inputText value="#{somadorMB.termoA}" size="4" label="Primeiro termo"/>
            +
            <h:inputText value="#{somadorMB.termoB}" size="4" label="Segundo termo"/>
            =
            <h:outputText value="#{somadorMB.soma}" />
            <br/>
            <h:commandButton action="#{somadorMB.somar()}" value="Calcular" />
        </h:form>
    </h:body>
</html>
```

3. Limpe e construa seu projeto e em seguida execute-o. Verifique o funcionamento do código criado acessando a página **somador.jsf.**

4. Tente somar valores não numéricos e perceba o que aconteceu. Entenda que componentes foram responsáveis pelo resultado esperado e discuta com o professor e os colegas o ciclo de vida do JSF.



5. Inspirado na solução desse exercício crie uma calculadora. Crie o ManagedBean e o XHTML para contemplar 4 botões: somar, subtrair, multiplicar e dividir. Veja o exemplo da tela esperada abaixo:



DESAFIO: Não permita divisão por zero! Trate (dica: use try/catch) e exiba uma mensagem. Pesquise e descubra como enviar uma mensagem de texto do ManagedBean para tela, intergindo com o componente <h:messages /> (o mesmo que foi incluido no XHTML do Somador de exemplo).

RESPONDA:

I. No contexto do JSF, o que é um Managed Bean (MB)? Qual o seu papel?
II. O que é Linguagem de Expressão (EL)? Onde ela é utilizada no JSF? Qual a sua sintaxe geral?
III. Por padrão, com que nome um MBs pode ser acessado via EL? Que anotação é responsável por fazer essa disponibilização?
IV. Um MB, assim como um JavaBean, tem propriedades e apresenta operações. O que no código do MB define a existência de uma propriedade? Como podemos acessar essa propriedade via EL? Dê um exemplo.
V. Cite, explique e exemplifique duas tags que permitem acionar métodos de um MB.