

PROGRAMAÇÃO PARA WEB I

AULA 15

Profa. Silvia Bertagnolli

JSF

PÁGINAS WEB

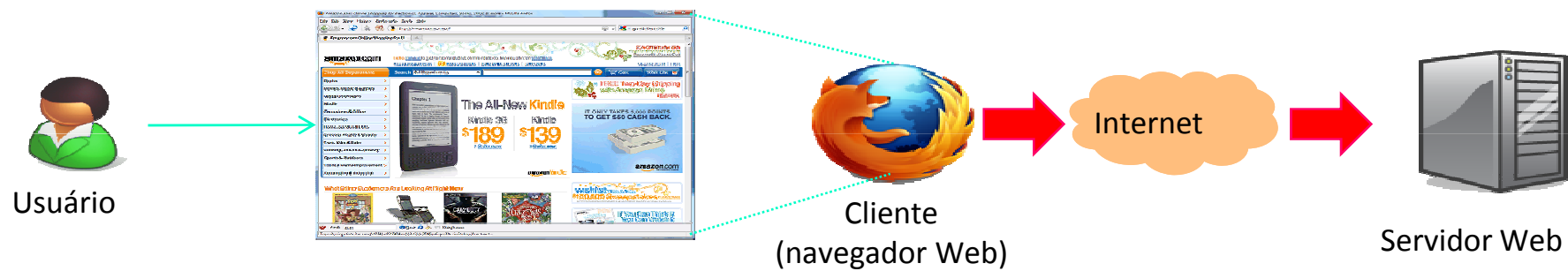
Páginas estáticas: uma página estática é aquela que, independentemente da situação sempre exibirá o mesmo conteúdo

Note que a formatação pode variar conforme modificamos os estilos vinculados (CSS)

Já o conteúdo é fixo ele só muda se editarmos a página

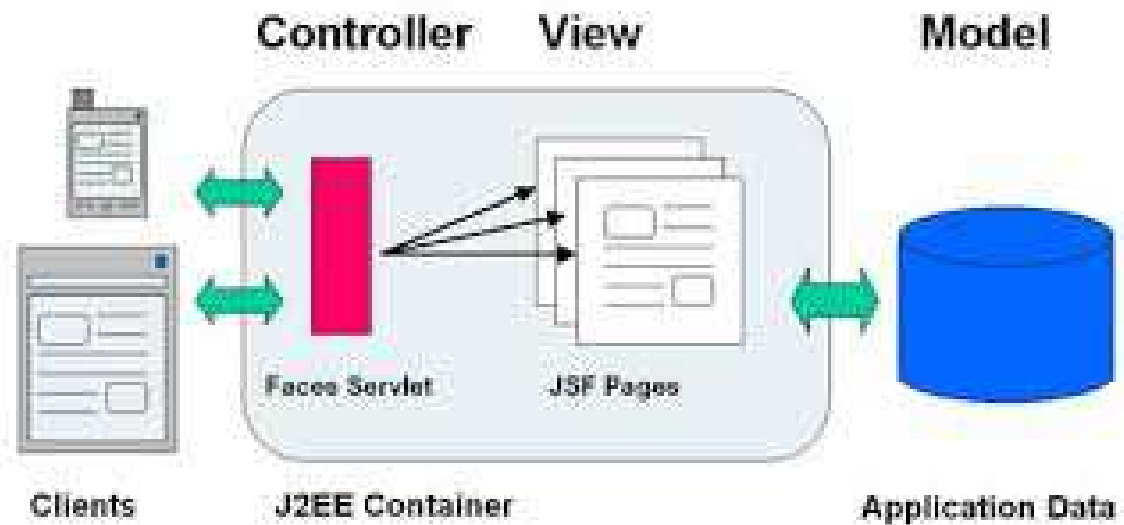
Página dinâmica: uma página é dinâmica quando o seu conteúdo sofre alterações, em um servidor, quando a página é “ativada” no navegador

ARQUITETURA WEB

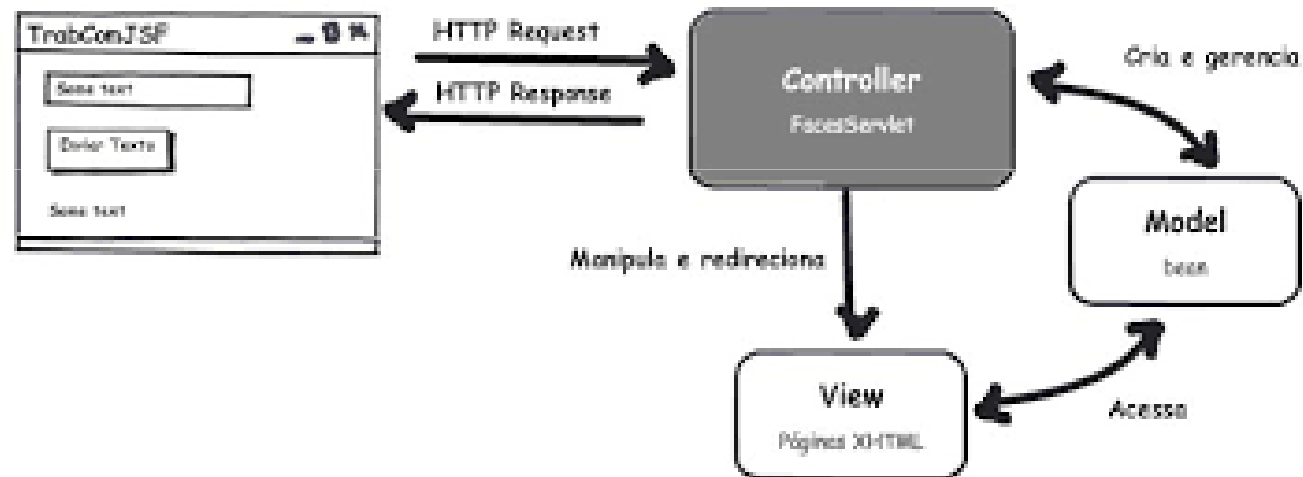


MVC

MVC — MODEL VIEW CONTROLLER



MVC — MODEL VIEW CONTROLLER



INTRODUÇÃO

O QUE É JSF?

Tecnologia que permite criar aplicações Java para Web utilizando componentes visuais pré-prontos, de forma que o desenvolvedor não se preocupe com Javascript e HTML

Basta adicionar os componentes (calendários, tabelas, formulários) e eles serão renderizados e exibidos em formato HTML

JSF: CARACTERÍSTICAS

Guarda o estado dos componentes

Separa as camadas

Especificação para várias implementações:

- JSF é uma especificação do Java EE
- Todo servidor de aplicações Java tem que vir com uma implementação dela

JSF: CARACTERÍSTICAS

Implementações do JSF:

Mojarra disponível em <http://javaserverfaces.java.net/>

MyFaces da *Apache Software Foundation* em:
<http://myfaces.apache.org/>

QUAL ESPECIFICAÇÃO VAMOS USAR?

Mojarra:

sem componentes sofisticados

especificação trata do que é fundamental

The screenshot shows a web form with the following elements:

- A text input field containing "JSF".
- A text input field containing "*****".
- A text area containing "JSF é component-based."
- A dropdown menu showing "JSF".
- A row of radio buttons with labels: JSF, Tapestry, vaadin, Wicket, GWT. The "vaadin" radio button is selected.
- A list box showing the options: JSF, Tapestry, vaadin, Wicket, GWT.
- A button labeled "salva".
- A blue underlined link labeled "salva".

<https://www.caelum.com.br/apostila-java-testes-jsf-web-services-design-patterns/introducao-ao-jsf-e-primefaces/>

VIEW

OUTRAS BIBLIOTECAS

- As bibliotecas possuem componentes e validadores avançados
- Para se ter uma ideia das potencialidades acesse “Demo” no PrimeFaces



<http://richfaces.org/>



<http://primefaces.org/>

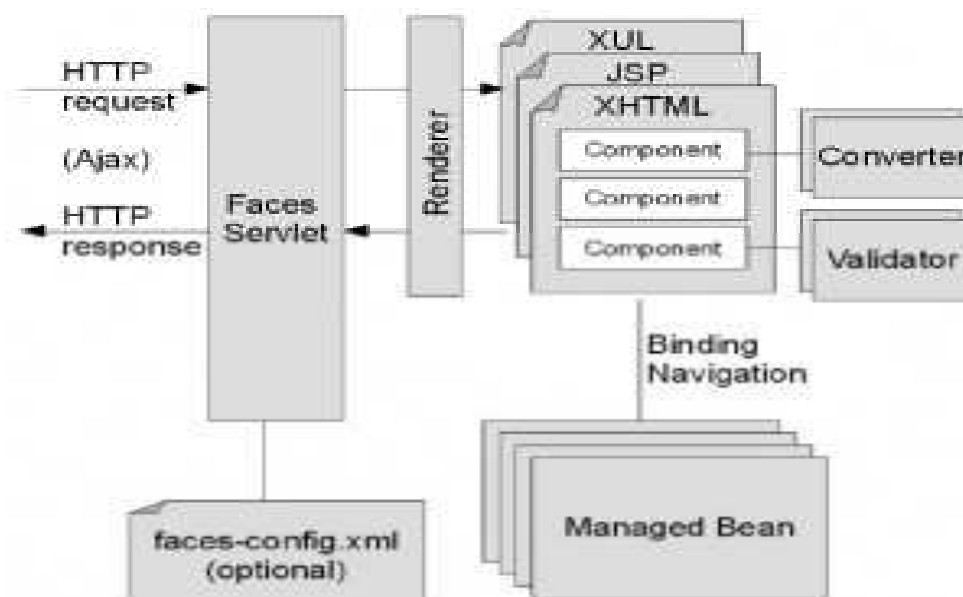


<http://icefaces.org/>

<https://www.caelum.com.br/apostila-java-testes-jsf-web-services-design-patterns/introducao-ao-jsf-e-primefaces/>

ARQUITETURA

JSF: ARQUITETURA



Fonte: <http://www.edsongoncalves.com.br/tag/jsf-2-0/>

JSF: ARQUITETURA

FacesServlet – é o servlet principal para a aplicação

Renderers – são os responsáveis por exibir um componente e traduzir uma entrada de valor realizada por um usuário em componente

Páginas XHTML, JSP – o JSF permite mais de um tipo de arquivo para renderizar seus componentes (pode ser páginas JSP ou Facelets)

JSF: ARQUITETURA

Converters – convertem o valor de um componente (como datas, moedas, porcentagem e outros) dando-lhes novos formatos

Validators – responsáveis por validar a entrada ocorrida no componente pelo usuário

Managed Bean – a lógica do negócio é gerenciada pelos managed beans, controlando também a navegação das páginas

Ajax – permite enviar/receber dados usando Ajax

PROJETO NETBEANS

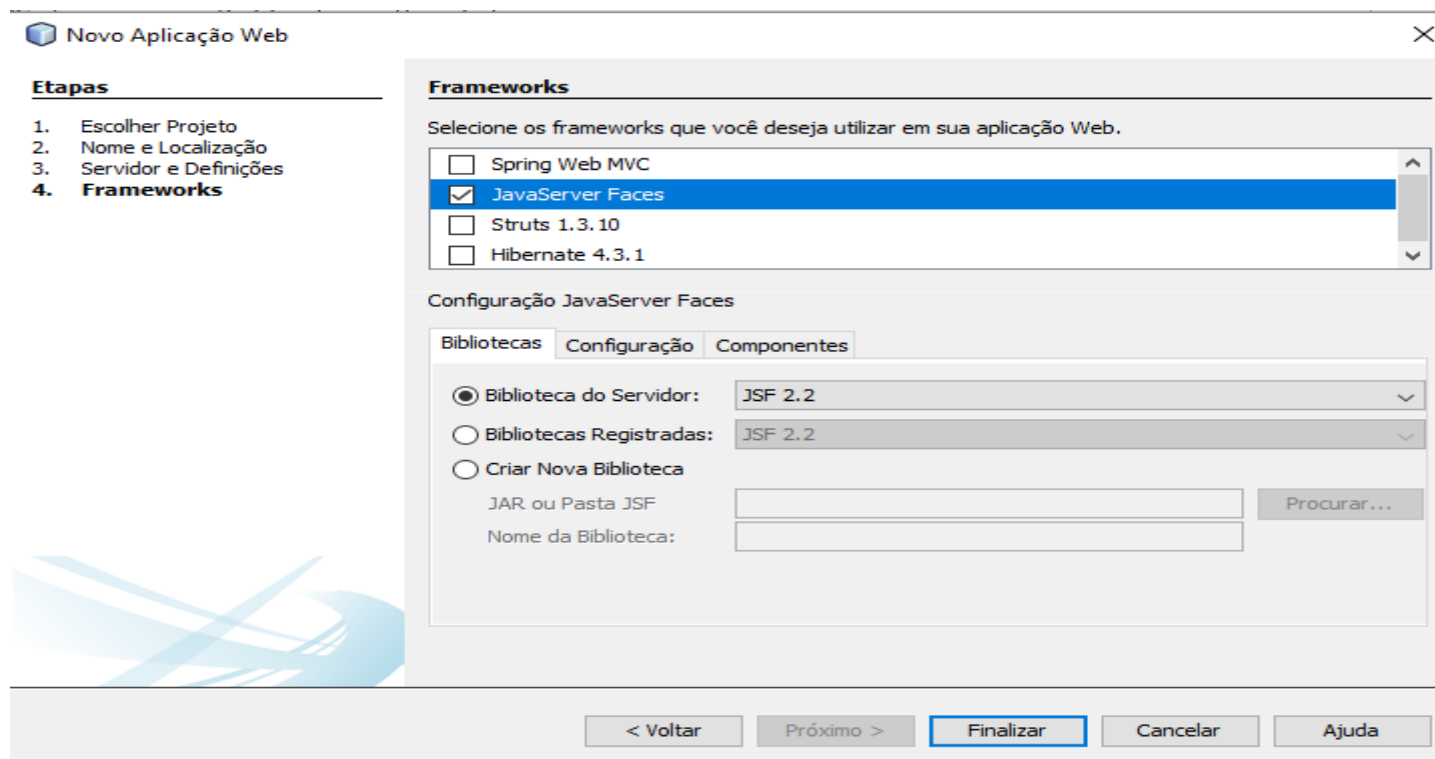
CRIANDO O PROJETO

No NetBeans vá no menu Arquivo e crie um novo projeto do tipo: JavaWeb -> aplicação Web

Selecione o GlassFish como servidor

Na tela de frameworks selecione o Java Server Faces

SELECIONANDO O JSF NO NETBEANS



Novo Aplicação Web [X]

Etapas

1. Escolher Projeto
2. Nome e Localização
3. Servidor e Definições
4. **Frameworks**

Frameworks

Selecione os frameworks que você deseja utilizar em sua aplicação Web.

- ☐ Spring Web MVC
- ☒ **JavaServer Faces**
- ☐ Struts 1.3.10
- ☐ Hibernate 4.3.1

Configuração JavaServer Faces

Bibliotecas | Configuração | Componentes

☒ Biblioteca do Servidor: JSF 2.2

☐ Bibliotecas Registradas: JSF 2.2

☐ Criar Nova Biblioteca

JAR ou Pasta JSF:

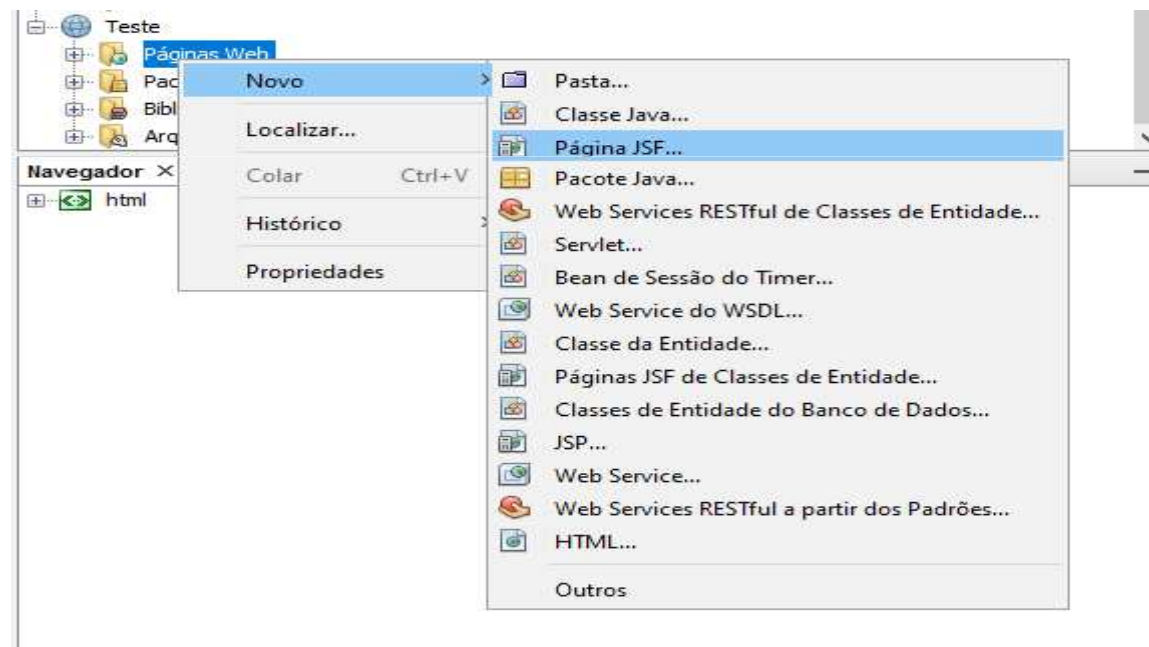
Nome da Biblioteca:

Procurar...

< Voltar Próximo > **Finalizar** Cancelar Ajuda

CRIANDO O PROJETO

Agora, podemos criar nossa primeira página



EXEMPLO 1

```
<?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://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    Hello from Facelets
  </h:body>
</html>
```

**Crie a página
Exemplo1.xhtml**

TAGS

TAGLIBS

As tags que representam os componentes do JSF estão em duas bibliotecas de tags: a **core** e a **html**

A **html** contém os componentes necessários para montarmos nossa tela gerando o HTML adequado

A segunda linha indica que estamos habilitando através do prefixo h as TAGs do JSF que renderizam HTML

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
```

TAGLIBS

As TAGs **core** possuem diversos componentes não visuais, como tratadores de eventos ou validadores

```
xmlns:f="http://java.sun.com/jsf/core"
```

JSF: USANDO H:

Usamos:

h:body no lugar de body

h:form no lugar de form

h:inputText no lugar de input type="text"

Para escrever um texto podemos usar:

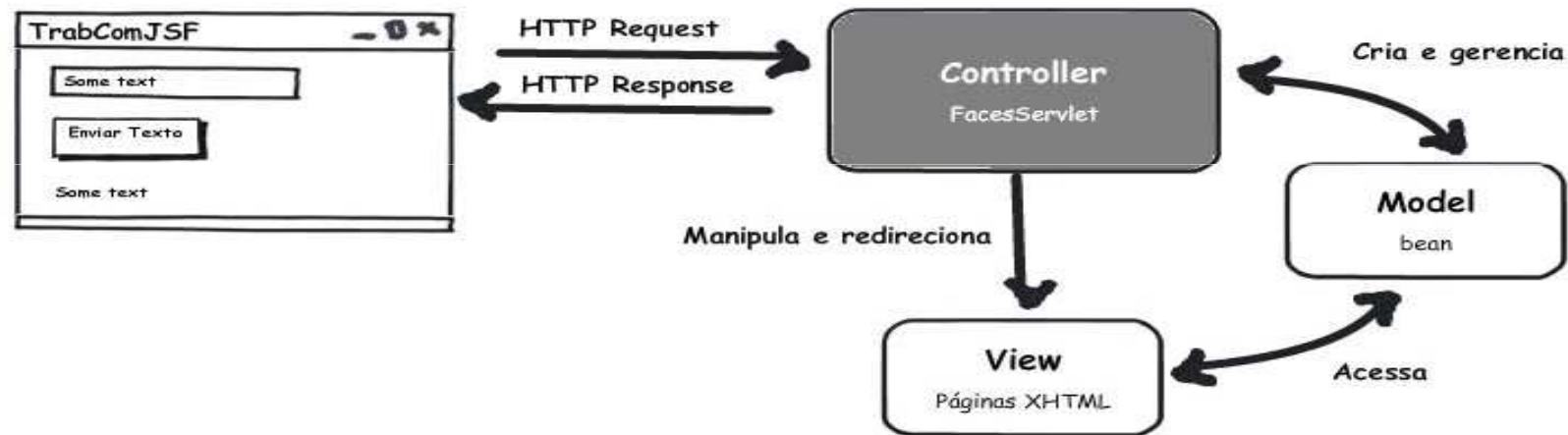
```
<h:body>
```

```
    <h:outputText value ="Olá Mundo!" /> <br />
```

```
</h:body>
```

**Analise e Execute a
página
Exemplo2.xhtml**

COMO FUNCIONA A EXECUÇÃO



Fonte: <http://www.edsongoncalves.com.br/tag/jsf-2-0/>

CICLO DE VIDA

JSF: VANTAGENS DO CICLO DE VIDA

O ciclo de vida permite:

Manter o **controle de estado** dos componentes de interface

Alinhar **ouvintes de eventos** com seus respectivos eventos

Controle de **navegação** entre páginas, que deve ser realizado pelo servidor

Permitir que **validações e conversões** sejam realizadas no lado do servidor

A ÁRVORE DE COMPONENTES

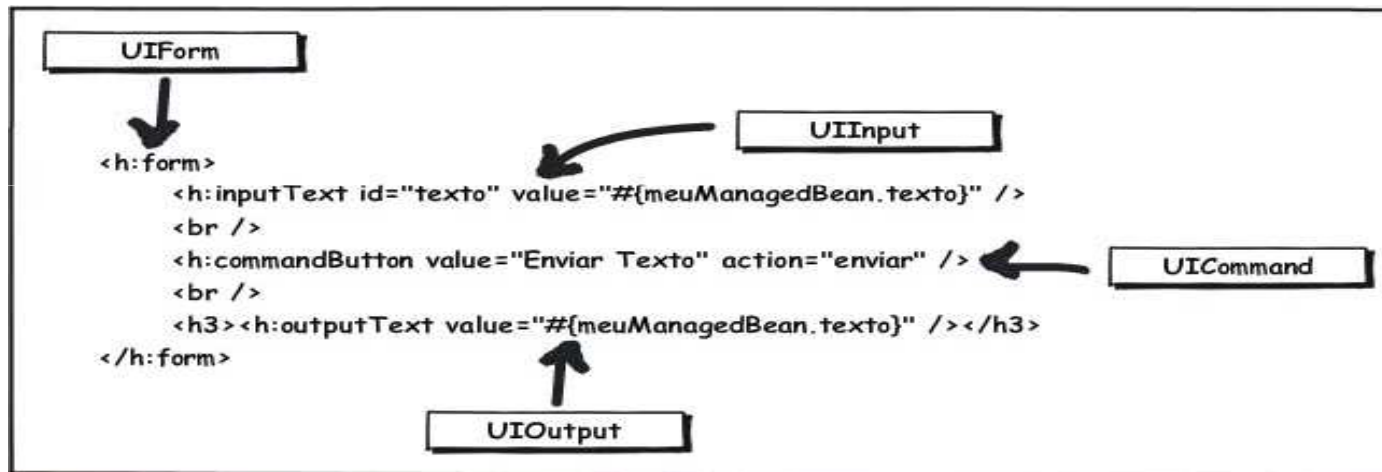
No JSF ele cria a tela lendo o arquivo xhtml

Com base nesse arquivo ele monta a árvore de componentes

Após montar a árvore ela é passada ao renderizador

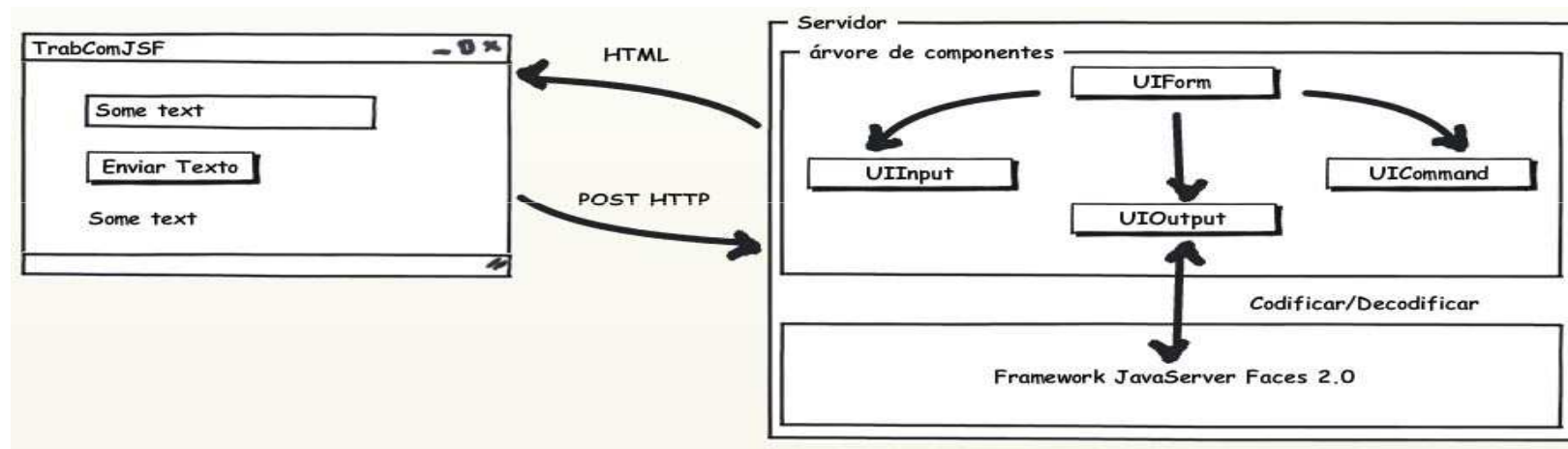
O JSF guarda na memória a árvore usada para gerar a tela através das requisições – JSF é considerado um framework stateful

A ÁRVORE DE COMPONENTES



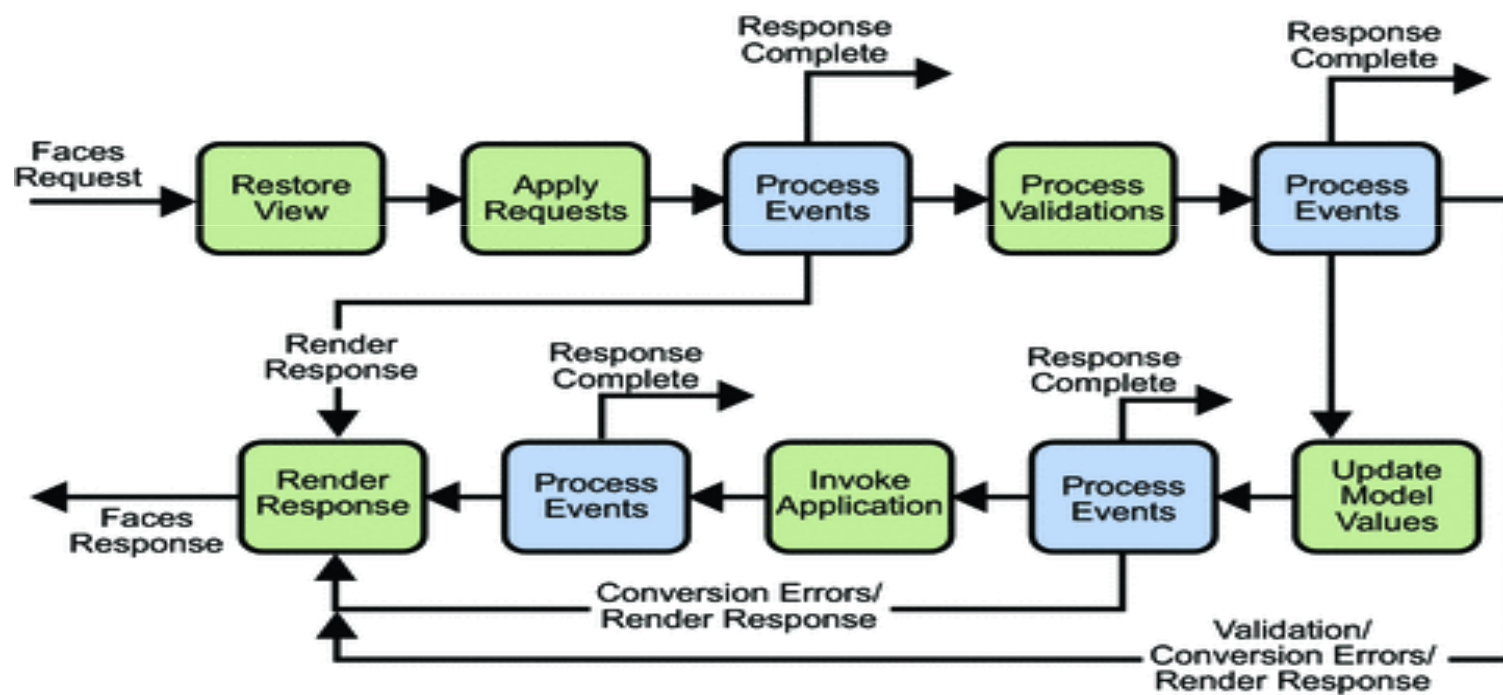
Fonte: <http://www.edsongoncalves.com.br/tag/jsf-2-0/>

A ÁRVORE DE COMPONENTES



Fonte: <http://www.edsongoncalves.com.br/tag/jsf-2-0/>

CICLO DE VIDA



CICLO DE VIDA: FASE 1

Restore View – Restaurar Apresentação:

Esta fase inicia o processamento da requisição do ciclo de vida por meio da construção da árvore de componentes do JSF

Se a árvore não existe ele constrói se já existe ele restaura a árvore

Cada árvore de componentes possui um identificador único durante todo o aplicativo

CICLO DE VIDA: FASE 2

Apply Request Values – Aplicar Valores da Requisição na Árvore de Componentes:

JSF busca os valores informados pelo usuário e coloca nos respectivos componentes

CICLO DE VIDA: FASE 3

Validate – Converter e Validar:

Depois do valor de cada componente ser atualizado, na fase de processo de validações, os componentes serão validados

Serão executadas as validações definidas pelo servidor em cada componente

Se não existem valores inválidos, o ciclo segue para a Fase 4

Caso contrário exibe mensagem de erro e o componente é marcado como inválido. Neste caso, o ciclo segue para a Fase 6

CICLO DE VIDA: FASE 4

Update Model – Atualizar Modelo:

Coloca as informações válidas dentro do modelo

Ao concluir essa fase o modelo está com os valores corretos

JSF entrega os objetos Java com seus tipos corretos e valores validados

CICLO DE VIDA: FASE 5

Invoke Application – Invocar Ação da Aplicação:

Nessa fase acontece a lógica da aplicação

Recomenda-se que as validações de negócio fiquem nesta fase

CICLO DE VIDA: FASE 6

Render Response – Renderizar a Resposta:

Esta é a fase final, onde é renderizada a página

Se for a primeira requisição da página os componentes são acrescentados à apresentação neste momento

Caso contrário, os componentes já foram incluídos e são analisadas mensagens de conversão ou erros de validação

Se ocorre algum erro de conversão ou validação as fases 4 e 5 não são executadas, o código é direccionado para essa fase, que mostra novamente o formulário para o usuário para que ele possa fazer as correções

FORMULÁRIOS

CRIANDO UM FORMULÁRIO

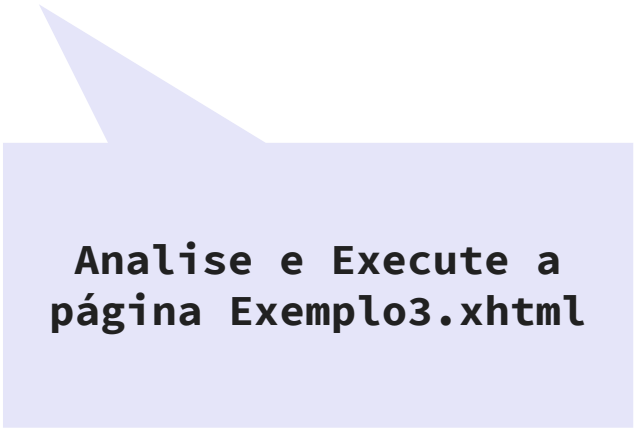
```
<h:form>
```

```
  <h:outputLabel for="nome" value="Digite seu nome:"/>
```

```
  <h:inputText id="nome" />
```

```
  <h:commandButton value="Ok" />
```

```
</h:form>
```



**Analise e Execute a
página Exemplo3.xhtml**

EXERCÍCIO 2

JSF E POJOS

POJO: PESSOA

```
public class Pessoa{  
    private String nome;  
  
    public Pessoa(){}  
    public Pessoa(String nome){  
        this.nome = nome;  
    }  
    public void setNome(String nome){ this.nome = nome; }  
    public String getNome(){return nome;}  
    //outros métodos  
}
```

Pessoa
- nome : String
+ toString() : String + imprime() : void

POJO X MANAGEDBEANS

O POJO, classes de modelo, interage com os componentes do JSF e são chamados de **Managed Beans**

Os dados da tela são passados para a classe usando os Managed Beans

Para indicar que uma classe é um Managed Beans usamos a anotação **@ManagedBean**

CLASSE PESSOA

```
package classes;
import javax.faces.bean.ManagedBean;
@ManagedBean
public class Pessoa {
    private String nome;
    public Pessoa(){}
    public String getNome() {    return nome;    }
    public void setNome(String nome) {    this.nome = nome;}
    @Override
    public String toString() {
        return "Pessoa{" + "nome=" + nome + '}';
    }
    public void imprime(){ System.out.println("Nome:"+nome);}
}
```

Localizar a
classe Pessoa
dentro do
projeto

ACESSANDO MANAGED BEANS

**Crie a página
Exemplo4.xhtml**

Aqui chamamos o
método getNome()

```
<h:body>  
  <h:outputText value="#{pessoa.nome}" />  
  <h:form>  
    <h:outputLabel for="nome" value="Digite seu nome:" />  
    <h:inputText id="nome" value="#{pessoa.nome}" />  
    <h:commandButton value="Ok" action="#{pessoa.imprime()}" />  
  </h:form>  
</h:body>
```

Aqui chamamos o
método setNome()

Aqui chamamos o
método imprime()

Obs.: para chamar os métodos usamos a sintaxe
#{managedbean.método}

EXERCÍCIOS 3 E 4

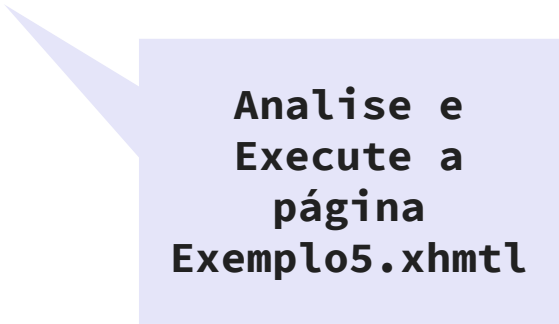
VALIDANDO CAMPOS

RENDERED

Como podemos mostrar um campo somente se ele estiver preenchido?

Para mostrar um `h:outputText` na tela apenas se a informação estiver preenchida é necessário usar o atributo `rendered`:

```
<h:outputText value=" #{pessoa.nome}" rendered="#{not  
empty pessoa.nome}"/>
```



**Analise e
Execute a
página
Exemplo5.xhtml**

NOMEANDO UM BEAN

MANAGED BEANS COM NOME

Você percebeu que o Managed Bean usado embora represente a classe Pessoa ele é usado como pessoa no código

Podemos dar outro nome para ele usando o atributo name da anotação @ManagedBean:

```
package classes;  
import javax.faces.bean.ManagedBean;  
@ManagedBean (name="PessoaBean")  
public class Pessoa {  
}
```

USANDO MANAGED BEAN COM NOME

Analise e Execute
a página
Exemplo6.xhtml

```
<h:form>
    <h:outputText value="#{PessoaBean.nome}" />
    <h:outputText value="Oi #{PessoaBean.nome}"
        rendered="#{not empty PessoaBean.nome}"/>
    <h:outputLabel for="nome" value="Digite seu nome:"/>
    <h:inputText id="nome" value="#{PessoaBean.nome}"/>
    <h:commandButton value="Ok"
        action="#{PessoaBean.imprime()}" />
</h:form>
```

GRID

GRADE DE COMPONENTES

Se você analisar a aparência das páginas produzidas até o momento verá que todos os componentes são dispostos em linha

Para organizá-los de forma alinhada podemos usar grades – componente `<h: panelGrid columns="x">`, onde x é o número de colunas que vamos organizar as informações da página

USANDO GRID

Os componentes à medida que são adicionados são alocados em colunas

Execute o Exemplo7.xhtml

```
<h:form>
  <h:panelGrid columns="2">
    <h:outputLabel for="nome" value="Digite seu nome:"/>
    <h:inputText id="nome" value="#{pessoa.nome}"/>
    <h:commandButton value="Ok"
                      action="#{pessoa.imprime()}" />
    <h:outputText value="Oi #{pessoa.nome}"
                  rendered="#{not empty pessoa.nome}"/>
  </h:panelGrid>
</h:form>
```

EXERCÍCIO 5

CRIANDO CONTROLLER

ACESSANDO UM CONTROLLER

Ao invés de se comunicar diretamente com a camada Model – classe Pessoa – podemos criar uma camada intermediária que fará o controle, através da classe PessoaBean

ACESSANDO UM CONTROLLER

```
package classes;  
import java.util.LinkedList;  
import javax.faces.bean.ManagedBean;
```

```
@ManagedBean  
public class PessoaBean {  
    private Pessoa pessoa = new Pessoa();  
    public Pessoa getPessoa() {  
        return pessoa;  
    }  
    public void setPessoa(Pessoa pessoa) {  
        this.pessoa = pessoa;  
    }  
}
```

Pessoa contém os atributos
e PessoaBean contém uma
composição para Pessoa

Análise a classe Pessoa

Recupera em PessoaBean o atributo pessoa (com método get) e altera o atributo nome do objeto

Execute o Exemplo8.xhtml

```
<h:form>
  <h:panelGrid columns="2">
    <h:outputLabel for="nome" value="Digite seu nome:"/>
    <h:inputText id="nome" value="#{pessoaBean.pessoa.nome}"/>
    <h:commandButton value="Ok"
      action="#{pessoaBean.pessoa.imprime()}" />

    <h:outputText value="Oi #{pessoaBean.pessoa.nome}"
      rendered="#{not empty pessoaBean.pessoa.nome}" />
  </h:panelGrid>
</h:form>
```

EXERCÍCIOS 6 A 11