

# JavaServer Faces (JSF)



# Especificação/IDE/Implementação

- Esse curso foi preparado em 03/2015 usando a seguinte especificação, IDE e implementação
- Especificação
  - JavaServer Faces 2.2 (05/2013, JEE 7)
  - JavaServer Faces 2.1 (11/2010)
  - JavaServer Faces 2.0 (07/2009, JEE 6)
- IDE
  - JDK 8u40
  - NetBeans 8.0.2 na distribuição Java EE
- Implementação
  - GlassFish 4.1 (vem no NetBeans)

# Agenda

- O que é JSF?
- JavaBeans
- Navegação
- Tags JSF
- Conversão de dados
- Validação de dados
- JSF com AJAX
- Templates

# O que é JSF?

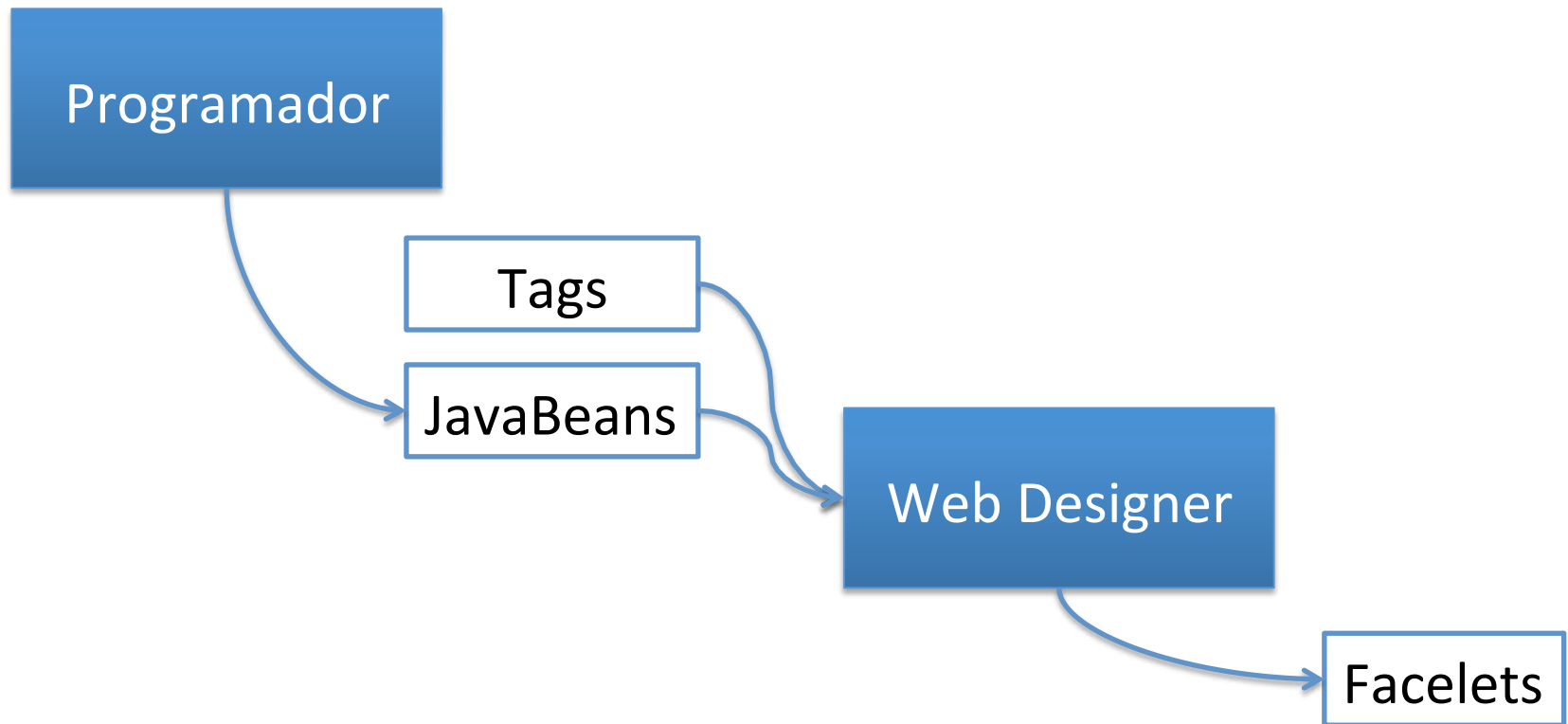
- Um framework para a camada de apresentação
- Visa promover uma separação clara entre comportamento e apresentação



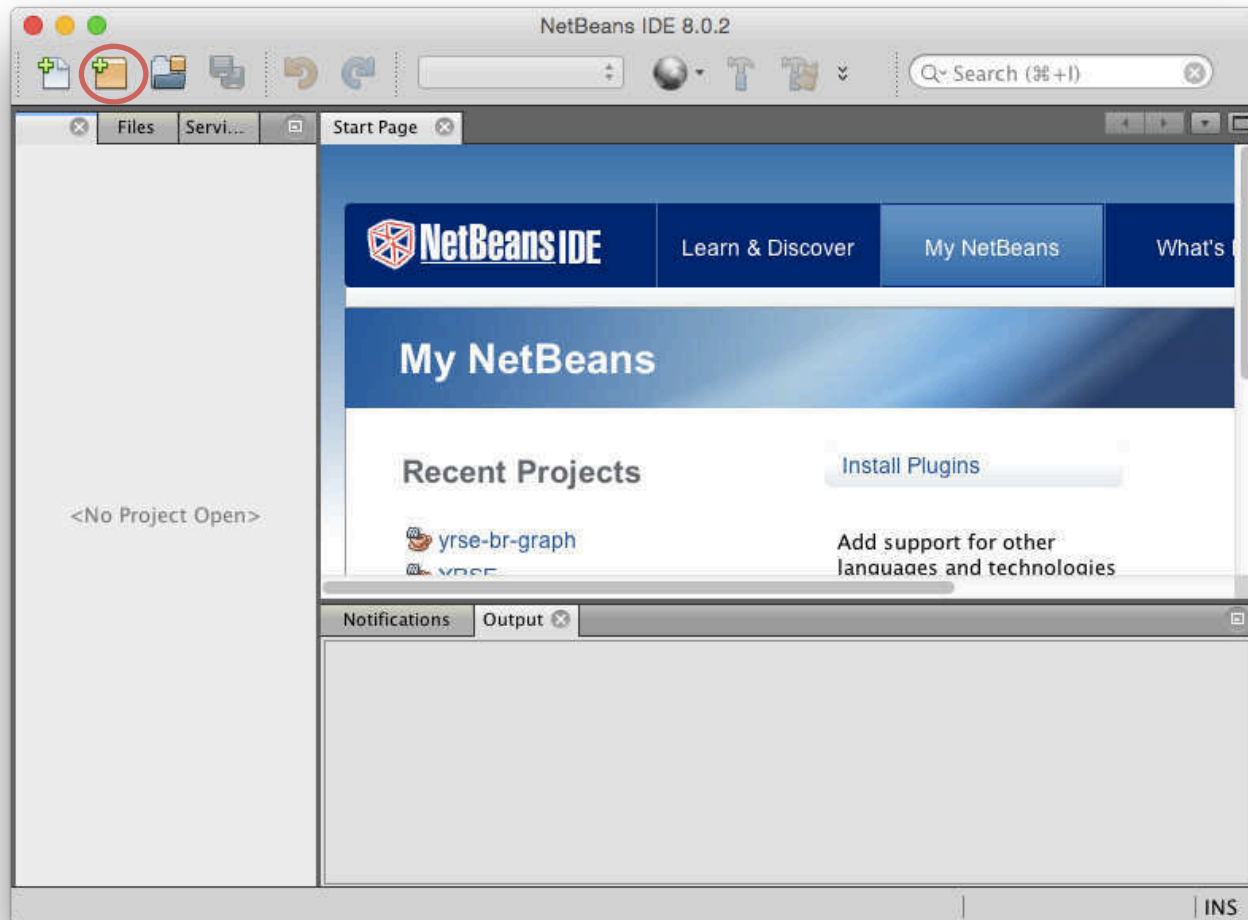
# Principais componentes

- Facelets (páginas XHTML)
  - Representam a interface do sistema
  - São transformados em HTML
- Classes Java (JavaBeans)
  - São automaticamente instanciados
  - Armazenam os dados usados para ler e escrever os campos das páginas
- Tags
  - Permitem a escrita de Facelets de forma declarativa
- Metadados
  - Configuram a aplicação
  - Podem ser tanto anotações quanto XML

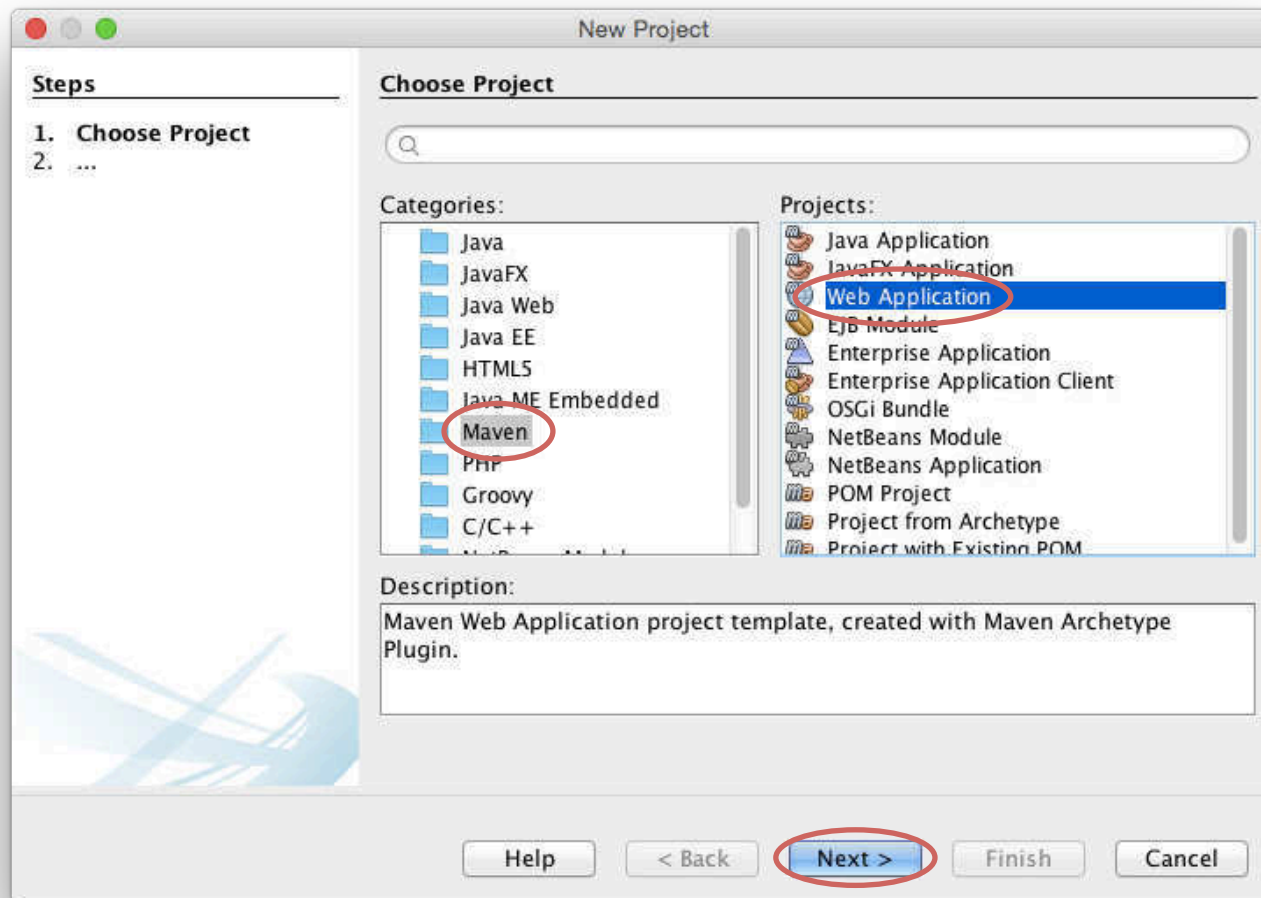
# Principais componentes



# Criando um projeto JSF



# Criando um projeto JSF





# Criando um projeto JSF

New Web Application

**Steps**

1. Choose Project
2. **Name and Location**
3. Settings

**Name and Location**

Project Name:

Project Location:

Project Folder:

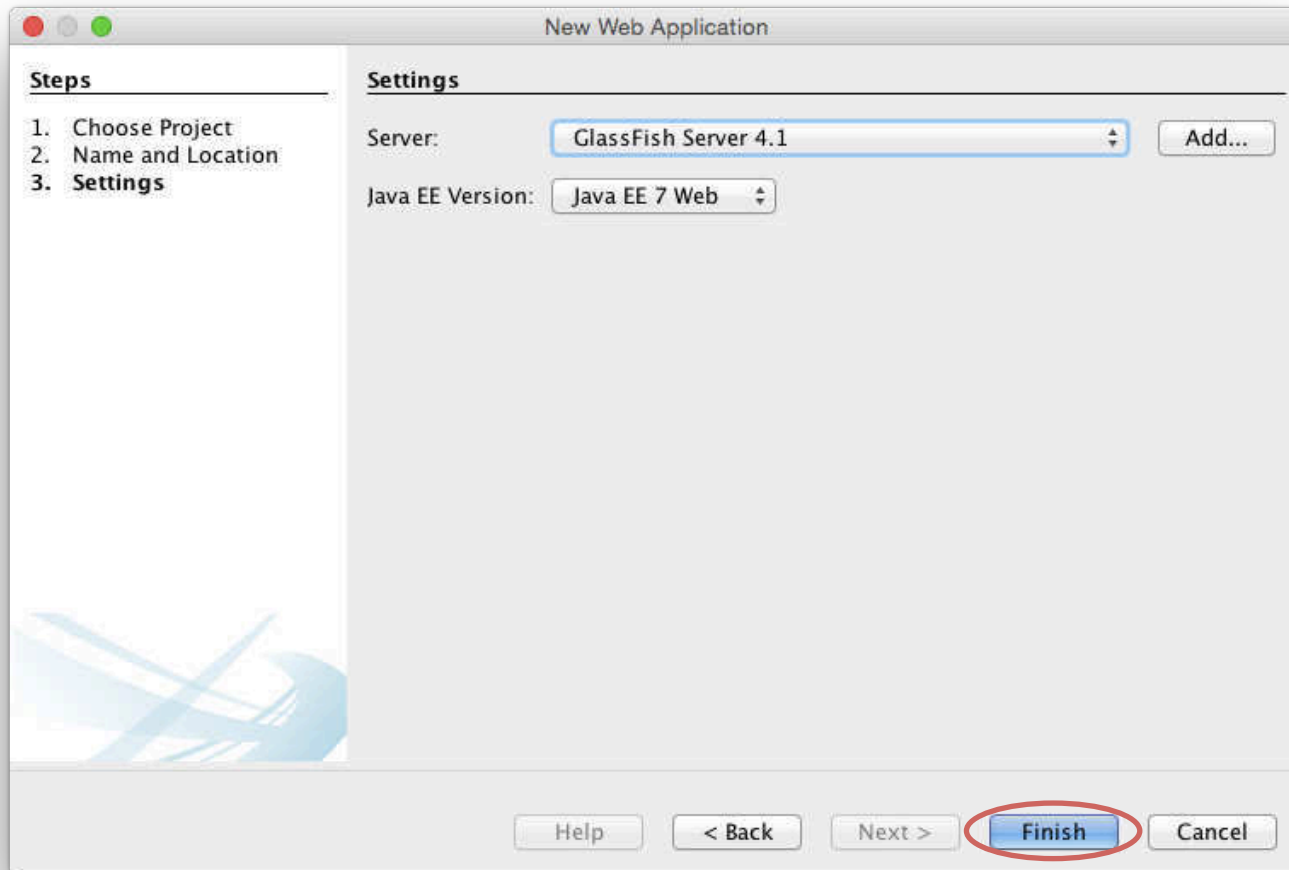
Artifact Id:

Group Id:

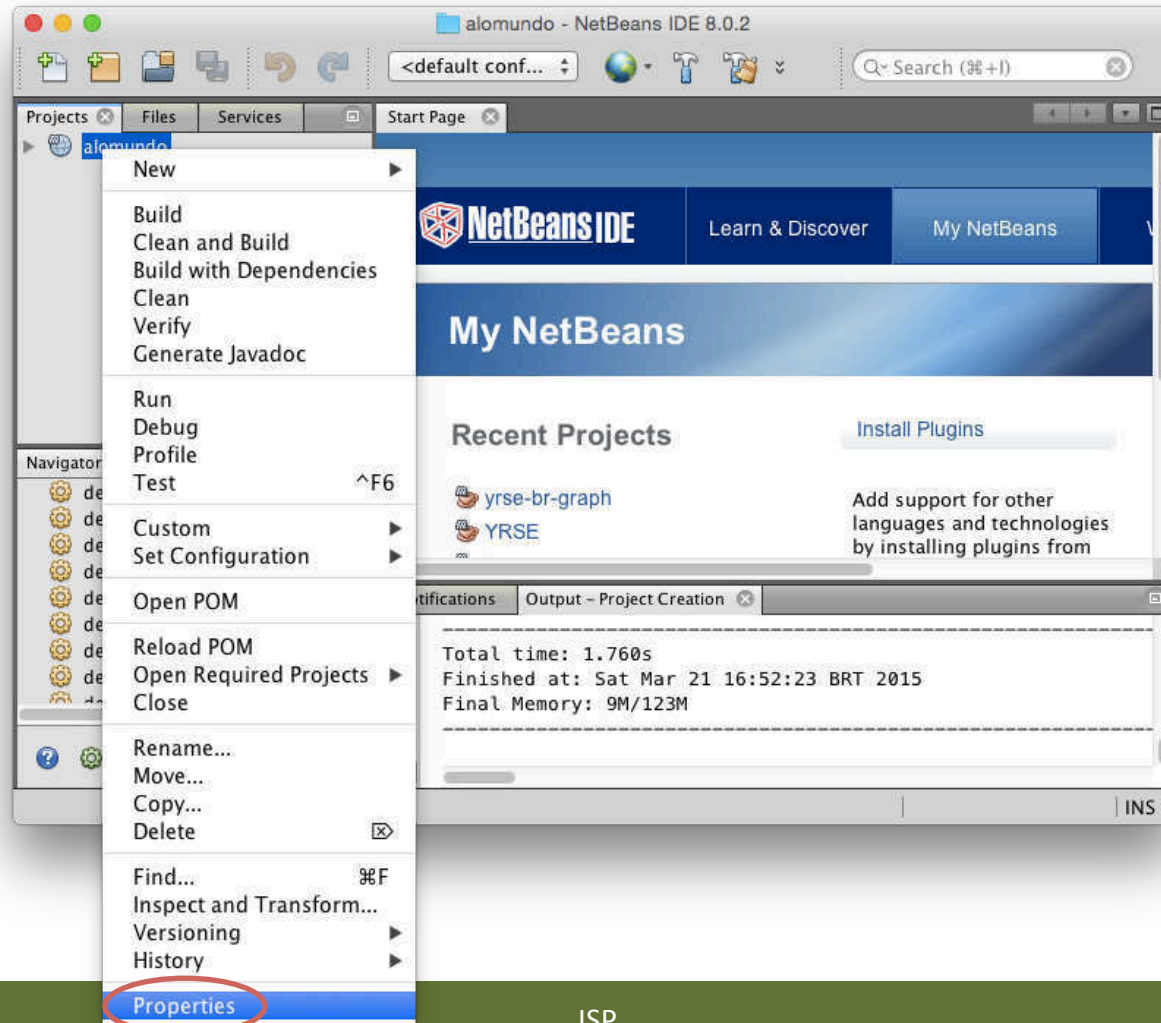
Version:

Package:  (Optional)

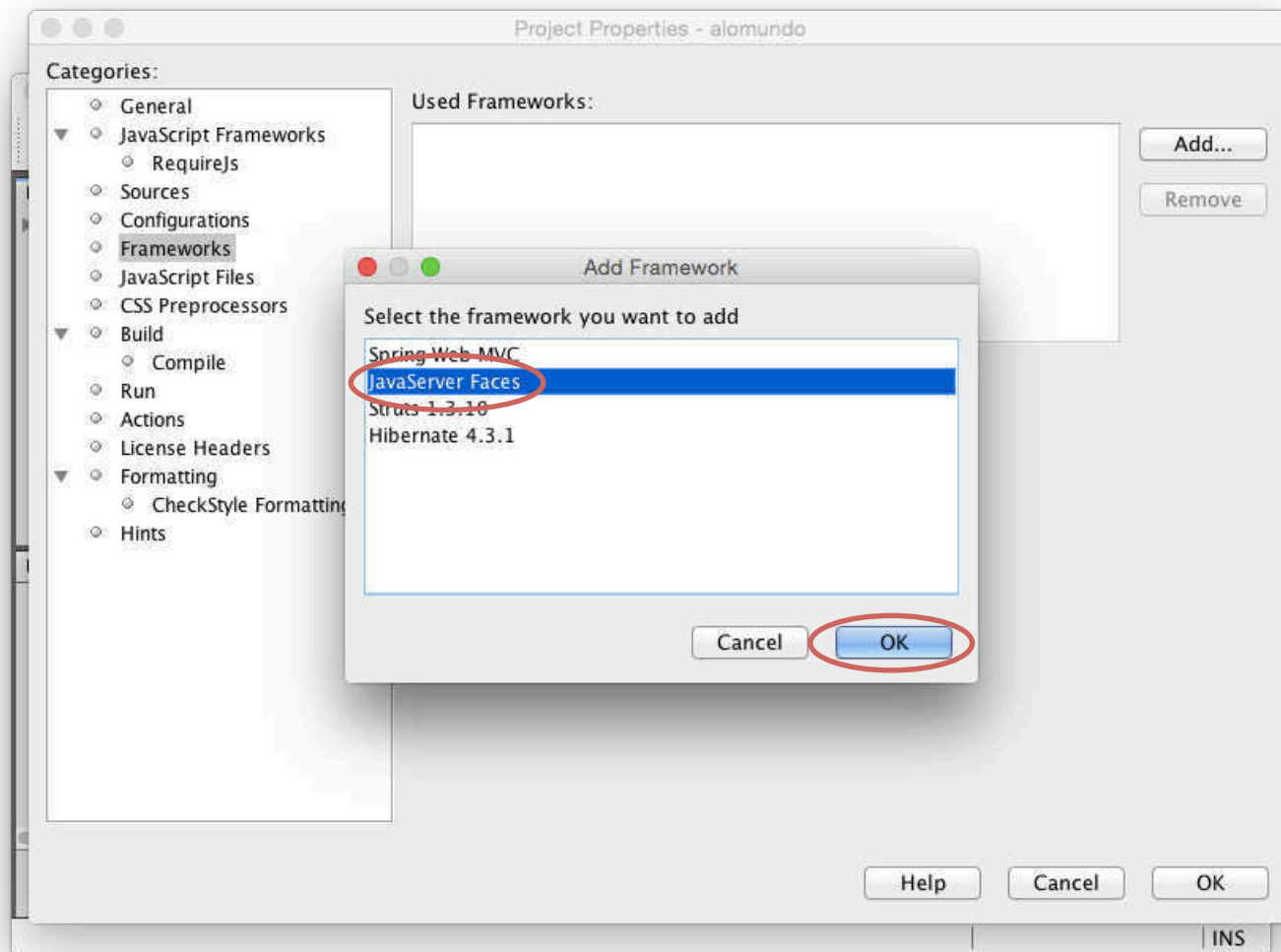
# Criando um projeto JSF



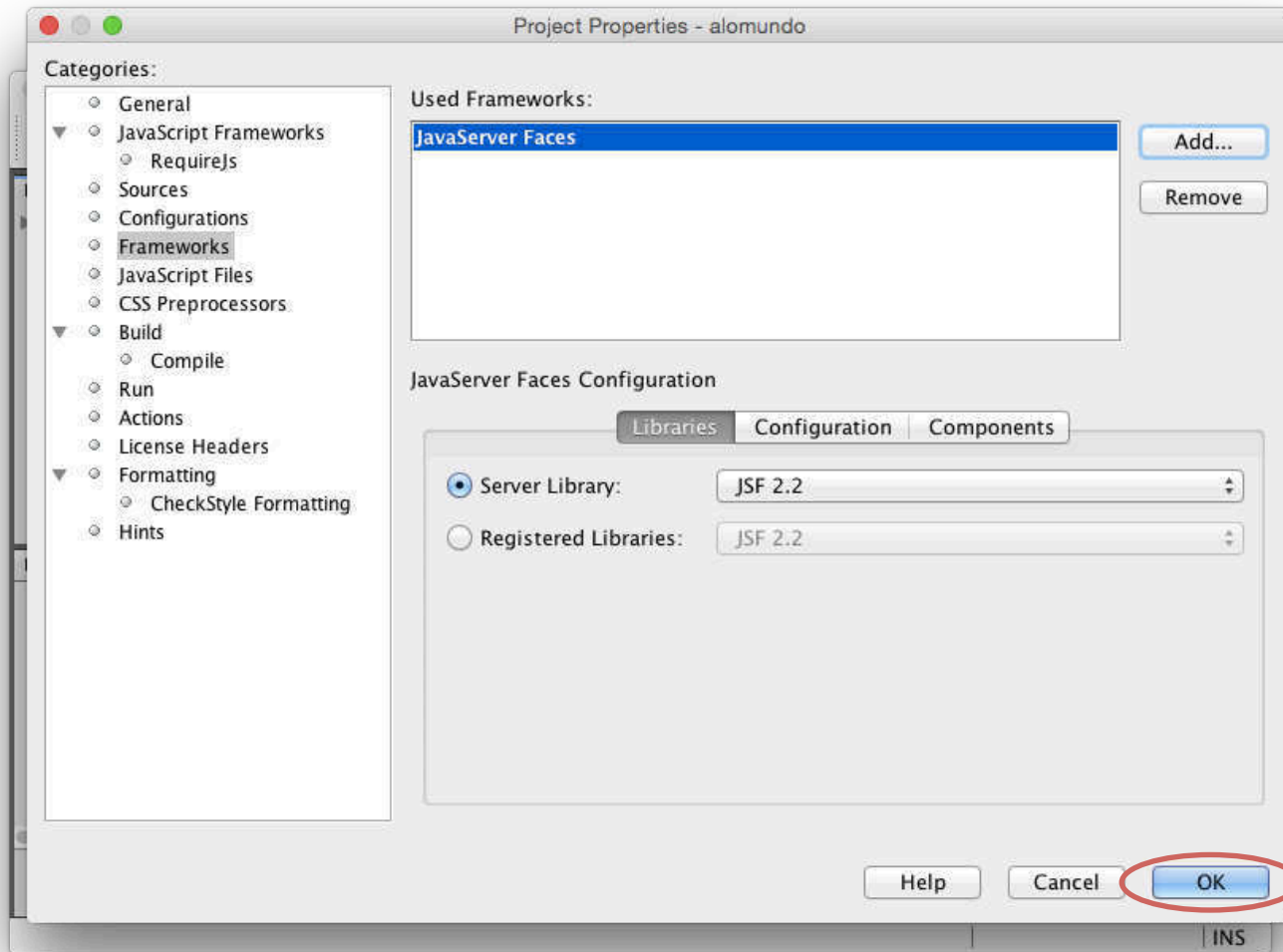
# Adicionando o Framework JSF em um projeto existente



# Adicionando o Framework JSF em um projeto existente



# Adicionando o Framework JSF em um projeto existente



# O que aconteceu?

- Foram basicamente criados dois arquivos
  - Descritor (WEB-INF/web.xml)
  - Facelet (index.xhtml)

# Descritor criado

```

...
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</
servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
...
<welcome-file-list>
  <welcome-file>faces/index.xhtml</welcome-file>
</welcome-file-list>
...

```

# Descritor criado

...

```

<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</
servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>

```

Implícito  
em JSF  
2.0 ou  
superior

...

```

<welcome-file-list>
  <welcome-file>faces/index.xhtml</welcome-file>
</welcome-file-list>

```

...



# Facelet criado

```

<?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>
  
```

# Página gerada ao acessar <http://localhost:8080/alomundo>

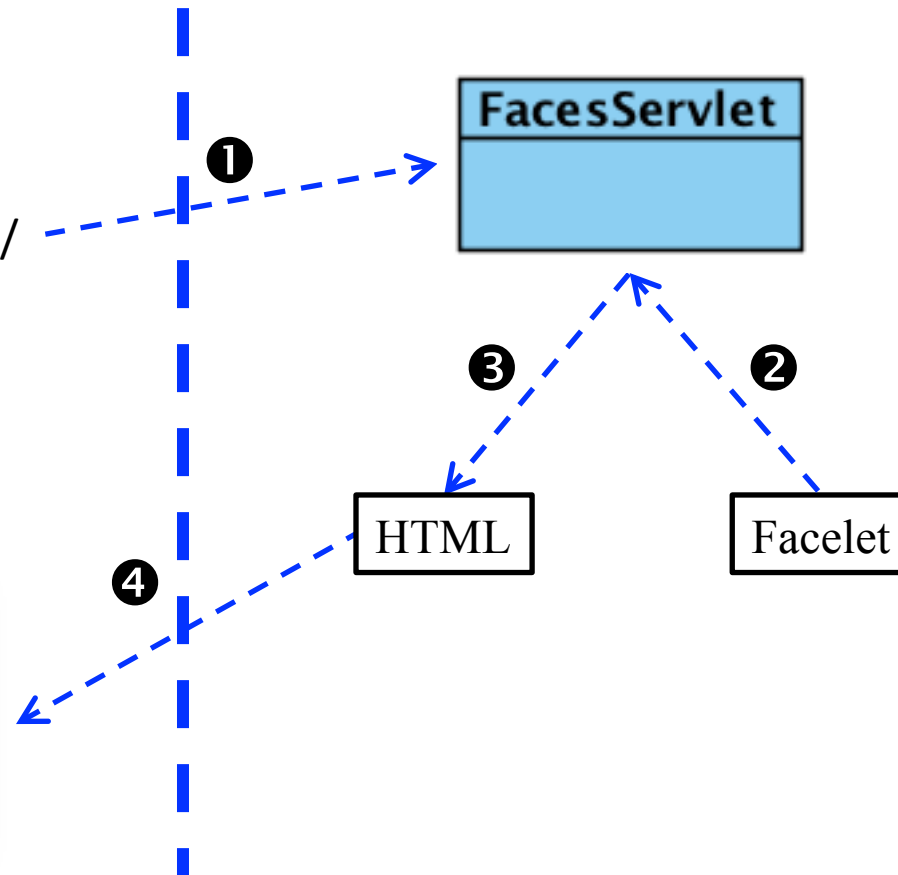
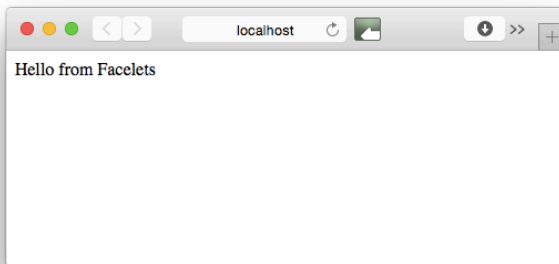
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head id="j_idt2">
    <title>Facelet Title</title>
  </head>
  <body>Hello from Facelets</body>
</html>
```

# O que aconteceu?

Cliente

Servidor

`http://localhost:8080/alomundo/`

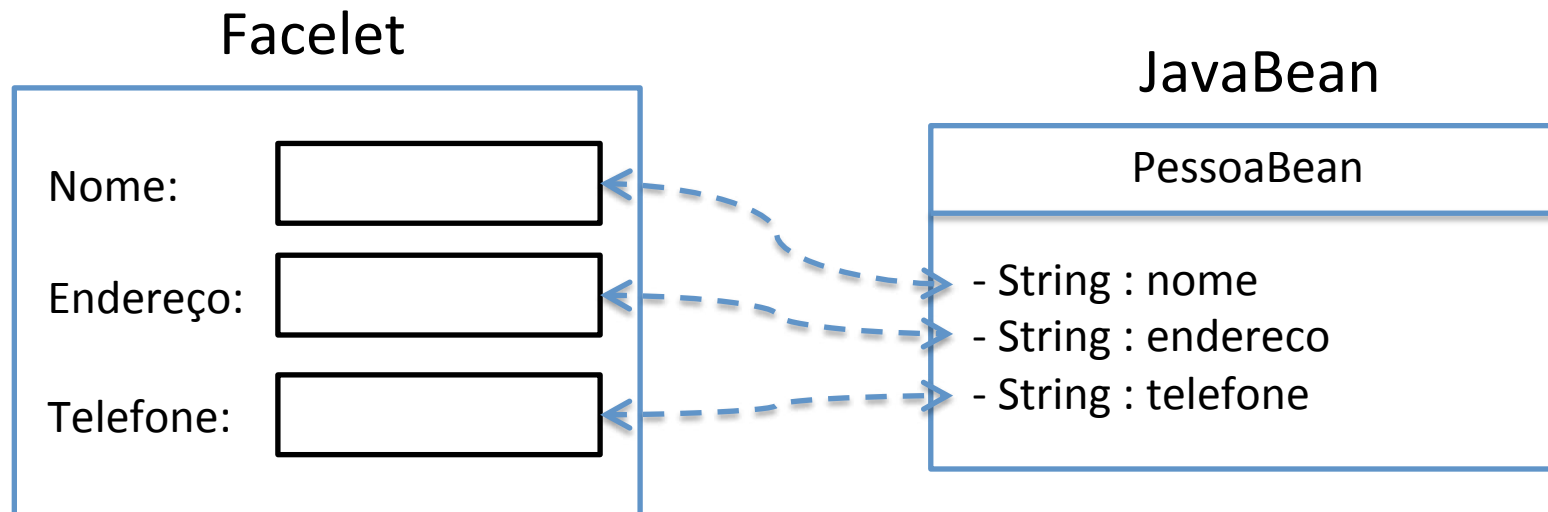


# Model-View-Controller (MVC)

- O JSF implementa o estilo MVC, separando as responsabilidades da aplicação
  - Model: JavaBeans
  - View: Facelets
  - Controller: FacesServlet

# JavaBeans

- São representações OO das entidades manipuladas nos Facelets



# JavaBeans

- Os JavaBeans usados pelo JSF são especiais, pois têm seus ciclos de vida **gerenciados pelo container**
  - Também são chamados de **Managed Beans** ou **Backing Beans**
- Recebem duas anotações
  - Nome do Bean
  - Escopo do Bean

# Nome do Bean

- Definido como anotação no Bean
  - `@Named(nome)`
- O valor definido nessa anotação pode ser referenciado nos Facelets
  - `#{nome}`
- Se o nome for omitido, é usado por convenção o nome da classe iniciado com minúscula

# Escopo dos Beans

- Os Beans devem ser anotados com a identificação do seu escopo
- Essa anotação indica quando o Bean deve ser instanciado



# Escopo dos Beans (mais importantes)

- **@RequestScoped**
  - Instanciado para cada requisição/resposta
  - Exemplo: cadastro de usuário
- **@SessionScoped**
  - Instanciado uma vez no início de cada sessão
  - Mantido por toda a sessão do usuário
  - Exemplo: carrinho de compras de um usuário
- **@ApplicationScoped**
  - Instanciado uma vez quando o servidor de aplicação inicia
  - Compartilhado por todos os usuários
  - Exemplo: log de usuários on-line em um fórum

# Escopo dos Beans (mais específicos)

- **@ConversationScoped**
  - Instanciado uma vez no início de cada conversação
  - Uma sessão pode conter várias conversações
  - Uma conversação pode ser encerrada pelo usuário
  - Exemplo: uma sessão de chat com o vendedor dentro de uma aplicação de e-commerce
- **@FlowScoped**
  - Instanciado uma vez no início de cada conversação
  - Equivalente ao @ConversationScoped, mas voltado para “Faces Flows”
  - Exemplo: wizard de configuração de um site

# Escopo dos Beans (mais específicos)

- @ViewScoped
  - Instanciado uma vez por página
  - Válido enquanto o usuário permanece na mesma página
  - Exemplo: site com jogos em páginas independentes
- @Dependent
  - O escopo é definido pelo escopo do componente que usa o bean
  - Exemplo: Bean de conversão de moedas, que pode ser usado sozinho ou dentro de um carrinho de compras
- ATENÇÃO: Beans com escopo *Session*, *Application* e *Conversation* devem implementar a interface *Serializable*

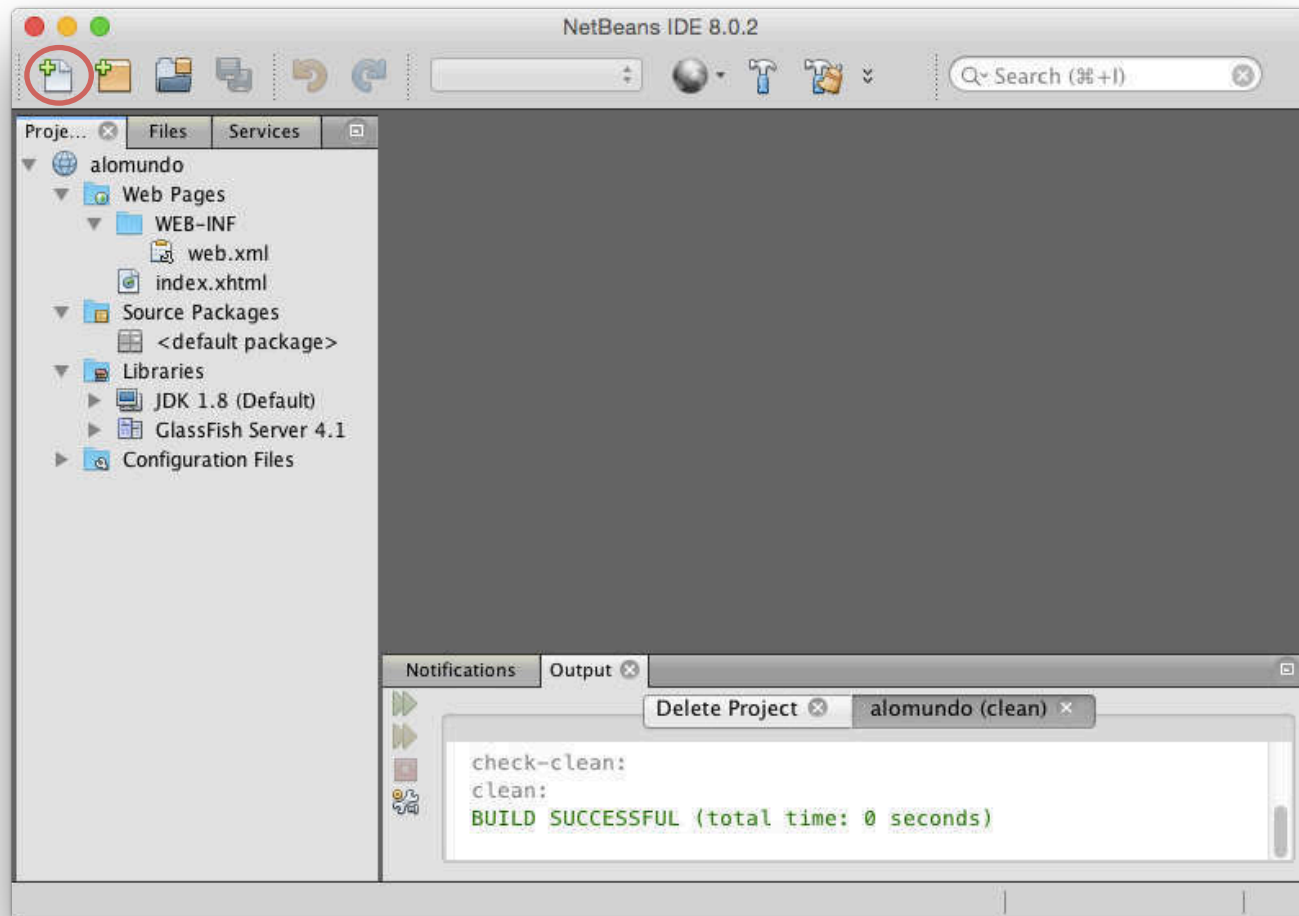
# Cancelando uma conversação

- A qualquer momento o Managed Bean pode cancelar uma conversação
- Para tal, a classe `javax.enterprise.context.Conversation` deve ser usada no Managed Bean com escopo de conversação
- Isso força a criação de um novo Bean na próxima iteração requisição/resposta

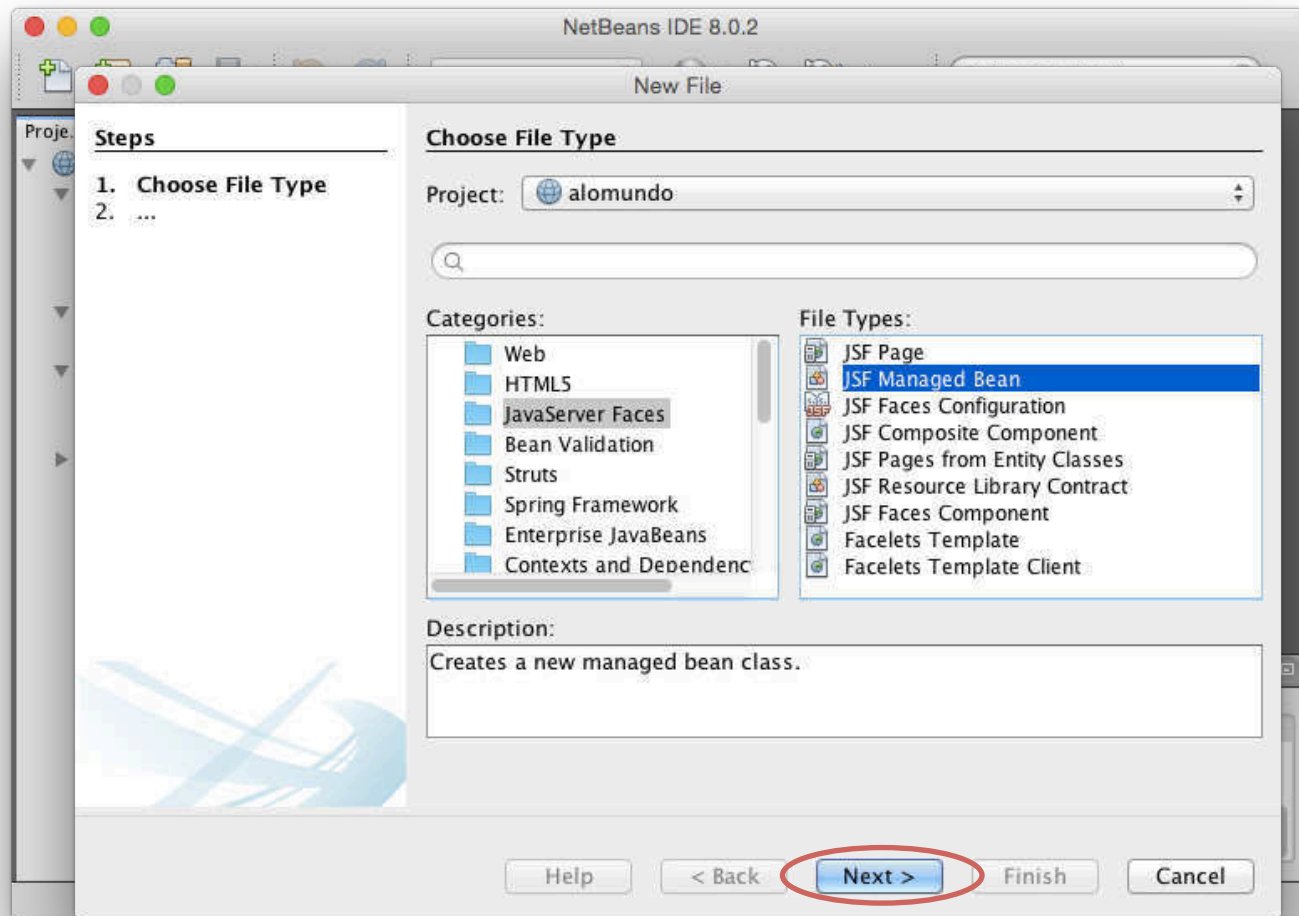
```

@Inject
Conversation conversation;
...
public void finalizaConversacao() {
    conversation.end();
}
  
```

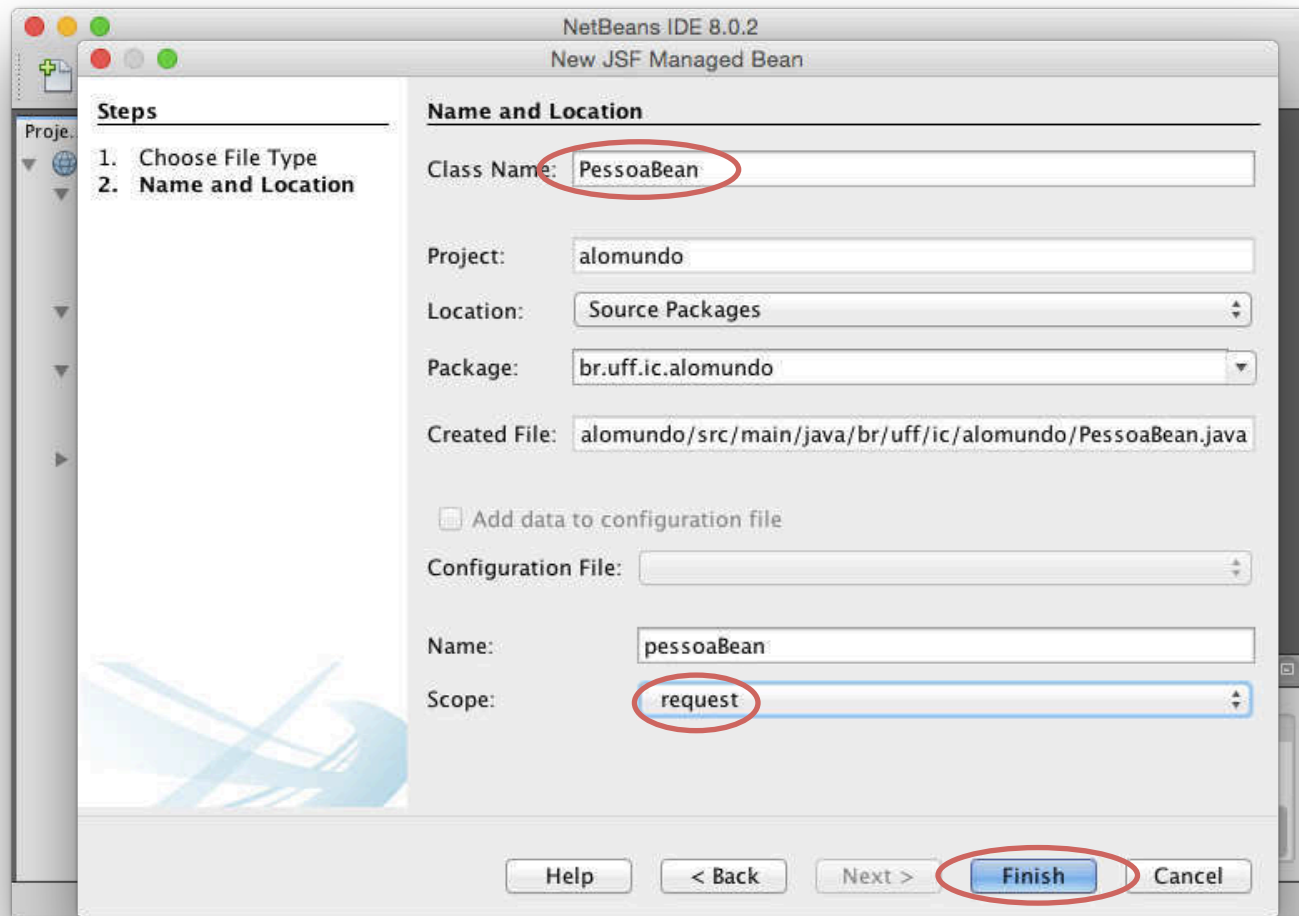
# Criando um Bean



# Criando um Bean



# Criando um Bean



# Bean criado

```

package br.uff.ic.alomundo;

import javax.inject.Named;
import javax.enterprise.context.RequestScoped;

@Named
@RequestScoped
public class PessoaBean {

    public PessoaBean() {
    }

}

```



# Criando propriedade *nome*

```

...
@Named
@RequestScoped
public class PessoaBean {

    private String nome;

    public PessoaBean() {
        nome = "Anônimo";
    }

    public String getNome() {
        return nome;
    }

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

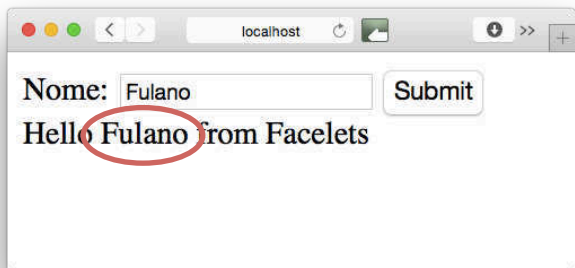
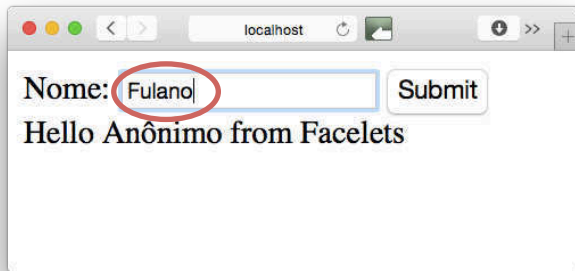
```

# Usando o Bean no Facelet

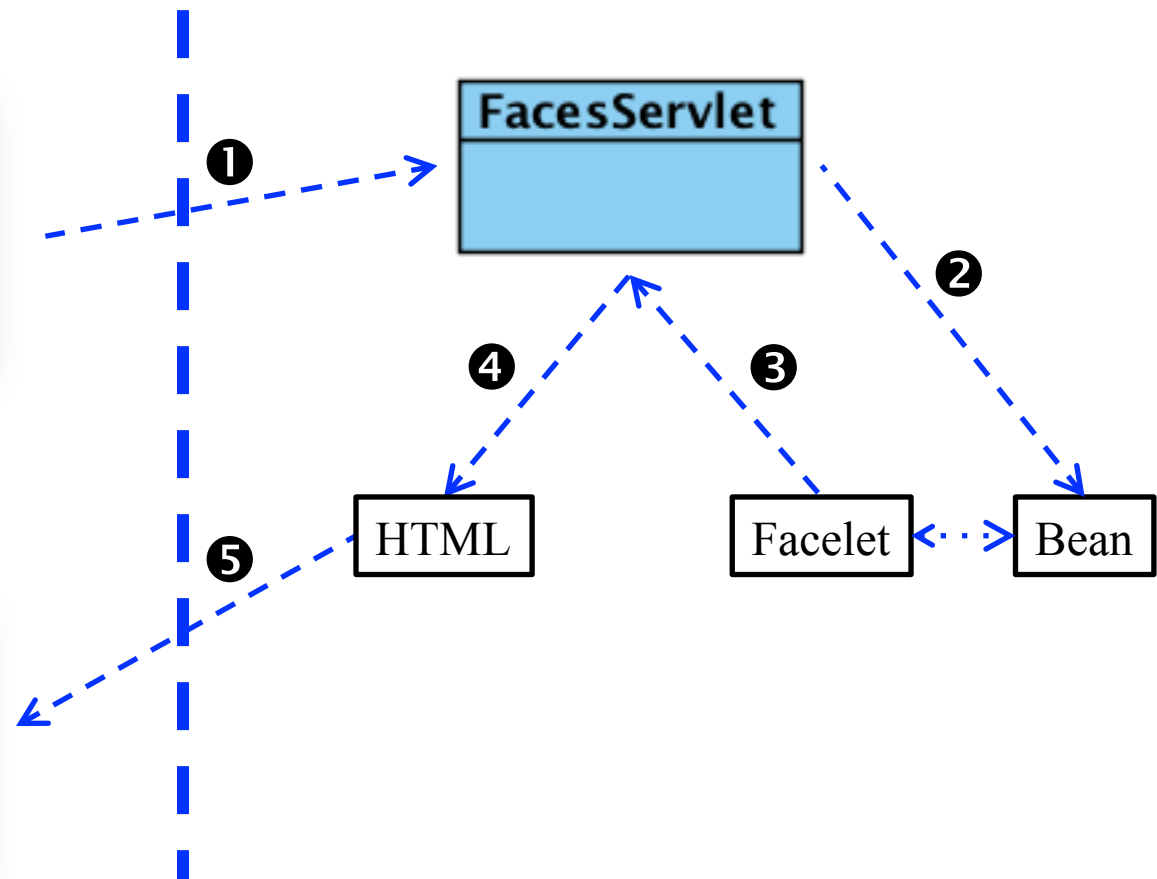
```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>
    <h:form>
      Nome: <h:inputText value="#{pessoaBean.nome}" />
      <h:commandButton value="Submit" />
    </h:form>
    Hello #{pessoaBean.nome} from Facelets
  </h:body>
</html>
```

# O que aconteceu?

Cliente



Servidor



# Navegação

- A navegação entre páginas pode ser feita de duas maneiras
  - Navegação direta: o método acionado retorna a página que deve ser aberta
  - Navegação indireta: o método acionado retorna um rótulo e o XML de configuração vincula o rótulo a uma página

# Navegação direta

- No Facelet

```
<h:commandButton value="Salva"
action="#{agendaBean.cadastra (contatoBean) }">
```

- No Bean

```
public String cadastra (ContatoBean contatoBean) {
    ...
    return "listagem.xhtml";
}
```

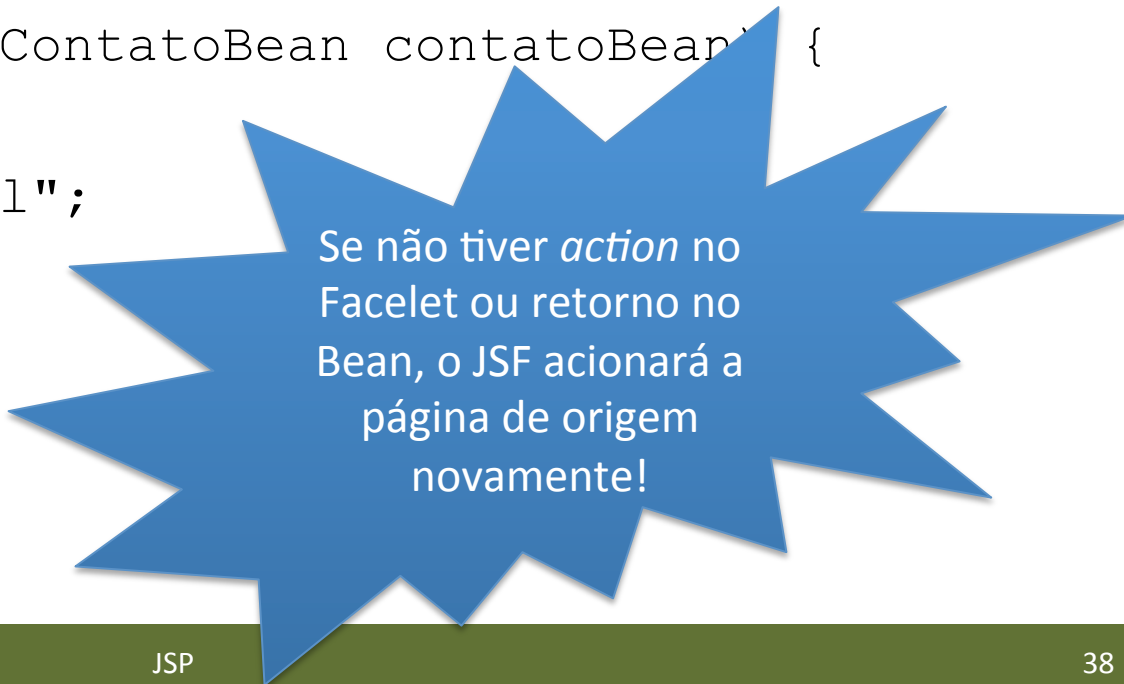
# Navegação direta

- No Facelet

```
<h:commandButton value="Salva"
action="#{agendaBean.cadasttra (contatoBean) }">
```

- No Bean

```
public String cadasttra (ContatoBean contatoBean) {
    ...
    return "listagem.xhtml";
}
```



Se não tiver *action* no Facelet ou retorno no Bean, o JSF acionará a página de origem novamente!

# Navegação indireta

- No Facelet

```
<h:commandButton value="Salva"
action="#{agendaBean.cadasttra (contatoBean) }">
```

- No Bean

```
public String cadastra (ContatoBean contatoBean) {
    ...
    return "cadastrado";
}
```

- No faces-config.xml

```
<navigation-rule>
  <from-view-id>cadastro.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>cadastrado</from-outcome>
    <to-view-id>listagem.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

# Exercício

- Criar uma aplicação usando JSF para somatório, onde o valor inicial e o valor final são informados
- Informar o número de vezes que
  - O usuário acessou o serviço na mesma sessão
  - Todos os usuários acessaram o serviço desde quando o servidor entrou no ar



# Tags JSF

- Até então, vimos a tag *h:inputText* e *h:commandButton* de forma intuitiva
  - Contudo, o JSF disponibiliza diversas coleções de tags, para diferentes propósitos
  - É necessário declarar a coleção de *tags* que será usada e um prefixo para representar a coleção
- ```
<html xmlns:prefixo="URI da coleção de tags">
```

# Tags JSF

## (coleções de tags)

- Interface com usuário
  - Prefixo padrão: h
  - URI: <http://xmlns.jcp.org/jsf/html>
- Templates
  - Prefixo padrão: ui
  - URI: <http://xmlns.jcp.org/jsf/facelets>
- Core
  - Prefixo padrão: f
  - URI: <http://xmlns.jcp.org/jsf/core>
- JSLT (vindas dos JSP)
  - Controle de fluxo
  - Manipulação de strings

# Tags de interface com usuário

- Componente visual
  - Formulário

- Tags

```
<h:form>
```

- Exemplo

```
<h:form>
```

```
...
```

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

# Tags de interface com usuário

- Componente visual
  - Botão
- Tag

```
<h:commandButton>
```

- Exemplo

```
<h:commandButton value="Submit"
action="#{pessoaBean.cadastra}" />
```

# Tags de interface com usuário

- Componente visual

- Link

- Tag

```
<h:commandLink>
```

- Exemplo

```
<h:commandLink value="Leia mais..."
action="leia.xhtml" />
```

# Tags de interface com usuário

- Componente visual
  - Tabela

- Tags

```
<h:dataTable>
```

```
<h:column>
```

```
<f:facet>
```

- Exemplo

```
<h:dataTable value="#{agendaBean.contatos}" var="contato">
```

```
<h:column>
```

```
<f:facet name="header">Nome</f:facet>
```

```
#{contato.nome}
```

```
</h:column>
```

```
<h:column>
```

```
<f:facet name="header">Telefone</f:facet>
```

```
#{contato.telefone}
```

```
</h:column>
```

```
</h:dataTable>
```

# Tags de interface com usuário

- Componente visual
  - Imagem
- Tags

```
<h:graphicImage>
```

- Exemplo

```
<h:graphicImage value="{usuarioBean.foto}" />
```

# Tags de interface com usuário

- Componente visual
  - Entrada de texto
- Tags

```
<h:inputText>
```

```
<h:inputTextarea>
```

```
<h:inputSecret>
```

- Exemplo

```
<h:inputText value="#{usuarioBean.nome}" />
```



# Tags de interface com usuário

- Componente visual

- Caixa de seleção



- Tags

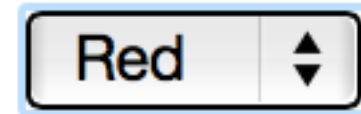
```
<h:selectBooleanCheckbox>
```

- Exemplo

```
<h:selectBooleanCheckbox  
value="#{usuarioBean.ativo}" />
```

# Tags de interface com usuário

- Componente visual
  - Lista de seleção única



- Tags

```
<h:selectOneMenu>
```

- Exemplo

```
<h:selectOneMenu value="#{corBean.opcao}">
  <f:selectItem itemValue="red" itemLabel="Red" />
  <f:selectItem itemValue="blue" itemLabel="Blue" />
  <f:selectItem itemValue="green" itemLabel="Green" />
</h:selectOneMenu>
```

# Tags de interface com usuário

- Componente visual
  - Botão de seleção única



- Tags

```
<h:selectOneRadio>
```

- Exemplo

```
<h:selectOneRadio value="#{corBean.opcao}">
  <f:selectItem itemValue="red" itemLabel="Red" />
  <f:selectItem itemValue="blue" itemLabel="Blue" />
  <f:selectItem itemValue="green" itemLabel="Green" />
</h:selectOneRadio>
```

# Tags de interface com usuário

- Componente visual
  - Lista de seleção única

- Tags

```
<h:selectOneListbox>
```



- Exemplo

```
<h:selectOneListbox value="#{corBean.opcao}">
  <f:selectItem itemValue="red" itemLabel="Red" />
  <f:selectItem itemValue="blue" itemLabel="Blue" />
  <f:selectItem itemValue="green" itemLabel="Green" />
</h:selectOneListbox>
```

# Tags de interface com usuário

- Componente visual
  - Caixa de seleção múltipla



- Tags

```
<h:selectManyCheckbox>
```

- Exemplo

```
<h:selectManyCheckbox value="#{corBean.opcoes}">
  <f:selectItem itemValue="red" itemLabel="Red" />
  <f:selectItem itemValue="blue" itemLabel="Blue" />
  <f:selectItem itemValue="green" itemLabel="Green" />
</h:selectManyCheckbox>
```

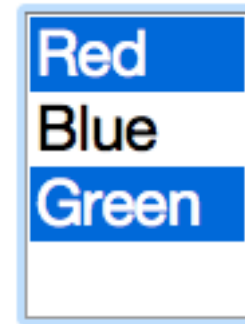
# Tags de interface com usuário

- Componente visual
  - Lista de seleção múltipla

- Tags

```
<h:selectManyMenu>
```

```
<h:selectManyListbox>
```



- Exemplo

```
<h:selectManyMenu value="#{corBean.opcoes}">
```

```
  <f:selectItem itemValue="red" itemLabel="Red" />
```

```
  <f:selectItem itemValue="blue" itemLabel="Blue" />
```

```
  <f:selectItem itemValue="green" itemLabel="Green" />
```

```
</h:selectManyMenu>
```

# Tags de interface com usuário

- Componente visual
  - Mensagens de erro
- Tags

```
<h:messages>
```

- Exemplo

```
<h:messages errorStyle="color:red" />
```

# Exercício

- Faça uma aplicação em JSF de agenda eletrônica
  - Cadastrar nome, sexo, telefone e aniversário, todos como *String* por enquanto
  - Listar em uma tabela todos os registros



# Conversão e Validação de Dados

- Os dados preenchidos nos formulários são sempre String
- Os dados dos *beans* tem tipos definidos (ex.: int, float, etc.) restrições quanto aos seus valores (ex.: formato de um número de telefone)
- Conversão
  - Garante que o **tipo** do campo no formulário está sendo mapeado adequadamente ao tipo do campo no Bean
- Validação
  - Garante que os **valores** do campo respeitam as restrições de formato impostas pela aplicação

# Conversão de dados

- A conversão é feita de forma automática para os tipos principais da linguagem declarados no Bean
- É possível configurar a conversão
- Exemplos

```
<h:inputText value="#{livroBean.preco}">
  <f:convertNumber currencySymbol="R$" type="currency"/>
</h:inputText>
<h:inputText value="#{livroBean.dataPublicacao}">
  <f:convertDateTime pattern="dd/MM/yyyy"/>
</h:inputText>
<h:inputText value="#{livroBean.devolucoes}">
  <f:convertNumber type="percent" />
</h:inputText>
```

# Conversão Customizada

- É possível **customizar a conversão** para um tipo específico de objeto
- Implementar a interface **javax.faces.convert.Converter**
  - Object **getAsObject**(FacesContext ctx, UIComponent component, String value)
  - String **getAsString**(FacesContext ctx, UIComponent component, Object value)
- Anotar a classe com **@FacesConverter**
- Declarar

```
<f:converter converterId="nomeDaClasse" />
```

# Validação de dados

- Serve para assegurar que os dados entrados estão aderentes ao esperado
- Necessitam interação com o servidor
  - Uma alternativa é usar em conjunto com JavaScript
- São feitos por tags pertencentes à coleção Core
  - Usualmente referenciada pelo prefixo **f**

```
<html xmlns:f="http://xmlns.jcp.org/jsf/core" >
```

# Tags de validação de dados (campo obrigatório)

- Obriga que um campo seja preenchido
- Tags

```
<f:validateRequired>
```

- **Exemplo**

```
<h:inputText value="#{UsuarioBean.nome}"
validatorMessage="O campo nome é obrigatório.">
    <f:validateRequired />
</h:inputText>
```

# Tags de validação de dados (tamanho do campo texto)

- Tags

```
<f:validateLength>
```

- Exemplo

```
<h:inputText value="#{usuarioBean.login}"
validatorMessage="Login deve ter de 5 a 8
caracteres">
```

```
    <f:validateLength minimum="5" maximum="8"/>
```

```
</h:inputText>
```

# Tags de validação de dados (número mínimo e máximo)

- Tags

```
<f:validateLongRange>
```

```
<f:validateDoubleRange>
```

- Exemplo

```
<h:inputText value="{usuarioBean.idade}"
validatorMessage="Idade deve ser entre 18 e 70 anos">
    <f:validateLongRange minimum="18" maximum="70"/>
</h:inputText>
```

# Tags de validação de dados (expressão regular)

- Tags

```
<f:validateRegex>
```

## Exemplo

```
<h:inputText value="#{usuarioBean.celular}"
validatorMessage="Celular inválido">
    <f:validateRegex pattern="^(\\+[0-9]{2})?\\
([0-9]{2}\\\\)9?[0-9]{4}\\\\-[0-9]{4}$" />
</h:inputText>
```



# Rápida revisão de regex

Símbolo	Propósito
^	Casa com o início do texto
\$	Casa com o final do texto
()	Define uma sub-expressão
[]	Casa com um dos elementos contidos no colchete
?	Indica que o elemento anterior é opcional
*	Indica que o elemento anterior ocorre zero ou mais vezes
+	Indica que o elemento anterior ocorre uma ou mais vezes
{m,n}	Indica que o elemento anterior ocorre de $n$ a $m$ vezes
a b	Indica que $a$ e $b$ são alternativas (ou exclusivo)
.	Casa com qualquer caractere
\	Caractere de escape

# Validação Customizada

- É possível **customizar a validação** dos valores de um campo
- Implementar a interface **javax.faces.validator.Validator**
  - void **validate**(FacesContext context, UIComponent component, Object value)
- Anotar a classe com **@FacesValidator**
- Declarar

```
<f:validator validatorId="nomeDaClasse"/>
```

# Tags de validação de dados (anotação no Bean)

- As validações também podem ser estabelecidas via anotações no próprio Bean
- Campo não nulo
  - @NotNull
- Valor máximo do campo
  - @DecimalMax
  - @Max
- Valor mínimo do campo
  - @DecimalMin
  - @Min
- Tamanho do campo
  - @Size
- Data no passado ou no futuro
  - @Past
  - @Future
- Expressão regular
  - @Pattern

# Exercício

- Revisite o exercício anterior incluindo as validações apropriadas para cada campo

# Uso de JSF com AJAX

- Para atualizar somente uma parte da página é possível usar a tag *f:ajax* que vem com o JSF
- Basta atribuir *ids* para os componentes que devem ser atualizados e indicar os *ids* na tag *f:ajax*
  - Atributo `execute`: indica quais **componentes devem ser enviados** para o servidor
  - Atributo `render`: indica quais **componentes devem ser atualizados** em resposta

# Uso de JSF com AJAX

- Os atributos *execute* e *render* aceitam alguns valores padrões
  - @none: nenhum componente (*default*)
  - @this: o componente que contém a tag *f:ajax*
  - @form: todos os componentes do form
  - @all: todos os componentes

# Uso de JSF com AJAX

- Exemplo

...

```
<h:commandButton value="Submit"
action="#{carrinhoBean.adicionaItem}" >
    <f:ajax execute="@all" render="total" />
</h:commandButton>
```

...

```
<h:outputText id="total" value="#{comprasBean.total}" />
```

# Exercício

- Revisite o exercício anterior fazendo com que a lista de contatos atualize via AJAX



# Uso de Templates

- Com JSF é possível criar uma página que tenha o *layout* da aplicação
- Todas as demais páginas irão fazer uso desse *layout*

# Uso de Templates (layout.xhtml)

...

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://
xmlns.jcp.org/jsf/html" xmlns:ui="http://xmlns.jcp.org/jsf/
facelets">
  <h:head>
    <title>
      <ui:insert name="title">Título Padrão</ui:insert>
    </title>
  </h:head>
  <h:body>
    <h1><ui:insert name="title">Título Padrão</ui:insert></h1>
    <hr/>
    <ui:insert name="content">Conteúdo Padrão</ui:insert>
    <hr/>
  </h:body>
</html>
```

# Uso de Templates

## (página que usa o *layout*)

...

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/
facelets" xmlns:h="http://xmlns.jcp.org/jsf/html"
template="./layout.xhtml">
  <ui:define name="title">Create a new book</ui:define>
  <ui:define name="content">
    <h:form>
      <h:outputLabel value="Nome: "/>
      <h:inputText value="#{usuarioBean.nome}"/>
      ...
      <h:commandButton value="Cadastrar"
        action="#{usuarioBean.cadastra}"/>
    </h:form>
  </ui:define>
</ui:composition>
```

# Exercício

- Revisite o exercício anterior para fazer uso de um *layout* padrão
  - Coloque no cabeçalho o nome da universidade, curso e disciplina
  - Coloque no rodapé seu nome
- Dica: o NetBeans ajuda a criar o *layout* e a criar as páginas que usam o *layout*

# Bibliografia

- “Java EE 7 Tutorial”, Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito
- “Java EE 7: The Big Picture”, Danny Coward
- “Java EE 7 Recipes”, Josh Juneau
- “Beginning Java EE 7”, de Antônio Gonçalves

# JavaServer Faces (JSF)

