Lição 10

Tópicos Avançados de JavaServer Faces

Objetivos

Ao final desta lição, o estudante será capaz de:

- Conhecer a classe *FacesContext*
- Entender os componentes validadores e para conversão de tipo
- Criar componentes e como desenvolver um conjunto de *tags* personalizadas

FacesContext e a Árvore de Componentes

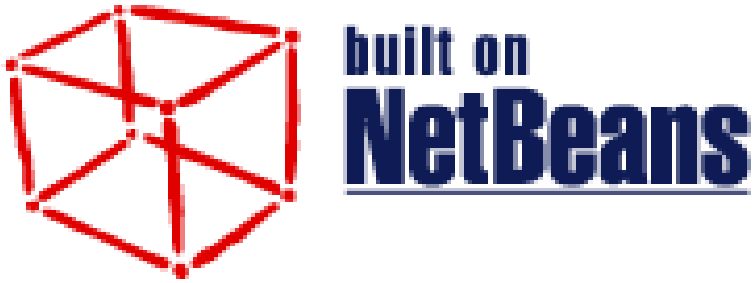
- Para cada *view* que utilize os elementos UI:
  - Existe uma estrutura de árvore de componentes responsável por sua modelagem
- A especificação do JSF requer que todas suas implementações:
  - Armazenem essa árvore de componentes dentro do objeto *FacesContext*

Árvore de Componentes

- Permite o acesso a todos os componentes da interface com o usuário e seus dados
- Adicionar componentes a *view* através de programação
- Alterar ou adicionar conversores ou validadores dinamicamente
- Remover componentes
- Utilizada para prover redirecionamento de tela dentro do *framework faces*

Árvore de Componentes

- Passaremos agora para o NetBeans

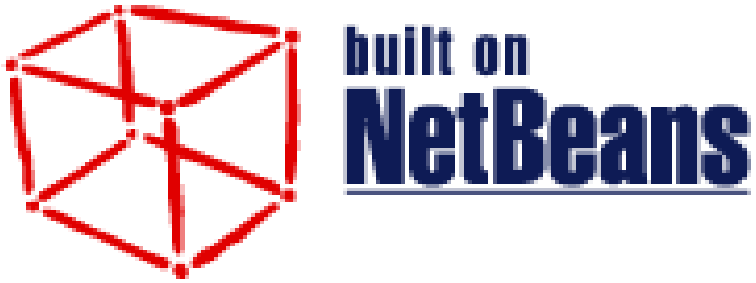


FacesContext e o *ExternalContext*

- Objeto *ExternalContext*, acessado através do *FacesContext*, permite acessar o ambiente onde o *framework* está sendo executado
- Para uma aplicação WEB, permite acessar:
  - O objeto *HttpServletRequest* representando a requisição atual
  - O mapa dos objetos armazenados na sessão do usuário
  - O objeto *ServletContext* que representa o contexto de toda a aplicação WEB

FacesContext e o ExternalContext

- Passaremos agora para o NetBeans

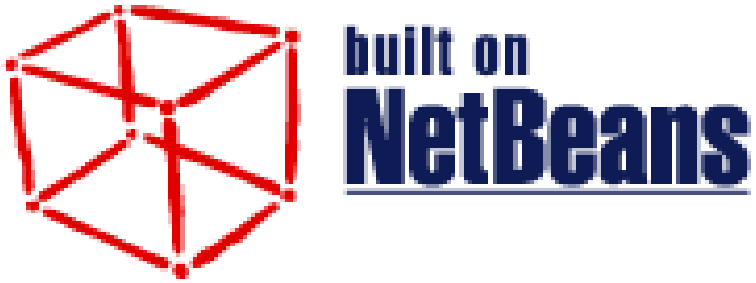


Validadores Padrão do JSF

- JSF provê um conjunto de validadores que podemos utilizar na nossa aplicação
- Três validadores padrões:
  - **DoubleRangeValidator**
    - Tag JSP: validateDoubleRange
    - Atributos: minimum, maximum
  - **LengthValidator**
    - Tag JSP: validateLength
    - Atributos: minimum, maximum
  - **LongRangeValidator**
    - Tag JSP: validateLongRange
    - Atributos: minimum, maximum

Utilizando os Validadores Padrão

- Passaremos agora para o NetBeans



Validação Personalizada

- Estender a classe do componente UI que recebe a requisição
- Podemos sobrescrever seu método de validação
  - Solução não-portável
- Criar um método de validação externo
- Criar nossas próprias implementações da interface *Validator* em separado
- Registrá-las no *framework* e então colocá-las no componente UI

Método de Validação Externo

- Criar um método de validação externo é idêntico a criar um método de aplicação para manipulação eventos:
  - Criar um método que esteja dentro de um certo conjunto de regras e dentro de um *JavaBean* gerenciado pelo *framework*
  - Fazer a ligação deste método com o componente UI apropriado
- O método deve estar conforme as seguintes regras:
  - Ser declarado *public*, com o retorno tipo *void*
  - Não existir limitações quanto ao nome do método
  - Receber os seguintes parâmetros, na seguinte ordem: *FacesContext ctxt*, *UIInput component*, *Object value*
  - Deve declarar o lançamento de uma exceção do tipo *ValidatorException*

Método de Validação Externo

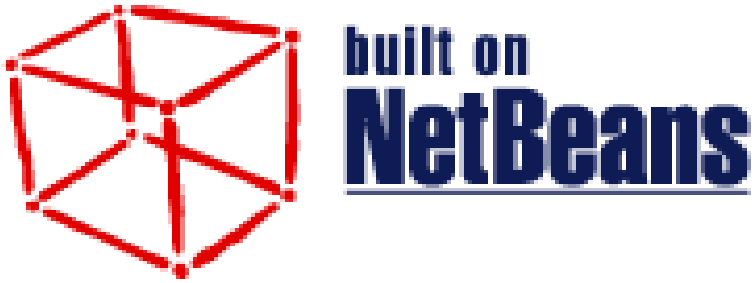
- Cabeçalho de um método de validação personalizado:

```
public void validatePassword(FacesContext ctxt,
        UIInput component, Object value)
                throws ValidatorException
```

  - O objeto da classe *FacesContext* nos provê acesso a fontes externas
  - O objeto da classe *UIInput* é a instância do componente de entrada que requer validação
  - Object possui o valor dentro do componente que requer a validação

Implementando o Método de Validação Externo

- Passaremos agora para o NetBeans

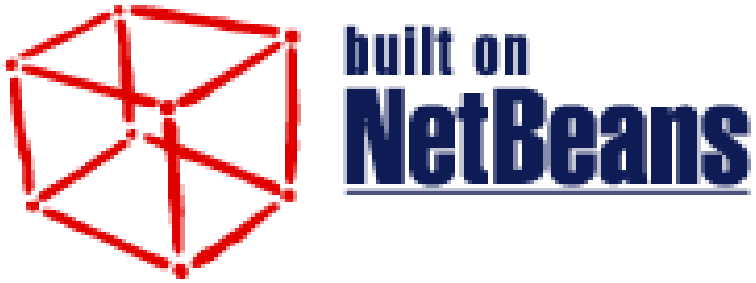


Externo vs. Em separado

- Diferença:
  - Como eles são utilizados
  - Um método do *validator* externo é mais utilizado para validação de código específico de um componente em particular
  - Uma implementação do *validator* em separado é utilizada para códigos de validação de propósito geral que serão reutilizados extensivamente dentro das aplicações

Implementação da Interface Validator em Separado

- Passaremos agora para o NetBeans



Conversores

- Converte os valores informados pelo usuário no formato ou tipo apropriado utilizado internamente pelo servidor
- **Bi-direcional**: Podem ser utilizados para alterar os dados internos que são mostrados para o usuário
- Definem os métodos *getAsString*() e *getAsObject*() que podem ser chamados pelo *framework* no momento apropriado

Conversores

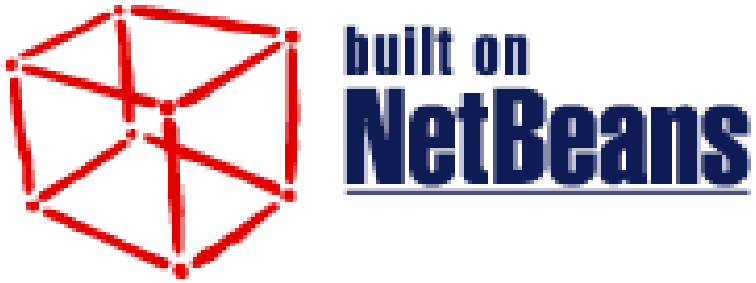
Converter Class	*Converter ID*
BigDecimalConverter	BigDecimal
BigIntegerConverter	BigInteger
IntegerConverter	Integer
ShortConverter	Short
ByteConverter	Byte
ShortConverter	Short
CharacterConverter	Character
FloatConverter	Float
DoubleConverter	Double
BooleanConverter	Boolean
DateTimeConverter	DateTime
NumberConverter	Number

DateTimeConverter

- Utilizado para converter entradas do usuário em instâncias de *java.util.Date*
- Provê um número de atributos com os quais o desenvolvedor pode especificar o formato utilizado na conversão
  - Formato especificado é forçado quando o usuário preenche sua entrada
  - Se violado, um erro de conversão ocorrerá e o *framework* não continuará o processamento

DateTimeConverter

- Passaremos agora para o NetBeans

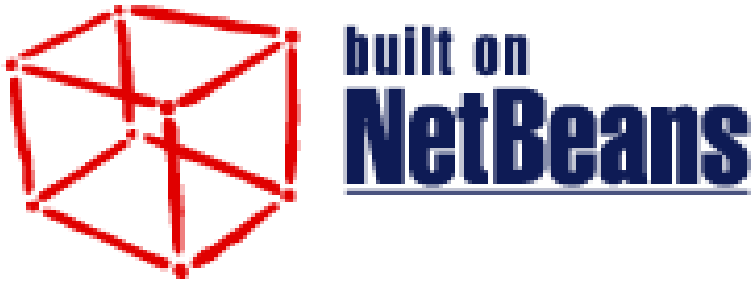


NumberConverter

- Utilizado para converter entradas de número e forçar formatações especiais nelas
- Quaisquer padrões ou símbolos indicados nos atributos são forçados como regras para o usuário
- Se violadas, um erro de conversão ocorrerá e o processamento será terminado

NumberConverter

- Passaremos agora para o NetBeans



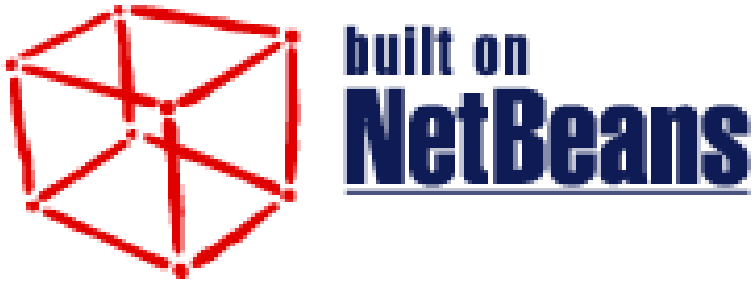
Conversores Personalizados

- Podem ser criados através da criação de uma classe que implemente a interface *Converter*
- Esta interface define dois métodos:

```
public Object getAsObject(FacesContext ctxt,
  UIComponent component, String input)
public Object getAsString(FacesContext ctxt,
  UIComponent component, Object source)
```

Conversores Personalizados

- Passaremos agora para o NetBeans

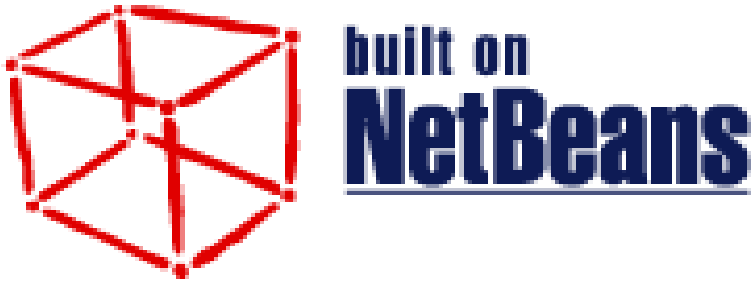


Registrando Tratadores de Ação em Componentes da View

- JSF introduz o conceito de programação baseada em eventos no ambiente WEB
- Alguns dos componentes UI que o JSF provê irão, dada a devida ação ou entrada do usuário
- Gerar eventos que podem ser processados por tratadores de ação

Registrando Tratadores de Ação em Componentes da View

- Passaremos agora para o NetBeans



Sumário

- FacesContext e a Árvore de Componentes
- *FacesContext* e o *ExternalContext*
- Validadores
- Conversores
- Registrando Tratadores de Ação

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**