Lição 8

Tópicos avançados no Framework Struts

Objetivos

Ao final desta lição, o estudante será capaz de:

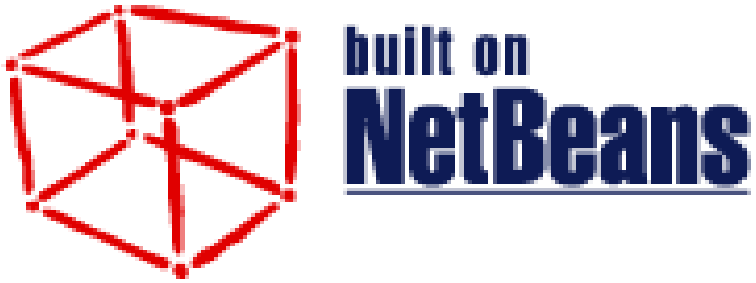
- Usar os *DynaActionForms* para minimizar o número de classes que precisamos desenvolver
- Prover validação do lado servidor em nossa aplicação com o *framework Validator*
- Compartimentalizar a camada de apresentação com o *framework Tiles*

DynaActionForms

- Funciona exatamente igual ao *ActionForms*
- Uma instância pode ser obtida e seus métodos chamados por *Action handlers*
- A principal diferença é que cada *DynaActionForm* não é definido ou declarado com uma classe separada
- Configurado dentro do arquivo *struts-config.xml*
- *Criar DynaActionForm é mais simples e rápido do que codificar uma instância completa de um ActionForm*
- Não é necessário listar todas as propriedades do formulário e criar os métodos *get* e *set* para cada uma delas
- Declarar o nome e tipo da propriedade
- E responsabilidade do *framework* prover uma instância de trabalho baseado nas informações

DynaActionForms

- Passaremos agora para o NetBeans



DynaActionForms

- São os seguintes os tipos Java suportados pelo *DynaActionForm*:
  - java.lang.BigDecimal
  - java.lang.BigInteger
  - boolean e java.lang.Boolean
  - char e java.lang.Character
  - double e java.lang.Double
  - float e java.lang.Float
  - int e java.lang.Integer
  - long e java.lang.Long
  - short e java.lang.Short
  - java.lang.String
  - java.lang.Date
  - java.lang.Time
  - java.sql.TimeStamp

DynaActionForms

- Mais conveniente, mas não é sempre a melhor solução
- Existem ainda situações onde usar *ActionForms* é mais apropriado:
  - *DynaActionForms* suporta apenas um conjunto limitado de tipos Java
  - *DynaActionForms* não suporta o conceito de herança

Validadores

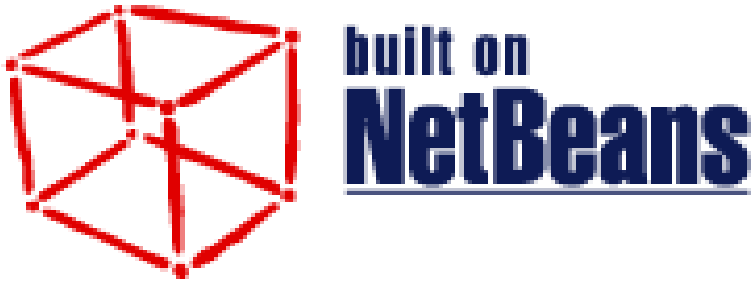
- Validação
  - Valida o formato e o conteúdo dos valores fornecido pelo usuário
  - Uma tarefa que deve ser realizada para toda entrada de dados
- *Framework* provido pelo *Struts* para aliviar a vida do desenvolvedor que teria que fazer a validação
- Vantagens:
  - Prove várias regras de validação pré-definidas
  - Elimina redundância no código de validação
  - Prove ponto único de manutenção

Validadores

- Passos requeridos para inclusão da funcionalidade de Validadores dentro de aplicações *Struts* já existentes:
  - Configure o *plug-in Validator*
  - Declare os formulários requerendo validação
  - Crie as mensagens mostradas no caso de falha da validação
  - Modifique o arquivo *struts-config* para ativar a validação automática

Configurando o *Validator Plug-In*

- Passaremos agora para o NetBeans



validator-rules.xml

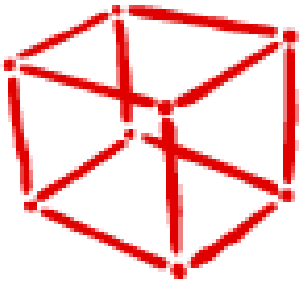
- Define as classes que implementam o código de validação
- O *framework* vem com uma cópia deste arquivo com classes de validação pré-definidas já configuradas
  - Nomes lógicos dos validadores que são entregues dentro do *framework*:
  - *required*
  - *mask*
  - *minlength*
  - *byte, short, integer, long, float, double*
  - *range*
  - *date*
  - *creditCard*
  - *email*
  - *maxlength*

validation.xml

- Declara quais formulários requerem validação
- Declara quais regras de validação devem ser implementadas
- O *framework* fornece a estrutura do arquivo
- Desenvolvedores devem configurar este arquivo

validation.xml

- Passaremos agora para o NetBeans



built on NetBeans

Definindo o Pacote de Recursos

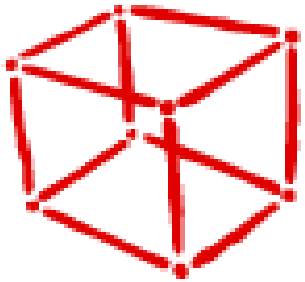
- O elemento <arg0> define a chave que necessita como entrada um conjunto de recursos
- *Framework* v*alidator* faz uso do mesmo conjunto de recursos que o *Struts framework* utiliza
- Pode ser encontrado no diretório *WEB-INF/classes* sob o nome *ApplicationResources.properties*
- Exemplo:

```
error.loginname.required=Por favor informe seu
  login

error.password.required=Informada senha em branco
  ou com menos de 4 caracteres
```

Usando o Pacote de Recursos

- Passaremos agora para o NetBeans



built on NetBeans

Tiles

- Permite a fácil definição de modelo (*templates)* e instâncias de telas as quais podemos usar com nossas aplicações
- É um modelo que pode ser reutilizado por qualquer página da aplicação
- Usar modelos permite a aplicação a aparência mais consistente e padronizada

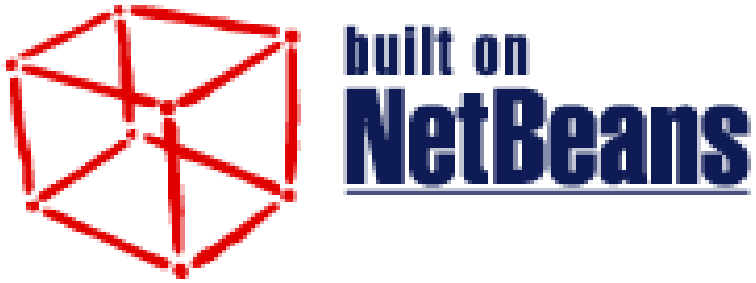
Modelos (*templates*)

Modelos (*templates*)

- Como podemos implementar um conjunto de páginas similares?
  - Refatorando as entidades de código separadas que seriam duplicada por todo código da aplicação
  - Usar modelos que definam o formato e a visão padrão das páginas
  - Qualquer mudança necessária em um dos aspectos da camada de apresentação seria aplicada a todas as páginas uma única vez

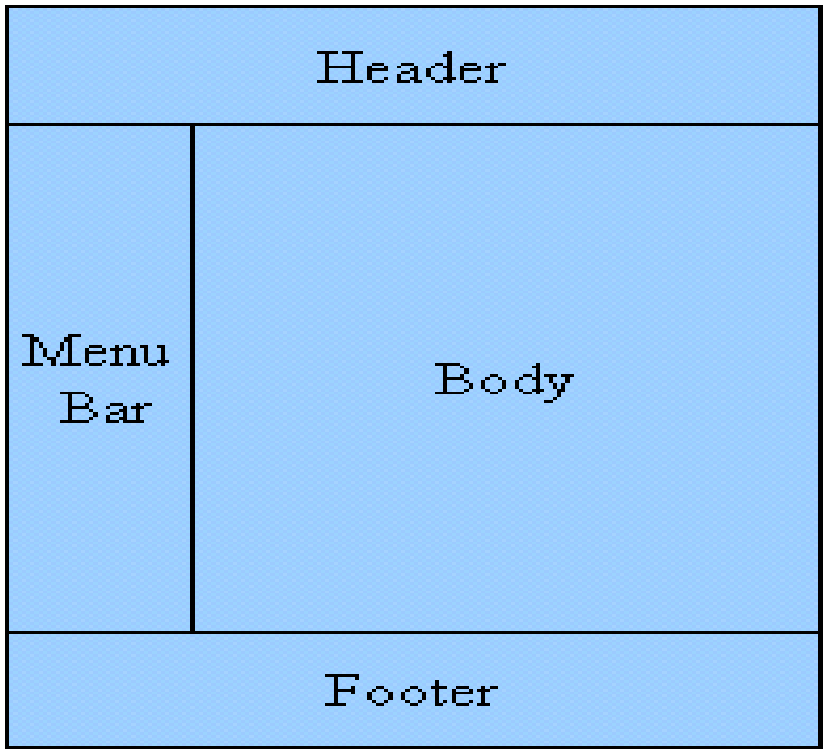
Preparando o *Tiles*

- Passaremos agora para o NetBeans



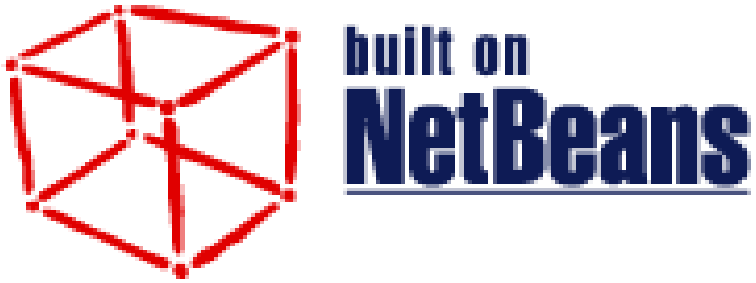
Construindo um Modelo de Layout

- O primeiro passo na construção de uma modelo é identificar os componentes a serem colocados



Construindo um Modelo de Layout

- Passaremos agora para o NetBeans

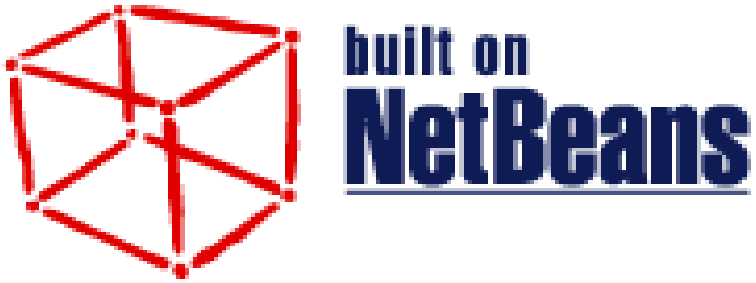


Criando Definições de Telas

- Uma vez que temos um modelo, podemos fazer uso dele para definir uma tela
- A criação de definições pode ser feita dentro do *Tiles framework* de duas maneiras:
  - Definição usando páginas JSP
  - Definição usando um arquivo de configuração XML

Criando Definições de Telas

- Passaremos agora para o NetBeans



Usando as Definições de Telas

- Para colocar uma Definição em uso, podemos usar o tag <tiles:insert> e fornecer a ele o nome da definição a ser mostrada:
  - Exemplo:

```
<%@ taglib
  uri="http://jakarta.apache.org/struts/tags-
  tiles" prefix="tiles" %>

<tiles:insert beanName="welcomePage"
  flush="true"/>
```

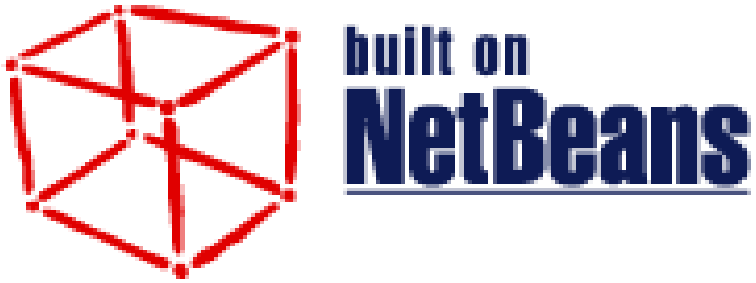
  - Problema: aumenta o número de JSPs requirida para mostrar diferentes telas para o usuário.

Usando as Definições de Telas

- Melhor abordagem: Fazer uso das definições como alvos dos *ActionForwards*
  - Pela inclusão do *Tiles framework* como um *plugin* do *Struts*, o *Struts* reconhece as definições de telas criadas usando o *Tiles*
  - Os nomes das telas podem ser usadas no lugar das reais localizações nos *tags* <*forward*>

Usando as Definições de Telas

- Passaremos agora para o NetBeans

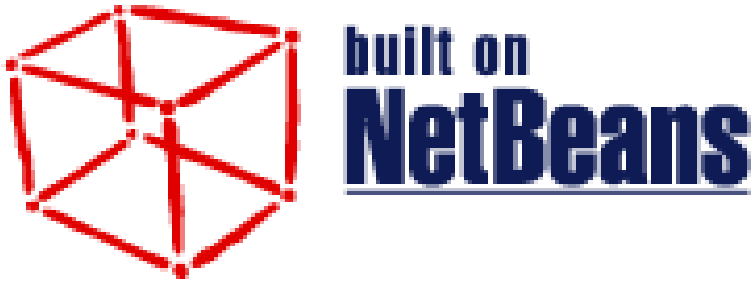


Estendendo as Definições de Telas

- Funciona na maioria das vezes como a herança de classes Java
- Telas estendida herda todas as propriedades e atributos das telas pai
- Permite a criação de uma definição de tela base
- Vantagens:
  - Evitamos a repetição das definições dos valores dos atributos
  - As mudanças são propagadas para todas as telas

Estendendo as Definições de Telas

- Passaremos agora para o NetBeans



Sumário

- *DynaActionForms*
- Validadores
- Pacote de Recursos
- Modelos com *Framework* Tiles

Parceiros

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