

## JBoss Seam

### Simplicidade e produtividade no desenvolvimento de aplicações Web

#### Aprenda a desenvolver aplicações Web utilizando a integração perfeita entre JSF e EJB 3

FÁBIO AUGUSTO FALAVINHA

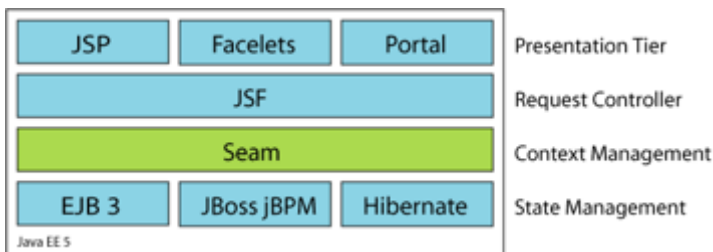
O JBoss Seam é um framework para desenvolvimento de aplicações Java EE que integra, facilmente, tecnologias como Asynchronous JavaScript and XML (AJAX), JavaServer Faces (JSF), Java Persistence API (JPA), Enterprise Java Beans (EJB 3.0) e Business Process Management (BPM).

O Seam foi desenvolvido para eliminar a complexidade em níveis de arquitetura e API. Oferece aos desenvolvedores total controle sobre a implementação da lógica de negócio sem se preocupar com a exposição das informações e/ou configuração excessiva de arquivos XML, dispondo de anotações para classes Java e componentes bem definidos para a camada de apresentação.

Gavin King, líder do projeto Seam e líder da especificação da *JSR 299: Web Beans* escreveu a proposta que visa padronizar e simplificar a forma de desenvolvimento sugerida pelo Seam, conforme citado no parágrafo anterior, promovendo assim, uma API para criação de aplicações web de fácil manutenção e ótima produtividade.

#### Iniciando no JBoss Seam

A principal funcionalidade do framework JBoss Seam é integrar JSF e EJB3 (veja a **Figura 1**). O Seam permite esta integração através de componentes gerenciados (*managed components*), sem exceder em lookups na árvore JNDI (acesso a recursos do servidor de aplicação, componentes EJBs, etc), declaração maçante de componentes JSF (*backing beans*) via arquivo *faces-config.xml* ou controlar a passagem de objetos entre as camadas de apresentação e negócio (utilização em excesso do *pattern Value Object*).



**Figura 1.** Integração do framework JBoss Seam em uma arquitetura Java EE

Desenvolver uma aplicação utilizando Seam é uma tarefa simples, mas deve seguir os seguintes conceitos:

- **Data Model:** Entidades (Entity) devem representar os objetos de negócio a serem persistidos e utilizados nas regras de negócio. Pode-se utilizar entidades gerenciadas via JPA ou POJOs (Plain Old Java Objects) mapeados via Hibernate;
- **User Interface (UI):** As páginas JSF exibem a interface gráfica, capturando os dados enviados via formulário e mostrando os resultados. Os campos, na interface gráfica, são mapeados aos objetos de negócio, acessado via JSF EL (*Expression Language*);
- **Handle Web Events:** A camada de negócio pode ser implementada utilizando EJB 3 session beans ou apenas classes Java anotadas. Esta camada atuará como *controller* para os eventos disparados pelas páginas JSF.

#### Desenvolvendo uma Agenda de Contatos

Vamos implementar uma Agenda de Contatos utilizando o framework Seam, obtendo uma aplicação web que contém funcionalidades de inclusão, remoção, alteração e pesquisa dos dados, o famoso CRUD.

Antes de começar a trabalhar, vamos criar o ambiente de desenvolvimento. Neste exemplo vamos usar a IDE NetBeans 6 com a plataforma Java 5 e o servidor de aplicação Glassfish (integrado com o NetBeans). A forma mais fácil de obter e instalar todos esses itens é ir em <http://download.netbeans.org/netbeans/6.0/final/> e baixar uma das distribuições que inclui o Glassfish.

Baixe também o framework JBoss Seam 2.0.0.GA (download em <http://www.jboss.com/products/seam>).

O download do código-fonte deste exemplo pode ser feito no site da revista.

### **Criando o Data Model**

Vamos desenvolver o modelo de dados que será a base para as regras de negócio a serem implementadas. Para isso criaremos uma classe **Contato** contendo os seguintes atributos: Nome, Número do Celular, Telefone Comercial, Telefone Residencial, E-mail e observações (veja a **Listagem 1**).

**Listagem 1.** Implementação da classe *Contato.java* e *Telefone.java*, utilizando anotações Seam e EJB 3 Entity Bean

```
package br.jm.seam.agendacontatos.entity;
```

```
@Entity
@Name("contato")
public class Contato implements Serializable {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name="NOME")
    private String nome;

    @OneToOne(fetch=FetchType.EAGER, cascade=CascadeType.ALL)
    private Telefone telefoneCelular;

    @OneToOne(fetch=FetchType.EAGER, cascade=CascadeType.ALL)
    private Telefone telefoneComercial;

    @OneToOne(fetch=FetchType.EAGER, cascade=CascadeType.ALL)
    private Telefone telefoneResidencial;

    @Column(name="EMAIL")
    private String email;

    public Contato() {
        this.telefoneCelular = new Telefone();
        this.telefoneComercial = new Telefone();
        this.telefoneResidencial = new Telefone();
    }
    /* getters / setters */
}
```

```
package br.jm.seam.agendacontatos.entity;
```

```
@Entity
public class Telefone implements Serializable {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name="DDI")
    private Integer ddi;

    @Column(name="DDD")
    private Integer ddd;

    @Column(name="NUMERO")
    private Integer numero;
    /* getters / setters */
}
```

Note que as anotações **@Entity**, **@Id**, **@GeneratedValue**, **@Column**, **@OneToOne** fazem parte do pacote **javax.persistence.\*** do EJB3 Entity Bean, e

identificam a classe como uma entidade (tabela) do banco de dados, assim como suas colunas.

Já a anotação **@Name (org.jboss.seam.annotations.Name)**, faz parte do framework Seam, permitindo registrar o nome pelo qual instâncias da classe **Contato** serão conhecidas pelo framework Seam. Todo o componente registrado pelo framework Seam pode ser referenciado por outros componentes (páginas JSF e EJB 3 Session Beans).

*Para que o framework Seam possa gerenciar um componente, deve-se colocar o arquivo **seam.properties** como marcador no pacote que conterá as classes do componente (marcadas com a anotação **@Name**), ou seja, o framework Seam irá procurar junto ao classloader da aplicação os pacotes que contêm este arquivo e a partir daí gerenciar as classes marcadas com a anotação. Em nosso exemplo, iremos colocar no mesmo nível que o pacote **"br"** em **"br.jm.seam.agendacontatos"**.*

Depois de implementar as entidades, vamos configurar o arquivo *persistence.xml* (Veja a **Listagem 2**).

**Listagem 2.** Configuração do arquivo de persistência *persistence.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/
xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="tim-environment-pu" transaction-type="JTA">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <jta-data-source>jdbc/___default</jta-data-source>
  </persistence-unit>
</persistence>
```

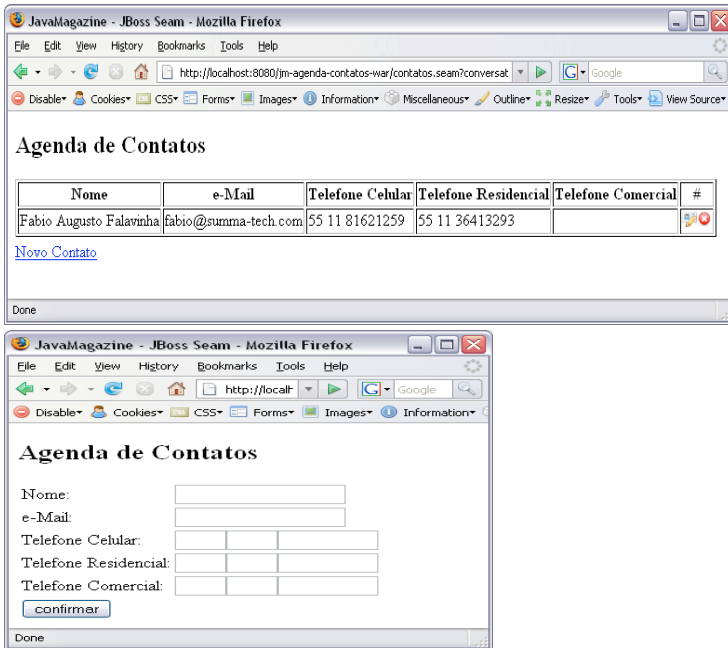
Para o nosso exemplo utilizaremos o *Oracle Toplink Essentials*, implementação da JPA, e já disponibilizado no Glassfish, juntamente com transações JTA, ou seja, todas as transações com o banco de dados serão feitas através do contêiner ejb, para simplificar o exemplo.

A conexão ao banco de dados será feita através do data source **jdbc/\_\_\_default**, configuração disponibilizada pelo Glassfish, com acesso ao banco de dados Derby, também incluído no servidor de aplicação.

As entidades implementadas serão mantidas no arquivo *agenda-contatos-entity.jar* que estará localizado no diretório */lib* do arquivo EAR. Este arquivo JAR fará parte do arquivo de deploy da aplicação.

### **Preparando a User Interface (UI)**

Para nosso exemplo, vamos criar duas páginas JSF (**Figura 2**). Uma contendo uma lista para visualizar os contatos cadastrados, podendo-se a partir dela: excluir, alterar ou incluir um novo contato. A segunda página contendo um formulário para inclusão de um novo contato ou alteração. (veja o código dos arquivos *contatos.xhtml* e *contato.xhtml* disponíveis para download no site da Java Magazine).



**Figura 2.** Tela inicial da Agenda de Contatos, contendo a lista de contatos e o formulário de contato, respectivamente

Veja que na página *contatos.xhtml* foi utilizado o componente `<h:dataTable>` para exibir uma lista de contatos. Dentro deste, temos componentes `<s:link>` para realizar a implementação das funcionalidades de **Editar Contato** e **Remover Contato**. Estes componentes permitem utilizar EL (Expression Language) com passagem de parâmetros na invocação de métodos, como pode ser visto na **Listagem 3**.

**Listagem 3.** `<s:link />` - Invocação de métodos com passagem de parâmetros

Lista de Contatos:

```
<h:dataTable border="1" value="#{contatos}" var="contato" ... >
```

Editar Contato:

```
<s:link id="editar" action="#{agenda.editar(contato)}">
  <h:graphicImageborder="0" url="imagens/alterar.png"/>
</s:link>
```

Remover Contato:

```
<s:link id="remover" action="#{agenda.remover(contato)}">
  <h:graphicImageborder="0" url="imagens/remover.png"/>
</s:link>
```

O objeto passado como argumento é a variável criada como índice para iteração da coleção de contatos.

Caso não queira passar o objeto **contato** como parâmetro, pode-se utilizar o componente `<h:commandLink>` invocando um método através de uma EL simples e declarar em seu *"Backing Bean"* um atributo do mesmo tipo com a anotação `@DataModelSelection` (**org.jboss.seam.annotations.datamodel.DataModelSelection**) (veja a **Listagem 4**).

**Listagem 4.** Invocando métodos sem passagem de parâmetros

Lista de Contatos:

```
<h:dataTable border="1" value="#{contatos}" var="contato" ... >
```

Editar Contato:

```
<h:commandLink action="#{agenda.editar}">
  <h:graphicImageborder="0" url="imagens/alterar.png"/>
</h:commandLink>
```

Remover Contato:

```
< h:commandLink action="#{agenda.remover}">
```

```
<h:graphicImageborder="0" url="imagens/remove.png"/>
</h:commandLink>
```

Backing Bean: Agenda  
**@Name("agenda")**  
 public class Agenda {

```
    @DataModelSelection
    private Contato contatoSelecionado;
}
```

A anotação **@DataModelSelection** marca o atributo **contatoSelecionado** da classe **Agenda** (componente Seam), no qual será injetado o objeto **contato** selecionado na lista de contatos.

### Configurando o Seam na Web

O framework Seam atua como um filtro das requisições feitas ao web-container. No caso, a servlet **FacesServlet** recebe as requisições, mas atua apenas como redirecionador na resposta ao cliente (ver **Listagem 5**).

**Listagem 5.** Configuração do arquivo *web.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <!-- Iniciando configuracao do Seam -->
  <listener>
    <listener-class>org.jboss.seam.servlet.SeamListener</listener-class>
  </listener>

  <servlet>
    <servlet-name>Seam Resource Servlet</servlet-name>
    <servlet-class>org.jboss.seam.servlet.ResourceServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Seam Resource Servlet</servlet-name>
    <url-pattern>/seam/resource/*</url-pattern>
  </servlet-mapping>

  <filter>
    <filter-name>Seam Filter</filter-name>
    <filter-class>org.jboss.seam.web.SeamFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>Seam Filter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <!-- Final da configuracao do Seam -->

  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/navigation.xml</param-value>
  </context-param>

  <context-param>
    <param-name>
      javax.faces.STATE_SAVING_METHOD
    </param-name>
    <param-value>client</param-value>
  </context-param>

  <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>*.seam</url-pattern>
```

```

</servlet-mapping>

<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>

</web-app>
  
```

O código-fonte, marcado em negrito na **Listagem 5**, habilita o uso do framework Seam na aplicação. Note que a configuração do framework Seam é baseada em um *listener* (**SeamListener**), *filter* (**SeamFilter**) e uma *servlet*, para manipulação de recursos como: arquivos JS (JavaScript), CSS, imagens.

Pronto! Ao fazer esta configuração sua aplicação já está apta para trabalhar com o Seam.

### Configurando JSF Facelets

Precisamos declarar no arquivo de configuração do framework JSF *faces-config.xml*, que o controle de páginas será feita através de Facelets, onde será utilizado XHTML (veja a marcação **javax.faces.DEFAULT\_SUFFIX** na **Listagem 5**) ao invés de páginas JSP, padrão JSF. (veja a **Listagem 6**).

**Listagem 6.** Configuração do arquivo *faces-config.xml*.

```

<faces-config version="1.2"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">

  <application>
    <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
  </application>

</faces-config>
  
```

### Navegação das páginas

Como foi mostrado na **Listagem 5**, declaramos o arquivo *navigation.xml* como arquivo de configuração do fluxo de páginas da aplicação (veja a **Listagem 7**).

**Listagem 7.** Configuração do arquivo *navigation.xml*.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config
PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
  <navigation-rule>
    <from-view-id>/contato.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>contatos</from-outcome>
      <to-view-id>/contatos.xhtml</to-view-id>
      <redirect />
    </navigation-case>
  </navigation-rule>

  <navigation-rule>
    <from-view-id>/contatos.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>novoContato</from-outcome>
      <to-view-id>/contato.xhtml</to-view-id>
      <redirect />
    </navigation-case>
    <navigation-case>
      <from-outcome>editarContato</from-outcome>
      <to-view-id>/contato.xhtml</to-view-id>
      <redirect />
    </navigation-case>
  </navigation-rule>
</faces-config>
  
```

A configuração vista acima é padrão do framework JSF, e possui algumas limitações:

- Não permite especificar parâmetros via *request* ao redirecionar para outros controles;
- Não há possibilidade de iniciar e finalizar *conversations* dentro das *rules* (*navigation-rule*);
- Não há possibilidade de recuperar valores arbitrários utilizando EL (por exemplo, avaliar propriedades de um objeto retornado de uma ação).

O framework Seam disponibiliza um controle de páginas via XML, através da configuração do arquivo *pages.xml* (veja a **Listagem 8**).

**Listagem 8.** Configuração do arquivo *pages.xml* com o mesmo fluxo de páginas descrito em *faces-config.xml*.

```
<pages>
  <page view-id="/contato.xhtml">
    <navigation>
      <rule if-outcome="contatos">
        <redirect view-id="/contatos.xhtml"/>
      </rule>
    </navigation>
  </page>

  <page view-id="/contatos.xhtml">
    <navigation>
      <rule if-outcome="novoContato">
        <redirect view-id="/contato.xhtml"/>
      </rule>
      <rule if-outcome="editarContato">
        <redirect view-id="/contato.xhtml"/>
      </rule>
    </navigation>
  </page>
</pages>
```

Veja que as tags *page* (informam ação de página, ação de navegação, recursos de internacionalização e parâmetros de página, para uma página específica), *navigation* (declara o contexto da navegação da página), *rule* (declara condições e regras para a navegação da página) e *redirect* (declara a página a ser redirecionada) são parecidas com as tags declaradas no arquivo *faces-config.xml*.

A configuração da navegação das páginas para o exemplo Agenda de Contatos é simples. Mas, caso sua aplicação necessite passar parâmetros ao redirecionar páginas, iniciar ou finalizar *Conversations* (modo de transação sobre um conjunto de funcionalidades a serem realizadas), ou capturar valores retornados pela EL em uma *Rule*, utilize o arquivo *pages.xml* para ter flexibilidade em descrever esta navegação entre as páginas JSF.

### Implementando componentes de negócio como EJBs – *Handle Web Events*

Nesta seção vamos implementar o componente de negócio que irá controlar todos os eventos disparados pelas páginas JSF.

Note que nas listagens vistas anteriormente, o componente Seam **agenda** foi referenciado diversas vezes, principalmente nas páginas JSF.

Vamos implementar este componente utilizando a tecnologia EJB 3, sendo um EJB Stateless Session Bean, veja a **Listagem 9**.

**Listagem 9.** EJB Stateless session bean *AgendaBean.java*

```
@Stateless
@Transactional(value=TransactionAttributeType.REQUIRED)
@Name("agenda")
public class AgendaBean implements Agenda {

    @In(required=false)
    @Out(required=false)
    private Contato contato;

    @PersistenceContext
    private EntityManager entityManager;
```

```
@DataModel
private List<Contato> contatos;

@Factory(value="contatos")
@SuppressWarnings("unchecked")
public void listarContatos() {
    this.contatos = this.entityManager.createQuery("select o from Contato
o").getResultList();
}

@Begin(join=true)
public String novoContato() {
    this.contato = new Contato();
    return ("novoContato");
}

@Begin(join=true)
public String editar(Contato contato) {
    this.entityManager.refresh(contato);
    this.contato = contato;
    return ("editarContato");
}

public String remover(Contato contato) {
    this.entityManager.remove(contato);
    return ("contatos");
}

@End
public String salvar() {
    if (this.contato.getId() != null) {
        this.entityManager.merge(this.contato);
    } else {
        this.entityManager.persist(this.contato);
    }
    return ("contatos");
}
}
```

A **Listagem 9** destaca em negrito as principais anotações utilizadas na implementação deste componente, são elas:

- **@In** e **@Out** (localizadas no pacote **org.jboss.seam.annotations**) – estão marcando o atributo **contato**, onde será injetada uma instância do componente Seam **Contato** (criada pela camada de apresentação), com o mesmo nome que o atributo. A mesma instância será exportada novamente para a camada de apresentação, em contexto de *request* também com o mesmo nome, mantendo o processamento das páginas JSF. Note que o atributo **required=false** está sendo utilizado para identificar ao framework Seam que a injeção dupla deste componente Seam não é obrigatório, pois estaremos utilizando o mesmo componente **agenda** em momentos que não necessitam de injeção de dependência, por exemplo ao exibir a lista de contatos;
- **@DataModel (org.jboss.seam.annotations.datamodel.DataModel)** – esta anotação irá exportar a coleção **contatos**, com o mesmo nome do atributo em contexto de *request*, como uma coleção a ser exibida nas páginas JSF;
- **@Factory(value="contatos") (org.jboss.seam.annotations.Factory)** – esta anotação marca o método que irá inicializar a coleção **contatos**, nome informado pelo atributo **value**, atribuindo o valor processado;
- **@Begin** e **@End** (localizadas no pacote **org.jboss.seam.annotations**) – estas duas anotações marcam os métodos que iniciam e terminam uma funcionalidade transacional da aplicação. No caso do exemplo da Agenda de Contatos, os métodos: **String novoContato ()** e **String editar (Contato contato)**, marcam o início da transação referente a abertura do formulário JSF para inclusão de um novo contato e abertura do formulário com os dados de um contato selecionado a partir da lista de contatos, para edição do mesmo, respectivamente. A transação é finalizada quando é invocado o método **String salvar()**, que persiste as informações do contato em banco de dados, terminando o estado de *conversation* no qual é identificado pelo framework Seam.

## Conclusão

O desenvolvimento de aplicações web deve ser ágil e produtivo. O framework Seam vem como solução de simplicidade e produtividade do desenvolvimento de aplicações focando-se no *core business*, deixando o “trabalho sujo” por conta dele. Com a apresentação das partes para o desenvolvimento de uma aplicação Seam e a explicação de anotações consideradas básicas ao desenvolvimento web, temos mais um modelo a ser seguido para desenvolver aplicações rápidas e com qualidade.

## Links

[www.jboss.com/products/seam](http://www.jboss.com/products/seam)

Página principal do JBoss Seam.

[www.seamframework.org](http://www.seamframework.org)

Página sobre o framework JBoss Seam.

[java.sun.com/javaee/javaserverfaces](http://java.sun.com/javaee/javaserverfaces)

Página oficial da tecnologia JavaServer Faces.

[java.sun.com/products/ejb](http://java.sun.com/products/ejb)

Página oficial da tecnologia Enterprise JavaBeans.

[glassfish.java.net](http://glassfish.java.net)

Página oficial do servidor de aplicação Glassfish.

[www.netbeans.org](http://www.netbeans.org)

Página oficial da IDE Open-Source NetBeans.

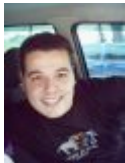
[jcp.org/en/jsr/detail?id=299](http://jcp.org/en/jsr/detail?id=299)

Página oficial da JSR Web Beans.

## Livros (opcional)

***JBoss Seam: Simplicity and power beyond Java EE, Michael Yuan e Thomas Heute, Prentice Hall, 2007***

Um bom livro para entender a mecânica do framework JBoss Seam e a integração entre JSF e EJB3, além de outras integrações com outros frameworks, dicas e exemplos sobre métodos de desenvolvimento de aplicações web com Seam.



**Fábio Augusto Falavinha (fabio@summa.com.br)** é Consultor Java/Java EE pela Summa Technologies do Brasil, atuando há mais de 8 anos com Java/Java EE. Bacharel em Ciência da Computação pela faculdade Sumaré e pós-graduado em Gestão de Qualidade de Software pela faculdade SENAC.

## Notas do DevMan:

Componentes Gerenciados (*Managed Components*): são componentes gerenciados pelo servidor de aplicação, seja através do web container ou ejb container. Suas dependências com outras classes podem ser injetadas (IoC), o mesmo acontece com os valores armazenados em suas propriedades, que podem ser recuperados sem ter dependências entre os componentes da aplicação.

EL (*Expression Language*): linguagem script que permite acesso aos componentes Java, separando o código Java de códigos utilizados nas páginas JSP.

Anotações (*Annotations*): incluída na versão Java 5, annotations são meta-informações que podem ser adicionadas a classes, métodos, variáveis, parâmetros e pacotes, podendo ser interpretadas pelo compilador Java, por ferramentas e

bibliotecas (para geração de código, arquivos XML, por exemplo) e em tempo de execução utilizando *Reflection*. O uso de annotations permite especificar ou alterar o comportamento da aplicação, sem manipular diretamente a lógica de programação. Em versões anteriores do Java, o comportamento da aplicação poderia ser modificado via arquivos XML, conhecidos como *deployment descriptors*.

Facelets: é um subprojeto do JSF mantido pela Sun. Facilmente integrado ao JSF e que trás as seguintes características:

- Facilidades na criação de templates e componentes reutilizáveis;
- Usa XHTML como tecnologia nas páginas;
- Precisão na validação do código-fonte;
- Permite incluir trechos de código JSF nas tags HTML;
- Melhora a performance, pois elimina o "JSP Compiler to Servlet".

### **BOX LEAD**

LEAD 2

#### **De que se trata o artigo:**

O artigo demonstra o desenvolvimento de uma aplicação Java EE utilizando o framework JBoss Seam, mostrando a integração entre as tecnologias JSF e EJB3.

#### **Para que serve:**

Este artigo serve para mostrar o uso do framework JBoss Seam, de maneira simples e produtiva, facilitando a vida do desenvolvedor na criação de aplicações Java EE utilizando o máximo das tecnologias JSF e EJB3. Assim, o desenvolvedor foca o desenvolvimento na camada de negócio utilizando EJB3 e na camada de apresentação utilizando JSF.

#### **Em que situação o tema é útil:**

Com a criação da JSR 299: Web Beans, uma API padrão, baseada no framework Seam, será criada para padronizar e simplificar o desenvolvimento de aplicações Java EE. Para quem gosta de utilizar EJB3 e JSF, o framework Seam pode ser a ferramenta necessária para dar mais produtividade ao desenvolvimento de aplicações web.

LEAD 3: BOX resumo Devman

#### **JBoss Seam – Resumo Devman:**

Utilizar o framework JBoss Seam é tirar o foco da configuração extensa e cansativa, para focar-se ao desenvolvimento do *core* da aplicação. A integração entre as duas tecnologias JSF e EJB3 trás maior interatividade entre os componentes da aplicação. Páginas JSF podem acessar através de EL componentes EJB que servirão como controladores das requisições e processadores das regras de negócio. Observe na **Listagem 9** que a configuração do EJB, como um componente Seam, é feita através da anotação @Name, onde este será acessado via EL nas páginas JSF (*contatos.xhtml* e *contato.xhtml*).