|  |  |  |
| --- | --- | --- |
| Universidade Federal de São Carlos  Departamento de Computação  Curso de Pós-Graduação Lato-Sensu em Computação  Desenvolvimento de Software para Web  **Proposta de Monografia**   |  |  | | --- | --- | | Título: | Integração entre as tecnologais JavaServer Faces e JPA: um estudo de caso. |   Autor: Marcelo Gomes de Paula  Orientador: Prof. Dr. Antonio Carlos dos Santos  São Carlos  Setembro 2009 |

**Resumo**

Aplicações para Web podem ser implementadas em diversas linguagens, sem com um ou vários frameworks, seguindo ou não padrões de projeto.

A combinação dos frameworks aplicado os padrões recomendados facilita o desenvolvimento dos softwares e a sua manutenção.

Neste trabalho será apresentado um estudo de caso de da integração do frameworks JavaServer Faces e JPA,

As vantagens e as desvantagens encontradas da sua utilização e também será apresentado os padrões envolvidos com estes frameworks.

**Lista de Figuras**

[Figura 1 12](#_Toc240998729)

[Figura 2: Diagrama de classes e interfaces básicas do JDBC 14](#_Toc240998730)

[Figura 3: Arquitetura JDBC 14](#_Toc240998731)

[Figura 4: Integração do JDBC 15](#_Toc240998732)

[Figura 5: registro de driver JDBC 16](#_Toc240998733)

[Figura 6: URL JDBC 16](#_Toc240998734)

[Figura 7: URLs para os principais bancos de dados 16](#_Toc240998735)

[Figura 8: Exemplo de conexão JDBC 17](#_Toc240998736)

[Figura 9: JDBC com singleton 17](#_Toc240998737)

[Figura 10: Tratar Exception 18](#_Toc240998738)

[Figura 11: Entidade 19](#_Toc240998739)

[Figura 12: Anotações JPA para entidade 20](#_Toc240998740)

[Figura 13: Anotações JPA Relacionamento 21](#_Toc240998741)

[Figura 14: Anotações JPA herança 21](#_Toc240998742)

[Figura 15: Inserir JPA 21](#_Toc240998743)

[Figura 16: Remover JPA 22](#_Toc240998744)

[Figura 17: Remover JPA 22](#_Toc240998745)

[Figura 18: FindBy Id JPA 24](#_Toc240998746)

[Figura 19: Find JPA 24](#_Toc240998747)

[Figura 20: Find many JPA 24](#_Toc240998748)

[Figura 21: Ciclo de vida JSF 26](file:///C:\Users\marcelo\Desktop\Monografia\Monografia_Entregar4.docx#_Toc240998749)

[Figura 22: Ciclo de Vida do JSF 26](file:///C:\Users\marcelo\Desktop\Monografia\Monografia_Entregar4.docx#_Toc240998750)

[Figura 23: Ciclo de Vida do JSF com conversão, validação e evento 28](file:///C:\Users\marcelo\Desktop\Monografia\Monografia_Entregar4.docx#_Toc240998751)

[Figura 24: JavaBeans 29](#_Toc240998752)

[Figura 25: Escopo request 30](#_Toc240998753)

[Figura 26: Escopo session 30](#_Toc240998754)

[Figura 27: Escopo application 31](#_Toc240998755)

[Figura 28: Configuração do JavaBeans 31](#_Toc240998756)

[Figura 29: Configuração do JavaBeans - exemplo 32](#_Toc240998757)

[Figura 30: Definir propriedades para o JavaBeans 32](#_Toc240998758)

[Figura 31: Propriedades do JavaBeans Exemplo 33](#_Toc240998759)

[Figura 32: Regras de navegação 35](#_Toc240998760)

[Figura 33: Árvore de componentes 36](file:///C:\Users\marcelo\Desktop\Monografia\Monografia_Entregar4.docx#_Toc240998761)

[Figura 34: Validador JSF 37](#_Toc240998762)

[Figura 35: Validador JSF exemplo 38](#_Toc240998763)

[Figura 36: Validador JSF uso 38](#_Toc240998764)

[Figura 37: Conversor JSF 39](#_Toc240998765)

[Figura 38: Declaração de conversor 39](#_Toc240998766)

[Figura 39: Exemplo de declaração de conversor 39](#_Toc240998767)

[Figura 40: Arquivo properties 40](#_Toc240998768)

[Figura 41: Página JSF com internacionalização 41](#_Toc240998769)

[Figura 42: PhaserLister 42](#_Toc240998770)

[Figura 43: Declaração do phaser-Listener 43](#_Toc240998771)

**Lista de Tabelas**

[Tabela 1: Classes de driver JDBC 16](#_Toc240998727)

[Tabela 2: Comparativo SQL VS JPQL 23](#_Toc240998728)

**Glossário**

|  |  |
| --- | --- |
| Java | Linguagem de programação |
| Framework |  |

Conteúdo

[1 Introdução ao problema, hipótese ou questão 8](#_Toc240998772)

[2 Objetivos da Monografia 8](#_Toc240998773)

[3 Metodologia 9](#_Toc240998774)

[4 Design Pattern e Framework 9](#_Toc240998775)

[4.1 Introdução ao Design Pattern 9](#_Toc240998776)

[4.2 MVC 10](#_Toc240998777)

[4.2.1 Introdução ao MVC 10](#_Toc240998778)

[4.2.2 FrameWork 12](#_Toc240998779)

[5 Persistência de dados 13](#_Toc240998780)

[5.1 Introdução Persistência de dados 13](#_Toc240998781)

[5.2 Transações 13](#_Toc240998782)

[5.3 JDBC 13](#_Toc240998783)

[5.3.1 Introdução ao JDBC 13](#_Toc240998784)

[5.3.2 Acessados banco de dados com JDBC 16](#_Toc240998785)

[5.4 JPA 18](#_Toc240998786)

[5.4.1 Introdução ao JPA 18](#_Toc240998787)

[5.4.2 Entidade 19](#_Toc240998788)

[5.4.3 Inserir 21](#_Toc240998789)

[5.4.4 Remover 21](#_Toc240998790)

[5.4.5 Atualizar 22](#_Toc240998791)

[5.4.6 Pesquisar 23](#_Toc240998792)

[6 JavaServer Faces 24](#_Toc240998793)

[6.1 Introdução ao JavaServer Faces 24](#_Toc240998794)

[6.2 Ciclo de vida do JavaServer Faces 25](#_Toc240998795)

[6.2.1 As fases do ciclo de vida do JSF 26](#_Toc240998796)

[6.2.2 Conversão, validação e evento 27](#_Toc240998797)

[6.3 Java Beans 28](#_Toc240998798)

[6.3.1 Introdução ao JavaBeans 28](#_Toc240998799)

[6.3.2 Exemplo de um JavaBeans 28](#_Toc240998800)

[6.3.3 Tipos de escopo do JavaBeans 29](#_Toc240998801)

[6.3.4 Definir propriedades no JavaBeans 32](#_Toc240998802)

[6.4 Navegação 33](#_Toc240998803)

[6.4.1 Introdução a navegação 33](#_Toc240998804)

[6.4.2 Criando a navegação 34](#_Toc240998805)

[6.5 UI Componentes 35](#_Toc240998806)

[6.5.1 Árvore de Componentes 35](#_Toc240998807)

[6.6 Componentes padrões do JSF 36](#_Toc240998808)

[6.6.1 Introdução aos validadores 36](#_Toc240998809)

[6.6.2 Criando um validador 37](#_Toc240998810)

[6.6.3 Usando um validador 38](#_Toc240998811)

[6.7 Conversores 38](#_Toc240998812)

[6.7.1 Introdução aos conversores 38](#_Toc240998813)

[6.7.2 Criado um conversores 38](#_Toc240998814)

[6.7.3 Usando um conversor 39](#_Toc240998815)

[6.8 Internacionalização 40](#_Toc240998816)

[6.8.1 Introdução a internacionalização 40](#_Toc240998817)

[6.8.2 Adicionando internacionalização a uma aplicação 40](#_Toc240998818)

[6.9 PhaseListener 41](#_Toc240998819)

[6.9.1 Introdução ao PhaseListener 41](#_Toc240998820)

[6.10 Implementação do PhaseListener 41](#_Toc240998821)

[7 Projeto 43](#_Toc240998822)

[7.1 Resumo do projeto 43](#_Toc240998823)

[7.2 Problema abordado 43](#_Toc240998824)

[8 Resultados potenciais teóricos e práticos da monografia 43](#_Toc240998825)

[9 Referências Bibliográficas 43](#_Toc240998826)

[10 Referências Bibliográfias 43](#_Toc240998827)

# Introdução ao problema, hipótese ou questão

Os processos de desenvolvimento de software tem seus prazos cada vez mais curtos com exigências cada vez maiores e diante deste novo cenário é percebido a dificuldade de algumas pessoas/empresas em usar de forma adequada as novas tecnologia aplicando boas práticas, conseqüentemente, o sucesso dos projetos de software desenvolvidos acaba não alcançando o objetivo esperado, podendo acarretar em prejuízos que transcendem o financeiro, como por exemplo, o sucesso profissional da equipe do projeto e a imagem organizacional frente ao mercado e aos seus clientes

A linguagem Java junto com seus frameworks facilita o desenvolvimento de sistema, porém devido a grande quantidade de frameworks disponíveis faz com que muitos programadores ao iniciar no desenvolvimento nesta plataforma se assustem, ao tentar aprender todos sem ter os conhecimentos básicos de OO e Java e acabam se confundindo, e não conseguindo aproveitar os recursos disponíveis.

Programadores experientes em linguagens procedurais quando iniciam no desenvolvimento Java sem conhecer os conceitos de OO e os padrões de projeto tendem a achar a organização das classes confusa é acabam misturando os conceitos e desenvolvendo sistemas difíceis de manter.

Desta forma este trabalho se consiste em apresentar a teoria envolvida nos desenvolvimento de aplicações com os frameworks JavaServer Faces e JPA, o projeto Guarabá Wally que será desenvolvido seguindo a proposta apresentada neste trabalho e um pequeno exemplo prático.

# Objetivos da Monografia

Assim, este trabalho tem como maior objetivo apresentar um caso de uso da integração das tecnologias JPA e JavaServer Faces fornecendo um modelo para o desenvolvimento de aplicações Web com os frameworks JSF e JPA oferecendo uma maior eficiência na escalabilidade e legibilidade do código.

Este caso de uso poderá servir de guia para profissionais que desejam fazer uso destas tecnologias em seus projetos

# Metodologia

Diante das diversas opções de possíveis implementações de aplicativos para WEB e da necessidade de se definir abordagem para ser utilizada no projeto Guarabá Wally. Foi realizado um estuda de caso de integração entre as tecnologias JavaServer Faces de JPA.

O JavaServer Faces foi escolhido por ser uma especificação que fazer parte do padrão Java Enterprise Edition, sendo um framework que implementa o consagrado padrão de projeto MVC, em que visa separar as responsabilidades das partes do projeto facilitando principalmente a manutenção do sistema. Porém o JSF não apresenta nenhuma solução para persistência dos dados, por isso para realizar a persistência dos dados foi escolhido o JPA.

O JPA é a especificação que permite mapear objetos em banco de dados relacionais, simplificando a integração do sistema com o banco de dados.

Através de revisões bibliográficas e auxílio de professores, profissionais e estudantes da área de desenvolvimento de software, este trabalho irá apresentar uma introdução as tecnologias envolvidas e apresentar uma metodologia a ser utilizada na implementação do sistema Guarabá Wally.

# Design Pattern e Framework

## Introdução ao Design Pattern

O conceito de padrão de projeto também conhecido pelo termo em inglês Desing Parther foi usado pela primeira vez na década de 70 pelo arquiteto e urbanista austríaco Christopher Alexander, observando que as construções apesar de serem diferentes em vários aspectos, enfrentavam problemas comuns em sua construção. Baseando nesta constatação Christopher documentou os problemas e as soluções para os mesmos e neste momento surgiu os padrões de projeto.

Em seus livros Notes on the Synthesis of Form,. The Timeless Way of Building e a Pattern Language, Christopher define um padrão de projeto e estabelece que um padrão de projeto deve apresentar as características de, encapsulamento, generalidade , equilíbrio abstração, abertura e combinatoriedade;

Na informática Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides também conhecidos como GOF (Ganf Of Four) iniciaram uma pesquisa baseada nos trabalhos de Christopher Alexandre para documentar os problemas e soluções enfrentados na área de análise e desenvolvimento de software e o resultado da pesquisa foi o livro Design Patterns: Elements of Reusable Object-Oriented Softwares.

Cada padrão deve descrever um problema e uma solução para o mesmo de forma que esta solução pode ser reutilizada em outras situações onde for encontrado o mesmo problema.

Os padrões GOF como são conhecidos são divididos em três categorias Padrões Criacionais , Padrões estruturais e Padrões Comportamentais .

Nos padrões Criacionais estão os padrões abstract factory , builder , factory method , prototype , singleton . Nos padrões estruturais estão os padrões adapter, bride, composite, decorator, facade, flyweight é proxy . Nos padrões comportamentais estão os padrões chain of responsibility, command, interpreter , interator, mediator, memento, observer, state, strategy, template method, visitor.

Os padrões criacionais são ligados a fase de instanciar novos objetos.

Os padrões estruturais são os padrões ligados a composição de classes e objetos

Os padrões comportamentais são padrões ligados a forma de interação entre as classes e objetos.

## MVC

### Introdução ao MVC

O padrão de projeto MVC (Model-Viel\_Controller) surgiu em 1979 no projeto de Smalltalk na Xerox por Trygve Reenskaug que atualmente vem sendo largamente utilizado no desenvolvimento Web.  
  
A aplicação deste padrão permite aumentar o nível de abstração da aplicação e reduzir o nível de acoplamento entre os componetes. Este padrão determina que a aplicações deve ter uma separação em 3 níveis Model, View Controller.

#### Model

O model ou modelo é o domínio especifico em que a aplicação simula ou implementa. Nele fica encapsulada toda a lógica da aplicação. O model pode ser simples como um integer (para uma aplicação contador) ou uma String (para uma aplicação de um editor de texto) ou pode ser complexa com objetos que instância subclasses (para uma aplicação de cadastro de cliente que contem endereço ou endereços) [Achar exemplo melhor] [Glenn E. Krasner - Stphen T. Pope] são responsabilidades da model, encapsular o estado da aplicação, responder á consulta sobre seu estado, expor as funcionalidades da aplicação, notificar as visões registradas sobre mudanças.

#### View

Na view ou apresentação esta contida toda a apresentação da aplicação, ela faz requisições de dados ao modelo e apresenta os dados representados pelo Model.  
Todas as saídas apresentas pelo sistema é visualizado somente pela view.

#### Controller

O Controller ou controlador atua como interface realizando a associação do model com a view.  
  
  
[Colocar a figura do artigo receita de bolo ou fazer um desenho mais bonito]  
  
  
Desde que o MVC foi apresentado muitas variações surgiu o mais comum é o MVC da forma que é apresentado e o MVC2 que tem como principal diferença a responsabilidade sobre a navegação na view, onde o no MVC a View é responsável por chamar as outras view, enquanto que no MVC2 a responsabilidade da navegação é transferida para o controller.

### FrameWork

#### Introdução ao Framework

O Framework é uma implementação utilizada em diversos projetos, provendo funcionalidades genéricas.

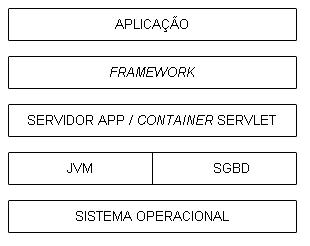
O uso de framework nos projetos traz vantagens como maior produtividade, redução do número de erros na codificação.

Os frameworks se diferencia dos padrões de projeto, por ser uma implementação real, enquanto o padrão de projeto é apenas uma idéia para resolução de um problema, um framework pode solucionar diversos problemas de um domínio, tentando capturar uma ou mais funções comum a várias aplicações, enquanto um padrão de projeto é especifico a um único problema, o framework pode implementar nenhum, um ou vários padrões de projetos, embora seja indicado que os framework utilize esta ferramenta.

Para que uma aplicação seja considerada um framework ela deve possuir as capacidade de ser reusável, extensível, seguro e eficiente.

Uma aplicação pode ser utilizada nenhum, um ou vários frameworks.

Na figura XXX é possível verificar a arquitetura básica de uma aplicação Java Web baseada em frameworks.



Figura

Fonte: <http://www.frameworkdemoiselle.gov.br/menu/framework/sobre/framework-definicao-1>

Vista em 17/09/09 as 23:17

# Persistência de dados

## Introdução Persistência de dados

A persistência se refere ao processo de armazenamento e captura em qualquer dispositivo físico que permite a recuperação posterior dos mesmos, como por exemplo, arquivo ou um SGDB.

## Transações

Transações se referem a um ou um conjunto de instruções que devem ser executadas como sendo um único trabalho, sendo executadas completamente ou deve ser totalmente cancelada inclusive as partes que foram executadas com sucesso, não podendo ficar com uma parte pendente.

Um exemplo de transação seria a transferência de valores entre contas de um banco, onde a transferência só poderá acontecer se for realizado o débito da conta de origem e o crédito na conta de destino, caso alguma das duas transações não for executadas com sucesso, todas devem ser anuladas.

## JDBC

### Introdução ao JDBC

O Java DataBase Connection ou em português conexão a base de dados do Java popularmente conhecido pela sigla JDBC é uma API de acesso a banco de dados para o Java desenvolvida pela Sun para acesso a banco de dados, que permite estabelecer uma conexão com uma base de dados, enviar instruções SQL e processar os resultados.

O JDBC realiza seu trabalho através de um conjunto de interfaces Java, implementado de forma diferente por cada fornecedor do SGBD.

O conjunto de classes que implementam as interfaces do JDBC são chamados de drivers, esta implementação é necessária para realizar a conexão a um banco de dados.

Na figura XX classes e interfaces do JDBC API pode ser visto o diagrama de classe das classes e interfaces básicas do JDBC.

|  |
| --- |
| Criar o diagrama de classes  ref. Livro Database programming with jdbc and Java (hemo ) pagina 29 |

Figura : Diagrama de classes e interfaces básicas do JDBC

Na construção de aplicativo com Java que acessem bancos de dados utilizando o JDBC, o desenvolvedor, não precisa pensar nas características específicas de cada banco de dados, pois todas as características especificas de cada SGDB é tratada pelo driver JDBC, permitindo que o programado se preocupe apenas com as necessidades da aplicação.

Na figura XX é possível ver a ilustração da arquitetura do driver JDBC.

|  |
| --- |
| Aplicação  B  Aplicação  A  JDBC  Postgres  MySQL  Oracle |

Figura : Arquitetura JDBC

Os drivers JDBC são classificados em quatro categorias: JDBC-ODBC, API-Nativo, Driver de Protocolo de rede, Driver Nativo.

O driver JDBC\_ODBC é um gerenciador de Driver ODBC instalado localmente para conexão com o banco de dados.

Para utilizar este driver é necessário configurar o Gerenciador de driver ODBC.

O Driver Híbrido, é um driver escrito parcialmente em Java e parcialmente em C. Este driver geralmente necessita de uma API cliente especifica para o banco de dados.

O Driver nativo usa um protocolo de rede e este driver é escrito totalmente em Java, mas ele não se conecta ao banco, ele se conecta a uma camada intermediária chamada de middleware que atua como um tradutor entre o driver e o banco.

O Driver nativo é escrito em Java, e usa o protocolo de rede do servidor de banco de dados conectando diretamente com o mesmo, normalmente este driver é fornecido pelo fabricante do banco de dados.

Um sistema Java, que utiliza o JDBC, geralmente percorre o seguinte fluxo, abrir uma conexão com o servidor de banco de dados, envia uma conexão SQL ao servidor e no final processa as resposta enviadas do banco.

Na figura XX pode ser visto como os driver interage com o SGBD.

|  |
| --- |
| Desenhar figura do livro  ref. Livro Database programming with jdbc and Java (hemo ) pagina 31 |

Figura : Integração do JDBC

No site <http://developers.sun.com/product/jdbc/drivers> é possível verificar uma lista de fornecedores e SGDB suportados pelos drivers desenvolvidos pelos mesmos.

### Acessados banco de dados com JDBC

#### Criado a conexão

Para realizar uma conexão com o banco de dados, o sistema deverá importar a classe JDBC. Registrar o driver, chamar o método getConnection informando a URL, database, nome do usuário do banco e a senha do mesmo.

O registro do driver é realizado adicionando a instrução:

|  |
| --- |
| Class.forName(“nomeDoDriver”); |

Figura : registro de driver JDBC

Na tabela, classes de drivers JDB é possível verificar algumas classes de driver JDBC.

|  |  |
| --- | --- |
| Banco de Dados | Classe do Driver |
| PostgreSQL | postgreslq.Driver |
| MySQL | Com.mysql.jdbc.Driver |
| Oracle | Oracle.jdbc.driver.OracleDriver |
| Sysbase | Com.sybase.jdbc2.jdbc.SybDriver |

Tabela : Classes de driver JDBC

O método que abre a conexão com o banco getConnection deve ser chamado com a seguinte instrução:

|  |
| --- |
| DriverManager.getConnection("URL do banco ", "usuário", "senha"); |

Figura : URL JDBC

Na tabela URL JDBC pode ser visto alguns exemplos de URL JDBC

|  |  |
| --- | --- |
| Banco de Dados | URLs JDBC |
| PostgreSQL | jdbc:postgresql://host:porta/database |
| MySQL | jdbc:mysql://host:porta/database |
| MiniSQL | jdbc:mysql://host:porta/database |
| Oracle | jdbc:oracle:thin:@host:porta:SID |
| SysBase | Jdbc:sysbase:tds:host:porta/SID |

Figura : URLs para os principais bancos de dados

A tabela URL JDBC mostra as URLs JDBC para diversos bancos onde:

Host é o nome do computaodor no qual o servidor de banco de dados esta instalado. Caso o mesmo seja omitido na URL a maioria dos drivers assume o endereço como localhost (127.0.0.1) com padrão e isso vai fazer com que a busca seja realizada na mesma máquina onde o sistema está em execução.

Database é o nome do banco de dados que precisa estabelecer a conexão.

SID é um identificador do sistema Oracle que pode ser verificado no arquivo oratab geralmente encontrado no /etc (em sistemas POSIX).

Porta é a porta TCP/IP que o SGDB esta recebendo conexão. Caso a mesma seja omitida os drivers assumem que a porta padrão do banco será assumida.

Exemplo de uma classe que conecta ao um banco de dados.:

|  |
| --- |
| refazer baser no livro “JDBC and Java pagina 38”  public class ConnectionFactory {  public static Connection getConnection() throws SQLException {  try {  Class.forName("com.mysql.jdbc.Driver");  return DriverManager.getConnection("jdbc:mysql://localhost/teste","root","");  } catch (ClassNotFoundException e) {  throw new SQLException(e.getMessage());  }  }  } |

Figura : Exemplo de conexão JDBC

O método getConnection, foi colocado dentro da instrução try catch pois caso o mesmo não encontre o driver no classpath irá lançar uma exceção do tipo ClassNotFoundException.

Figura : JDBC com singleton

#### Enviando comando SQL ao Servidor

O envio de comando SQL ao SGDB pode ser realizado por meio do objeto do tipo Statement t, este método deve ser chamado entre a instrução try catch devido ao mesmo poder gerar uma exceção do tipo SQLException caso exista algum erro no camando SQL enviado.

Exemplo:

|  |
| --- |
| Statement stmt = null;  Try{  stmt = ConnectionFactory.getConnection();  stmt.excuteUpdate(SQL);  } catch(SQLException e){  System.err.println(e.getMessage() );  } finally{  stmt.close();  } |

Figura : Tratar Exception

Note que a variável stmt recebe a conexão vinda do método getConnection na classe Connection, em seguida executa a instrução SQL que estiver na variável do tipo string SQL.

#### Preparando o SQL

O método preparedStatement é uma sub-interface de Statement e representa uma declaração SQL pré-compilada. Esta declaração pode incluir parâmetros que facilitam a criação de instruções SQL, pois permite alterar os parâmetros sem reescrever, mudando seus valores a cada chamada.

Os valores podem ser aplicados na string que guarda o comando SQL enviando o comando a um objeto do tipo PreparedStatement e no local onde deveria estar os valores o mesmo deve ser substituído por um ponto de interrogação para cada campo, posteriormente o objeto do tipo PreparedStatement deve receber os parâmetros necessários pelos metódos set mais o tipo de informação que deseja, por exemplo, para inserir String setString.

Os parâmetros devem ser passados por estes métodos, pois são neles que é realizado o cast ou seja a conversão dos tipos de Java para SQL.

## JPA

### Introdução ao JPA

O Framework Java Persistence API ou simplesmente JPA foi definida na JSR-220 (Entreprise Java Beans, Versão3), para padronizar o mapeamento de objeto/relacional na plataforma Java. Apesar de estar descrita na especificação do EJB, na versão 3 a JPA não depende do container para funcionar, sendo possível usar em soluções com apenas o Java SE.

O JPA é baseada no conceito POJO (Plain Old JavaObject) e incorpora idéias do framework Hibernate para padronizar o mapeamento Objeto/Relacional em Java.

O mapeamento objeto/relacional é um mapeamento automatizado e transparente de persistência de objetos da aplicação Java, em tabelas de um banco de dados relacional, usando metadados para descrever o mapeamento entre os objetos e o banco de dados.

Com o JPA podemos utilizar as idéias criadas para o hibernate sem ficar amarrado a um fornecedor, pois sendo o JPA uma especificação outros, fabricantes puderam apresentar alternativas de implementação do JPA, como por exemplo, o TopLink e o EclipseLink.

### Entidade

Uma entidade no JPA é uma classe Java estilo POJO que possui a anotação @Entity que representa uma tabela no banco de dados.

Na figura XXX é possível ver um exemplo de uma entidade.

|  |
| --- |
| import Javax.persistence.Entity;  @Entity  public class Pesquisador implements Serializable {  private static final long serialVersionUID = 1L;  @Id  private Long id;  private String nome;  private int idade;  //Gets and Sets |

Figura : Entidade

Na tabela XXX pode ser visto todas as anotações possíveis no JPA para a Entidade.

|  |  |
| --- | --- |
| Anotação | Descrição |
| @Entity | Indica que a classe representa uma entidade, que nada mais é que um POJO, representando um objeto de negócio. |
| @Table | Mapeia o nome da tabela, a omissão deste parâmetro faz com que o nome da classe seja utilizado. |
| @Id | Identificador da entidade(Chave primária da tabela). |
| @GeneratedValue | Define que terá seus identificadores automaticamente gerados. |
| @Column | Mapeia o nome da coluna existente na tabela relacional. |
| @Temporal | Identifica o tipo com valores relativo a tempo. |
| @Lob | Identifica o tipo com valor relativo a textos com grande quantidade de caracteres. |
| @NamedQueries | São usadas para agrupar as consultas existentes na entidade. |
| @NamedQuery | Define as consultas declaradas na anotação @NamedQueries. |
| @Transient | Indica que o campo não deve ser persistido em banco de dados. |
| @basic | Define a estratégia de fetch para uma propriedade de uma classe. |
| @Enumerated | Define a propriedade como do tipo enumerado. |
| @Version | Permite declara uma propriedade para controlar lock otimista |

Figura : Anotações JPA para entidade

Fonte: livro Dominando o Java Server faces e facelets spring e jpa pg 104 e 105

Os mapeamentos de associações também são realizado por anotações, na tabela XXX pode observar as anotações para realizar os mapeamentos.

|  |  |
| --- | --- |
| Anotação | Descrição |
| @OneToOne | 1:1  um para um. |
| @OneToMany | 1:N ou 1:\*  Um para vários. |
| @ManyToOne | N:1 ou \*:1  Vários para um |
| @ManyToMany | N:N ou \*:\*  Vários para vários |
| @Embeddable | Classes Embutidas. |
| @Embeddedid | Chaves composta. |

Figura : Anotações JPA Relacionamento

As anotações para realizar a herança, podem ser vista na tabela XXXX

|  |  |
| --- | --- |
| Anotação | Descrição |
| @Inheritance | Indica o mapeamento de herança que será realizado, este pode ser JOINED, SINGLE\_TABLE ou TABLE\_PER\_CLASS |
| @DistriminatorValue | Indica qual o campo será usado para realizar o mapamento da herança. |

Figura : Anotações JPA herança

### Inserir

O método responsável por realizar a inserção dos dados no JPA é persist que retorna uma instância do objeto persistido, na figura XXX pode ser visto uma função de exemplo que persiste um objeto.

|  |
| --- |
| public static void persist(Object object) {  EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceUnitPU");  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  try {  em.persist(object);  em.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  em.getTransaction().rollback();  } finally {  em.close();  }  } |

Figura : Inserir JPA

### Remover

O método para remover um objeto com o JPA é o remove, o uso é similar ao método persist usado para inserir um objeto, porém este método não retorna nenhum valor.

Na figura XXX pode ser visto um exemplo do seu uso.

|  |
| --- |
| public static void persist(Object object) {  EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceUnitPU");  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  try {  em.remove(object);  em.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  em.getTransaction().rollback();  } finally {  em.close();  }  } |

Figura : Remover JPA

### Atualizar

A atualização dos dados se dá pelo método merge, que retorna uma instância do objeto atualizado. Na figura XXX pode ser visto um exemplo do uso.

|  |
| --- |
| public static void Update(Object object) {  EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceUnitPU");  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  try {  em.merge(object);  em.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  em.getTransaction().rollback();  } finally {  em.close();  }  } |

Figura : Remover JPA

### Pesquisar

A pesquisa pode ser realizada pelo ID, com o método find, ou por meio de uma query que pode retornar um ou uma lista de registros.

Nas figuras XXX, XXX e XXX pode ser visto um exemplo de como realizar uma consulta por um ID, por uma query retornando um registro, ou por uma query retornando uma lista de registros.

As querys em JPA podem ser criadas utilizando a linguagem SQL ou pela linguagem JPQL (Java Persistence Query language) que é a linguagem de consulta primária do JPA por manter a independência com o banco de dados.

Sua sintaxe é semelhante a linguagem SQL diferenciando fundamentalmente na seleção de uma tabela, pois nesta linguagem é a entidade que é especificada.

Na figura XXX pode ser visto um exemplo da mesma consulta realizada em SQL e JPQL.

|  |  |
| --- | --- |
| SQL | JPQL |
| SELECT \* FROM tabela; | SELECT Object(t) from Tabela as t; |

Tabela : Comparativo SQL VS JPQL

A criação da query é realizada com o método createQuery(), de EntityManager passando como parâmetro a query, ou com o createNamedQuery() passando como parâmentro o nome da query.

A interface EntityManager é responsável pelas operações de persistência, os outros métodos utilizados nas buscas são os setFirstResult() e setMaxResults() que são respectivamente, o primeiro registro a ser retornado na busca e o máximo de resultado que a consulta deve retornar.

|  |
| --- |
| public Objeto findById(Integer id) {  EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceUnitPU");  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  try {  Objeto objeto = em.find(Objeto.class, id);  em.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  em.getTransaction().rollback();  } finally {  em.close();  }  return objeto;  } |

Figura : FindBy Id JPA

|  |
| --- |
| public Objeto find() {  EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceUnitPU");  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  try {  Objeto = em.createNamedQuery("Query").getSingleResult();  em.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  em.getTransaction().rollback();  } finally {  em.close();  }  return objeto;  } |

Figura : Find JPA

|  |
| --- |
| public List find() {  EntityManagerFactory emf = Persistence.createEntityManagerFactory("PersistenceUnitPU");  EntityManager em = emf.createEntityManager();  em.getTransaction().begin();  try {  List objetos = em.createNamedQuery("Query").getResultList();  em.getTransaction().commit();  } catch (Exception e) {  e.printStackTrace();  em.getTransaction().rollback();  } finally {  em.close();  }  return objetos;  } |

Figura : Find many JPA

# JavaServer Faces

## Introdução ao JavaServer Faces

O JavaServer Faces também conhecido como JSF é a tecnologia padrão para o desenvolvimento de aplicações Web da tecnologia J2EE 1.4 (ou superior), surgiu da experiência com as tecnologias Java Servlets, JavaServer Pages (JSP), com o objetivo de simplificar o desenvolvimento de aplicações Web, através do conceito baseado em componentes.

O JSF é um framework MVC, para o desenvolvimento de aplicações Web, que permite o desenvolvimento de aplicação Web usando um modelo RAD (Rapid Application Development ou Desenvolvimento rápido de aplicação) similar ao utilizado no desenvolvimento de aplicações desktops por anos como as aplicações desenvolvidas com Delphi ou Visual Basic, ou seja, arrastando e soltando os componente para a aplicação, definindo seus valores como propriedades.

## Ciclo de vida do JavaServer Faces

O ciclo de vida do JavaServer Faces é composto por fase que são:

1. Restore view (Recuperar a tela)
2. Apply request values (Aplicar valores do request)
3. Process validation (Processo de validação)
4. Update model values (Atualizar valores do modelo)
5. Invoke Application (Chamar a aplicação)
6. Render response (retornar resposta)

As seis fases estão descritas na ordem em que elas geralmente ocorrem, porém à ordem pode ser alterada pulando fases ou saindo do ciclo de vida do JSF, por exemplo, quando um valor é invalido, a visão atual seria mostrada e alguma das fases poderia não ser executadas, opcionalmente você poderia configurar um comportamento para sua aplicação para ser redirecionada para outra página para exibir uma mensagem de erro mais amigável ao usuário.

Observando a figura 21 é possível ver o fluxo do ciclo de vida do JSF padrão.

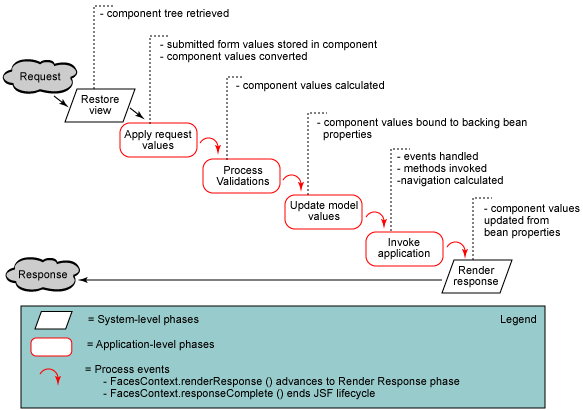


Figura : Ciclo de vida JSF

Figura 22: Ciclo de Vida do JSF

### As fases do ciclo de vida do JSF

#### Restore view

Recupera a árvore de componentes para a página solicitada se a mesma já foi exibida anteriormente, caso seja a primeira vez que este evento ocorra então é construída uma nova árvore de componente.

Todos os componentes são definidos conforme seu estado anterior, desta forma o JSF mantém automaticamente as informações atribuídas. Por exemplo, caso o usuário tenha preenchido um formulário com alguma informação não válida, ao visualizar novamente a página todas as entradas são exibidas novamente para que o usuário possa corrigi-las.

#### Apply request values

Para cada componente seus valores são recuperados, estes valores podem ser obtidos por request (get), header (post) ou cookies.

#### Process Validation

Nesta fase cada componente irá ter seus valores validados.

As regras a serem aplicadas na validação podem ser pré-definidas ou definidas pelo desenvolvedor;

#### Update model values

Atualiza os valores atuais do modelo no servidor, através do Backing Beans que são as classes que liga o jsf com as classes com as regras de negócio.

#### Invoke application

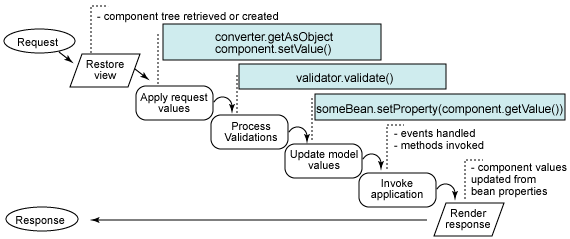
Chama os listeners de ação e as ações para os componentes de comando.

#### Render response

Salva o estado e carrega a visão seguinte.

### Conversão, validação e evento

Durante o ciclo de vida do JSF pode ocorrer conversão, validação e eventos que podem ser vistos na figura 22.

Figura 23: Ciclo de Vida do JSF com conversão, validação e evento

## Java Beans

### Introdução ao JavaBeans

Segundo o site [http://Java.sun.com/Javase/technologies/desktop/Javabeans/index.jsp](http://java.sun.com/javase/technologies/desktop/javabeans/index.jsp) JavaBeans pode ser definido com um componente de arquitetura da plataforma Java 2, reusáveis que facilita a criação de aplicações sofisticadas.

O JavaBeans pode ser visto, com classes Java que possuem atributos com métodos gets e sets, e outros métodos. Com o uso do JavaBeans é possível separar a interface da lógica de negócio.

Os JavaBeans são largamente utilizados para o desenvolvimento Web, mas sua concepção inicial se deu na reutilização em programação visual, os famosos “Beans” do swing.

### Exemplo de um JavaBeans

|  |
| --- |
| package meupacote;  public class MeuBean {  private String nome;  private int idade;  public int getIdade() {  return idade;  }  public void setIdade(int idade) {  this.idade = idade;  }  public String getNome() {  return nome;  }  public void setNome(String nome) {  this.nome = nome;  }  public MeuBean() {  }  } |

Figura : JavaBeans

O JavaBeans criado possui dois atributos e seus respectivos métodos gets e sets.

### Tipos de escopo do JavaBeans

#### Introdução ao escopo dos JavaBeans

O escopo do JavaBeans vai definir o tempo de existência do mesmo na aplicação, após o mesmo ser criado.

O escopo do JavaBeans no JSF pode ser definido como, request, session, application e none.

##### Escopo request

No escopo request, os dados são mantidos até o final da requisição, ou seja, desde o momento em que a solicitação a uma página é realizada até que a página seja enviada para o cliente.

Declarar um JavaBean com escopo request equivale a instanciar um objeto da seguinte forma:

|  |
| --- |
| meuBean = new meuBean();  request.setParameter(“meuBean”,meuBean); |

Figura : Escopo request

##### Escopo session

O escopo session será visível em toda a sessão, tornando possível acessar seus dados de qualquer página JSP que participa da sessão com o usuário.

A sessão é criada no momento que o cliente faz a primeira solicitação ao servidor e fica disponível enquanto o cliente estiver com a mesma sessão do navegador aberta, caso o cliente fecha e abre novamente o navegador, uma nova sessão é iniciada.

Declarar um JavaBean com escopo session equivale a instanciar um objeto da seguinte forma:

|  |
| --- |
| if (request.getParameter("meuBean") != null) {  MeuBean meuBean = new meuBean();  request.setParameter(“meuBean”,meuBean);  } else {  MeuBean meuBean = request.getParameter("meuBean");  } |

Figura : Escopo session

##### Escopo application

O escopo application torna-se visível por toda a aplicação, para todas as páginas JSP e Servlets no sistema, independente do usuário e ou navegador.

Declarar este escopo com session, equivale a instanciar um objeto da seguinte forma:

|  |
| --- |
| MeuBean meuBean = request.getParameter("meuBean"); |

Figura : Escopo application

##### Escopo none

O escopo none, não tem nenhum dado salvo nos outros escopos. Ele existirá somente no tempo de vida que sua JSP é formada e mais nada.

O escopo none, deve ser utilizado quando o JavaBean com este escopo é chamado por outro JavaBean de outro escopo, ou seja, ele somente existirá na aplicação no momento de sua utilização não armazenando ou salvando nenhum dado ou referência a seu respeito.

Declarar um JavaBean com escopo none equivale a criar uma nova instância cada vez que algum método for acessado.

#### Configurando o JavaBeans

A configuração do JavaBeans no JSF é realizada no arquivo faces-config.xml, adicionando, o nome, a classe e definindo o escopo do mesmo.

Sintaxe:

|  |
| --- |
| <managed-bean>  <managed-bean-name>Nome</managed-bean-name>  <managed-bean-class>classe</managed-bean-class>  <managed-bean-scope>Escopo</managed-bean-scope>  </managed-bean> |

Figura : Configuração do JavaBeans

No exemplo da figura 28 o Nome é o nome que será acessado posteriormente, classe é a classe que o JavaBean representa, contendo o caminho completo da mesma (os pacotes) e Escopo é o escopo do mesmo.

Exemplo:

|  |
| --- |
| <managed-bean>  <managed-bean-name>meuBean</managed-bean-name>  <managed-bean-class>pacote.MeuBean</managed-bean-class>  <managed-bean-scope>session</managed-bean-scope>  </managed-bean> |

Figura : Configuração do JavaBeans - exemplo

### Definir propriedades no JavaBeans

Com o JavaBeans é possível se necessário definir propriedades ao mesmo, esta definição é realizada no arquivo de configuração faces-config.xml, adicionando as tags managed-property, property-name e value, com seus respectivos valores.

Exemplo:

|  |
| --- |
| <managed-bean>  <managed-bean-name>meuBean</managed-bean-name>  <managed-bean-class>pacote.meuBean</managed-bean-class>  <managed-bean-scope>session</managed-bean-scope>  <managed-property>  <property-name>nome</property-name>  <value>Marcelo</value>  </managed-property>  </managed-bean> |

Figura : Definir propriedades para o JavaBeans

Para este JavaBean monstrado na figura 30, foi adicionado a propriedade nome, com o valor Marcelo sendo neste caso obrigatório que a classe do JavaBean tenha o atributo nome do tipo String e os métodos get e set para este atributo, sempre que o mesmo for iniciado vai ter o atributo nome instanciado com o valor Marcelo.

Uma outra opção é permitir que um JavaBean acesse o valor de outro JavaBeans para isto, deve definir a propriedade de um bean com o valor de outro, por exemplo:

|  |
| --- |
| <managed-bean>  <managed-bean-name>meuBean</managed-bean-name>  <managed-bean-class>pacote.meuBean</managed-bean-class>  <managed-bean-scope>session</managed-bean-scope>  <managed-property>  <property-name>outroBean</property-name>  <value>#{outroBean}</value>  </managed-property>  </managed-bean> |

Figura : Propriedades do JavaBeans Exemplo

Para este JavaBean a classe que implementa o JavaBean deve ter obrigatório o atributo outroBean do mesmo tipo que o bean outroBean, e os métodos get e set para este atributo.

## Navegação

### Introdução a navegação

É comum que os sistemas Web contenham links para outras páginas do mesmo sistema e para outros sistemas, no JSF os links para páginas dentro do sistema são feitos com a chamada navegação, que pode ser estática ou dinâmica.

A navegação consiste na seleção de outra view, página JSF por exemplo, sendo a estática se caracteriza por direcionar sempre para o mesmo alvo, enquanto que na dinâmica o alvo pode variar dependendo dos dados informados.

Tanto a navegação estática, quanto a dinâmica pode ser um forward ou um redirect. Esta navegação pode acontecer por meio de um link ou um botão.

O Forward é executado pelo servlet, o navegador não sabe o que está acontecendo por isso a URL é preservada, todo o reload da página será executada na requisição original.

O Redirect realiza uma nova requisição ao servidor, por isso a URL muda e o reload da página será executado em uma nova requisição, este processo é mais lento que um forward.

### Criando a navegação

A configuração da navegação é realizada no arquivo faces-config.xml, incluindo as tags navigation-rule que contém a tag from-view-id e as navigation-case que contém as tags description, from-action, from-outcome, to-vire-id e redirect.

A tag navigation-rule informa que está iniciando a configuração da navegação, from-view-id é para informar de qual view, deve vir a ação, esta opção pode ser omitida fazendo com que a regra seja válida para qualquer origem, a tag navigation-case vai agrupar os caso de navegação e contém a descrição, cujo objetivo é apenas para documentação é pode ser omitida, from-action indica de qual método deve vir a solicitação, caso seja omitida será valida para todos os métodos desta forma o retorno da navegação será sempre o mesmo caracterizando uma navegação estática, from-outcome é a string de retorno que deve ser recebida para que a regra seja válida, to-view-id é onde deve ser informado a view que deseja ser exibida após esta execução e redirect que vai indiciar que a navegação deve ser do tipo redirect, a omissão desta tag irá fazer com que o servidor realize um forward.

Uma navigation-rule pode ter uma ou várias navigation-case.

Exemplo:

|  |
| --- |
| <navigation-rule>  <from-view-id>origem.jsp</from-view-id>  <navigation-case>  <description>descrição da regra de navegação</description>  <from-action>#{meuBean.metodoDeNavegacao}</from-action>  <from-outcome>stringDeRetorno</from-outcome>  <to-view-id>destino.jsp</to-view-id>  <redirect/>  </navigation-case>  </navigation-rule> |

Figura : Regras de navegação

## UI Componentes

UI componentes são elementos que compõem a interface da aplicação, são configuráveis e reutilizáveis, podendo ser simples como um botão ou composto como uma tabela.

Todo componente implementa a interface UIComponetBase que implementa as seguintes interfaces ActionSource ValueHolder StateHolder NamingContainer EditableValueHolder.

ActionSource, permite o componente dispara eventos de ação.

ValueHolder faz com que o componente mantenha um valor local para acessar o seu modelo com conversão de tipos.

StateHolder salva os estados dos objetos entre os requests.

NamingContainer faz com que os componentes tenham IDs unicos.

EditableValueHolder aplicada para componentes editáveis, herda de StateHolder e adiciona suporte à validação e a emissão de value-change events.

### Árvore de Componentes

A árvore de componentes é formada pelas tags dos componentes onde cada tag é associada a um handler class.

Quando a página é lida as tag handler são executadas.

Exemplo de uma árvore de componentes.

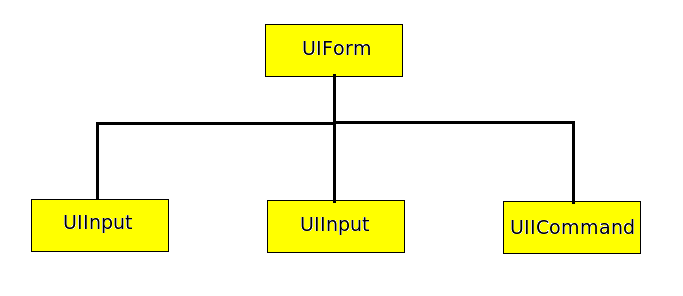


Figura : Árvore de componentes

Esta seria uma página com um formulário, dois campos de entrada de texto e um botão.

## Componentes padrões do JSF

Os componentes padrões do JSF são divididos em dois grupos, core e html, no grupo dos componentes core estão os componentes que são renderizados independente da tecnologia, neste grupos estão os componentes actionListener, attribute, convertDateTime, convertNumber, converter, facet, loadBundle, param, phaseListener, selectItem, setPropertyActionListener, subview, validateDoubleRange, validateLength, validateLongRange, validator, valueChangeListener, verbatim, view. Os componentes do grupo html são os componentes que representa as tags HTML, que são column, commandButton, commandLink, dataTable, form, graphicImage, inputHidden, inputSecret, inputText, inputTextarea, message, messages, outputFormat, outputLabel, outputLink, outputText, panelGrid, panelGroup, selectBooleanCheckbox, selectManyCheckbox, selectManyMenu, selectOneListbox, selectOneMenu, selectOneRadio.

### Introdução aos validadores

Os validadores vão validar as entradas dos usuários na tela.

Os validadores existentes são: validateLength, validateDoubleRange e validateLongRange.

### Criando um validador

Para criar um validador deve-se criar uma classe que implementa a interface Validator e o método validate, que será os métodos a ser executado quando a validação for invocada.

Dentro do método podem realizar qualquer teste lógico, caso o valor não estava valido para criar a mensagem de erro, deve criar um objeto do tipo FacesMessage e adicionar a mensagem de erro a ele e lançar uma exceção do tipo ValidatorException passando como argumento ao construtor o objeto do tipo FacesMessage criado.

Exemplo, um validador que sempre gera erro:

|  |
| --- |
| import Javax.faces.application.FacesMessage;  import Javax.faces.component.UIComponent;  import Javax.faces.context.FacesContext;  import Javax.faces.validator.Validator;  import Javax.faces.validator.ValidatorException;  public class MeuValidador implements Validator {  public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {  FacesMessage messages = new FacesMessage("Campo inválido");  throw new ValidatorException(messages);  }  } |

Figura : Validador JSF

O validador precisa ser configurado no faces-config.xml, para isso deve adicionar as tags validator, para conter a configuração do validador, validator-id para informar o id do validador e validator-class que deve contar o nome da classe que implementa o validador com o pacote.

Exemplo:

|  |
| --- |
| <validator>  <validator-id>meuValidador</validator-id>  <validator-class>pacote.MeuValidador</validator-class>  </validator> |

Figura : Validador JSF exemplo

### Usando um validador

Para associar um campo a um validador, deve-se incluir o componente f:validator informando o id do validador em validatorId entre as tags de entrada do texto.

Exemplo

|  |
| --- |
| <h:inputText >  <f:validator validatorId="meuValidador" />  </h:inputText> |

Figura : Validador JSF uso

## Conversores

### Introdução aos conversores

Os conversores são classes para converter os tipos dos dados, os conversores nativos do JSF converte de string para BigDecimalConverter, BigIntegerConverter ,BooleanConverter, ByteConverter, CharacterConverter, DateTimeConverter, DoubleConverter, FloatConverter, IntegerConverter, LongConverter, NumberConverter e ShortConverter ou personalizado.

Um exemplo de conversor personalizado poderia ser uma entrada do número de CPF.

### Criado um conversores

Para criar um conversor deve criar uma classe que implementa a interface Converter e os métodos getAsObject e getAsString que recebem os seguintes parâmetros: context do tipo FacesContext, component do tipo UIComponent e value do tipo Object, retornando respectivamente um objeto do tipo Object e String.

Exemplo:

|  |
| --- |
| import Javax.faces.component.UIComponent;  import Javax.faces.context.FacesContext;  import Javax.faces.convert.Converter;  public class MeuConversor implements Converter {  public Object getAsObject(FacesContext context, UIComponent component, String value) {  //aqui faz a conversão  return value;  }  public String getAsString(FacesContext context, UIComponent component, Object value) {  //aqui faz a conversão  return (String) value;  }  } |

Figura : Conversor JSF

O conversor precisa ser configurado no faces-config.xml, adicionando as tags converter, que vai conter as tags converter-id e converter-class que vão ter respectivamente o id do conversor e a classe com o pacote que implementa o conversor.

Exemplo:

|  |
| --- |
| <converter>  <converter-id>meuConversor</converter-id>  <converter-class>meupacote.MeuConversor</converter-class>  </converter> |

Figura : Declaração de conversor

### Usando um conversor

Para converter o valor de um campo de String para outro tipo, o componente converter deve ser incluído entre as tags de do componente de entrada de texto, informando na propriedade validator-Id o id do conversor que deseja utilizar.

Exemplo:

|  |
| --- |
| <h:inputText >  <f:validator validatorId="meuValidador" />  </h:inputText> |

Figura : Exemplo de declaração de conversor

## Internacionalização

### Introdução a internacionalização

O JSF tem suporte a internacionalização, fazendo com que seja possível, visualizar um sistema em várias línguas.

### Adicionando internacionalização a uma aplicação

No projeto deve ser adicionado o arquivo com a extensão properties contendo o uma chave de acesso o sinal de igual seguido pelo valor da chave para cada língua que deseja adicionar suporte.

Na figura XXX pode ser visto um exemplo de um arquivo .properties

|  |
| --- |
| name=nome  password=senha  age=idade |

Figura : Arquivo properties

No arquivo faces-config.xml deve configurar os arquivos com suporte as línguas, na figura XXX pode ser visto um exemplo de como configurar o faces-config.xml para suportar a língua portuguesa e inglesa, para o arquivo mensagens.properties

|  |
| --- |
| <application>  <message-bundle>mensagens</message-bundle>  <locale-config>  <default-locale>pt\_BR</default-locale>  <supported-locale>pt\_BR</supported-locale>  <supported-locale>en\_US</supported-locale>  </locale-config>  </application> |

Na página, JSF deve ser adicionado o a tag para carregar o suporte a internacionalização loadBundle.

Na figura XXX pode ser visto um exemplo de uma página JSF com suporte a internacionalização.

|  |
| --- |
| <%@page contentType="text/html"%>  <%@page pageEncoding="UTF-8"%>  <%@taglib prefix="f" uri="http://Java.sun.com/jsf/core"%>  <%@taglib prefix="h" uri="http://Java.sun.com/jsf/html"%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  "http://www.w3.org/TR/html4/loose.dtd">  <f:view>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Pagina com suporte a internacionalização</title>  </head>  <body>  <f:loadBundle basename="mensagens" var="msgBundle"/>  <h:outputText value="#{msgBundle.titulo}" />  </body>  </html>  </f:view> |

Figura : Página JSF com internacionalização

## PhaseListener

### Introdução ao PhaseListener

Entre as fases é possível implementar eventos, que pode ocorrer antes e depois da fase chamando respectivamente os métodos beforePhase e afterPhase.

## Implementação do PhaseListener

Para construir a classe que vai conter os métodos a serem executados, a mesma deve implementar a interface PhaseListener, e os metodos afterPhase, beforePhase getPhaseId, onde os metodos afterPhase e beforePhase recebem um objeto do tipo PhaseEvent e não tem retorno e o getPahserID não recebe nenhum valor e retorna um objeto do tipo PhaserId.

Exemplo de uma que implementa a interface PhaserListener.

|  |
| --- |
| import Javax.faces.application.NavigationHandler;  import Javax.faces.context.FacesContext;  import Javax.faces.event.PhaseEvent;  import Javax.faces.event.PhaseId;  import Javax.faces.event.PhaseListener;  import Javax.servlet.http.HttpSession;  public class xerife implements PhaseListener {  public void afterPhase(PhaseEvent event) {  System.out.println("Xerife em ação – Depois da fase");  }  public void beforePhase(PhaseEvent event) {  System.out.println("Xerife em ação – Antes da fase");  }  public PhaseId getPhaseId() {  return PhaseId.ANY\_PHASE;  }  } |

Figura : PhaserLister

A fase que o evento vai ocorrer depende do valor retornado no método getPhaserID, que pode retornar um PhaserId da seguintes formas:

PhaseId.ANY\_PHASE, irá executar em todas as fases passiveis;

PhaseId.APPLY\_REQUEST\_VALUES, irá executar na fase apply request values (aplicar valores de requisição);

PhaseId.INVOKE\_APPLICATION, irá executar na fase invoke application (invocar a aplicação);

PhaseId.PROCESS\_VALIDATIONS, irá executar na fase process validations (processar validações);

PhaseId.RENDER\_RESPONSE, irá executar na fase render response (renderizar resposta);

PhaseId.RESTORE\_VIEW, irá executar na fase restore view( restaurar visão) ;

PhaseId.UPDATE\_MODEL\_VALUES, irá executar na fase update model values (atualizar valores do modelo);

No faces-config.xml deve ser inserida as tags lifecycle que vai conter as tags phaser-listener com nome da classe que implementa a interface PhaseListener com o pacote.

Exemplo:

|  |
| --- |
| <lifecycle>  <phase-listener>pacote.xerife</phase-listener>  </lifecycle> |

Figura : Declaração do phaser-Listener

# Projeto

## Resumo do projeto

O projeto Guaraba Wally é um sistema que vista permitir que as pessoas utilizem seus celulares para localizar produtos.

Os produtos a ser localizado precisam ser cadastrados na página do projeto.

## Problema abordado

Uma aplicação pode ser desenvolvida de diversas maneiras apresentando sendo que cada uma apresenta vantagens e desvantagens quando comparada uma com a outra.

O problema abordado foi a busca por uma metodologia para construção do projeto utilizando dois dos principais frameworks da atualidade, o JavaServer Faces e o JPA.

# Resultados potenciais teóricos e práticos da monografia

Uma aplicação que integra as tecnologias JSF e JPA

# Referências Bibliográficas

Gonalves, Edson Desenvolvendo Aplicações Web com netBeansIDE 5.5. ed. Ciência Moderna, 2007.

Gomes, Yuri Marx P. Java na Web com JSF, Sptring, Hibernate e Netbeans 6. ed Ciência Moderna, 2008.

Revista Clube dephi + PHP ano 8 edição 102.

Livro do Head first design pattern

O outro livro

# Referências Bibliográfias

Freeman, E., Freeman, E., Sierra, K., & Bates, B. *Head First Design Patterns.* O Relly.