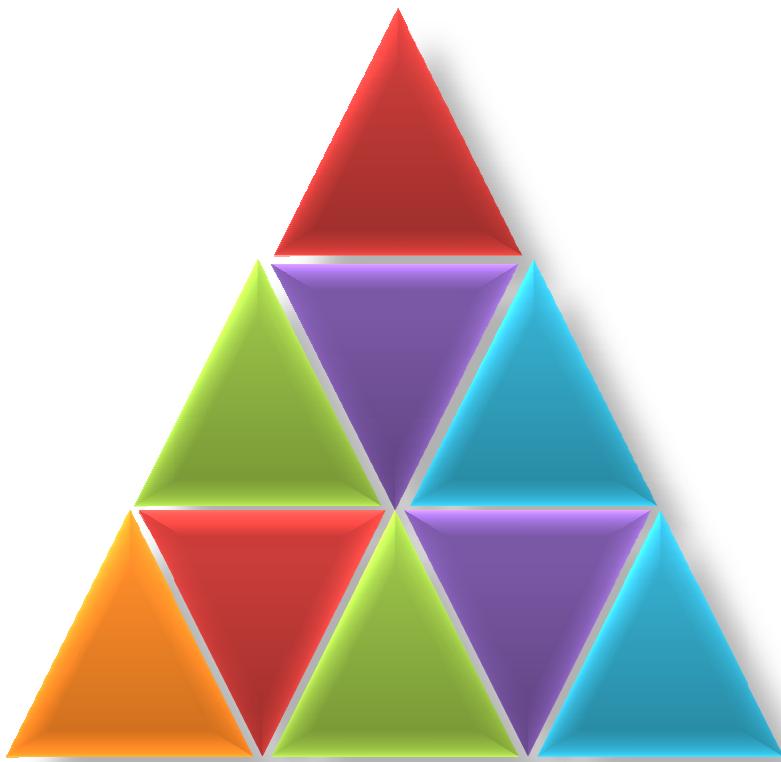


JAVA PARA WEB NA PRÁTICA

Construindo aplicações utilizando JSF, JPA e Primefaces



ROSICLÉIA FRASSON

PREFÁCIO

Este material foi escrito com o intuito de facilitar a consulta e o acompanhamento da disciplina de Programação Web, tendo como premissa explicar em mais detalhes temas abordados em sala de aula.

Neste material podem ser encontrados conceitos e dicas necessárias ao desenvolvimento de uma aplicação completa utilizando JSF com JPA, tendo como biblioteca de persistência o Hibernate e a biblioteca de componentes Primefaces. Nesta segunda edição, foram incluídos os princípios da programação web com capítulos exclusivos sobre HTML5 e CSS3, além de conceitos básicos da programação Java EE, como Servlets e páginas JSP.

Para a montagem deste material, diversas referências consagradas foram utilizadas. Na bibliografia as mesmas podem ser consultadas. Vale ressaltar que este material não substitui a leitura de livros consagrados sobre o tema.

Todos os códigos presentes neste material podem ser consultados em:
<https://drive.google.com/?tab=mo&authuser=0#folders/0B8cQpHd6a5gENnJYd0ZFTkdKVGM>.

Caso possua correções, sugestões ou mesmo esclarecer alguma dúvida sinta-se livre para entrar em contato: rosicleiafrasson@gmail.com.

Um bom estudo!

Criciúma, julho de 2014.

Rosicleia Frasson

SUMÁRIO

JPA	3
Generics	27
HTML.....	35
CSS	47
JAVA EE	71
Servlets	78
JSP	88
Frameworks MVC	99
JSF	102
Componentes JSF.....	113
Escopos JSF	150
Facelets.....	158
Filtros	163
Bibliotecas de componentes	205
Primefaces	208

JPA

Aplicações comerciais são desenvolvidas pela necessidade de coletar e armazenar grandes quantidades de dados e eventualmente esses dados são transformados para gerar informações para as companhias em forma de relatórios. O armazenamento desses dados normalmente é feito utilizando sistemas gerenciadores de banco de dados (SGBDs). Existem inúmeros SGBDs no mercado e a grande maioria deles trabalha com o modelo relacional, onde os dados são armazenados em tabelas com linhas e colunas.

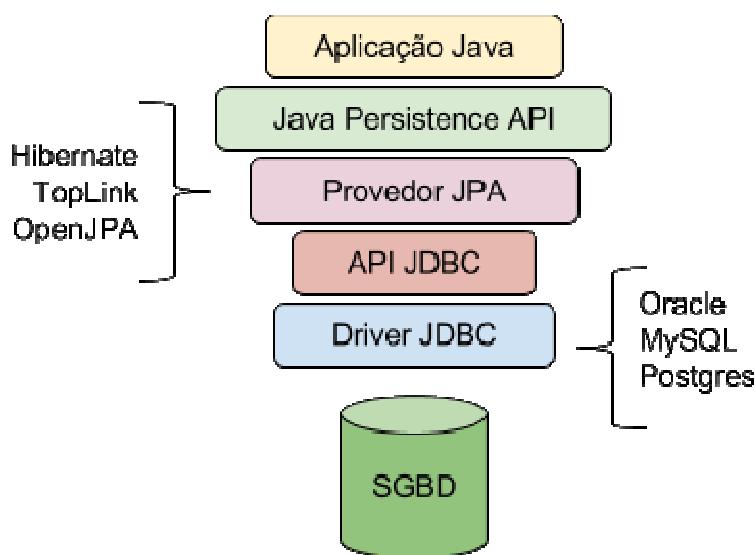
Aplicações desenvolvidas na linguagem Java, são escritas utilizando o paradigma orientado a objetos e a manipulação de dados, como descrita anteriormente, segue o modelo relacional. Como os modelos utilizados são de paradigmas diferentes, é necessário, que os dados sejam ajustados cada vez que passam de um modelo para o outro. Em outras palavras, a cada nova operação de inserção, atualização, remoção ou pesquisa, uma quantidade imensa de código deve ser implementada para tratar essas incompatibilidades de estruturas.

Com a popularização do Java notou-se que os programadores gastavam muito tempo com a codificação de queries SQL e nos códigos JDBC que manipulavam as mesmas. Além da produtividade, o acesso via JDBC dificulta a troca de bancos de dados, visto que algumas queries possuem diferenças significativas dependendo do fabricante.

Para facilitar a vida dos programadores surge o conceito de ORM (Mapeamento objeto-relacional) que propõe que a transformação entre os paradigmas orientado a objetos e relacional ocorra de maneira transparente para o programador. Utilizando este conceito, ao invés do programador efetuar as devidas conversões, o mesmo pode utilizar um framework para efetuar as alterações necessárias.

Java Persistence API e Ferramentas ORM

O Hibernate é o framework ORM mais conhecido do mercado e como os demais existentes disponibiliza toda a programação necessária para o mapeamento objeto-relacional e persistência dos dados.



Com a aceitação dos frameworks ORM pela comunidade de programadores e prevendo o surgimento de muitos outros frameworks ORM, foi incluída, a partir do Java 5, uma API de persistência

chamada de Java Persistence API ou JPA que padroniza o mapeamento objeto-relacional. A JPA propõe a possibilidade de trabalhar diretamente com as classes sem a utilização de consultas nativas de cada banco de dados.

A JPA é apenas uma especificação, ou seja, muitos textos, normas e interfaces do Java de como uma implementação deve se comportar. Atualmente existem no mercado diversas implementações da JPA, entre elas, as mais conhecidas são o Hibernate da JBoss, EclipseLink da EclipseFundation e OpenJPA da Apache.

Como os provedores de persistência seguem a especificação, é possível migrar de framework durante o desenvolvimento de uma aplicação. E como é o provedor que cuida da transformação dos dados para os SGBDs, também é possível efetuar a troca do fabricante de banco de dados sem alterações na aplicação.

Algumas características da JPA:

- Qualquer classe que contenha um construtor sem parâmetros, pode ser tornar persistente sem nenhuma alteração no código. Apenas algumas anotações são necessárias.
- Como a JPA, oferece diversos valores defaults, bastam poucas anotações para efetuar o mapeamento.
- As consultas com a JPA são efetuadas através da JPQL (Java Persistence Query Language) que possui sintaxe similar as queries SQL.

Anotações

Para efetuar o mapeamento das classes em JPA, o desenvolvedor pode optar entre as anotações diretamente no código fonte ou criar um arquivo XML. As anotações são mais recomendadas, pois a informação fica no código fonte sendo mais fácil a visualização e compreensão dos parâmetros utilizados.

As anotações são inseridas diretamente no código fonte, antes do elemento a ser anotado. Para identificar, cada anotação deve ser precedida por uma @. As anotações precisam dos imports que pertencem ao pacote java.persistence.

O mecanismo de persistência define padrões que se aplicam a grande maioria das aplicações. Dessa forma, o desenvolvedor necessita anotar apenas quando deseja substituir o valor padrão.

As anotações também podem possuir parâmetros. Os parâmetros são especificados dentro de parênteses como par nome-valor e separados por vírgula. A ordem dos parâmetros não interfere na execução.

Entidades

As entidades ou entities são utilizadas pela JPA para efetuar o mapeamento objeto-relacional. Dessa forma elas devem refletir a estrutura da tabela no banco de dados. Classes Java podem ser facilmente transformadas em entidades.

Para que um POJO Java seja considerada uma entidade é necessário que ela possua os seguintes requisitos:

- Possuir um construtor sem parâmetros. Outros construtores podem ser adicionados.
- Ter a anotação @Entity. Essa anotação indica a JPA que a classe deve ser transformada em uma tabela no banco de dados. Por padrão o nome da tabela é o mesmo nome da classe. Se o desenvolvedor preferir outro nome para a tabela, é necessário adicionar o parâmetro name com o nome desejado.

- Toda entidade tem que possuir uma identidade única. O identificador da entidade equivale a primary key da tabela. Esse campo deve ser anotado com @Id. Em geral esse campo é um número sequencial, mas pode ter outros tipos de valores.
- Os atributos da classe devem ser privados e os mesmos devem possuir os métodos de acesso (gets/sets).

Segue um exemplo de entidade.

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Funcionario {

    @Id
    private int matricula;
    private String nome;
    private double salario;

    public Funcionario() {
    }

    public int getMatricula() {
        return matricula;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }

    public String getNome() {
        return nome;
    }

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

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
```

```
    this.salario = salario;
}
}
```

Entity Manager

Na JPA, a persistência de um objeto é feita invocando o gerenciador de entidades (Entity Manager). Uma Entity Manager é responsável pela troca de informações entre a aplicação e um esquema de armazenamento de dados e pela abertura e fechamento das transações.

O contexto de persistência (Persistence Context) é o espaço de memória, onde os objetos persistentes se encontram. Dentro do contexto de persistência, as instâncias de entidades e seu ciclo de vida são gerenciados. O ciclo de vida de uma entidade pode possuir quatro estados diferentes:

- New (novo): A entidade foi criada, porém, ainda não foi persistida na base de dados, ou seja, ela ainda não é reconhecida pela JPA porque nunca passou pelo gerenciador de persistência EntityManager.
- Managed (gerenciado): A entidade foi persistida e encontra-se num estado gerenciável. Mudanças no estado da entidade são sincronizadas assim que uma transação for finalizada com sucesso, ou seja, a JPA garante que a entidade terá representação idêntica na base de dados.
- Removed (removido) : A entidade foi marcada para ser removida da base de dados. Essa remoção será efetuada quando a transação com o banco de dados for finalizada com sucesso.
- Detached(desacoplado): A entidade foi persistida no banco de dados, porém, encontra-se em um estado que não está associado ao contexto persistível, ou seja, alterações em seu estado não são refletidos na base de dados.

A interface EntityManager define métodos que são usados para interagir com o contexto de persistência, ou seja, ela é usada para criar e remover instâncias persistentes de uma entidade, para encontrar entidades pela sua chave primária e para efetuar consultas de entidades.

O conjunto de entidades que pode ser gerenciado por uma instância da EntityManager é definida por uma unidade de persistência. Segue uma lista com os métodos mais utilizados da EntityManager:

- createNamedQuery: cria uma instância de consulta para a execução de uma consulta nomeada.
- createQuery: cria uma instância de consulta para a execução de uma instrução de linguagem de consulta de persistência Java.
- find: efetua a busca pela primary key e traz os dados imediatamente.
- getReference: efetua uma busca pela primary key, os dados são carregados apenas quando o estado do objeto é acessado, aliviando o tráfego do banco de dados.
- getTransaction: retorna o objeto da transação com o banco de dados.
- merge: mescla o estado da entidade no contexto da persistência atual.
- persist: faz uma instância de entidade gerenciada e persistente.
- remove: remove a instância da entidade do banco de dados.

Para criar uma EntityManager é necessária uma instância de EntityManagerFactory ou uma fábrica de EntityManager. Como a EntityManagerFactory é um processo custoso e demorado, apenas uma instância da fábrica é usada por aplicação. Exceto quando é necessário acessar mais de uma fonte de dados diferente.

Persistence Unit

A unidade de persistência é utilizada para configurar as informações referentes ao provedor JPA e ao banco de dados utilizado.

Para definir a unidade de persistência é utilizado um arquivo XML, chamado persistence.xml que deve ser criado dentro da pasta META-INF do projeto. Neste arquivo podem ser definidas várias unidades de persistência.

JPQL

Como a linguagem SQL possui mudanças significativas para cada fabricante de banco de dados para efetuar consultas utilizando a JPA, é recomendado utilizar a JPQL (Java Persistence Query Language).

A JPQL é uma API de queries, similar as queries SQL que define consultas para entidades e seu estado persistente. Com o uso da JPQL, as consultas se tornam portáteis, independente do fabricante de banco de dados utilizado.

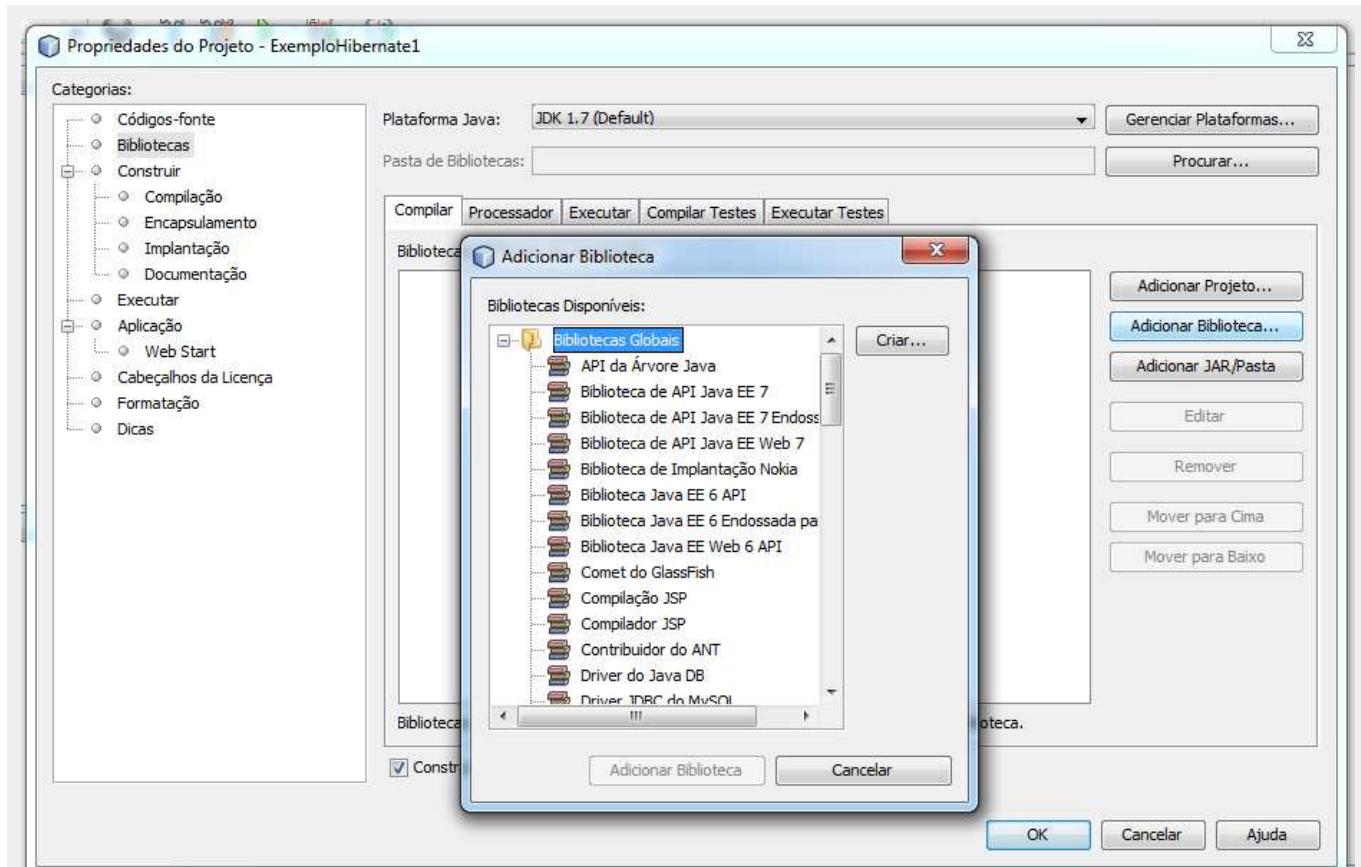
Um objeto query pode ser obtido através do Entity Manager usando o método createQuery. As queries podem ser estáticas ou dinâmicas. Uma query estática pode ser definida através de anotação @NamedQuery e possui uma performance melhor do que a query dinâmica, pois a JPA armazena cache da consulta, podendo executar a mesma query várias vezes apenas alterando os valores.

PASSO-A-PASSO

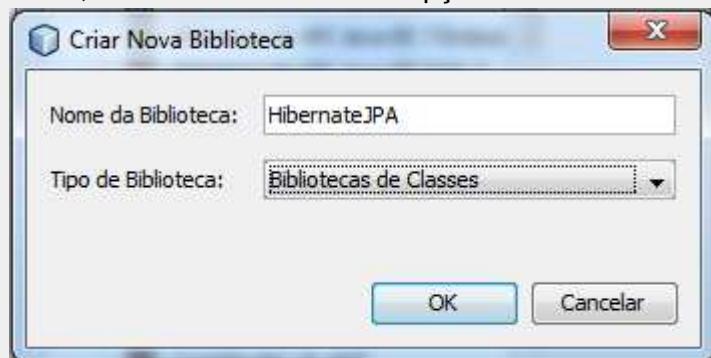
Configurando o ambiente

Neste exemplo, será usado o Hibernate com JPA. Sendo assim, o primeiro passo é colocar os jars correspondentes no classpath do projeto que podem ser encontrados no site oficial do Hibernate: www.hibernate.org. Serão necessários todos os jars obrigatórios (required), os jars da especificação JPA, que estão na pasta com o nome desta.

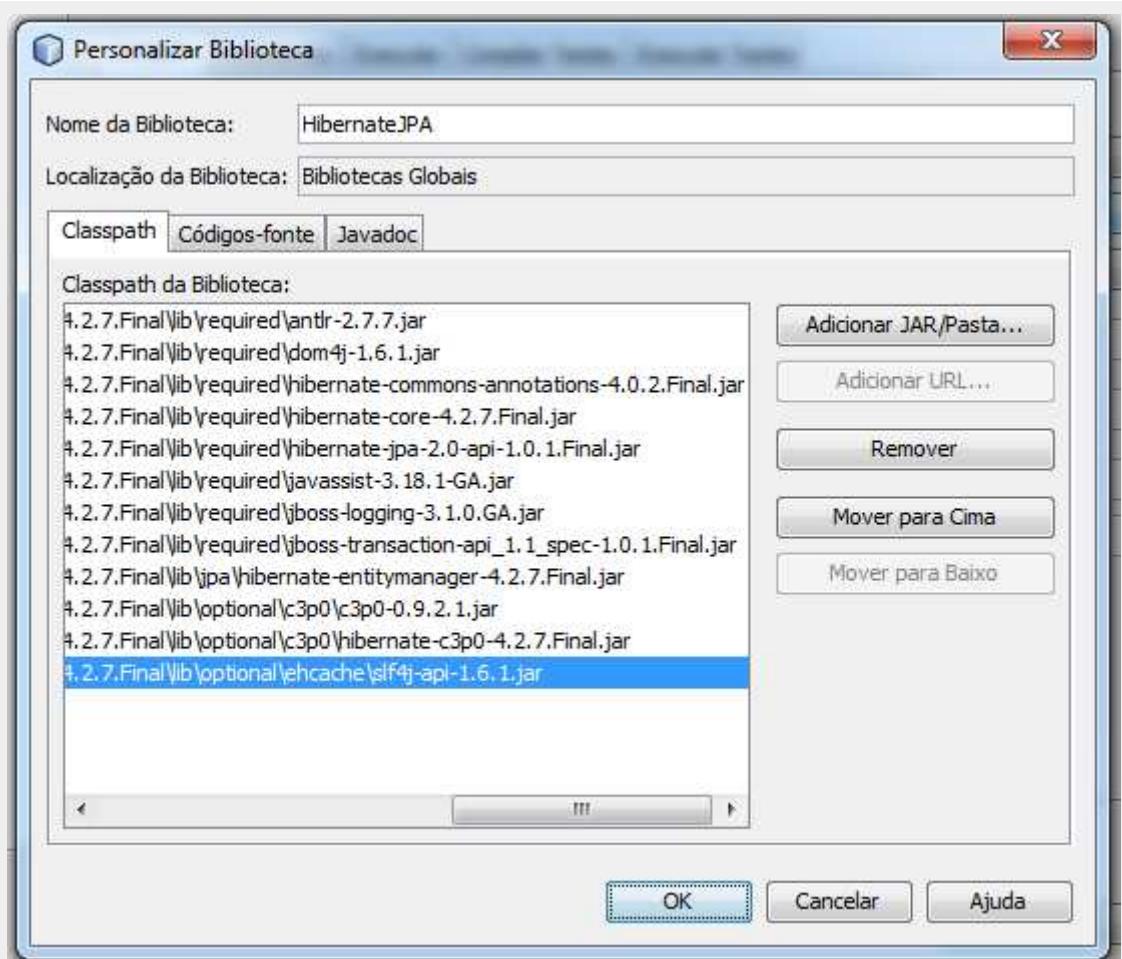
1. Para configuração no Netbeans, é necessário criar uma aplicação Java SE. Em exemplos posteriores será feita a configuração em projetos Java EE.
2. Após a criação do projeto, as propriedades do mesmo devem ser acessadas e no menu Bibliotecas, é necessário acionar a opção Adicionar Biblioteca.



3. Na caixa de diálogo aberta, é necessário acessar a opção Criar.



4. Após dar um nome para a biblioteca e clicar em OK, será aberta uma janela para adicionar os jars da biblioteca criada. São necessários todos os jars da pasta required e jpa. Opcionalmente podem ser inclusos outros jars. A figura a seguir, contém a lista de todos os jars adicionados.



5. Após ser criada, é necessário que a biblioteca seja adicionada ao projeto.
6. Além do Hibernate, também é necessário adicionar o driver JDBC do banco utilizado. Neste caso, será utilizado o MySQL.

Entidades

Para que o JPA possa mapear corretamente cada tabela do banco de dados para uma classe Java foi criado o conceito de Entity (Entidade). Para uma classe Java ser considerada uma Entity, necessariamente ela precisa possuir a anotação `@Entity`, um construtor público sem parâmetros e um campo anotado com `@Id`.

Neste exemplo será feito um CRUD de um funcionário, que contém como atributos matrícula, que será gerada automaticamente, nome e salário. Segue o mapeamento da classe Funcionario.

```
package br.com.rosicleiafrasson.exemplohibernate1.modelo;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Funcionario {
```

```
    @Id  
    @GeneratedValue  
    private int matricula;  
    private String nome;  
    private double salario;  
  
    public int getMatricula() {  
        return matricula;  
    }  
  
    public void setMatricula(int matricula) {  
        this.matricula = matricula;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public double getSalario() {  
        return salario;  
    }  
  
    public void setSalario(double salario) {  
        this.salario = salario;  
    }  
}
```

Por convenção, a classe Funcionario será mapeada para uma tabela com o mesmo nome (Funcionario). O atributo matricula está anotado com @Id, indicando que o mesmo representa a chave primária da tabela. A anotação @GeneratedValue indica que o valor terá autoincremento.

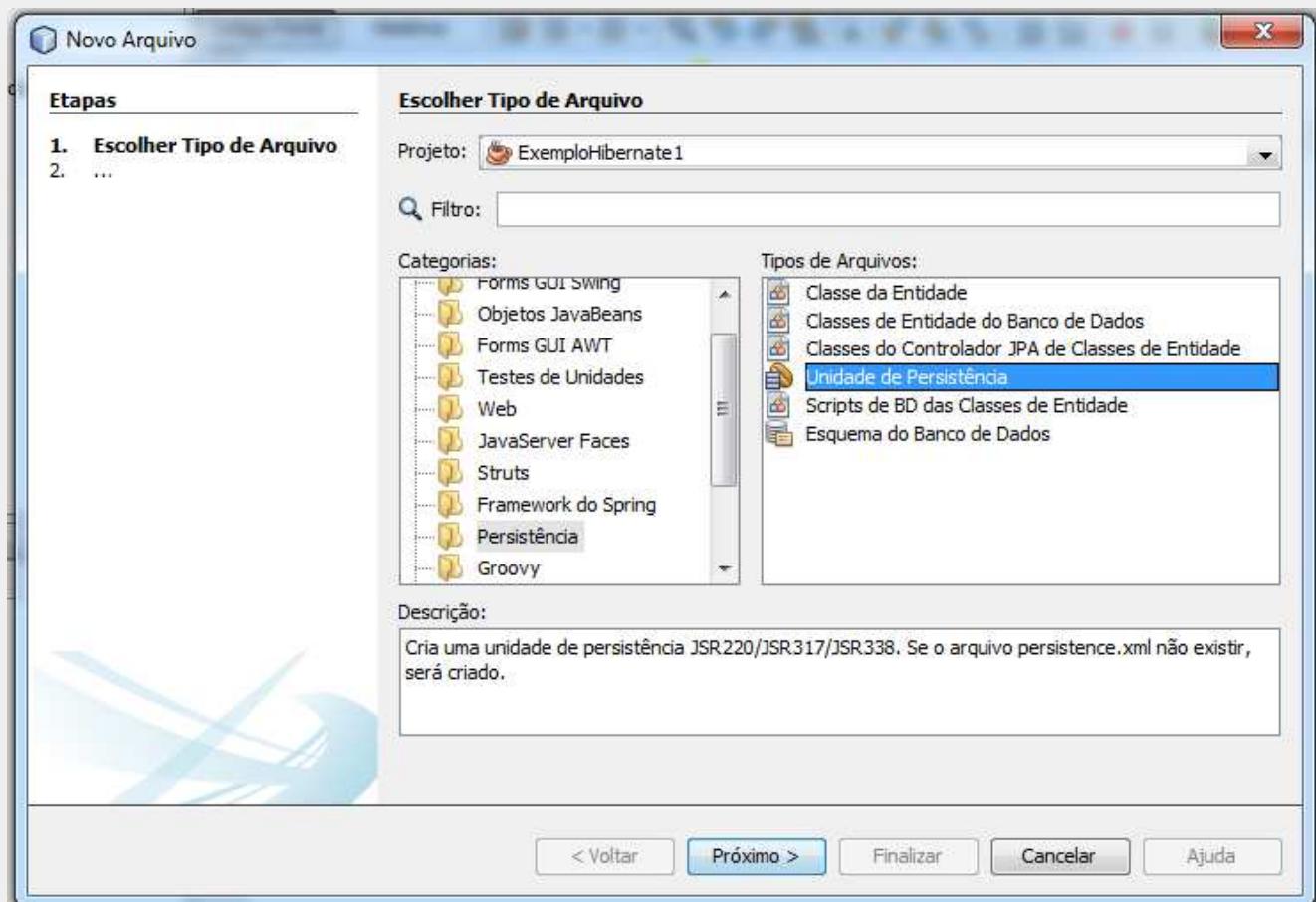
Se for necessário alterar o nome da tabela ou o nome da coluna podem ser usadas as anotações @Table e/ou @Column. Na ausência dessas configurações, a JPA irá mapear a classe Funcionario para uma tabela com o mesmo nome e os atributos matricula, nome e salario para colunas com os respectivos nomes também.

Persistence.xml

Para que a JPA consiga mapear e executar transações em um banco é necessário definir

algumas configurações, como por exemplo, provedor de persistência, dialeto usado para manipulação, driver de conexão, usuário e senha do banco, entidades que farão parte do contexto; algumas configurações avançadas do Hibernate como pool de conexões, controle de cache, entre outros. Essas configurações são armazenadas no arquivo persistence.xml.

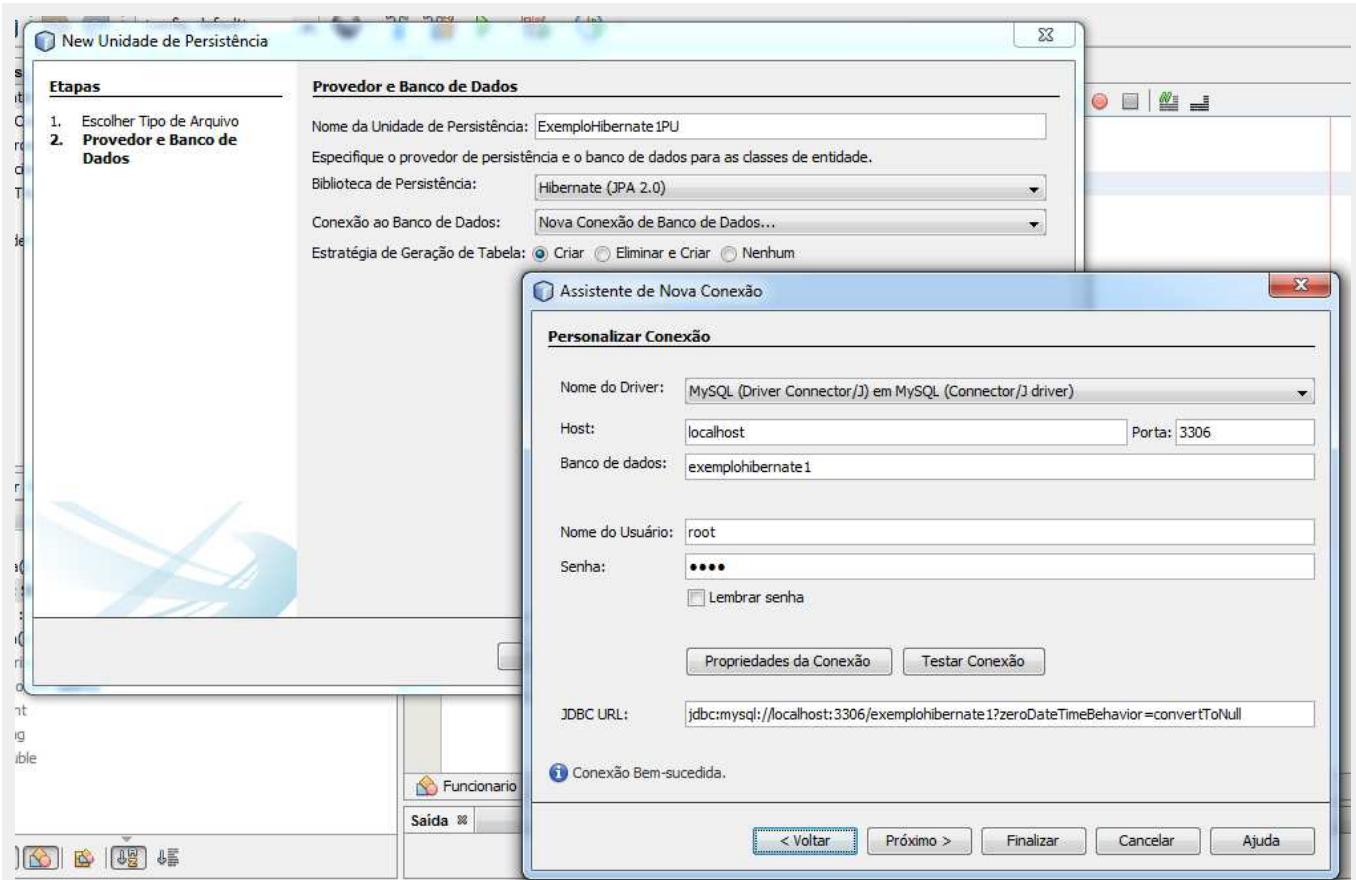
1. No Netbeans para criar um arquivo persistence.xml, basta clicar com o botão secundário em cima do projeto, escolher a opção Novo, Outros, categoria Persistência e o tipo de arquivo Unidade de Persistência.



2. Na tela provedor e banco de dados é necessário definir um nome para a unidade de persistência, especificar o provedor de persistência, neste caso, será utilizado o Hibernate 2.0. Em fonte de dados, é necessário informar a base de dados. Se esta não estiver listada, é necessário configurá-la. Para isso, a opção Nova Conexão de Banco de Dados deve ser acessada, e o Driver de Conexão selecionado, para este exemplo, será o driver do MySQL.

3. Na tela Assistente de Nova Conexao é necessário informar o nome do banco, o usuário e a senha. Na opção Testar Conexao, é possível verificar se os parâmetros para a conexão foram definidos corretamente.

4. A opção Estratégia de Geração de Tabela podem ser escolhidas entre as opções Criar, Eliminar e Criar e Nenhum. Selecionando a primeira delas - Criar - novas tabelas são criadas se necessário, porém os dados não são apagados. A segunda opção - Eliminar e Criar - apaga as tabelas existentes no banco e cria novamente, apagando os dados existentes. A terceira opção - Nenhum - não altera a estrutura da base de dados.



5. Após a finalização, é criado um arquivo persistence.xml dentro da pasta META-INF que deve estar no classpath do projeto, portanto, junto as classes.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" 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/persistence_2_0.xsd">
  <persistence-unit name="ExemploHibernate1PU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="javax.persistence.jdbc.url"
        value="jdbc:mysql://localhost:3306/exemplohibernate1?zeroDateTimeBehavior=convertToNull"/>
      <property name="javax.persistence.jdbc.password" value="root"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="hibernate.cache.provider_class"
        value="org.hibernate.cache.NoCacheProvider"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.format_sql" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

```
</persistence-unit>  
</persistence>
```

O arquivo gerado possui um cabeçalho, declarado com a tag persistence. Dentro desta tag, é necessário definir um conjunto de configurações para indicar o provedor de persistência, o fabricante e as configurações do banco de dados . Esse conjunto de configurações é chamado de persistence-unit, que deve possuir um nome, neste caso, foi definido como ExemploHibernate1PU, mas poderia ser dado qualquer outro nome.

Dentro das properties são colocadas tags property contendo um atributo name e um atributo value. No atributo name é indicado a configuração que deve ser feita e no atributo value, o conteúdo da configuração. É possível observar no arquivo persistence.xml criado que existem tags property para configuração da url, driver, user e password, que equivalem respectivamente as configurações do endereço, onde a base de dados está hospedada, o driver de conexão do banco de dados correspondente, o usuário e a senha do mesmo. É importante perceber que essas propriedades começam com javax.persistence.jdbc. Isso acontece porque elas fazem parte da especificação da JPA.

Além das propriedades da especificação JPA, é possível também configurar propriedades específicas da implementação. Uma delas é a escolha do dialeto do banco de dados. Quando a conexão é feita via JDBC, utiliza-se códigos em SQL e alguns comandos são específicos de um determinado banco de dados. Como o Hibernate pode trabalhar com diversos fabricantes de bancos de dados é necessário informar qual é o dialeto do banco, ou seja, a variação do SQL utilizado pelo mesmo. No javadoc do Hibernate, disponível em: <http://docs.jboss.org/hibernate/orm/4.1/javadocs/> pode ser encontrada a lista completa dos dialetos. Como será utilizado o MySQL para o armazenamento de dados é necessário inserir o dialeto do mesmo utilizando a seguinte propriedade no persistence.xml.

```
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect" />
```

A propriedade hibernate.hbm2ddl.auto permite que tabelas sejam criadas ou tenham suas estruturas alteradas automaticamente. Setando o valor para update, novas tabelas, colunas ou relacionamentos são criados, se necessário, porém os dados não são apagados. Essa configuração deve ser utilizada apenas na fase de desenvolvimento. Quando o software estiver em produção é aconselhável desabilitar essa configuração.

O Hibernate possui uma propriedade que permite que os comandos SQL sejam impressos no console quando estão sendo executados. Para ativar esse recurso, é necessário adicionar uma nova propriedade chamada show_sql com o valor true. É interessante que os comandos sejam impressos formatados. Isso é possível com a propriedade format_sql. Segue abaixo os comandos que devem ser adicionados no arquivo persistence.xml.

```
<property name="hibernate.show_sql" value="true"/>  
<property name="hibernate.format_sql" value="true"/>
```

Entity Manager

Criadas as configurações, é necessário fazer com que a aplicação se comunique com a base de dados. Para isso, deve ser criada uma classe que garanta essa comunicação. A título de exemplo, essa classe deve se chamar PersistidorFuncionario e invocar alguns métodos para que as

informações da aplicação sejam persistidas na base de dados.

A classe Persistence carrega o arquivo XML e inicializa as configurações. A classe EntityManagerFactory descobre quem cria as conexões e é necessário uma instância da mesma. Para obter uma instância de EntityManagerFactory é utilizado o método createEntityManagerFactory indicando qual é a persistence-unit que foi definida no persistence.xml, no nosso caso ExemploHibernate1PU. Executando o código abaixo dentro de um método main, a tabela funcionário deve ser criada no banco.

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ExemploHibernate1PU");
```

Com a instância de EntityManagerFactory é necessário invocar o método createEntityManager. Este método devolve um objeto do tipo EntityManager que representa a conexão com a unidade de persistência. O EntityManager é responsável por gerenciar as entidades, através dele é possível gerenciar o ciclo de vida das entidades, efetuar operações de sincronização com a base de dados (insert, delet e update), consultar entidades entre outros. Uma Entity está em um contexto gerenciável, quando a mesma está associada a um EntityManager. Isso garante que todas as operações realizadas em um objeto Entity são refletidas na base de dados.

```
EntityManager em = emf.createEntityManager();
```

Inserção

Para testar uma inserção de um funcionário na base de dados, é necessário instanciar e popular um objeto do tipo Funcionario e passar o mesmo para a EntityManager realizar a persistência. Também é indispensável pedir uma transação para a EntityManager e invocar os métodos begin e commit. O método begin inicializa os recursos da transação e o método commit confirma a transação corrente. Para salvar a entidade gerenciada na base de dados, é necessário invocar o método persist.

```
Funcionario f = new Funcionario();
f.setNome("João da Silva Junior Aguinaldo");
f.setSalario(611500.678);

em.getTransaction().begin();
em.persist(f);
em.getTransaction().commit();
```

É importante mencionar que EntityManagerFactory é uma fábrica para criar EntityManager. Por essa razão, é interessante que exista apenas uma EntityManagerFactory para toda a aplicação. A existência de várias fábricas de conexões espalhadas pela aplicação resultam em lentidão do software, entre outros problemas. Sendo assim, é importante garantir que a EntityManagerFactory seja criada apenas uma vez, isso é possível com a implementação de uma classe que possua um atributo estático para armazenar essa instância e que o mesmo seja uma constante, para que não seja criado outras vezes. Essa classe pode ser chamada de JPAUtil.

Na classe JPAUtil, também é necessário implementar um método getEntityManager que devolva uma EntityManager, desta forma, é possível ter acesso a um objeto de conexão sempre que necessário.

```
package br.com.rosicleiafrasson.exemplohibernate1.util;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class JPAUtil {

    private static final EntityManagerFactory emf = Persistence.
        createEntityManagerFactory("ExemploHibernate1PU");

    public static EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

Com a existência de uma fábrica de conexões centralizada, é necessário alterar a classe PersistidorFuncionario, para que a mesma utilize a classe JPAUtil.

```
EntityManager em = JPAUtil.getEntityManager();
```

Consulta

Com os dados gravados na base de dados, é possível realizar consultas. As consultas em JPA podem ser feitas utilizando SQL, porém não é recomendado, pois, ao usar SQL, a aplicação fica presa a um determinado fabricante de banco de dados. Existe uma linguagem chamada de JPQL (Java Persistence Query Language) que é portável e possui a sintaxe e estrutura bem similar ao SQL.

Para realizar uma pesquisa por todos os funcionários é utilizada a String “select f from Funcionario f”. Na clausula from deve ser indicada a entidade a ser buscada, nesse caso Funcionario. É necessário também a criação de um alias (apelido) para a entidade, no exemplo mostrado f.

O método createQuery recebe a consulta com a String com a consulta desejada e o tipo de objeto que a consulta deve retornar. Para executar a consulta, é invocado o método getResultList que devolve uma lista com os objetos indicados no momento da chamada ao createQuery.

```
EntityManager em = JPAUtil.getEntityManager();

Query q = em.createQuery("select f from Funcionario f", Funcionario.class);

List<Funcionario> funcionarios = q.getResultList();

for (Funcionario f : funcionarios) {
    System.out.println("\nMatrícula: " + f.getMatricula()
        + "\nNome: " + f.getNome())
```

```

        + "\nSalário: " + f.getSalario() + "\n");
    }

em.close();

```

Remoção

Para efetuar uma remoção, basta chamar o método remove, passando um objeto Funcionario com o id populado.

```

EntityManager em = JPAUtil.getEntityManager();

Funcionario f = em.getReference(Funcionario.class, 1);

em.getTransaction().begin();
em.persist(f);
em.getTransaction().commit();

```

Mapeamento

Além das anotações @Entity, @Id e @GeneratedValue já citadas anteriormente, a JPA disponibiliza uma série de anotações, que permitem ao desenvolvedor efetuar os relacionamentos da base de dados e modificar as configurações padrão de mapeamento da JPA.

Por padrão, ao anotar uma classe com @Entity, o JPA fará o mapeamento para uma tabela como o mesmo nome na base de dados. Caso seja necessário que a tabela possua um outro nome, a anotação @Table deve ser usada juntamente com a propriedade name, como mostrado no trecho de código a seguir.

```

@Entity
@Table (name = "tb_funcionario")
public class Funcionario {

```

Os atributos das classes também são mapeados para as tabelas da base de dados com o mesmo nome que se apresentam nas classes. Para modificar os nomes das colunas, é necessário utilizar a anotação @Column juntamente com a propriedade name. Além de modificar o nome da coluna, outras restrições podem ser impostas. Entre elas, podem ser citadas:

- length: Limita a quantidade de caracteres de uma String.
- nullable: Indica se o campo pode ou não possuir valor null.
- unique: Indica se uma coluna pode ou não ter valores repetidos.

O código mostrado a seguir indica que o atributo nome será mapeado como nome_fun, aceitará no máximo 100 caracteres, não pode ser nulo e dois funcionários não podem possuir o mesmo nome.

```

@Column(name = "nome_fun", length = 100, unique = true, nullable = false)
private String nome;

```

Para armazenar dados grandes, como imagens e sons, o atributo deve ser anotado com @Lob. O código a seguir demonstra o uso da anotação.

```
@Lob  
private byte[] foto;
```

Os tipos java.util.Date e java.util.Calendar necessitam da anotação @Temporal para serem mapeados. Esta anotação exige que o programador defina como a data será armazenada. Existem três opções de armazenamento:

- TemporalType.DATE: Armazena apenas a data.
- TemporalType.TIME: Armazena apenas a hora.
- TemporalType.TIMESTAMP: Armazena a data e a hora.

Segue o código utilizado para mapear uma data utilizando uma das anotações descritas.

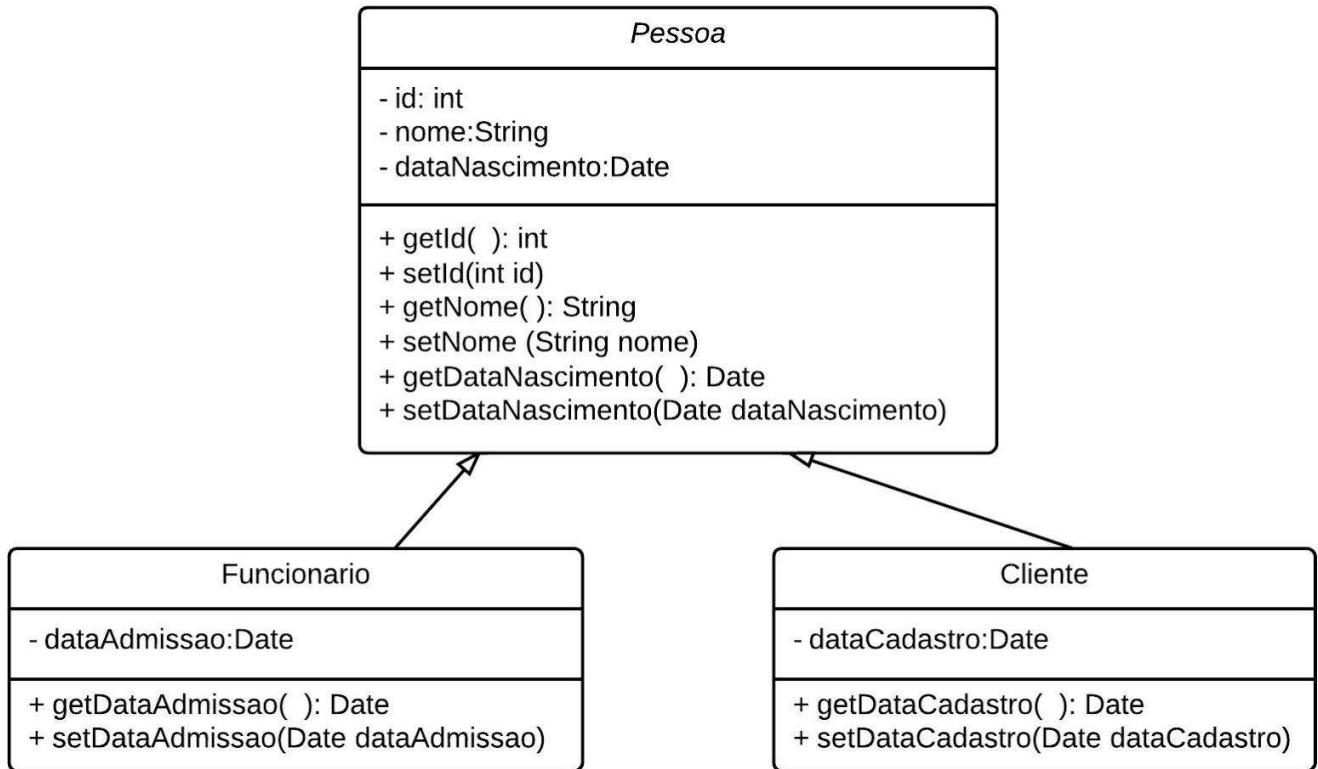
```
@Temporal (TemporalType.DATE)  
private Date dataNascimento;
```

Eventualmente, um campo contido em uma classe não pode ser mapeado para a base de dados. Para evitar que o atributo seja mapeado, é necessário inserir a anotação @Transient como mostrado a seguir.

```
@Transient  
private String confirmaSenha;
```

Existem diversas formas de mapear a herança em Java utilizando a JPA. Uma delas é utilizando a anotação @MappedSuperclass. Utilizando esta anotação, toda a classe que herdar da superclasse terá em sua tabela, os atributos pertencentes a classe e a superclasse. A anotação pode ser utilizada tanto em classes concretas, quanto abstratas.

Para demonstrar o mapeamento, será utilizado o diagrama de classes ilustrado a seguir.



O mapeamento das classes ilustradas, deve ser feito conforme os trechos de código abaixo.

```

package br.com.rosicleiafrasson.exemplohibernate1.modelo;
import java.util.Date;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@MappedSuperclass
public class Pessoa {
    @Id
    @GeneratedValue
    private int id;
    private String nome;
    @Temporal(TemporalType.DATE)
    private Date dataNascimento;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
  
```

```
public String getName() {
    return nome;
}

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

public Date getDataNascimento() {
    return dataNascimento;
}

public void setDataNascimento(Date dataNascimento) {
    this.dataNascimento = dataNascimento;
}
}
```

```
package br.com.rosicleiafrasson.exemplohibernate1.modelo;

import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
public class Funcionario extends Pessoa{

    @Temporal(TemporalType.DATE)
    private Date dataAdmissao;

    public Date getDataAdmissao() {
        return dataAdmissao;
    }

    public void setDataAdmissao(Date dataAdmissao) {
        this.dataAdmissao = dataAdmissao;
    }

}
```

```

package br.com.rosicleiafrasson.exemplohibernate1.modelo;

import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
public class Cliente extends Pessoa{

    @Temporal(TemporalType.DATE)
    private Date dataCadastro;

    public Date getDataCadastro() {
        return dataCadastro;
    }

    public void setDataCadastro(Date dataCadastro) {
        this.dataCadastro = dataCadastro;
    }
}

```

A base de dados apresentada a seguir corresponde ao mapeamento citado.

funcionario	cliente
id INT(11)	id INT(11)
dataNascimento DATE	dataNascimento DATE
nome VARCHAR(255)	nome VARCHAR(255)
dataAdmissao DATE	dataCadastro DATE

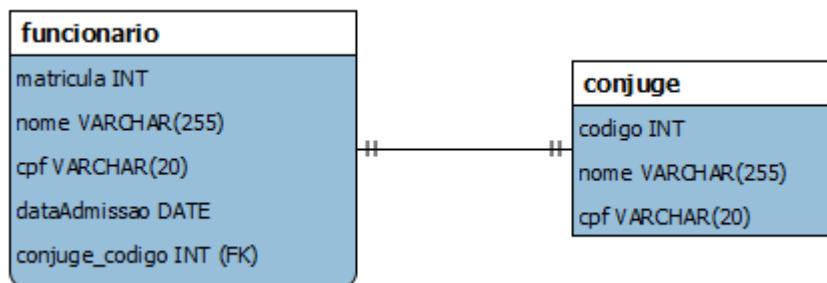
Relacionamentos

Uma das maiores divergência no mapeamento objeto-relacional ocorre nos relacionamentos existentes entre as classes. A JPA define quatro tipos de relacionamento de acordo com a sua cardinalidade: OneToOne, OneToMany, ManyToOne e ManyToMany.

OneToOne

Também conhecido como relacionamento um para um, ocorre quando um registro da tabela pode possuir ligação com apenas um registro de outra tabela. É o caso por exemplo, do relacionamento do funcionário com o cônjuge. Um funcionário só pode ter um cônjuge e um cônjuge só pode pertencer

a um funcionário.



Para efetuar o relacionamento é utilizada a anotação `@OneToOne`. É importante perceber que em Java, o relacionamento é feito com o objeto inteiro, ou seja, todos os dados do cônjuge, são carregados, juntamente com o funcionário. Na base de dados, o código do cônjuge, é o único dado correspondente a ligação entre as tabelas na entidade funcionário.

```

package br.com.rosicleiafrasson.cap1mapeamentoonetoono.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Conjugue {
    @Id @GeneratedValue
    private int codigo;
    private String nome;
    private String cpf;

    //Gets e sets
}
  
```

```

package br.com.rosicleiafrasson.cap1mapeamentoonetoono.model;

import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
public class Funcionario {
  
```

```

@Id
@GeneratedValue
private int matricula;
private String nome;
private String cpf;
@Temporal(TemporalType.DATE)
private Date dataAdmissao;
@OneToOne
@JoinColumn(name = "conjuge_codigo")
private Conjuge conjugue;

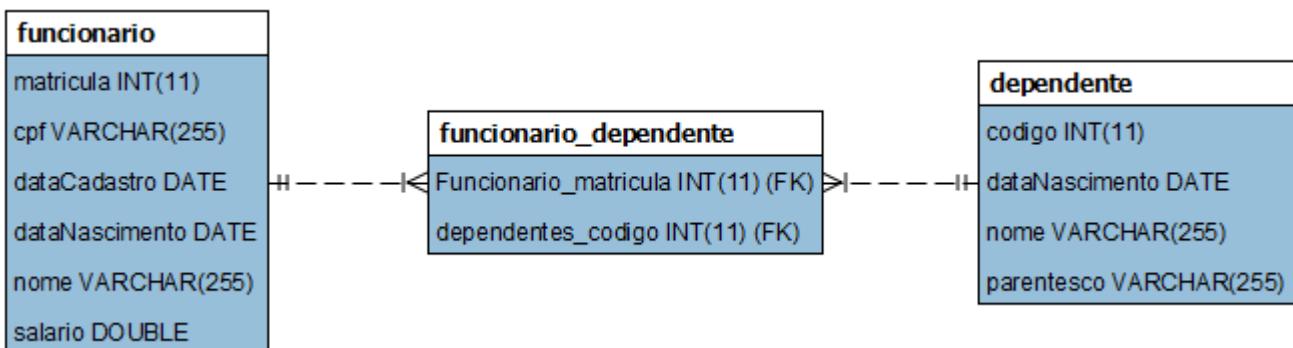
//Gets e sets
}

```

One to Many

O relacionamento um para muitos informa que o registro de uma entidade está relacionado com vários registros de outra. Como exemplo desse relacionamento pode ser citado o funcionário e seus dependentes. Um funcionário pode possuir zero ou vários dependentes.

Na base de dados, além das tabelas funcionário e dependente uma terceira tabela deve ser gerada para relacionar os registros dos dependentes com o registro do funcionário.



Na JPA, para efetuar o mapeamento do relacionamento um para muitos, é utilizada a anotação `@OneToMany`, como exemplificado no código a seguir.

```

package br.com.rosicleiafrasson.cap1mapeamentoonetomany.model;

import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

```

```
@Entity  
public class Dependente {  
  
    @Id @GeneratedValue  
    private int codigo;  
    private String nome;  
    @Temporal(TemporalType.DATE)  
    private Date dataNascimento;  
    private String parentesco;  
  
    //Gets e sets  
}
```

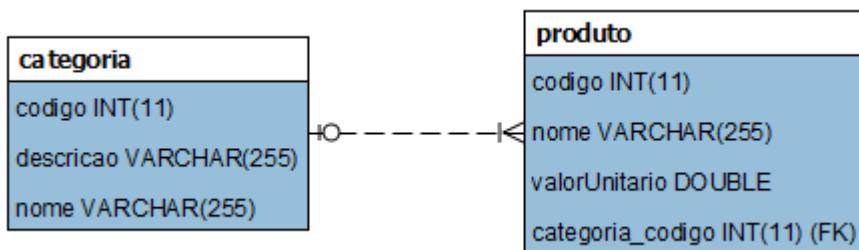
```
package br.com.rosicleiafrasson.cap1mapeamentoonetomany.model;  
  
import java.util.Collection;  
import java.util.Date;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.JoinTable;  
import javax.persistence.OneToMany;  
import javax.persistence.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
public class Funcionario {  
  
    @Id @GeneratedValue  
    private int matricula;  
    private String nome;  
    private String cpf;  
    @Temporal(TemporalType.DATE)  
    private Date dataNascimento;  
    @Temporal(TemporalType.DATE)  
    private Date dataCadastro;  
    private double salario;  
    @OneToMany  
    private Collection<Dependente> dependentes;  
  
    //Gets e sets  
}
```

Por padrão, o nome da tabela gerada para indicar o relacionamento é a junção das duas tabelas, funcionário e dependente (funcionario_dependente). Se necessário alterar o padrão, pode ser utilizada a anotação @JoinTable com a propriedade name.

```
@OneToMany
@JoinTable(name = "func_depen")
private Collection<Dependente> dependentes;
```

Many to One

No relacionamento muitos para um, muitos registros de uma entidade estão relacionados com um registro de outra entidade. Como exemplo deste relacionamento, pode ser citado, o relacionamento entre o produto e sua categoria. Muitos produtos devem possuir a mesma categoria.



A anotação @ManyToOne é responsável por efetuar este mapeamento na JPA. O código a seguir, demonstra a utilização desta.

```
package br.com.rosicleiafrasson.cap1mapeamentomanytoone.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Categoria {

    @Id
    @GeneratedValue
    private int codigo;
    private String nome;
    private String descricao;

    //Gets e sets
}


```

```
package br.com.rosicleiafrasson.cap1mapeamentomanytoone.model;

import javax.persistence.Entity;
```

```

import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class Produto {

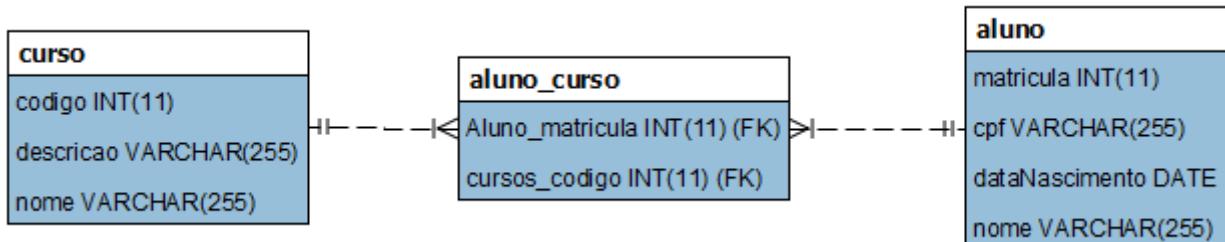
    @Id
    @GeneratedValue
    private int codigo;
    private String nome;
    private double valorUnitario;
    @ManyToOne
    private Categoria categoria;

    //Gets e sets
}

```

Many To Many

O relacionamento muitos para muitos indica que múltiplos registros de uma entidade podem se relacionar com múltiplos registros de outra entidade. É o caso por exemplo do relacionamento entre alunos e cursos. Um aluno pode estar matriculado em mais de um curso e um curso pode ter mais de um aluno.



Na JPA, existe a anotação `@ManyToMany` para indicar esse relacionamento. Um exemplo de utilização desta anotação pode ser visto no código a seguir.

```

package br.com.rosicleiafrasson.cap1mapeamentomanytomany.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Curso {

```

```
    @Id  
    @GeneratedValue  
    private int codigo;  
    private String nome;  
    private String descricao;  
  
    //Gets e sets  
  
}
```

```
package br.com.rosicleiafrasson.cap1mapeamentomanytomany.model;  
import java.util.Collection;  
import java.util.Date;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.ManyToMany;  
import javax.persistence.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
public class Aluno {  
  
    @Id  
    @GeneratedValue  
    private int matricula;  
    private String nome;  
    private String cpf;  
    @Temporal(TemporalType.DATE)  
    private Date dataNascimento;  
    @Temporal(TemporalType.DATE)  
  
    @ManyToMany  
    private Collection<Curso> cursos;  
  
    //Gets e sets  
  
}
```

Generics

Os generics estão presentes em diversas linguagens de programação e permite ao desenvolvedor construir códigos que podem ser utilizados com vários tipos de dados. Na linguagem Java, os tipos genéricos foram inclusos na versão 5.0, fato considerado pela comunidade de programadores como uma grande evolução da linguagem. A utilização de generics permite tipos na declaração de métodos, interfaces e classes, dessa forma, permite que o programador desenvolva código mais limpo, seguro e reutilizável.

Bons exemplos do uso do Generics são encontrados na API do Java na interface Collections. Tanto a interface quanto as implementações desta utilizam os tipos genéricos. A interface List, por exemplo, que herda de Collection, possui uma implementação chamada ArrayList que é responsável pelo armazenamento dos dados em forma de lista ligada. Não seria interessante que o programador se preocupasse em sobrescrever os métodos a cada nova classe criada em que houvesse a necessidade de armazenamento em forma de lista, para que os mesmos fossem suportados pela estrutura. A classe ArrayList como todas as classes pertencentes a hierarquia de Collections são genéricas, permitindo que o uso destas possa ser feito por qualquer objeto.

O uso adequado de tipos genéricos apresenta inúmeras vantagens, a maior delas é o grande reaproveitamento de código, além disso as classes genéricas eliminam a necessidade de cast e deixam o código mais robusto.

Em códigos que utilizam genéricos, o compilador é responsável por remover o parâmetro de tipo e substituir por tipos reais. Esse processo ocorre na tradução para bytecodes e é definido como erasure.

Classes genéricas

As classes genéricas ou parametrizadas aceitam parâmetros em sua declaração. Como os parâmetros contidos nos métodos, os parâmetros de tipos - definição dos parâmetros genéricos - , podem ser utilizados com diferentes entradas.

```
public class ExemploGenerico<T> {  
}
```

Na declaração de classe, é necessário colocar após o nome da classe, o indicador genérico. O indicador genérico é definido por uma letra que será substituída no momento da compilação e podem representar apenas tipos por referência. Tipos primitivos não são permitidos, porém podem manipular objetos de classes empacotadoras de tipos. O indicador genérico fica dentro de uma seção de parâmetros delimitada por colchetes angulares (< e >). Cada seção de parâmetros pode ter um ou mais indicadores, separados por vírgulas.

Por convenção, os indicadores são nomeados por letras maiúsculas únicas e obedecem a seguinte padronização:

- E - Elemento
- K - Chave
- N - Número
- T - Tipo
- V - Valor

Para utilizar uma classe genérica, a instanciação é feita da seguinte forma:

```
ExemploGenerico<ExemploObjeto> eg = new ExemploGenerico<ExemploObjeto>();
```

Métodos genéricos

Um grande recurso da orientação a objetos é a sobrecarga de métodos. Na sobrecarga, é possível a definição de métodos com o mesmo nome, porém com tipo ou quantidade de parâmetros diferentes. Caso haja a necessidade de um método funcionar para tipos de objetos diferentes, são definidos métodos para cada tipo de dado.

O uso de generics permite que uma única declaração de método com argumentos de tipos diferentes seja codificada, nos casos em que as operações realizadas forem idênticas para todos os tipos de dados.

```
public T pesquisar (I pk) {  
    T tipo;  
}
```

Da mesma forma, que nas classes genéricas, os métodos genéricos possuem uma seção de parâmetros de tipos, que precedem o tipo de retorno do método. Nos métodos genéricos, os indicadores podem representar o tipo de retorno, tipos de parâmetros e tipos de variáveis locais.

Os métodos genéricos podem ser sobrecarregados por métodos genéricos e não genéricos. Na ocorrência de sobrecarga, o compilador procura pelo método que mais se adequa aos tipos de argumentos especificados na chamada. Primeiramente ele procura nos métodos não genéricos, caso não encontre uma correspondência precisa de tipos, tenta localizar um método genérico correspondente.

PASSO-A-PASSO

DAO Genérico

Quando a persistência dos dados de uma aplicação é feita utilizando JPA, a implementação do CRUD é idêntica, diferindo apenas o tipo de dados utilizado. Desta forma, uma boa solução para reduzir a quantidade de código implementado seria a utilização de Generics. Além da redução de código, o uso de classes genéricas facilita posteriores modificações, pois os métodos ficam todos centralizados.

O primeiro passo para a criação de um DAO genérico é a criação de uma interface contendo a assinatura dos métodos comuns a serem implementados por todos os DAOs. Segue a seguir.

```
package br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia.dao;  
  
import java.util.List;  
import javax.persistence.EntityManager;  
  
public interface DAO <T, I> {
```

```

    T save(T entity);

    boolean remove(Class <T> classe, I pk);

    T getById(Class<T> classe, I pk);

    List<T> getAll(Class<T> classe);

    EntityManager getEntityManager();

}

```

Concluída a interface DAO, o próximo passo é a criação de uma classe que implementa este DAO. Essa classe será chamada de DAOJPA. Por questões de organização a classe criada será abstrata e todas as entidades também possuirão seus DAOs específicos que devem herdar de DAO e DAOJPA. As implementações específicas não necessitam sobrescrever os métodos comuns já implementados.

```

package br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia;

import br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia.dao.DAO;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import javax.swing.JOptionPane;

public abstract class DAOJPA<T, I> implements DAO<T, I> {

    private JPAUtil conexao;

    @Override
    public T save(T entity) {
        T saved = null;
        try {
            EntityManager().getTransaction().begin();
            saved = EntityManager().merge(entity);
            EntityManager().getTransaction().commit();
        } catch (Exception e) {
            if (EntityManager().getTransaction().isActive() == false) {
                EntityManager().getTransaction().begin();
            }
        }
    }
}

```

```
        getEntityManager().getTransaction().rollback();
        JOptionPane.showMessageDialog(null, "Erro ao salvar elemento na base de
dados" + e.getMessage());
    }
    return saved;
}

@Override
public boolean remove(Class<T> classe, I pk) {
    boolean estado = false;
    try {
        getEntityManager().getTransaction().begin();
        getEntityManager().remove(getEntityManager().getReference(classe, pk));
        getEntityManager().getTransaction().commit();
    } catch (Exception e) {
        if (getEntityManager().getTransaction().isActive() == false) {
            getEntityManager().getTransaction().begin();
        }
        getEntityManager().getTransaction().rollback();
        JOptionPane.showMessageDialog(null, "Erro ao remover elemento na base de
dados" + e.getMessage());
    }
    return estado;
}

@Override
public T getById(Class<T> classe, I pk) {
    try {
        return getEntityManager().find(classe, pk);
    } catch (NoResultException e) {
        return null;
    }
}

@Override
public List<T> getAll(Class<T> classe) {
    return getEntityManager().createQuery("select x from " + classe.getSimpleName()
+ " x").getResultList();
}

@Override
```

```

public EntityManager getEntityManager() {
    if (conexao == null) {
        conexao = new JPAUtil();
    }
    return conexao.getEntityManager();
}
}

```

O DAOJPA é responsável pelas principais ações de movimentação dos dados de todas as entidades. Dessa forma as inserções, remoções, atualizações na base de dados são definidas nesta classe. As pesquisas pela chave primária e a pesquisa de todos os registros de determinada entidade também já estão implementadas. Vale ressaltar, que os métodos citados não precisam ser sobreescritos em seus DAOs específicos, já que a implementação definida na classe DAOJPA funciona para todas as entidades.

A fim de exemplificar, a utilização de um DAO genérico para mapeamento de uma entidade, foi criada a classe Pessoa, com apenas dois atributos. As regras de mapeamento prevalecem as mesmas já citadas.

```

package br.com.rosicleiafrasson.cap2jpagenerico.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Pessoa {

    @Id @GeneratedValue
    private int id;
    private String nome;

    //Gets e sets
}

```

Para que a entidade Pessoa, possa utilizar as operações do DAO genérico é necessário criar uma interface que extenda de DAO, neste exemplo PessoaDAO. Os parâmetros Pessoa e Integer são necessários porque a interface DAO, como classe genérica, é parametrizada e necessita dos parâmetros tipo de classe e tipo de chave primária, nesse caso, Pessoa e Integer.

Se forem necessários métodos de busca específicos para a entidade Pessoa, ou seja, diferentes dos métodos existentes no DAO genérico, os mesmos devem ser declarados na interface PessoaDAO.

```

package br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia.dao;
import br.com.rosicleiafrasson.cap2jpagenerico.model.Pessoa;

```

```
public interface PessoaDAO extends DAO <Pessoa, Integer>{  
}
```

Além da interface, também faz-se necessário a criação de uma classe concreta que estenda de DAOJPA e implemente a interface correspondente a interface da entidade. Nessa classe, se necessário, são efetuadas as implementações dos métodos específicos a entidade Pessoa.

```
package br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia;  
  
import br.com.rosicleiafrasson.cap2jpagenerico.model.Pessoa;  
import br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia.dao.PessoaDAO;  
  
public class PessoaDAOJPA extends DAOJPA<Pessoa, Integer> implements PessoaDAO{  
}
```

A utilização de tipos genéricos não exclui a necessidade de uma fábrica de entityManager, a JPAUtil. Segue a implementação da mesma.

```
package br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia;  
  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
  
public class JPAUtil {  
  
    private static final String UNIT_NAME = "JPAGenericoPU";  
    private EntityManagerFactory emf = null;  
    private EntityManager em = null;  
  
    public EntityManager getEntityManager() {  
  
        if (emf == null) {  
            emf = Persistence.createEntityManagerFactory(UNIT_NAME);  
        }  
  
        if (em == null) {  
            em = emf.createEntityManager();  
        }  
        return em;  
    }  
}
```

É importante perceber que a definição de uma variável para o armazenamento do nome da unidade de persistência aumenta a visibilidade do código. As condicionais para verificação da

existência de objetos EntityManager e EntityManagerFactory antes a criação dos mesmos, aumenta a performance da aplicação.

O arquivo persistence.xml segue a estrutura padrão anteriormente apresentada.

Para atestar a validade dos métodos genéricos foi efetuada a inserção de uma pessoa na base de dados e posteriormente a listagem das pessoas que estão cadastradas. Segue o código de teste.

```
package br.com.rosicleiafrasson.cap2jpagenerico.teste;

import br.com.rosicleiafrasson.cap2jpagenerico.model.Pessoa;
import br.com.rosicleiafrasson.cap2jpagenerico.model.persistencia.PessoaDAOJPA;
import java.util.List;

public class Teste {

    public static void main(String[] args) {
        PessoaDAOJPA dao = new PessoaDAOJPA();

        Pessoa p = new Pessoa();
        p.setNome("Teste");

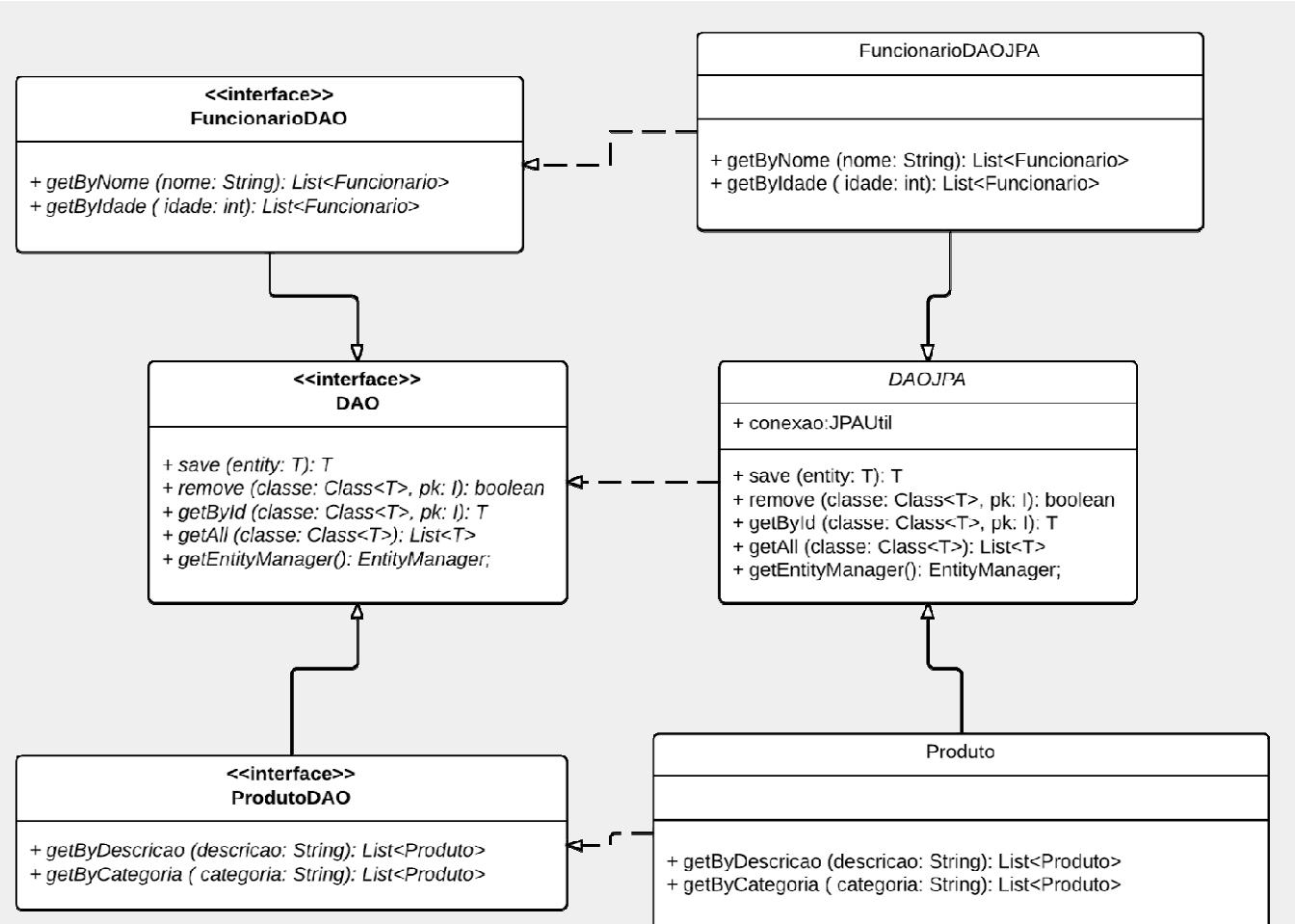
        dao.save(p);

        List<Pessoa> pessoas = dao.getAll(Pessoa.class);

        for (Pessoa pessoa : pessoas) {
            System.out.println(pessoa.getNome());

        }
    }
}
```

Em aplicações com uma quantidade maior de entidades, a utilização do DAO genérico é semelhante com o exemplo apresentado. Todas as entidades devem possuir seu DAO que extende do DAO genérico e seu DAOJPA que implementa o seu DAO e extende do DAOJPA. A seguir uma ilustração de uma aplicação que contém as entidades funcionário e produto.



É importante perceber que a entidade funcionário possui métodos de pesquisa por nome e idade e a entidade produto possui os métodos de pesquisa por descrição e categoria. Como os métodos citados, não são utilizados por todas as entidades, o ideal é que os mesmos sejam implementados em seus DAOs específicos.

HTML

O W3C, órgão responsável pela padronização da internet, define HTML (Hyper Text Markup Language) como uma linguagem de marcação para a construção de páginas web, ou seja, todas as páginas web do mundo inteiro são construídas utilizando HTML.

O HTML é uma linguagem de marcação formada por tags. Diversas tags são disponibilizadas pela linguagem e cada uma possui uma funcionalidade específica. As tags são definidas com os caracteres < e > e seu nome. As tags precisam ser abertas e fechadas. O fechamento de uma tag é feito utilizando uma barra /.



Estrutura HTML

Todos os documentos HTML possuem obrigatoriamente uma estrutura padrão composta pelas tags html, head e body. Segue um detalhamento sobre essas tags essenciais em documentos HTML:

<html> </html>	Essa tag indica o início e o fim do documento html. Dentro desta tag são necessárias as tags head e body, apresentadas a seguir.
<head> </head>	O elemento head representa o cabeçalho de um documento. Dentro do cabeçalho devem ser inseridas informações de configuração para o browser. Dessa forma, o conteúdo da tag head não é exibido pelo navegador.
<body> </body>	Dentro da tag body são definidas informações que devem ser exibidas pelo browser.

Além das tags obrigatórias anteriormente descritas, um documento HTML também necessita da instrução DOCTYPE. Essa instrução indica ao navegador a versão do HTML que o documento foi escrito. Ao utilizar a versão 5 do HTML a instrução DOCTYPE deve ser semelhante a exibida a seguir:

```
<!DOCTYPE html>
```

A seguir está ilustrada a estrutura padrão de uma página HTML.

```

<!DOCTYPE html>
<html>
  <head>
    <title>Título da página</title>
    <meta charset="UTF-8">
  </head>
  <body>
  </body>

```

```
</body>
</html>
```

Como visto no código acima, todo documento html deve iniciar com a tag `<html>` e fechar com `</html>`. Todas as demais tags devem ficar entre a abertura e o fechamento desta.

A tag `<head>` possui informações para o navegador. Os dados contidos dentro desta tag não são exibidos na área do documento no navegador. É imprescindível que dentro da tag `<head>` exista a tag `<title>`, que indica o título do documento que é exibido na barra de título do navegador.

Para que os caracteres como acentos e cedilha apareçam corretamente no navegador, é necessário informar o tipo de codificação utilizada por meio da configuração charset na tag `<meta>`. Atualmente para essa configuração é utilizado o valor UTF-8, também chamado de Unicode.

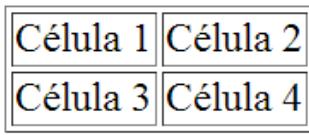
A tag `<body>` contém o corpo do documento que é exibido pelo navegador. Dentro do body é que serão colocados todos os elementos que devem aparecer na janela.

O DOCTYPE é um comando especial que indica ao navegador qual a versão do html utilizar. No trecho de código ilustrado, a declaração de DOCTYPE simples, indica que deve ser usada a última versão do html.

Além da estrutura apresentada, existem inúmeras tags que podem ser utilizadas em documentos HTML. Segue uma lista das mais utilizadas:

Tag	Descrição
h1, h2, h3, h4, h5 e h6	<p>Define cabeçalhos. A importância dos cabeçalhos é dada pelos números após o h. Onde h1 é o cabeçalho mais importante e o h6 o de menor importância.</p> <pre><h1>Título 1</h1> <h2>Título 2</h2> <h3>Título 3</h3> <h4>Título 4</h4> <h5>Título 5</h5> <h6>Título 6</h6></pre> <p style="text-align: center;">Título 1</p> <p style="text-align: center;">Título 2</p> <p style="text-align: center;">Título 3</p> <p style="text-align: center;">Título 4</p> <p style="text-align: center;">Título 5</p> <p style="text-align: center;">Título 6</p>
a	<p>Define um link que leva o usuário para outros documentos.</p> <pre>link para outra página</pre>

	página no mesmo diretório
link	Define um link para fontes externas que serão usadas no documento. <pre><link rel="stylesheet" type="text/css" href="style.css"></pre>
p	Define um parágrafo. <pre><p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam eget ligula eu lectus lobortis condimentum. Aliquam nonummy auctor massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla at risus. Quisque purus magna, auctor et, sagittis ac, posuere eu, lectus. Nam mattis, felis ut adipiscing.</p> <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam eget ligula eu lectus lobortis condimentum. Aliquam nonummy auctor massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla at risus. Quisque purus magna, auctor et, sagittis ac, posuere eu, lectus. Nam mattis, felis ut adipiscing.</p></pre> <p style="font-size: small;">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam eget ligula eu lectus lobortis condimentum. Aliquam nonummy auctor massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla at risus. Quisque purus magna, auctor et, sagittis ac, posuere eu, lectus. Nam mattis, felis ut adipiscing.</p> <p style="font-size: small;">Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam eget ligula eu lectus lobortis condimentum. Aliquam nonummy auctor massa. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla at risus. Quisque purus magna, auctor et, sagittis ac, posuere eu, lectus. Nam mattis, felis ut adipiscing.</p>
ul / li	ul: Define uma lista de elementos. li: Define um elemento da lista. <pre> Sorvete Chocolate Café </pre> <ul style="list-style-type: none"> • Sorvete • Chocolate • Café
table / tr / td	table: Define uma tabela. td: Define uma célula da tabela. tr: Define uma linha da tabela. <pre><table></pre>

	<pre> <tr> <td>Célula 1</td> <td>Célula 2</td> </tr> <tr> <td>Célula 3</td> <td>Célula 4</td> </tr> </table> </pre> 
--	--

Formulários HTML

Os formulários são usados para obter dados do usuário para serem processados. Um formulário é composto por três partes: a tag form, os elementos do formulários e os botões de envio. As tags form delimitam o formulário, tendo como principais atributos:

- action: especifica o que será executado quando o usuário enviar os dados.
- method: Especifica se o método de envio será get ou post. O método get envia o conjunto de dados junto com a URL. No método post, os dados não são mostrados na URL.

Os formulários devem ser compostos por elementos de interação com o usuário. Segue uma lista com os principais:

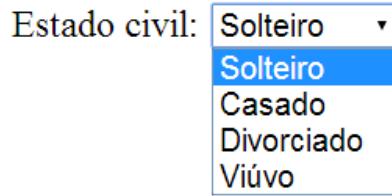
- input: Este elemento cria vários tipos de controle e o atributo type é quem define o tipo de controle que será utilizado. Alguns valores para o atributo type:

Valor atributo type	Descrição
text	<p>Campo de entrada de dados com apenas uma linha.</p> <p>Nome: <code><input type="text" name="nome" id="name" /></code></p> 
password	<p>Campo de senha.</p> <p>Senha: <code><input type="password" name="senha" /></code></p> 
radio	Botão de opção.

	<p>Sexo:
</p> <pre><input type="radio" name="sexo" value="m">Masculino
 <input type="radio" name="sexo" value="f">Feminino
</pre>
	<p>Sexo:</p> <input checked="" type="radio"/> Masculino <input type="radio"/> Feminino
checkbox	<p>Caixa de seleção.</p> <pre>Veículo:
 <input type="checkbox" name="veiculo" value="Bicicleta">Bicicleta
 <input type="checkbox" name="veiculo" value="Carro">Carro
 <input type="checkbox" name="veiculo" value="Motocicleta">Motocicleta
</pre>
	<p>Veículo:</p> <input type="checkbox"/> Bicicleta <input type="checkbox"/> Carro <input type="checkbox"/> Motocicleta
submit	<p>Botão para envio de formulário.</p> <pre><input type="submit" value="Salvar"></pre>
	<input type="button" value="Salvar"/>
reset	<p>Botão para limpar os campos do formulário.</p> <pre><input type="reset" value="Cancelar"></pre>
	<input type="button" value="Cancelar"/>

- select: Elemento para a criação de listas de seleção.

```
Estado civil:
<select>
  <option value="solteiro">Solteiro</option>
  <option value="casado">Casado</option>
  <option value="divorciado">Divorciado</option>
  <option value="viuwo">Viúvo</option>
</select>
```



- **textArea:** Campo de entrada de dados que pode possuir mais de uma linha.

```
Observações: <br>
<textarea name="observacoes" cols="30" rows="4"></textarea>
```

Observações:

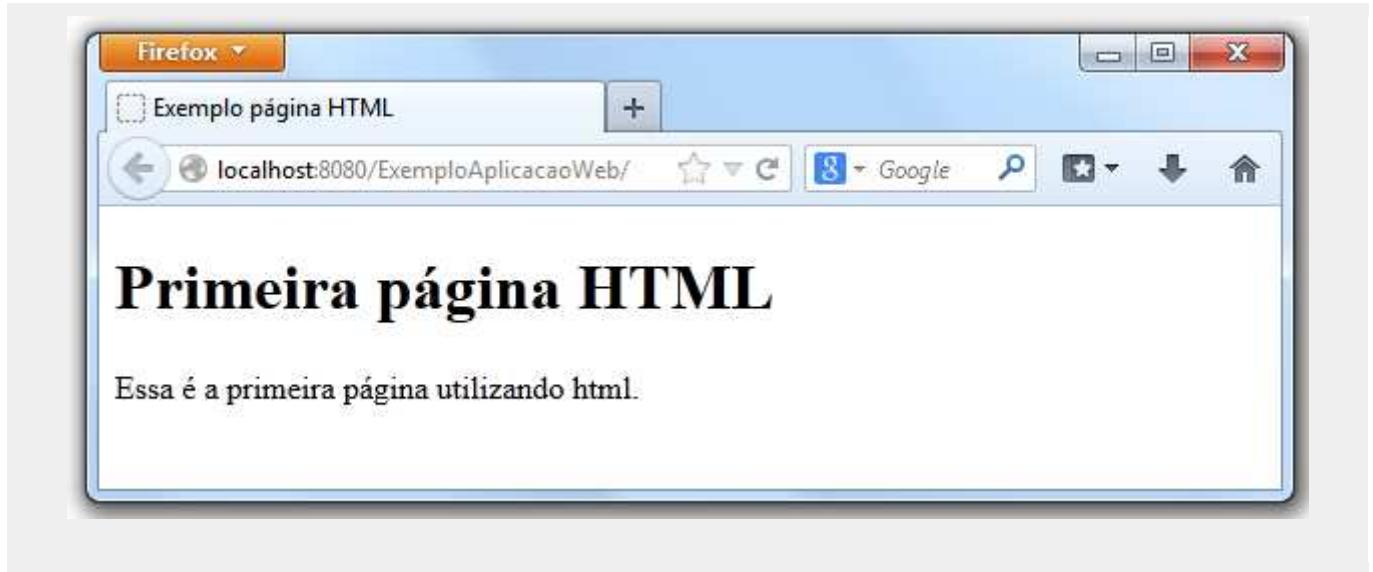
PASSO-A-PASSO

Exemplo página html

1. No projeto criado no passo-a-passo anterior, o arquivo index.html deve conter o código mostrado abaixo.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Exemplo página HTML</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <h1>Primeira página HTML</h1>
        <p>Essa é a primeira página usando HTML.</p>
    </body>
</html>
```

2. Ao executar a aplicação, deve ser mostrada a janela ilustrada abaixo.



HTML 5

Estrutura básica do HTML 5



- <header>: Especifica o cabeçalho de um documento ou de uma seção.
- <footer>: Especifica o rodapé de um documento ou seção. Normalmente o rodapé de um documento contém o autor do documento, informações de copyright, links para termos de uso e

informações de contato.

- **<section>**: Delimita as seções de um documento.
- **<nav>**: Representa uma seção da página que contém um conjuntos de links de navegação. Geralmente esse elemento é utilizado no menu principal e nos blocos de links colocados no rodapé.
- **<aside>**: Representa um conteúdo secundário que não faz parte do conteúdo principal do site. Esta marcação é bastante utilizada em sidebars e publicidade.
- **<article>**: Delimita um conteúdo que se constitui em um componente autosuficiente que pode ser distribuído ou reusado de forma independente. É recomendado o uso deste elemento em posts de fóruns e blogs, notícias e comentários.
- **<main>**: Define o conteúdo principal da página, ou seja, representa o conteúdo mais importante da página.

Novos tipos de campos

Novos valores atributo	Descrição
tel	<p>Campo usado para armazenar um telefone. Por não existir um padrão definido para números de telefone, este tipo não possui máscara de formatação ou validação. Se necessário, a validação deve ser feita com o uso do atributo pattern.</p> <div style="background-color: #e0f2ff; padding: 10px; margin-top: 10px;"> <p>Telefone:</p> <pre><input type="tel" name="tel" ></pre> </div> <div style="margin-top: 20px;"> <p>Telefone: <input type="text"/></p> </div>
search	<p>Campo de busca. Por ser um campo de busca a estilização do componente é diferenciado.</p> <div style="background-color: #e0f2ff; padding: 10px; margin-top: 10px;"> <pre><input type="search" name="procura" ></pre> </div> <div style="margin-top: 20px; border: 1px solid #ccc; padding: 5px; display: inline-block;"> teste X </div>
email	<p>Campo de email com formatação e validação.</p> <div style="background-color: #e0f2ff; padding: 10px; margin-top: 10px;"> <p>Email:</p> <pre><input type="email" name="email" ></pre> </div> <div style="margin-top: 20px;"> <p>Email: <input type="text"/></p> </div>
url	<p>Campo para endereço url com formatação e validação.</p>

	<p>Site:</p> <pre><input type="url" name="site" ></pre> <p>Site: <input type="text"/></p>
datetime	<p>Campo destinado a receber data e hora.</p> <p>Data de nascimento:</p> <pre><input type="datetime" name="data" ></pre> <p>Data de nascimento: <input type="text"/></p>
datetime-local	<p>Campo destinado a receber data e hora segundo a hora local, ou seja trata automaticamente a diferença entre fusos horários.</p> <p>Data de cadastro:</p> <pre><input type="datetime-local" name="data" ></pre> <p>Data de cadastro: <input type="text"/> dd/mm/aaaa --:--:00</p>
date	<p>Campo destinado a data.</p> <p>Data de cadastro:</p> <pre><input type="date" name="data" ></pre> <p>Data de cadastro: <input type="text"/> dd/mm/aaaa</p>
month	<p>Campo destinado a receber mês e ano.</p> <p>Mês:</p> <pre><input type="month" name="mes" ></pre> <p>Mês: ----- de ----</p>
week	<p>Campo destinado a receber a semana e o ano.</p> <p>Semana:</p> <pre><input type="week" name="semana" ></pre>

	<p>Semana: <input type="week" value="Semana --, ----"/></p>
time	<p>Campo destinado para receber hora.</p> <p>Horário:</p> <pre><input type="time" name="horário" ></pre> <p>Horário: <input type="time" value="-- :-- :00"/></p>
number	<p>Campo destinado a entrada de números.</p> <p>Idade:</p> <pre><input type="number" name="idade" ></pre> <p>Idade: <input type="number" value="4"/></p>
range	<p>Campo destinado a receber um número dentro de um intervalo.</p> <p>Volume:</p> <pre><input type="range" name="volume" min="0" max="100" ></pre> <p>Volume: <input type="range" value="50"/></p>
color	<p>Campo seletor de cor.</p> <p>Cor:</p> <pre><input type="color" name="cor" ></pre> <p>Cor: <input type="color" value="#000000"/></p>

Recursos de formulários

Recurso	Descrição
autofocus	Atribui o foco ao campo com o atributo quando a página é carregada.
placeholder	Atribui dentro de um input um texto que será apagado quando o usuário começar a digitar dentro de um campo.
required	Determina que o preenchimento do campo é obrigatório.
maxlength	Limita a quantidade de caracteres em um campo. No HTML5, o elemento textArea também possui essa propriedade.
pattern	Determina que o preenchimento do campo deve seguir um padrão definido por uma expressão regular.

Elementos de audio e vídeo

Elemento	Descrição
audio	<p>Adiciona um som na página. É necessário declarar o caminho do arquivo de som no atributo src do elemento. O atributo control adiciona controles de audio como os botões play e pause e controle de volume.</p> <pre><audio controls> <source src="kalimba.mp3" type="audio/mpeg"> Seu navegador não suporta o elemento de audio. </audio></pre> 
video	<p>Adiciona um elemento de vídeo na página. Da mesma forma que no elemento de audio, é necessário declarar o caminho do arquivo. O atributo control adiciona os controles de vídeo como play, pause e som.</p> <pre><video width="320" height="240" controls> <source src="amostra.mp4" type="video/mp4"> Seu navegador não suporta o elemento de vídeo. </video></pre>



CSS

Os usuários de páginas web estão cada vez mais exigentes. As páginas web devem ter boa usabilidade, possuir uma navegação simples e agradável, onde o usuário possa concluir sua tarefa em poucos passos. Além disso, o usuário deseja páginas que sejam visualmente atraentes. Como visto no capítulo anterior a criação de páginas utilizando apenas componentes JSF, não deixa as mesmas esteticamente bonitas.

Para aprimorar o visual, é necessário aplicar CSS (Cascading Style Sheets) nas páginas. O CSS é usado para definir a aparência da página. Com ele é possível definir a cor, o tamanho, o posicionamento, entre outros detalhes, dos componentes HTML.

Declaração do CSS

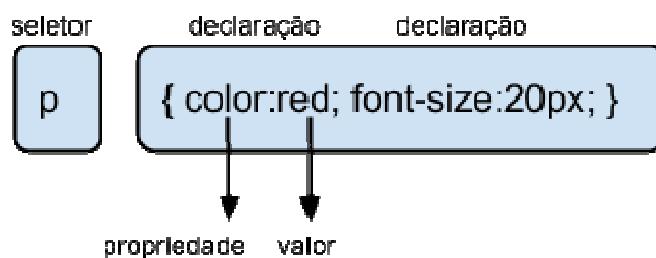
O CSS pode ser declarado diretamente na tag ou na seção head de uma página HTML, no entanto, é mais comum e produtivo utilizar folhas de estilo externas, já que essa abordagem permite que o mesmo estilo seja aplicado em diversas páginas. Essas folhas de estilos possuem a extensão .css.

A indicação do uso de uma folha de estilo deve ficar dentro da tag head. Segue exemplo de declaração de uma folha de estilo.

```
<head>
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
```

Sintaxe do CSS

Uma regra CSS pode ser dividida em duas partes: seletor e declaração. O seletor normalmente é o elemento HTML que deve receber o estilo. A declaração é composta por uma propriedade e um valor. Por sua vez a propriedade é um atributo de estilo que será aplicado. Cada propriedade deve possuir um valor.



Além de definir um estilo para um elemento HTML, é possível especificar seletores utilizando classes e ids. Um id é uma identificação única, só pode ser utilizado uma vez na página html. Já a classe é reutilizável, ou seja, pode ser repetida na mesma página.

Os seletores podem ser combinados para aplicar formatações mais específicas. Também é possível aplicar um bloco de declarações para mais de um seletor.

HTML	CSS
<pre><div id="cabecalho"> Aqui vai o cabeçalho do site </div> <div id="conteudo"> <div class="subtitulo"> <p>subtítulo 1 </p> </div> <div class="subtitulo"> <p>subtítulo 2 </p> </div> </div> <div id="rodape"> Aqui vai o rodapé do site </div></pre>	<pre>#cabecalho{ /*Declarações*/ } #conteudo{ /*Declarações*/ } #rodape{ /*Declarações*/ } .subtitulo{ /*Declarações*/ }</pre>

Propriedades

background-color	Especifica a cor de um elemento de fundo. Em CSS a cor pode ser especificada: <ul style="list-style-type: none"> em hexadecimal: #FFFFFF. em RGB: rgb(255,0,0). pelo nome da cor: red. <pre>body{ background-color: #082767; }</pre>
background-image	Indica um arquivo de imagem para ser exibido no fundo do elemento. <pre>body{ background-image: url("fundo.png"); }</pre>
background-repeat	Controla a repetição da imagem de fundo. Essa propriedade pode conter os seguintes valores: <ul style="list-style-type: none"> repeat: a imagem é repetida tanto na horizontal quanto na vertical. repeat-x: a imagem é repetida na horizontal. repeat-y: a imagem é repetida na vertical. no-repeat: a imagem não é repetida. <pre>body{ background-image: url("fundo.png"); }</pre>

	<pre style="background-color: #e0f2ff; padding: 10px;">background-repeat: no-repeat; }</pre>
background-attachment	Define se a imagem de fundo é fixa ou se rola junto com a página. Possui dois valores fixed e scroll. <pre style="background-color: #e0f2ff; padding: 10px;">body{ background-image: url("fundo.png"); background-attachment: scroll; }</pre>
color	Define a cor de um elemento. Normalmente a cor é especificada em números hexadecimais, porém também ser definida através de seu nome ou RGB. <pre style="background-color: #e0f2ff; padding: 10px;">p{ color: #23FF56; }</pre>
text-align	Define o alinhamento horizontal do texto. Os valores disponíveis para esta propriedade são: <ul style="list-style-type: none"> • center: texto alinhado ao centro. • right: texto alinhado à direita. • left: texto alinhado à esquerda. • justify: texto justificado. <pre style="background-color: #e0f2ff; padding: 10px;">p{ text-align: left; }</pre>
text-decoration	Propriedade que permite adicionar ou retirar a decoração de um texto. Pode conter os seguintes valores: <ul style="list-style-type: none"> • underline: texto sublinhado. • line-through: texto tachado. • overline: texto com sobrelinha. • blink: texto que pisca. • none: sem decoração. <pre style="background-color: #e0f2ff; padding: 10px;">h1{ text-decoration: underline; }</pre>
text-transform	Permite transformações no formato caixa-alta ou caixa-baixa no texto. Permite os seguintes valores: <ul style="list-style-type: none"> • capitalize: primeira letra das palavras em maiúscula. • lowercase: todas as letras em maiúsculas. • uppercase: todas as letras em minúsculas. • none: sem efeito. <pre style="background-color: #e0f2ff; padding: 10px;">h1{ text-transform: uppercase; }</pre>

text-indent	Permite que um recuo seja aplicado na primeira linha do parágrafo.
	<pre>p{ text-indent: 30px; }</pre>
font-family	<p>Define a família de fonte utilizada. Normalmente é definida uma lista de fontes e sua prioridade para apresentação na página. Dessa forma, se a primeira fonte da lista não estiver instalada na máquina do usuário, deverá ser usada a segunda. Se a segunda também não estiver instalada usa-se a terceira e assim até ser encontrada uma fonte instalada.</p> <p>As fontes podem ser definidas pelo nome da família, como por exemplo, Times New Roman, Georgia, Arial, Tahoma. Também podem ser definidas fontes de famílias genéricas, ou seja, fontes que pertencem a um grupo com aparência uniforme. Dentre as famílias genéricas podem ser citadas:</p> <ul style="list-style-type: none"> • serif: fontes que possuem pé. • sans-serif: fontes que não possuem pé. • monospace: fontes que possuem todos os caracteres com uma largura fixa. <p>É recomendado que a lista de fontes seja encerrada com uma fonte genérica.</p> <pre>h1 { font-family: arial, verdana, sans-serif; }</pre>
font-style	Define o estilo da fonte que pode ser normal, italic e oblique.
	<pre>h1 { font-style: italic; }</pre>
font-size	Define o tamanho da fonte. O tamanho da fonte pode ser definido em pixel ou em. O tamanho de 1 em equivale a 16 px.
	<pre>h1 { font-size: 2.5em; }</pre>
	<pre>h1 { font-size: 40px; }</pre>
list-style-type	Define o estilo de marcador de uma lista. Em uma lista não ordenada os marcadores devem ser sempre do mesmo tipo e podem assumir os seguintes valores:
	<ul style="list-style-type: none"> • none: sem marcador. • disc: círculo preenchido.

	<ul style="list-style-type: none"> ● circle: círculo não preenchido. ● square: quadrado cheio. <p>Já nas listas ordenadas, os marcadores podem ser:</p> <ul style="list-style-type: none"> ● decimal: número decimal. ● decimal-leading-zero: o marcador é um número decimal com dois dígitos. ● upper-latin: o marcador é composto por letras maiúsculas. ● lower-latin: o marcador é composto por letras minúsculas. ● upper-roman: o marcador é composto por letras romanas. <pre><code>ul { list-style-type: circle; }</code></pre>
width	Define a largura de um elemento. <pre><code>h1 { width: 200px; border: 1px solid black; background: orange; }</code></pre>
height	Define a altura de um elemento. <pre><code>h1 { height: 50px; border: 1px solid black; background: orange; }</code></pre>
float	Desloca um elemento para a esquerda ou para a direita. <pre><code>#figura { float:left; width: 100px; }</code></pre> Para limpar a flutuação dos elementos posteriores, é necessário utilizar a propriedade clear com o valor both.

Links

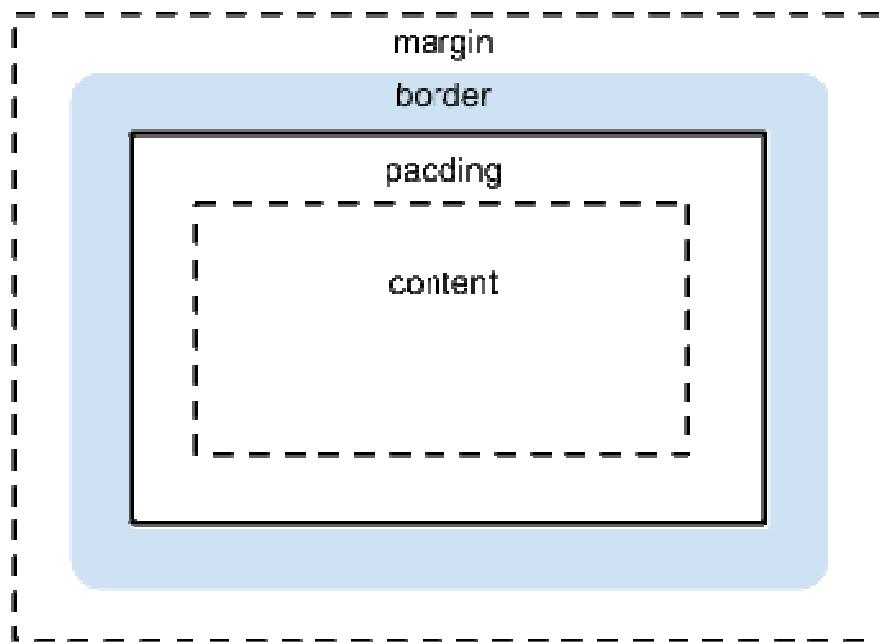
Os links podem ser formatados usando qualquer propriedade CSS. Diferentemente do que ocorre com outros seletores, os links podem ser formatados de acordo com o seu estado.

a:link	Estiliza um link não visitado. <pre><code>a : link {</code></pre>
---------------	--

	<pre style="background-color: #e0f2ff; padding: 10px;">color: green; }</pre>
a:visited	Estiliza um link já visitado pelo usuário. <pre style="background-color: #e0f2ff; padding: 10px;">a: visited { color: blue; }</pre>
a:hover	Estiliza o link quando o mouse está sobre ele. <pre style="background-color: #e0f2ff; padding: 10px;">a:hover: visited { background-color:#FF704D; }</pre>
a: active	Estiliza o link quando está sendo clicado. <pre style="background-color: #e0f2ff; padding: 10px;">a:active { background-color:#FF704D; }</pre>

Box Model

O box model é uma espécie de caixa que envolve todos os elementos HTML. O box model possui opções de ajuste de margem, borda, preenchimento e conteúdo para cada elemento.



- margin: Área limpa ao redor da borda. A margem não possui uma cor de fundo, que é completamente transparente.
- border: Área que gira em torno do conteúdo e do padding.

- padding: Área limpa ao redor do conteúdo.
- content: Área onde o texto e as imagens aparecem.

border-style	<p>Define o estilo da borda. Alguns estilos suportados:</p> <ul style="list-style-type: none"> • none: sem borda. • dotted: borda pontilhada. • dashed: borda tracejada. • solid: borda em linha sólida. • double: define duas bordas. <pre><code>h1 { border-style: dotted; }</code></pre> <p>Para que as demais propriedades de borda surtam efeito, o border-style deve ser diferente de none. Os lados que compõem a borda podem possuir propriedades diferentes. Nesse caso, é necessário indicar o lado em que a propriedade será aplicada: top, bottom, right e left.</p>
border-width	<p>Define o tamanho da borda. A espessura pode ser definida em pixels ou utilizando os valores thin, medium, e thick (fina, média e grossa).</p> <pre><code>h1 { border-width: thick; }</code></pre>
border-color	<p>Define a cor da borda.</p> <pre><code>h1 { border-color: gold; }</code></pre>
margin	<p>Define uma margem para o elemento. Esta propriedade adiciona espaço após o limite do elemento. É possível indicar o lado que a margem será aplicada utilizando bottom, top, left e right. É possível também aplicar margens diferentes em uma mesma propriedade. Sendo que o primeiro valor representa top, o segundo right, o terceiro bottom e o último left.</p> <pre><code>body{ margin: 25px 50px 75px 100px; } h1{ margin-top: 10px; margin-bottom: 20px; margin-left: 15px; margin-right: 30px; }</code></pre>
padding	<p>Define uma margem interna para o elemento, ou seja, a distância entre o limite do</p>

elemento, sua borda e seu conteúdo.

```
body{  
    padding-top:25px;  
    padding-bottom:25px;  
    padding-right:50px;  
    padding-left:50px;  
}
```

CSS3

O CSS3 é o novo padrão para CSS. É compatível com as versões anteriores e possui alguns recursos interessantes. A seguir são listados alguns.

border-radius

Adiciona bordas arredondadas ao elemento.

```
div{  
    border-radius:25px;  
}
```



box-shadow

Define sombra aos elementos da página. Este atributo necessita de alguns valores para especificar as características da sombra:

- Deslocamento horizontal da sombra: Corresponde ao primeiro valor do atributo. Se for definido um valor positivo a sombra deve aparecer à direita do elemento. Para sombra aparecer à esquerda do elemento, é necessário definir um valor negativo.
- Deslocamento vertical da sombra: Corresponde ao segundo valor da propriedade. Corresponde ao deslocamento vertical da sombra com relação ao elemento que a produz. Se for definido um valor positivo a sombra aparece abaixo do elemento e um valor negativo aplica a sombra acima do elemento.
- Esfumaçado: É o terceiro valor da propriedade. O valor zero indica que a sombra não é esfumaçada. Valores maiores aumentam o efeito.
- cor da sombra: É o último elemento. Podem ser utilizados os nomes das cores, rgb ou hexadecimal.

Para o Firefox essa propriedade é definida como -moz-box-shadow. Para o Chrome e Safari a propriedade é definida como -webkit-box-shadow.

```
#caixa{  
    width: 100px;  
    height: 100px;  
    background: #CFE2F3;  
    box-shadow: 5px 5px 3px #333;
```

	<pre> -webkit-box-shadow: 5px 5px 3px #333; -moz-box-shadow: 5px 5px 3px #333; } </pre> 
text-shadow	<p>Propriedade que aplica sombras a um texto. Esta propriedade possui quatro parâmetros: o primeiro deles representa o deslocamento da sombra para a direita (valor positivo) ou para a esquerda (valor negativo), o segundo representa o deslocamento da sombra para baixo (valor positivo) ou para cima (valor negativo), o quarto representa o raio para o esfumaçado da sombra e o último parâmetro representa a cor da sombra.</p> <pre> h1{ text-shadow: 1px 1px 4px #9B30FF; } </pre>
	Texto com sombra
linear-gradient	<p>Define um gradiente sem a utilização de imagens. A propriedade aceita valores para o ângulo do gradiente e as cores utilizadas.</p> <p>Vale ressaltar que no navegador Safari o gradiente é definido com a propriedade: -webkit-linear-gradient, no Opera: -o-linear-gradient e no Firefox: -moz-linear-gradient.</p> <pre> #caixa{ width: 300px; height: 100px; background-image: linear-gradient(to bottom, white, blue); } </pre> 
	<p>Para auxiliar na criação de gradientes, existem algumas ferramentas disponíveis. Seguem duas delas:</p> <ul style="list-style-type: none"> • http://www.colorzilla.com/gradient-editor/ • http://www.css3factory.com/linear-gradients/.
@font-face	Propriedade que permite a utilização de famílias de fontes fora do padrão do sistema operacional.

```
@font-face {  
    font-family: alexbrush;  
    src: url('AlexBrush-Regular.ttf');  
}  
  
p{  
    font-family: alexbrush;  
    font-size: 40px;  
}
```



Para que a fonte escolhida funcione da maneira correta, é necessário utilizar várias extensões da mesma. Os formatos de fontes aceitos pelos navegadores padrões são .ttf, .eot, .otf, .svg e .svgz. Arquivos de fontes podem ser encontrados em <http://www.dafont.com/> e <http://www.google.com/fonts/>.

PASSO-A-PASSO

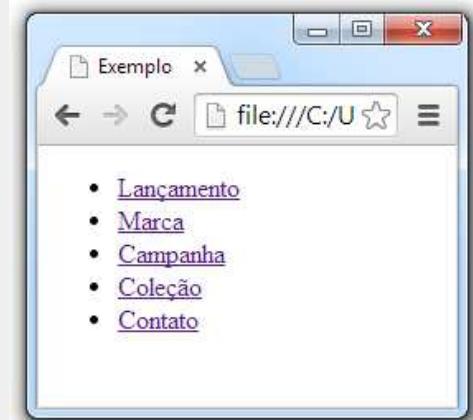
Construção de um menu horizontal

1. Inicialmente é necessário construir uma lista não ordenada. Os itens da lista devem ser links. Esta lista deve ficar dentro do elemento NAV.

```
<!DOCTYPE HTML>  
<html>  
  
<head>  
    <title>Exemplo</title>  
    <meta charset="UTF-8">  
    <link rel="stylesheet" href="css/style.css" />  
</head>  
  
<body>  
    <header>  
        <nav id="menu">  
            <ul>  
                <li><a href="#lancamento">Lançamento</a></li>  
                <li><a href="#marca">Marca</a></li>  
                <li><a href="#campanha">Campanha</a></li>  
                <li><a href="#colecao">Coleção</a></li>
```

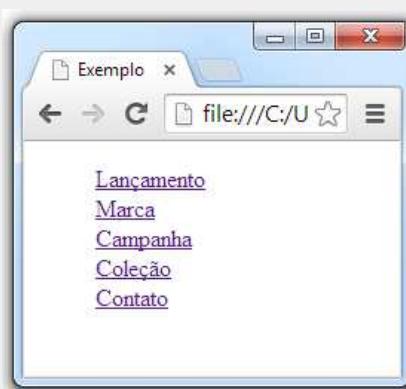
```
        <li><a href="#contato">Contato</a></li>
    </ul>
</nav>
</header>
</body>

</html>
```



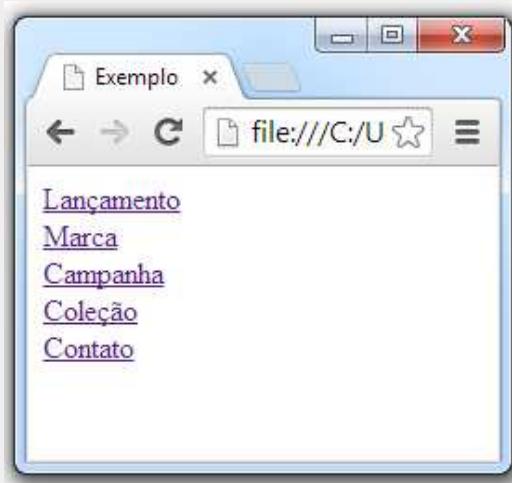
2. Com o CSS a formatação da lista deve ser removida.

```
#menu ul {
    list-style-type: none;
}
```



3. Por padrão as listas HTML possuem um recuo a esquerda. Para remover o recuo é necessário setar valor 0 para margin e para o padding.

```
#menu ul {
    margin:0;
    padding:0;
    list-style-type: none;
}
```



4. Para forçar que os elementos fiquem em uma mesma linha é necessário aplicar a formatação a seguir.

```
#menu ul li {  
    display: inline;  
}
```

[Lançamento](#) [Marca](#) [Campanha](#) [Coleção](#) [Contato](#)

5. Para remover o sublinhado do texto, é necessário retirar a formatação do link.

```
#menu ul li a {  
    text-decoration: none;  
}
```

[Lançamento](#) [Marca](#) [Campanha](#) [Coleção](#) [Contato](#)

6. Cada item da lista deve se comportar como uma caixa, para que a formatação possa ser aplicada. O elemento padding concede esse efeito. Para que o mesmo seja visível foi aplicada uma cor de fundo nos links presentes na lista. Também foi alterada a cor, o tamanho e o peso da fonte.

```
#menu ul li a {  
    text-decoration: none;  
    padding: 10px 30px;  
    color: #FFFFFF;  
    background-color: #1C1C1C;  
    font-size: 18px;  
    font-weight: bold;  
}
```

[Lançamento](#)

[Marca](#)

[Campanha](#)

[Coleção](#)

[Contato](#)

7. Para que o menu fique atraente, é interessante alterar a formatação quando o mouse for posicionado sobre o elemento. Foi alterada a cor da fonte e a cor de fundo.

```
#menu ul li a:hover {
    background-color:#999999;
    color:#000000;
    box-shadow: 0 5px 10px #CCC;
    text-shadow: 1px 1px 2px #FFFFFF;
}
```

8. Para que o menu fique centralizado na página, é necessário adicionar uma nova formatação.

```
#menu ul {
    margin:0;
    padding:0;
    list-style-type: none;
    text-align: center;
}
```

PASSO-A-PASSO

Construção formulário de contato

1. Inicialmente é necessário construir um formulário de contato. Aproveitando as tags da HTML5 o formulário foi colocado dentro de uma section. Com o intuito de deixar o formulário mais clean, os campos não possuem rótulo, apenas uma informação indicativa sobre o que ele representa usando o atributo placeholder.

Para auxiliar na formatação os botões estão dispostos dentro de uma div.

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Exemplo formulário contato</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="css/style.css" />
    </head>

    <body>
        <section>
            <header>
```

```

<h2>Entre em contato!</h2>
<h6>Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.</h6>
</header>

<form method="post" action="#">
    <input type="text" name="nome" id="nome" class="cxTexto" placeholder="Nome" />
    <input type="text" name="email" id="email" class="cxTexto" placeholder="Email" />
    <input type="text" name="assunto" id="assunto" class="cxTexto"
placeholder="Assunto" />
    <textarea name="mensagem" id="mensagem" placeholder="Mensagem"></textarea>

    <div class="pnBotoes">
        <a href="#" class="button">Enviar Mensagem</a>
        <a href="#" class="button">Limpar Formulário</a>
    </div>
</form>
</section>

</body>
</html>

```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

The screenshot shows a contact form with the following elements:

- Three input fields: "Nome", "Email", and "Assunto".
- Two buttons: "Enviar Mensagem" and "Limpar Formulário".
- A large text area labeled "Mensagem" with placeholder text.

2. Para melhorar a aparência do formulário é necessário aplicar algumas regras de CSS. Inicialmente devem ser definidas as regras aplicadas a todo o corpo da página. No trecho de código a seguir podem ser identificadas regras para alterar a cor do fundo e da fonte da página.

```

body{
    background-color: #303030;
    color: #FFFFFF;
}

```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

Nome Email Assunto

[Enviar Mensagem](#) [Limpar Formulário](#)

2. Para formatar o título foram aplicadas regras para alterar o tamanho da fonte, deixar o texto centralizado e com uma sombra. Além disso, foram atribuídos valores para margem do elemento, lembrando que o atributo h2 possui um tamanho de margem default que é diferente em cada navegador.

```
h2{
    font-size: 30px;
    text-align: center;
    text-shadow: -1px -1px 0px #181818;
    margin: 0 0 10px 0;
}
```

3. O subtítulo do formulário no momento é apenas um Lorem ipsum. Este tipo de texto é muito utilizado na diagramação, com a finalidade de verificar o layout, a tipografia e a formatação quando não é possível colocar o conteúdo real. As formatações aplicadas são similares a formatação do título, apenas um atributo referente ao peso da fonte.

```
h6{
    font-size: 18px;
    text-align: center;
    font-weight: lighter;
    margin: 0 0 10px 0;
}
```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

Nome Email Assunto

[Enviar Mensagem](#) [Limpar Formulário](#)

4. Para deixar o formulário centralizado é necessário que o mesmo tenha um tamanho definido. A propriedade margin com os valores 0 auto centraliza qualquer elemento de bloco.

```
form{
    width: 900px;
    margin: 0 auto;
}
```

5. Nas caixas de entrada de texto foi definida uma margem superior para atribuir um espaçamento entre os elementos. A propriedade padding aumenta o espaço entre o texto e a borda do elemento, neste caso foi setado o valor de 10 pixels. Também foi alterado o tamanho, a cor e o tipo de fonte utilizado. Uma melhor estilização foi possível aumentando a largura dos elementos, para que estes fossem renderizados na largura total do formulário. Outra alteração importante foi a retirada da borda padrão e a definição de uma borda arredondada e com sombreamento. A parte interna dos componentes ganhou uma cor de fundo diferente para um melhor destaque.

```
input, textarea{
    margin: 15px 0 0 0;
    border: 0;
    background: #282828;
    box-shadow: inset 0px 2px 5px 0px rgba(0,0,0,0.50), 0px 1px 0px 0px
    rgba(255,255,255,0.050);
    color: #FFFFFF;
    border-radius: 8px;
    width: 100%;
    padding: 10px;
    font-size: 18px;
    font-family: Times New Roman, serif;
}
```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

Nome

Email

Assunto

Mensagem

[Enviar Mensagem](#) [Limpar Formulário](#)

6. Para formatar o atributo placeholder é necessário manipular uma pseudo-class ::placeholder. No exemplo a seguir foi alterado a cor do texto.

```
::-moz-placeholder {
    color: #CDCDCD;
}
```

7. O campo mensagem deve ser um pouco maior, dessa forma deve ser aplicado uma altura para o mesmo.

```
textarea{
    height: 150px;
}
```

8. Também foi atribuída um sombreamento diferenciado para os campos com foco.

```
input:focus, textarea:focus{
    box-shadow: inset 0px 2px 5px 0px rgba(0,0,0,.05), 0px 1px 0px 0px
    rgba(255,255,255,.025), inset 0px 0px 2px 1px #74CAEE;
}
```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

[Enviar Mensagem](#) [Limpar Formulário](#)

9. Para retirar a formatação de link e melhorar o visual do botão algumas regras foram aplicadas. Inicialmente foi colocada uma margem no painel que abriga os botões para forçar um espaçamento com os demais campos do formulário. Além disso foram colocadas bordas, sombreamento, cor de fundo e espaçamento. Também foi retirada a formatação do link.

As pseudo-classes hover e active formatam o botão na passagem do mouse e quando o elemento é clicado.

```
.pnBotoes{
    margin-top: 50px;
}

.button {
    border-top: 1px solid #96d1f8;
    background: #65a9d7;
    background: -moz-linear-gradient(top, #3e779d, #65a9d7);
    padding: 15px 20px;
    border-radius: 9px;
    box-shadow: rgba(0,0,0,1) 0 1px 0;
    text-shadow: rgba(0,0,0,.4) 0 1px 0;
    color: white;
    font-family: Times New Roman, serif;
    text-decoration: none;
    font-size: 20px;
```

```
    font-weight: bold;
    margin-right: 10px;
}

.button:hover {
    border-top-color: #28597a;
    background: #28597a;
    color: #ffffff;
}

.button:active {
    border-top-color: #1b435e;
    background: #1b435e;
}
```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

10. Para centralizar os botões no formulário é necessário definir uma posição absoluta para o objeto e definir uma distância de 50% do lado esquerdo do documento (left: 50%). Para facilitar essa manipulação a mesma será aplicada na div que abriga os botões.

```
.pnBotoes{
    margin-top: 50px;
    position: absolute;
    left: 50%;
}
```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

Nome

Email

Assunto

Mensagem

Enviar Mensagem **Limpar Formulário**

11. O CSS não usa o centro do objeto como referência para o posicionamento, mas sim as extremidades. Sendo assim, o que ficará no centro será o canto esquerdo do objeto.

Para alinhar no centro é necessário identificar a largura do objeto (neste caso 400px). Identificada a largura é necessário definir uma margem do lado esquerdo com o valor negativo da metade da largura do objeto (200px).

```
.pnBotoes{  
    margin-top: 50px;  
    position: absolute;  
    left: 50%;  
    margin-left: -200px;  
}
```

Entre em contato!

Ornare nulla proin odio consequat sapien vestibulum ipsum sed lorem.

Nome

Email

Assunto

Mensagem

Enviar Mensagem **Limpar Formulário**

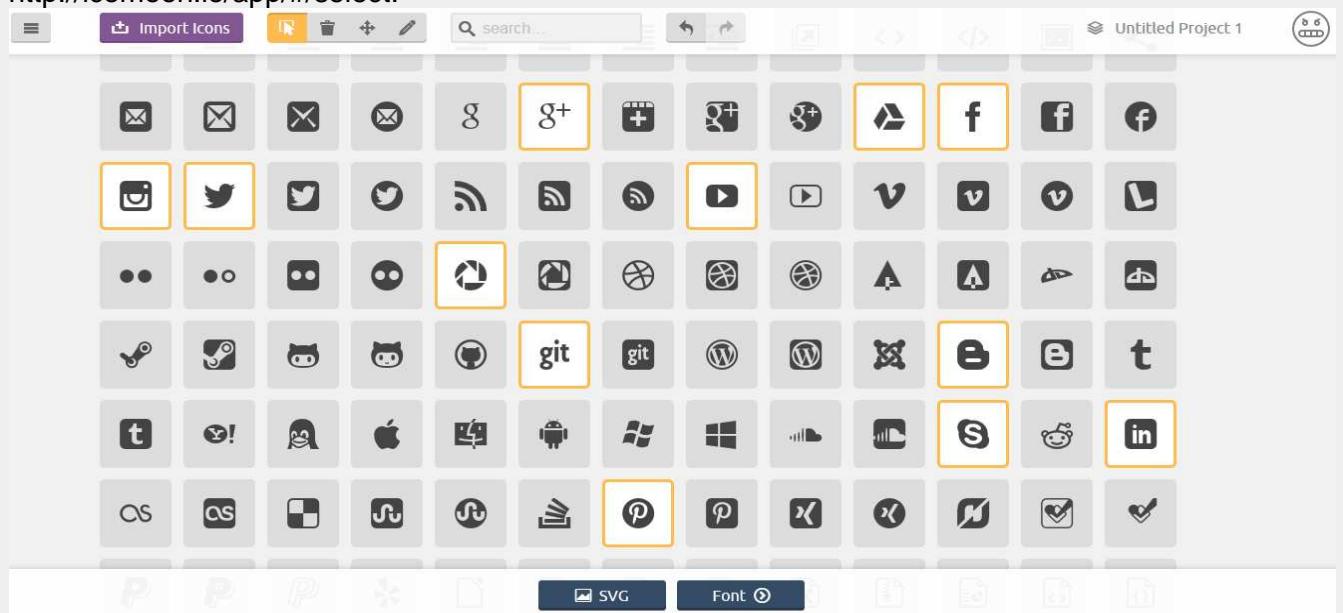
PASSO-A-PASSO

Construção barra redes sociais com font icons

Para a construção de um barra de links para as redes sociais é necessário a utilização de ícones. Antes do advento do CSS3 eram utilizadas imagens para este fim. No entanto, com o aumento da variedade de dispositivos com resoluções de telas diferentes e diversidade de densidade de pixels ficou inviável trabalhar desta forma.

Atualmente a melhor solução para utilização de ícones é a utilização de uma família de fonte no formato de ícone. Como exemplos de fonts de ícones podem ser citados o fontello e o icomoons. No exemplo a seguir será utilizado o icomoons.

1. Inicialmente é necessário acessar o repositório da fonte e escolher os ícones que serão utilizados: <http://icomoon.io/app/#/select>.



2. Clicando em Font, a página é redirecionada para os ícones escolhidos com seus respectivos identificadores.

Icon Name	Code
googleplus	e600
google-drive	e601
facebook	e602
instagram	e603
twitter	e604
youtube	e605
picassa	e606
github	e607
skype	e608
linkedin	e609
pinterest	e610

3. É necessário fazer o download. Na pasta do download juntamente com os arquivos de fontes tem um arquivo html e uma folha de estilo css com o demo dos ícones baixados. Para utilizá-los, é necessário copiar a pasta fonts para a pasta do projeto onde os ícones devem ser utilizados.

4. Os botões referentes as redes sociais devem ser inseridos em uma lista de links. A seguir está representado o código HTML da página. Em cada elemento da lista é atribuída uma classe para posterior formatação.

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Exemplo redes sociais</title>
        <meta charset="UTF-8">
        <link rel="stylesheet" href="css/style.css" />
    </head>

    <body>
        <div id="social">
            <ul>
                <li><a class="icon-googleplus" href="#"></a></li>
                <li><a class="icon-googledrive" href="#"></a></li>
                <li><a class="icon-facebook" href="#"></a></li>
                <li><a class="icon-instagram" href="#"></a></li>
                <li><a class="icon-twitter" href="#"></a></li>
                <li><a class="icon-youtube" href="#"></a></li>
                <li><a class="icon-picassa" href="#"></a></li>
                <li><a class="icon-github" href="#"></a></li>
                <li><a class="icon-blogger" href="#"></a></li>
                <li><a class="icon-skype" href="#"></a></li>
                <li><a class="icon-linkedin" href="#"></a></li>
                <li><a class="icon-pinterest" href="#"></a></li>
            </ul>
        </div>
    </body>
</html>
```

5. Para utilizar a família de fontes é necessário declará-la no CSS da página.

```
@font-face {
    font-family: 'icomoon';
    src:url('fonts/icomoon.eot?fe14q2');
    src:url('fonts/icomoon.eot?#iefixfe14q2') format('embedded-opentype'),
        url('fonts/icomoon.woff?fe14q2') format('woff'),
        url('fonts/icomoon.ttf?fe14q2') format('truetype'),
```

```
    url('fonts/icomoon.svg?fe14q2#icomoon') format('svg');
```

```
font-weight: normal;
```

```
font-style: normal;
```

```
}
```

6. Para adicionar o ícone é necessário criar um pseudo-elemento antes (before) de qualquer objeto que tenha no atributo class um valor que inicia com icon-.

```
[class^="icon-"]:before, [class*="icon-"]:before {
```

```
    font-family: 'icomoon';
```

```
    font-style: normal;
```

```
    speak: none;
```

```
}
```

7. O pseudo-elemento before trabalha em parceria com uma propriedade chamada content. Sendo assim em cada classe é atribuída na propriedade content, o código do ícone correspondente.

```
.icon-googleplus:before {
```

```
    content: '\e600';
```

```
}
```



```
.icon-goolgedrive:before {
```

```
    content: '\e601';
```

```
}
```



```
.icon-facebook:before {
```

```
    content: '\e602';
```

```
}
```



```
.icon-instagram:before {
```

```
    content: '\e3';
```

```
}
```



```
.icon-twitter:before {
```

```
    content: '\e603';
```

```
}
```



```
.icon-youtube:before {
```

```
    content: '\e6';
```

```
}
```



```
.icon-picassa:before {
```

```
    content: '\e604';
```

```
}

.icon-github:before {
    content: '\e605';
}

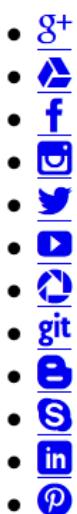
.icon-blogger:before {
    content: '\e606';
}

.icon-skype:before {
    content: '\ec';
}

.icon-linkedin:before {
    content: '\ed';
}

.icon-pinterest:before {
    content: '\e607';
}
```

8. Acessando a página é possível verificar que os ícones já foram carregados. No entanto, a formatação segue o padrão de formatação das listas.



9. Para melhorar o design da página, algumas regras de formatação foram definidas.

```
#social ul {
    margin:0;
```

```
padding:0;
list-style-type: none;
text-align: center;

}

#social ul li {
    display: inline;
}

#social ul li a {
    text-decoration: none;
    padding: 10px;
    color: #FFFFFF;
    background-color: #1C1C1C;
    font-size:18px;
    font-weight:bold;
    border-radius: 3px;
}

#social ul li a:hover {
    color:#000000;
    box-shadow: 0 5px 10px #CCC;
    text-shadow: 1px 1px 2px #FFFFFF;
    background: #999999;
}
```

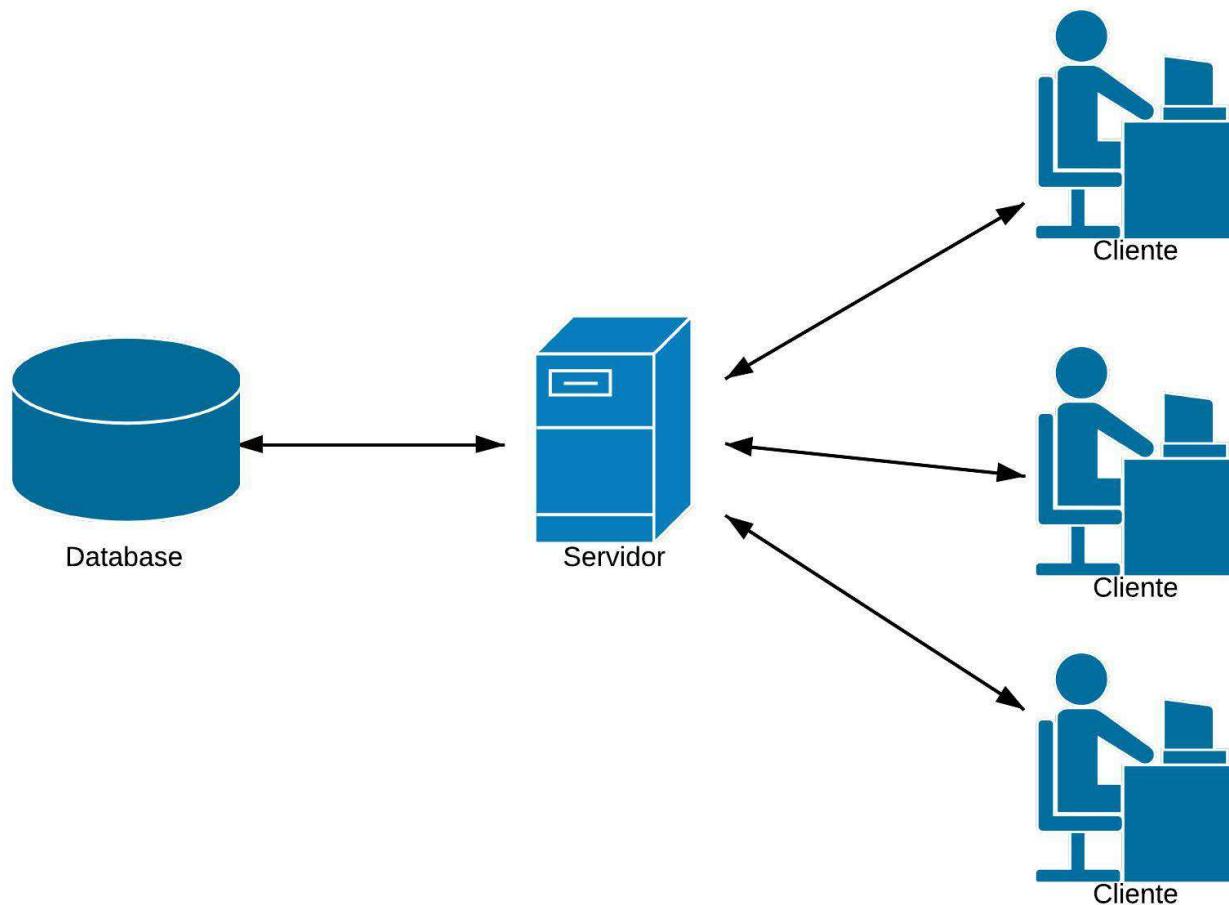


JAVA EE

Com a popularização da internet no cenário mundial, as páginas web estão cada vez mais presentes no dia-a-dia da população. Como tendência desse crescimento, as aplicações corporativas estão migrando para a plataforma web.

As aplicações web apresentam diversas vantagens se comparadas com as aplicações desktop. A principal delas é que não existe a necessidade de instalar a aplicação em cada máquina onde será utilizada. Toda a programação fica armazenada em um servidor - ou vários servidores, dependendo da infraestrutura - e os usuários conseguem acesso ao software através de um navegador.

Como não existe a necessidade de instalar o software, o usuário pode acessar a aplicação de qualquer computador que possua um browser. A portabilidade também é muito maior, visto que o comportamento dos navegadores em sistemas operacionais distintos é praticamente o mesmo. O processo de atualização de versão dos softwares também é simplificado, pois apenas o servidor precisa ser atualizado. A figura a seguir demonstra a arquitetura padrão de um software utilizando JEE.



Na linguagem Java, para o desenvolvimento de aplicações que rodem em um servidor, existe um conjunto de especificações já padronizadas. Essas especificações fazem parte da Java Enterprise Edition (JEE) e são publicadas pela Oracle e pelo Java Community Process (JCP) e podem ser consultadas em <http://jcp.org>.

A idéia é que o desenvolvedor codifique apenas a lógica de negócios da aplicação. Recursos como portabilidade, robustez, segurança entre outros, devem ser fornecidos pelas implementações do JEE, diminuindo o tempo de desenvolvimento, os riscos do projeto e os problemas com manutenção.

Tecnologias do Java EE

Dentre as APIs disponibilizadas pela JEE, algumas são bastante utilizadas no desenvolvimento de aplicações. A documentação das mesmas estão publicadas na JCP e estão organizadas em JSRs (Java Specification Requests). Cada JSR possui um número de identificação e contempla todas as informações sobre a API na qual se refere. Segue uma lista com as JSRs mais utilizadas:

- Java Persistence (JSR 317): Como já mencionado em capítulos anteriores, a JPA é uma API para mapeamento objeto relacional. Embora faça parte da especificação do Java EE, ela também pode ser usada em aplicações JSE (Java Standard Edition).
- Java Servlet (JSR 315): Esta API proporciona a inclusão de conteúdo dinâmico em um Container JEE.
- Java Server Pages (JSR 245): Permite a criação de páginas dinâmicas de modo mais simples do que com a utilização de Servlets. Um JSP faz as funções de um script em uma página web e é compilado como servlet em seu container. Uma página JSP contém dois tipos de texto: estático, representado por marcações HTML, WML, ou XML, e elementos dinâmicos JSP.
- Java Server Faces (JSR 314): Tecnologia desenvolvida para permitir o desenvolvimento de aplicações web com a aparência e facilidade de uma aplicação desktop.
- Java Message Service API - JMS (JSR 914) : API que padroniza a troca de mensagens entre programas distintos, com o uso de mecanismos de criação, envio, recebimento e leitura de mensagens.
- Java Authentication and Authorization Service - JAAS (JSR) : API para padrão do Java para segurança.
- Java Transaction API - JTA (JSR 907): Especifica o controle de transação pelo Container.
- JavaMail API (JSR 919): API para suporte ao envio e recebimento de e-mails em Java.

Servidor de aplicação

A JEE é apenas uma especificação e como tal, necessita de uma implementação do conjunto de normas e regras presentes. Cabe aos fornecedores de servidores de aplicação implementar essas regras definidas pela JEE. A especificação Java EE é implementada pelos servidores de aplicação.

Existem inúmeros servidores de aplicação no mercado, dentre os mais conhecidos estão o JBoss Application Server da RedHat, o GlassFish da Sun, o Apache Geronimo da Apache, o WebLogic Application Server da Oracle e o IBM Websphere Application Server da IBM.

Os servidores de aplicação necessitam implementar toda a especificação Java EE e são responsáveis pelos mecanismos de tolerância a falhas, balanceamento de carga, gerenciamento de componentes, gerenciamento de transações e console de gerenciamento. Dessa forma, em um projeto JEE, o desenvolvedor deve preocupar-se apenas com a resolução de problemas referentes à regras de negócio e não com questões de infraestrutura.

Servlet Container

O Java EE possui várias especificações, entre elas, algumas específicas para lidar com o desenvolvimento de uma aplicação Web: JSP, Servlets, JSF. Em alguns casos, não existe a necessidade de um servidor que implemente o JEE completo mas apenas a parte web da especificação. Aplicações de pequeno e médio porte se encaixam nessa categoria.

Para essas aplicações existem pequenos servidores que implementam apenas parte do JEE.

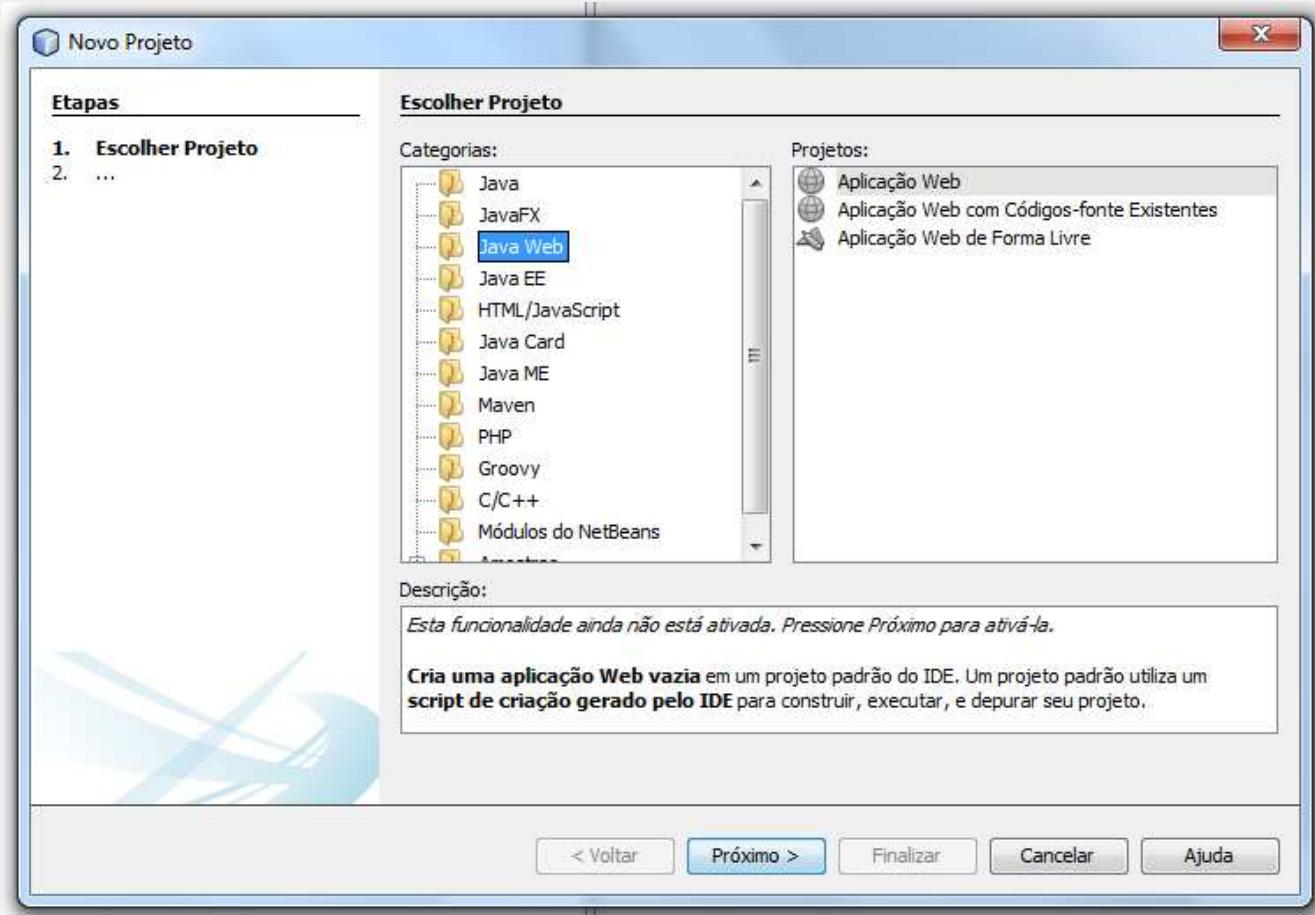
Estes servidores são chamados de ServletContainer. O ServletContainer mais famoso do mercado é o Apache Tomcat.

PASSO-A-PASSO

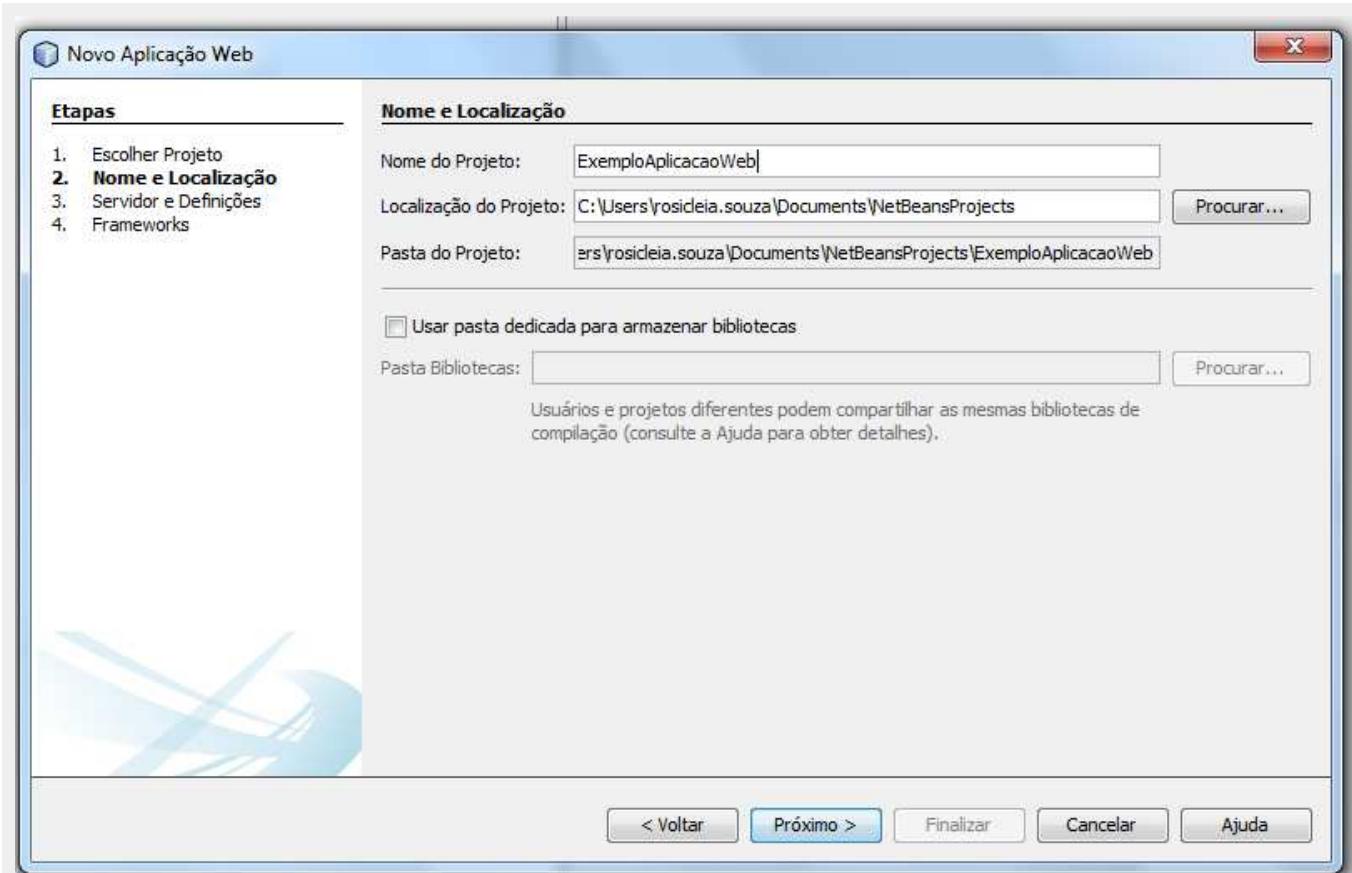
Configurando o ambiente

Para este exemplo será utilizada a ferramenta Netbeans e como servidor de aplicação o Glassfish. O Glassfish foi escolhido por já vir instalado junto com o Netbeans, ser open source e gratuito e estar ganhando força no mercado nos últimos anos. Ele também é um dos poucos a suportar o JEE 7.

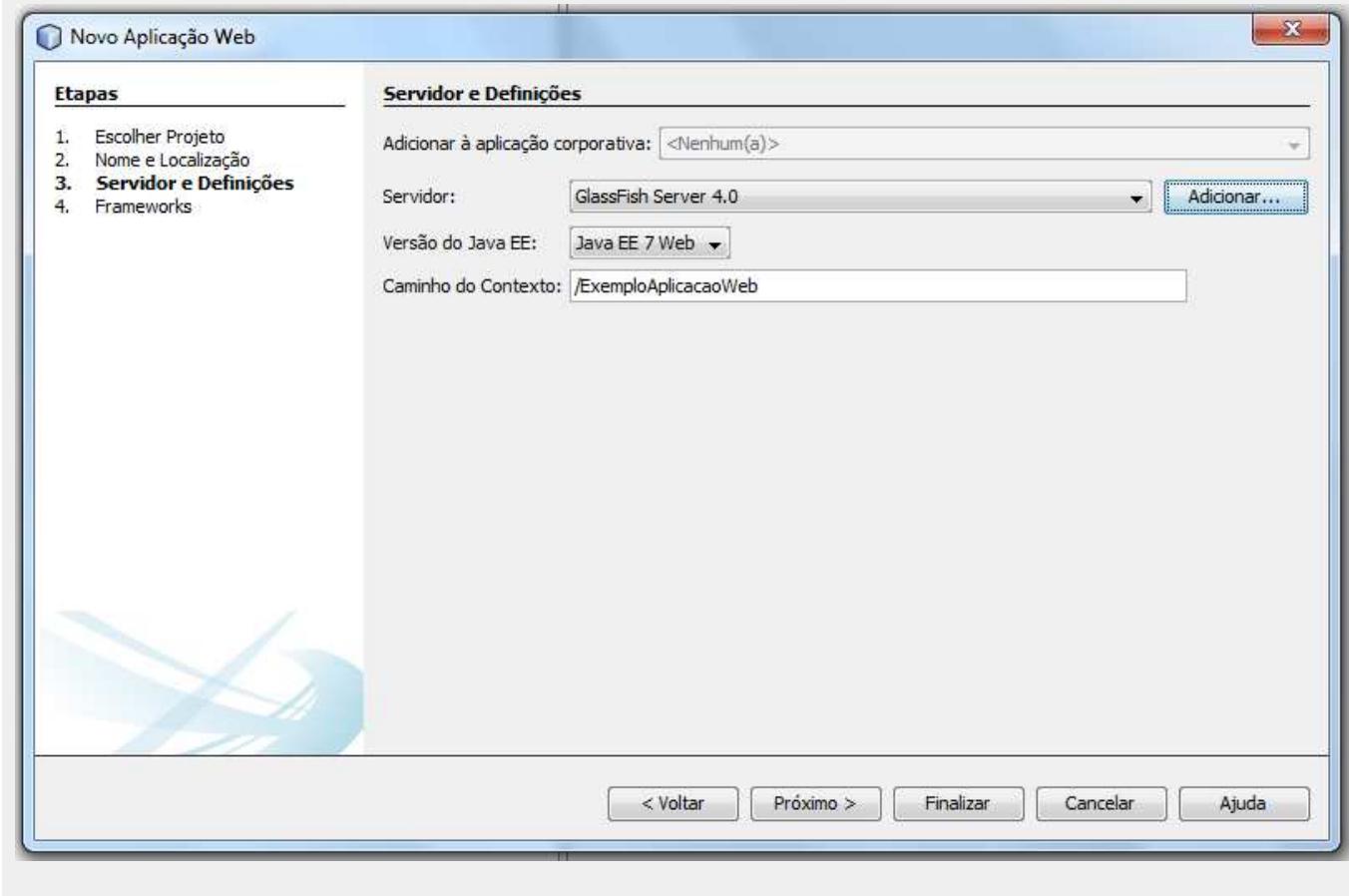
1. Na criação do projeto, guia Escolher Projeto, a Catgoria Java Web deve ser selecionada e em Projetos Aplicação Web.



2. Na guia Nome e Localização, é necessário dar um nome ao projeto e identificar a pasta onde o mesmo ficará armazenado.



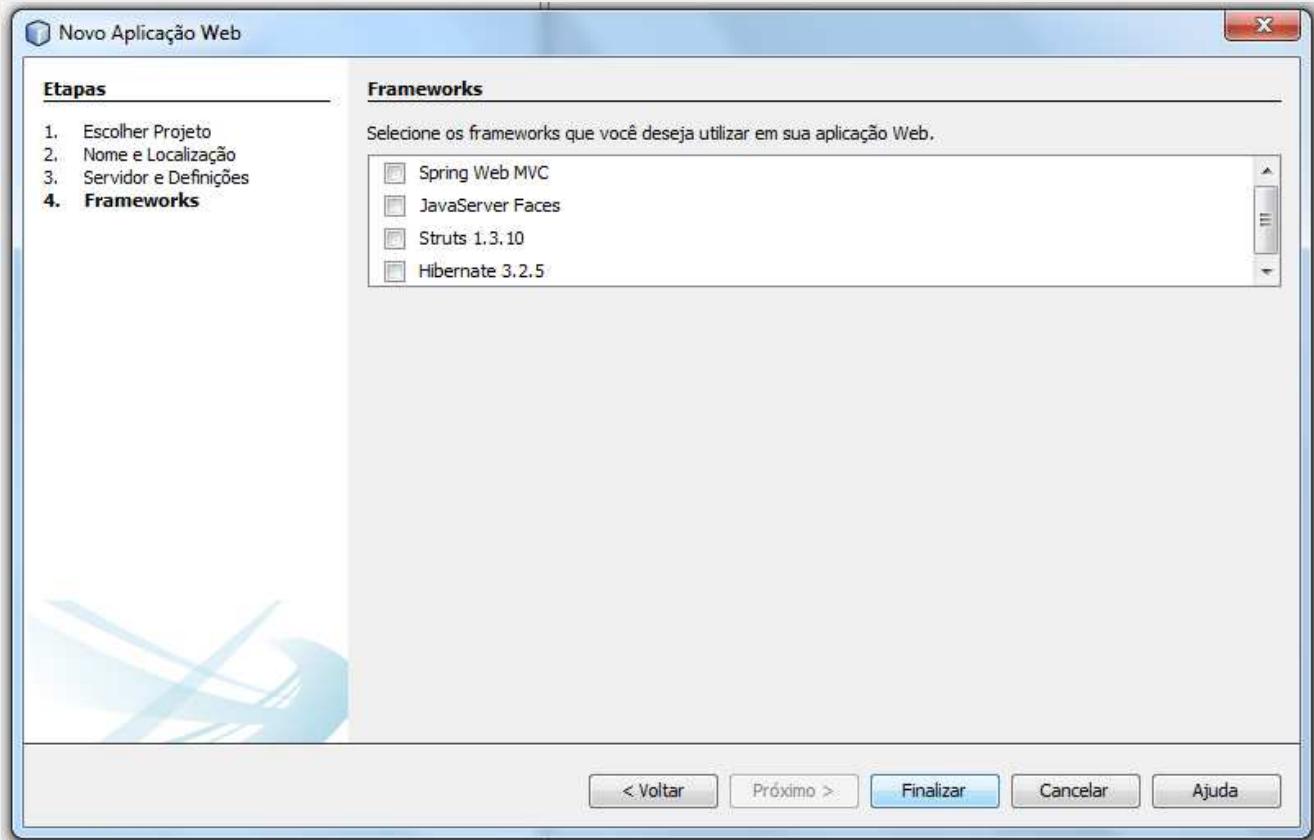
3. Na guia Servidores e Definições deve ser escolhido como Servidor o Glassfish 4.0.



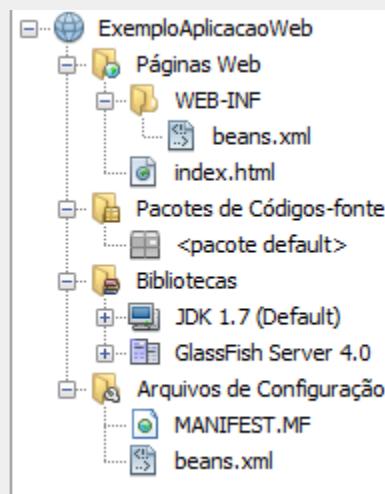
4. Em Java EE, o caminho do contexto é o caminho no qual a aplicação pode ser acessada após ser implantada no servidor, ou seja, os contextos servem para diferenciar sites ou projetos distintos em um mesmo servidor. Por padrão o caminho do contexto gerado é o mesmo nome da aplicação. Nesta configuração apresentada, considerando a porta default do glassfish - 8080 - para o acesso da aplicação deve ser utilizado o seguinte endereço:

http://localhost:8080/ExemploAplicacaoWeb/

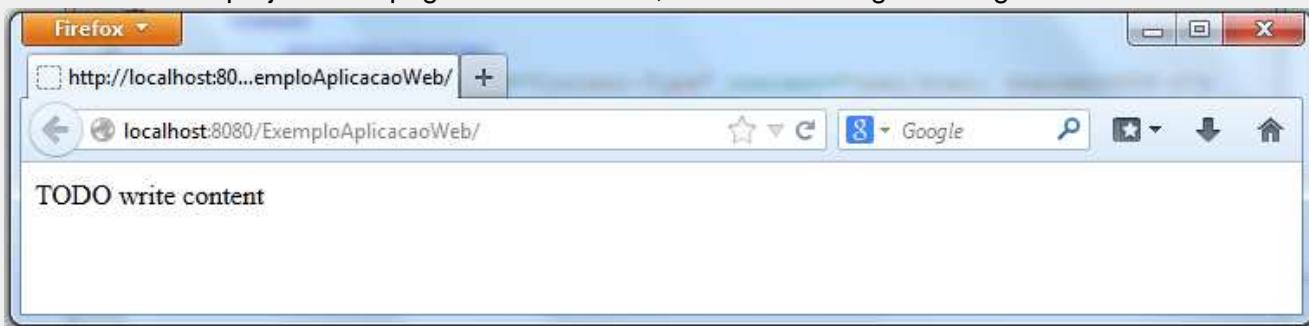
5. Na guia Frameworks, neste momento nenhum deles será selecionado.



6. O projeto criado obedece o padrão de pastas listado a seguir. A seguir, serão detalhados os conteúdos destas pastas. Por ora, é importante perceber o servidor de aplicação e a versão do Java EE utilizados.



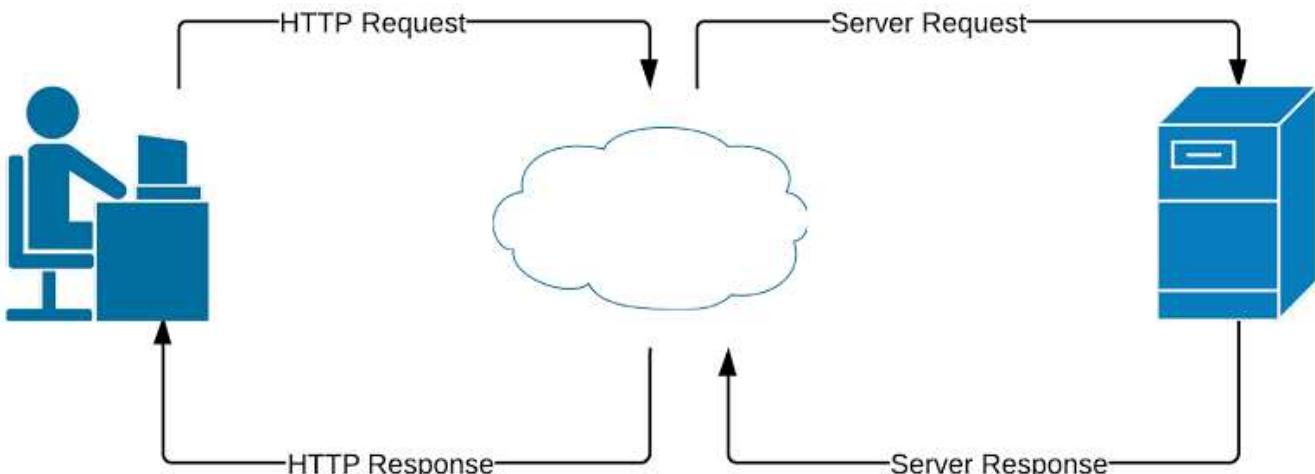
7. Ao executar o projeto uma página web é aberta, semelhante à figura a seguir.



Nas aplicações Java Web, alguns diretórios e arquivos devem existir por padrão. Segue uma descrição sucinta dos mesmos:

- Páginas Web: Este diretório representa o conteúdo raiz do projeto. Nessa pasta deve ser armazenado o arquivo index.html e todos os arquivos .html, .jsp, .xhtml, .jsf, .css e imagens. Dentro desta pasta pode ser criada uma estrutura de pastas para organização dos arquivos. Tudo o que é colocado dentro desta pasta é acessível na URL do projeto.
- WEB-INF: Essa pasta faz parte da especificação JEE e armazena as configurações e recursos necessários para o projeto rodar no servidor. Todo o projeto Java Web precisa ter esta pasta. Ela é inacessível na URL do projeto, o que garante que os usuários não conseguem visualizar o conteúdo dos arquivos presentes nesta pasta.
- web.xml: Também é chamado de descritor de implantação. Este arquivo contém as informações para realizar os ajustes dos parâmetros da aplicação, como mapeamentos, segurança, filtros, entre outros. Este arquivo fica dentro da pasta WEB-INF e sua utilização é necessária apenas se existir a necessidade de adicionar servlets e/ou parâmetros de inicialização.
- Pacotes de Código-fonte: Nesta pasta ficam armazenados os arquivos .java da aplicação.
- Bibliotecas: Nesta pasta ficam armazenadas todas as bibliotecas necessárias para que o projeto seja executado corretamente.

O desenvolvimento Web e o protocolo HTTP



Em aplicações Web, o usuário não possui o software instalado em seu computador. Toda a aplicação fica hospedada em um servidor e o acesso é feito utilizando o protocolo HTTP através do

modelo request-response.

No modelo request-response, o cliente solicita a execução de uma tarefa - request. Por sua vez o servidor recebe a requisição, realiza e responde para o cliente - response. Sendo assim, quando o endereço de uma página é digitado em um browser, uma requisição está sendo gerada ao servidor. O mesmo ocorre quando um botão ou um link é acionado. Por sua vez, o navegador, entende somente HTML, dessa forma os dados são trafegados utilizando somente esta linguagem.

PASSO-A-PASSO

Exemplo página html

1. No projeto criado no passo-a-passo anterior, o arquivo index.html deve conter o código mostrado abaixo.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Exemplo página HTML</title>
        <meta charset="UTF-8">
    </head>
    <body>
        <h1>Primeira página HTML</h1>
        <p>Essa é a primeira página usando HTML.</p>
    </body>
</html>
```

2. Ao executar a aplicação, deve ser mostrada a janela ilustrada abaixo.



Servlets

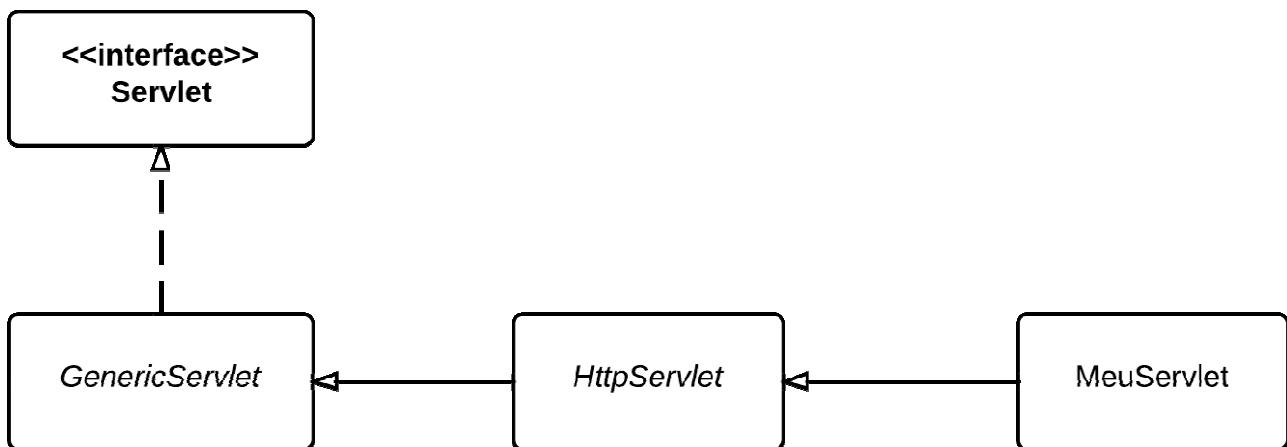
No início da era da internet, as páginas web eram todas estáticas, ou seja, não existia nenhuma interação com o usuário. Porém, com o passar do tempo, notou-se que a web tinha enorme poder de comunicação e para a interação com os usuários, seriam necessárias páginas que gerassem conteúdo dinamicamente baseadas nas requisições dos mesmos.

Na plataforma Java, a tecnologia capaz de gerar páginas dinâmicas são as Servlets. Embora, existam atualmente diversos frameworks para facilitar o desenvolvimento em Java Web, todos eles criam um nível de abstração sobre Servlets.

As Servlets possuem como responsabilidade o tratamento de solicitações. Em outras palavras, as Servlets recebem requisições provenientes de um browser ou outro cliente HTTP qualquer, obtêm os dados embutidos e produz uma resposta.

```
@WebServlet(urlPatterns = {" /MeuServlet"})
public class MeuServlet extends HttpServlet {
```

A nível de implementação uma Servlet é uma classe Java que estende da interface javax.servlet.http.HttpServlet. Existem duas classes na API de servlets que implementam esta interface: GenericServlet, que como o próprio nome diz é uma classe genérica que atende requisições de qualquer protocolo e sua subclasse HttpServlet que atende requisições HTTP.



O servlet permite por meio de programação Java definir todo o código HTML que será exibido ao cliente. No entanto, o programador precisa preparar todo o texto que será mostrado ao cliente, sendo ele estático ou não.

Ciclo de vida das Servlets

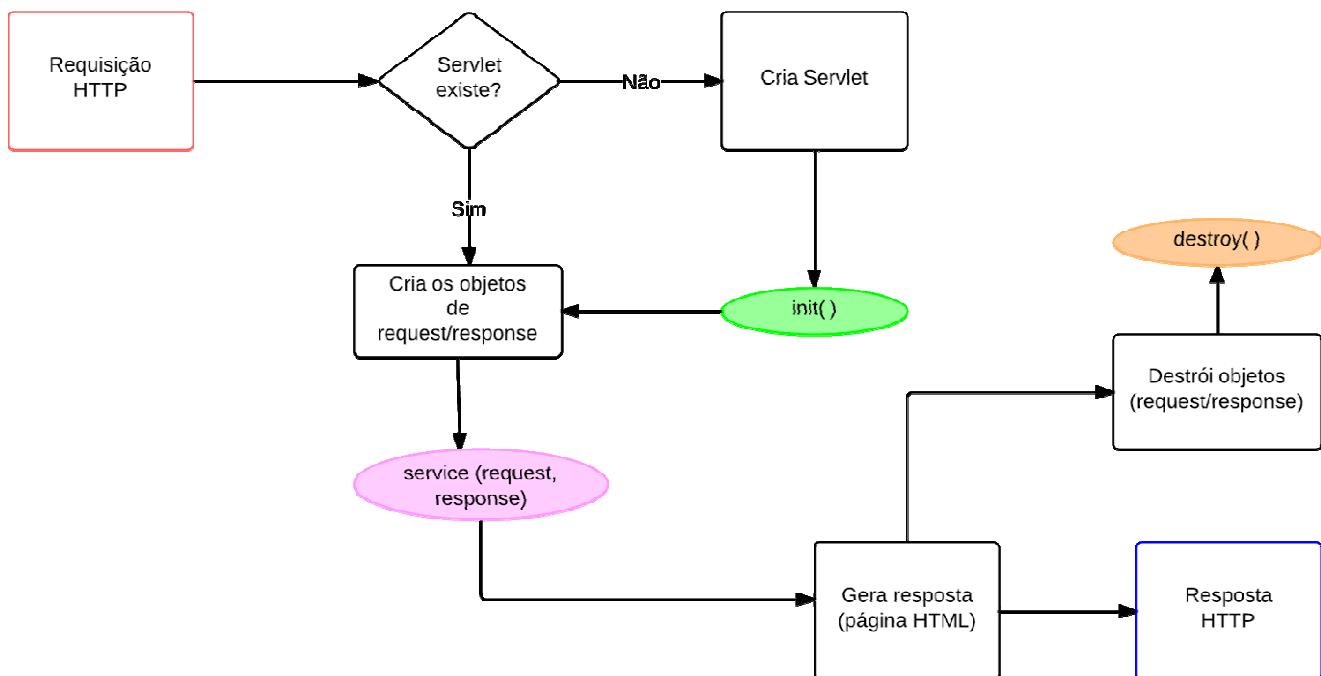
O funcionamento de uma Servlet é formado basicamente por três etapas:

- Inicialização: O servidor é responsável pela inicialização da servlet pelo método init e a utiliza durante todo o seu período ativo. O container pode criar o servlet no momento da sua inicialização ou quando chegar a primeira requisição.
- Chamadas a métodos de serviço: São as requisições enviadas ao servidor durante todo o ciclo de vida das Servlets. O método service é o responsável pelo tratamento dessas requisições.
- Finalização: O servidor destrói a Servlet, carregada na inicialização. Esse processo é efetuado

através do método `destroy`.

O ciclo de vida de uma Servlet é controlado pelo container. Cada requisição recebida pelo servidor é passada para o container que tem como responsabilidade carregar a classe, criar e inicializar uma instância da mesma, controlar as requisições até a remoção da Servlet da memória.

Na inicialização do servidor, o servlet é carregado, porém não entra em funcionamento. No momento em que uma requisição chega ao servidor, caso a servlet ainda não esteja carregada - isso ocorre na primeira requisição -, a mesma é inicializada através do método `init`. A partir deste momento todas as requisições são tratadas pelo método `service`. O container cria um objeto de requisição (`ServletRequest`) e um objeto de resposta (`ServletResponse`) e após chama o método `service`, passando os objetos criados como parâmetro. Após o envio da resposta, os objetos são destruídos. Esse processo se repete até que o Servlet seja destruído, através do método `destroy`.



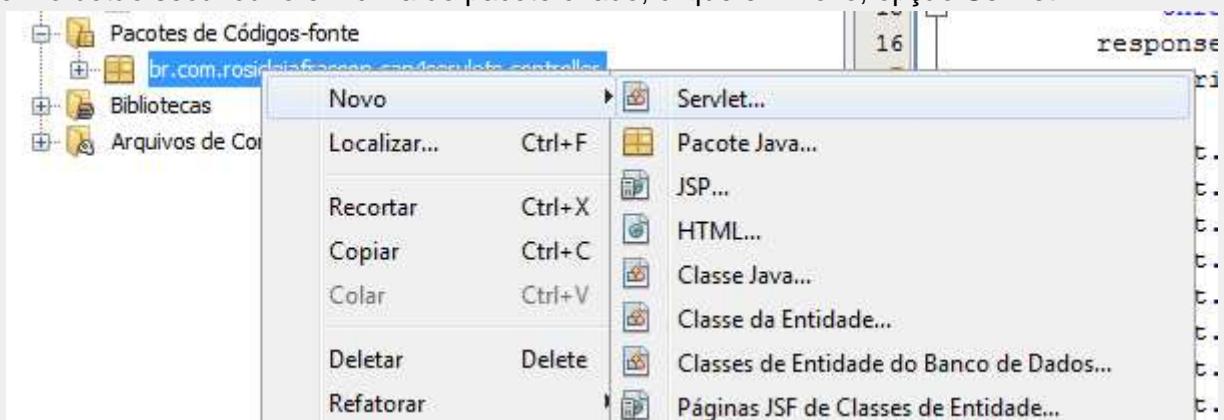
O fato da Servlet permanecer carregada durante o tempo que o servidor está ativo e a aplicação carregada, permite que dados armazenados em variáveis de classe persistam ao longo de diversas requisições. Isso garante que processos custosos sejam efetuados apenas uma vez durante todo o ciclo de vida da Servlet. É o caso, por exemplo, de uma conexão com a base de dados.

PASSO-A-PASSO

- Após a criação de um projeto Java web no Netbeans é recomendada a criação de um pacote para o armazenamento da classe Servlet. A estrutura do projeto deve ser similar a figura a seguir:



2. Com o botão secundário em cima do pacote criado, clique em Novo, opção Servlet.



3. O Netbeans gera uma estrutura padrão de Servlet, como mostrado a seguir.

```

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(urlPatterns = {" /PrimeiraServlet"})
public class PrimeiraServlet extends HttpServlet {

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
}

```

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet PrimeiraServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet PrimeiraServlet at " + request.getContextPath() +
"</h1>");
        out.println("</body>");
        out.println("</html>");
    } finally {
        out.close();
    }
}

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on
the left to edit the code.">
/**
 * Handles the HTTP
 * <code>GET</code> method.
 *
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Handles the HTTP
 * <code>POST</code> method.
 *
 * @param request servlet request

```

```
* @param response servlet response
* @throws ServletException if a servlet-specific error occurs
* @throws IOException if an I/O error occurs
*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Returns a short description of the servlet.
 *
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
}// </editor-fold>
}
```

É importante perceber que a classe possui a anotação @WebServlet contendo o nome da Servlet e a URL que deve ser utilizada para acesso da mesma. Em versões posteriores do java o mapeamento deveria ser feito no web.xml, atualmente este mapeamento é feito utilizando apenas a anotação.

Após a execução do projeto, a Servlet pode ser acessada em: <http://localhost:8080/Cap4Servlets/PrimeiraServlet> e deve exibir o conteúdo mostrado na imagem a seguir.



Vale lembrar que o conteúdo mostrado na tela está presente dentro do método processRequest, que é responsável por processar as requisições. No exemplo apresentado, a Servlet não executa nenhuma lógica e apenas exibe um conteúdo estático na tela. O método getWriter da variável response obtém um objeto que representa a saída a ser enviada ao usuário. Por sua vez, o objeto criado out do tipo PrintWriter imprime a resposta para o cliente.

Método Get

O método GET, quando utilizado em formulários, indica que os valores dos parâmetros são passados através da URL junto dos nomes dos mesmos, separados por &. Separa o endereço da URL dos dados do formulário com um ponto de interrogação '?'.

O método GET deve ser usado apenas quando a quantidade de informações passadas for pequena e não for secreta, como em uma pesquisa, por exemplo.

Método Post

Utilizando o método POST, os dados são passados dentro do corpo do protocolo HTTP, sem aparecer na URL que é mostrada no navegador.

Esse método é utilizado em formulários de cadastro, onde os dados não podem ser visualizados ou alterados na URL.

Métodos doGet e doPost

Os métodos doGet e/ou doPost devem ser implementados na Servlet indicando como será efetuado o tratamento do formulário. Os métodos doGet e doPost são invocados pelo método service e cabe a este detectar o tipo da requisição recebida e invocar o método adequado para tratá-la.

Vale lembrar que requisições get são processadas pelo método doGet e requisições post são processadas pelo método doPost. Ambos os métodos possuem dois parâmetros HttpServletRequest e HttpServletResponse.

PASSO-A-PASSO

Método Get

1. Crie um projeto Java Web no Netbeans e altere o arquivo index.html de modo que fique semelhante ao trecho de código listado a seguir.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Método GET</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width">
    </head>
    <body>
        <form name="teste" method="GET" action="/Cap4ServletGet/ServletController">
            <table>
                <tr>
                    <td>Nome: </td>
                    <td><input type="text" name="nome"></td>
```

```
</tr>

<tr>
    <td>Idade: </td>
    <td><input type="text" name="idade"></td>
</tr>

<tr>
    <td></td>
    <td>
        <input type="submit" value="Salvar"/>
        <input type="reset" value="Limpar"/>
    </td>
</tr>
</table>
</form>
</body>
</html>
```

2. Em seguida, deve ser criada uma Servlet com os comandos abaixo.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name="ServletController", urlPatterns = {"/ServletController"})
public class ServletController extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter pw = response.getWriter();
        String nome = request.getParameter("nome");
        int idade = Integer.parseInt(request.getParameter("idade"));
        pw.println("<html><body>");
        pw.println("<h1>Valores digitados</h1>");
        pw.println("Nome: " + nome);
        pw.println("</br>");
        pw.println("Idade: " + idade);
```

```
        pw.println("</body></html>");  
    }  
  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        processRequest(request, response);  
    }  
  
    @Override  
    protected void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        processRequest(request, response);  
    }  
}
```

Como pode ser visualizado, a classe ServletController é um Servlet que implementa o método doGet. Sendo assim, responde requisições do tipo HTTP via método GET. A resposta desse Servlet é processada através dos valores dos parâmetros que são passados através da URL. O método getParameter é responsável por acessar as informações passadas pelos parâmetros na URL.

Vale lembrar que como os dados são passados via URL, existe a possibilidade do usuário efetuar a alteração do mesmo pela URL. Por este motivo, requisições via método get devem ser utilizadas apenas em operações que dados da base de dados não serão alterados.

Método Post

1. Crie um projeto Java web no Netbeans e altere o arquivo index.html como no código a seguir. É importante perceber que o formulário possui o método POST.

```
<!DOCTYPE html>  
<html>  
    <head>  
        <title>Método POST</title>  
        <meta charset="UTF-8">  
        <meta name="viewport" content="width=device-width">  
    </head>  
    <body>  
        <form name="teste" method="POST"  
        action="/Cap4ServletPost/ServletController">  
            <table>  
                <tr>
```

```
<td>Nome: </td>
<td><input type="text" name="nome"></td>
</tr>

<tr>
    <td>Idade: </td>
    <td><input type="text" name="idade"></td>
</tr>

<tr>
    <td></td>
    <td>
        <input type="submit" value="Salvar"/>
        <input type="reset" value="Limpar"/>
    </td>
</tr>
</table>
</form>
</body>
</html>
```

2. Em seguida, crie uma Servlet e deixe a mesma com os seguintes comandos.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name="ServletController", urlPatterns = {"/ServletController"})
public class ServletController extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        PrintWriter pw = response.getWriter();
        String nome = request.getParameter("nome");
        int idade = Integer.parseInt(request.getParameter("idade"));
        pw.println("<html><body>");
        pw.println("<h1>Valores digitados</h1>");
        pw.println("Nome: " + nome);
        pw.println("</br>");
```

```
        pw.println("Idade: " + idade);
        pw.println("</body></html>");
    }

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
}
```

Como pode ser visualizado este exemplo é semelhante ao anterior, execeto o fato de usar o método post ao invés do método get. A diferença primordial dessas duas abordagens é que no método get os dados aparecem na URL e no método post os dados não são exibidos no navegador.

JSP

Os Servlets permitem que conteúdos dinâmicos sejam adicionados as páginas HTML. Porém, o uso de Servlets exige que grande parte do código estático HTML seja escrito nas Servlets como código Java, diminuindo assim a legibilidade do código e dificultando a manutenção das páginas. Outro problema encontrado com o uso das Servlets, é que o trabalho de deixar as páginas mais bonitas, desenvolvido pelo designer, ficou muito difícil, visto que este profissional, na maioria das vezes, não possui conhecimento na linguagem de programação Java.

A tecnologia JSP (Java Server Pages) permite que seja adicionado comportamento dinâmico nas páginas HTML, ou seja, um arquivo com extensão .jsp pode possuir elementos HTML e códigos Java embutidos. A grande vantagem na utilização desta tecnologia, se comparada, as demais existentes como ASP e PHP, é que a mesma é independente de plataforma. Os trechos de códigos dinâmicos, dentro dos arquivos JSP são delimitados com <% e %>.

De maneira oculta, as páginas JSP são compiladas em Servlets. A compilação é feita no primeiro acesso. Nos acessos subsequentes, a requisição é redirecionada ao servlet que foi gerado a partir do arquivo JSP. Levando em consideração as afirmativas apresentadas, o primeiro acesso a um arquivo JSP é sempre mais lento, por conta do processo de compilação.

A tecnologia JSP permite separar a apresentação da lógica de negócio, facilitando o desenvolvimento de aplicações robustas, onde o programador e o designer podem trabalhar em um mesmo projeto de forma independente. Com JSP também é possível reutilizar conteúdos dinâmicos.

Elementos de uma página JSP

- Diretivas: Fornecem informações gerais acerca da página JSP. Sua sintaxe é <%@ diretiva %>. As principais diretivas são page, include e taglib.

- page: A diretiva page é responsável por fornecer informações sobre a página JSP. Deve ser usada no início da página. No exemplo mostrado a seguir, o atributo contentType indica o tipo de saída do documento e o atributo pageEncoding define o conjunto de caractere a ser utilizado.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

- include: A diretiva include permite que um arquivo texto seja inserido em uma página JSP. Esta diretiva pode ser utilizada em qualquer lugar da página e quantas vezes forem necessárias. No exemplo de código mostrado a seguir, deve ser adicionado ao arquivo corrente, o conteúdo do arquivo header.html.

```
<%@include file="header.html"%>
```

- taglib: A diretiva taglib indica que um conjunto de tags personalizadas estará à disposição para ser utilizada. Também associa um prefixo a uma biblioteca de tags.

```
<%@taglib prefix="teste" uri="taglib.tdl"%>
```

- Declarações: As declarações permitem a adição de código Java, ou seja, qualquer código válido em uma classe Java, pode estar dentro de uma declaração, como por exemplo, declaração e inicialização de variáveis, declaração e implementação de métodos. Um detalhe importante é que as declarações não geram saídas. Deste modo, as mesmas são usadas em conjunto com expressões ou scriptlets. As declarações devem estar entre <%! e %>.

```
<%!int i = 0;%>
```

- Expressão: As expressões geram saídas para a página. Utilizam-se as expressões para instruções que retornam valor.

```
<%=area;%>
```

- Scriptlet: Os scriptlets são utilizados em processamentos mais complexos do que as expressões. Dentro dos scriptlets é possível inserir condicionais e loops.

```
<%
    double base = Double.parseDouble(request.getParameter("base"));
    double altura = Double.parseDouble(request.getParameter("altura"));
    double area = base * altura;
    double perimetro = (base + altura) * 2;
%>
```

- Comentários de uma página JSP: Os comentários indicam ao compilador que trechos de código devem ser ignorados. Os comentários em JSP devem ser escritos entre <%-- e --%>.

```
<%--
    Document      : resultado
    Created on   : 20/12/2013, 11:33:30
    Author        : Rosicléia Frasson
--%>
```

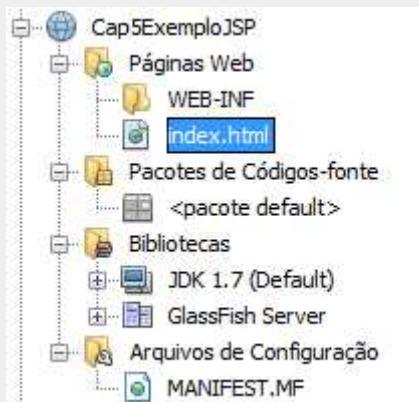
Ciclo de vida de uma página JSP

No momento em que uma página JSP é requerida através de um browser, esta será processada pelo servidor. Existe diferença entre o primeiro acesso à página e os demais. No primeiro acesso, a página é transformada em uma Servlet, compilada, gerando dessa forma o bytecode. A partir do bytecode é gerada a página HTML que é enviada de volta ao browser do cliente. O processo de montagem da página HTML pode requerer informações da base de dados, arquivos texto, entre outros. Caso, a requisição seja para uma página anteriormente acessada, uma verificação é feita com o intuito de averiguar se houve alguma mudança. Em caso afirmativo, é feita a compilação novamente. Caso contrário o HTML é montado.

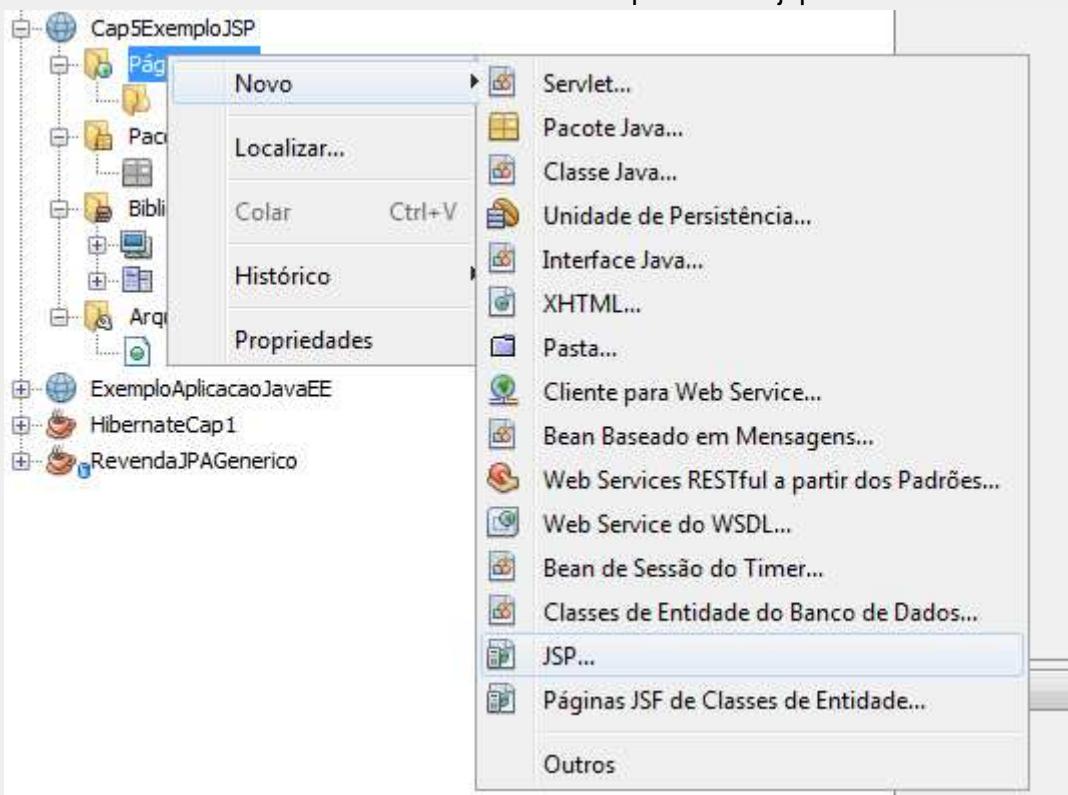
Devido ao procedimento citado, o primeiro acesso a página JSP é lento. A partir do segundo acesso, as páginas JSP são mais velozes.

PASSO-A-PASSO

1. No Netbeans deve ser criado um projeto Java web, o mesmo deve ficar com a estrutura mostrada a seguir.



2. O arquivo index.html deve ser removido e criado um arquivo index.jsp.



3. No arquivo criado, deve ser inserido o código a seguir.

```
<%--  
    Document : index  
    Created on : 20/12/2013, 11:10:29  
    Author    : Rosicléia Frasson  
--%>  
  
<%@page language="java" contentType="text/html" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
    <head>  
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Exemplo página JSP</title>
</head>
<body>
    <h1>Exemplo página JSP</h1>
    <p>Cálculo área retângulo</p>
    <form action="resultado.jsp" method="post">
        <table>
            <tr>
                <td>Base</td>
                <td><input type="text" name="base" /></td>
            </tr>
            <tr>
                <td>Altura</td>
                <td><input type="text" name="altura" /></td>
            </tr>
            <tr>
                <td colspan="2">
                    <input type="submit" value="Calcular" />
                    <input type="reset" value="Limpar" />
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

É importante perceber que o atributo action da tag form especifica a página em que os dados do formulário serão enviados e o atributo method, o método que será utilizado.

4. Para exibir a área do retângulo, é necessário criar uma nova página JSP. Esta deve ter o nome de resultado.jsp, pois na ação do formulário esta página está sendo referenciada. O código da mesma está ilustrado a seguir.

```
<%-->
Document      : resultado
Created on   : 20/12/2013, 11:33:30
Author       : Rosicleia Frasson
--%>

<%@page language="java" contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
```

```
8">
    <title>Exemplo página JSP</title>
</head>
<body>
<%
    double base =
Double.parseDouble(request.getParameter("base"));
    double altura =
Double.parseDouble(request.getParameter("altura"));
    double area = base * altura;
    double perimetro = (base + altura) * 2;
%>
<h1>Exemplo página JSP</h1>
<p>Cálculo área retângulo</p>
<table border="1">
    <tr>
        <td colspan="2"><b>Dados fornecidos pelo usuário</b></td>
    </tr>
    <tr>
        <td>Base</td>
        <td><%=base%></td>
    </tr>
    <tr>
        <td>Altura</td>
        <td><%=altura%></td>
    </tr>
    <tr>
        <td colspan="2">
            <b>Dados calculados pelo aplicativo</b>
        </td>
    </tr>
    <tr>
        <td>Área</td>
        <td><%=area%></td>
    </tr>
</table>
</html>
```

No scriptlet contido logo após o início do bloco body, primeiramente são obtidos os valores dos parâmetros - base e altura - enviados pelo usuário e convertidos para o tipo double. Em seguida, é efetuado o cálculo da área.

Os valores da base, altura e área são exibidos na página através de expressões.

5. Na execução do projeto deve ser exibida a seguinte página.

The screenshot shows a Firefox browser window with the title bar "Firefox". The address bar displays "localhost:8080/Cap5ExemploJSP/". The main content area contains the heading "Exemplo página JSP" and the text "Cálculo área retângulo". Below this, there are two input fields: "Base" with value "2" and "Altura" with value "3". Underneath the input fields are two buttons: "Calcular" and "Limpar".

6. Os valores de base e altura devem ser informados e ao clicar no botão calcular, deve ser exibida a página a seguir com o cálculo da área a partir dos dados informados.

The screenshot shows a Firefox browser window with the title bar "Firefox". The address bar displays "localhost:8080/Cap5ExemploJSP/resultado.jsp". The main content area contains the heading "Exemplo página JSP" and the text "Cálculo área retângulo". Below this, there are two tables: "Dados fornecidos pelo usuário" and "Dados calculados pelo aplicativo". The first table has rows for "Base" (2.0) and "Altura" (3.0). The second table has a single row for "Área" (6.0).

Dados fornecidos pelo usuário	
Base	2.0
Altura	3.0

Dados calculados pelo aplicativo	
Área	6.0

Java Beans

```

public class Pessoa {

    private String nome;
    private String cpf;
    private int idade;          Propriedades

    public Pessoa() {           Construtor default
    }

    public String getNome() { return nome; }

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

    public String getCpf() { return cpf; }

    public void setCpf(String cpf) { this.cpf = cpf; }

    public int getIdade() { return idade; }

    public void setIdade(int idade) { this.idade = idade; }          Métodos
                                                                    de
                                                                    acesso

}

```

Um bean é uma classe Java que segue uma série de convenções:

- Propriedades: Também chamadas de atributos, são responsáveis por armazenar o estado do bean. Devem possuir modificador de acesso private.
- Um construtor sem argumentos: A classe pode possuir diversos construtores, porém, deve ter o construtor padrão. Nos casos em que apenas o construtor default é necessário, o mesmo pode estar implícito.
- Atributos privados e métodos de acesso get e set: Para cada propriedade é necessária a existência dos métodos de acesso.

PASSO-A-PASSO

Para exemplificar o uso do bean, o exemplo apresentado anteriormente será modificado.

1. Deve ser criado um novo projeto java web no Netbeans.
2. O arquivo index.html deve ser removido e criado um arquivo index.jsp com o código a seguir.

```

<%-
    Document : index
    Created on : 20/12/2013, 11:10:29
    Author    : Rosicleia Frasson
--%>

```

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemplo página JSP</title>
    </head>
    <body>
        <h1>Exemplo página JSP</h1>
        <p>Cálculo área retângulo</p>
        <form action="ServletController" method="post">
            <table>
                <tr>
                    <td>Base</td>
                    <td><input type="text" name="base" /></td>
                </tr>
                <tr>
                    <td>Altura</td>
                    <td><input type="text" name="altura" /></td>
                </tr>
                <tr>
                    <td colspan="2">
                        <input type="submit" value="Calcular" />
                        <input type="reset" value="Limpar" />
                    </td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

3. Também é necessária a criação de um bean chamado Retangulo, onde serão armazenados os atributos e os respectivos métodos de acesso. Esta classe deve ficar dentro do pacote modelo. O código da mesma está descrito a seguir.

```
package br.com.rosicleiafrasson.cap7exemplojavabean.modelo;

public class Retangulo {

    private double base;
    private double altura;
    private double area;
```

```
public double getBase() {
    return base;
}

public void setBase(double base) {
    this.base = base;
}

public double getAltura() {
    return altura;
}

public void setAltura(double altura) {
    this.altura = altura;
}

public double getArea() {
    return base * altura;
}

}
```

4. Também é necessário a criação de uma servlet que captura os dados da página index e envia para o bean.

```
package br.com.rosicleiafrasson.cap7exemplojavabean.controller;

import br.com.rosicleiafrasson.cap5exemplojavabean.modelo.Retangulo;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "ServletController", urlPatterns = {"/ServletController"})
public class ServletController extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
```

```
Retangulo r = new Retangulo();
r.setBase(Double.parseDouble(request.getParameter("base")));
r.setAltura(Double.parseDouble(request.getParameter("altura")));
request.setAttribute("retanguloBean", r);
RequestDispatcher rd = request.getRequestDispatcher("resultado.jsp");
rd.forward(request, response);
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}
}
```

5. Por fim, a criação da página resultado.jsp que exibe os dados calculados pelo aplicativo.

```
<%--
    Document : index
    Created on : 20/12/2013, 11:10:29
    Author : Rosicleia Frasson
--%>

<%@page import="br.com.rosicleiafrasson.cap7exemplojavabean.modelo.Retangulo"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Exemplo página JSP</title>
    </head>
    <body>
        <%
            Retangulo ret = (Retangulo) request.getAttribute("retanguloBean");
        %>
        <h1>Cálculo área retângulo</h1>
        <table border="1">
```

```
<tr>
    <td colspan="2"><b>Dados fornecidos pelo usuário</b></td>
</tr>
<tr>
    <td>Base</td>
    <td><%=ret.getBase()%></td>
</tr>
<tr>
    <td>Altura</td>
    <td><%=ret.getAltura()%></td>
</tr>
<tr>
    <td colspan="2"><b>Dados calculados pelo aplicativo</b></td>
</tr>
<tr>
    <td>Área</td>
    <td><%=ret.getArea()%></td>
</tr>

</table>

</body>
</html>
```

É importante perceber que a utilização de páginas jsp, juntamente com os java beans e as servlets, separam a lógica de negócio da visualização. Como este é um aplicativo que contém poucos requisitos de negócio não é perceptível a vantagem desta separação. Em aplicações complexas, seria muito difícil misturar códigos jsp com a lógica de negócio.

Frameworks MVC

É notória a dificuldade de manutenção quando códigos HTML ficam misturados com código Java.

Pensando nisso, a comunidade de desenvolvedores começou a desenvolver ferramentas que auxiliem a separação das responsabilidades na aplicação. Essas ferramentas são conhecidas como frameworks MVC ou controladores MVC.

O MVC separa a codificação do software em três partes:

- **Modelo:** Parte do código onde estão representados os objetos de negócio. São esses objetos que mantém o estado da aplicação e fornecem ao controlador. O modelo é responsável pelos dados, regras de acesso e modificação dos dados. Essa parte do código é responsável também por se comunicar com a base de dados e outros sistemas existentes.
- **Visualização:** Responsável pela interface do usuário. É na camada de visualização que é definida a forma como os dados são apresentados e as ações dos usuários são encaminhadas para o controlador.
- **Controle:** Responsável por efetuar a ligação entre o modelo e a visualização. É no controle que são interpretadas as solicitações dos usuários, acionando as opções corretas a serem realizadas no sistema encaminhando as mesmas ao modelo e retornando as visualizações das solicitações correspondentes.

A utilização do MVC provê algumas vantagens no processo de desenvolvimento. Entre elas pode ser citada o reaproveitamento dos componentes do modelo. Ao utilizar o MVC, é possível utilizar os objetos de negócio definidos na camada modelo em diversas classes/arquivos de visualização. Além disso, com o aumento da complexidade do software, a manutenção do mesmo torna-se um processo mais difícil. A separação das responsabilidades em camadas pré-estabelecidas deixa a manutenção do código mais compreensível.

Diante da aceitação do MVC pela comunidade de programadores Java, surgiram alguns frameworks baseados no mesmo. Seguem os mais conhecidos.

Struts

The screenshot shows the Apache Struts website. At the top, there is a navigation bar with links for "Apache Struts", "Support", "Documentation", and "Contributing". Below the navigation is the Apache Software Foundation logo. The main header is "Struts™". A sub-header below it says "Last Published: 2014-01-15". The main content area has a title "Apache Struts" and a brief description: "Apache Struts is a free, open-source, MVC framework for creating elegant, modern Java web applications. It favors convention over configuration, is extensible using a plugin architecture, and ships with plugins to support REST, AJAX and JSON." There are two buttons at the bottom of this section: "Download" and "Technology Primer".

Struts 2.3.16 GA

Apache Struts 2.3.16 GA has been released on 8 december 2013. [Version notes](#)

Want to help?

We welcome your help! If you want to learn more about how to build and patch the Struts 2 codebase, please read the [Developer Docs](#).

Apache Struts 1

End-Of-Life (EOL)

The Struts 1.x web framework has reached its end of life and is no longer officially supported.

O Struts é um framework gratuito, de código aberto, disponibilizado pela Apache Software Foundation <<http://struts.apache.org/>>. Este framework foi desenvolvido para facilitar o desenvolvimento de aplicações Java EE utilizando o MVC.

Com o uso do Struts, através do ActionForm, os dados de um formulário HTML são recuperados mais facilmente. Além disso, o framework conta com o Action, que é um frontController que delega o processamento da requisição para um componente específico. O Struts conta ainda com uma biblioteca de tags para a construção das páginas JSP e um arquivo de configurações XML, que indicam as Actions responsáveis para o atendimento de determinada requisição.

VRaptor

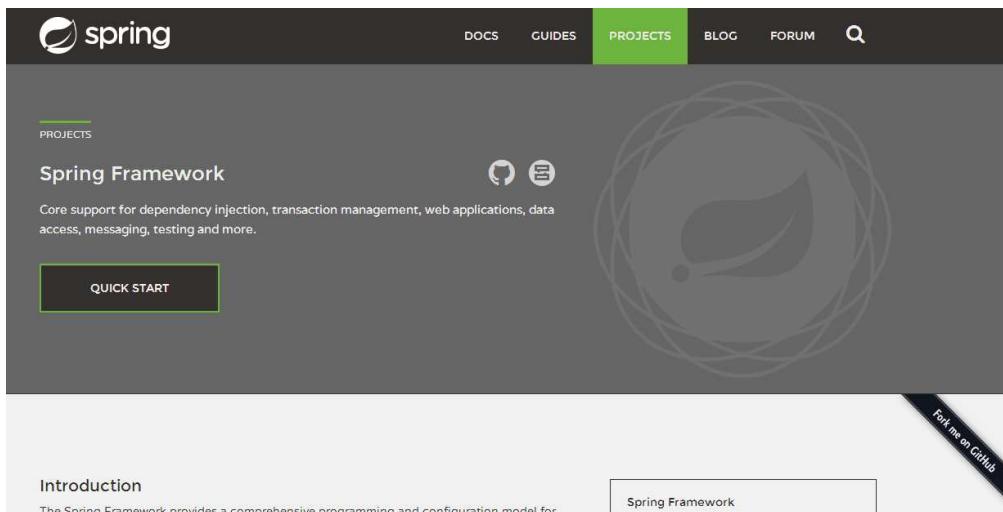
O VRaptor é um framework desenvolvido em território nacional. Foi criado na USP por Guilherme Silveira e atualmente é mantido pela Caelum. É gratuito, open source e possui documentação em português.



Uma das grandes vantagens do framework, é que o mesmo encapsula as principais classes de Servlets como HttpServletRequest, HttpServletResponse e Session. Por encapsular a lógica dos Servlets, garante alta produtividade e baixa curva de aprendizado.

Spring

É um framework open source para gerenciar componentes e um conjunto de serviços para interfaces de usuários, transações e persistência. Além do módulo core, que representa o cerne do framework, existe o Spring Data para persistência e o Spring Security para segurança da aplicação.



JSF

O Java Server Faces (JSF) foi desenvolvido pelo comitê JCP - Java Community Process - e está especificado na JSR 314 (Java Specification Request). Por ser uma especificação, existem diversos fornecedores que desenvolvem a implementação.

O JSF provê um conjunto de componentes baseados em eventos para a construção da interface com o usuário. Esses componentes possuem um modelo para manipulação de eventos. Além disso, o JSF possui suporte de diversos fabricantes que fornecem conjuntos de componentes prontos para utilização. Além disso, o JSF inclui APIs para validação das entradas dos usuários, definição e controle de navegação das páginas e suporte a internacionalização.

O JSF também permite a integração com diversos frameworks como os já citados Spring e Struts. Além da integração com frameworks de mapeamento objeto-relacional como Hibernate, EclipseLink.

JSF

A tecnologia Java Server Faces (JSF) foi desenvolvida com o intuito de deixar o desenvolvimento de aplicativos JEE com aparência, facilidade e produtividade de sistemas desktop. Ela é composta por um modelo de componentes para interfaces do usuário, um modelo de programação orientada a eventos, validação do lado servidor e conversão de dados.

O JSF é uma especificação lançada juntamente com JEE 6 e pode ser consultada em: <http://jcp.org/en/jsr/detail?id=314>. Sendo uma especificação, existem diversas implementações da mesma. A implementação referencial é a Mojarra da Oracle. Existe também uma outra implementação bastante conhecida que é a MyFaces da Apache.

O JSF é baseado no MVC e a separação das camadas fica muito clara para o desenvolvedor. A camada de controle nesta tecnologia é basicamente formada por uma servlet chamada de FacesServlet, por arquivos de configuração e por manipuladores de ação e observadores de eventos. O FacesServlet possui como responsabilidade o recebimento de requisições dos usuários, redirecionando as mesmas para o modelo e retornando a resposta devida. Nos arquivos de configuração estão contidas as regras de navegação. Os manipuladores de eventos recebem dados da camada de visualização, acessam o modelo e retornam o resultado através do FacesServlet.

A camada de visualização do JSF, é composta por páginas com extensão xhtml. Essas páginas são construídas utilizando um conjunto de componentes, que incluem caixas de texto, botões, formulários, tabelas, calendários, entre outros, que são adicionados, renderizados e exibidos em formato html. Deste modo, o desenvolvedor que utiliza tal tecnologia não possui a necessidade de escrever código HTML.

O uso do JSF para o desenvolvimento de projetos possui inúmeras vantagens. Entre elas podem ser citadas:

- componentes customizados: além dos componentes básicos fornecidos pelo framework, podem ser utilizadas bibliotecas de componentes de terceiros e/ou a fabricação de novos componentes.
- conversão de dados: os dados digitados pelo usuário podem facilmente ser convertidos para tipos específicos como datas, números, entre outros.
- validação: o JSF facilita o processo de validações básicas como campos requeridos, formatos de cpf, cnpj, entre outros.
- manipulação de erros: a manipulação de erros, bem como customização de mensagens de erro apropriadas são facilitadas com o uso do JSF.
- suporte a internacionalização: o JSF suporta aplicações multiidiomas com o recurso i18n.

Estrutura básica de uma página JSF

O conteúdo de uma página JSF é definido no corpo da tag <html>. Esse conteúdo é dividido em duas partes: o cabeçalho, delimitado pela tag <h:head> (e não pela tag <head>), e o corpo, delimitado pela tag <h:body> (e não pela tag <body>).

As bibliotecas de tags que serão utilizadas para construir a página devem ser importadas através do pseudo-atributo xmlns aplicado à tag <html>.

```
<%xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

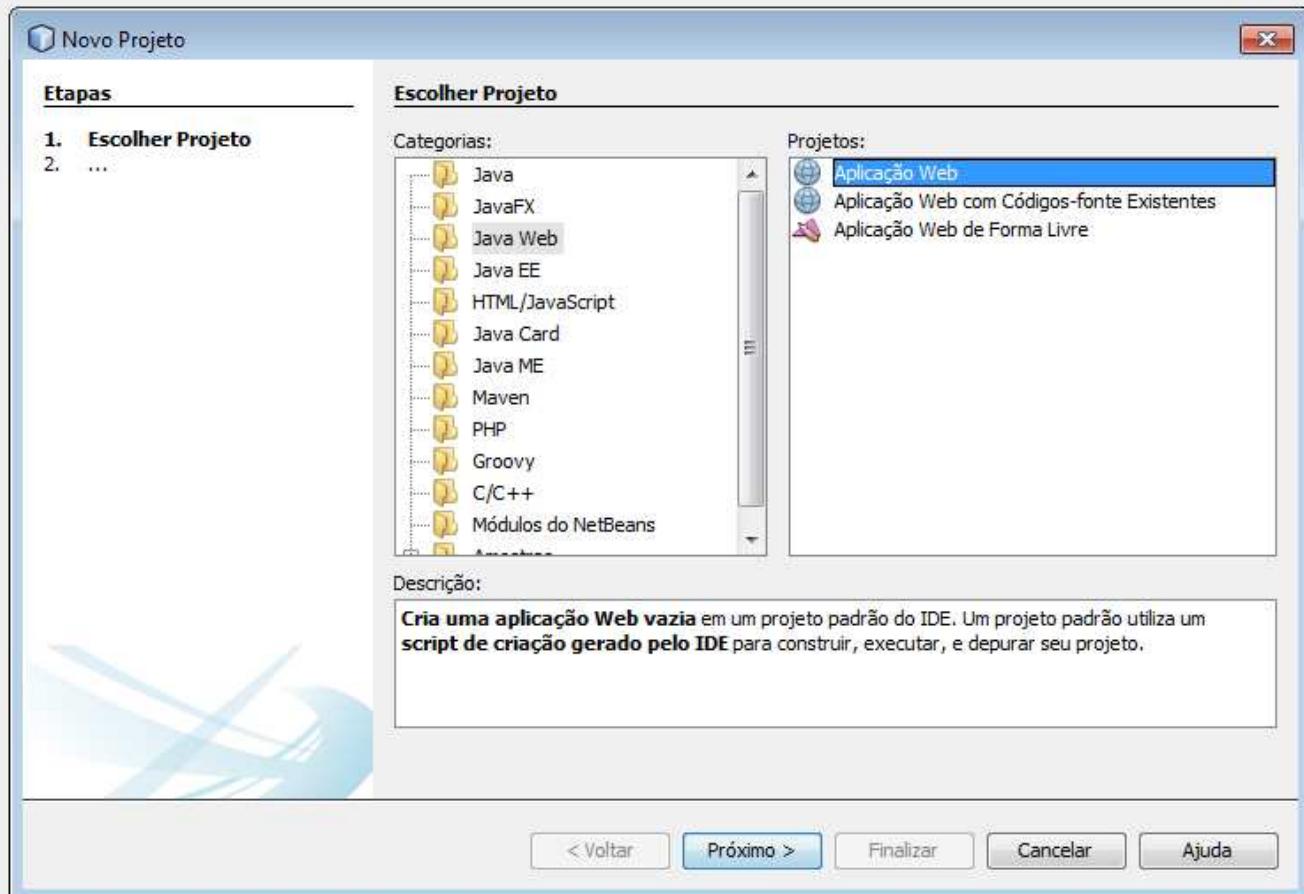
```
xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>

</h:head>
<h:body>
    Hello JSF!
</h:body>
</html>
```

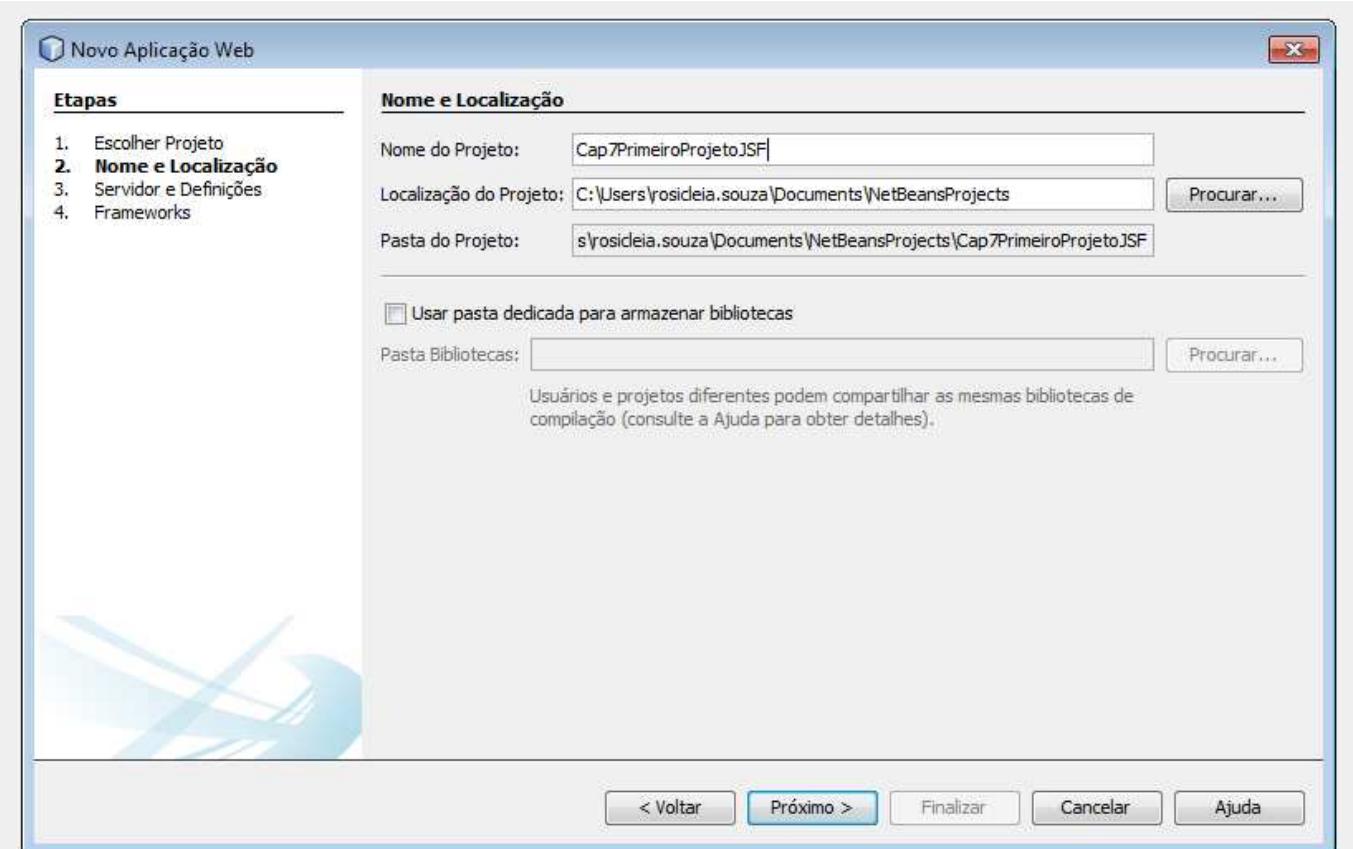
PASSO-A-PASSO

Primeiro Projeto JSF

1. Para criar um projeto JSF no Netbeans é necessário escolher a categoria Java Web e em Projetos Aplicação Web.



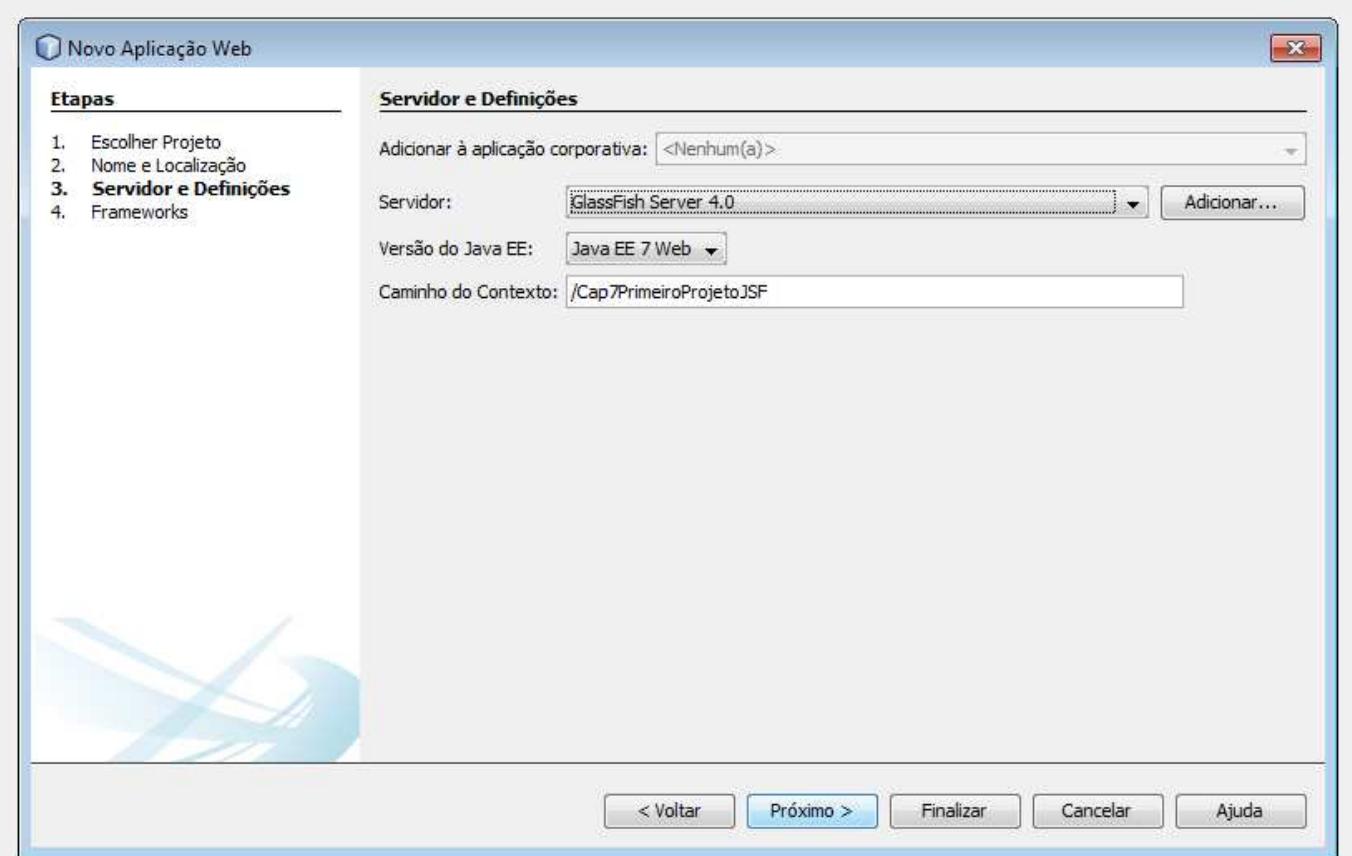
2. Na Guia Nome e Localização, é necessário dar um nome ao projeto e apontar onde o mesmo será salvo.



3. Na Guia Servidor e Definições será escolhido como servidor o GlassFish Server 4.0. A escolha por este servidor se deu em função do mesmo ser instalado junto com o Netbeans. Outros servidores podem ser utilizados, porém, eles devem ser instalados.

A versão do Java utilizada é a versão 7.

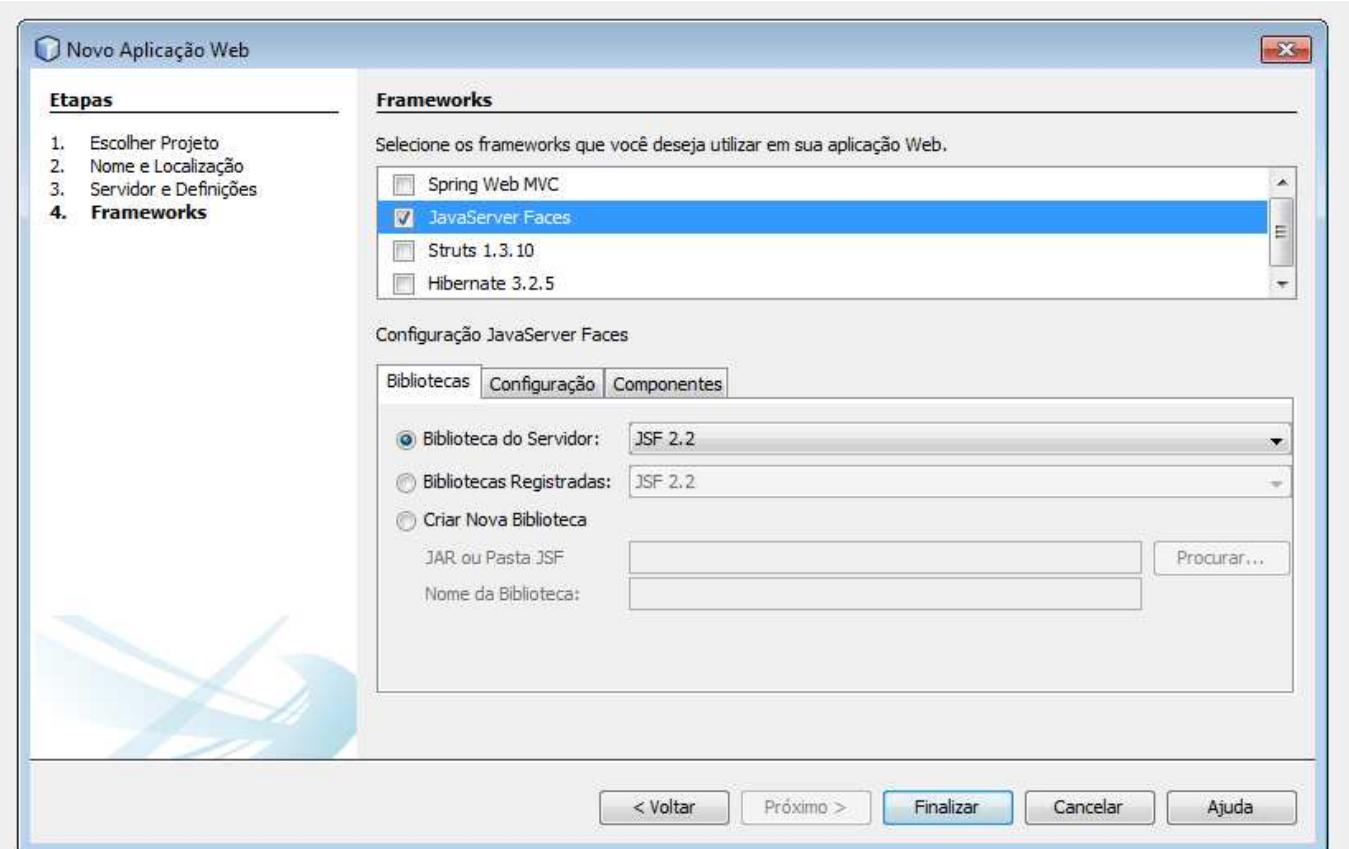
O caminho do contexto é o caminho que será utilizado para acessar a aplicação. Neste exemplo, o caminho do contexto é mesmo nome da aplicação.



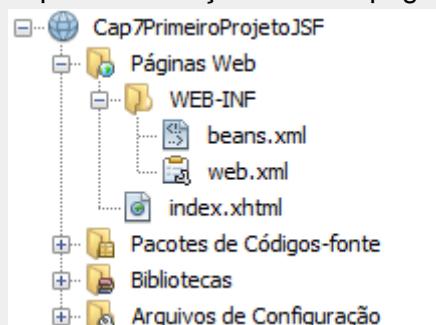
4. Na Guia Frameworks, é necessário sinalizar a IDE que será utilizado o framework JSF. Em bibliotecas, deve ser selecionada a versão do JSF que será utilizada.

Em componentes, não será utilizada nenhuma biblioteca de componentes neste projeto.

Em configuração, o padrão de URL do Servlet JSF é /faces/*. É importante ressaltar que nos projetos JSF, existe uma Servlet que efetua o controle da aplicação. Como já mencionado esta servlet possui o nome de Faces Servlet. No arquivo web.xml, montado automaticamente pelo Netbeans é apontada a classe que implementa a Faces Servlet e o padrão de url associado a esta servlet. No exemplo apresentado, toda vez que o caminho da URL iniciar por /faces/, a Faces Servlet é invocada.



5. Deve ser criado um projeto com estrutura similar à figura a seguir. É importante perceber que o JSF utiliza arquivos com extensão .xhtml para construção de suas páginas.



6. Na estrutura de arquivos e pastas criadas existe um arquivo que merece grande atenção, é o web.xml que fica dentro da pasta WEB-INF. No mesmo constam alguns parâmetros importantes para o projeto. Um parâmetro que merece atenção no momento é o servlet. O elemento servlet-class declara javax.faces.webapp.FacesServlet fazendo com que a classe seja carregada automaticamente. A url pattern declarada é /faces/*, o que indica que a Servlet é invocada cada vez que o caminho da URL iniciar por /faces/. O elemento session-timeout indica que a sessão expira em 30 minutos. Por sua vez, o elemento welcome-file indica a página inicial da aplicação. Vale salientar que o nome do arquivo que corresponde a página inicial é index.xhtml e como configurado anteriormente a url pattern faces antecede o nome do arquivo.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
<context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</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>/faces/*</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
<welcome-file-list>
    <welcome-file>faces/index.xhtml</welcome-file>
</welcome-file-list>
</web-app>

```

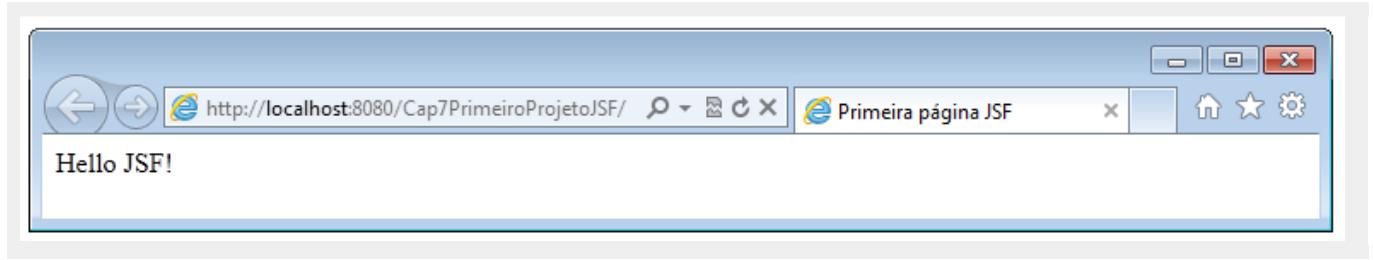
7. A página index.xhtml deve ser modificada e conter o código a seguir. É importante perceber a declaração da biblioteca html do JSF com o prefixo h: xmlns:h="<http://java.sun.com/jsf/html>". Essa declaração permite que os componentes do JSF sejam utilizados. No exemplo apresentado são utilizados os componentes h:head, que indica o cabeçalho da página e h:body que indica o corpo da página.

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Primeira página JSF</title>
    </h:head>
    <h:body>
        Hello JSF!
    </h:body>
</html>

```

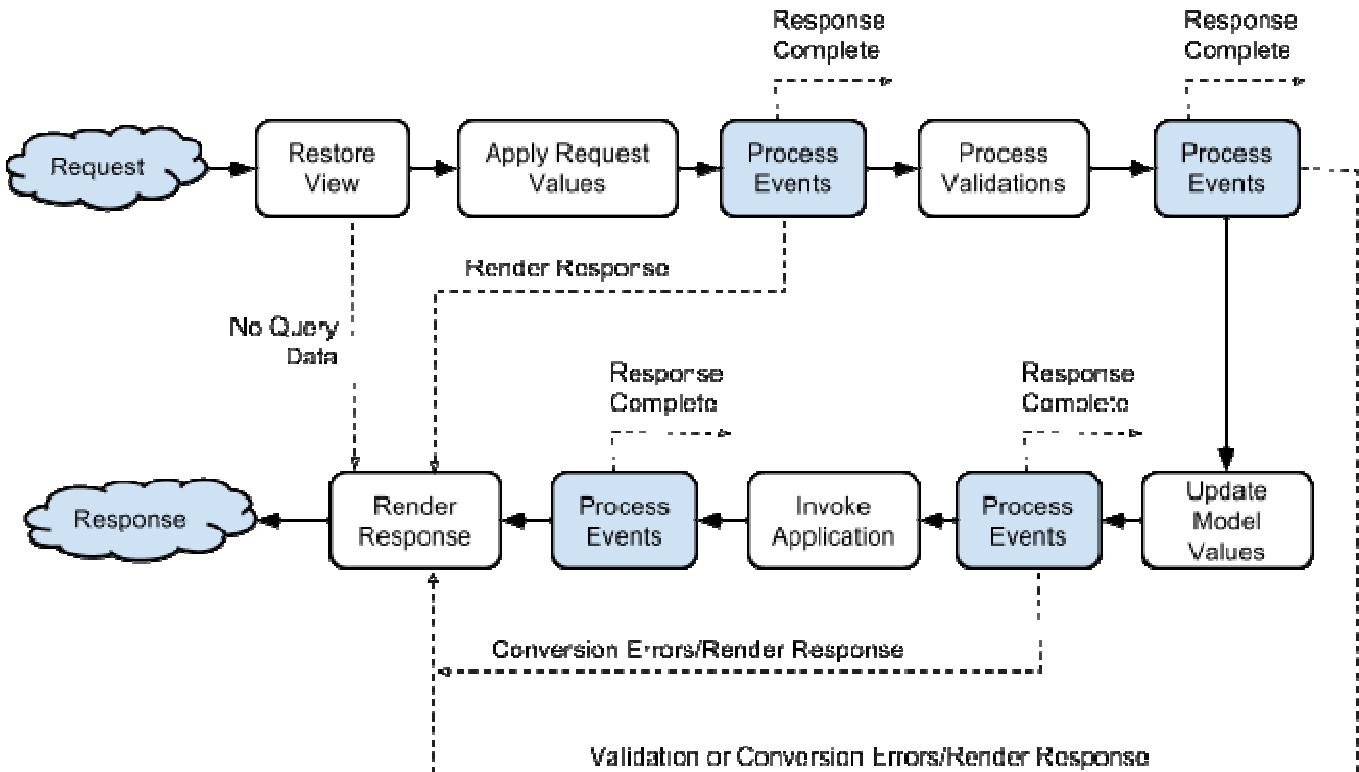
8. Executando a aplicação, deve ser exibida a seguinte página.



Ciclo de vida JSF

Pelo fato do JSF ser uma tecnologia baseada nas páginas JSP, seu ciclo de vida é semelhante. Do mesmo modo, que nas páginas JSP, em JSF, as requisições dos usuários são efetuadas utilizando o protocolo HTTP e as respostas são devolvidas como páginas HTML. Porém, no JSF, como existe a necessidade de conversão e validação dos dados antes do envio ao servidor, o ciclo de vida é um pouco mais complexo e dividido em várias fases:

- **Restore View:** Nesta fase se a página já foi exibida anteriormente, a árvore de componentes é recuperada. Caso seja a primeira exibição da página, a árvore de componentes é construída. O JSF mantém os dados do formulário. Isso significa que se a requisição é rejeitada pelo servidor, as entradas antigas são reexibidas para que o usuário possa corrigí-las. Caso não existam dados a serem submetidos, o request vai direto para a fase Render Response.
- **Apply Request Values:** Nesta fase, os dados contidos no formulário são recuperados e setados nos respectivos componentes.
- **Process Validation:** Esta fase transforma as strings dos componentes em valores locais que podem ser objetos de qualquer tipo. Na criação de uma página JSF podem ser anexados validadores que realizam verificações sobre os valores locais. Se no processo de validação não ocorrer nenhum erro, o ciclo de vida continua. Caso ocorra erro de validação ou conversão, o JSF vai para a fase Render Response, reexibindo a página para que o usuário possa corrigir os valores.
- **Update Model Values:** Nesta fase os valores locais são utilizados para atualizar as propriedades dos beans que estão ligados aos componentes.
- **Invoke Application:** Nesta fase ocorre a execução do componente acionado.
- **Render Response:** Nesta fase a resposta é codificada e enviada para o navegador. É importante ressaltar que se ocorrer algum erro de conversão ou validação na fase Process Validation, o ciclo de vida passa para esta fase para que o formulário seja reexibido e os dados possam ser alterados. No primeiro acesso a página, em que o JSF monta a árvore e exibe para o usuário também o ciclo de vida possui apenas a fase de Restore View e Render Response.



Managed Bean

Em aplicações que utilizam a tecnologia JSF, existe uma separação bem clara entre a camada de apresentação e a lógica de negócios. Para efetuar a comunicação entre as duas camadas citadas, são utilizados os managed beans. Como principais responsabilidades dos managed beans podem ser citados o fornecimento de dados que devem ser exibidos nas telas, recebimento de dados enviados nas requisições e a execução de tarefas de acordo com as ações dos usuários.

Os managed beans podem ser configurados através do arquivo faces-config ou através de anotações. As anotações são mais utilizadas devido a simplicidade das mesmas. A utilização da anotação @ManagedBean do pacote javax.faces.bean torna uma classe um managed bean.

```
import javax.faces.bean.ManagedBean;

@ManagedBean
public class Hello { }
```

Utilizando esta anotação, o JSF aplica como nome do managed bean o mesmo nome da classe, porém com a primeira letra minúscula. Esse padrão pode ser alterado utilizando o atributo name da anotação.

```
@ManagedBean (name = "outroNome")
public class Hello { }
```

Para que um managed bean disponibilize dados para as telas é necessário a criação dos

métodos gets.

```
private String saudacao = "Primeira página JSF 2.0";

public String getSaudacao(){
    return saudacao;
}
```

Para o recebimento de dados pelos managed bean é necessário a criação dos métodos sets.

```
public void setSaudacao(String saudacao) {
    this.saudacao = saudacao;
}
```

Os managed beans também podem conter métodos para o tratamento de ações do usuário, como o cadastro de um objeto em uma base de dados por exemplo.

Expression Language

Expression Language - EL - é a linguagem utilizada para que os managed beans possam ser referenciados nas páginas xhtml. Nos arquivos xhtml, a EL é delimitada através dos símbolos #{} .

Os managed beans são acessados pelas telas através do seu nome. As propriedades do mesmo também podem ser acessadas.

```
<h:outputText value="#{hello.saudacao}">
```

Embora não seja recomendado, a EL aceita algumas operações. Seguem os operadores aceitos pela EL:

- aritméticos : +, -, *, / e %. Para os dois últimos também podem ser utilizadas as variantes alfabéticas DIV e MOD.
- relacionais: <, <=, >, >=, == e != e suas respectivas variantes alfabéticas lt, le, gt, ge, ne e not.
- lógicos: &&, || e ! e suas variantes alfabéticas and, or e not.
- operador empty: verifica se uma variável é nula ou vazia ou uma colação ou array possui tamanho igual a zero.

PASSO-A-PASSO

Utilização do Managed Bean e Expression Language

1. Deve ser criado um projeto web no Netbeans como mostrado no exemplo anterior.
2. É necessária a criação de um pacote controller para armazenar o managed bean que deve ter o nome de Hello. O managed bean é uma classe do Java. Segue o código da mesma.

```
package br.com.rosicleiafrasson.cap9managedbean;
import javax.faces.bean.ManagedBean;
```

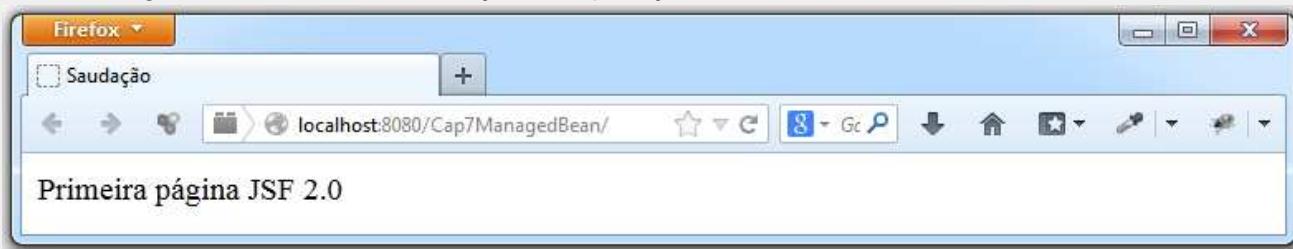
```
@ManagedBean  
public class Hello {  
  
    private String saudacao = "Primeira página JSF 2.0";  
  
    public String getSaudacao(){  
        return saudacao;  
    }  
}
```

3. Também deve ser alterado o código do arquivo index.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html">  
  <h:head>  
    <title>Saudação</title>  
  </h:head>  
  <h:body>  
    <h:form>  
      <h:outputText value="#{hello.saudacao}"/>  
    </h:form>  
  </h:body>  
</html>
```

É importante perceber a utilização da expression language na tag outputText. O nome do managed bean é o mesmo nome da classe, porém inicia com a letra minúscula. Vale ressaltar também que a variável saudação é acessada diretamente. Porém, é necessário o método de acesso. Se o mesmo não estiver implementado, o valor não é acessado e será gerado um erro durante a execução da aplicação.

A seguir, o resultado da execução da aplicação.



É importante perceber que os componentes JSF são transformados em HTML. Segue o código fonte da página apresentada.

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

    <body>
        <form id="j_idt5" name="j_idt5" method="post"
action="/Cap7ManagedBean/faces/index.xhtml;jsessionid=55db7da934971a30a221d6888434"
enctype="application/x-www-form-urlencoded">
            <input type="hidden" name="j_idt5" value="j_idt5" />
            Primeira página JSF 2.0<input type="hidden" name="javax.faces.ViewState"
id="j_id1:javax.faces.ViewState:o" value="6832589163449025133:5102407919025672837"
autocomplete="off" />
        </form>
    </body>
</html>
```

Componentes JSF

No JSF 2.0 as telas são definidas por meio arquivos .xhtml. A especificação do JSF possui um conjunto de componentes que podem ser utilizados na construção dessas telas. Esses componentes estão divididos duas bibliotecas:

- HTML (<http://java.sun.com/jsf/html>): A biblioteca HTML possui os componentes visuais que geram o HTML da página.
- Core (<http://java.sun.com/jsf/core>): A biblioteca core possui componentes não visuais, como validadores.

Com a tecnologia JSF, os componentes são renderizados em tempo de execução, o que permite que um mesmo componente seja renderizado de acordo com o tipo de cliente que está solicitando, ou seja, uma mesma página será renderizada de formas diferentes se for acessado por um computador e por um celular.

Para a utilização dos componentes é necessário que as bibliotecas estejam importadas nos arquivos xhtml, através do pseudo-atributo xmlns. Segue um exemplo de como devem ser importadas as duas bibliotecas mencionadas.

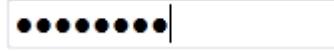
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
```

É importante perceber que foram declarados prefixos para as duas bibliotecas. Para a biblioteca core foi declarado o prefixo f e para a biblioteca html foi declarado o prefixo h. Podem ser declarados qualquer prefixo porém os mencionados são considerados padrão e usados pelas maioria dos desenvolvedores.

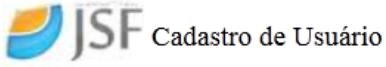
Biblioteca HTML

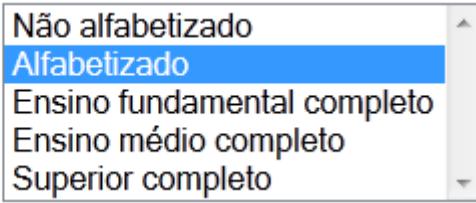
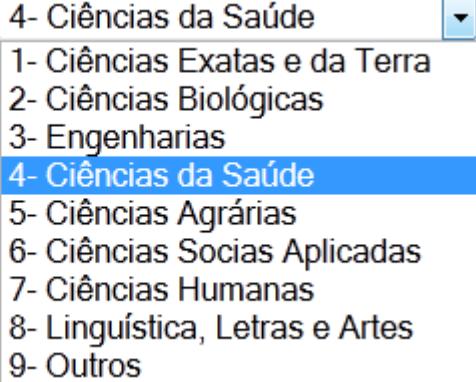
Elementos HTML

Tag	Descrição
form	<p>Formulário HTML. Embora nos formulários HTML é necessário indicar os atributos method e action, em JSF estes valores não são definidos pelo desenvolvedor, pois, o method é sempre post e a ação igual ao endereço da mesma página.</p> <pre><?xml version='1.0' encoding='UTF-8' ?> <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"> <html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html"> <h:head></pre>

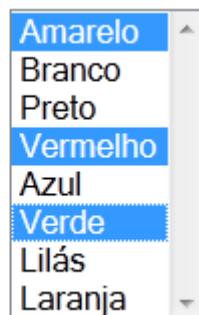
	<pre> <title>Exemplo formulário</title> </h:head> <h:body> <h:form> <!--Aqui devem ser colocados os campos do formulário--> </h:form> </h:body> </html></pre>
inputText	Campo de entrada de dados com apenas uma linha. Pode ser vinculado com um atributo de um managed bean. <pre><h:inputText value="#{usuarioBean.usuario.nome}" id="nome"/></pre> 
inputTextArea	Campo de entrada de dados com múltiplas linhas. <pre><h:inputTextarea value="#{usuarioBean.usuario.observacoes}" id="observacoes"/></pre> 
inputSecret	Campo de entrada de dados do tipo senha. Os caracteres digitados ficam ocultos. <pre><h:inputSecret value="#{usuarioBean.usuario.senha}" id="senha" /></pre> 
inputHidden	Campo oculto. Este campo pode ser utilizado para inserir informações que devem ser enviadas automaticamente quando um formulário é enviado ao servidor. <pre><h:inputHidden/></pre>
outputLabel	Renderiza o elemento label da HTML. Normalmente é utilizado para rotular campos de formulário. O atributo for vincula este componente com outro. <pre><h:outputLabel value="Login: " for="login" /></pre> 
outputLink	Renderiza links externos na página.

	<pre><h:outputLink value="http://www.google.com"> <h:outputText value="Para mais informações clique aqui"/> </h:outputLink></pre> <p style="text-align: center;">Para mais informações clique aqui</p>
outputFormat	<p>Permite a adição de texto parametrizado, ou seja, o texto possui espaços reservados para serem substituídos por valores reais durante o processamento.</p> <pre><h:outputFormat value="Olá {0}"> <f:param value="#{loginBean.usuario}" /> </h:outputFormat></pre> <p style="text-align: center;">Olá Maria</p>
outputText	<p>Renderiza textos simples na tela.</p> <pre><h:outputText value="Cadastro de Usuário"/></pre> <p style="text-align: center;">Cadastro de Usuário</p>
commandButton	<p>Este componente envia os dados de um formulário HTML para um servidor através do método post. Pode ser associado a métodos de ação de um managed bean através dos atributos action e actionListener.</p> <pre><h:commandButton value="Cadastrar" action="#{usuarioBean.adicionaUsuario}"/></pre> <p style="text-align: center;">Cadastrar</p>
commandLink	<p>Componente com função similar ao commandButton. Apenas com aparência de um link.</p> <pre><h:commandLink value="Cadastrar" action="#{usuarioBean.adicionaUsuario}"/></pre> <p style="text-align: center;">Cadastrar</p>
message	<p>Exibe uma mensagem de um componente específico. Geralmente é utilizada em conjunto com validadores e conversores. Para utilizar essa tag, é necessário definir um id para o componente referente a mensagem e associar message a este id através do atributo for.</p> <pre><h:outputLabel value="Idade: " for="idade" /> <h:panelGroup> <h:inputText value="#{usuarioBean.usuario.idade}" id="idade" validatorMessage="Só aceitamos candidatos que possuam idade entre 18 e 80 anos."> <f:validateLongRange minimum="18" maximum="80"/></pre>

	<pre></h:inputText> <h:message for="idade"/> </h:panelGroup></pre> <p>Idade: <input type="text" value="90"/> Só aceitamos candidatos que possuam idade entre 18 e 80 anos.</p>
messages	<p>Este componente permite que os erros dos diversos componentes de um formulário sejam exibidos juntos em um local específico da tela.</p> <pre><h:messages/></pre>  <ul style="list-style-type: none"> • O login deve ser composto apenas por letras. • A senha deve possuir no mínimo 6 caracteres e no máximo 10 caracteres. • O valor do peso deve estar entre 45 e 250. • Só aceitamos candidatos que possuam idade entre 18 e 80 anos.
graphicImage	<p>Este componente adiciona uma imagem na tela. O endereço da imagem deve ser informado no atributo value e pode ser usado o caminho relativo da mesma.</p> <pre><h:graphicImage value="/imagens/jsf.jpg"/></pre> 
selectOneListBox	<p>Lista de seleção onde apenas um item pode ser selecionado. Os itens do componente podem ser estáticos, como mostrado no exemplo abaixo ou dinâmicos.</p> <pre><h:outputLabel value="Escolaridade: " for="escolaridade"/> <h:selectOneListbox id="escolaridade"> value="#{usuarioBean.usuario.escolaridade}" > <f:selectItem itemValue="1" itemLabel="Não alfabetizado"/> <f:selectItem itemValue="2" itemLabel="Alfabetizado"/> <f:selectItem itemValue="3" itemLabel="Ensino fundamental completo"/> <f:selectItem itemValue="4" itemLabel="Ensino médio completo"/> <f:selectItem itemValue="5" itemLabel="Superior completo"/> </h:selectOneListbox></pre>

	<p>Escolaridade:</p> 
selectOneMenu	<p>Componente conhecido como combo box. No uso deste componente apenas uma opção pode ser selecionada. Os itens do componente podem ser estáticos ou dinâmicos. No exemplo a seguir, os elementos estão sendo carregados através de uma lista que pode representar dados adivindos de um banco de dados.</p> <pre><h:outputLabel value="Área de Atuação: " for="areaAtuacao"/> <h:selectOneMenu id="areaAtuacao" value="#{usuarioBean.usuario.areaAtuacao}"> <f:selectItems value="#{usuarioBean.areas}" /> </h:selectOneMenu></pre>
	<p>Área de Atuação:</p> 
selectOneRadio	<p>Caixa de seleção. No uso deste componente, entre as várias opções, apenas uma pode ser selecionada. Este componente também pode ser populado com valores estáticos ou dinâmicos.</p> <pre><h:outputLabel value="Sexo: " for="sexo" /> <h:selectOneRadio value="#{usuarioBean.usuario.sexo}" id="sexo"> <f:selectItem itemLabel="Feminino" itemValue="F"/> <f:selectItem itemLabel="Masculino" itemValue="M"/> </h:selectOneRadio></pre> <p>Sexo: <input checked="" type="radio"/> Feminino <input type="radio"/> Masculino</p>
selectBooleanCheckBox	<p>Caixa de seleção. Permite que o usuário faça seleções do tipo sim ou não.</p> <pre><h:selectBooleanCheckbox id="aceito" value="#{usuarioBean.aceite}" /> <h:outputLabel value="Li e concordo com" /></pre>

	<pre style="background-color: #e0f2ff; padding: 5px;">os termos de uso" for="aceito"/></pre> <p><input type="checkbox"/> Li e concordo com os termos de uso</p>
selectManyCheckBox	<p>Caixa de seleção. No uso deste componente, várias opções podem ser marcadas.</p> <pre style="background-color: #e0f2ff; padding: 10px;"> <h:outputLabel value="Hobbies: " for="hobbies"/> <h:selectManyCheckbox id="hobbies" value="#{usuarioBean.usuario.hobbies}" layout="pageDirection"> <f:selectItem itemValue="Músicas"/> <f:selectItem itemValue="Filmes"/> <f:selectItem itemValue="Culinária"/> <f:selectItem itemValue="Artesanato"/> <f:selectItem itemValue="Decoração"/> <f:selectItem itemValue="Livros"/> <f:selectItem itemValue="Passeios turísticos"/> <f:selectItem itemValue="Prática de esportes"/> </h:selectManyCheckbox></pre> <p>Hobbies:</p> <ul style="list-style-type: none"> <input type="checkbox"/> Músicas <input checked="" type="checkbox"/> Filmes <input type="checkbox"/> Culinária <input checked="" type="checkbox"/> Artesanato <input type="checkbox"/> Decoração <input type="checkbox"/> Livros <input checked="" type="checkbox"/> Passeios turísticos <input type="checkbox"/> Prática de esportes
selectManyListBox	<p>Componente similar ao selectOneListBox, porém, neste componente vários itens podem ser selecionados com a ajuda do botão CTRL.</p> <pre style="background-color: #e0f2ff; padding: 10px;"> <h:outputLabel value="Escolaridade: " for="escolaridade"/> <h:selectOneListbox id="escolaridade" value="#{usuarioBean.usuario.escolaridade}" > <f:selectItem itemValue="1" itemLabel="Não alfabetizado"/> <f:selectItem itemValue="2" itemLabel="Alfabetizado"/> <f:selectItem itemValue="3" itemLabel="Ensino fundamental completo"/> <f:selectItem itemValue="4" itemLabel="Ensino médio completo"/> <f:selectItem itemValue="5" itemLabel="Superior completo"/> </h:selectOneListbox></pre>

	<p style="text-align: center;">Cores Prediletas:</p> 
selectManyMenu	<p>Componente similar ao selectOneMenu, com a diferença de que o mesmo permite que sejam selecionados vários itens de uma vez utilizando a tecla CTRL.</p> <pre><h:outputLabel value="Idiomas: " for="idiomas"/> <h:selectManyMenu id="idiomas" value="#{usuarioBean.usuario.idiomas}" style="height: 90px"> <f:selectItem itemValue="Inglês"/> <f:selectItem itemValue="Francês"/> <f:selectItem itemValue="Alemão"/> <f:selectItem itemValue="Espanhol"/> <f:selectItem itemValue="Mandarim"/> </h:selectManyMenu></pre>
	<p style="text-align: center;">Idiomas:</p> 

```

        <f:selectItem itemLabel="Feminino" itemValue="F"/>
        <f:selectItem itemLabel="Masculino" itemValue="M"/>
    </h:selectOneRadio>

</h:panelGrid>

```

Nome:	<input type="text"/>
Sexo:	<input checked="" type="radio"/> Feminino <input type="radio"/> Masculino
Login:	<input type="text"/>
Senha:	<input type="text"/>

panelGroup

Este componente é utilizado para organizar as telas. Com ele é possível agrupar vários componentes em apenas um nó, colocando em apenas uma célula do panelGrid, por exemplo.

```

<h:panelGroup>
    <h:outputLabel value="Login: " for="login" />
    <h:graphicImage value="/imagens/ajuda.png" alt="ajuda"/>
</h:panelGroup>

<h:panelGroup>
    <h:inputText value="#{usuarioBean.usuario.login}"
id="login" validatorMessage="O login deve ser composto apenas por
letras e deve possuir entre 6 e 18 caracteres.">
        <f:validateRegex pattern="[a-z]{6,18}" />
    </h:inputText>
    <h:message for="login"/>
</h:panelGroup>

<h:outputLabel value="Senha: " for="senha"/>
<h:panelGroup>
    <h:inputSecret value="#{usuarioBean.usuario.senha}"
id="senha" validatorMessage="A senha deve possuir no mínimo 6
caracteres e no máximo 10 caracteres.">
        <f:validateLength maximum="10" minimum="6" />
    </h:inputSecret>
    <h:message for="senha"/>
</h:panelGroup>

```

	<p>Login: </p> <p>Senha:</p>
dataTable	<p>Componente que gera uma tabela HTML. Pode ser vinculado a um managed bean para preenchimento de dados dinâmicos.</p> <p>No exemplo a seguir, é gerada uma tabela que representa uma agenda de contatos. Esses contatos poderiam estar armazenados em uma base de dados. A título de exemplo, os contatos estão sendo adicionados manualmente no managed bean.</p> <p>Segue o bean Contato, onde está definido que cada objeto deste tipo deve possuir um nome e um telefone. Também foi definido um construtor com estes dois atributos para facilitar a criação de um novo contato.</p> <pre>package br.com.rosicleiafrasson.cap10componentestabela.modelo; public class Contato { private String nome; private String telefone; public Contato(String nome, String telefone) { this.nome = nome; this.telefone = telefone; } public String getNome() { return nome; } public void setNome(String nome) { this.nome = nome; } public String getTelefone() { return telefone; } public void setTelefone(String telefone) { this.telefone = telefone; } }</pre>

A seguir, está ilustrado o managed bean ContatoBean que possui uma lista de contatos. No método de acesso a esta lista são adicionados dois contatos para que os mesmos populem a tabela.

```
package br.com.rosicleiafrasson.cap10componentestabela.controller;

import br.com.rosicleiafrasson.cap10componentestabela.modelo.Contato;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;

@ManagedBean
public class ContatoBean {

    private List<Contato> contatos = new ArrayList<>();

    public List<Contato> getContatos() {
        contatos.clear();
        contatos.add(new Contato("Maria", "(48) 9924-9078"));
        contatos.add(new Contato("João", "(11) 3645-6754"));
        return contatos;
    }
}
```

Na página, onde está o componente dataTable é definido através do atributo value os dados da tabela. O atributo var nomeia o elemento da iteração corrente para se ter acesso ao índice da mesma nas colunas.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>Exemplo Tabela</title>
    </h:head>
    <h:body>
        <h:form>
            <h:dataTable border="1"
value="#{contatoBean.contatos}" var="contato">
                <f:facet name="header">
                    <h:outputText value="Agenda de Contatos"/>
                </f:facet>
```

```

<h:column>
    <f:facet name="header">
        <h:outputText value="Nome"/>
    </f:facet>
    <h:outputText value="#{contato.nome}"/>
</h:column>

<h:column>
    <f:facet name="header">
        <h:outputText value="Telefone"/>
    </f:facet>
    <h:outputText
        value="#{contato.telefone}"/>
</h:column>
</h:dataTable>
</h:form>
</h:body>
</html>

```

Agenda de Contatos	
Nome	Telefone
Maria	(48) 9924-9078
João	(11) 3645-6754

column

Define uma coluna em uma tabela.

```

<h:column>
    <f:facet name="header">
        <h:outputText value="Telefone"/>
    </f:facet>
    <h:outputText value="#{contato.telefone}"/>
</h:column>

```

Telefone
(48) 9924-9078
(11) 3645-6754

Tags de atributos básicos HTML

Atributo	Descrição
id	Identifica o componente. Pode ser utilizado em todos os elementos HTML. <pre><h:inputText value="#{usuarioBean.usuario.nome}" id="nome"/></pre>
binding	Associa o componente da camada de visão à sua camada de controle. Pode ser utilizado em todos os elementos HTML. <pre><h:inputText binding="#{candidatoBean.candidato.cpf}" id="cpf"/></pre>
rendered	Renderiza ou não um componente. O valor false indica que o componente não deve ser renderizado. Pode ser utilizado em todos os elementos HTML. <pre><h:panelGrid columns="4" rendered="#{funcionarioBean.ehUsuario}"> <h:outputLabel for="login" value="Login: "/> <h:inputText id="login" value="#{funcionarioBean.funcionario.login}" /> <h:outputLabel for="senha" value="Senha: "/> <h:inputText id="senha" value="#{funcionarioBean.funcionario.senha}"/> </h:panelGrid></pre>
styleClass	Especifica uma classe CSS que contém definições de estilo. Pode ser utilizado em todos os elementos HTML. <pre><h:inputText value="#{candidatoBean.candidato.nome}" id="nome" styleClass="cxTexto" /></pre>
value	Associa o componente da camada de visão à sua camada de controle. Pode ser utilizado nos input, output e command. <pre><h:inputText value="#{candidatoBean.candidato.cpf}" id="cpf" /></pre>
valueChangeListener	Evento disparado quando o usuário muda o valor selecionado no controle. <pre><h:selectOneMenu valueChangeListener="#{testeBean.carregarCidades}"> <f:selectItems value="#{testeBean.estados}" /> <f:ajax event="change" render="cidades" /> </h:selectOneMenu> <h:selectOneMenu id="cidades"> <f:selectItems value="#{testeBean.cidades}" /> </h:selectOneMenu></pre>
converter	Permite a utilização de um converter próprio, ou seja, um converter específico do projeto.

	<pre><h:selectOneMenu value="#{cadastrarMoradorBean.morador.apartamento}" converter="entityConverter"> <f:selectItems value="#{cadastrarMoradorBean.apartamentos}" var="apartamento" itemLabel="#{apartamento.apartamento}" /> </h:selectOneMenu></pre>
validator	Permite a utilização de um validator próprio, ou seja, um validator específico do projeto.
required	Indica se um campo é obrigatório. Pode ser utilizado nos inputs. <pre><h:inputText value="#{candidatoBean.candidato.nome}" id="nome" required="true" /></pre>

Atributos HTML 4.0

Atributo	Descrição
accesskey	Cria uma tecla de atalho para dar foco em um elemento. Embora a documentação W3C sugere que a combinação de teclas para acionar o atributo accesskey seja Ctrl + Alt + tecla, a maioria dos navegadores utilizam Alt + tecla ou Shift + Alt + tecla. <pre><h:commandButton value="Cadastrar" action="#{candidatoBean.adicionarCandidato}" accesskey="C" /></pre>
acceptcharset	Indica o charset que deve ser utilizado. <pre><h:form acceptcharset="ISO-8859-1"></pre>
alt	Texto alternativo para elementos não textuais como imagens. <pre><h:graphicImage value="/imagens/estrela.png" alt="estrela"/></pre>
border	Valor em pixel para a largura da borda de um componente. <pre><h:panelGrid columns="2" border="3"> <h:outputLabel value="Nome: " /> <h:inputText /> </h:panelGrid></pre> 
dir	Define a direção do texto. Pode ser ltr - esquerda ou rtl - direita. <pre><h:inputText value="#{candidatoBean.candidato.nome}" id="nome" dir="rtl" /></pre> 
disabled	Desabilita um componente. No exemplo a seguir o input está desabilitado, ou seja, não permite que nenhum valor seja digitado no mesmo.

	<pre><h:inputText value="#{candidatoBean.candidato.nome}" id="nome" disabled="true" /></pre>
maxlength	<p>Define a quantidade máxima de caracteres de uma caixa de texto.</p> <pre><h:inputText value="#{candidatoBean.candidato.login}" id="login" maxlength="12" /></pre> 
readonly	<p>O valor do componente fica visível. Porém não pode ser alterado. Normalmente é utilizado em conjunto com teclados virtuais ou calendários, pois o valor do campo fica disponível para leitura, porém não pode ser alterado.</p>
rel	<p>Relação entre o documento atual e um link especificado com o atributo href.</p> <pre><link href="/css/estilo.css" type="text/css" rel="stylesheet"/></pre>
rows	<p>Número visível de linhas em uma área de texto.</p> <pre><h:inputTextarea value="#{usuarioBean.usuario.observacoes}" id="observacoes" rows="4"/></pre> 
size	<p>Define o tamanho de uma caixa de texto.</p> <pre><h:inputText value="#{candidatoBean.candidato.nome}" id="nome" dir="rtl" size="75" /></pre> 
style	<p>Aplica um estilo a um componente.</p> <pre><h:selectManyMenu id="idiomas" value="#{candidatoBean.candidato.idiomas}" style="height: 90px" ></pre>
tabindex	<p>Define a ordem em que um elemento recebe o foco usando a tecla TAB. O valor para este atributo deve ser um número inteiro entre 0 e 32767.</p>

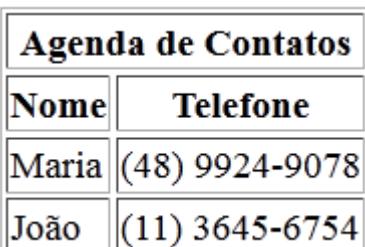
Atributos de eventos DHTML

Atributo	Descrição
onblur	Evento disparado quando o componente perde o foco.
onchange	Evento disparado quando o valor do campo é alterado.
onclick	Evento disparado quando o componente é clicado.

onfocus	Evento disparado quando o componente recebe foco.
onkeydown	Evento disparado assim que a tecla é pressionada.
onkeypress	Evento disparado assim que a tecla é pressionada. É disparado após o onkeydow.
onkeyup	Evento disparado quando a tecla pressionada é solta.
onmousedown	Evento disparado quando o botão do mouse é pressionado sobre o elemento.
onmousemove	Evento disparado quando o mouse se move sobre o elemento.
onmouseout	Evento disparado quando o mouse sair de cima do componente.
onmouseover	Evento disparado quando o mouse passar em cima do componente.
onmouseup	Evento disparado quando o botão do mouse é liberado.
onreset	Evento disparado quando o botão de reset do formulário é acionado.
onselect	Evento disparado quando o texto contido no componente é selecionado.
onsubmit	Evento disparado quando o formulário é submetido.

Biblioteca Core

Tag	Descrição
view	Serve para delimitar a região onde será criada a árvore de componentes do JSF. Essa tag só é necessária com o uso de JSP, no facelet não é necessário.
subview	Serve para evitar que os IDs dos componentes se repitam na árvore de componentes, nos casos em que existem includes de páginas dinâmicos.
facet	Adiciona uma faceta a um componente. No caso das tabelas, por exemplo, com esta tag é possível adicionar cabeçalho ou rodapé. <pre><h:dataTable border="1" value="#{contatoBean.contatos}" var="contato"> <f:facet name="header"> <h:outputText value="Agenda de Contatos"/> </f:facet> <h:column> <f:facet name="header"> <h:outputText value="Nome"/> </f:facet> <h:outputText value="#{contato.nome}"/> </h:column></pre>

	<pre> <h:column> <f:facet name="header"> <h:outputText value="Telefone"/> </f:facet> <h:outputText value="#{contato.telefone}"/> </h:column> </h:dataTable> </pre>
	
attribute	Cria um par de nome/valor que define o valor de um atributo nomeado associado com a tag que o contém.
param	Adiciona um parâmetro a um componente.
action	É um evento de ação disparado quando o componente clicado. O evento é executado na fase Invoke Application. O action contribui para a navegação das páginas, porém não possui informações sobre o evento. <pre><h:commandLink value="Cargo" action="/faces/paginas/cargo.xhtml"/></pre>
actionListener	Também é um evento de ação disparado quando o componente é clicado e executado na fase Invoke Application. Diferentemente do action, o actionListener possui informações sobre o evento, porém não contribui para a navegação de páginas.
valueChangeListener	Evento disparado quando o valor de um componente é alterado. Pode ser usado por exemplo, para carregar uma lista de cidades de acordo com o estado selecionado.
Conversores:	<p>Conversão é o processo que garante que os dados informados pelo usuário se transformem em um tipo específico. O processo de conversão ocorre na fase Apply Request Values. Os valores convertidos não são aplicados aos beans neste momento, eles apenas são convertidos e aplicados a objetos que representam os componentes e são chamados de valores locais.</p> <p>No JSF, a conversão para os tipos primitivos é feita de forma implícita. Os objetos do tipo BigInteger e BigDecimal também são convertidos implicitamente. É possível converter valores de entrada e de saída. Para os tipos que não possuem conversor padrão ou em casos em que o tipo possui, porém não é adequado, é possível definir conversores explícitos. Os conversores disparam mensagens que podem ser configuradas através do elemento converterMessage. Também é possível trabalhar com conversores explícitos nos casos em que os conversores implícitos não forem suficientes.</p>

convertDateTime	Permite a formatação de datas utilizando um pattern. Esse conversor pode ser aplicado em dados do tipo java.util.Date.
	<pre><h:outputLabel value="Data Nascimento: " for="dtNascimento"/> <h:panelGroup> <h:inputText value="#{candidatoBean.candidato.dataNascimento}" id="dtNascimento" converterMessage="Data no formato inválido. Utilize DD/MM/AAAA." > <f:convertDateTime pattern="dd/MM/yyyy" type="date"/> </h:inputText> <h:message for="dtNascimento" /> </h:panelGroup></pre> <p>Data Nascimento: <input type="text" value="144454"/> Data no formato inválido. Utilize DD/MM/AAAA.</p>
convertNumber	<p>Permite a formatação de um número de acordo com uma definição prévia. Seguem alguns atributos desta tag:</p> <ul style="list-style-type: none"> ● currencySymbol: Define um símbolo na formatação de moedas. <pre><h:outputLabel value="Último Salário: " for="ultimoSalario" /> <h:inputText value="#{candidatoBean.candidato.ultimoSalario}" id="ultimoSalario"> <f:convertNumber currencySymbol="R\$" type="currency"/> </h:inputText></pre> <p>Último salário: <input type="text" value="R\$ 0,00"/></p> <ul style="list-style-type: none"> ● maxFractionDigits: Define o número máximo de dígitos que serão formatados na parte fracionária do resultado. No fragmento de código a seguir, o atributo pode ter no máximo 3 dígitos na parte fracionária. Caso a parte fracionária tenha uma maior quantidade de dígitos do que foi definido, o número será arredondado. <pre><h:outputLabel value="Peso: " for="peso" /> <h:inputText value="#{candidatoBean.candidato.peso}" id="peso"> <f:convertNumber maxFractionDigits="3" /> </h:inputText></pre> <p>Peso: <input type="text" value="70,5678"/></p> <p>O campo acima será convertido para o resultado a seguir.</p> <p>Peso: <input type="text" value="70,568"/></p> <ul style="list-style-type: none"> ● maxIntegerDigits: Define a quantidade máxima de dígitos da parte inteira de um número. <pre><h:outputLabel value="Peso: " for="peso" /></pre>

```
<h:inputText value="#{candidatoBean.candidato.peso}" id="peso">
    <f:convertNumber maxIntegerDigits="3" />
</h:inputText>
```

- minFractionDigits: Define a quantidade mínima de dígitos da parte fracionária de um número.

```
<h:outputLabel value="Altura: " for="altura" />
<h:inputText value="#{candidatoBean.candidato.altura}" id="altura">
    <f:convertNumber minFractionDigits="2" />
</h:inputText>
```

- minIntegerDigits: Define a quantidade mínima de dígitos da parte inteira de um número.

```
<h:outputLabel value="Altura: " for="altura" />
<h:inputText value="#{candidatoBean.candidato.altura}" id="altura">
    <f:convertNumber minFractionDigits="2" minIntegerDigits="1" />
</h:inputText>
```

Altura:

- pattern: Define um padrão de formatação personalizado através de uma expressão regular.

```
<h:outputLabel value="Altura: " for="altura" />
<h:inputText value="#{candidatoBean.candidato.altura}" id="altura">
    <f:convertNumber pattern="#0.00" />
</h:inputText>
```

Altura:

- type: Este atributo especifica como a string deve ser formatada. Possui três valores válidos: number, currency e percentage. Number é o valor padrão, currency é o valor usado para moedas e percentage usado para percentuais.

```
<h:outputLabel value="Valor: " for="valor" />
<h:inputText value="#{candidatoBean.candidato.valor}" id="valor">
    <f:convertNumber type="percent" />
</h:inputText>
```

	<p>Valor: <input type="text" value="0%"/></p>
converter	É utilizado para a utilização de conversores criados pelo desenvolvedor, ou seja, conversores que não fazem parte da especificação.
Validadores	Durante o processo de construção de aplicativos, além de converter valores digitados pelo usuário em tipos específicos, é necessário verificar se os mesmos estão conforme algumas regras de negócio estabelecidas. A validação ocorre na fase Process Validation. Os validadores disparam mensagens que podem ser configuradas com o elemento validatorMessage.
validateDoubleRange	<p>Utilizado para verificar se um valor numérico real está entre um determinado intervalo de números.</p> <pre><h:panelGroup> <h:inputText value="#{usuarioBean.usuario.peso}" id="peso" validatorMessage="O valor do peso deve estar entre 45 e 250."> <f:convertNumber maxFractionDigits="3" maxIntegerDigits="3" /> <f:validateDoubleRange minimum="45.00" maximum="250.00"/> </h:inputText> <h:message for="peso" /> </h:panelGroup></pre> <p>Peso: <input type="text" value="500"/> O valor do peso deve estar entre 45 e 250.</p>
validateLength	<p>Verifica se uma string possui uma quantidade mínima ou máxima de letras.</p> <pre><h:outputLabel value="Senha: " for="senha"/> <h:panelGroup> <h:inputSecret value="#{usuarioBean.usuario.senha}" id="senha" validatorMessage="A senha deve possuir no mínimo 6 caracteres e no máximo 10 caracteres."> <f:validateLength maximum="10" minimum="6"/> </h:inputSecret> <h:message for="senha"/> </h:panelGroup></pre> <p>Senha: <input type="text"/> A senha deve possuir no mínimo 6 caracteres e no máximo 10 caracteres.</p>
validateLongRange	Verifica se um número inteiro está entre um determinado intervalo de números.

	<pre> <h:outputLabel value="Idade: " for="idade" /> <h:panelGroup> <h:inputText value="#{usuarioBean.usuario.idade}" id="idade" validatorMessage="Só aceitamos candidatos que possuam idade entre 18 e 80 anos."> <f:validateLongRange minimum="18" maximum="80"/> </h:inputText> <h:message for="idade"/> </h:panelGroup> </pre>
	<p>Idade: <input type="text" value="90"/> Só aceitamos candidatos que possuam idade entre 18 e 80 anos.</p>
validateRegex	<p>Verifica se um texto respeita determinada expressão regular.</p> <pre> <h:panelGroup> <h:outputLabel value="Login: " for="login" /> <h:graphicImage value="/imagens/ajuda.png" alt="ajuda"/> </h:panelGroup> <h:panelGroup> <h:inputText value="#{usuarioBean.usuario.login}" id="login" validatorMessage="O login deve ser composto apenas por letras e deve possuir entre 6 e 18 caracteres."> <f:validateRegex pattern="[a-z]{6,18}" /> </h:inputText> <h:message for="login"/> </h:panelGroup> </pre>
	<p>Login:  <input type="text" value="123"/> O login deve ser composto apenas por letras e deve possuir entre 6 e 18 caracteres.</p>
validator	<p>Este componente deve ser utilizado para validadores customizados.</p>
loadBundle	<p>Permite carregar um pacote de recursos do Locale da página atual e armazenar o mesmo em um mapa no escopo de request.</p>
selectItems	<p>Especifica itens para um componente de seleção. Utilizado para valores dinâmicos. No exemplo a seguir, os itens estão armazenados em uma lista e podem representar valores advindos de uma base de dados.</p> <pre> <h:outputLabel value="Área de Atuação: " for="areaAtuacao"/> <h:selectOneMenu id="areaAtuacao" value="#{usuarioBean.usuario.areaAtuacao}"> </pre>

```
<f:selectItems value="#{usuarioBean.areas}"/>
</h:selectOneMenu>
```

Área de Atuação:

The dropdown menu contains the following options:

- 1- Ciências Exatas e da Terra
- 2- Ciências Biológicas
- 3- Engenharias
- 4- Ciências da Saúde**
- 5- Ciências Agrárias
- 6- Ciências Sociais Aplicadas
- 7- Ciências Humanas
- 8- Linguística, Letras e Artes
- 9- Outros

selectItem

Especifica um item para um componente de seleção. Utilizado para valores estáticos.

```
<h:outputLabel value="Hobbies: " for="hobbies"/>
<h:selectManyCheckbox id="hobbies"
value="#{usuarioBean.usuario.hobbies}" layout="pageDirection">
    <f:selectItem itemValue="Músicas"/>
    <f:selectItem itemValue="Filmes"/>
    <f:selectItem itemValue="Culinária"/>
    <f:selectItem itemValue="Artesanato"/>
    <f:selectItem itemValue="Decoração"/>
    <f:selectItem itemValue="Livros"/>
    <f:selectItem itemValue="Passeios turísticos"/>
    <f:selectItem itemValue="Prática de esportes"/>
</h:selectManyCheckbox>
```

- Hobbies:
- Músicas
 - Filmes
 - Culinária
 - Artesanato
 - Decoração
 - Livros
 - Passeios turísticos
 - Prática de esportes

PASSO-A-PASSO

Cadastro de currículo

Para exemplificar o uso de componentes JSF, uma página para cadastro de currículos será montada.

1. Após a criação do projeto JSF, é necessário criar as classes de modelo. No pacote modelo é necessária a criação da classe Cargo que deve armazenar os possíveis cargos dos candidatos.

```
package br.com.rosicleiafrasson.curriculo.model;

public class Cargo {
    private int codigo;
    private String descricao;

    //Gets e sets
}
```

2. Ainda no pacote modelo, a classe Conhecimento tem por objeto armazenar os conhecimentos técnicos que os candidatos possuem.

```
package br.com.rosicleiafrasson.curriculo.model;

public class Conhecimento {

    private int codigo;
    private String nome;

    //Gets e sets
}
```

3. Para facilitar possíveis manutenções, o endereço do usuário também deve possuir uma classe.

```
package br.com.rosicleiafrasson.curriculo.model;

public class Endereco {
    private String endereco;
    private String complemento;
    private String municipio;
    private String bairro;
    private String estado;

    //Gets e sets
}
```

4. Para finalizar o pacote modelo, é necessária a classe candidato.

```
package br.com.rosicleiafrasson.curriculo.model;

import java.util.Date;
import java.util.List;

public class Candidato {

    private int codigo;
    private String nome;
    private String cpf;
    private Date dataNascimento;
    private char sexo;
    private char nacionalidade;
    private String raca;
    private String estadoCivil;
    private boolean ehDeficiente;
    private String telefone;
    private String email;
    private Endereco endereco;
    private String login;
    private String senha;
    private List<String> formacoes;
    private List<String> idiomas;
    private List<Conhecimento> conhecimentos;
    private List<Cargo> cargosPretendidos;
    private String experiencias;
    private double ultimoSalario;
    private double pretensaoSalarial;
    private int cargaHoraria;

    public Candidato() {
        endereco = new Endereco();
    }

    //Gets e sets
}
```

5. O pacote controller é composto pela classe CandidatoBean que tem como responsabilidade efetuar a ligação entre o modelo e a camada de visualização composta pelas páginas xhtml.

É importante perceber que os cargos e conhecimentos estão sendo adicionados de maneira estática. Porém os mesmos poderiam estar armazenados em uma base de dados.

```
package br.com.rosicleiafrasson.curriculo.controller;
```

```
import br.com.rosicleiafrasson.curriculo.model.Candidato;
import br.com.rosicleiafrasson.curriculo.model.Cargo;
import br.com.rosicleiafrasson.curriculo.model.Conhecimento;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean
@SessionScoped
public class CandidatoBean {

    private List<Candidato> candidatos;
    private Candidato candidato;
    private List<Cargo> cargos;
    private List<Conhecimento> conhecimentos;

    public CandidatoBean() {
        candidatos = new ArrayList<>();
        candidato = new Candidato();
        cargos = new ArrayList<>();
        conhecimentos = new ArrayList<>();

        cargos.add(new Cargo(1, "Analista de Sistemas"));
        cargos.add(new Cargo(2, "Analista de Qualidade e Processos"));
        cargos.add(new Cargo(3, "Analista Implementador BI"));
        cargos.add(new Cargo(4, "Administrador de Banco de Dados"));
        cargos.add(new Cargo(5, "Analista de Infraestrutura de TI"));
        cargos.add(new Cargo(6, "Analista de Negócio"));
        cargos.add(new Cargo(7, "Analista de Suporte"));
        cargos.add(new Cargo(8, "Analista de Testes"));
        cargos.add(new Cargo(9, "Analista Programador Java"));
        cargos.add(new Cargo(10, "Arquiteto de Software"));
        cargos.add(new Cargo(11, "Designer Gráfico"));
        cargos.add(new Cargo(12, "Estagiário"));

        conhecimentos.add(new Conhecimento(1, "MS Project"));
        conhecimentos.add(new Conhecimento(2, "Modelagem de Dados"));
        conhecimentos.add(new Conhecimento(3, "Gestão da Qualidade ( ISO, CMMI)"));
        conhecimentos.add(new Conhecimento(4, "Gerência de Projetos"));
        conhecimentos.add(new Conhecimento(5, "Testes de Software"));
        conhecimentos.add(new Conhecimento(6, "UML"));
```

```
conhecimentos.add(new Conhecimento(7, "Análise e Projeto de Sistemas"));
conhecimentos.add(new Conhecimento(8, "Usabilidade/Ergonomia de Software"));
conhecimentos.add(new Conhecimento(9, "Pontos de Função"));
conhecimentos.add(new Conhecimento(10, "Metodologias de Desenvolvimento"));
conhecimentos.add(new Conhecimento(11, "Redação"));
conhecimentos.add(new Conhecimento(12, "Trabalho em Equipe"));
conhecimentos.add(new Conhecimento(13, "Enterprise Architect"));
conhecimentos.add(new Conhecimento(14, "Mapeamento de Processos"));
conhecimentos.add(new Conhecimento(15, "Levantamento de Requisitos"));
conhecimentos.add(new Conhecimento(16, "Treinamento de Sistemas"));
conhecimentos.add(new Conhecimento(17, "Implantação de Sistemas"));
conhecimentos.add(new Conhecimento(18, "CSS"));
conhecimentos.add(new Conhecimento(19, "HTML"));

}

public Candidato getCandidato() {
    return candidato;
}

public void setCandidato(Candidato candidato) {
    this.candidato = candidato;
}

public List<Cargo> getCargos() {

    return cargos;
}

public void setCargos(List<Cargo> cargos) {
    this.cargos = cargos;
}

public List<Conhecimento> getConhecimentos() {
    return conhecimentos;
}

public void setConhecimentos(List<Conhecimento> conhecimentos) {
    this.conhecimentos = conhecimentos;
}

public List<Candidato> getCandidatos() {
```

```
        return candidatos;
    }

    public String adicionarCandidato() {
        candidatos.add(candidato);
        candidato = new Candidato();
        return "tabela";
    }
}
```

6. Por fim, as páginas index.xhtml e tabela.xhtml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>Currículo</title>
    </h:head>

    <h:body>

        <div id="cabecalho">
            <h1>Revolução People Soft</h1>
            <p>Vinha fazer parte da empresa que está transformando o Brasil</p>
        </div>

        <div id="pagina">

            <div id="tituloPrincipal">
                <h:panelGrid columns="2">
                    <h:graphicImage value="/imagens/estrela.png" alt="estrela"/>
                    <h3>CADASTRE SEU CURRÍCULO</h3>
                </h:panelGrid>
            </div>

            <div id="formulario">
                <h:form>
```

```
<h:messages/>

<fieldset>

    <legend>DADOS PESSOAIS</legend>

    <h:inputHidden value="#{candidatoBean.candidato.codigo}" />
    <h:panelGrid columns="2">

        <h:outputLabel value="Nome: " for="nome"/>
        <h:inputText value="#{candidatoBean.candidato.nome}" id="nome"
dir="rtl" size="75" required="true" />

        <h:outputLabel value="CPF: " for="cpf"/>
        <h:inputText value="#{candidatoBean.candidato.cpf}" id="cpf" />

        <h:outputLabel value="Data Nascimento: " for="dtNascimento"/>
        <h:panelGroup>
            <h:inputText value="#{candidatoBean.candidato.dataNascimento}"
id="dtNascimento" converterMessage="Data no formato inválido. Utilize DD/MM/AAAA." >
                <f:convertDateTime pattern="dd/MM/yyyy" type="date"/>
            </h:inputText>
            <h:message for="dtNascimento" />
        </h:panelGroup>

        <h:outputLabel value="Sexo: " for="sexo" />
        <h:selectOneRadio value="#{candidatoBean.candidato.sexo}" id="sexo">
            <f:selectItem itemLabel="Feminino" itemValue="F"/>
            <f:selectItem itemLabel="Masculino" itemValue="M"/>
        </h:selectOneRadio>

        <h:outputLabel value="Nacionalidade: " for="nacionalidade" />
        <h:selectOneMenu value="#{candidatoBean.candidato.nacionalidade}"
id="nacionalidade" >
            <f:selectItem itemLabel="Brasileira" itemValue="B"/>
            <f:selectItem itemLabel="Estrangeira" itemValue="E"/>
        </h:selectOneMenu>

        <h:outputLabel value="Raça: " for="raca" />
        <h:selectOneListbox value="#{candidatoBean.candidato.raca}" id="raca"
size="3">
```

```
<f:selectItem itemLabel="Branca" itemValue="Branca"/>
<f:selectItem itemLabel="Preta" itemValue="Preta"/>
<f:selectItem itemLabel="Parda" itemValue="Parda"/>
<f:selectItem itemLabel="Indígena" itemValue="Indígena"/>
<f:selectItem itemLabel="Amarela" itemValue="Amarela"/>
<f:selectItem itemLabel="Não desejo declarar" itemValue="Não
declarou"/>
</h:selectOneListbox>

<h:outputLabel value="Estado civil: " for="estCivil" />
<h:selectOneListbox value="#{candidatoBean.candidato.estadoCivil}"
id="estCivil" size="3">
    <f:selectItem itemLabel="Solteiro" itemValue="Solteiro"/>
    <f:selectItem itemLabel="Casado" itemValue="Casado"/>
    <f:selectItem itemLabel="Viúvo" itemValue="Viúvo"/>
    <f:selectItem itemLabel="Divorciado" itemValue="Divorciado"/>
    <f:selectItem itemLabel="Outro" itemValue="Outro"/>
</h:selectOneListbox>

<h:selectBooleanCheckbox id="deficiente"
value="#{candidatoBean.candidato.ehDeficiente}"/>
<h:outputLabel value="Portador de necessidades especiais"
for="deficiente"/>

<h:outputLabel value="Telefone: " for="telefone"/>
<h:inputText value="#{candidatoBean.candidato.telefone}"
id="telefone"/>

<h:outputLabel value="Email " for="email"/>
<h:inputText value="#{candidatoBean.candidato.email}" id="email"
size="75"/>

</h:panelGrid>
</fieldset>

<br />
<br />

<fieldset>
    <legend>ENDEREÇO</legend>
```

```
<h:panelGrid columns="2">
    <h:outputLabel value="Endereço: " for="endereco"/>
    <h:inputText value="#{candidatoBean.candidato.endereco.endereco}"
id="endereco" size="75" />

    <h:outputLabel value="Complemento: " for="complemento"/>
    <h:inputText value="#{candidatoBean.candidato.endereco.complemento}"
id="complemento" size="75"/>

    <h:outputLabel value="Município: " for="municipio"/>
    <h:inputText value="#{candidatoBean.candidato.endereco.municipio}"
id="municipio" size="75" />

    <h:outputLabel value="Bairro: " for="bairro"/>
    <h:inputText value="#{candidatoBean.candidato.endereco.bairro}"
id="bairro" size="75"/>

    <h:outputLabel value="Estado: " for="estado"/>
    <h:selectOneMenu value="#{candidatoBean.candidato.endereco.estado}"
id="estado" >
        <f:selectItem itemLabel="AC" itemValue="AC"/>
        <f:selectItem itemLabel="AL" itemValue="AL"/>
        <f:selectItem itemLabel="AP" itemValue="AP"/>
        <f:selectItem itemLabel="AM" itemValue="AM"/>
        <f:selectItem itemLabel="BA" itemValue="BA"/>
        <f:selectItem itemLabel="CE" itemValue="CE"/>
        <f:selectItem itemLabel="DF" itemValue="DF"/>
        <f:selectItem itemLabel="ES" itemValue="ES"/>
        <f:selectItem itemLabel="GO" itemValue="GO"/>
        <f:selectItem itemLabel="MA" itemValue="MA"/>
        <f:selectItem itemLabel="MT" itemValue="MT"/>
        <f:selectItem itemLabel="MS" itemValue="MS"/>
        <f:selectItem itemLabel="MG" itemValue="MG"/>
        <f:selectItem itemLabel="PA" itemValue="PA"/>
        <f:selectItem itemLabel="PB" itemValue="PB"/>
        <f:selectItem itemLabel="PR" itemValue="PR"/>
        <f:selectItem itemLabel="PE" itemValue="PE"/>
        <f:selectItem itemLabel="PI" itemValue="PI"/>
        <f:selectItem itemLabel="RJ" itemValue="RJ"/>
        <f:selectItem itemLabel="RN" itemValue="RN"/>
        <f:selectItem itemLabel="RS" itemValue="RJ"/>
```

```
<f:selectItem itemLabel="RO" itemValue="RO"/>
<f:selectItem itemLabel="RR" itemValue="RR"/>
<f:selectItem itemLabel="SC" itemValue="SC"/>
<f:selectItem itemLabel="SP" itemValue="SP"/>
<f:selectItem itemLabel="SE" itemValue="SE"/>
<f:selectItem itemLabel="TO" itemValue="TO"/>
</h:selectOneMenu>
</h:panelGrid>
</fieldset>
<br />
<br />

<fieldset>
<legend>DADOS DE ACESSO</legend>

<h:panelGrid columns="2">

    <h:panelGroup>
        <h:outputLabel value="Login: " for="login" />
        <h:graphicImage value="/imagens/ajuda.png" alt="ajuda"/>
    </h:panelGroup>
    <h:panelGroup>
        <h:inputText value="#{candidatoBean.candidato.login}" id="login"
validatorMessage="O login deve ser composto apenas por letras e deve possuir entre 6 e 18
caracteres.">
            <f:validateRegex pattern="[a-z]{6,18}"/>
        </h:inputText>
        <h:message for="login"/>
    </h:panelGroup>

    <h:outputLabel value="Senha: " for="senha"/>
    <h:panelGroup>
        <h:inputSecret value="#{candidatoBean.candidato.senha}"
id="senha" validatorMessage="A senha deve possuir no mínimo 6 caracteres e no máximo 10
caracteres.">
            <f:validateLength maximum="10" minimum="6"/>
        </h:inputSecret>
        <h:message for="senha"/>
    </h:panelGroup>

</h:panelGrid>
```

```
</fieldset>

<br />
<br />

<fieldset>
    <legend>FORMAÇÃO E EXPERIÊNCIAS PROFISSIONAIS</legend>

    <h:panelGrid columns="2">

        <h:outputLabel value="Formação: " for="formacao"/>
        <h:selectManyListbox id="formacao"
value="#{candidatoBean.candidato.formacoes}" size="3">
            <f:selectItem itemValue="1" itemLabel="Ensino Médio"/>
            <f:selectItem itemValue="2" itemLabel="Curso Técnico"/>
            <f:selectItem itemValue="3" itemLabel="Graduação"/>
            <f:selectItem itemValue="4" itemLabel="Especialização"/>
            <f:selectItem itemValue="5" itemLabel="Mestrado"/>
            <f:selectItem itemValue="6" itemLabel="Doutorado"/>
        </h:selectManyListbox>
        <h:outputText/>
        <h:outputText/>

        <h:outputLabel value="Idiomas: " for="idiomas"/>
        <h:selectManyMenu id="idiomas"
value="#{candidatoBean.candidato.idiomas}" style="height: 90px" >
            <f:selectItem itemValue="Inglês"/>
            <f:selectItem itemValue="Francês"/>
            <f:selectItem itemValue="Alemão"/>
            <f:selectItem itemValue="Espanhol"/>
            <f:selectItem itemValue="Mandarim"/>
        </h:selectManyMenu>
        <h:outputText/>
        <h:outputText/>
    </h:panelGrid>

    <h:panelGrid columns="1">
        <h:outputLabel value="Habilidades/Conhecimentos: "
for="conhecimento"/>
            <h:selectManyCheckbox id="conhecimento"
value="#{candidatoBean.candidato.conhecimentos}" layout="pageDirection" >
```

```
        <f:selectItems value="#{candidatoBean.conhecimentos}"/>
    </h:selectManyCheckbox>

    <h:outputLabel value="Cargos Pretendidos: " for="cargosPre"/>
    <h:selectManyListbox id="cargosPre"
value="#{candidatoBean.candidato.cargosPretendidos}">
        <f:selectItems value="#{candidatoBean.cargos}"/>
    </h:selectManyListbox>

    <h:outputLabel value="Experiências Profissionais: "
for="exProfissional"/>
    <h:inputTextarea value="#{candidatoBean.candidato.experiencias}"
id="exProfissional" rows="10" cols="80"/>

    <h:outputLabel value="Último Salário: " for="ultimoSalario" />
    <h:inputText value="#{candidatoBean.candidato.ultimoSalario}"
id="ultimoSalario">
        <f:convertNumber currencySymbol="R$" type="currency"/>
    </h:inputText>

    <h:outputLabel value="Pretensão Salarial: " for="pretensaoSalarial"
/>
    <h:inputText value="#{candidatoBean.candidato.pretensaoSalarial}"
id="pretensaoSalarial">
    </h:inputText>

    <h:outputLabel value="Carga Horária: " for="carga" />
    <h:inputText value="#{candidatoBean.candidato.cargaHoraria}"
id="carga">
    </h:inputText>

</h:panelGrid>
</fieldset>

<br />

<h:commandButton value="Cadastrar"
action="#{candidatoBean.adicionarCandidato}" />

<br />
```

```
<br />

<h:outputLink value="http://www.google.com">
    <h:outputText value="Para mais informações clique aqui"/>
</h:outputLink>
<br />
<br />
<br />

</h:form>
</div>
</div>
</h:body>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

<h:head>
    <title>Currículo</title>
</h:head>

<h:body>
    <h:form>
        <h3>Candidatos</h3>
        <h:dataTable border="1" var="candidatos" value="#{candidatoBean.candidatos}">
            <h:column>
                <f:facet name="header">Nome</f:facet>
                #{candidatos.nome}
            </h:column>

            <h:column>
                <f:facet name="header">CPF</f:facet>
                #{candidatos.cpf}
            </h:column>

            <h:column>
```

```
<f:facet name="header">Data de Nascimento</f:facet>
<h:outputText value="#{candidatos.dataNascimento}">
    <f:convertDateTime pattern="dd/MM/yyyy" />
</h:outputText>
</h:column>

<h:column>
    <f:facet name="header">Telefone</f:facet>
    #{candidatos.telefone}
</h:column>

</h:dataTable>
<br/>
<h:commandButton value="Cadastrar" action="cadastro" />
</h:form>
</h:body>
</html>
```

Vale lembrar que a codificação apresentada salva o candidato apenas em uma lista. Sendo assim, os dados são temporários, ou seja, para cada execução é criada uma lista vazia. Serve apenas para exemplificar o uso dos componentes JSF.
Executando o projeto, a página exibida deve ser semelhante a figura a seguir.

Revolução People Soft

Venha fazer parte da empresa que está transformando o Brasil

The screenshot shows a web page with the title "Revolução People Soft" and a subtitle "Venha fazer parte da empresa que está transformando o Brasil". Below this, there is a large green star icon and the text "CADASTRE SEU CURRÍCULO". The form is divided into sections:

- DADOS PESSOAIS**:
 - Nome: [Text input field]
 - CPF: [Text input field]
 - Data Nascimento: [Text input field]
 - Sexo:
○ Feminino
○ Masculino
 - Nacionalidade: [Select dropdown] set to "Brasileira".
 - Branca
 - Preta
 - Parda
 - Raça:
 - Solteiro
 - Casado
 - Viúvo
 - Estado civil: [Select dropdown]

Candidatos

Nome	CPF	Data de Nascimento	Telefone
Mario Antunes da Costa	789.789.564-09	10/10/1987	(48) 3632 - 2345
Fernanda Torres	345.678.123-78	05/06/1856	(48) 9932 - 5678

Cadastrar

PASSO-A-PASSO

Com o intuito de deixar o projeto apresentado - cadastro de currículo - com uma melhor aparência serão inseridas algumas regras de estilo.

1. No projeto criado no capítulo anterior, é necessário indicar a folha de estilo que será aplicada a página.

```
<h:head>
    <title>Currículo</title>
    <h:outputStylesheet library="css" name="style.css"/>
</h:head>
```

2. O próximo passo é a criação da folha de estilo. Segue a seguir a codificação da mesma.

```
body{
    margin: 0;
    padding: 0;
    background: #F6F6F6;
    font-size: 13px;
    text-align: justify;
}

#cabecalho{
    width: 800px;
    height: 100px;
    margin: 0 auto;
}

#cabecalho h1, #cabecalho p{
    margin: 0;
    padding: 0;
}

#cabecalho h1{
```

```
        font-size: 52px;
        color: #20A001;
    }

#cabecalho p{
    margin-top: 0px;
    margin-left: 30px;
    padding: 0 0 0 4px;
    font-size: 15px;
    font-weight: normal;
    color: #86EA66;
    text-transform: uppercase;
}

#pagina{
    width: 800px;
    margin: 0 auto;
    padding: 0px;
    background: #ffffff;
    box-shadow: 5px 4px 10px #333, -5px -5px 10px #ccc;

}

#tituloPrincipal{
    text-transform: uppercase;
    background-color: #20A001;
    color: #FFFFFF;
    font-size: 15px;
}

#formulario{
    margin: 20px;
}

#formulario input, textarea, select, option, body{
    font-family: Arial, Helvetica, sans-serif;
    color: #83826A;
}

fieldset{
    border-radius: 5px;
}
```

Revolução People Soft

VENHA FAZER PARTE DA EMPRESA QUE ESTÁ TRANSFORMANDO O BRASIL



CADASTRE SEU CURRÍCULO

DADOS PESSOAIS

Nome:

CPF:

Data Nascimento:

Sexo: Feminino Masculino

Nacionalidade: Brasileira

Raça:
Branca
Preta
Parda

Estado civil:
Solteiro
Casado
Viúvo



Portador de necessidades especiais

Telefone:

Email:

ENDEREÇO

Endereço:

Complemento:

Município:

Escopos JSF

Request Scope

Os objetos armazenados com escopo request sobrevivem apenas a uma submissão do ciclo de vida do JSF, ou seja, os dados recebidos são processados e uma resposta é enviada ao cliente, após o envio da resposta os dados são apagados da memória do servidor. A cada requisição é criada uma nova instância do managed bean, sendo assim, as informações não são compartilhadas entre as requisições.

O escopo request possui o menor tempo de vida entre os escopos, sendo assim os objetos permanecem pouco tempo na memória, melhorando a performance da aplicação. RequestScoped é o escopo apropriado em telas onde não são necessárias chamadas ajax e não é necessário armazenar dados entre as requisições dos usuários. Um bom exemplo para utilização desse escopo é a submissão de um formulário simples.

```
@ManagedBean  
 @RequestScoped  
 public class ExemploEscopoRequest {  
 }
```

Os managed beans com escopo request não necessitam da anotação @RequestScoped, pois este é o escopo padrão. Porém é uma boa prática utilizar a anotação.

Session Scope

Os objetos armazenados com o escopo session sobrevivem enquanto a seção do usuário estiver ativa. Em outras palavras, ao utilizar este escopo, os atributos do managed bean terão seus valores mantidos até o fim da sessão do usuário.

O escopo de sessão é recomendado para o armazenamento de informações do usuário e dados de preferência deste. Vale ressaltar que a sessão é definida pelo vínculo do usuário com o navegador. Sendo assim, se dois navegadores distintos forem abertos pelo mesmo usuário, duas seções diferentes serão criadas no servidor.

Uma seção pode ser destruída em duas situações: a primeira delas é quando a própria aplicação decide finalizar a sessão, que é o que ocorre quando um usuário faz o logout e a outra situação se dá quando o servidor decide expirar a sessão. O tempo para que uma seção seja expirada pode ser configurada no web.xml.

```
@ManagedBean  
 @SessionScoped  
 public class ExemploEscopoSession {  
 }
```

Embora a utilização do Session Scope seja relativamente fácil, é importante ter muita cautela ao anotar um managed bean com este escopo. É importante lembrar que um objeto com escopo session permanecerá na memória durante toda a seção. Isso significa que quanto maior a quantidade de usuários maior será a quantidade de memória que o servidor terá que gerenciar, ou seja, a cada nova seção criada um novo managed bean com todos os atributos será alocado na memória do servidor.

Application Scoped

A utilização do escopo de aplicação cria uma instância do managed bean no momento em que a classe é requisitada e a mantém até a finalização da aplicação. Com o application Scoped, o managed bean estará disponível para todos os usuários da aplicação enquanto a mesma estiver sendo executada. É importante ressaltar que informações que não devem ser compartilhadas não devem possuir este escopo.

O escopo de aplicação geralmente é utilizado para guardar valores de configuração e realização de cache manual como o carregamento de listagem de estados e municípios.

```
@ManagedBean  
@ApplicationScoped  
public class ExemploEscopoAplicacao {  
}
```

Através da configuração @ManagedBean (eager = true), o servidor instancia o managed bean antes que qualquer tela da aplicação seja acessada. Isto significa que a informação será carregada em memória antes de ser solicitada pelo usuário.

View Scoped

O escopo de visão mantém os dados enquanto o usuário permanecer na página. No momento em que há troca de página o objeto é excluído da memória.

View Scoped possui um tempo de vida maior que o Request Scoped e menor que o Session Scoped e é muito indicado para páginas que possuem requisições ajax.

É importante salientar que um managed bean com anotação @ViewScoped só é removido da memória se a mudança de página for feita pelo método POST o que pode acarretar em objetos desnecessários na memória, caso a navegação for feita via link.

```
@ManagedBean  
@ViewScoped  
public class ExemploEscopoView {  
}
```

PASSO-A-PASSO

Com o intuito de demonstrar a diferença entre os escopos dos managed beans, deve ser criado um novo projeto JSF.

1. O pacote modelo deve conter uma classe com o nome de Aluno. Esse aluno deve possuir o atributo nome.

```
package br.com.rosicleiafrasson.cap11escopos.modelo;  
  
public class Aluno {
```

```
private String nome;

public String getNome() {
    return nome;
}

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

2. Para controlar os dados recebidos e enviados pela visualização, é necessário criar um managed bean. Neste primeiro exemplo será utilizado o escopo request.

```
package br.com.rosicleiafrasson.cap11escopos.controle;

import br.com.rosicleiafrasson.cap11escopos.modelo.Aluno;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;

@ManagedBean
@RequestScoped
public class AlunoBean {

    private List<Aluno> alunos = new ArrayList<>();
    private Aluno aluno = new Aluno();

    public String adicionaAluno() {
        this.alunos.add(this.aluno);
        this.aluno = new Aluno();
        return "index";
    }

    public List<Aluno> getAlunos() {
        return alunos;
    }

    public Aluno getAluno() {
        return aluno;
    }
}
```

```
public void setAluno(Aluno aluno) {  
    this.aluno = aluno;  
}  
}
```

3. A página de visualização deve ser composta por um campo de texto e um botão para adição dos alunos e um componente de repetição para exibir todos os alunos cadastrados. Como a aplicação não está conectada a uma base de dados, a lista dos alunos será mantida durante o tempo de vida do managed bean.

```
<?xml version='1.0' encoding='UTF-8' ?>  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html"  
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">  
  <h:head>  
    <title>Primeira página JSF</title>  
  </h:head>  
  <h:body>  
    <h:form>  
      <h:panelGrid columns="2">  
        <h:outputLabel value="Nome: " for="nome"/>  
        <h:inputText value="#{alunoBean.aluno.nome}" id="nome"/>  
  
        <h:commandButton value="Adicionar"  
action="#{alunoBean.adicionaAluno}"/>  
      </h:panelGrid>  
    </h:form>  
  
    <h:panelGroup rendered="#{not empty alunoBean.alunos}">  
      <h1>Alunos cadastrados</h1>  
      <ul>  
        <ui:repeat value="#{alunoBean.alunos}" var="aluno">  
          <li>  
            <h:outputText value="#{aluno.nome}"/>  
          </li>  
        </ui:repeat>  
      </ul>  
    </h:panelGroup>  
  </h:body>  
</html>
```

Executando a aplicação uma página semelhante a página mostrada a seguir deve ser exibida.



É importante perceber que quando o escopo do managed bean é request, a cada requisição, ou seja, cada vez que o usuário acionar o botão adicionar, a lista fica vazia. Isso ocorre porque os managed beans com escopo request duram o tempo de uma requisição.

Alterando o managed bean para o escopo de sessão, é possível perceber que os alunos permanecem na lista cada vez que o usuário aciona o botão de adicionar.

```
package br.com.rosicleiafrasson.cap11escopos.controle;

import br.com.rosicleiafrasson.cap11escopos.modelo.Aluno;

import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean
@SessionScoped
public class AlunoBean {
```

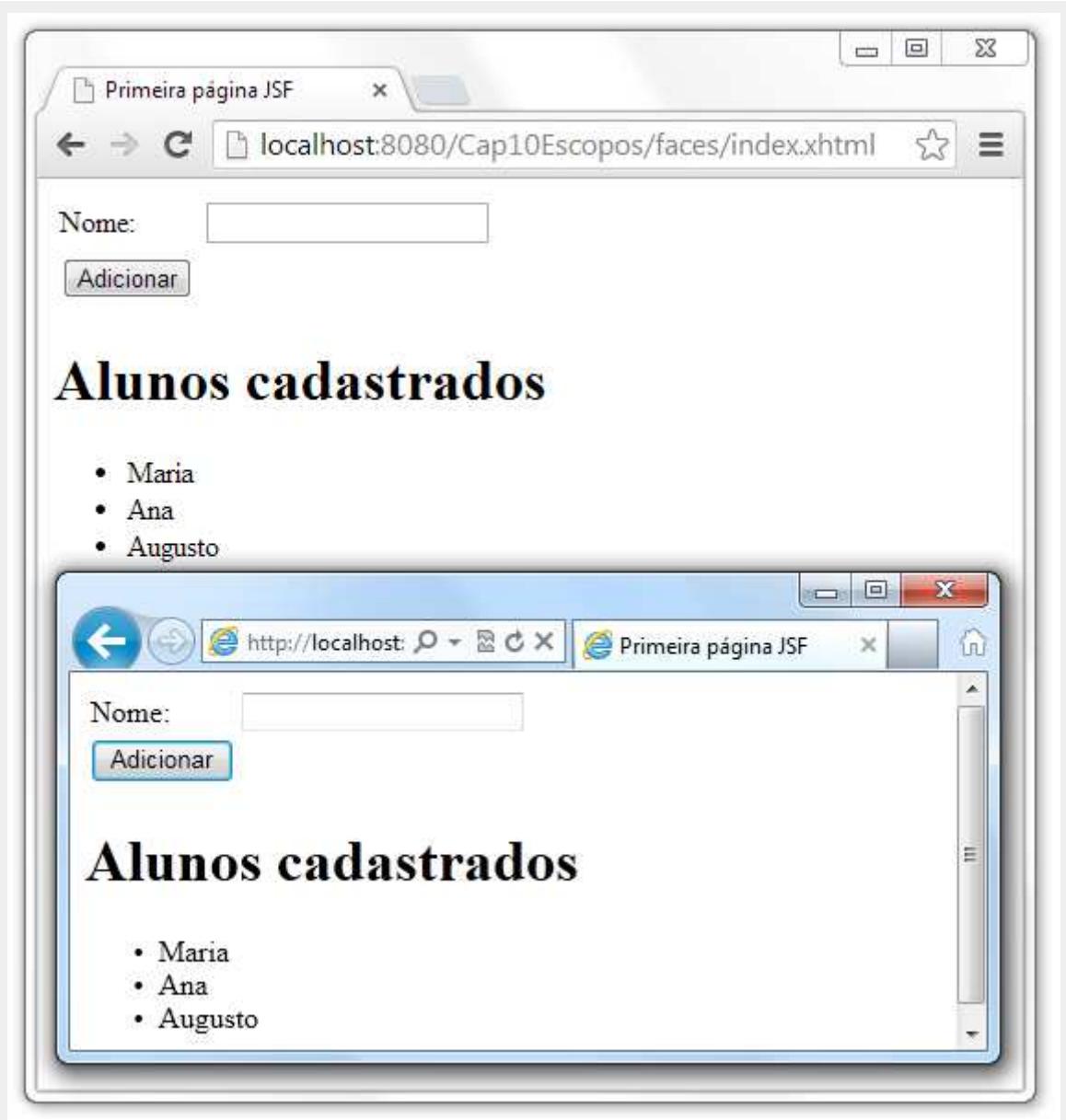
Como a seção muda a cada usuário ou navegador distinto, é necessário abrir a página em um outro navegador para visualizar como o SessionScoped funciona. Como mostra a imagem a seguir, o mesmo software em seções distintas possui em sua lista de alunos, valores distintos.



Se o managed bean for alterado para escopo de aplicação, é possível perceber que mesmo em seções distintas os valores contidos na lista de alunos continuam os mesmos.

```
ackage br.com.rosicleiafrasson.cap11escopos.controle;
import br.com.rosicleiafrasson.cap11escopos.modelo.Aluno;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ApplicationScoped;
import javax.faces.bean.ManagedBean;

@ManagedBean
@ApplicationScoped
public class AlunoBean {
```



Para finalizar resta exemplificar o escopo de visão. Como já mencionado o escopo de visão existe na memória enquanto o usuário permanecer na página exibida. No exemplo exposto, o método adicionar é um método void. Isso significa que ao acionar o botão adicionar, a página continua a mesma. Portanto, para demonstrar este escopo é necessário uma modificação no método adicionar, para que o mesmo seja redirecionado. No exemplo mostrado será redirecionado para a mesma página, mas o comportamento será o mesmo se a página redirecionada for diferente.

```
package br.com.rosicleiafrasson.cap11escopos.controle;

import br.com.rosicleiafrasson.cap11escopos.modelo.Aluno;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;
```

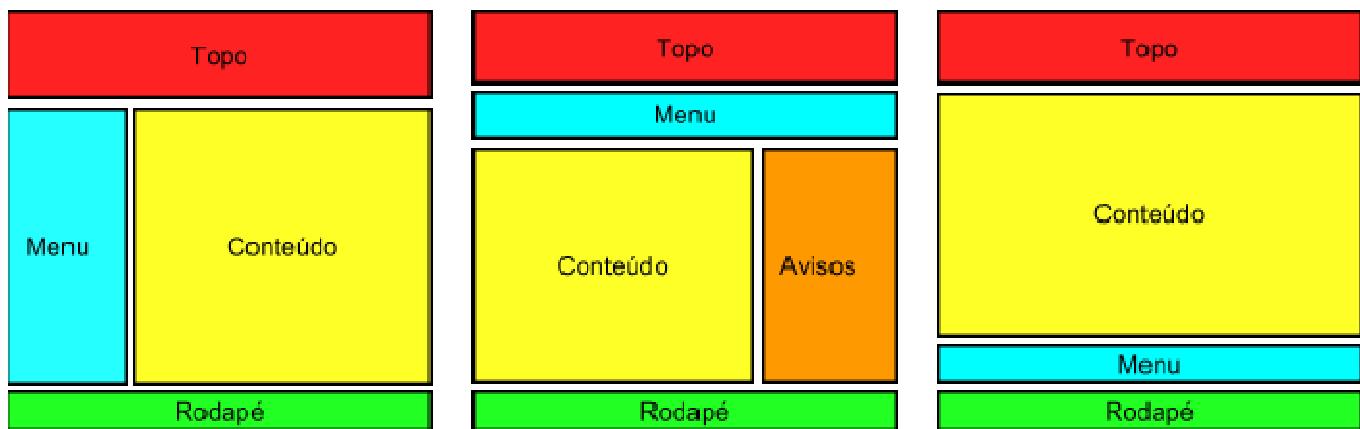
```
@ManagedBean  
@ViewScoped  
public class AlunoBean {  
  
    private List<Aluno> alunos = new ArrayList<>();  
    private Aluno aluno = new Aluno();  
  
    public String adicionaAluno() {  
        this.alunos.add(this.aluno);  
        this.aluno = new Aluno();  
        return "index";  
    }  
}
```

Executando o projeto é possível perceber que a lista está sempre vazia. Isso ocorre porque cada vez que o botão adicionar é acionado a página é redirecionada. O ViewScoped é muito útil em páginas com chamadas ajax.

Facelets

Facelets é um framework utilizado para a construção de páginas com templates padronizados, bem como para a criação de componentes customizáveis.

As aplicações, em sua grande maioria, possuem um modelo padrão para as páginas. O menu, o cabeçalho e o rodapé tendem a ter a mesma estrutura no aplicativo inteiro. Codificar estes componentes em todas as páginas torna difícil a manutenção das mesmas. O suporte a templates do facelets permite a definição de uma estrutura visual padrão que pode ser reaproveitada em diversas telas.



Na padronização das telas com o uso do facelets é possível construir uma estrutura fixa com o que é semelhante em todas as telas como topo, menu e rodapé.

Além da facilidade de manutenção, redução drástica na quantidade de código e consequente diminuição no tempo de desenvolvimento, a utilização de facelets provê outras vantagens. A principal delas é que com a utilização da tecnologia o código a execução do código fica de 30% a 50% mais rápida do que em páginas jsp. O facelets é independente de servidor de aplicação e está incorporado ao JSF 2.0.

Para construir um template para uma aplicação, é necessário identificar um padrão em um determinado conjunto de telas. Em posse deste padrão, faz-se a montagem do esqueleto utilizando trechos estáticos e dinâmicos. Vale ressaltar que um template nunca será executado, pois é uma página incompleta, deste modo, é importante que os templates sejam salvos dentro da pasta WEB-INF para que o usuário não consiga acessá-las diretamente.

Em páginas que utilizam facelets é necessário importar o namespace de facelets: <http://xmlns.jcp.org/jsf/facelets>. Seguem algumas tags dessa biblioteca:

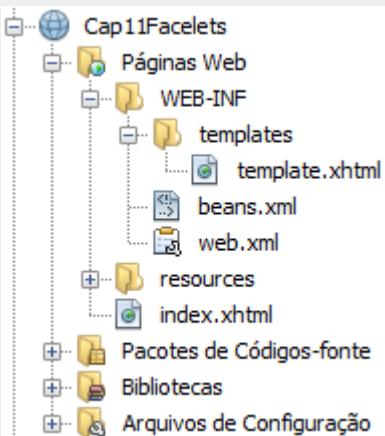
Tag	Descrição
ui:insert	Define uma área de substituição no template, ou seja, demarca onde será, por exemplo, o menu, o topo e o corpo da página.
ui: include	Insere um fragmento de tela dentro de uma página. Essa tag é muito utilizada quando o código fonte de uma página fica muito extenso. Esse código pode ser separado em vários arquivos, o que facilita a manutenção.
ui:define	Define o conteúdo de uma área criada no template como ui:insert.
ui:composition	Define o relacionamento de composição entre a tela e o template.

PASSO-A-PASSO

Criação de um template

Como já mencionado um template nunca deve ser executado, pois trata-se de uma página incompleta. Dessa forma é recomendado que os templates fiquem dentro da pasta WEB-INF para que o usuário não tenha acesso a mesma.

1. Após a criação de um novo projeto, é necessário criar uma pasta templates dentro do diretório WEB-INF. Dentro da pasta criada, é necessário um arquivo com extensão .xhtml. No caso ilustrado foi criado o arquivo template.xhtml.



2. No template criado é necessário definir os trechos estáticos e dinâmicos. O trecho de código apresentado possui o trecho dinâmico conteúdo e os trechos estáticos: topo, menuLateral e rodapé.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:head>

        </h:head>

    <h:body>

        <div id="topo">
            Topo
        </div>

        <div id="conteudo">
            <ui:insert name="conteudo">Conteúdo</ui:insert>
        </div>

    </h:body>

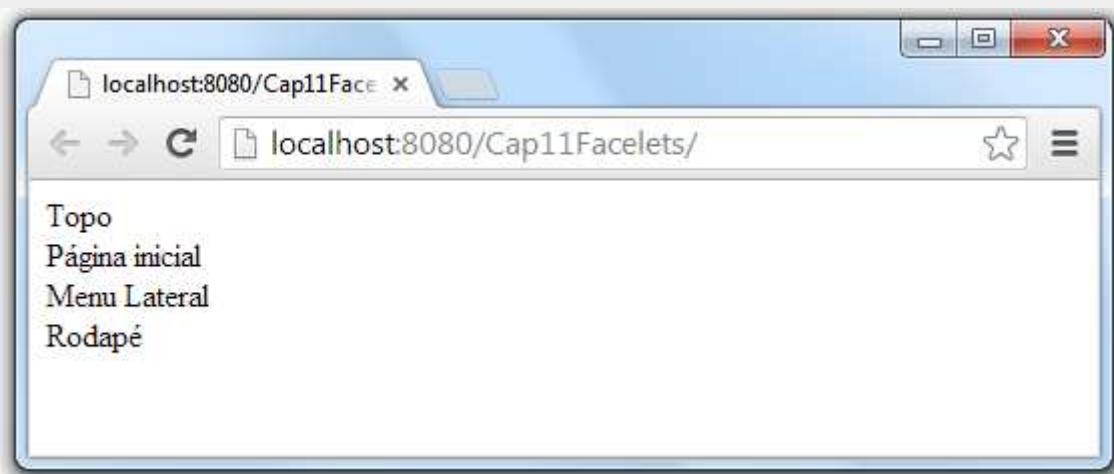
```

```
<div id="menuLateral">  
    Menu Lateral  
</div>  
  
<div id="rodape">  
    Rodapé  
</div>  
  
</h:body>  
  
</html>
```

3. Na utilização deste template, é importante definir o que será colocado no template nos trechos dinâmicos. Neste caso o único trecho dinâmico é o conteúdo. Portanto a página index, cujo código está mostrado a seguir possui apenas a definição do conteúdo, o menu, rodapé e cabeçalho exibidos serão os que foram definidos no template.

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"  
    template=".//WEB-INF/templates/template.xhtml"  
    xmlns = "http://www.w3.org/1999/xhtml">  
  
    <ui:define name="conteudo">  
        Página inicial  
    </ui:define>  
  
</ui:composition>
```

4. Ao executar a página index do projeto, deve ser exibida uma página semelhante a imagem a seguir.

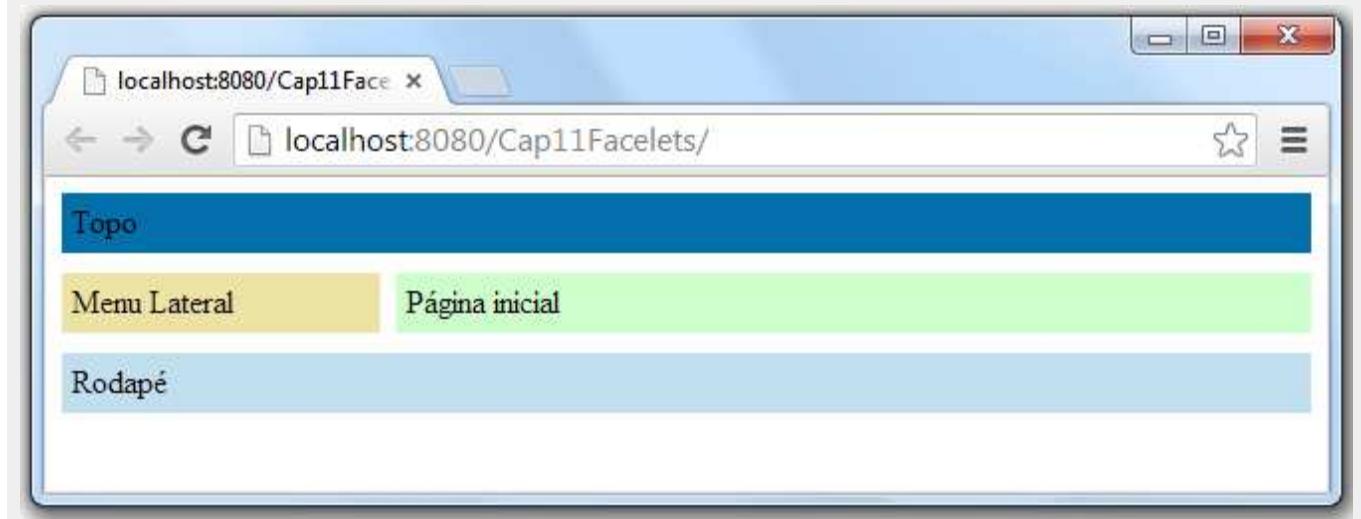


É importante perceber que a página exibida possui trechos definidos no template e na página

index.

5. O posicionamento de cada trecho deve ser feito utilizando CSS.

```
#topo {  
    position: relative;  
    background-color: #036fab;  
    padding: 5px;  
    margin: 0px 0px 10px 0px;  
}  
  
#rodape {  
    position: relative;  
    background-color: #c2defe;  
    padding: 5px;  
    margin: 10px 0px 0px 0px;  
}  
  
#menuLateral {  
    float: left;  
    background-color: #ece3a5;  
    padding: 5px;  
    width: 150px;  
}  
  
#conteudo {  
    float: right;  
    background-color: #ccffcc;  
    padding: 5px;  
    width: 450px;  
}
```



6. Para simplificar a construção das páginas, também é possível que as mesmas sejam divididas. No exemplo mostrado o menu poderia ser criado em uma página distinta e incluso no template.

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns ="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">

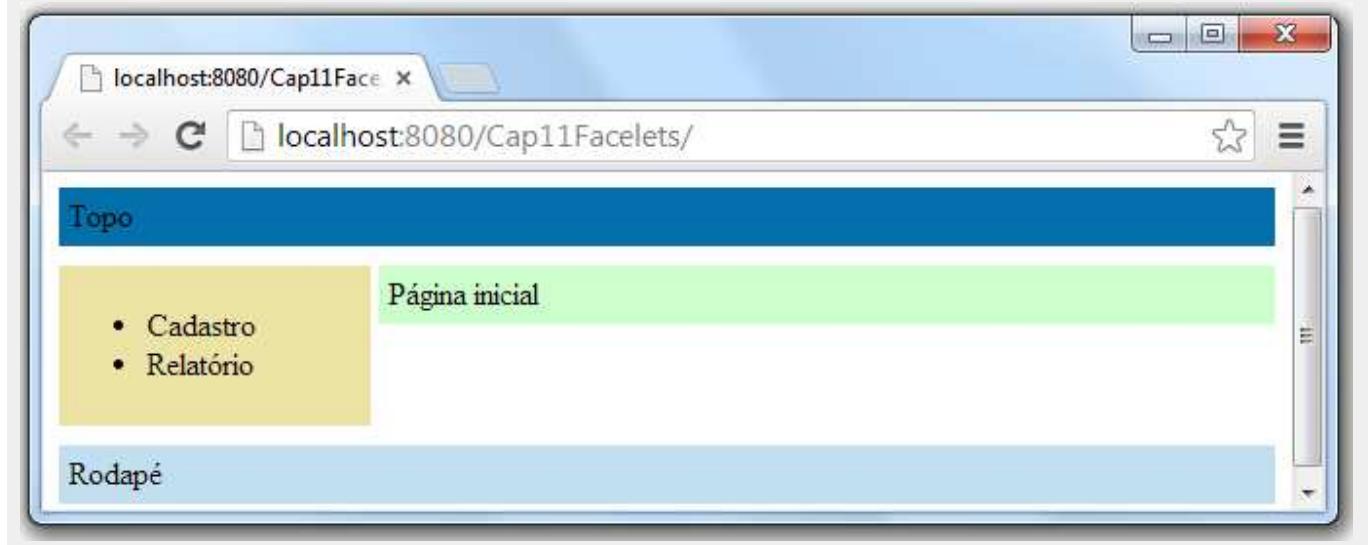
    <h:form>
        <ul>
            <li>Cadastro</li>
            <li>Relatório</li>
        </ul>

    </h:form>

</ui:composition>
```

7. Na página onde o menu deve ser incluso, neste caso, o template, deve ser adicionado o comando a seguir.

```
<div id="menuLateral">
    <ui:include src="menu.xhtml"/>
</div>
```



Filtros

Lançados na especificação Servlet 2.3, os filtros são classes que permitem que códigos sejam executados antes da requisição e depois que a resposta é gerada, ou seja, eles permitem que os objetos HttpServletRequest e HttpServletResponse sejam acessados antes dos servlets. Os filtros interceptam requisições e respostas, sendo totalmente transparentes para os clientes e para os servlets.

A nível de implementação um filtro é uma classe Java que implementa a interface javax.servlet.Filter.

```
public class JPAFilter implements Filter{
```

A interface Filter exige a implementação de três métodos: init, destroy e doFilter.

```
@Override
public void init(FilterConfig filterConfig) throws ServletException {
}

@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
}

@Override
public void destroy() {
}
```

- init: método executado quando o filtro é carregado pelo container. Como parâmetro é passado um FilterConfig. Um objeto FilterConfig representa a configuração para o filtro. Desse objeto é possível obter o nome do filtro, os parâmetros de inicialização e o ServletContext.
- destroy: método executado quando o filtro é descarregado no container. Este método avisa o filtro que ele está sendo desativado, para que o mesmo possa liberar eventuais recursos da memória.
- doFilter: método que executa a filtragem. O método doFilter recebe três parâmetros: ServletRequest, ServletResponse e FilterChain. Os objetos ServletRequest e ServletResponse são os mesmos que serão passados a um servlet.

Não é interessante que um filtro processe toda a requisição. A grande utilidade dos filtros é a interceptação de vários requests semelhantes, sendo assim, os filtros são interessantes para encapsular tarefas recorrentes. Cada filtro encapsula apenas uma responsabilidade. Porém, eles podem ser concatenados ou encadeados para permitir que uma requisição passe por mais de um filtro.

O objeto FilterChain é utilizado pelo Filter para executar o próximo filtro, ou, se este for o último, indicar ao container que este deve seguir seu processamento. Sendo assim, os comandos colocados antes da chamada chain.doFilter são executados na ida e os comandos colocados após são executados na volta.

```
chain.doFilter(request, response);
```

É importante ressaltar que em todas as requisições feitas ao container é verificado se existe um

filtro associado ao recurso solicitado. Em caso afirmativo, a requisição é redirecionada para o filtro. Este por sua vez executa os seus comandos e após permite que o processamento normal do request prossiga. Os filtros também podem ser utilizados para tomada de decisões, ou seja, eles podem decidir se uma requisição é executada ou se interrompem o caminho normal da requisição.

O uso de filtros permite que informações sejam armazenadas na requisição. Para efetuar este armazenamento é utilizado o método setAttribute no request. O método setAttribute necessita de uma identificação para o objeto que está sendo armazenado na requisição e o objeto que deve ser guardado. Para acessar este objeto é utilizado o método getAttribute.

```
request.setAttribute("EntityManager", manager);
```

Para funcionar os filtros devem ser registrados. Esse registro pode ser feito no web.xml ou através de anotação. A seguir está sendo demonstrado como registrar um filtro através de anotação. O atributo servletNames define os servlets que serão filtrados pelo filtro.

```
@WebFilter(servletNames = "Faces Servlet")
```

PASSO-A-PASSO

Conexão com banco de dados com Filter

A inicialização e finalização de uma unidade de persistência deve ser efetuada apenas uma vez durante a execução da aplicação. Como já mencionado, os filtros possuem esta característica. Eles são carregados quando a aplicação é executada e destruídos quando a aplicação é finalizada. Sendo assim, será demonstrado a utilização de um filtro para a criação da fábrica de conexão e controle das transações com a base de dados.

1. Após criar um novo projeto, o primeiro passo é a criação da unidade de persistência. No Netbeans é possível a criação de uma unidade de persistência de forma automática como mostrado no capítulo 1. O código da unidade de persistência deve ficar similar ao código abaixo.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" 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_2_0.xsd">
  <persistence-unit name="cap13ConexaoBancoFiltroPU" transaction-
  type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="hibernate.hbm2ddl.auto"
        value="update"/>
      <property name ="hibernate.dialect"
        value ="org.hibernate.dialect.MySQLDialect"/>
      <property name ="javax.persistence.jdbc.driver"
        value ="com.mysql.jdbc.Driver"/>
      <property name ="javax.persistence.jdbc.user"
```

```

        value = "root"/>
<property name ="javax.persistence.jdbc.password"
          value =""/>
<property name ="javax.persistence.jdbc.url"
          value
="jdbc:mysql://localhost:3306/cap13ConexaoBancoFiltro"/>
</properties>
</persistence-unit>
</persistence>
```

2. Com o persistence.xml criado, é necessário criar uma classe, neste exemplo, com o nome de JPAFilter dentro do pacote filter. Essa classe deve implementar a interface javax.servlet.Filter.

```

import javax.servlet.Filter;
public class JPAFilter implements Filter{
}
```

3. A interface Filter exige que sejam implementados os métodos init, destroy e doFilter.

```

@Override
public void init(FilterConfig filterConfig) throws ServletException {
}

@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
}

@Override
public void destroy() {
}
```

4. A classe EntityManagerFactory descobre quem cria as conexões e é necessário uma variável para armazenar uma instância da mesma.

```
private EntityManagerFactory factory;
```

5. Para obter uma instância de EntityManagerFactory é utilizado o método createEntityManagerFactory indicando qual é a persistence-unit que foi definida no persistence.xml, no nosso caso cap13ConexaoBancoFiltro. Esse procedimento é feito no método init.

```

public void init(FilterConfig filterConfig) throws ServletException {
    this.factory = Persistence.createEntityManagerFactory("cap13ConexaoBancoFiltroPU");
}
```

6. O método destroy é responsável por destruir a instância de EntityManagerFactory.

```
public void destroy() {
```

```
        this.factory.close();
    }
```

7. O método doFilter abre a conexão, armazena o gerenciador de entidades no request, inicializa os recursos da transação com o método begin. O método commit confirma a transação caso não ocorra algum erro ou rollback caso algum erro ocorrer.

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
    EntityManager manager = this.factory.createEntityManager();
    request.setAttribute("EntityManager", manager);
    manager.getTransaction().begin();
    chain.doFilter(request, response);

    try {
        manager.getTransaction().commit();
    } catch (Exception e) {
        manager.getTransaction().rollback();
    } finally {
        manager.close();
    }
}
```

8. Para funcionar, o filtro tem que ser registrado. A anotação exibida a seguir indica que o filtro será aplicado no Faces Servlet, que é o servlet do JSF.

```
@WebFilter(servletNames = "Faces Servlet")
```

9. Com o intuito de testar o filtro, será construída a aplicação completa. Sendo assim, o próximo passo é criar os beans da aplicação. Neste caso, será simulado um software para utilização de um rh para controle dos funcionários. Os beans já estão com as anotações da JPA.

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Lob;

@Entity
public class Cargo {

    @Id @GeneratedValue
    private int codigo;
    private String nome;
```

```
@Lob  
private String descricao;  
//Gets e sets  
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
  
@Entity  
public class Endereco {  
  
    @Id @GeneratedValue  
    private int codigo;  
    private String endereco;  
    private String bairro;  
    private String cidade;  
    private String estado;  
    private int numero;  
    private String complemento;  
    //Gets e sets  
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model;  
  
import java.util.Date;  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.ManyToOne;  
import javax.persistence.OneToOne;  
import javax.persistence.Temporal;  
import javax.persistence.TemporalType;  
  
@Entity  
public class Funcionario {
```

```
    @Id  
    @GeneratedValue  
    private int codigo;  
    private String nome;  
    private String cpf;  
    @Temporal(TemporalType.DATE)  
    private Date dataNascimento;  
    @ManyToOne  
    private Cargo cargo;  
    private String telefone;  
    private String email;  
    @OneToOne(cascade = CascadeType.ALL)  
    private Endereco endereco;  
  
    public Funcionario() {  
        endereco = new Endereco();  
    }  
    //Gets e sets  
}
```

10. A camada de persistência será construída utilizando um DAO genérico. Dessa forma deve conter uma interface com todos os métodos comuns e a implementação desta. É importante perceber que diferente do CRUD apresentado no capítulo 2, com a utilização de filtros a classe DAOJPA não possui os comandos referentes ao controle de transações. Este controle está implementado no JPAFilter. Também não é necessária a criação de uma classe para servir como fábrica de EntityManager, esta fábrica também está embutida no JPAFilter.

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao;  
  
import java.io.Serializable;  
import java.util.List;  
  
public interface DAO<T, I extends Serializable> {  
  
    void save(T entity);  
  
    void remove(Class<T> classe, I pk);  
  
    T getById(Class<T> classe, I pk);  
  
    List<T> getAll(Class<T> classe);  
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia;

import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao.DAO;
import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import javax.persistence.Query;

public class DAOJPA<T, I extends Serializable> implements DAO<T, I> {

    private EntityManager manager;

    public DAOJPA(EntityManager manager) {
        this.manager = manager;
    }

    @Override
    public void save(T entity) {
        this.manager.merge(entity);
    }

    @Override
    public void remove(Class<T> classe, I pk) {
        T entity = this.getById(classe, pk);
        this.manager.remove(entity);
    }

    @Override
    public T getById(Class<T> classe, I pk) {
        try {
            return this.manager.find(classe, pk);
        } catch (NoResultException e) {
            return null;
        }
    }

    @Override
    public List<T> getAll(Class<T> classe) {
        Query q = this.manager.createQuery("select x from "
            + classe.getSimpleName() + " x");
    }
}
```

```
        return q.getResultList();
    }
}
```

11. Para que as entidades possam utilizar o DAO genérico, é necessário criar a interface e sua implementação para cada uma delas.

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao;

import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Cargo;

public interface CargoDAO extends DAO<Cargo, Integer> {
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia;

import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Cargo;
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao.CargoDAO;
import javax.persistence.EntityManager;

public class CargoDAOJPA extends DAOJPA<Cargo, Integer> implements CargoDAO {

    public CargoDAOJPA(EntityManager manager) {
        super(manager);
    }
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao;

import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Endereco;

public interface EnderecoDAO extends DAO<Endereco, Integer> {
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia;

import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Endereco;
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao.EnderecoDAO;
import javax.persistence.EntityManager;

public class EnderecoDAOJPA extends DAOJPA<Endereco, Integer> implements EnderecoDAO {
```

```
public EnderecoDAOJPA(EntityManager manager) {  
    super(manager);  
}  
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao;  
  
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Funcionario;  
  
public interface FuncionarioDAO extends DAO<Funcionario, Integer> {  
}
```

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia;  
  
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Funcionario;  
import  
br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao.FuncionarioDAO;  
import javax.persistence.EntityManager;  
  
public class FuncionarioDAOJPA extends DAOJPA<Funcionario, Integer> implements  
FuncionarioDAO {  
  
    public FuncionarioDAOJPA(EntityManager manager) {  
        super(manager);  
    }  
}
```

12. Após a construção da camada de persistência, é necessária a construção dos managed beans para interagir com as telas da aplicação. Segue o código da classe CargoBean.

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.controller;  
  
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Cargo;  
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.CargoDAOJPA;  
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao.CargoDAO;  
import java.util.List;  
import javax.faces.bean.ManagedBean;  
import javax.faces.context.ExternalContext;  
import javax.faces.context.FacesContext;  
import javax.persistence.EntityManager;  
import javax.servlet.http.HttpServletRequest;
```

```
@ManagedBean
public class CargoBean {

    private Cargo cargo;
    private List<Cargo> listaCargos;

    public CargoBean() {
        cargo = new Cargo();
    }

    public String insere() {
        EntityManager manager = this.getManager();
        CargoDAO dao = new CargoDAOJPA(manager);
        dao.save(cargo);
        this.cargo = new Cargo();
        this.listaCargos = null;
        return "/paginas/lista-cargos.xhtml?faces-redirect=true";
    }

    public String preparaAlteracao() {
        EntityManager manager = this.getManager();
        CargoDAO dao = new CargoDAOJPA(manager);
        this.cargo = dao.getById(Cargo.class, cargo.getCodigo());
        return "/paginas/cargo.xhtml";
    }

    public void remove() {
        EntityManager manager = this.getManager();
        CargoDAO dao = new CargoDAOJPA(manager);
        dao.remove(Cargo.class, cargo.getCodigo());
        this.listaCargos = null;
    }

    private EntityManager getManager() {
        FacesContext fc = FacesContext.getCurrentInstance();
        ExternalContext ec = fc.getExternalContext();
        HttpServletRequest request = (HttpServletRequest) ec.getRequest();
        return (EntityManager) request.getAttribute("EntityManager");
    }

    public Cargo getCargo() {
        return cargo;
    }
}
```

```
}

public void setCargo(Cargo cargo) {
    this.cargo = cargo;
}

public List<Cargo> getListaCargos() {
    if (this.listaCargos == null) {
        EntityManager manager = this.getManager();
        CargoDAO dao = new CargoDAOJPA(manager);
        this.listaCargos = dao.getAll(Cargo.class);
    }
    return listaCargos;
}
}
```

13. Na classe FuncionarioBean exibida a seguir, é importante perceber que é efetuada uma busca na base de dados para que o cargo escolhido seja atribuído ao funcionário. Uma alternativa a essa busca seria a criação de um conversor da classe Cargo.

```
package br.com.rosicleiafrasson.cap13conexaobancofiltro.controller;

import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Cargo;
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.Funcionario;
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.CargoDAOJPA;
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.FuncionarioDAOJPA;
import br.com.rosicleiafrasson.cap13conexaobancodel.persistencia.dao.CargoDAO;
import br.com.rosicleiafrasson.cap13conexaobancofiltro.model.persistencia.dao.FuncionarioDAO;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.persistence.EntityManager;
import javax.servlet.http.HttpServletRequest;

@ManagedBean
public class FuncionarioBean {

    private Funcionario funcionario;
    private List<Funcionario> listaFuncionarios;
    private int cargoId;

    public FuncionarioBean() {
```

```
funcionario = new Funcionario();
}

public String insere() {
    EntityManager manager = this.getManager();
    CargoDAO cargoDao = new CargoDAOJPA(manager);
    FuncionarioDAO dao = new FuncionarioDAOJPA(manager);

    if (this.cargoId != 0) {
        Cargo cargo = cargoDao.getById(Cargo.class, cargoId);
        this.funcionario.setCargo(cargo);
    }

    dao.save(funcionario);
    funcionario = new Funcionario();
    this.listaFuncionarios = null;
    return "/paginas/lista-funcionarios.xhtml?faces-redirect=true";
}

public String preparaAlteracao() {
    EntityManager manager = this.getManager();
    FuncionarioDAO dao = new FuncionarioDAOJPA(manager);
    this.funcionario = dao.getById(Funcionario.class, funcionario.getCodigo());
    return "/paginas/funcionario.xhtml";
}

public void remove() {
    EntityManager manager = this.getManager();
    FuncionarioDAO dao = new FuncionarioDAOJPA(manager);
    dao.remove(Funcionario.class, funcionario.getCodigo());
}

private EntityManager getManager() {
    FacesContext fc = FacesContext.getCurrentInstance();
    ExternalContext ec = fc.getExternalContext();
    HttpServletRequest request = (HttpServletRequest) ec.getRequest();
    return (EntityManager) request.getAttribute("EntityManager");
}

public Funcionario getFuncionario() {
    return funcionario;
}
```

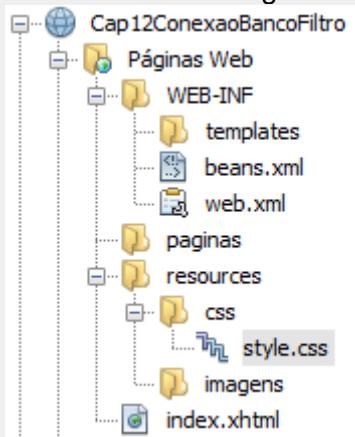
```
public void setFuncionario(Funcionario funcionario) {
    this.funcionario = funcionario;
}

public int getCargoId() {
    return cargoId;
}

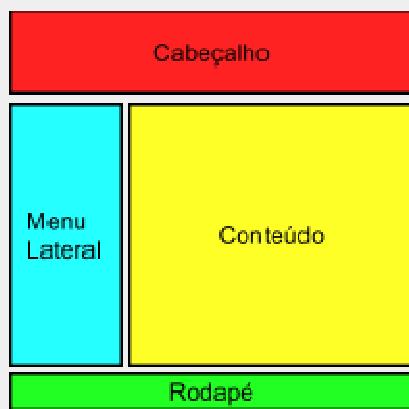
public void setCargoId(int cargoId) {
    this.cargoId = cargoId;
}

public List<Funcionario> getListaFuncionarios() {
    if (this.listaFuncionarios == null) {
        EntityManager manager = this.getManager();
        FuncionarioDAO dao = new FuncionarioDAOJPA(manager);
        this.listaFuncionarios = dao.getAll(Funcionario.class);
    }
    return listaFuncionarios;
}
}
```

14. Dentro da pasta Páginas Web é necessário criar a seguinte estrutura de pastas.



15. A camada de visualização do aplicativo deve ser iniciada com a criação do arquivo template. A estrutura das páginas da aplicação deve ser similar a imagem seguinte.



Sendo assim, o template deve possuir os blocos estáticos cabeçalho, menu lateral e rodapé e o bloco dinâmico conteúdo. Por questões de organização e para facilitar uma posterior manutenção o menu lateral será codificado em uma página distinta e incluso no template. A seguir constam os códigos do template, menu e da página index.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <h:head>
        <title>Gestão de Recursos Humanos</title>
        <h:outputStylesheet library="css" name="style.css"/>
    </h:head>

    <h:body>

        <div id="cabecalho">
            <h1>Gestão de Recursos Humanos</h1>
            <p>Versão 1.0</p>
        </div>

        <div id="pagina">

            <div id="conteudo">
                <ui:insert name="conteudo">
                </ui:insert>
            </div>

            <div id="menuLateral">
                <ui:include src="../menu.xhtml">
            </div>
        
```

```
</ui:include>
</div>

</div>

<div style="clear:both">&nbsp;</div>

<div id="rodape">
    <p>&copy;2014 All Rights Reserved &nbsp;&bull;&nbsp; Rosicleia Frasson</p>
</div>

</h:body>

</html>
```

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
                xmlns ="http://www.w3.org/1999/xhtml"
                xmlns:h="http://xmlns.jcp.org/jsf/html">
```

Menu

```
</ui:composition>
```

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
                template=".//WEB-INF/templates/template.xhtml"
                xmlns ="http://www.w3.org/1999/xhtml">
```

```
<ui:define name="conteudo">
    Página inicial
</ui:define>
```

```
</ui:composition>
```

16. Executando o projeto deve ser exibida uma página similar a imagem a seguir.



17. Para melhorar o aspecto da página serão inclusas algumas regras no arquivo style.css.

```
body{  
    margin: 0;  
    padding: 0;  
    background: #F6F6F6;  
    font-size: 14px;  
    text-align: justify;  
}  
  
body input, textarea, select, option{  
    font-family: Arial, Helvetica, sans-serif;  
    color: #83826A;  
    font-size: 12px;  
}  
  
body input{  
    width: 200px;  
}  
  
h1, h2, h3{  
    font-family: Arial, Helvetica, sans-serif;  
    font-weight: normal;  
    color: #666666;  
}
```

```
h1 {  
    letter-spacing: -2px;  
    font-size: 3em;  
}  
  
p,ul{  
    line-height: 200%;  
}  
  
a{  
    color: #34BA01;  
}  
  
a:hover{  
    color: #34BA01;  
}  
  
#cabecalho{  
    width: 920px;  
    height: 100px;  
    margin: 0 auto;  
}  
  
#cabecalho h1, p{  
    margin: 0;  
    padding: 0;  
}  
  
#cabecalho h1{  
    font-size: 52px;  
    color: #20A001;  
}  
  
#cabecalho p{  
    margin-top: -15px;  
    padding: 0px 0px 0px 4px;  
    font-size: 20px;  
    font-weight: normal;  
    color: #86EA66;  
    text-transform: lowercase;  
}
```

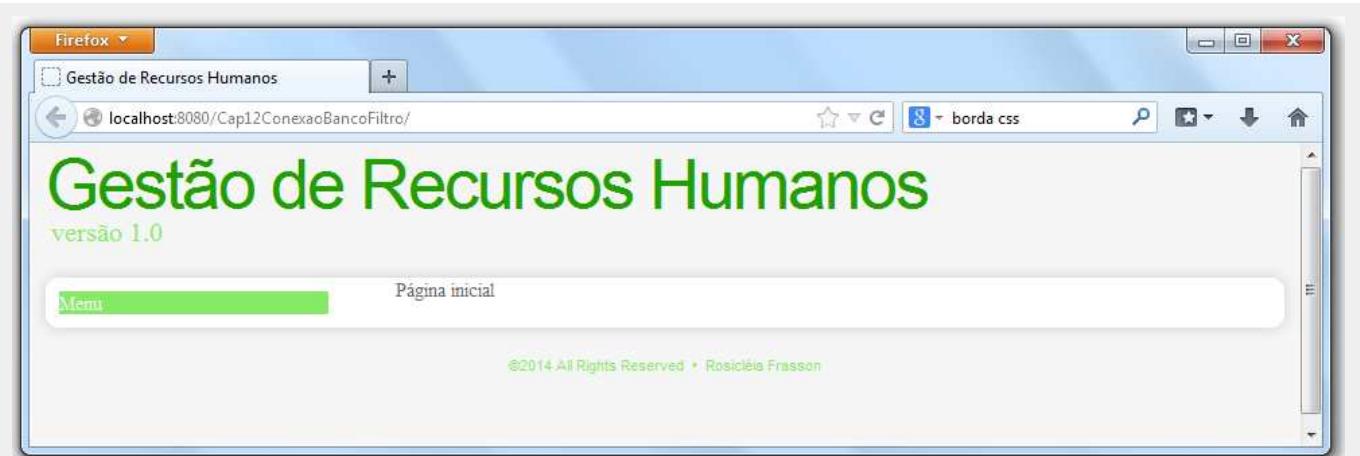
```
#pagina{
    width: 920px;
    margin: 0 auto;
    background: #ffffff;
    border-radius: 10px;
    box-shadow: 0px 0px 10px #cccccc;
    position: relative;
}

#conteudo{
    width: 660px;
    margin: 0;
    color: #666666;
    float: right;
}

#menuLateral{
    background: #86EA66;
    width: 200px;
    color: #FFFFFF;
    border-radius: 10px 0 0 10px;
    border: 10px solid #ffffff;
}

#rodape{
    width: 800px;
    margin: 0 auto;
    height: 50px;
    text-align: center;
    font-size: 11px;
    font-family: Arial, Helvetica, sans-serif;
    color: #86EA66;
    margin-bottom: 27px;
}
```

18. Ao executar a aplicação é possível perceber que o aspecto da página melhorou consideravelmente.



19. O próximo passo é implementar o menu. Seguem os comandos do arquivo menu.xhtml e as regras de estilo aplicadas ao menu.

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:form>

        <ul>

            <li>
                <h2>Cadastros</h2>
                <ul>
                    <li><h:commandLink value="Cargo"
                        action="/faces/paginas/cargo.xhtml"/></li>
                    <li><h:commandLink value="Funcionário"
                        action="/faces/paginas/funcionario.xhtml"/></li>
                </ul>
            </li>

            <li>
                <h2>Listagens</h2>
                <ul>
                    <li><h:commandLink value="Cargo"
                        action="/faces/paginas/lista-cargos.xhtml"/></li>
                    <li><h:commandLink value="Funcionário"
                        action="/faces/paginas/lista-funcionarios.xhtml"/></li>
                </ul>
            </li>

            <li>
```

```
<h2>Relatórios</h2>
<ul>
    <li><h:commandLink value="Cargo" /></li>
    <li><h:commandLink value="Funcionário" /></li>
</ul>
</li>

</ul>

</h:form>

</ui:composition>

#menuLateral ul{
    margin: 10px;
    padding: 0;
    list-style: none;
}

#menuLateral li{
    margin: 0;
    padding: 0;
}

#menuLateral li ul{
    margin-bottom: 40px;
    padding: 0 15px;
}

#menuLateral li ul li{
    margin: 0;
    padding: 0;
    line-height: 35px;
    border-bottom: 1px #A3F389 dashed;
}

#menuLateral h2{
    width: 180px;
    height: 32px;
    padding-left: 10px;
    background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0,
```

```
#1D9B01),color-stop(1, #35BA01));  
background-image: -o-linear-gradient(bottom, #1D9B01 0%, #35BA01 100%);  
background-image: -moz-linear-gradient(bottom, #1D9B01 0%, #35BA01 100%);  
background-image: -webkit-linear-gradient(bottom, #1D9B01 0%, #35BA01 100%);  
background-image: -ms-linear-gradient(bottom, #1D9B01 0%, #35BA01 100%);  
background-image: linear-gradient(to bottom, #1D9B01 0%, #35BA01 100%);  
font-size: 18px;  
color: #FFFFFF;  
}  
  
#menuLateral a{  
text-decoration: none;  
color: #FFFFFF;  
font-weight: bold;  
}  
  
#menuLateral a:hover{  
text-decoration: none;  
}
```

20. Ao executar a aplicação, é possível observar o menu criado.

Gestão de Recursos Humanos

versão 1.0



21. Criado o menu, é necessário construir as páginas correspondentes a manutenção dos cargos e

funcionários do aplicativo. Inicialmente será construído a página referente ao cadastro de cargo. Segue o código correspondente.

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="../WEB-INF/templates/template.xhtml"
    xmlns ="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">

    <ui:define name="conteudo">
        <h:form acceptcharset="ISO-8859-1">

            <h:messages errorStyle="color:red" layout="table"/>

            <h:inputHidden value="#{cargoBean.cargo.codigo}" />

            <fieldset>
                <legend>Cadastro de Cargo</legend>

                <h:panelGrid columns="3">

                    <h:outputLabel value="Nome: " for="nome"/>
                    <h:inputText id="nome"
                        value="#{cargoBean.cargo.nome}" size="63"
                        required="true"
                        requiredMessage="É necessário informar o nome do cargo."/>
                    <h:message for="nome" errorStyle="color:red; display:block"/>

                    <h:outputLabel value="Descrição: " for="descricao"/>
                    <h:inputTextarea id="descricao"
                        value="#{cargoBean.cargo.descricao}"
                        cols="50" rows="10"/>
                    <h:outputLabel/>

                </h:panelGrid>

            </fieldset>

            <br />
            <h:panelGroup>
                <h:commandButton value="Salvar"
                    action="#{cargoBean.insere}" accesskey="S"
                    styleClass="botao"/>
            </h:panelGroup>
        </h:form>
    </ui:define>

```

```

        <h:commandButton value="Cancelar" type="reset" styleClass="botao"/>
    </h:panelGroup>

    </h:form>
</ui:define>

</ui:composition>

```

22. Ao executar a aplicação e acessar o menu cadastro cargo deve ser exibido na área reservada ao conteúdo um formulário similar ao apresentado na figura a seguir.

Cadastro de Cargo

Nome:

Descrição:

23. Com o intuito de melhorar a aparência deste serão inclusas algumas regras CSS.

```

.botao{
    background:-webkit-gradient( linear, left top, left bottom, color-stop(0.05, #108c00),
color-stop(1, #37bf01 ) );
    background:-moz-linear-gradient( center top, #108c00 5%, #37bf01 100% );
    filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#108c00',
endColorstr='#37bf01');
    background-color:#108c00;
    -webkit-border-top-left-radius:6px;
    -moz-border-radius-topleft:6px;
    border-top-left-radius:6px;
    -webkit-border-top-right-radius:6px;
    -moz-border-radius-topright:6px;
    border-top-right-radius:6px;
    -webkit-border-bottom-right-radius:6px;
    -moz-border-radius-bottomright:6px;
    border-bottom-right-radius:6px;
}

```

```
-webkit-border-bottom-left-radius:6px;
-moz-border-radius-bottomleft:6px;
border-bottom-left-radius:6px;
text-indent:0;
border:1px solid #dcdcdc;
display:inline-block;
color:#ffffff;
font-family:arial;
font-size:13px;
font-weight:normal;
font-style:normal;
height:30px;
line-height:50px;
width:80px;
text-decoration:none;
text-align:center;
}

.botao:hover {
    background:-webkit-gradient( linear, left top, left bottom, color-stop(0.05, #37bf01),
color-stop(1, #108c00 ) );
    background:-moz-linear-gradient( center top, #37bf01 5%, #108c00 100% );
    filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#37bf01',
endColorstr='#108c00');
    background-color:#37bf01;

}

.botao:active {
    position:relative;
    top:1px;
}

#conteudo{
    padding: 20px;
}

fieldset{
    border-radius: 5px;
}
```

24. Ao executar a aplicação é possível perceber que o formulário ficou com aspecto melhor.

Cadastro de Cargo

Nome:

Descrição:

Salvar **Cancelar**

25. Neste momento, é possível testar se os dados inseridos no formulário estão sendo persistidos na base de dados.

Cadastro de Cargo

Nome:

Descrição: Desenvolver, implementar, prestar suporte e manutenção em sistemas de informação, assegurando o atendimento às necessidades de usuários no tocante à solução de problemas na área de informática.

Salvar **Cancelar**

	→ T ←	codigo	descricao	nome
<input type="checkbox"/>		1	Desenvolver, implementar, prestar suporte e manutenção em sistemas de informação, assegurando o atendimento às necessidades de usuários no tocante à solução de problemas na área de informática.	Analista de sistemas

26. Além da inserção, é importante que os dados cadastrados possam ser visualizados no aplicativo. Também são necessárias as opções de edição e remoção de dados cadastrados. Neste exemplo, as funcionalidades citadas estão implementadas na página lista-cargos.

No código a seguir merece destaque a tag `f:setPropertyActionListener` que passa a classe `CargoBean` o cargo selecionado.

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="../WEB-INF/templates/template.xhtml"
    xmlns ="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

<ui:define name="conteudo">
    <h:form acceptcharset="ISO-8859-1">

        <h:dataTable value="#{cargoBean.listaCargos}"
            var="cargo"
            styleClass="tabela"
            headerClass="cabecalho"
            cellpadding ="10">
            <f:facet name="header">
                <h:outputText value="Cargos" />
            </f:facet>

            <h:column>
                <f:facet name="header">
                    <h:outputText value="Código" />
                </f:facet>
                <h:outputText value="#{cargo.codigo}"/>
            </h:column>

            <h:column>
                <f:facet name="header">
                    <h:outputText value="Cargo" />
                </f:facet>
                <h:outputText value="#{cargo.nome}"/>
            </h:column>

            <h:column>
                <f:facet name="header">
                    <h:commandLink action="/faces/paginas/cargo.xhtml">
                        <h:graphicImage library="imagens" name="add.png" />
                    </h:commandLink>
                </f:facet>
                <h:commandLink
                    action="#{cargoBean.preparaAlteracao}">
                    <f:setPropertyActionListener value="#{cargo}">
                        target="#{cargoBean.cargo}"/>
                </h:commandLink>
            </h:column>
        </h:dataTable>
    </h:form>
</ui:define>
```

```
<h:graphicImage library="imagens" name="edit.png"/>
</h:commandLink>
<h:commandLink
    action="#{cargoBean.remove}"
    <f:setPropertyActionListener value="#{cargo}"
        target="#{cargoBean.cargo}" />

    <h:graphicImage library="imagens" name="delete.png"/>
</h:commandLink>
</h:column>
</h:dataTable>

</h:form>
</ui:define>

</ui:composition>
```

27. Para melhorar o aspecto da tabela devem ser inclusas algumas imagens - add, delete e edit - e regras CSS para formatação da tabela.

```
.tabela{
    margin:0px;padding:0px;
    width:100%;
    box-shadow: 5px 5px 2px #888888;
    border:1px solid #3f7f00;
}

.tabela table{
    border-collapse: collapse;
    border-spacing: 0;
    width:100%;
    height:100%;
    margin:0px;
    padding:0px;
}

.tabela tr:nth-child(odd){ background-color:#d4ffaa; }
.tabela tr:nth-child(even) { background-color:#ffffff; }
.tabela td{
    vertical-align:middle;
```

```
border:1px solid #3f7f00;
border-width:0px 1px 1px 0px;
text-align:center;
padding:7px;
font-size:12px;
font-family:Arial;
font-weight:normal;
color:#000000;

}.tabela tr:last-child td{
    border-width:0px 1px 0px 0px;
}.tabela tr td:last-child{
    border-width:0px 0px 1px 0px;
}.tabela tr:last-child td:last-child{
    border-width:0px 0px 0px 0px;
}

.tabela .cabecalho{
    background:-o-linear-gradient(bottom, #5fbf00 5%, #3f7f00 100%);    background:-webkit-gradient( linear, left top, left bottom, color-stop(0.05, #5fbf00), color-stop(1, #3f7f00) );
    background:-moz-linear-gradient( center top, #5fbf00 5%, #3f7f00 100% );
    filter:progid:DXImageTransform.Microsoft.gradient(startColorstr="#5fbf00",
endColorstr="#3f7f00");    background: -o-linear-gradient(top,#5fbf00,3f7f00);
    background-color:#5fbf00;
    border:0px solid #3f7f00;
    text-align:center;
    border-width:0px 0px 1px 1px;
    font-size:14px;
    font-family:Arial;
    font-weight:bold;
    color:#ffffff;
}

.tabela tr:first-child td:first-child{
    border-width:0px 0px 1px 0px;
}
.tabela tr:first-child td:last-child{
    border-width:0px 0px 1px 1px;
}
```

28. Ao executar a aplicação é possível verificar os cargos cadastrados e testar a edição e remoção dos dados.

Cargos		
Código	Cargo	
1	Analista de teste	 
2	Analista de sistemas	 
3	Programador	 

29. As telas de cadastro e listagem também devem ser codificadas para o funcionário.

```

<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="../WEB-INF/templates/template.xhtml"
    xmlns ="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

    <ui:define name="conteudo">
        <h:form acceptcharset="ISO-8859-1">

            <h:messages errorStyle="color:red" layout="table"/>

            <h:inputHidden value="#{funcionarioBean.funcionario.codigo}" />

            <fieldset>
                <legend>Cadastro de Funcionário</legend>

                <h:panelGrid columns="2">

                    <h:outputLabel value="Nome: " for="nome"/>
                    <h:panelGroup>
                        <h:inputText id="nome"
                            value="#{funcionarioBean.funcionario.nome}"
                            required="true"
                            requiredMessage="É necessário informar o nome do
                            funcionário." />
                        <h:message for="nome" errorStyle="color:red"/>
                    </h:panelGroup>

                    <h:outputLabel value="CPF: " for="cpf"/>
                
```

```
<h:panelGroup>
    <h:inputText id="cpf"
        value="#{funcionarioBean.funcionario.cpf}"
        required="true"
        requiredMessage="É necessário informar o CPF."/>
    <h:message for="cpf" errorStyle="color:red"/>
</h:panelGroup>

<h:outputLabel value="Data de Nascimento: " for="dtNascimento"/>
<h:panelGroup>
    <h:inputText id="dtNascimento"
        value="#{funcionarioBean.funcionario.dataNascimento}"
        required="true"
        requiredMessage="É necessário informar a data de
nascimento."
        converterMessage="Data no formato inválido. Utilize
dd/mm/aaaa">
        <f:convertDateTime pattern="dd/MM/yyyy" type="date"/>
    </h:inputText>
    <h:message for="dtNascimento" errorStyle="color:red"/>
</h:panelGroup>

<h:outputLabel value="Cargo: " for="cargo"/>
<h:selectOneMenu id="cargo"
    value="#{funcionarioBean.cargoId}">
    <f:selectItems value="#{cargoBean.listaCargos}" var="cargo"
        itemLabel="#{cargo.nome}" itemValue="#{cargo.codigo}"/>
</h:selectOneMenu>

<h:outputLabel value="Telefone: " for="telefone"/>
<h:inputText id="telefone"
    value="#{funcionarioBean.funcionario.telefone}"/>

<h:outputLabel value="Email: " for="email"/>
<h:inputText id="email"
    value="#{funcionarioBean.funcionario.email}"/>
</h:panelGrid>

<hr />
<h:outputText value="Endereço"/>
```

```
<h:panelGrid columns="4">

    <h:outputLabel value="Endereço: " for="endereco"/>
    <h:inputText id="endereco"
value="#{funcionarioBean.funcionario.endereco.endereco}">

        <h:outputLabel value="Número: " for="numero"/>
        <h:panelGroup>
            <h:inputText id="numero"
value="#{funcionarioBean.funcionario.endereco.numero}"
validatorMessage="O número não pode ser negativo.>
                <f:validateLongRange minimum="0"/>
            </h:inputText>
            <h:message for="numero"/>
        </h:panelGroup>

        <h:outputLabel value="Complemento: " for="complemento"/>
        <h:inputText id="complemento"
value="#{funcionarioBean.funcionario.endereco.complemento}">

            <h:outputLabel value="Bairro: " for="bairro"/>
            <h:inputText id="bairro"
value="#{funcionarioBean.funcionario.endereco.bairro}">

            <h:outputLabel value="Estado: " for="estado"/>
            <h:inputText id="estado"
value="#{funcionarioBean.funcionario.endereco.estado}">
            <h:outputLabel value="Cidade: " for="cidade"/>
            <h:inputText id="cidade"
value="#{funcionarioBean.funcionario.endereco.cidade}">
        </h:panelGrid>

    </fieldset>

    <br />
    <h:panelGroup>
        <h:commandButton value="Salvar"
action="#{funcionarioBean.insere}"
styleClass="botao"/>
        <h:commandButton value="Cancelar" type="reset" styleClass="botao"/>
    </h:panelGroup>
```

```
</h:panelGroup>

</h:form>
</ui:define>

</ui:composition>
```

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="../WEB-INF/templates/template.xhtml"
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

    <ui:define name="conteudo">
        <h:form acceptcharset="ISO-8859-1">
            <h:dataTable value="#{funcionarioBean.listaFuncionarios}"
                var="funcionario"
                styleClass="tabela"
                headerClass="cabecalho">
                <f:facet name="header">
                    <h:outputText value="Funcionários"/>
                </f:facet>

                <h:column>
                    <f:facet name="header">
                        <h:outputText value="Código" />
                    </f:facet>
                    <h:outputText value="#{funcionario.codigo}"/>
                </h:column>

                <h:column>
                    <f:facet name="header">
                        <h:outputText value="Nome" />
                    </f:facet>
                    <h:outputText value="#{funcionario.nome}"/>
                </h:column>

                <h:column>
                    <f:facet name="header">
                        <h:outputText value="Data de nascimento" />
                    </f:facet>
```

```
<h:outputText value="#{funcionario.dataNascimento}">
    <f:convertDateTime pattern="dd/MM/yyyy" type="date"/>
</h:outputText>

</h:column>

<h:column>
    <f:facet name="header">
        <h:outputText value="Cargo" />
    </f:facet>
    <h:outputText value="#{funcionario.cargo.nome}" />
</h:column>

<h:column>
    <f:facet name="header">
        <h:commandLink action="/faces/paginas/funcionario.xhtml">
            <h:graphicImage library="imagens" name="add.png" />
        </h:commandLink>
    </f:facet>
    <h:commandLink
        action="#{funcionarioBean.preparaAlteracao}">
        <f:setPropertyActionListener value="#{funcionario}">
            target="#{funcionarioBean.funcionario}" />

        <h:graphicImage library="imagens" name="edit.png" />
    </h:commandLink>
    <h:commandLink
        action="#{funcionarioBean.remove}">
        <f:setPropertyActionListener value="#{funcionario}">
            target="#{funcionarioBean.funcionario}" />

        <h:graphicImage library="imagens" name="delete.png" />
    </h:commandLink>
</h:column>
</h:dataTable>
</h:form>
</ui:define>
</ui:composition>
```

30. A aparência das telas responsáveis pela manutenção dos funcionários devem ser similares as imagens a seguir.

Cadastro de Funcionário

Nome:	<input type="text"/>
CPF:	<input type="text"/>
Data de Nascimento:	<input type="text"/>
Cargo:	<input type="text" value="Analista de teste"/> <input type="button" value="▼"/>
Telefone:	<input type="text"/>
Email:	<input type="text"/>

Endereço

Endereço:	<input type="text"/>	Número:	<input type="text" value="0"/>
Complemento:	<input type="text"/>	Bairro:	<input type="text"/>
Estado:	<input type="text"/>	Cidade:	<input type="text"/>

Funcionários

Código	Nome	Data de nascimento	Cargo	+
1	Ana Luiza Valdezia	28/08/1990	Analista de sistemas	
3	João de Souza	09/10/2010	Analista de teste	

PASSO-A-PASSO

Mecanismo de Autenticação

Em aplicações comerciais é necessário que o aplicativo possua um mecanismo de autenticação de usuários. Com JSF existem diversas maneiras de efetuar a autenticação. No exemplo apresentado a seguir é utilizado o Filter para este controle.

1. Algumas alterações no projeto criado anteriormente devem ser efetuadas. Primeiramente é necessária a inserção dos campos login e senha na classe Funcionario.

```
package br.com.rosicleiafrasson.cap13autenticacaofiltro.model;

import java.util.Date;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

@Entity
public class Funcionario {

    @Id
    @GeneratedValue
    private int codigo;
    private String nome;
    private String cpf;
    @Temporal(TemporalType.DATE)
    private Date dataNascimento;
    @ManyToOne
    private Cargo cargo;
    private String telefone;
    private String email;
    @OneToOne(cascade = CascadeType.ALL)
    private Endereco endereco;
    @Column(unique = true)
    private String login;
    private String senha;

    public Funcionario() {
        endereco = new Endereco();
    }
    //Gets e sets
}
```

2. Os campos login e senha devem ser inclusos na página de cadastro de funcionários.

```
<hr />
<h:panelGroup id="dadosLogin">
    <h:panelGrid columns="4">
        <h:outputLabel for="login" value="Login: "/>
        <h:inputText id="login" value="#{funcionarioBean.funcionario.login}" />

        <h:outputLabel for="senha" value="Senha: "/>
        <h:inputText id="senha" value="#{funcionarioBean.funcionario.senha}"/>
    </h:panelGrid>
```

```
</h:panelGroup>
```

3. Um filtro para o controle de acesso deve ser criado.

```
package br.com.rosicleiafrasson.cap13autenticacaofiltro.filter;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class ControleAcessoFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        try {
            HttpServletRequest httpReq = (HttpServletRequest) request;
            HttpServletResponse httpRes = (HttpServletResponse) response;
            HttpSession session = httpReq.getSession(true);
            String url = httpReq.getRequestURL().toString();
            if (session.getAttribute("usuarioLogado") == null && precisaAutenticar(url)) {
                httpRes.sendRedirect(httpReq.getContextPath() + "/faces/login.xhtml");
            } else {
                chain.doFilter(request, response);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void destroy() {
```

```
}

private boolean precisaAutenticar(String url) {
    return !url.contains("login.xhtml") && !url.contains("javax.faces.resource");
}
}
```

4. Na existência de mais de um filtro em uma aplicação, é recomendado que o registro dos filtros seja feito no web.xml ao invés de utilizar anotações.

```
<filter>
    <filter-name>ControleAcesso</filter-name>
    <filter-class>br.com.rosicleiafrasson.rh.filter.ControleAcesso</filter-class>
</filter>
<filter-mapping>
    <filter-name>ControleAcesso</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
<filter>
    <filter-name>JPAPFilter</filter-name>
    <filter-class>br.com.rosicleiafrasson.rh.filter.JPAPFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>JPAPFilter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
```

5. Para verificar se o usuário que está tentando se autenticar está cadastrado, é necessário efetuar uma busca na base de dados. Deste modo, as classes FuncionarioDAO e FuncionarioDAOJPA devem ser alteradas.

```
package br.com.rosicleiafrasson.cap13autenticacaofiltro.model.persistencia.dao;

import br.com.rosicleiafrasson.cap13autenticacaofiltro.model.Funcionario;

public interface FuncionarioDAO extends DAO<Funcionario, Integer> {

    boolean login(String usuario, String senha);
}
```

```
package br.com.rosicleiafrasson.cap13autenticacaofiltro.model.persistencia;

import br.com.rosicleiafrasson.cap13autenticacaofiltro.model.Funcionario;
import br.com.rosicleiafrasson.cap13autenticacaofiltro.model.persistencia.dao.FuncionarioDAO;
```

```
import javax.persistence.EntityManager;
import javax.persistence.Query;

public class FuncionarioDAOJPA extends DAOJPA<Funcionario, Integer> implements FuncionarioDAO {

    private EntityManager manager;

    public FuncionarioDAOJPA(EntityManager manager) {
        super(manager);
        this.manager = manager;
    }

    @Override
    public boolean login(String login, String senha) {
        Query q = this.manager.createQuery("select f from Funcionario f where f.login = :login and f.senha = :senha", Funcionario.class);
        q.setParameter("senha", senha);
        q.setParameter("login", login);
        if (q.getResultList().size() > 0) {
            return true;
        }
        return false;
    }
}
```

6. O passo seguinte é a construção de um managed bean para controlar a autenticação do usuário.

```
package br.com.rosicleiafrasson.cap13autenticacaofiltro.controller;

import br.com.rosicleiafrasson.cap13autenticacaofiltro.model.persistencia.FuncionarioDAOJPA;
import
br.com.rosicleiafrasson.cap13autenticacaofiltro.model.persistencia.dao.FuncionarioDAO;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.persistence.EntityManager;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

@ManagedBean
@SessionScoped
```

```
public class LoginBean {

    private String usuario;
    private String senha;

    public String autentica() {
        FacesContext fc = FacesContext.getCurrentInstance();
        EntityManager manager = this.getManager();
        FuncionarioDAO dao = new FuncionarioDAOJPA(manager);
        if (dao.login(usuario, senha)) {
            ExternalContext ec = fc.getExternalContext();
            HttpSession session = (HttpSession) ec.getSession(false);
            session.setAttribute("usuarioLogado", true);
            return "/paginas/cargo.xhtml";
        } else {
            FacesMessage fm = new FacesMessage("Usuario e/ou senha inválidos");
            fm.setSeverity(FacesMessage.SEVERITY_ERROR);
            fc.addMessage(null, fm);
            return "/login";
        }
    }

    public String registraSaida() {
        FacesContext fc = FacesContext.getCurrentInstance();
        ExternalContext ec = fc.getExternalContext();
        HttpSession session = (HttpSession) ec.getSession(false);
        session.removeAttribute("usuario");
        return "/login";
    }

    private EntityManager getManager() {
        FacesContext fc = FacesContext.getCurrentInstance();
        ExternalContext ec = fc.getExternalContext();
        HttpServletRequest request = (HttpServletRequest) ec.getRequest();
        return (EntityManager) request.getAttribute("EntityManager");
    }

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
```

```
    this.usuario = usuario;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}
}
```

7. Para que o usuário possa efetuar a autenticação é necessária a página de login, exibida a seguir.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:pt="http://xmlns.jcp.org/jsf/passthrough">

<h:head>
    <title>Gestão de Recursos Humanos</title>
    <h:outputStylesheet library="css" name="style.css"/>
</h:head>

<h:body>

    <div id="paginaLogin">
        <h2>LOGIN </h2>
        <h:form>
            <h:panelGrid columns="1">

                <h:inputText value="#{loginBean.usuario}" id="usuario"
pt:placeholder="Usuário" />
                <br />

                <h:inputSecret value="#{loginBean.senha}" id="senha"
pt:placeholder="Senha"/>

                <br />

```

```
<h:commandButton value="OK" action="#{loginBean.autentica}"  
styleClass="botaoLogin"/>  
</h:panelGrid>  
  
</h:form>  
  
<h:messages />  
</div>  
  
</h:body>  
</html>
```

8. Com o intuito de melhorar a aparência da página de login, devem ser aplicadas algumas regras CSS.

```
#paginaLogin{  
    position:absolute;  
    left:50%;  
    top:50%;  
    margin-left:-105px;  
    margin-top:-100px;  
    width: 210px;  
    background: #ffffff;  
    border-radius: 10px;  
    box-shadow: 0px 0px 10px #cccccc;  
    padding: 20px;  
}  
  
#paginaLogin .botaoLogin{  
    font:bold 18px Arial, Helvetica, sans-serif;  
    font-style:normal;  
    color:#ffffff;  
    background:#5e5e54;  
    border:0px solid #ffffff;  
    text-shadow:-1px -1px 1px #222222;  
    box-shadow:2px 2px 5px #000000;  
    -moz-box-shadow:2px 2px 5px #000000;  
    -webkit-box-shadow:2px 2px 5px #000000;  
    border-radius:0px 10px 10px 10px;  
    -moz-border-radius:0px 10px 10px 10px;  
    -webkit-border-radius:0px 10px 10px 10px;  
    width:77px;  
    height: 40px;
```

```
padding:10px 20px;  
cursor:pointer;  
margin:0 auto;  
}  
  
#paginaLogin input{  
    height: 20px;  
}
```

9. É importante acrescentar no template um link para que o usuário possa fazer o logout. Dessa forma, dentro do cabeçalho deve existir a div seção exibida a seguir.

```
<div id="secao">  
    <h:form>  
        <h:outputFormat value="Olá {0}">  
            <f:param value="#{loginBean.usuario}" />  
        </h:outputFormat>  
        <br />  
        <h:commandLink value="Logout" action="#{loginBean.registraSaida}" />  
    </h:form>  
</div>
```

10. Para alinhar corretamente a div seção algumas regras CSS.

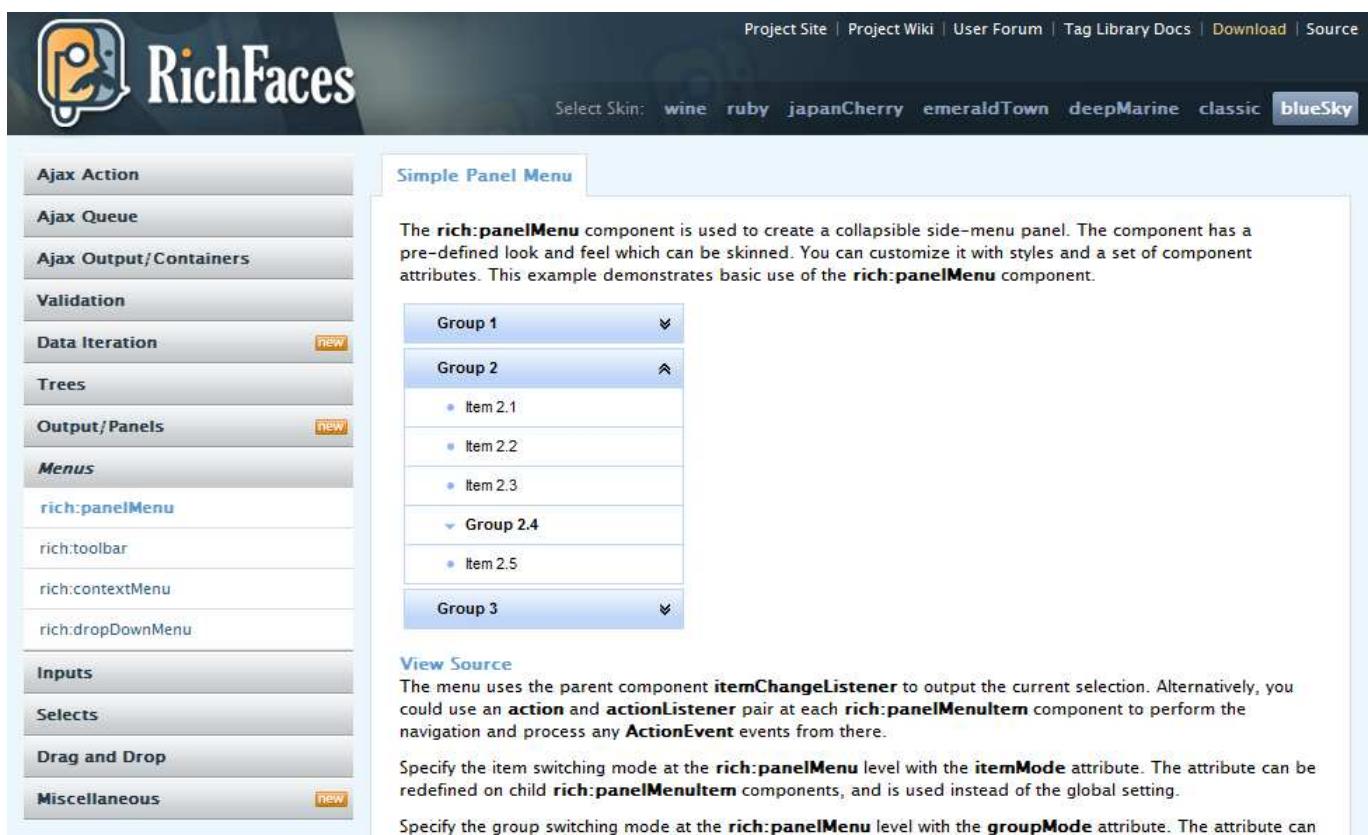
```
#cabecalho #secao{  
    margin-top: -20px;  
    float: right;  
}  
  
#cabecalho #secao a{  
    text-decoration: none;  
}  
  
#cabecalho #secao a:hover{  
    text-decoration: underline;  
}
```

Bibliotecas de componentes

O JSF possui em sua implementação padrão todos os componentes básicos da HTML. Em muitos casos esses componentes são suficientes, embora a tecnologia permite que os desenvolvedores criem seus próprios componentes que podem ser utilizados em diversos projetos. O uso de componentes prontos melhora a qualidade visual da aplicação além de aumentar a produtividade.

Devido a grande demanda por componentes mais robustos surgiram diversas bibliotecas prontas para uso em projetos JSF. Entre as bibliotecas mais utilizadas pelos desenvolvedores podem ser citadas: ADF Faces, IceFaces, Primefaces e RichFaces.

RichFaces



The screenshot shows the RichFaces showcase website. On the left, there is a sidebar with a navigation menu:

- Ajax Action
- Ajax Queue
- Ajax Output/Containers
- Validation
- Data Iteration new
- Trees
- Output/Panels new
- Menus**
- rich:panelMenu**
- rich:toolbar
- rich:contextMenu
- rich:dropDownMenu
- Inputs
- Selects
- Drag and Drop
- Miscellaneous new

The main content area has a title "Simple Panel Menu". Below it, a text block explains the **rich:panelMenu** component. It says: "The **rich:panelMenu** component is used to create a collapsible side-menu panel. The component has a pre-defined look and feel which can be skinned. You can customize it with styles and a set of component attributes. This example demonstrates basic use of the **rich:panelMenu** component." Below the text is a visual representation of the menu, which consists of three groups: Group 1, Group 2, and Group 3. Group 2 is expanded, showing five items: Item 2.1, Item 2.2, Item 2.3, Group 2.4, and Item 2.5. Group 1 and Group 3 are collapsed, indicated by arrows.

Below the menu, there are two sections with links:

- View Source** [View Source](#)
- Specify the item switching mode at the rich:panelMenu level with the itemMode attribute.** [Specify the item switching mode at the rich:panelMenu level with the itemMode attribute.](#)
- Specify the group switching mode at the rich:panelMenu level with the groupMode attribute.** [Specify the group switching mode at the rich:panelMenu level with the groupMode attribute.](#)

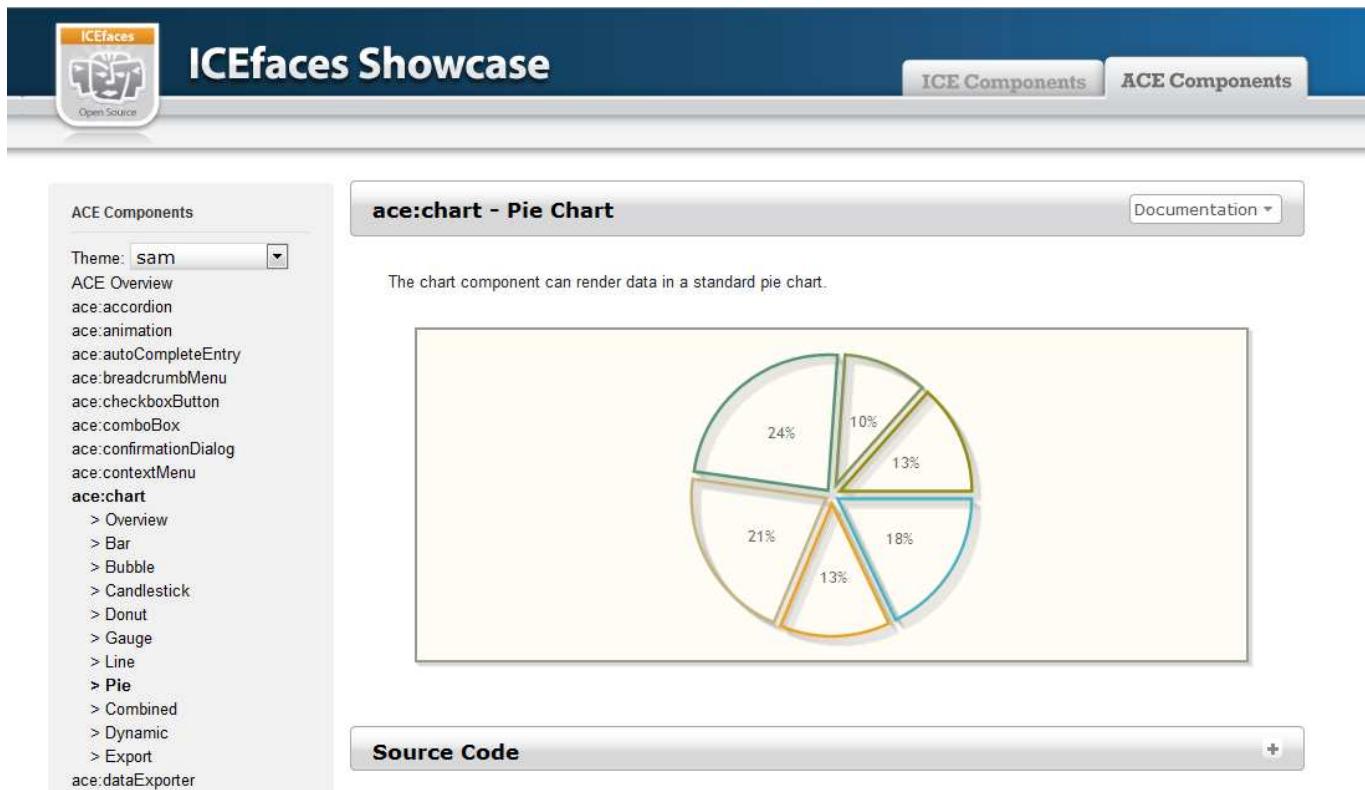
O RichFaces é uma biblioteca de responsabilidade da JBoss que pertence ao grupo Red Hat. Possui código aberto, suporte a ajax e em torno de 60 componentes prontos para utilização. O desenvolvedor também pode criar seus próprios componentes. No showcase, os componentes são exibidos juntamente com o código necessário para a sua utilização.

Dentre os componentes mais importantes podem ser citados calendário, auto complete, tabela com paginação e o menu dropdown.

O esquema de cores apresentadas pelos componentes da biblioteca pode ser alterado com a utilização de temas. Atualmente no showcase do primefaces existem 7 temas definidos.

A documentação do framework é bastante diversificada contendo além do guia do desenvolvedor, FAQ e screencasts. Também contém um fórum para esclarecimento de dúvidas.

Icefaces



The screenshot shows the ICEfaces Showcase website. At the top, there's a navigation bar with the ICEfaces logo, the text "ICEfaces Showcase", and buttons for "ICE Components" and "ACE Components". On the left, there's a sidebar titled "ACE Components" with a dropdown menu set to "Theme: sam". The menu lists various components: ACE Overview, ace:accordion, ace:animation, ace:autoCompleteEntry, ace:breadcrumbMenu, ace:checkboxButton, ace:comboBox, ace:confirmationDialog, ace:contextMenu, ace:chart (with sub-options like Overview, Bar, Bubble, Candlestick, Donut, Gauge, Line, Pie, Combined, Dynamic, Export, ace:dataExporter). In the center, there's a main content area with a title "ace:chart - Pie Chart" and a sub-instruction "The chart component can render data in a standard pie chart." Below this is a pie chart divided into six segments with percentages: 24%, 10%, 13%, 18%, 13%, and 21%. At the bottom of the main content area is a "Source Code" button.

O Icefaces é uma biblioteca de componentes open source desenvolvida pela IceSoft (<http://www.icesoft.org/java/projects/ICEfaces/overview.jsf>). Possui como finalidade a integração entre as tecnologias JSF e ajax de forma nativa. Sendo assim, todos os componentes do icefaces dão suporte a ajax.

Possui em torno de 40 componentes dentre os quais podem ser citados tabela com paginação, calendário, campos de texto com máscara, barra de progresso e geração de gráficos estatísticos. O show case pode ser acessado em: <http://icefaces-showcase.icesoft.org> e possui a descrição do componente, ilustração, código fonte para utilização e um link para a documentação.

A documentação é extensa contendo tutoriais, vídeos e documentação no formato wiki. Além de disponibilizar fórum e chat para apoio.

Primefaces

O primefaces é desenvolvido pela Prime Teknoloji (Turquia) e pode ser acessado em: <http://www.primefaces.org>. É um framework open source. Possui suporte nativo a ajax e em torno de 150 componentes em seu showcase, sendo considerada a biblioteca de componentes mais avançada do mercado.

O framework provê uma versão melhorada de praticamente todos os componentes JSF - entrada e saída de textos, menus de seleção e botões. Além disso, o primefaces possui uma ampla gama de recursos diferenciados como captcha, gerador de gráficos estatísticos, galeria de imagens, barra de progresso, entre outros. Um detalhe importante é que no showcase todos os componentes são apresentados juntamente com as linhas de código necessárias para a sua utilização. O primefaces possui também cerca de 30 temas pré-definidos que alteram a aparência dos componentes. O framework ainda disponibiliza uma versão para dispositivos móveis.

A documentação do primefaces é de fácil acesso contendo a demonstração de todos os componentes. O framework conta ainda com fórum e extensa comunidade com ativa participação e colaboração.

Uma grande vantagem da utilização do primefaces é a sua facilidade de configuração. Para utilizá-lo apenas um único arquivo jar é necessário. Não é necessário nenhum arquivo xml de configuração e não há dependências.

PRIME FACES

aristo

Components Mobile Push Ext Mock OS X

Ajax Core

Basics
Partial Processing
Validation
Fragment

Selectors
Polling
RemoteCommand
AjaxStatus
SearchExpressions

Events
Selects
Listeners
Counter

Input

AutoComplete
BooleanButton
BoolCheckbox
Calendar
CheckboxMenu
ColorPicker
Editor
Inplace

InputMask
InputText
InputTextarea
Keyboard
ManyButton
ManyMenu
ManyCheckbox
MultiSelectListbox

OneButton
OneMenu
OneListbox
OneRadio
Password
Rating
Spinner
Slider

Button

Button CommandButton CommandLink SplitButton

Data

Carousel
DataExporter
DataList
DataGrid
DataTable

Mindmap
PickList
OrderList
Ring
Schedule

Tree
Horizontal
TreeTable

Galleria

Galleria is a content gallery component featuring various customization options.

Images



galleria5.jpg

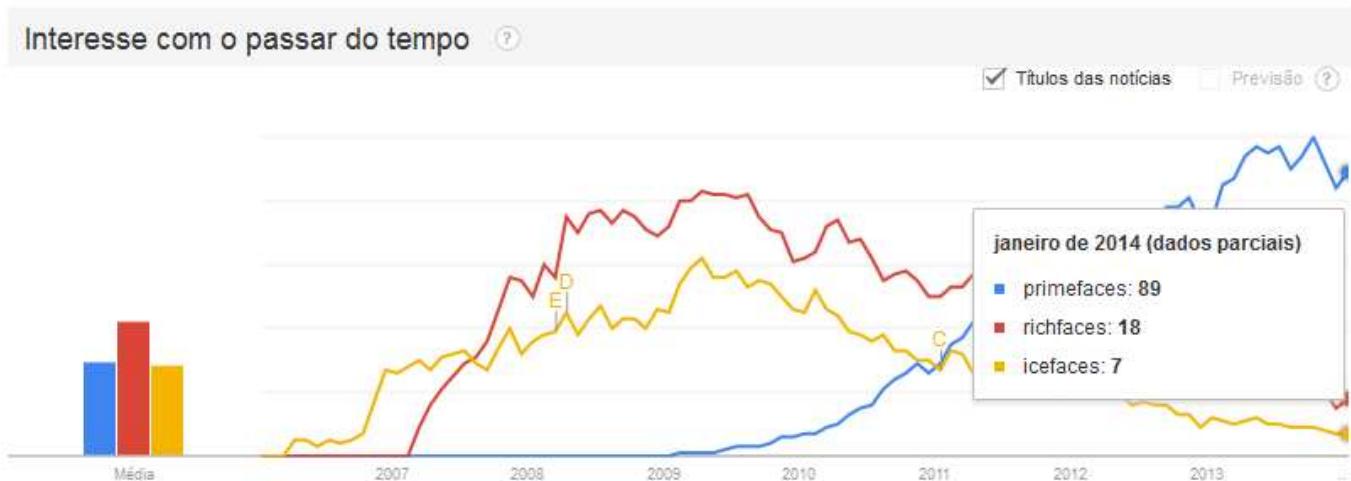
Image Description for galleria5.jpg



Custom Content

Primefaces

Como descrito no capítulo anterior, o primefaces atualmente é uma boa escolha para uma biblioteca de componentes. Embora dentre os frameworks de componentes utilizados, o primefaces seja o mais recente, a sua utilização cresceu drasticamente nos últimos anos. A seguir está ilustrada uma pesquisa feita por meio do Google Trends que indica a popularidade do framework.



Pelos motivos citados, o primefaces será a biblioteca de componentes utilizada neste material.

Componentes

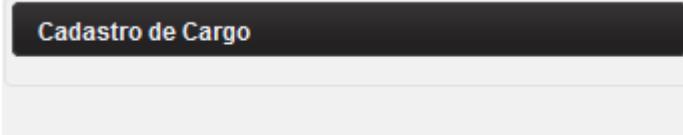
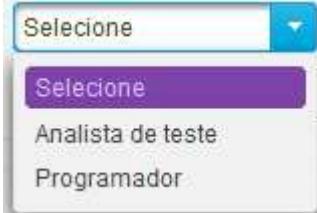
O rico conjunto de componentes presente na biblioteca do primefaces é um dos principais motivos de sua ascensão. Em virtude da quantidade de componentes nem todos serão citados neste material. Porém, todo o conjunto pode ser consultado no showcase da biblioteca: <http://www.primefaces.org/showcase/ui/home.jsf>.

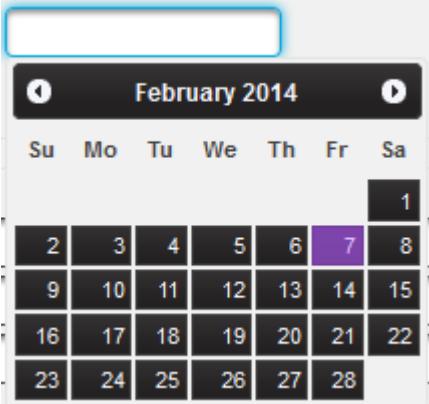
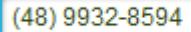
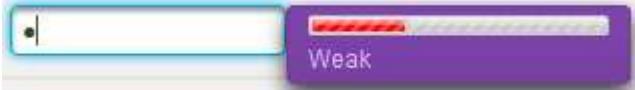
Alguns componentes citados existem no JSF puro. No entanto, utilizando os componentes do primefaces, podem ser aproveitados os conjuntos de skins que a biblioteca oferece. Para utilizar os componentes do primefaces é necessário incluir o namespace:

```
xmlns:p="http://primefaces.org/ui">
```

A seguir, alguns componentes do primefaces:

Componente	Descrição
p:panel	Painel onde podem ser alocados outros componentes. O painel pode conter um cabeçalho. <pre><p:panel header="Cadastro de Cargo"> </p:panel></pre>

	
p:inputText	Campo de entrada de dados com apenas uma linha. <pre><p:inputText id="nome" size="50" value="#{cargoBean.cargo.nome}" required="true" requiredMessage="É necessário informar o nome do cargo."/></pre> 
p:selectOneMenu	Componente conhecido como combo box. <pre><p:selectOneMenu id="cargo" value="#{funcionarioBean.cargoId}" style="width:150px;"> <f:selectItem itemLabel="Selecione" itemValue="" /> <f:selectItems value="#{cargoBean.listaCargos}" var="cargo" itemLabel="#{cargo.nome}" itemValue="#{cargo.codigo}"/> </p:selectOneMenu></pre> 
p:calendar	Componente de entrada usado para selecionar uma data. <pre><p:calendar id="dtNascimento" value="#{funcionarioBean.funcionario.dataNascimento}" required="true" requiredMessage="É necessário informar a data de nascimento." converterMessage="Data no formato inválido. Utilize dd/mm/aaaa" pattern="dd/MM/yyyy"> <f:convertDateTime pattern="dd/MM/yyyy" type="date"/> </p:calendar></pre>

	
p:inputMask	<p>Campo de entrada de texto que permite a adição de uma máscara.</p> <pre><p:inputMask id="telefone" value="#{funcionarioBean.funcionario.telefone}" mask="(99) 9999-9999"/></pre> 
p:inputTextarea	<p>Campo de entrada de texto composto por mais de uma linha.</p> <pre><p:inputTextarea id="descricao" value="#{cargoBean.cargo.descricao}" cols="50" rows="10"/></pre> 
p:password	<p>Campo de senha.</p> <pre><p:password id="senha" value="#{funcionarioBean.funcionario.senha}" feedback="true"/></pre> 
p:commandButton	<p>Botão utilizado para envio de dados em um formulário.</p> <pre><p:commandButton action="#{cargoBean.remove}" icon="ui-icon-close"> <f:setPropertyActionListener value="#{cargo}" target="#{cargoBean.cargo}" /></pre>

	<pre></p:commandButton></pre> 
p:dataTable	Tabela. Neste componente podem ser incluídas informações como quantidade de linhas, paginação, ordenação de elementos, pesquisa, entre outros. <pre><p:dataTable id="tabela" var="cargo" value="#{cargoBean.listaCargos}"> <f:facet name="header"> Cargos </f:facet> <p:column sortBy="codigo" headerText="Código"> <h:outputText value="#{cargo.codigo}" /> </p:column> <p:column sortBy="nome" headerText="Nome"> <h:outputText value="#{cargo.nome}" /> </p:column> <p:column> <f:facet name="header"> <p:commandButton action="/faces/paginas/cargo.xhtml" icon="ui-icon-plus"> </p:commandButton> </f:facet> <p:commandButton action="#{cargoBean.preparaAlteracao}" icon="ui-icon-pencil"> <f:setPropertyActionListener value="#{cargo}" target="#{cargoBean.cargo}" /> </p:commandButton> <p:commandButton action="#{cargoBean.remove}" icon="ui-icon-close"> <f:setPropertyActionListener value="#{cargo}" target="#{cargoBean.cargo}" /> </p:commandButton> </p:column> </p:dataTable></pre>

```
</p:commandButton>  
</p:column>  
</p:dataTable>
```

Cargos		
Código	Nome	
1	Analista de teste	
3	Programador	

p:layout

Componente utilizado para a montagem de layouts. O componente layoutUnit permite a divisão da página em blocos.

```
<p:layout fullPage="true">  
  
    <p:layoutUnit position="north" size="100" header="Gestão de  
    Recursos Humanos" >  
  
        <h:graphicImage library="imagens" name="logo.png" />  
  
    </p:layoutUnit>  
  
    <p:layoutUnit position="south" size="35" header="&copy;2014  
    Todos os direitos reservados &nbsp;&bull;&nbsp; Rosicleia  
    Frasson" style="text-align: center">  
  
    </p:layoutUnit>  
  
    <p:layoutUnit position="west" size="200" resizable="true"  
    closable="true" collapsible="true">  
        <ui:include src="../menu.xhtml"/>  
    </p:layoutUnit>  
  
    <p:layoutUnit position="center">  
        <div id="conteudo">  
            <ui:insert name="conteudo">  
                </ui:insert>  
            </div>  
    </p:layoutUnit>
```

The screenshot shows a web application interface. At the top, there is a light blue header bar containing the code snippet: </p:layout>. Below this is a dark header bar with the text "Gestão de Recursos Humanos". Underneath the header is a logo for "RH Solutions" featuring a green leaf icon. To the right of the logo is a sidebar menu with the following items: "Administração", "Recrutamento e Seleção", "Serviços", and "Relatórios Gerenciais". To the right of the sidebar, the text "Página inicial" is displayed. At the bottom of the page is a dark footer bar with the copyright notice "©2014 Todos os direitos reservados • Rosicléia Frasson".

Icons

O jquery possui uma série de ícones que podem ser utilizados com o primefaces. Esses ícones podem ser obtidos em: <http://jqueryui.com/themeroller/>.



É interessante optar por estes ícones já que a aparência dos mesmos é alterada conforme o tema da página.

Skins

O primefaces possui integração com o framework ThemeRoller do JQuery. Atualmente existem 24 temas disponíveis para serem utilizados nas aplicações.



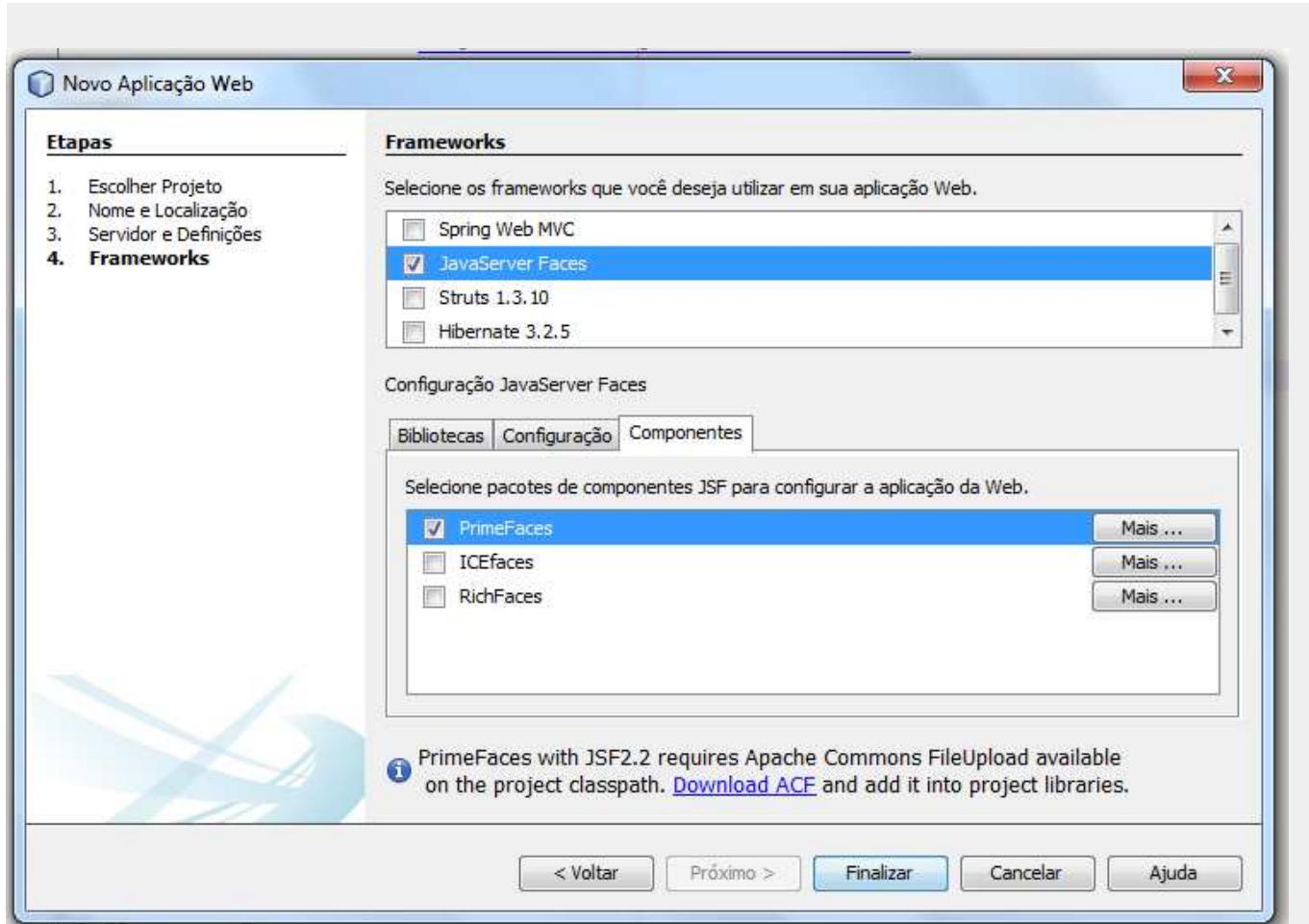
Além dos temas existentes na página do primefaces, é possível criar novos temas na página: <http://jqueryui.com/themeroller/>.



PASSO-A-PASSO

Configuração

1. Para utilizar o primefaces é necessário efetuar o download da biblioteca no site do projeto: <http://primefaces.org/downloads>. É interessante fazer o download da versão 4.0.
2. O projeto é criado da mesma forma que um projeto JSF puro. No entanto, na aba frameworks é necessário apontar como framework de componentes o primefaces. Vale lembrar que durante a criação do primeiro projeto utilizando o primefaces é necessário acionar o botão Mais e apontar para a biblioteca que foi efetuado o download.



3. É importante perceber que o Netbeans cria automaticamente as páginas index e welcomePrimefaces. O arquivo welcomePrimefaces possui algumas tags da biblioteca.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">

    <f:view contentType="text/html">
        <h:head>
            <f:facet name="first">
                <meta content='text/html; charset=UTF-8' http-equiv="Content-Type"/>
                <title>PrimeFaces</title>
            </f:facet>
        </h:head>

        <h:body>
```

```
<p:layout fullPage="true">

    <p:layoutUnit position="north" size="100" resizable="true" closable="true"
collapsible="true">
        Header
    </p:layoutUnit>

    <p:layoutUnit position="south" size="100" closable="true" collapsible="true">
        Footer
    </p:layoutUnit>

    <p:layoutUnit position="west" size="175" header="Left" collapsible="true">
        <p:menu>
            <p:submenu label="Resources">
                <p:menuitem value="Demo" url="http://www.primefaces.org/showcase-
labs/ui/home.jsf" />
                <p:menuitem value="Documentation"
url="http://www.primefaces.org/documentation.html" />
                <p:menuitem value="Forum" url="http://forum.primefaces.org/" />
                <p:menuitem value="Themes" url="http://www.primefaces.org/themes.html" />
            </p:submenu>
        </p:menu>
    </p:layoutUnit>

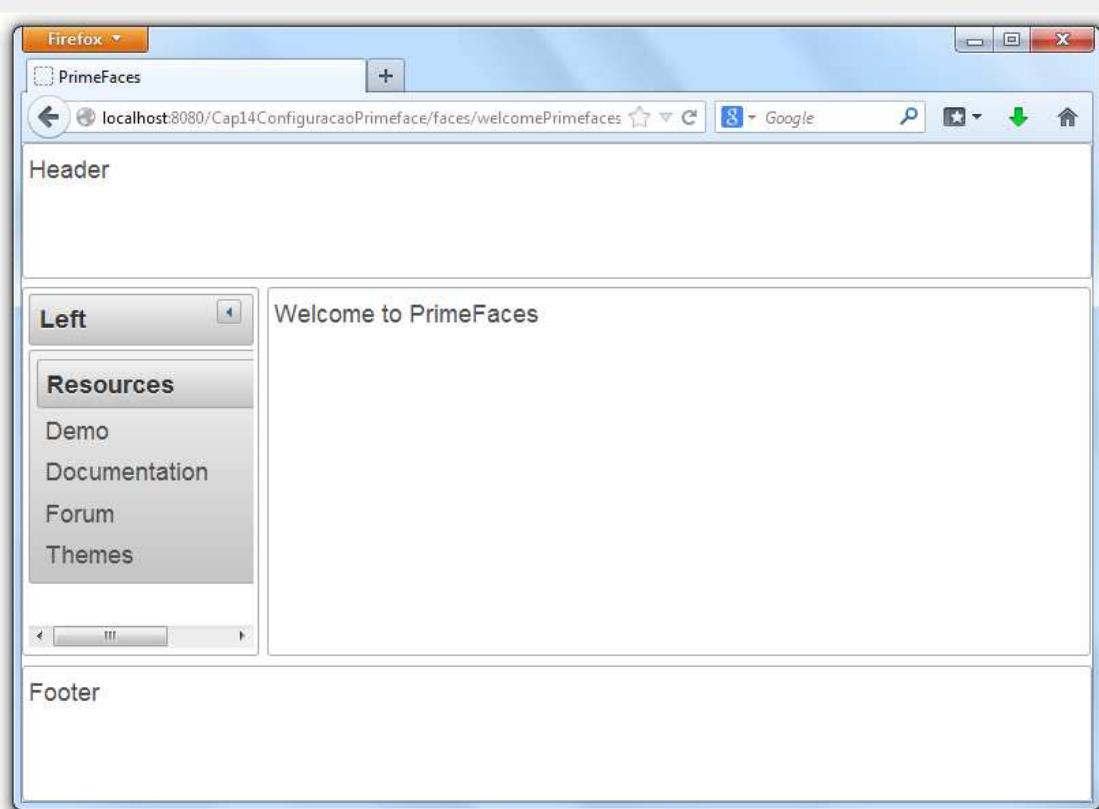
    <p:layoutUnit position="center">
        Welcome to PrimeFaces
    </p:layoutUnit>

</p:layout>

</h:body>

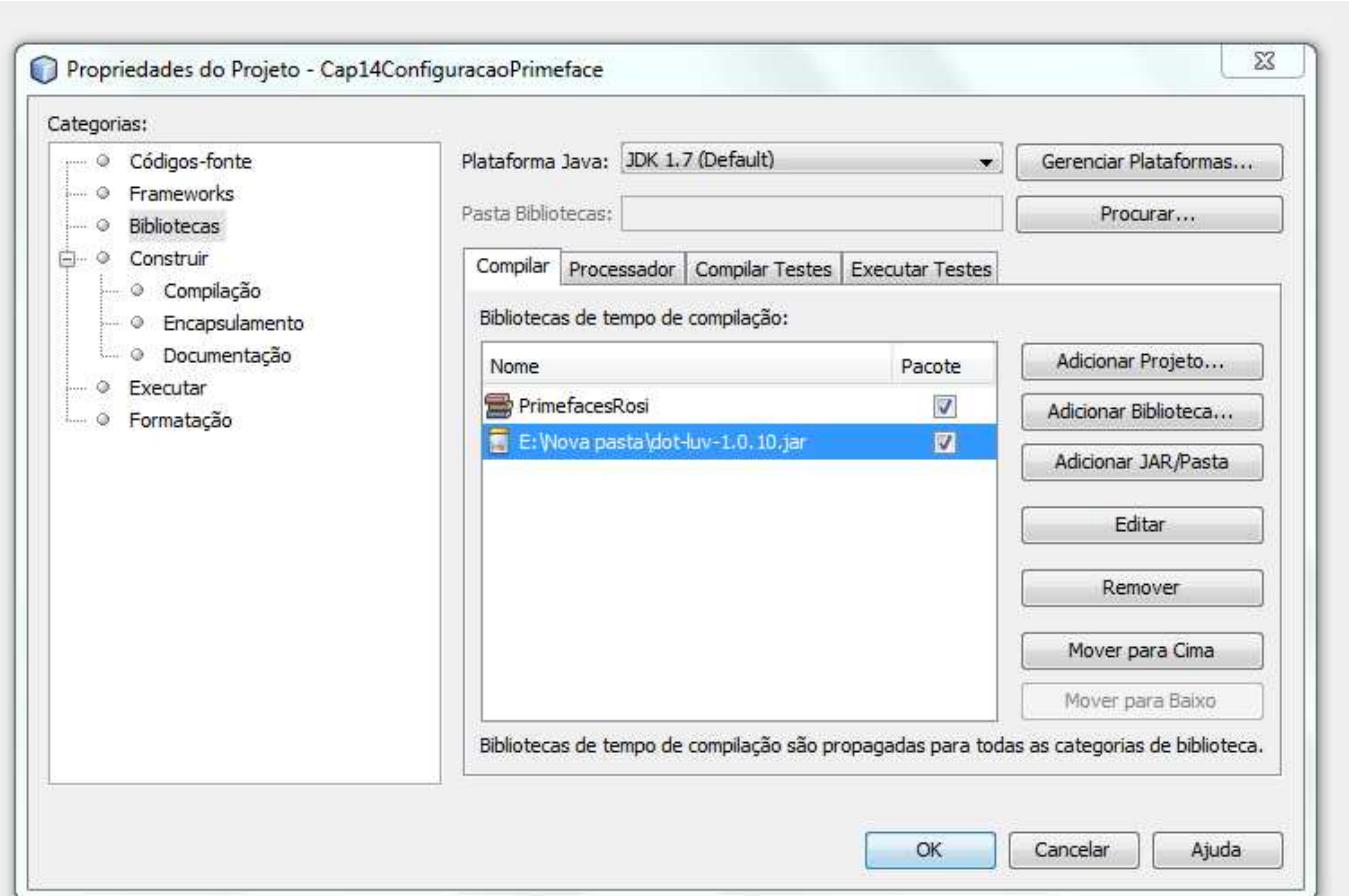
</f:view>
</html>
```

4. Ao executar o projeto a página renderizada deve ser similar a imagem seguinte.



5. A página apresentada possui o tema padrão do primefaces. Para alterar o mesmo é necessário efetuar o download do jar na página do primefaces: <http://repository.primefaces.org/org/primefaces/themes/>. Neste exemplo, será utilizada a biblioteca dot-luv.

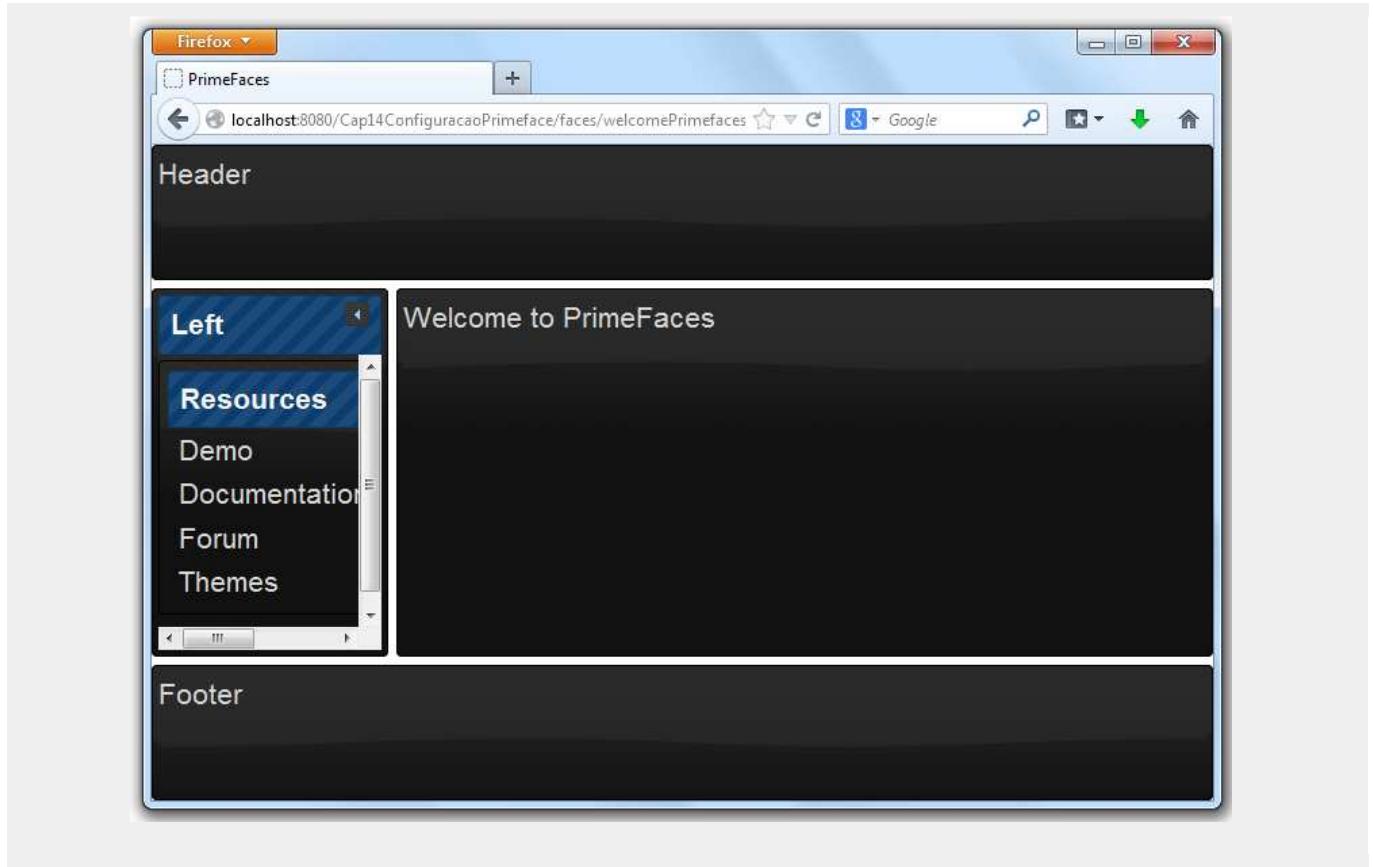
6. O próximo passo é adicionar a biblioteca ao projeto. No menu propriedade, biblioteca, adicionar jar/pasta, selecione o jar correspondente a biblioteca.



7. O web.xml também deve ser alterado para indicar que o tema deve ser alterado.

```
<context-param>
    <param-name>primefaces.THEME</param-name>
    <param-value>dot-luv</param-value>
</context-param>
```

8. Ao executar o projeto é possível perceber a alteração feita na página.



PASSO-A-PASSO

Aplicativo RH

Como o intuito é apenas demonstrar os componentes da biblioteca, este projeto terá as mesmas funcionalidades do projeto apresentado no capítulo anterior. Dessa forma, apenas a codificação referente a visualização deve ser alterada. As demais camadas do projeto continuam as mesmas.

1. Como já mencionado, as alterações serão efetuadas apenas na camada de visualização. O primeiro arquivo a ser modificado é o template, que é a base para outras páginas. É importante perceber que as divs foram substituídas pelo componente layoutUnit.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui">

    <h:head>
        <title>Gestão de Recursos Humanos</title>
        <h:outputStylesheet library="css" name="style.css"/>
```

```
</h:head>

<h:body>

    <p:layout fullPage="true">

        <p:layoutUnit position="north" size="100" header="Gestão de Recursos Humanos" >

            <h:graphicImage library="imagens" name="logo.png" />

        </p:layoutUnit>

        <p:layoutUnit position="south" size="35" header="&copy;2014 Todos os direitos reservados &nbsp;&bull;&nbsp; Rosicléia Frasson" style="text-align: center">

            </p:layoutUnit>

        <p:layoutUnit position="west" size="200" resizable="true" closable="true" collapsible="true">
            <ui:include src="../menu.xhtml"/>
        </p:layoutUnit>

        <p:layoutUnit position="center">
            <div id="conteudo">
                <ui:insert name="conteudo">
                </ui:insert>
            </div>
        </p:layoutUnit>

    </p:layout>
</h:body>
</html>
```

2. O arquivo menu.xhtml deve ser alterado para a utilização de um menu do primefaces.

```
<ui:composition xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:p="http://primefaces.org/ui">

    <h:form>
```

```

<p:panelMenu style="width: 193px;">
    <p:submenu label="Administração">
        <p:menuitem value="Cargos" url="/faces/paginas/lista-cargos.xhtml"/>
        <p:menuitem value="Funcionários" url="/faces/paginas/lista-
funcionarios.xhtml"/>
    </p:submenu>

    <p:submenu label="Recrutamento e Seleção">
        <p:menuitem value="Banco de Currículos" url="#" />
        <p:menuitem value="Avaliações" url="#" />
        <p:menuitem value="Processos Seletivos" url="#" />
    </p:submenu>

    <p:submenu label="Serviços">
        <p:menuitem value="Treinamento e Desenvolvimento" url="#" />
        <p:menuitem value="Folha de Pagamento" url="#" />
        <p:menuitem value="Ponto Eletrônico" url="#" />
        <p:menuitem value="Gestão de Benefícios" url="#" />
    </p:submenu>

    <p:submenu label="Relatórios Gerenciais">
        <p:menuitem value="Rotatividade" url="#" />
        <p:menuitem value="Produtividade" url="#" />
        <p:menuitem value="Custo" url="#" />
    </p:submenu>

</p:panelMenu>

</h:form>

</ui:composition>

```

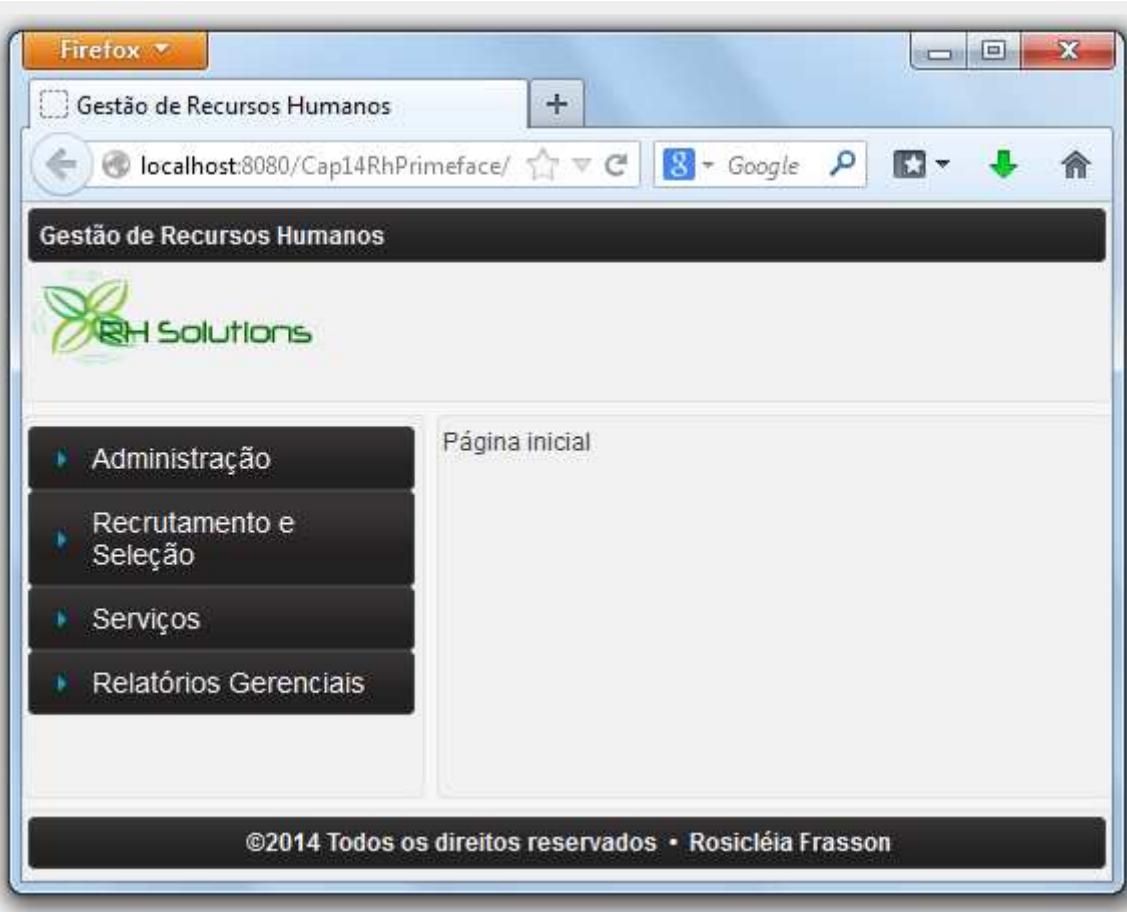
3. Para melhorar a aparência das páginas, deve ser alterado o tema das páginas. Vale lembrar que a biblioteca correspondente ao tema escolhido deve estar adicionada ao projeto. Neste exemplo, está sendo utilizado o skin rocket.

```

<context-param>
    <param-name>primefaces.THEME</param-name>
    <param-value>rocket</param-value>
</context-param>

```

4. Executando o projeto a página apresentada deve ser similar a imagem a seguir.



5. As páginas de cadastro e listagem de cargos e funcionários também devem ser alteradas. Seguem a codificação e a aparência das páginas.

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="../WEB-INF/templates/template.xhtml"
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:p="http://primefaces.org/ui">

    <ui:define name="conteudo">

        <h:form acceptcharset="ISO-8859-1">
            <p:messages id="messages" closable="true"/>
            <h:inputHidden value="#{cargoBean.cargo.codigo}"/>
            <p:panel header="Cadastro de Cargo">
                <h:panelGrid columns="2">

                    <h:outputLabel value="Nome: " for="nome"/>
                    <p:inputText id="nome" size="50"
                        value="#{cargoBean.cargo.nome}"
                        required="true"
                        requiredMessage="É necessário informar o nome do cargo." />
                
```

```
nome do cargo."/>

        <h:outputLabel value="Descrição: " for="descricao"/>
        <p:inputTextarea id="descricao"
                         value="#{cargoBean.cargo.descricao}"
                         cols="50" rows="10"/>
        <h:outputLabel/>

        <!-- Botões de Salvar e Cancelar -->
        <p:panel>
            <p:commandButton ajax="false"
                           value="Salvar"
                           action="#{cargoBean.insere}"/>
            <p:commandButton value="Cancelar"
                           type="reset"/>
        </p:panel>
    </h:panelGrid>
</p:panel>
</h:form>
</ui:define>

</ui:composition>
```

Cadastro de Cargo

Nome:

Descrição:

Salvar

Cancelar

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
                template="../WEB-INF/templates/template.xhtml"
                xmlns = "http://www.w3.org/1999/xhtml"
                xmlns:h="http://xmlns.jcp.org/jsf/html"
                xmlns:f="http://xmlns.jcp.org/jsf/core"
```

```
xmlns:p="http://primefaces.org/ui">

<ui:define name="conteudo">
    <h:form acceptcharset="ISO-8859-1" id="form">

        <p:dataTable id="tabela" var="cargo" value="#{cargoBean.listaCargos}"
sortMode="multiple" rows="8" paginator="true">
            <f:facet name="header">
                Cargos
            </f:facet>

            <p:column sortBy="codigo" headerText="Código">
                <h:outputText value="#{cargo.codigo}" />
            </p:column>

            <p:column sortBy="nome" headerText="Nome">
                <h:outputText value="#{cargo.nome}" />
            </p:column>

            <p:column>
                <f:facet name="header">
                    <p:commandButton action="/faces/paginas/cargo.xhtml" icon="ui-icon-plus">
                        </p:commandButton>
                </f:facet>

                <p:commandButton
                    action="#{cargoBean.preparaAlteracao}" icon="ui-icon-pencil">
                    <f:setPropertyActionListener value="#{cargo}" target="#{cargoBean.cargo}" />
                </p:commandButton>

                <p:commandButton
                    action="#{cargoBean.remove}" icon="ui-icon-close">
                    <f:setPropertyActionListener value="#{cargo}" target="#{cargoBean.cargo}" />
                </p:commandButton>
            </p:column>
        </p:dataTable>

    </h:form>
```

```
</ui:define>  
  
</ui:composition>
```



Código	Nome	
1	Analista de teste	 
3	Programador	 

```
<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"  
    template="../WEB-INF/templates/template.xhtml"  
    xmlns = "http://www.w3.org/1999/xhtml"  
    xmlns:h="http://xmlns.jcp.org/jsf/html"  
    xmlns:f="http://xmlns.jcp.org/jsf/core"  
    xmlns:p="http://primefaces.org/ui">  
  
    <ui:define name="conteudo">  
  
        <h:form acceptcharset="ISO-8859-1">  
            <p:panel header="Cadastro de Funcionário">  
                <p:messages id="messages" closable="true"/>  
                <h:inputHidden value="#{funcionarioBean.funcionario.codigo}" />  
  
                <p:fieldset legend="Dados Pessoais" toggleable="true">  
                    <h:panelGrid columns="4" width="100%">  
  
                        <h:outputLabel value="Nome: " for="nome"/>  
                        <p:inputText id="nome" size="50"  
                            value="#{funcionarioBean.funcionario.nome}"  
                            required="true"  
                            requiredMessage="É necessário informar o nome do  
                            funcionário." />  
  
                        <h:outputLabel value="CPF: " for="cpf"/>  
                        <p:inputMask id="cpf"  
                            value="#{funcionarioBean.funcionario.cpf}" />  
                    </h:panelGrid>  
                </p:fieldset>  
            </p:panel>  
        </h:form>  
    </ui:define>
```

```
        required="true"
        requiredMessage="É necessário informar o CPF."
        mask="999.999.999-99"/>

        <h:outputLabel value="Data de Nascimento: "
for="dtNascimento"/>

        <p:calendar id="dtNascimento"
value="#{funcionarioBean.funcionario.dataNascimento}"
        required="true"
        requiredMessage="É necessário informar a data de
nascimento."
        converterMessage="Data no formato inválido. Utilize
dd/mm/aaaa"
        pattern="dd/MM/yyyy">
        <f:convertDateTime pattern="dd/MM/yyyy" type="date"/>
    </p:calendar>

        <h:outputLabel value="Cargo: " for="cargo"/>
    <p:selectOneMenu id="cargo" value="#{funcionarioBean.cargoId}"
style="width:150px;">
        <f:selectItem itemLabel="Selecione" itemValue="" />
        <f:selectItems value="#{cargoBean.listaCargos}" var="cargo"
itemLabel="#{cargo.nome}" itemValue="#{cargo.codigo}"/>
    </p:selectOneMenu>

        <h:outputLabel value="Telefone: " for="telefone"/>
    <p:inputMask id="telefone"
value="#{funcionarioBean.funcionario.telefone}"
mask="(99) 9999-9999"/>

        <h:outputLabel value="Email: " for="email"/>
    <p:inputText id="email" size="50"
value="#{funcionarioBean.funcionario.email}"/>
</h:panelGrid>

</p:fieldset>

<p:fieldset legend="Endereço" toggleable="true" >
```

```
<h:panelGrid columns="4" width="100%>

    <h:outputLabel value="Endereço: " for="endereco"/>
    <p:inputText id="endereco"
value="#{funcionarioBean.funcionario.endereco.endereco}" size="55"/>

    <h:outputLabel value="Número: " for="numero"/>

    <p:inputMask id="numero"
value="#{funcionarioBean.funcionario.endereco.numero}"
validatorMessage="O número não pode ser negativo."
mask="9?9999999">
        <f:validateLongRange minimum="0"/>
    </p:inputMask>

    <h:outputLabel value="Complemento: " for="complemento"/>
    <p:inputText id="complemento"
value="#{funcionarioBean.funcionario.endereco.complemento}" size="55"/>

    <h:outputLabel value="Bairro: " for="bairro"/>
    <p:inputText id="bairro"
value="#{funcionarioBean.funcionario.endereco.bairro}" size="55"/>

    <h:outputLabel value="Estado: " for="estado"/>
    <p:inputText id="estado"
value="#{funcionarioBean.funcionario.endereco.estado}" size="55"/>

    <h:outputLabel value="Cidade: " for="cidade"/>
    <p:inputText id="cidade"
value="#{funcionarioBean.funcionario.endereco.cidade}" size="55"/>
</h:panelGrid>
</p:fieldset>

<p:fieldset legend="Dados de Acesso" toggleable="true">

    <h:panelGrid columns="4">
        <h:outputLabel for="login" value="Login: "/>
        <p:inputText id="login"
value="#{funcionarioBean.funcionario.login}" />
```

```

        <h:outputLabel for="senha" value="Senha: "/>
        <p:password id="senha"
value="#{funcionarioBean.funcionario.senha}" feedback="true"/>
        </h:panelGrid>

</p:fieldset>

<p:panel>
    <p:commandButton ajax="false"
        value="Salvar"
        action="#{funcionarioBean.insere}"/>
    <p:commandButton value="Cancelar"
        type="reset"/>
</p:panel>
</p:panel>

</h:form>
</ui:define>
</ui:composition>

```

Cadastro de Funcionário

— Dados Pessoais

Nome:	<input type="text"/>	CPF:	<input type="text"/>
Data de Nascimento:	<input type="text"/>	Cargo:	<input type="text" value="Selecione"/>
Telefone:	<input type="text"/>	Email:	<input type="text"/>

— Endereço

Endereço:	<input type="text"/>	Número:	<input type="text" value="0"/>
Complemento:	<input type="text"/>	Bairro:	<input type="text"/>
Estado:	<input type="text"/>	Cidade:	<input type="text"/>

— Dados de Acesso

Login:	<input type="text"/>	Senha:	<input type="text"/>
--------	----------------------	--------	----------------------

Salvar
Cancelar

```

<ui:composition xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="../WEB-INF/templates/template.xhtml"
    xmlns = "http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"

```

```
xmlns:p="http://primefaces.org/ui">

<ui:define name="conteudo">
    <h:form acceptcharset="ISO-8859-1">
        <p:dataTable value="#{funcionarioBean.listaFuncionarios}"
            sortMode="multiple" rows="25" paginator="true"
            var="funcionario">
            <f:facet name="header">
                <h:outputText value="Funcionários"/>
            </f:facet>

            <p:column>
                <f:facet name="header">
                    <h:outputText value="Código" />
                </f:facet>
                <h:outputText value="#{funcionario.codigo}"/>
            </p:column>

            <p:column>
                <f:facet name="header">
                    <h:outputText value="Nome" />
                </f:facet>
                <h:outputText value="#{funcionario.nome}"/>
            </p:column>

            <p:column>
                <f:facet name="header">
                    <h:outputText value="Data de nascimento" />
                </f:facet>
                <h:outputText value="#{funcionario.dataNascimento}">
                    <f:convertDateTime pattern="dd/MM/yyyy" type="date"/>
                </h:outputText>
            </p:column>

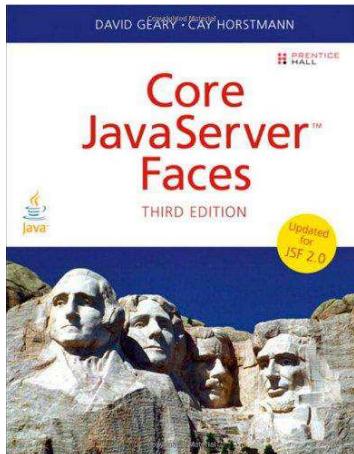
            <p:column>
                <f:facet name="header">
                    <h:outputText value="Cargo" />
                </f:facet>
                <h:outputText value="#{funcionario.cargo.nome}"/>
            </p:column>

            <p:column>
```

```
<f:facet name="header">
    <p:commandButton action="/faces/paginas/funcionario.xhtml"
icon="ui-icon-plus">
        </p:commandButton>
    </f:facet>
    <p:commandButton
        action="#{funcionarioBean.preparaAlteracao}" icon="ui-icon-pencil">
        <f:setPropertyActionListener value="#{funcionario}">
            target="#{funcionarioBean.funcionario}">
    </p:commandButton>
    <p:commandButton
        action="#{funcionarioBean.remove}" icon="ui-icon-close">
        <f:setPropertyActionListener value="#{funcionario}">
            target="#{funcionarioBean.funcionario}">
    </p:commandButton>
</p:column>
</p:dataTable>
</h:form>
</ui:define>
</ui:composition>
```

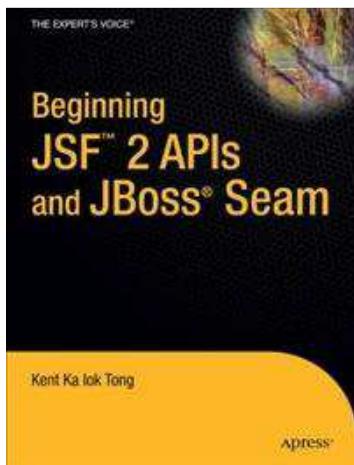
Funcionários				
Código	Nome	Data de nascimento	Cargo	
3	João de Souza	08/10/2010	Analista de teste	 

Bibliografia



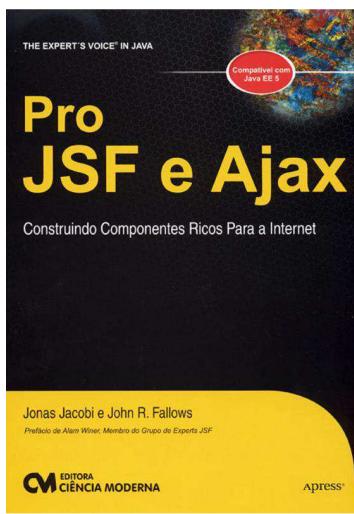
Core Java Server Faces

Autor: Geary, David / Horstmann, Cay
Editora: Prentice Hall
Publicação: 2010
Edição: 3



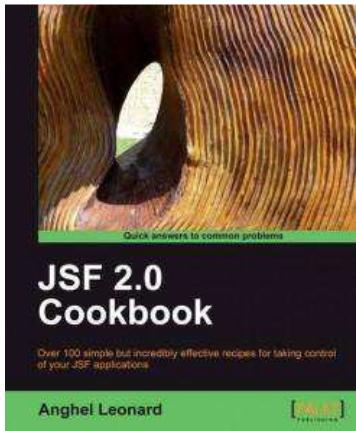
Beginning JSF 2 APIs and JBoss Seam

Autor: TONG, Kent Ka Iok
Editora: Apress
Publicação: 2009



Pro JSF e Ajax

Autor: JACOBI, Jonas/ FALLOWS, John R.
Editora: Ciência Moderna
Publicação: 2006



JSF 2.0 Cookbook
Autor: LEONARD, Anghel
Editora: Packt
Publicação: 2010

W3 Schools
<http://www.w3schools.com/>

