Java na WEB

Objetivos:

Capacitar o aluno a:

- estruturar o diretório de aplicação do Apache TomCat.
- configurar parcialmente o arquivo web.xml.
- desenvolver páginas web dinâmica com a tecnologia JSP.
- escolher os marcadores necessários para elaboração de páginas HTMLs.
- utilizar alguns objetos implícitos.

SUMÁRIO

Prof. Vida.
vida@utfpr.edu.br

1. Programação Java para a WEB com o TomCat

A estrutura de diretório

O Apache TomCat, é um servidor que, ao ser instalado, monta a estrutura de diretório ilustrada na Figura 1.

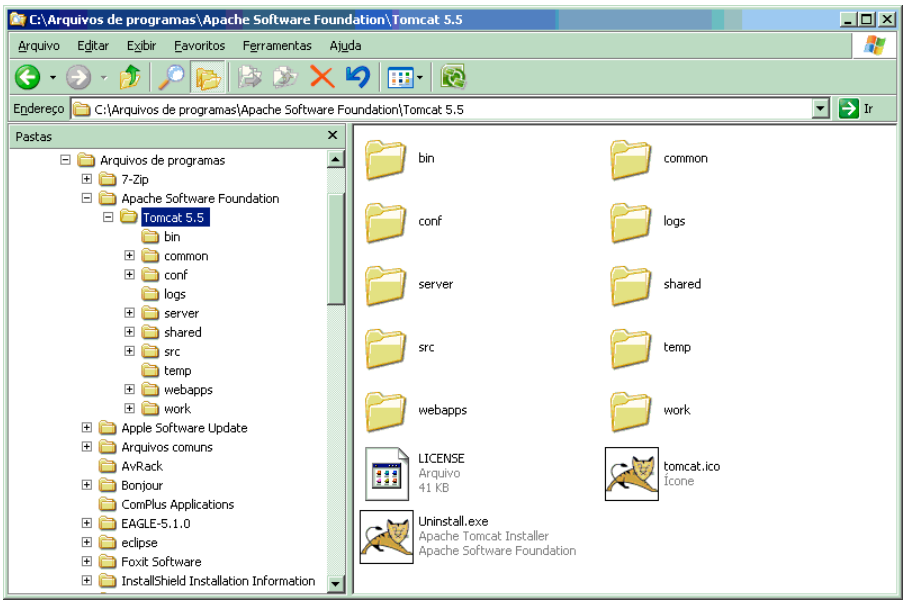


Figura 1-Estrutura de diretórios no Apache

Em qual pasta deve-se inserir um projeto?

A pasta *webapps* indica a entrada das aplicações web para o servidor. Por exemplo, se desejarmos desenvolver um projeto chamado **teste** esse projeto poderá ser inserido em uma estrutura de diretórios conforme ilustrado na Figura 2. Cada projeto deve ser instalado em sua própria pasta.

Na Figura 2, os arquivos são organizados nas categorias *jsp, html* e *images*. Logo, cada diretório deverá armazenar o seu correspondente tipo de arquivo.

*** O diretório WEB-INF informa ao servidor web alguns aspectos importantes e não deve ser omitido. Recomenda-se que esse diretório possua uma específica estrutura interna para armazenar classes e bibliotecas (lib), conforme ilustrado na Figura 2.

Os aspectos lidos pelo servidor são definidos no arquivo **web.xml** armazenado na pasta WEB-INF.

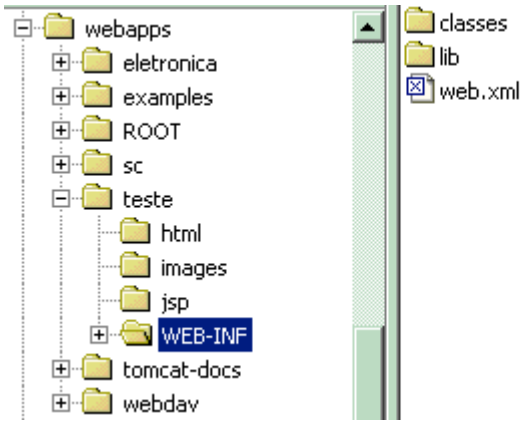


Figura 2-Estrutura de Diretório de um Projeto chamado teste.

Exemplo de conteúdo do arquivo web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
    xsi:schemaLocation=
      "http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
  <display-name>Welcome to Java na WEB</display-name>
  <description>
     Bem Vindo !
  </description>
  <welcome-file-list>
       <welcome-file>./jsp/index.jsp</welcome-file>
       <welcome-file>./jsp/index.html</welcome-file>
       <welcome-file>./html/index.html</welcome-file>
  </welcome-file-list>


<!—JSPC servlet mappings start -->
    <servlet>
        <servlet-
name>org.apache.jsp.index_jsp</servlet-name>
        <servlet-
class>org.apache.jsp.index_jsp</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-
name>org.apache.jsp.index_jsp</servlet-name>
        <url-pattern>/index.jsp</url-pattern>
    </servlet-mapping>
<!—JSPC servlet mappings end -->
```

```
</web-app>
```

Exemplo de um arquivo index.jsp

```
<html>
<head><title> Aplicacao WEB </title></head>
<body>
<h1> Informacoes</h1>
Host do Servidor: <b><%=request.getServerName()
%></b><br>
<%
java.util.Date date = new java.util.Date();
out.println("Hoje eh o dia " + String.valueOf(date));
%>
</html>
```

O arquivo index.jsp no servidor

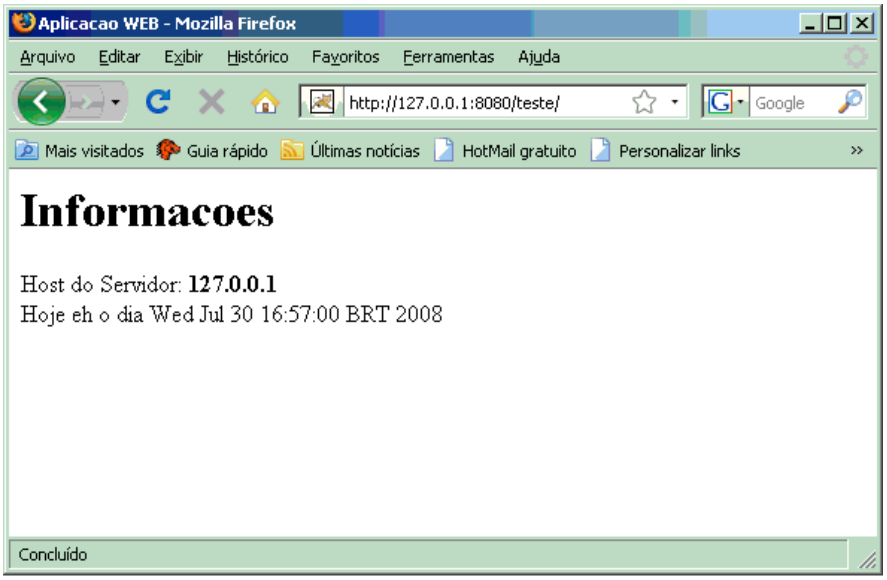


Figura 3 Exemplo de Execução do Arquivo index.jsp

Como receber um parâmetro de uma página web?

Todo e qualquer parâmetro ou variável submetida por um cliente ao servidor pode ser acessado através do comando **`request.getParameter(<nome do parametro>)`**.

Por exemplo, as Figuras 4 e 5 e apresentam duas possíveis execuções do arquivo boasvindas.jsp. O códio jsp desse arquivo é apresentado em seguida.

Código do arquivo boasvindas.jsp

```
<html><body>
<% String parametro = request.getParameter("nome");
 if (parametro==null) parametro = "Desconhecido";
  out.println("Ola " + parametro);%>
</body></html>
```

Exemplo de execução do arquivo boasvindas.jsp

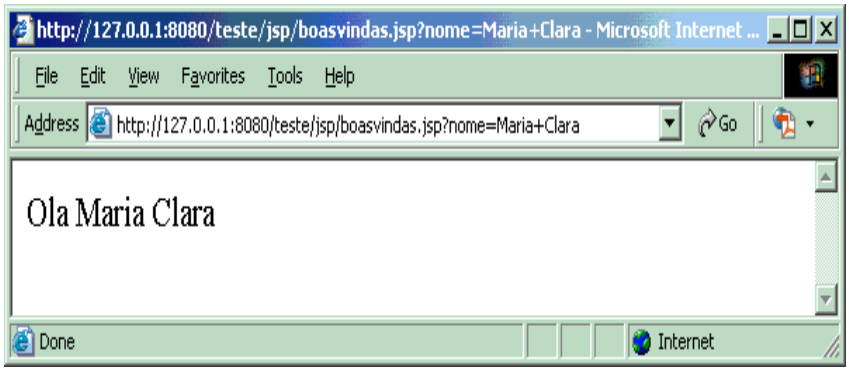


Figura 4-Exemplo de Execução do Arquivo boasvindas.jsp com Passagem de Parâmetro

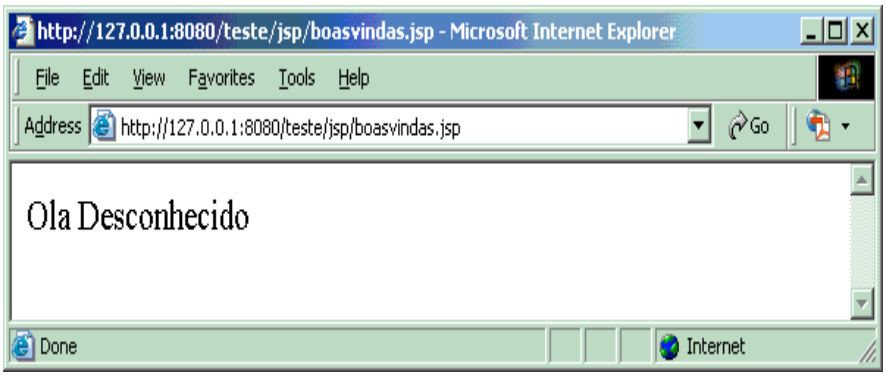


Figura 5-Exemplo de Execução do Arquivo boasvindas.jsp sem Passagem de Parâmetro

Como instanciar uma classe?

As classes que a aplicação necessitará utilizar devem ser agrupadas em pacotes e armazenadas na pasta **WEB-INF\classes**. A Figura 6 exemplifica esse conceito.

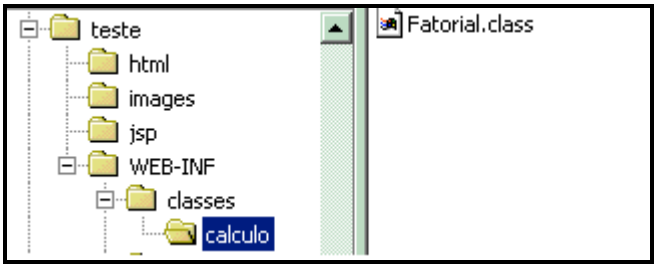


Figura 6-Estrutura de Armazenamento de Classes

No arquivo JSP, o equivalente à instrução **`import`** da programação Java é o comando **`page import`**.

Por exemplo, considerando que uma determinada classe esteja armazenada na pasta chamada calculo ilustrada na Figura 6, a instrução para acessar ao pacote é exemplificada abaixo.

```
<%@ page import="calculo.*"  %>
```

Exemplo: A classe Fatorial

```
package calculo;
public class Fatorial
{
      public double calcule(double x)
      {     double saida = 1;
            for (double i=x; i>1; i--) saida = saida *
i;
            return saida;
      }
}
```

O arquivo calculafatorial.jsp

```
<%@ page import="calculo.*"  %>
<html> <body>
<% String parametro = request.getParameter("numero");
 if (parametro!=null)
  {
   double                    num                    =
(Double.valueOf(parametro)).doubleValue();
   Fatorial fat = new Fatorial();
   out.println("<p>O fatorial de " + num + " eh " +
fat.calcule(num) + "</p>");
  }
  else
  {out.println("<p>Necessito de um parametro numero
para calcular o fatorial.</p>");}
%> </body> </html>
```

Exemplo de Execução:

A Figura 7 exemplifica a execução do arquivo calculafatorial.jsp.

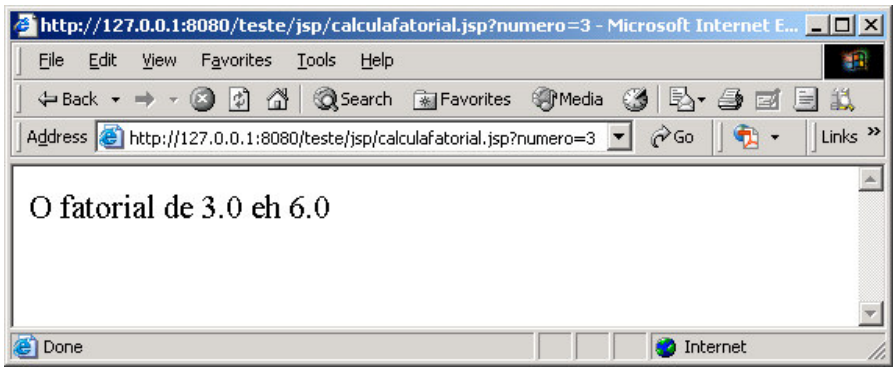


Figura 7-Exemplo de Execução do Arquivo calculafatorial.jsp

2. JSP

2.1 O que é JSP?

Java Server Pages – JSP - é uma tecnologia que simplifica o processo de desenvolvimento de sites web dinâmicos. Com JSP, os programadores WEB podem incorporar elementos dinâmicos em páginas da web usando a linguagem Java embutida no código HTML com alguns marcadores simples.

O código abaixo exemplifica um programa/arquivo JSP. A instrução java é fornecida entre os marcadores **<%** e **%>**. Tudo que estiver entre esses marcadores é considerado linguagem Java. O que estiver fora desses marcadores é instrução HTML.

2.2 Exemplo de código JSP: arquivo numeros.jsp

```
<html>
<head><title> Aplicacao WEB </title></head>
<body>
<%-- codigo java a serguir --%>
<%
// codigo Java
for (int i = 0; i < 20; i++)
 out.println("<br>" + String.valueOf(i));
%>
</html>
```

2.3 Resultado da Execução:

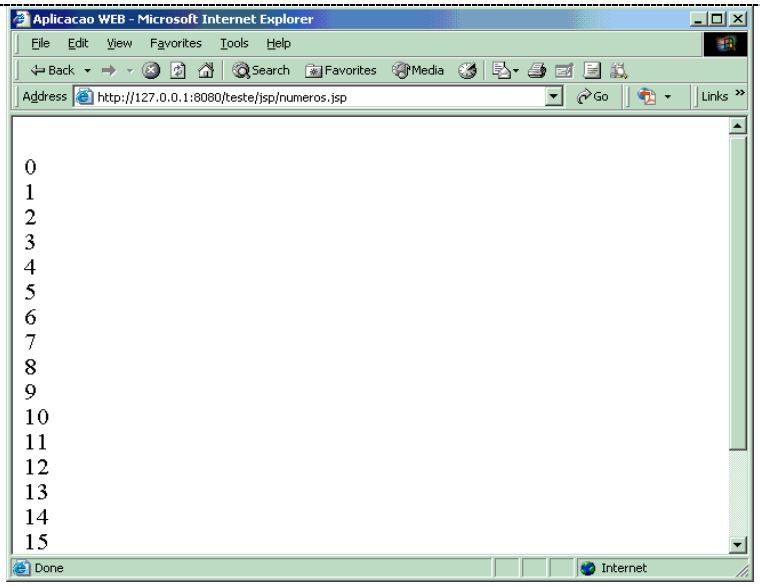


Figura 8 - Exemplo de Execução do Programa Contador

2.4 Estilos de Tags

A sintaxe do código JSP baseia-se em tags. Há quatro tipos de Tags, a saber:

- Diretiva: Contém informações que ajudam o contêiner a configurar e rodar uma página JSP.
- Declaração: Pode conter a declaração de variávies, constantes ou métodos.
- Expressão: É uma expressão escrita em Java que produz um resultado a ser inserido na página.
- Scriptlet: É um bloco de código escrito em Java.

2.4.1 Diretivas:

Diretivas são tags utilizadas para incluir informações sobre a própria página JSP. Há três tipos diferentes de diretivas, a saber:

- page: configura as propriedades de uma página jsp.
- include: insere o conteúdo de um outro arquivo.

2.4.1.1 Diretiva Page:

A diretiva `page` é definida de acordo com a seguinte sintaxe:

```
<@page atributo1=valor atributo2=valor2 ...>
```

Por exemplo:

```
<%@page import="calculo.*"  %>
<!-- indica classes e interfaces a serem importadas-->
<%@page errorPage=”trataErros.jsp” %>
<!-- indica a página que fará o tratamento de erros-->
```

São alguns atributos possíveis:

Atributo	Padrão	Descrição
info	“”	Permite definir textos que auxiliem an documentação.
language	“java”	especifica a linguagem de criação a ser usada nos scripts.
content Type		indica o tipo MIME da resposta a ser gerada.
extends	none	identifica a super classe da página JSP.
import	none	Permite importar uma classe ou um pacote.
session	true	Utilizado para indicar se a página participa ou não do gerenciamento de sessão.
buffer	8kb	Controla o buffer de saída.
autoFlush	true	Define que o buffer de saída deverá ser automaticamente descarregado quando ele estiver cheio.
isThreadSafe	true	Define se uma página JSP deve ser capaz de responder a solicitações simultaneas.
errorPage	none	Define uma página alternativa a ser apresentada caso ocorra um erro em tempo de execução.
isErrorPage	false	É usado para marcar uma página JSP que serve como a página de erro para uma ou mais páginas JSPs.

2.4.1.2 Diretiva Include:

A diretiva `include` é utilizada para inserir o conteúdo de outro arquivo em uma página JSP.

Exemplo:

```
<%@ include file="cabecalho.jsp" %>
...
<%@ include file="rodape.jsp" %>
```

2.4.2 Declarações:

Marcadores utilizados para declarar variáveis, constantes ou métodos.

Exemplo:

```
<%!  private int x = 3;  %>
<%!  public double Fatorial(double n)
        { if (n <= 1) {return 1;}
           else return (n*Fatorial(--n));
        } %>
 <% out.println(“O fatorial de “ +
           String.valueOf(x) + “ vale “
  + String.ValueOf(Fatorial(x)) ); %>
```

2.4.3 Expressões:	As expressões são marcadores utilizados para embutir na página JSP o resultado da avaliação de uma expressão Exemplo: `A valor de PI é <%=Math.PI%>`
2.4.5 Scriptlets	Scsiptlets são blocos de códigos de programação. Dentro desse bloco, a linguagem considerada é o Java. Exemplo: `<% // codigo Java` `for (int i = 0; i < 20; i++)` `out.println(" " + String.valueOf(i));` `%>`
Exercícios:	1) Crie uma aplicação para a web que escolha aleatoriamente um número entre 1 e 10 e exiba a tabuada desse número. 2) Crie uma aplicação que forneça o seno e o cosseno dos ângulos variando entre 0 a 360 graus com incremento de 1.

3. HTML 4.0 (Resumo de alguns Marcadores)

Marcadores de Texto e Arquivo:

<BODY> ... </BODY>	**Uso:**	Contém o conteúdo do documento.
	Atributos:	• background=" " <! imagem de segundo plano> • bgcolor=" " <! define a cor de segundo plano> • text=" " <! define a cor do texto de segundo plano. • link=" " <! cor do link > • vlink=" " <! cor do link visitado> • alink=" " <!cor do link ativo >
	Nota	Pode existir somente um único elemento BODY. O elemento BODY pode ser substituido por um elemento FRAMESET

	Exemplo:	`<html>`
		`<body bgcolor="yellow" text="red">`
		`ola !`
		`</body>`
		`<html>`

<! ..... >	**Uso:**	Usado para inserir comentários.
	Atributos:	nenhum
	Nota:	Os comentários não estão restritos a uma única linha e podem ter qualquer tamanho.
	Exemplo:	<! isso é um comentário ignorado pelo navegador >
<H1>... </H1> a <H6>... </H6>	**Uso:**	Cabeçalhos que são utilizados para estruturar a informação.

	Atributos:	align=" " <! controle de alinhamento. Os valores possíveis são {left, center, right, justify} > dir=" " <! Direção do texto. Os valores possíveis são {ltr, rtl}>
	Nota:	Os navegadores apresentam o tamanho do texto de acordo com a sua importância.

	Exemplo:	`<html>`
		`<body>`
		`<h1 align="center"> Teste </h1>`
		`<h2 dir="ltr"> Teste2 </h2>`
		`<h3 dir="rtl"> Teste3 </h3>`
		`</body>`
		`<html>`

<HEAD>...</HEAD>	**Uso:**	Cabeçalho do documento.
	Atributos:	
	Nota:	Pode existir apenas um único marcador <HEAD> no documento

	Exemplo:	`<html>`
		`<HEAD> JSP </HEAD>`
		`<body>`
		`<h1 align="center"> Teste </h1>`
		`<h2 dir="ltr"> Teste2 </h2>`
		`<h3 dir="rtl"> Teste3 </h3>`
		`</body>`
		`<html>`

<HR>	**Uso:**	Régua horizontal. Utilizado para separar seções.
	Atributos:	• align=" " <! controle de alinhamento. Os valores possíveis são {left, center, right, justify} > • noshade • size=" " <! define o tamanho da régua > • width = " " <! define a largura da régua >
	Nota:	Os navegadores apresentam o tamanho do texto de acordo com a sua importância.

	Exemplo:	`<html>`
		`<body>`
		`<hr align="center" noshade size=10 >`
		`<hr align="right"  size=10 width=40>`
		`</body>`
		`<html>`

<HTML>...</HTML>	**Uso:**	Delimita todo o documento.
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`ola`
		`</body>`
		`<html>`

** ... **	**Uso:**	Define um parágrafo.
	Atributos:	align=" " <! controle de alinhamento. Os valores possíveis são {left, center, right, justify} >
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`<p> Ola </p>`
		`</body>`
		`<html>`

** ... **	**Uso:**	negrita um texto.
	Atributos:	align=" " <! controle de alinhamento. Os valores possíveis são {left, center, right, justify} >
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`Ola <strong> Ola </strong>`
		`</body>`
		`<html>`

** ... **	**Uso:**	texto em negrito.
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`Ola <b>  Ola </b>`
		`</body>`
		`<html>`

^{...}	**Uso:**	cria um sobrescrito.
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`ola <sup> 456 </sup>`
		`</body>`
		`<html>`

_{...}	**Uso:**	cria um subscrito.
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`ola <sub>  123 </sub>`
		`</body>`
		`<html>`

<BASEFONT>	**Uso:**	Define o tamanho da fonte base.
	Atributos:	• size=" " <! tamanho relativo 1 a 7 > • color=" " • face=""
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		
		`<basefont size="14" color="blue"`
		`face="Arial">`
		`teste`
		`</body>`
		`<html>`

<BIG> ... </BIG>	**Uso:**	Texto com fonte Grande
	Atributos:	

	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`<big> Ola </big> ola`
		`</body>`
		`<html>`

<FONT> **...** **</FONT>**	**Uso:**	altera a fonte
	Atributos:	• size =”” : tamanho relativo 1 a 7 • color=”” : define o valor da cor • face=”” : define o tipo de fonte
	Nota:	
	Exemplo:	`<html>` `<body>` `<font size="1" color="red"` `face="Times"> Ola </font>` `<font size="2" color="blue"` `face="Arial"> Ola </font>` `<font size="3" color="green"` `face="Arial"> Ola </font>` ` ` `<font size="4" color="red"` `face="Times"> Ola </font>` `<font size="5" color="blue"` `face="Arial"> Ola </font>` `<font size="6" color="green"` `face="Arial"> Ola </font>` ` ` `<font size="7" color="red"` `face="Times"> Ola </font>` `<font size="7" color="blue"` `face="Arial"> Ola </font>` `<font size="7" color="green"` `face="Arial"> Ola </font>` `</body>` `</html>`
<u> **...** **</u>**	**Uso:**	Texto sublinhado
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`Ola <u>ola</u>`
		`</body>`
		`</html>`

<s> **...** **</s>**	**Uso:**	Texto tachado
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`Ola <s>ola</s>`
		`</body>`
		`</html>`

**** **...** ****	**Uso:**	Texto com fonte pequena
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`Ola <small>ola</small>`
		`</body>`
		`</html>`

<tt> ... </tt>	**Uso:**	Texto monoespaçado
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`Ola <tt>ola</tt>`
		`</body>`
		`</html>`

** ... **	**Uso:**	Cria uma lista
	Atributos:	
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`<li> Primeiro`
		`<li> Segundo`
		`</body>`
		`</html>`

**<a> ... **	**Uso:**	Define links e ancoras
	Atributos:	name=" " : define uma ancora href=" " : o url vinculado. target=" " : define o local ande o recurso será apresentado. Valores possíveis {_blank, _parent, _self, _top}. accesskey=" ":define uma tecla de atalho.
	Nota:	

	Exemplo:	`<html>`
		`<body>`
		`<a href="http://www.google.com.br"`
		`target="_blank" > ola </a>`
		`</body>`
		`</html>`

Marcadores de Tabelas:

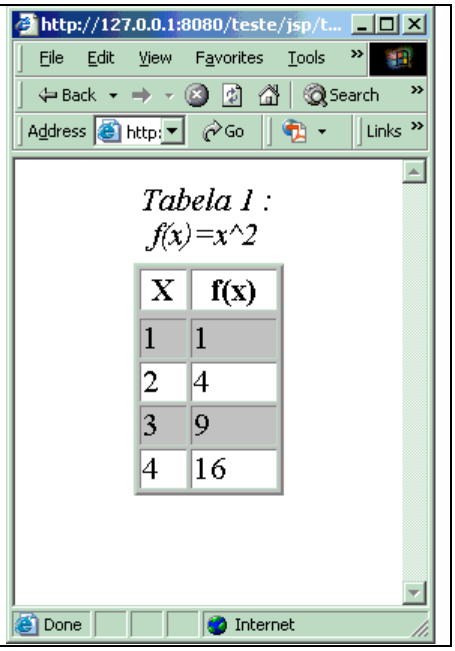
<TABLE>. ..</TABLE>	**Uso:**	Criar uma tabela

	Atributos:	• align="" : controle de alinhamento. Valores possíveis {left, center, right, justify} • bgcolor="": define a cor de segundo plano. • width="" : define a largura da tabela • cols="" : define o número de colunas. • border="" : define a largura em pixels da borda. • frame="" : define os lados visíveis de uma tabela. Valores possíveis {void, above, below, hsides, lhs, ths, vsides, box, border} • rules="": define réguas visíveis em uma tabela. Valores possíveis {none, groups, rows, cols, all}. • cellspacing="":espaçamento entre células. • cellpadding="" : espaçamento nas células.
	Nota:	
<CAPTION> ... </CAPTION>	**Uso:**	Apresenta a legenda de uma tabela.
	Atributos:	align="" : Controle de alinhamento. Valores possíveis {left, center, right, justify}
	Nota:	
<TD> ... </TD>	**Uso:**	Define o conteúdo de uma célula
	Atributos:	• nowrap="" : desativa a mudança automática de linha em uma célula. • bgcolor="": define a cor de segunda plano. • rowspan="" : o número de linhas ocupado por uma célula. • colspan="" : o número de colunas ocupado por uma célula. • align="" : Controle de alinhamento. Valores possíveis {left, center, right, justify} • valign="" : Alinha verticalmente o conteúdo das células. Valores possíveis {top, middle, botton, baseline}.
	Nota:	
<TR> ... </TR>	**Uso:**	Define uma linha de células da tabela.
	Atributos:	• bgcolor="": define a cor de segunda plano. • align="" : Controle de alinhamento. Valores possíveis {left, center, right, justify} • valign="" : Alinha verticalmente o conteúdo das células. Valores possíveis {top, middle, botton, baseline}.
	Nota:	
	Exemplo:	
<TH>...</TH>	**Uso:**	Define o conteúdo do cabeçalho da tabela.

	Atributos:	• nowrap="" : desativa a mudança automática de linha em uma célula. • bgcolor="": define a cor de segunda plano. • rowspan="" : o número de linhas ocupado por uma célula. • colspan="" : o número de colunas ocupado por uma célula. • align="" : Controle de alinhamento. Valores possíveis {left, center, right, justify} • valign="" : Alinha verticalmente o conteúdo das células. Valores possíveis {top, middle, botton, baseline}.

Exemplo de Marcadores de Tabelas:

```
<html>
<body>
<table border="2" align="center"
bgcolor="silver" width="100"
frame="box" rules="all">
<caption> <i>Tabela 1 : f(x)=x^2
</i> </caption>
<tr bgcolor="white"><th> X </ht>
<th> f(x) </th><tr>
<tr bgcolor="silver"><td> 1 </td>
<td> 1 </td> </tr>
<tr bgcolor="white"><td> 2 </td>
<td> 4 </td> </tr>
<tr bgcolor="silver"><td> 3 </td>
<td> 9 </td> </tr>
<tr bgcolor="white"><td> 4 </td>
<td> 16 </td> </tr>
</table>
</body>
</html>
```



Marcadores de Formulários:

<FORM> **. . .** **</FORM>**	**Uso:**	Cria um formulário.
	Atributos:	• action="": URL destino. • method="" : Qual o método HTTP de envio. Valores possíveis {post, get} • target="" : Determina onde a URL de destino será apresentada. Valores possíveis {_blank, _parent, _self, _top}.
	Exemplo:	`<form action = 'leAtributo.jsp' method='POST'>`
<INPUT>	**Uso:**	Define Controles em formulários

Atributos	• type=" ": O tipo de controle de entrada. Valores possíveis {text, password, checkbox, radio, button, submit, reset, file, hidden, image} • name=" ": O nome do controle. • value=" ": O valor inicial do controle. Atributo obrigatório apra botões de opção e caixas de seleção. • checked : define botões de opção com o estado marcado. • disabled : desativa o controle. • readonly : para tipos de senha de texto. • size="", a largura do componente em pixels • maxlength="" : o número máximo de caracteres que podem ser digitados. • align="" : Controle de alinhamento. Valores possíveis {left, center, right, justify} • tabindex="" : define a ordem de tabulação entre os elementos.
Exemplo:	`<html>` `<body>` `<form>` ` <input type="text" name="texto" value="...">` ` <input type="password" name="senha" size="8" maslength="8">` ` <input type="checkbox" name="sucos" > Laranja` ` <input type="checkbox" name="sucos" > Uva` ` <input type="radio" name="sexo" value="m" checked> Masculino` ` <input type="radio" name="sexo" value="f" > Feminino` ` <input type="button" name="btOk" Value="Ok">` ` <input type="submit" name="submit" value="Enviar">` ` <input type="reset" name="reset" value="limpa">` ` <input type="file" name="file" value="arquivo">` ` <input type="hidden" name="pi" value="3.1415">` `<bt><input type="image" name="figura" src="http://127.0.0.1:8080/teste/images/tomcat.gif">` `</form>` `</body>` `</html>`

<table>
<tr><td rowspan="4"><SELECT>
...
<OPTION>
...
</OPTION>
...
</SELECT></td><td>Uso:</td><td>Define uma caixa de seleção.</td></tr>
<tr><td>Atributos:</td><td>• name="": nome do elemento.
• size="" : tamanho
• multiple : permite múltiplas seleções.
• disable : desabilita o componente.
• tabindex="" : define a ordem de tabulação.
• valign="" : Alinha verticalmente o conteúdo das células. Valores possíveis {top, middle, botton, baseline}.</td></tr>
<tr><td>Exemplo 1:</td><td><pre><html>
<body>
<form>
<select name="sel" multiple>
<option value=1> Curso 1</option>
<option value=2> Curso 2</option>
<option value=3> Curso 3</option>
<option value=4> Curso 4</option>
</select>
</form>
</body>
</html></pre></td></tr>
<tr><td>Exemplo 2</td><td><pre><html>
<body>
<form>
<select name="sel">
<option value=1> Curso 1</option>
<option value=2> Curso 2</option>
<option value=3> Curso 3</option>
<option value=4> Curso 4</option>
</select>
</form>
</body>
</html></pre></td></tr>
<tr><td rowspan="3"><TEXTAREA>
...
</TEXTAREA></td><td>Uso:</td><td>Cria uma área para entrada de texto.</td></tr>
<tr><td>Atributos:</td><td>• name="" : nome do controle
• rows="" : quantidade de linhas.
• cols="" :quantidade de colunas.
• disabled: desativa o controle
• readonly: o usuário não poderá digitar.
• tabindex: define a ordem de tabulação.</td></tr>
<tr><td>Exemplo:</td><td><pre><html>
<body>
<form>
<textarea name = "comentarios" rows=10
cols=30>digite alguma coisa</textarea>
</form>
</body>
</html></pre></td></tr>
</table>

4. Objetos Implícitos.

4.1 Introdução

Conforme apresentado, um programador pode fazer uso de qualquer classe java que ele tenha desenvolvido em uma página JSP. Para acessar um objeto de uma página JSP, ele necessita inserir a instrução `import`. Por exemplo, **`<%@page import="calculo.*" %>`**.

Além desse recurso, o programador ainda pode fazer uso de nove objetos inplícitos. Eles são denominados objetos inplícitos pois a sua existência é implicita a qualquer página JSP:

Os nove objetos implícitos são listado a seguir:

- Objetos relacionados ao servlet:
  - page
  - config
- Objetos relacionados s Input/Output:
  - **request**
  - response
  - out
- Objetos Contextuais:
  - **session**
  - **application**
  - **pageContext**
- Objetos relacionados a tratamento de erros:
  - exception

Atributos escondidos:

Os objetos **request, session, application** e **pageContext** possuem a habilidade de armazenar e recuperar valores de atributos arbitrários. Ao definir e obter o valor de um atributo, essas informações podem ser compartilhadas por diversas páginas JSPs.

São métodos comuns para armazenar atributos:

- setAttribute(String key, String valor)
- public Enumeration getAttributeNames()
- public String getAtribute(String key)
- removeAttibute(key)

Exemplo de definição Atributos:

```
<html>
<body>
<%
session.setAttribute("compositor"
            ,"Antonio Carlos Jobim");
session.setAttribute("musica"
            ,"Garota de Ipanema");
session.setAttribute("estilo"
            ,"Bossa Nova");
%>
<form action = 'leAtributo.jsp' method='POST'>
<input type="submit" value='continue'>
</form>
</body>
</form>
```

Exemplo de leitura de Atributos:

```
<%@page import="java.util.*"  %>
<html>
<body>
<%
Enumeration atributos =
session.getAttributeNames();
Object escondido;

for (; atributos.hasMoreElements() ;)
 { escondido = atributos.nextElement();
   out.println("<br> Atributo armazenado: ["
                + escondido +"]");
   out.println(" ; Valor = "
        + session.getAttribute((String)escondido));
 }
%>
</body>
</form>
```

4.2. Objeto page

Implementa a interface javax.servlet.jsp.HttpJspPage e representa a própria página JSP. Ele pode ser utilizado como uma referência ao servlet que será gerado a partir desta página durante o processo de tradução.

Na prática, o objeto para raramente é usado quando a linguagem de criação de scripts de JSP é o Java.

4.3 Objeto config

O objeto config armazena dados de configuração de servlet – na forma de inicialização – para o servlet no qual a JSP é compilada. Ele implementa a interface javax.servlet.ServletContext.

Na prática, esse objeto é raramente usado. As páginas JSP raramente precisam acessar dados através de parâmetros de inicialização.

4.4 Objeto request

Objeto que representa a solicitação que acionou o processamento da página atual.

Métodos que estão associados ao objeto:

- public Enumeration getParameterNames( ) => retorna os nomes de todos os parâmetros de solicitação.
- public String getParameter(String key) => retorna o primeiro valor de um único parâmetro de solicitação.
- public String[ ] getParameterValues(String key) => Recupera todos os valores para um único parâmetro de solicitação.

- getHeaderNames( ) => Recupera o nome de todos os cabeçalhos associados com a solicitação.
- getHeader(key) => Retorna o valor de um único cabeçalho de solicitação.
- getHeaders(key) => Retorna todos os valores para um único cabeçalho de solicitação.
- getIntHeader(key) => Retorna um único cabeçalho de solicitação, como un número inteiro.
- getDateHeader(key) => Retorna o valor de um único cabeçalho de solicitação como uma data.
- public Cookie[ ] getCookies( ) => Retorna todos os cookies associados com a solicitação.
- getMethod( ) => Retorna o método de HTTP (ex. GET, POST)

para a solicitação.

- getRequestURL( ) => Retorna o URL de solicitação.
- getQueryString( ) => Retorna a cadeia de consulta que segue o URL de solicitação, se houver algum.
- getSession (flag) => Recupera os dados de sessão para a solicitação.
- getRequestDispacher(path) => Cria um dispacher de solicitação para o URL local indicado.
- getRemoteHost( ) => Retorna o nome do host que enviou a solicitação.
- getRemoteAddr( ) => Retorna o endereço de rede do host que enviou a solicitação.
- getRemoteUser( ) => Retorna o nome do usuário que enviou a solicitação, se conhecido.

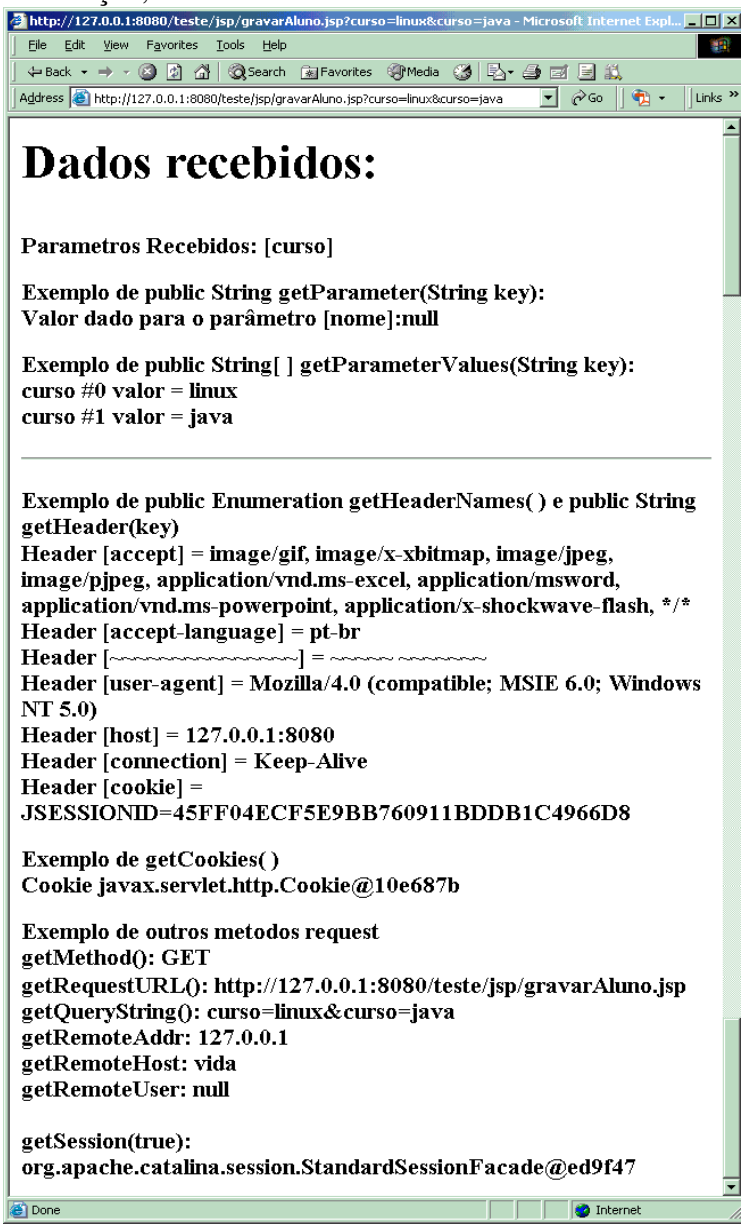


Figura 9 - Snapshot do exemplo Métodos Request

Exemplo de alguns métodos do objeto request:

```
<%@page import="java.util.*"  %>
<html>
<body>
<h1> Dados recebidos:</h1>
<p>
<%
  Enumeration e = request.getParameterNames( );
  out.println("<br><b>Parametros Recebidos:<b>");
  for (; e.hasMoreElements() ;)
     out.println("["+e.nextElement()+"]");
%>
</p>
<p> <b>Exemplo de public String getParameter(String
key):</b>
<%
 out.println("<br>Valor dado para o parâmetro "
        + "[nome]:" + request.getParameter("nome"));
%>
</p>

<p> <b> Exemplo de public String[ ]
getParameterValues(String key):</b>
<%
  String[] cursos =
request.getParameterValues("curso");
  for (int i = 0; i < cursos.length; i++)
   out.println("<br>curso #" +i+ " valor = " +
                cursos[i]);
%>
</p>
<hr> <p> <b> Exemplo getHeaderNames( ) e
getHeader(key)</b>
<%
    Enumeration h = request.getHeaderNames();
    Object aux;
    for (;h.hasMoreElements();)
    { aux = h.nextElement();
       out.println("<br>Header [" +aux+ "] = " +
                 request.getHeader((String)aux));}
%>
</p>
<p> <b> Exemplo de getCookies( )</b>
<%  javax.servlet.http.Cookie[] c =
                          request.getCookies();
    for (int i = 0; i < c.length; i++)
       out.println("<br>Cookie " + c[i]);
%>
</p><p><b> Exemplo de outros metodos request</b>
<% out.println("<br> getMethod(): "
               + request.getMethod());
   out.println("<br> getRequestURL(): "
               + request.getRequestURL());
   out.println("<br> getQueryString(): "
               + request.getQueryString());
   out.println("<br> getRemoteAddr: "
               + request.getRemoteAddr());
   out.println("<br> getRemoteHost: "
               + request.getRemoteHost());
   out.println("<br> getRemoteUser: "
               + request.getRemoteUser());
   out.println("<br><br> getSession(true): "
               + request.getSession(true));%>
</p></body></html>
```

Exemplo de Sugestão de Formulário de envio:

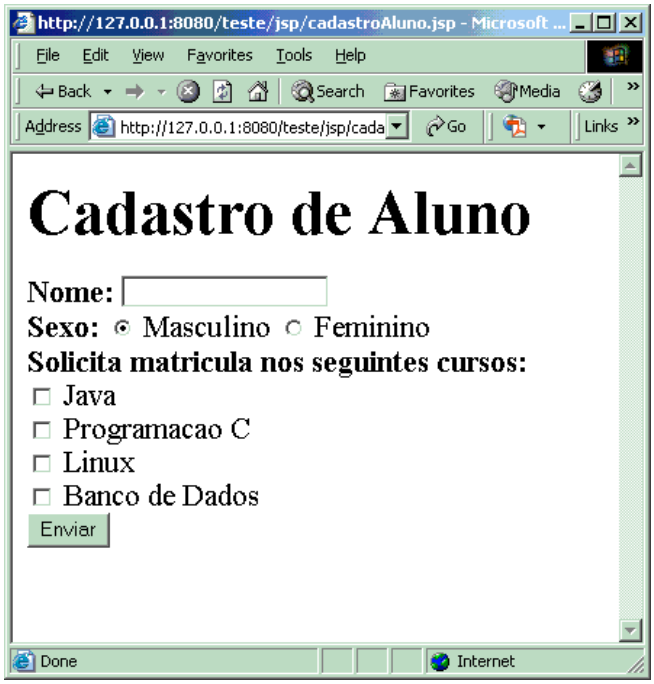


Figura 10-Exemplo de formulário

Código do Formulário de envio

```
<html>
<body>
<h1> Cadastro de Aluno </h1>
<form action="gravarAluno.jsp" method="POST">
  <b> Nome: </b> <input type="text" name="nome"
                        size = "20">
  <br> <b>Sexo:</b>
  <input type="radio" name="sexo" value="M"
                              checked /> Masculino
  <input type="radio" name="sexo" value="F" />
                               Feminino
  <br>
  <b> Solicita matricula nos seguintes cursos:</b>
  <br> <input type="checkbox" name="curso"
                         value ="java"/> Java
  <br> <input type="checkbox" name="curso"
                         value ="c"/> Programacao C
  <br> <input type="checkbox" name="curso"
                         value ="linux"/> Linux
  <br> <input type="checkbox" name="curso"
                         value ="bd"/> Banco de Dados
  <br><input type="submit" value="Enviar" />
</body>
</html>
```

4.5 Objeto response:

Implementa a interface javax.servlet.http.HttpServletResponde e representa a resposta a ser produzida pelo servlet gerada a partir de uma página JSP.

O objeto response pode ser utilizado para manipular vários aspectos da resposta a ser produzida, tais como o tipo e tamanho da mesma e os cabeçalhos HTTP que serão enviados junto.

Alguns Métodos do objeto response:

`setContentType( )`	Define o tipo MIME e, opcionalmente, acodificação de caracteres do conteúdo da resposta.
`addCookie(cookie)`	Adiciona o cookie especificado à resposta.
`setDateHeader( nome, date)`	Atribui o valor de data especificado no cabeçalho nomeado.
`setHeader( nome, valor)`	Atribui o valor de cadeia especificado ao cabeçalho nomeado.
`sendRedirect(url)`	Envia uma resposta para o navegador indicando que ele deveria solicitar um URL alternativo.

Exemplo do objeto response:

```
<html>
<body>
<p> Nao armazena no cahe do navegador.<p>
<p> Primeiro, ele define o cabeçalho "Expires"
para uma data no passado. Isto indica ao navegador
que a página já expirou, e o seu conteúdo não deve
ser armazenado no cache.
</p>
<p>
O valor "no-cache" para o cabeçalho Pragma é
fornecido pela versão 1.0 do protocolo HTTP, para
indicar que os navegadores e servidores proxy não
devem armazenar um cache a página.
A versão 1.1 de HTTP substitui este cabeçalho por
um cabeçalho "Cache-Control" mais específico, mas
recomenda a inclusão do cabeçalho "Pragma"também,
para a compatibilidade com as versões anteriores.
</P>

<% response.setDateHeader("Expires",0);
   response.setHeader("Pragma","no-cache");
   if (request.getProtocol().equals("HTTP/1.1"))
      {response.setHeader("Cache-Control","no-
cache");}
%>
</body>
</html>
```

4.6 Objeto session

Objeto que representa a sessão atual de um usuário individual.

Todas as solicitações feitas por um usuário, que são partes uma série de interações com o servidor web, são consideradas parte de uma sessão. Enquanto o servidor web receber novas solicitações feitas pelos usuários, a sessão persiste. Se um longo período de tempo passar sem que o usuário faça uma nova solicitação, a sessão expira.

Um dos principais usos para o objeto session é o de armazenar e recuperar valores de atributos a fim de transmitir informações específicas de usuários entre as páginas.

Exemplo:

<% Pessoa p = new Pessoa(“Jose”);
session.setAttribute(“usuario”, p); %>

Uma vez armazenado, o dado pode ser lido através do seuinte trecho de código:

<% Pessoa usu = (Pessoa) session.getAttribute(“usuario”); %>

<table>
<tr><td rowspan="6">Métodos do objeto session:</td><td>getId( )</td><td>Retorna o ID da sessão.</td></tr>
<tr><td>getCreationTime()</td><td>Retorna a hora na qual a sessão foi criada.</td></tr>
<tr><td>getLastAccessedTime()</td><td>Retorna a última vez que uma solicitação associada com a sessão foi recebida.</td></tr>
<tr><td>getMaxInactiveInterval( )</td><td>Retorna o tempo máximo (em segundos) entre solicitações pelo qua a sessão será mantida.</td></tr>
<tr><td>isNew()</td><td>Retorna true se onavegador do usuário ainda não tiver confirmado o ID de sessão.</td></tr>
<tr><td>invalidate()</td><td>Descarta a sessão, libera todos os objetos armazenados como atributos.</td></tr>
<tr><td>4.7 Objeto application</td><td colspan="2">Objeto que representa a aplicação à qual a página JSP pertence.</td></tr>
<tr><td>Exemplo do objeto application:</td><td colspan="2">

```
<html>
<body>
<%
  out.println("Servidor:"
        +application.getServerInfo());
  out.println(
  "<br>Versao principal da API do servlet:"
       + application.getMajorVersion());
  out.println(
   "<br>Versao secundaria da API do Servlet:"
       + application.getMinorVersion());
  out.println("<br> Tipo de MIME:"
  + application.getMimeType("application.jsp")
           );
%>
</body>
</html>
```

</td></tr>
<tr><td>4. 8 Objeto pageContext</td><td colspan="2">Oferece métodos essenciais para o armazenamento de informações em memória para que possam ser recuperadas em momento posterior, dentro da mesma página ou em outras páginas ou até por outros tipos de componentes da aplicação.</td></tr>
</table>

4.9 Objeto exception:

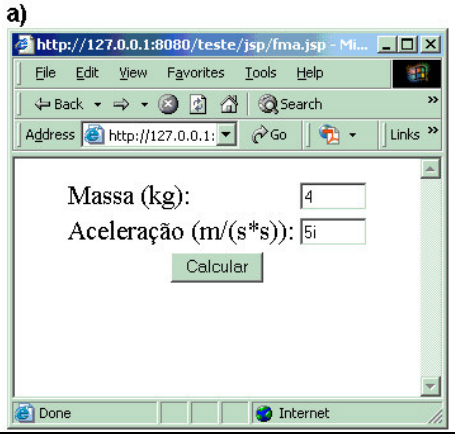
É uma instância da classe Throwable e é utilizado para o tratamento de erros ou exceções. Esse objeto apenas está disponível em páginas JSPs que definem o atributo isErrorPage da diretiva page para true.

Como exemplo, vamos considerar uma aplicação que receba parâmetros de um formulário. Ao analisar esses padrâmetros, um erro pode ocorrer, como por exemplo, conversão do tipo de dado. Se um erro ocorrer, uma página específica pode ser apresentada. Nessa página, o erro é tratado.

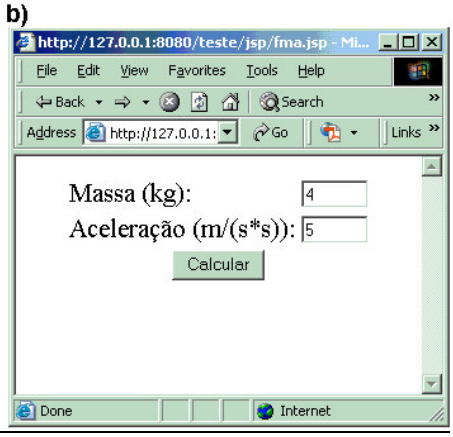
Arquivo fma.jsp

```
<html>
<body>
<center>
<form action="calcular.jsp" method="post">
<table>
<tr><td>Massa (kg):</td><td><input type="text"
name="m" value="" size="5"></td></tr>
<tr><td>Aceleração (m/(s*s)):</td><td><input
type="text" name="a" value="" size="5"></td></tr>
<tr><td colspan="2" align="center"><input
type="submit" value="Calcular"/> </td></tr>
</table>
</body></html>
```

a)



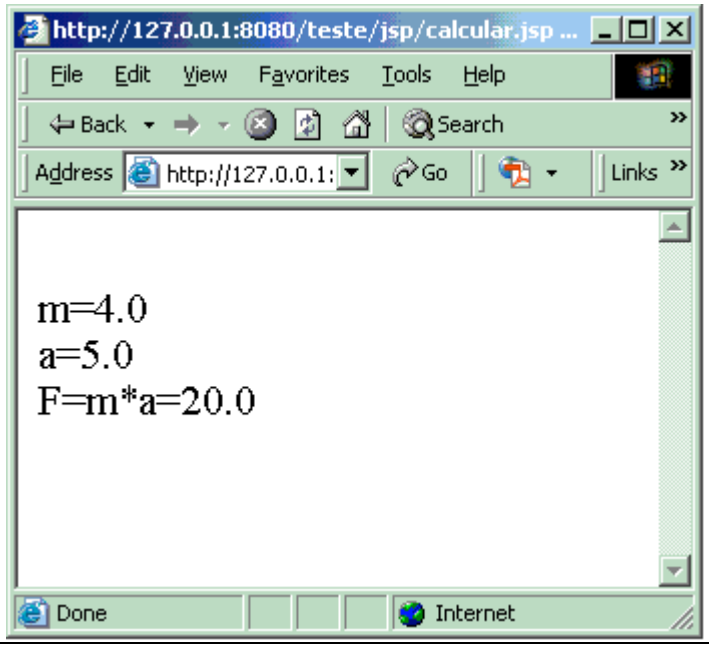
b)



Arquivo calcular.jsp

```
<%@page errorPage="erro.jsp" %>
<html>
<body>
<% double m = Double.parseDouble((String)
request.getParameter("m"));
   double a = Double.parseDouble((String)
request.getParameter("a"));

   out.println("<br>m="+ m);
   out.println("<br>a=" + a);
   out.println("<br>F=m*a=" + (m*a));
%>
</body>
</html>
```



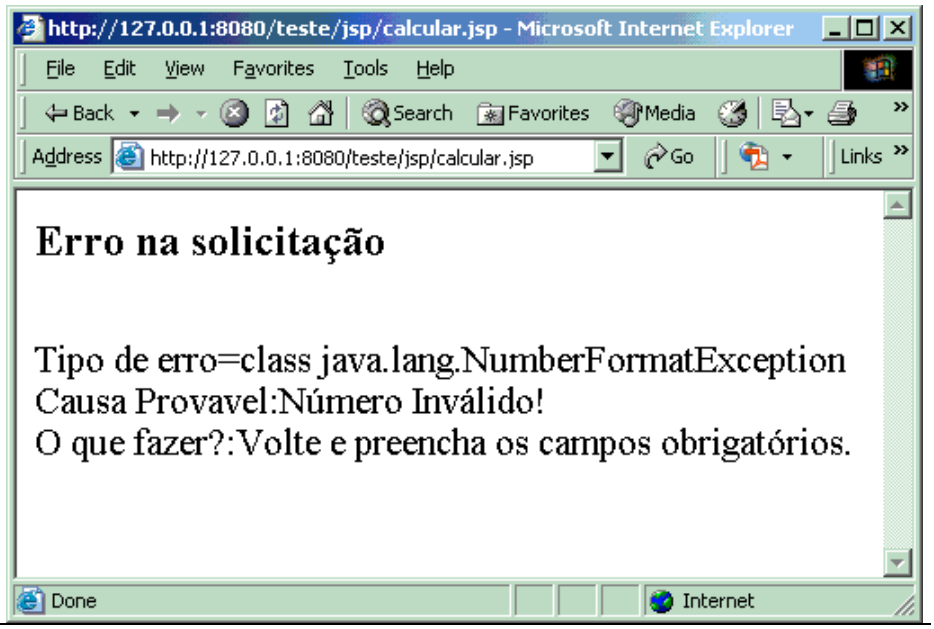
Arquivo erro.jsp

```
<%@page isErrorPage="true"%>
<html>
<body>
<h3> Erro na solicitação </h3>
<%
  String tipo ="";
  String causa="Desconhecida";
  String oQueFazer="Não definido !";
%>
<%
   if (exception != null)
    { tipo = exception.getClass().toString();

      if
(tipo.indexOf("java.lang.NullPointerException")>=0)
      { causa = "Faltou informar um dado!";
        oQueFazer="Volte e preencha os campos
obrigatórios.";
      }

      if
(tipo.indexOf("java.lang.NumberFormatException")>=0)
      { causa = "Número Inválido!";
        oQueFazer="Volte e preencha os campos
obrigatórios.";
      }
```

```
        out.println("<br>Tipo de erro="+tipo    );
        out.println("<br>Causa Provavel:" + causa);
        out.println("<br>O que fazer?:" + oQueFazer);
      }
%>
</body>
</html>
```



5. Exercícios:

1) Criar uma aplicação JSP com a seguinte estrutura:
   - index.html formulário que permite que o usuário pode registre os seus dados pessoais tais como: nome, data de nascimento, sexo, endereço, grau de escolaridade {ensino fundamental, ensino médio, graduação, especialização, mestrado e doutorado}.
   - processa.jsp: aplicação que exibe os dados fornecidos.

2) Criar uma aplicação JSP com a seguinte estrutura:
   - index.html: permite que o usuário entre com três números, a saber: a, b e c.
   - processa.jsp: aplicação que exibe o valor das raízes x1 e x2, considerando que a, b e c são coeficientes de uma equação de segundo grau $a.x^2 + bx + c = 0.$
3) Criar uma aplicação JSP que permita que o usuário digite um texto em uma TextArea, selecione o tipo de fonte {Arial, Times, courier ou System}, o estilo {itálico, negrito e sublinhado } e clique no botão enviar. Um novo arquivo JSP é formado com o texto digitado sendo apresentado com a configuração definida pelo usuário.

Referência Bilbiográfica:

[1] (Fields, D. K., Kolb, M.A.) Desenvolvendo na Web com Java Server Pages. Rio de Janeiro. Editora Ciência Moderna LTDA (2000). ISBN 85-7393-00-0. 558p.

[2] (Pekowsky, L.) JavaServer Pages, Second Edition. Addison Wesley. August 15, 2003. ISBN : 0-321-15079-1 . 368p.

[3] (Santos, R. R.) Java na Web – Programando Sites Dinâmicos. Rio de Janeiro. Editora Axcel Books do Brasil (2007). ISBN: 85-7323-159-9. 373p.