

# Programação Java para Web

## Projeto UERJ-FAPERJ

Prof. Austeclynio Pereira  
2023

# Bibliografia

- Java para Web com Servlets, JSP e EJB – Budi Kurniawan
- Murach's Java Servlets and JSP – Andrea Steelman
- Head First Servlets & JSP – Bryan Basham, Kathy Sierra e Bert Bates
- ◆ Enterprise Java Developer's Guide – S. Narayanan, Junhe Liu
- ◆ The J2EE Tutorial – Sun Microsystems
- ◆ Core Servlets and JavaServer Pages – Vol I – Marty Hall
- ◆ Como o Tomcat Funciona – Budi Kurniawan e Paul Deck

# Programação Java para Web

- ♦ Início em 28/02/2023;
- ♦ Término em 11/04/2023;
- ♦ Horário: das 16h às 19h;
- ♦ Dias da semana: 3as e 5as;
- ♦ Conteúdo do curso:  
**Será enviado para o e-mail dos participantes**

Avaliação - Projeto ao final do curso;

# Foco do Curso

- ◆ Capacitar para desenvolver aplicações back-end em Java;
- ◆ Apresentando:
  - Fundamentos da arquitetura cliente servidor.
  - Padrão MVC.
  - Desenvolvimento de servlets.
  - Autenticação.
  - Gerenciamento de sessões.
  - Conexão com o banco de dados MySql.
  - Ajax Json.

# Ferramentas para o desenvolvimento dos sites

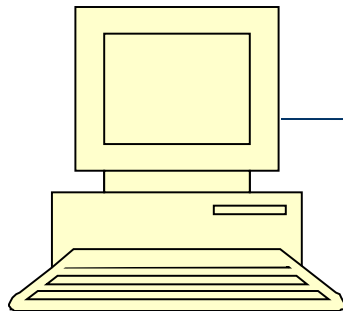
- ◆ IDE NetBeans ou Eclipse ou JCreator.
- ◆ MySql a partir da versão 5.7.
- ◆ TomCat a partir da versão 7.
- ◆ Java a partir da versão 1.8.0\_91.

# Modelo cliente-servidor

- ◆ Principal padrão utilizado na Internet.
- ◆ Os clientes requisitam os serviços e o servidor realiza os serviços solicitados pelos clientes.
- ◆ Necessidade de uma rede de computadores, de um protocolo de comunicação e de um mecanismo de localização.

# Modelos Arquiteturais – uma camada

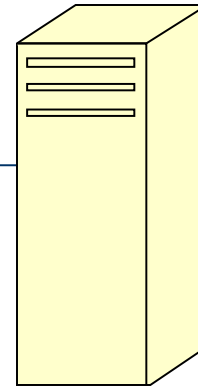
**Máquina Cliente**



**Web Browser**



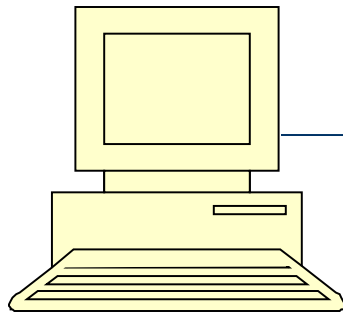
**Servidor Web**



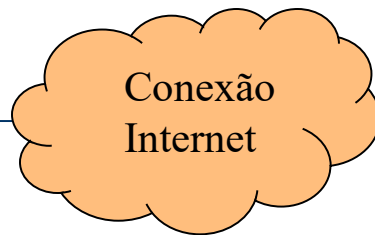
- **Apresentação**
- **Regras do negócio**
- **Persistência**

# Modelos Arquiteturais – duas camadas

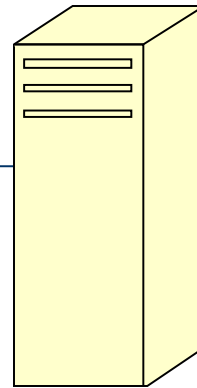
**Máquina Cliente**



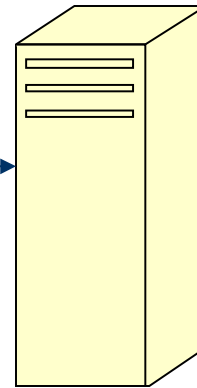
**Web Browser**



**Servidor Web**



**Servidor DataBase**

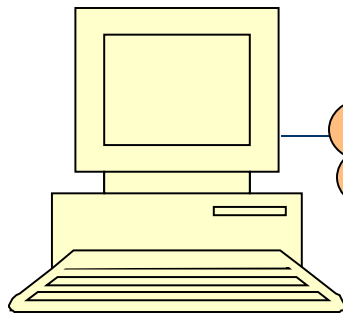


- **Apresentação**
- **Regras do negócio**

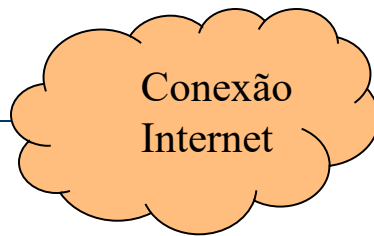


# Modelos Arquiteturais – três camadas

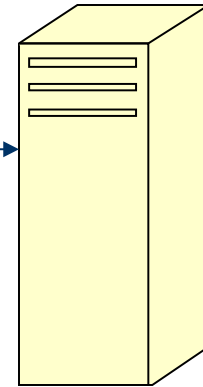
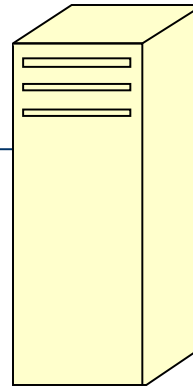
**Máquina Cliente**



**Web Browser**



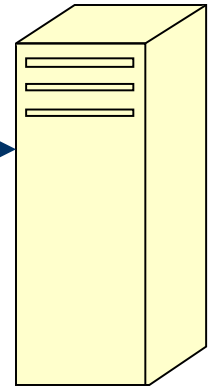
**Servidor Web**



**Servidor de Aplicações**

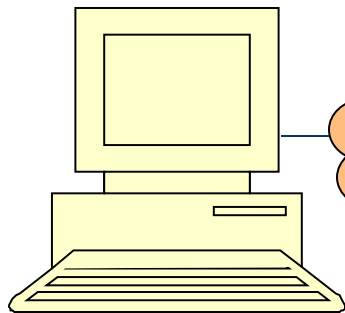


**Servidor DataBase**

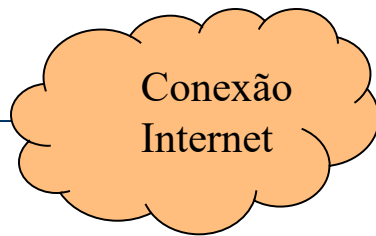


# Modelos Arquiteturais – três camadas

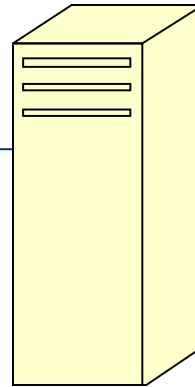
**Máquina Cliente**



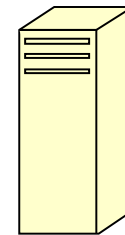
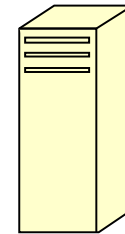
**Web Browser**



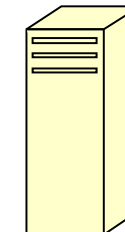
**Servidor Web**



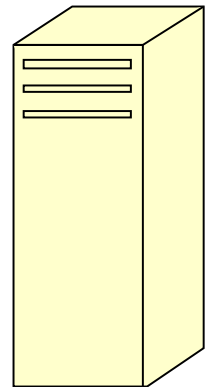
**Autenticação**



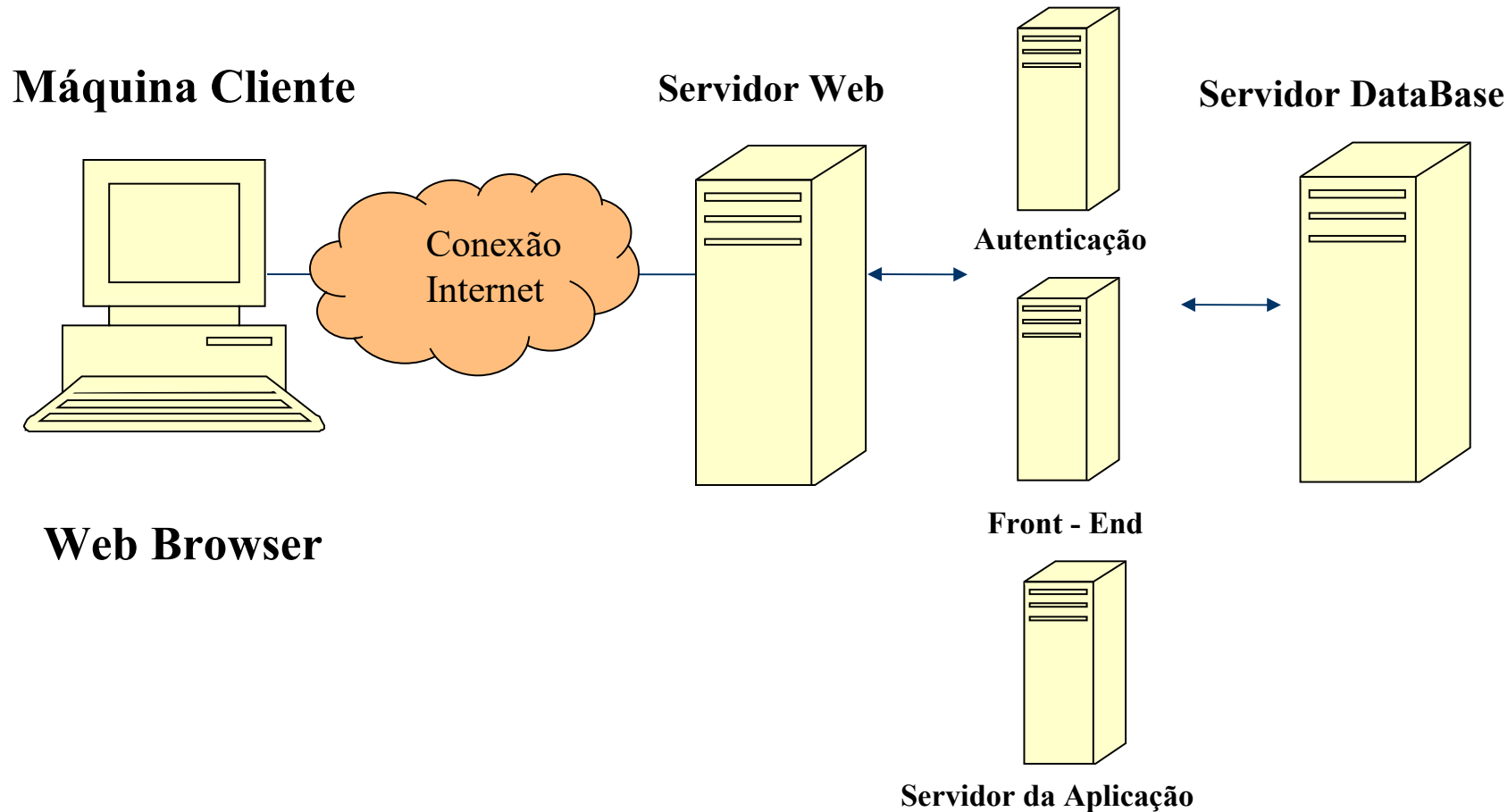
**Front - End**



**Servidor DataBase**



**Servidor da Aplicação**



# Ciclo Request Response

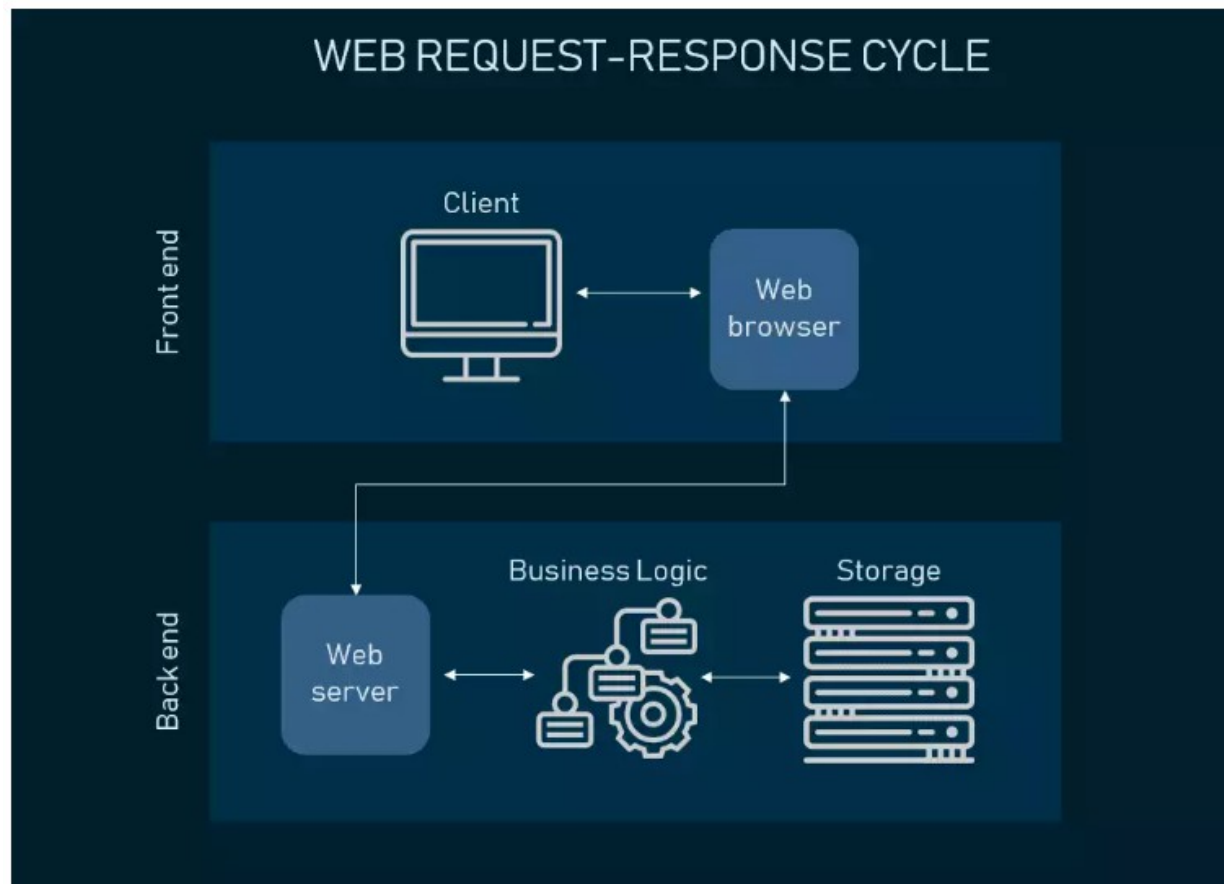
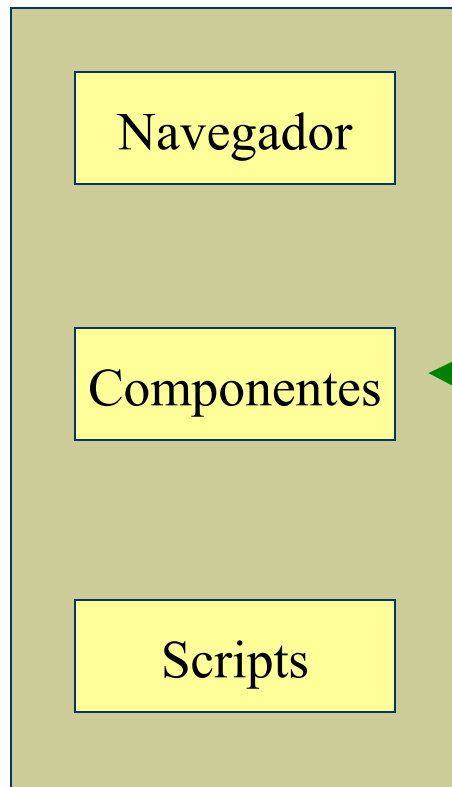


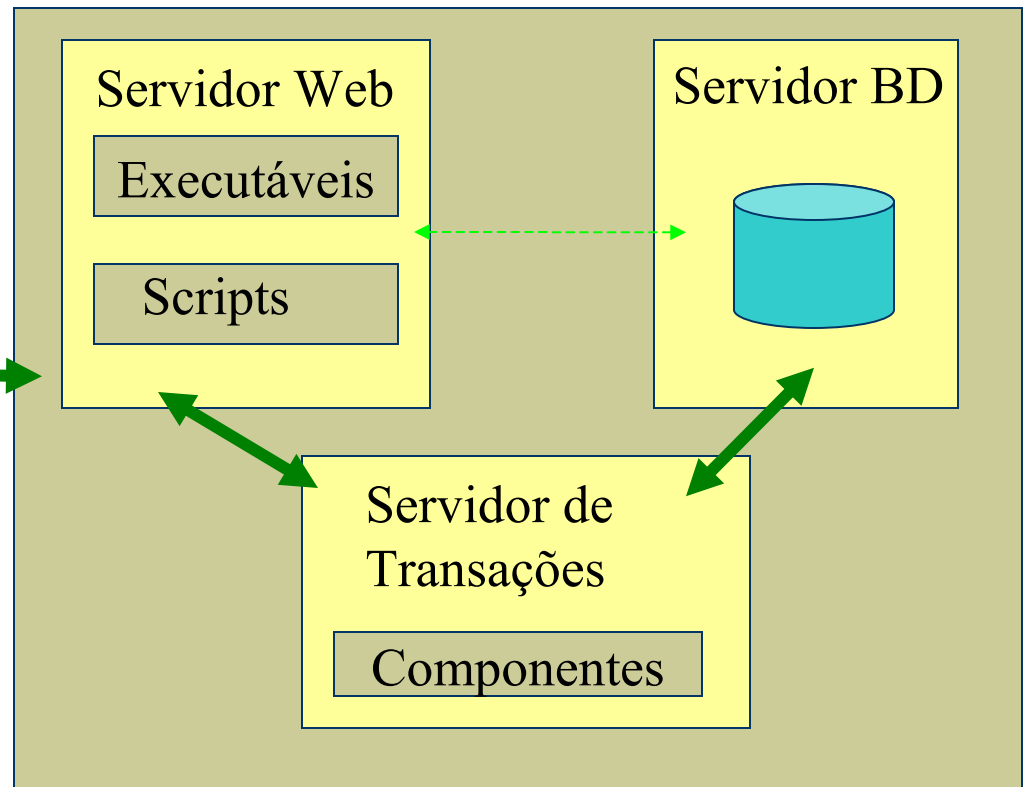
Figura extraída de <https://www.altexsoft.com/>

# Elementos típicos de uma aplicação *web*

## Cliente



## Servidor



# Elementos típicos de uma aplicação *web* - lado cliente

- ♦ **Scripts** – normalmente utilizados para validar dados de entrada. Diminui o número de requisições ao servidor. Ex: JavaScript.
- ♦ **Componentes** – podem conter parte da lógica do negócio, desonerando o servidor. Exs: Applets e Active-X.

# Elementos típicos de uma aplicação *web* - lado servidor

- ♦ **Common Gateway Interface** – módulo executável que produz páginas e informações para o cliente. Cada invocação gera um outro processo.
- ♦ **Scripts** – gera uma página HTML para o cliente ou transfere a página para outro servidor. Podem misturar lógica do negócio com apresentação. Exs.: ASP, JSP, PHP, Angular e React.
- ♦ **Componentes** – módulos executáveis invocados por scripts ou por outros módulos executáveis. Exs.: COM+ e EJB.
- ♦ **Executáveis** – executados em um mesmo processo, capacidade de gerenciar sessões, formulários e cookies.Ex.: Servlets.

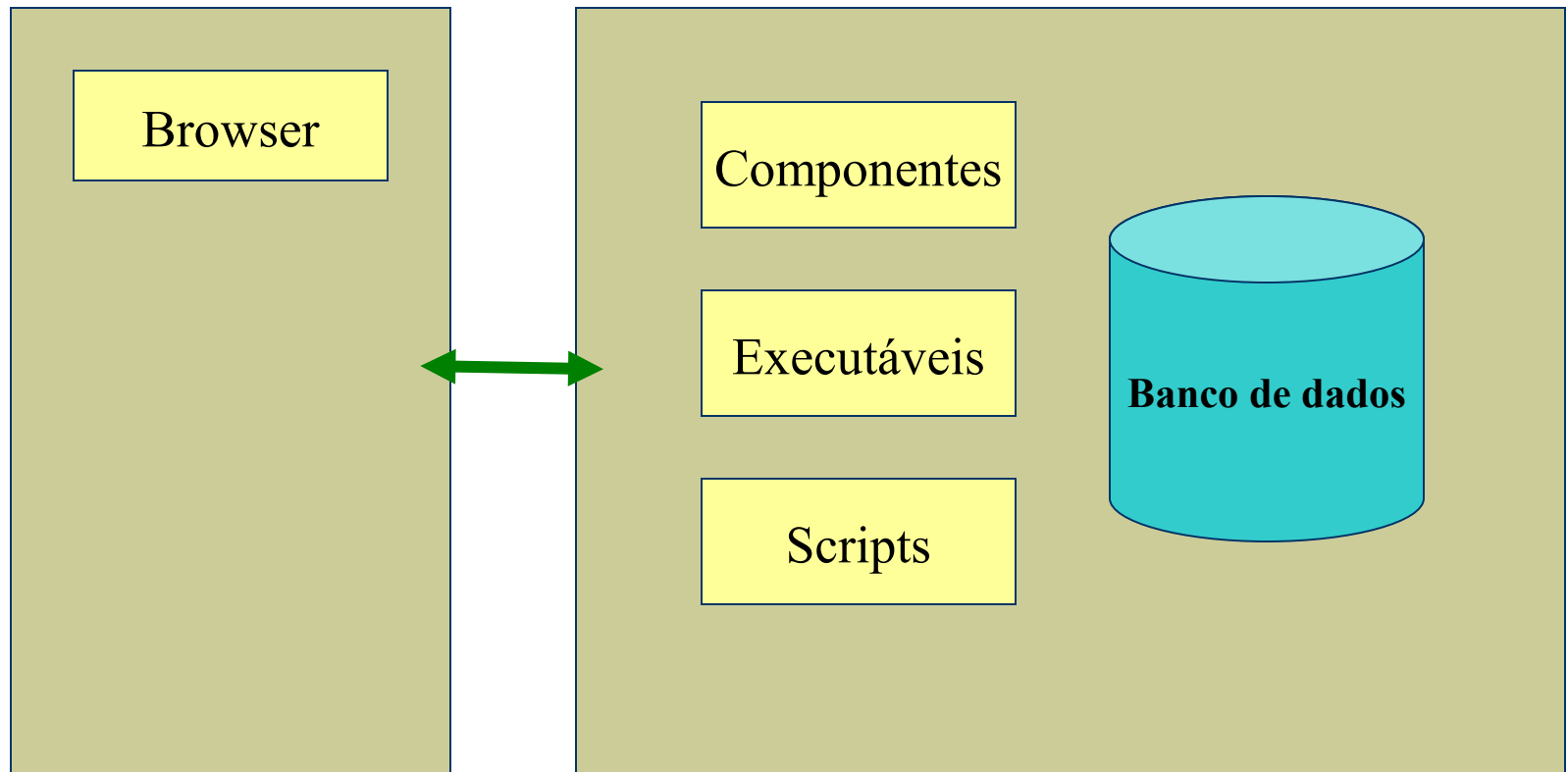
# Estilos Arquiteturais

- ♦ **Thin client** – utilização mínima dos recursos da máquina cliente, praticamente tudo é tratado pelo servidor.
- ♦ **Scripted client** – *scripts* na máquina cliente para a verificação de dados.
- ♦ **Thick client** – distribuição da lógica do negócio entre a máquina cliente e a máquina servidora.

# Elementos típicos da Internet thin client

**Cliente**

**Servidor**

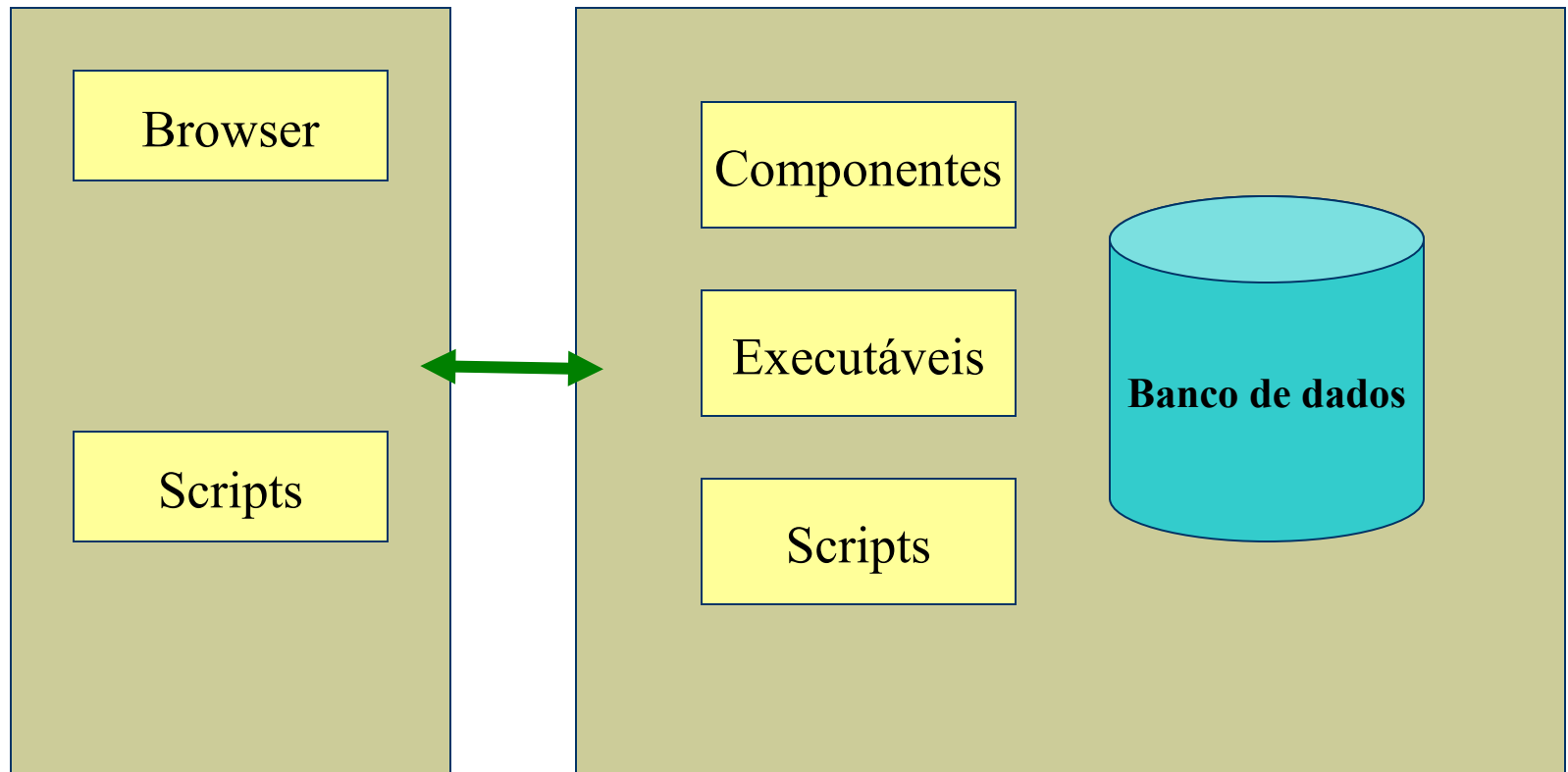




# Elementos típicos da Internet scripted client

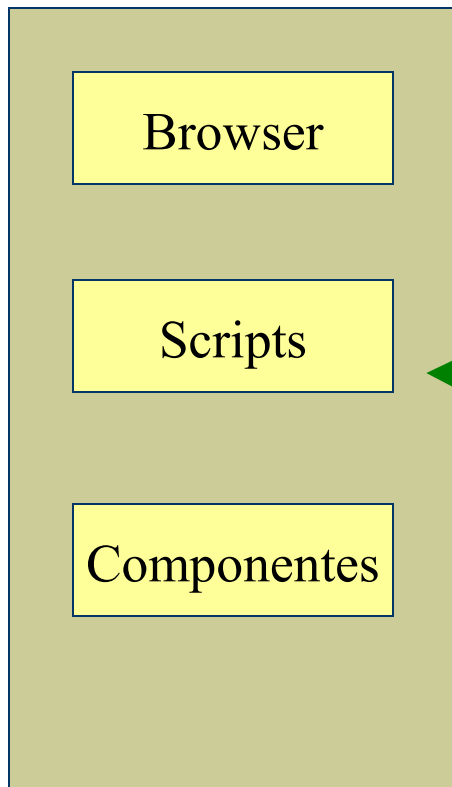
**Cliente**

**Servidor**

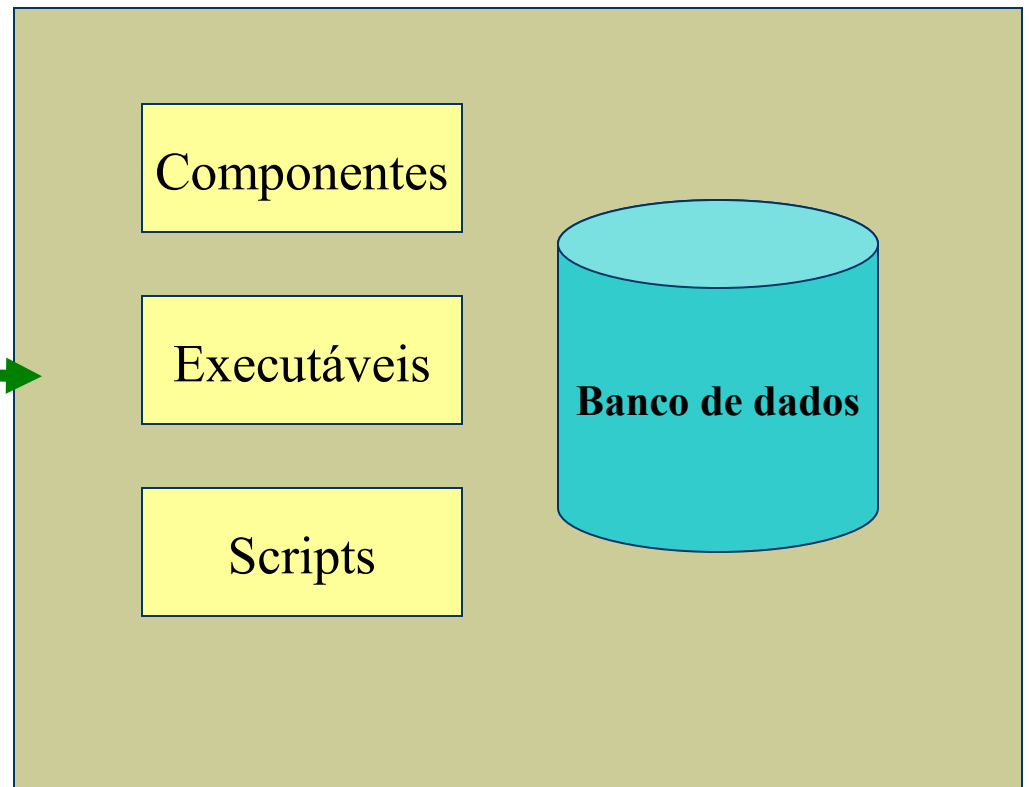


# Elementos típicos da Internet thick client

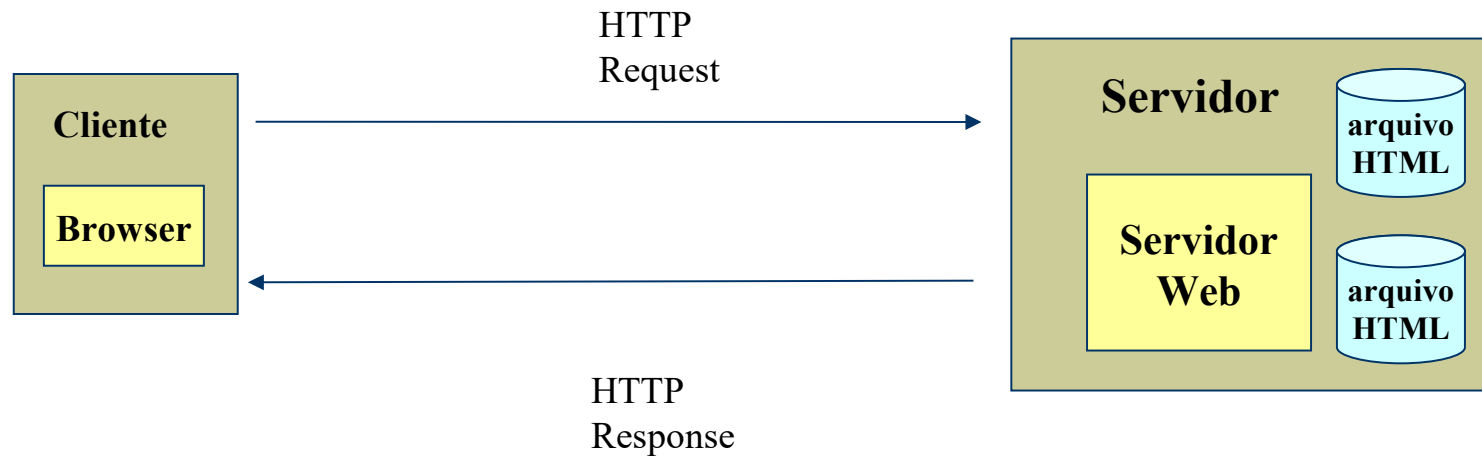
**Cliente**



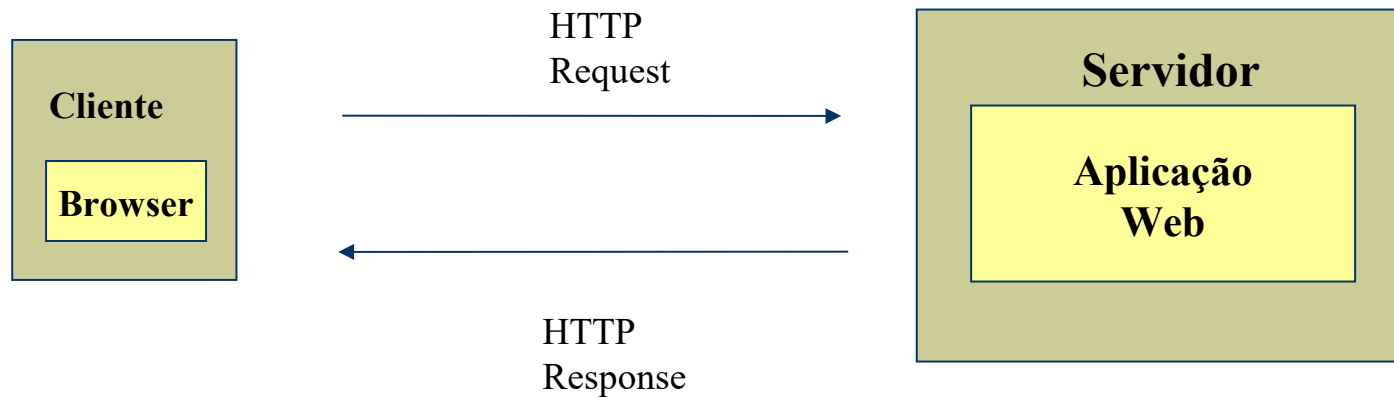
**Servidor**



# Como as páginas *web* estáticas trabalham



# Como as páginas *web* dinâmicas trabalham



# O protocolo HTTP

- ◆ Protocolo de rede situado em uma camada acima da TCP/IP.
- ◆ Possui características específicas para aplicações baseadas na Web.
- ◆ A estrutura de um diálogo do tipo HTTP é uma simples seqüência de operações *request/response*.
- ◆ O *web browser* faz o *request* e o *web server* responde.

## Elementos chaves de um *request*

- ◆ Ação a ser realizada. Representada por um dos métodos do HTTP.
- ◆ Página que desejamos obter acesso (URL).
- ◆ Parâmetros do formulário invocado.

## Elementos chaves de um *response*

- ◆ Código de retorno do *request*.
- ◆ Tipo do conteúdo retornado(texto, figura, HTML etc).
- ◆ Conteúdo(o texto HTML, a figura etc).

# Retornando um HTML

## Cabeçalho HTTP

```
<html>
<head>
...
</head>
<body>
<img src=...>
</body>
</html>
```

Conteúdo  
HTTP

Gera um outro *request*.

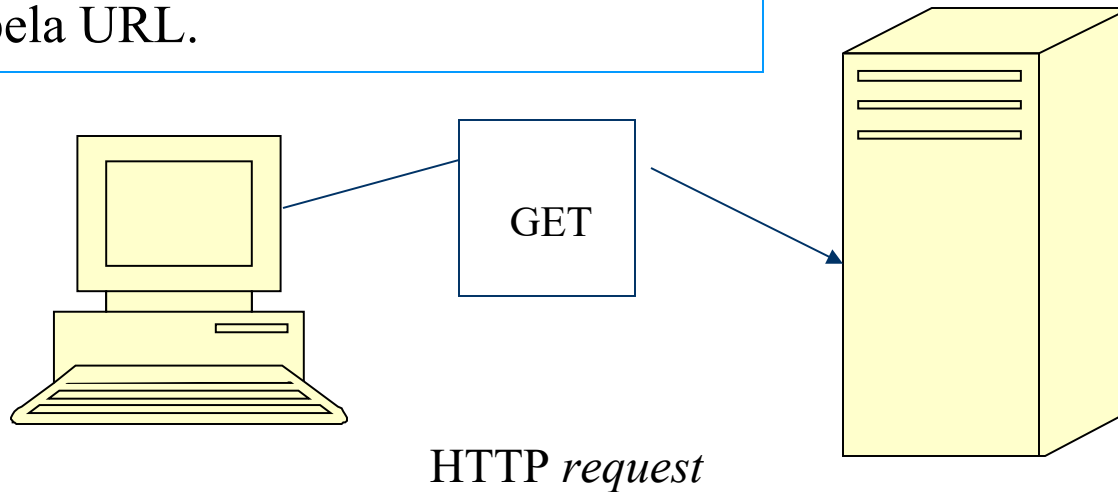


# Métodos utilizados pelo *request*

- ♦ Um *request* solicita serviços ao *web server* através de métodos do protocolo HTTP.
- ♦ Métodos do HTTP: **GET**, **POST**, HEAD, TRACE, PUT, DELETE, OPTIONS e CONNECT.
- ♦ O *Web browser* envia um HTTP **GET** para o servidor solicitando um recurso. Pode ser: uma página HTML, um JPEG, um PDF etc.
- ♦ O **POST** pode solicitar um recurso e, ao mesmo tempo, enviar um formulário com dados.
- ♦ O **GET** envia dados pela URL!

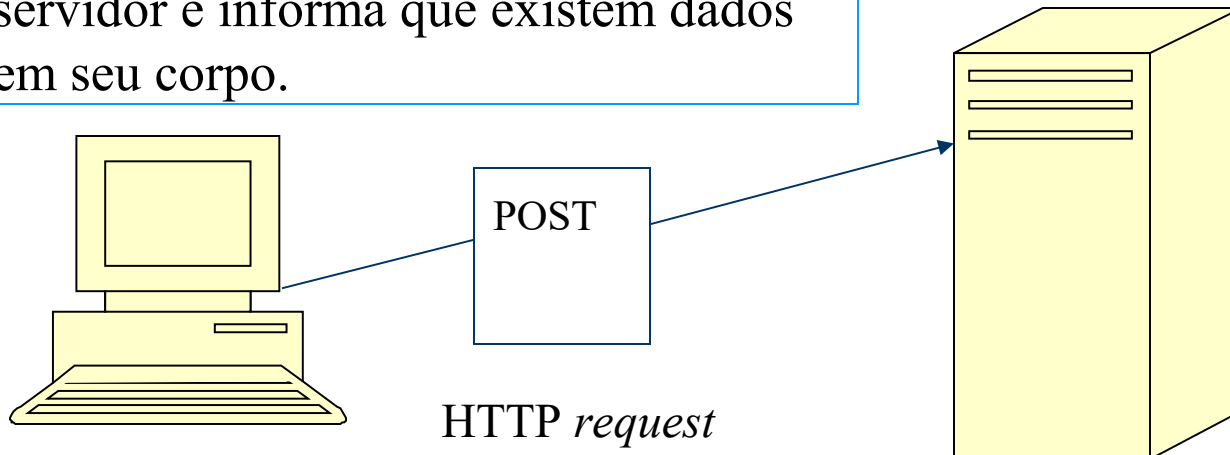
# HTTP *request* GET

O GET pede uma página ao servidor passando os parâmetros Agência=3080 e Conta=123456 pela URL.



# HTTP *request* POST

O POST pede uma página ao servidor e informa que existem dados em seu corpo.



# HTTP *response*

- ◆ Composto de um cabeçalho e de um corpo.
- ◆ O cabeçalho serve para informar ao *web browser* :
  - Se a requisição foi ou não bem sucedida.
  - O tipo do conteúdo que está sendo passado(conhecido como MIME *type*).
- ◆ O corpo contém o conteúdo que será renderizado pelo *web browser*.

# Fluxo primário de uma operação *request response*

- ◆ Usuário seleciona uma URL.
- ◆ *Web browser* cria um HTTP GET *request*.
- ◆ O HTTP GET é enviado para o *Web server*.
- ◆ O *Web server* localiza a página solicitada.
- ◆ *Web server* gera um HTTP *response*.
- ◆ O HTTP *response* é enviado para o *Web browser*.
- ◆ O *Web browser* renderiza o HTML.

# Uniform Request Locator (URL)

◆ <http://www.nce.ufrj.br:80/concursos/login.html>

Protocolo


Nome do Servidor.  
Possui um IP address.

Porta da Aplicação.  
Default=80

Nome do recurso  
solicitado.  
Default=index.html

Caminho onde  
o servidor vai  
localizar o recurso.

Obs.: Caso seja utilizado o método GET  
a URL conterá os parâmetros que serão  
passados para o servidor.



# Que código Java escrevemos para a *web*?



- ◆ *Servlets*.
- ◆ JavaServer Pages(JSP).
- ◆ Classes de negócio.
- ◆ Classes de acesso ao banco de dados.

# *Servlets*

- ♦ Introduzidos pela Sun em 1996 com o propósito de acrescentar conteúdo dinâmico aos aplicativos *web*.
- ♦ Um *servlet* é uma classe Java executada por um *container*.
- ♦ Tem como benefícios: bom desempenho, portabilidade, rápido ciclo de desenvolvimento e robustez.



# *Servlet*

```
import javax.servlet.*;  
import javax.servlet.http.*;
```

```
public class PrimeiroServlet extends HttpServlet {
```

```
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response)  
        throws IOException, ServletException{
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println("<HTML>");  
        out.println("<HEAD>");  
        out.println("<TITLE>Java para web com servlet e JSP</TITLE>");  
        out.println("</HEAD>");  
        out.println("<BODY>");  
        out.println("BemVindo ao curso de Java para web");  
        out.println("</BODY>");  
        out.println("</HTML>");
```

```
    }
```

```
}
```

# JavaServer Page

## ◆ Exemplo 1: HTML puro

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
Java para web  
</BODY>  
</HTML>
```

## ◆ Exemplo 2: HTML + código Java = JSP

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<%  
    out.println("Java para web");  
%>  
</BODY>  
</HTML>
```

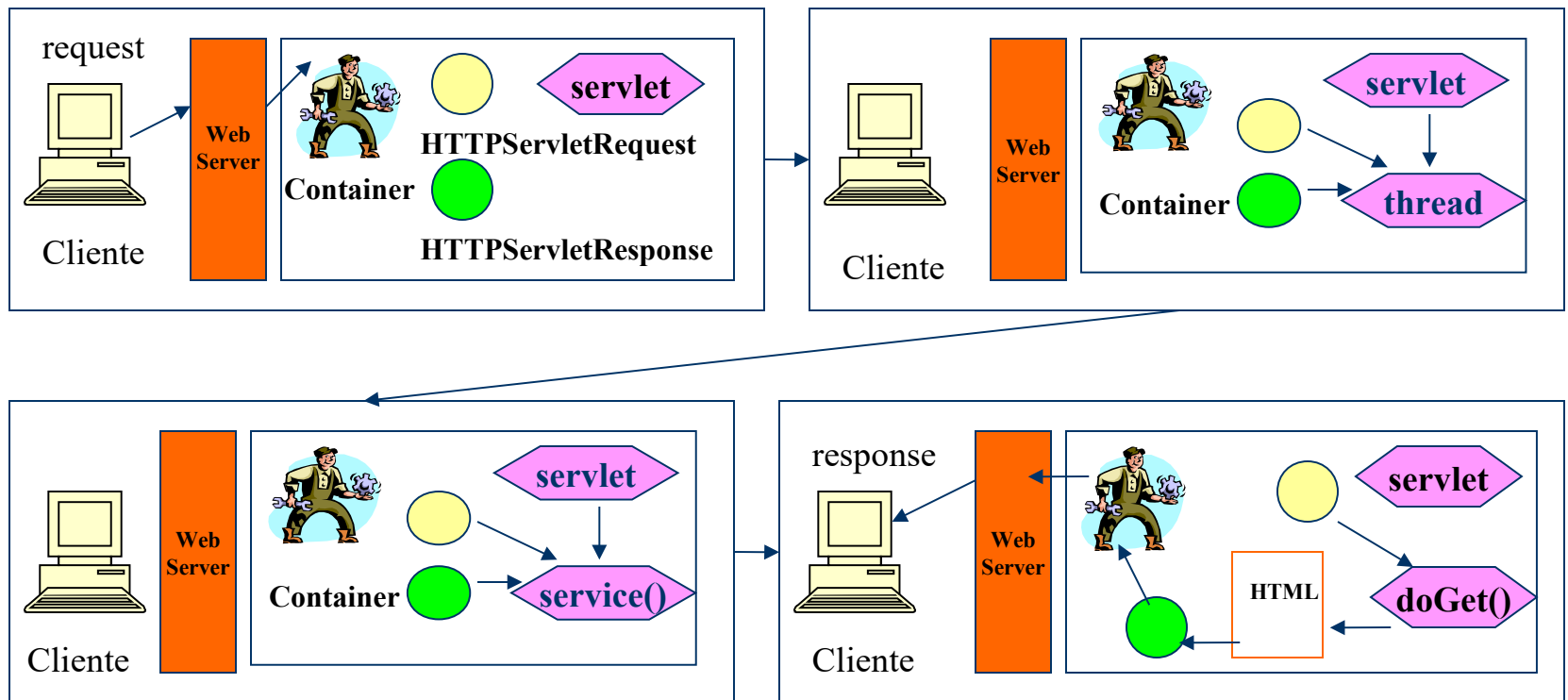
# Container

- ♦ O *web server* não sabe tratar páginas dinâmicas.
- ♦ Necessidade de um *container* para abrigar *servlets* e JSPs.
- ♦ O TomCat é um dos mais populares *containers* do mercado.
- ♦ O *web server* solicita ao *container* as páginas dinâmicas.
- ♦ *Servlets* e JSPs não possuem um método `main()`. São carregados pelo *container*.

# Container - Propósitos

- ◆ Suporte à comunicação de alto nível. Isenta os desenvolvedores de *servlets* de escreverem *sockets*.
- ◆ Administra o ciclo de vida dos *servlets*.
- ◆ Suporte à múltiplas *threads*.
- ◆ Suporte à segurança. Transparente para o desenvolvedor.
- ◆ “Transforma” um JSP em um *servlet*.
- ◆ Pode atuar também como *web server*.

# Container – Tratando requests



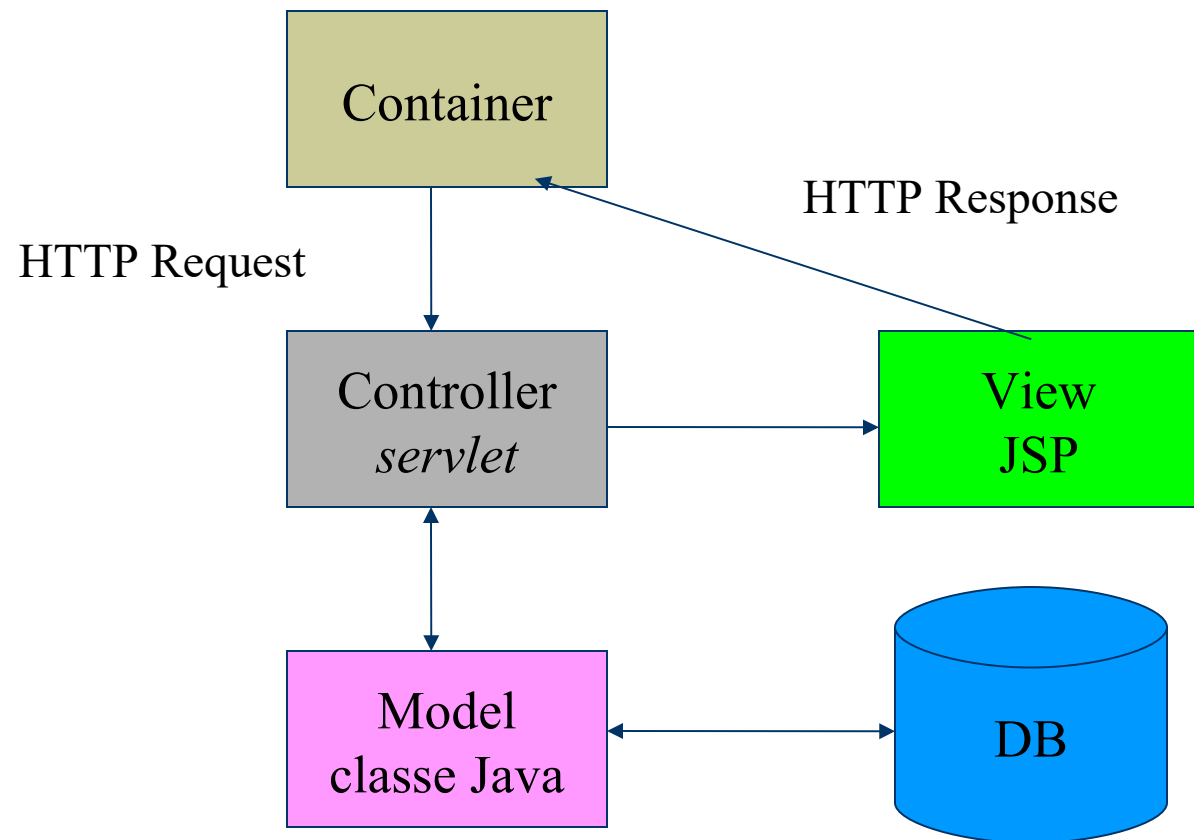
# Uma aplicação *web* em Java



# Model View Controller (MVC)

- ◆ Padrão de projeto empregado nas aplicações *web*.
- ◆ Separa a lógica do negócio da apresentação.
- ◆ A lógica fica em classes Java específicas.
- ◆ Possibilita o reuso destas classes por outros aplicativos.
- ◆ Divide mais claramente as responsabilidades:
  - A classe é o Model.
  - O *servlet* é o Controller.
  - A JSP é a View.

# Model View Controller (MVC)





# Model View Controller (MVC)

- ♦ O **Controller** recebe os dados do cliente, critica-os, e os repassa ao Model.
- ♦ O **Model** aplica as regras do negócio e retorna a informação para quem as solicitou.
- ♦ A **View** obtém o estado do Model, repassado pelo Controller, apresentando-o ao cliente.

# Criando uma aplicação *web*

- ◆ São necessários 4 passos:
  1. Definir as páginas que serão vistas pelo cliente.
  2. Criar o ambiente de desenvolvimento.
  3. Criar o ambiente de produção/distribuição.
  4. Realizar os testes.

Obs.: Presume-se, por óbvio, que todas as modelagens pertinentes já foram realizadas.

# Aplicação *Sugestão Musical*

## Visão do cliente

nonononononononononononononononononnn

Selecione o estilo musical preferido:

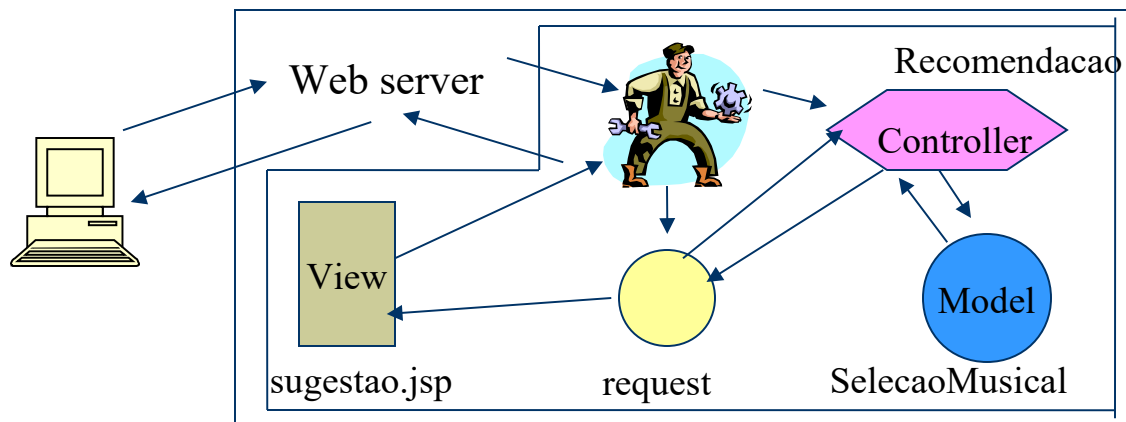
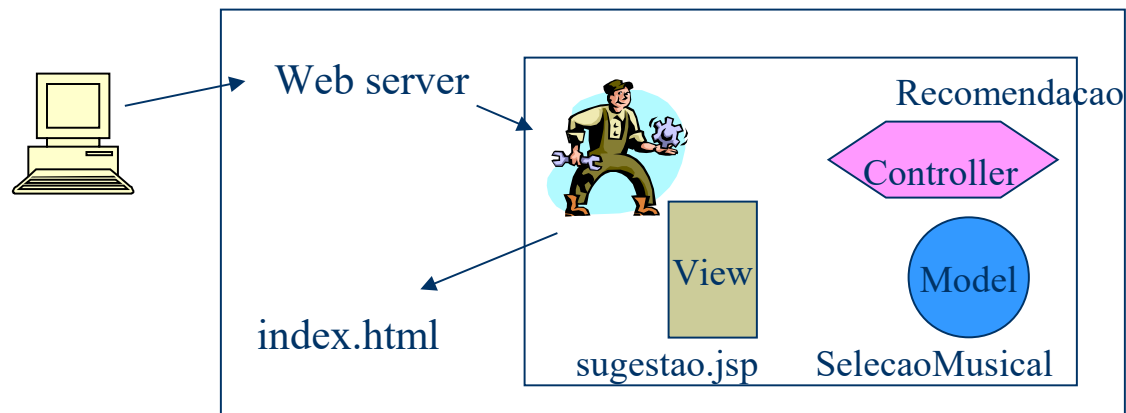
**Enviar**

nononononononononononononononononnn

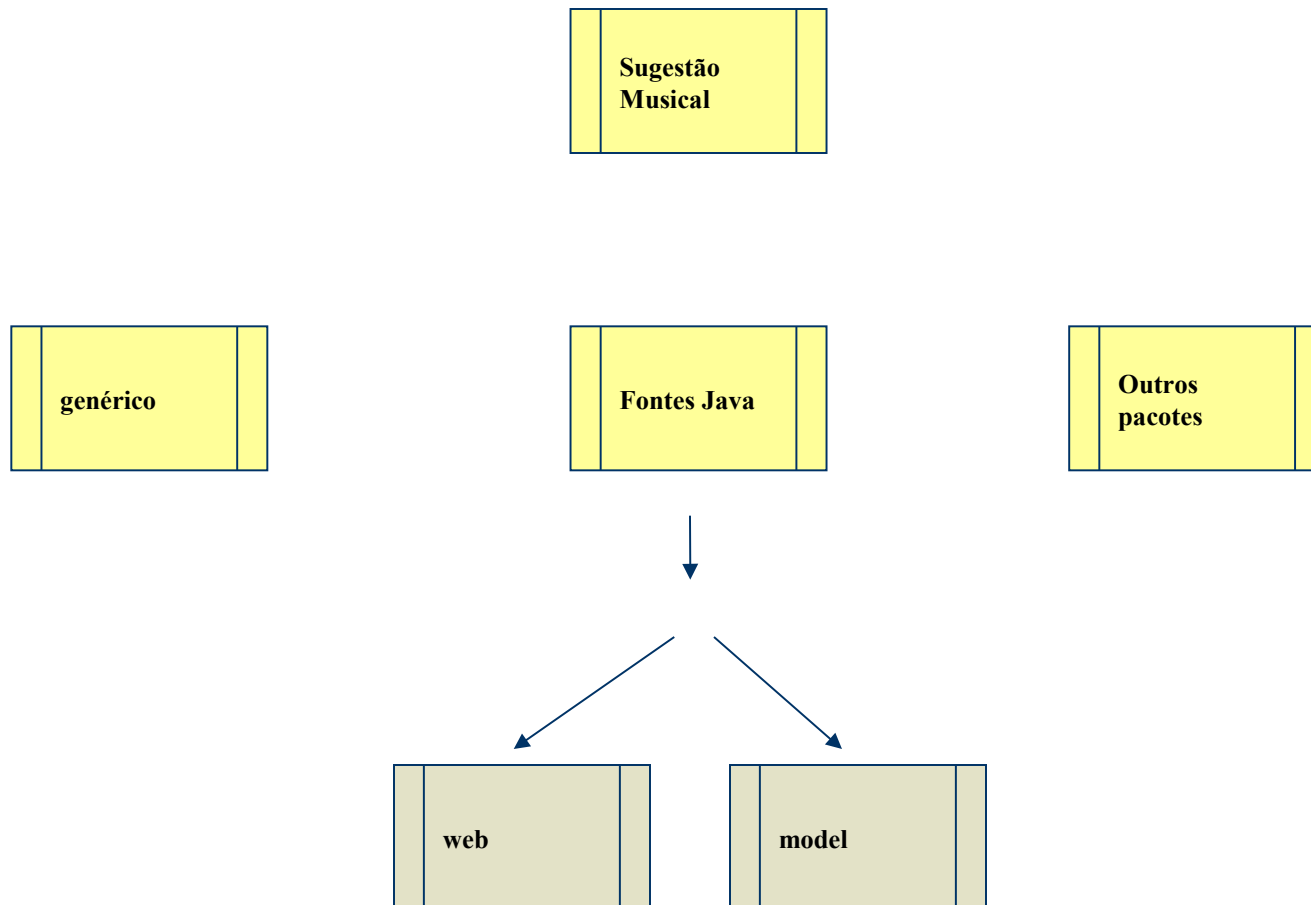
Recomendação musical :

Led Zeppelin  
U2  
The Who  
Yes

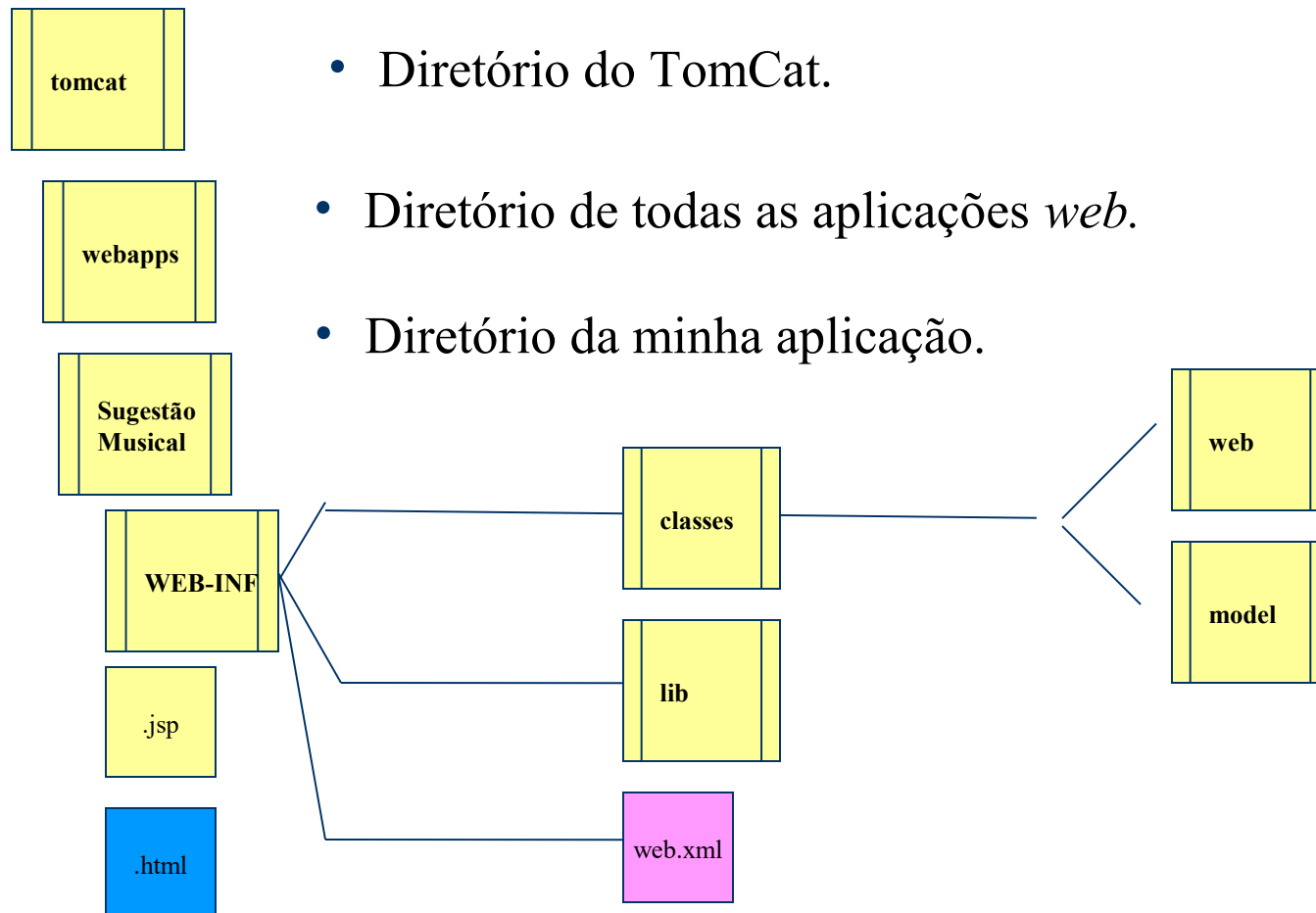
# Arquitetura do aplicativo



# Ambiente de desenvolvimento



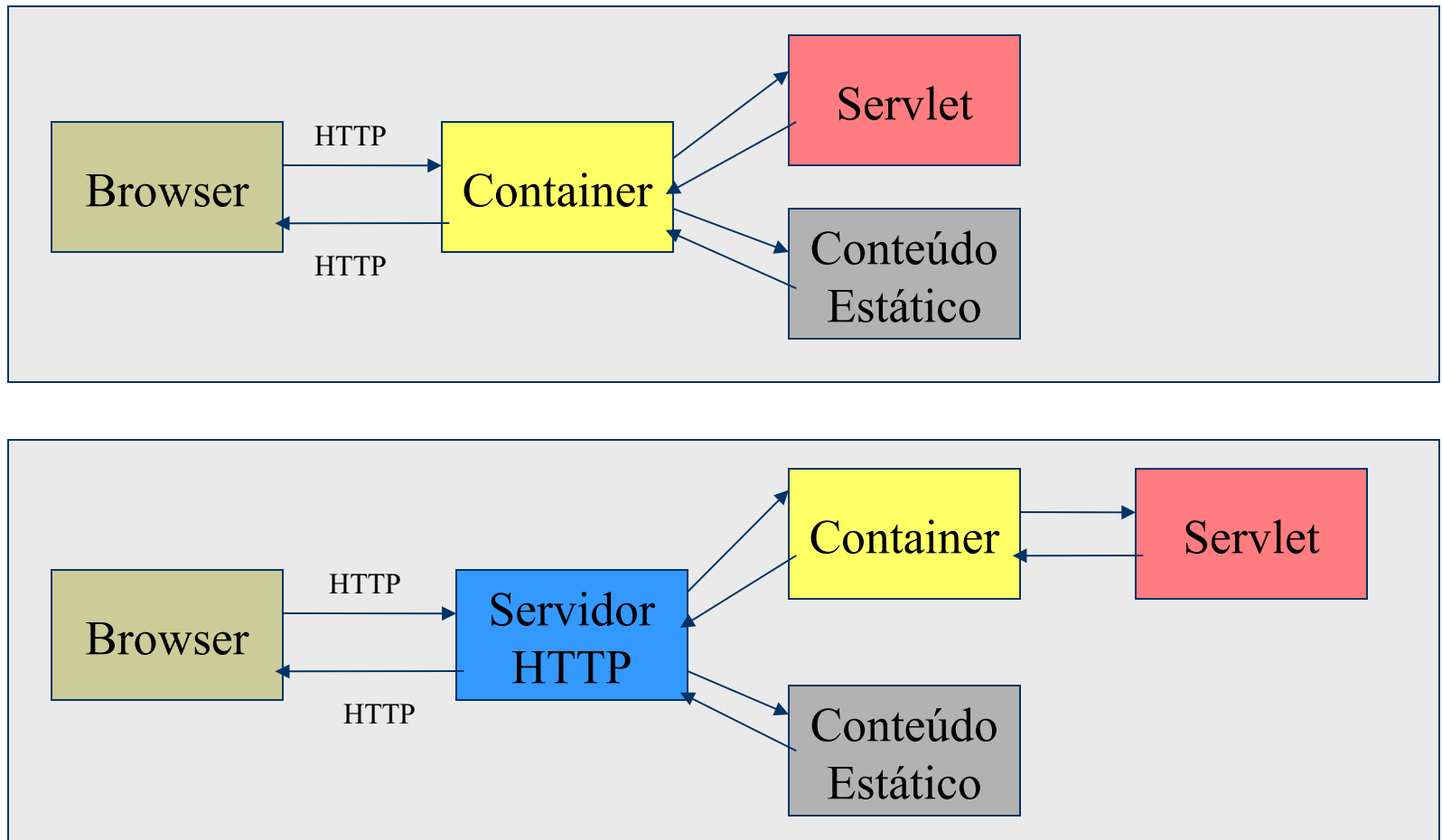
# Ambiente de testes



## Características do sub-diretório WEB-INF

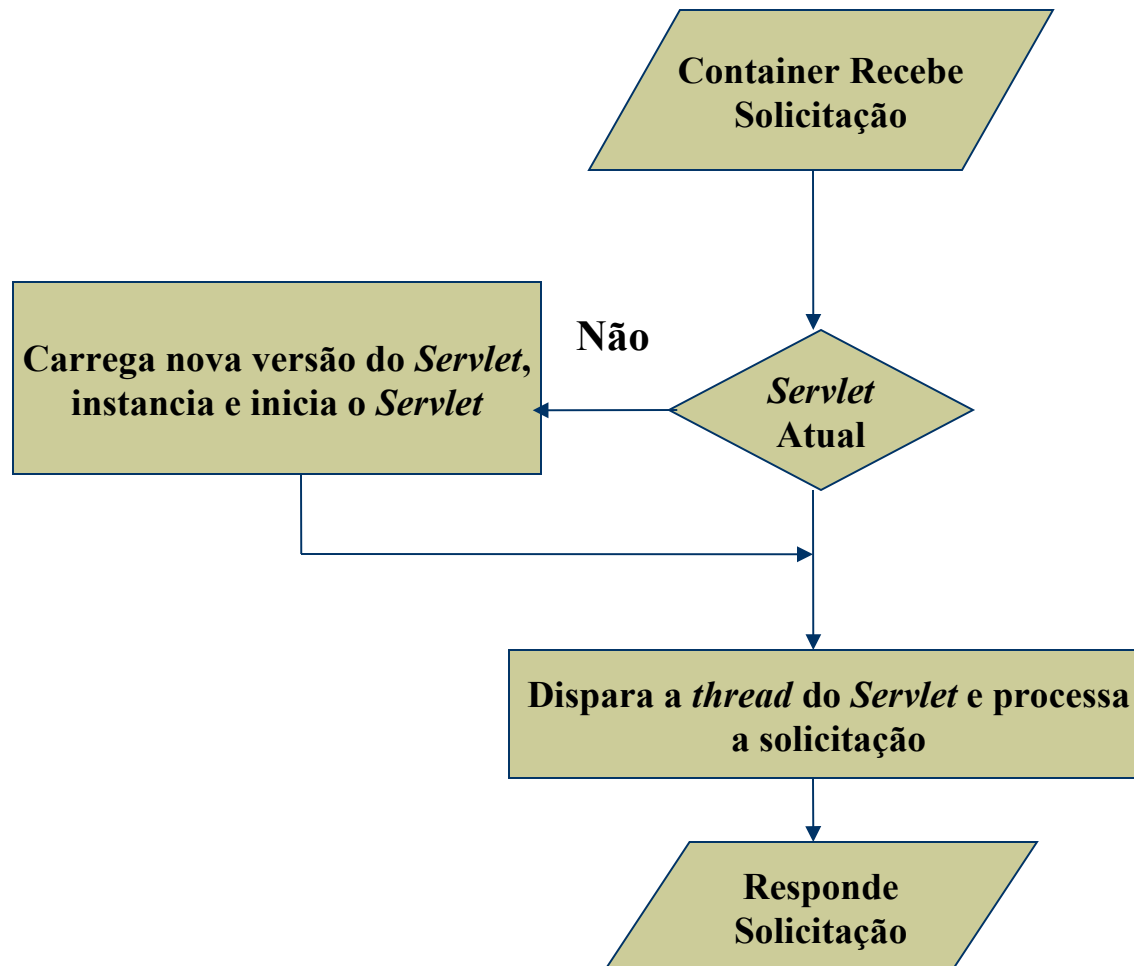
- ♦ Não fica visível para o *web browser* cliente.
- ♦ Residência do arquivo descritor `web.xml`.
- ♦ Os *servlets* residirão no sub-diretório **classes**.
- ♦ As classes que refletem as regras do negócio também residirão no sub-diretório **classes**.

# Ativando um *servlet*





# A carga de um *servlet*



# A distribuição descritiva

- ◆ É um documento XML que contém informações que descrevem os servlets.
- ◆ Denominado **web.xml**.
- ◆ Possui a *tag* **web-app** que descreve todos os servlets da aplicação.
- ◆ Associados a cada servlet têm as *tags* **<servlet-name>** , **<servlet-class>** e **<servlet-mapping>**.
- ◆ **<servlet-name>** é o nome que o Tomcat irá referenciar o servlet.
- ◆ **<servlet-class>** é o nome efetivo do servlet sem a extensão *.class*.

## A distribuição descritiva <servlet-mapping>

- ◆ Associa um URL a cada servlet.
- ◆ Evita que o nome do servlet seja apresentado no *web browser*.
- ◆ Utiliza a *tag* <**url-pattern**>.
- ◆ <**url-pattern**> define um nome que estará associado ao servlet desejado.

# A distribuição descritiva

```
<web-app>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/loginservlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>MinhaCompra</servlet-name>
    <servlet-class>CompraServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MinhaCompra</servlet-name>
    <url-pattern>/minhacompra</url-pattern>
  </servlet-mapping>
</web-app>
```

# Como invocar um *servlet* ?

- ◆ Invocando um servlet :
  - <http://localhost:8080/minhaapp/loginservlet>
  - <http://www.dcc.ufrj.br/minhaapp/minhacompra>
- ◆ Form *tags* para invocar um servlet:
  - `<form action=“../loginservlet” method=“get”>`
  - `<form action=“../minhacompra” method=“post”>`

# Sugestão Musical

## A página *index.html*

```
<html><body>
<h1 align="center" >Selecione o estilo musical preferido:</h1>
<form method="POST" action="EscolhaGrupo">
  <select name="estilo" size="1">
    <option> Rock
    <option>Samba
    <option> Opera
    <option> MPB
  </select><br>
  <center>
    <input type="SUBMIT" value="Enviar" >
  </center>
</form>
</body>
</html>
```

# Sugestão Musical

## O *servlet* Recomendacao

```
package web;
import model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Recomendacao extends HttpServlet{
    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {

        response.setContentType("text/html;charset=UTF-8");

        String estilo = request.getParameter("estilo");
        SelecaoMusical selecao = new SelecaoMusical();
        ArrayList<String> retorno = selecao.getLista(estilo);
        request.setAttribute("listaRecomendada" , retorno);

        RequestDispatcher vista = request.getRequestDispatcher("/jsp/sugestao.jsp");

        vista.forward(request, response);

    }
}
```

# Sugestão Musical

## A classe SelecaoMusical

**package model;**

**import java.util.\*;**

**public class SelecaoMusical{**

**public ArrayList getLista(String estilo){**

**ArrayList<String> grupos = new ArrayList<String>();**

**if (estilo.equals("Rock")){**

**grupos.add("Led Zeppelin");**

**grupos.add("The Who");**

**grupos.add("U2");**

**grupos.add("Yes");**

**}**

**else if (estilo.equals("Samba")){**

**grupos.add("Zeca Pagodinho");**

**grupos.add("Fundo de Quintal");**

**grupos.add("Dona Ivone Lara");**

**grupos.add("Martinho da Vila");**

**}**



# Sugestão Musical

## A classe SelecaoMusical

```
else if (estilo.equals("Opera")){
    grupos.add("Placido Domingo");
    grupos.add("Luciano Pavarotti");
    grupos.add("Jose Carreras");
    grupos.add("Enrico Caruso");
}
else {
    grupos.add("Chico Buarque");
    grupos.add("Milton Nascimento");
    grupos.add("Ellis Regina");
    grupos.add("Gonzaguinha");
}

return grupos;

}
```

# Sugestão Musical

## A JSP sugestao

```
<%@ page import="java.util.*" %>
<html>
<body>
  <h1 align =center="center"> Recomenda-se</h1>
  <% ArrayList<String> estilo =(ArrayList) request.getAttribute("listaRecomendada");
    for (String musica:estilo){
      out.print("<br>" + musica);
    }
  %>
</body>
</html>
```

# Sugestão Musical

## O descritor *web.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE web-app
```

```
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

```
  "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app>
```

```
  <servlet>
```

```
    <servlet-name>Musicas</servlet-name>
```

```
    <servlet-class>web.Recomendacao</servlet-class>
```

```
  </servlet>
```

```
  <servlet-mapping>
```

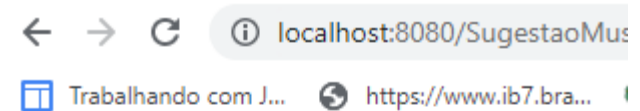
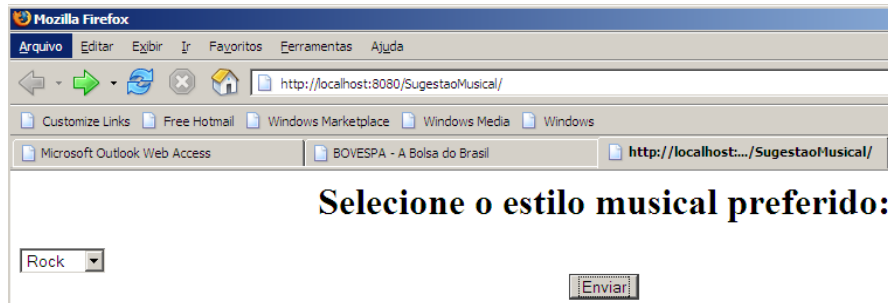
```
    <servlet-name>Musicas</servlet-name>
```

```
    <url-pattern>/EscolhaGrupo</url-pattern>
```

```
  </servlet-mapping>
```

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

# O resultado



## Recomenda-se:

Led Zeppelin  
The Who  
U2  
Yes

# Container TomCat

- ◆ Fornecido pela Apache Software Foundation;
- ◆ Utilizaremos a versão 7.0.82;
- ◆ Cópia desta em versão, para o nosso curso, encontra-se no Google Drive;
- ◆ Necessário a instalação prévia do Java, usaremos a versão 1.8.0\_91 do jdk(já instalado!);

Obs.: <https://cwiki.apache.org/confluence/display/tomcat/PoweredBy>

# Container TomCat - Instalação


- ◆ Para a instalação:

- Manter a porta padrão **8080**;
- Usar como login e senha a palavra **admin**;
- Usar todos os valores *default*, qual seja, só clicar next, next ...next;

- ◆ Vamos instalá-lo agora;













Obs.: Bom roteiro em <https://phoenixnap.com/kb/install-tomcat-windows>.

# Container TomCat - Iniciar

- ♦ Durante a instalação, será criado um diretório denominado **Apache Software Foundation**;
- ♦ O engine do TomCat chama-se **Catalina**;
- ♦ Para iniciar o Tomcat, procurar o Monitor Tomcat na barra do Windows e executar;
- ♦ Na área de serviços ativos do Windows aparecerá o ícone do Tomcat .

# Container TomCat - Diretório

Disco Local (C:) > Arquivos de Programas > Apache Software Foundation > Tomcat 7.0 >

	Nome	Data de modificação	Tipo	Tamanho
	bin	19/06/2022 13:04	Pasta de arquivos	
	conf	19/06/2022 14:42	Pasta de arquivos	
	lib	31/10/2022 19:35	Pasta de arquivos	
	logs	04/03/2023 18:44	Pasta de arquivos	
	temp	19/06/2022 13:04	Pasta de arquivos	
	webapps	16/01/2023 16:14	Pasta de arquivos	
	work	19/06/2022 14:42	Pasta de arquivos	
	LICENSE	29/09/2017 09:26	Arquivo	57 KB
	NOTICE	29/09/2017 09:26	Arquivo	2 KB
	RELEASE-NOTES	29/09/2017 09:26	Arquivo	9 KB
	tomcat	29/09/2017 09:26	Ícone	22 KB
	Uninstall	19/06/2022 13:04	Aplicativo	72 KB



# Container TomCat - Diretório

bin	Executáveis e scripts do TomCat.
conf	Arquivos de configuração do TomCat.
lib	Arquivos JAR que contém classes para todas as aplicações web e para o TomCat.
logs	Arquivos de log.
webapps	Aplicações Web hospedadas.
work	Arquivos temporários e diretórios para o TomCat.
temp	Arquivos temporários para o TomCat e JSPs pré-compiladas.

# Container TomCat

## Alguns dos Arquivos

- ◆ Sob o diretório config:
  - context(pool de conexões ao BD etc);
  - server(porta padrão ouvinte, recarga automática de servlets etc);
  - tomcat-users(login e senha dos usuários do TomCat)

# Testando a Sugestão Musical

- ♦ Examinando a interface do TomCat;
  - Digitar, na área de URL, localhost:8080
  - TomCat apresenta sua página inicial
  - Selecionar a opção Manager App
  - Entrar com login **admin** e senha **admin**
  - Examinemos o apresentado
  - Dar stop no Tomcat

# Testando a Sugestão Musical

- ◆ Fazer o download da pasta, localizada no Google Drive, SugestaoMusical\_Fontes para o disco de trabalho nosso;
- ◆ Fazer o download da pasta, localizada no Google Drive, SugestaoMusical para o subdiretório **webapps** do TomCat;
- ◆ Iniciar o TomCat;

# Testando a Sugestão Musical

- ◆ Digitar na URL:  
localhost:8080/SugestaoMusical
- ◆ Selecionar um estilo musical;
- ◆ Ver o resultado;
- ◆ Vamos aos logs do TomCat;

# Como codificar um formulário HTML

- ♦ Um formulário contém uma ou mais formas de entradas de dados tais como: *text boxes*, botões, *check boxes*, e *list boxes*.
- ♦ Um formulário **deve conter pelo menos um controle** tal como o botão SUBMIT.
- ♦ Qualquer dado associado ao controle será passado para o *servlet* ou para a JSP que está identificada pelo URL do atributo Action.

# Como codificar um formulário HTML

- ♦ Tag `<form>` `</form>` define o início e o fim do formulário.
- ♦ Possui os seguintes atributos:
  - **Action** – especifica o URL do servlet ou da JSP que será chamada quando o usuário clicar o botão SUBMIT.
  - **Method** – especifica que método do protocolo HTTP será usado na operação de *request*. Pode ser GET ou POST.

# Uso dos métodos GET e POST

- ◆ Quando usar o método GET ?
  - Se quiser transferir dados mais rapidamente.
  - Se o formulário HTML possui menos de 4 KB de tamanho.
  - Se não há problemas em os parâmetros aparecerem no URL.
- ◆ Quando usar o método POST ?
  - Se estiver transferindo mais do que 4 KB de tamanho.
  - Se não é conveniente os parâmetros aparecerem no URL.



# Como codificar um formulário HTML

- ◆ Tag `<input>` define o tipo da entrada.
- ◆ Atributos comuns:
  - **Name** – é o nome do tipo.
  - **Value** – é o valor *default* do controle.

# Como codificar um formulário HTML exemplo

- ◆ Código de um formulário HTML e seu resultado

```
<p>Um form que contém duas text boxes e um botão.</p>

<form action="confirma.jsp" method="post">
  <p>
    Nome:<input type="text" name="nome"><br>
    Email:<input type="text" name="sobrenome">
    <input type="submit" value="submit">
  </p>
</form>
```

Um form que contém duas text boxes e um botão.

Nome:

Email:

# Como codificar *text boxes*, *passwords* e campos *hidden*

- ◆ Atributos dos controles de texto:
  - **Type** – especifica o tipo do controle de entrada para os *text boxes*.
  - **Name** – especifica o nome do controle. Este é o nome que será utilizado pela aplicação JSP ou servlet.
  - **Value** – especifica o valor do dado no controle.
  - **Size** – especifica o tamanho do campo de controle em caracteres.
  - **Maxlength** – especifica o número máximo de caracteres que pode estar contido no campo.

## Tipos válidos para os *text boxes*

- ◆ Um tipo **Text** cria um *text box* padrão.
- ◆ Um tipo **Password** apresenta um *box* com asteriscos.
- ◆ Um tipo **Hidden** cria um campo *hidden* que armazena textos que não são apresentados pelo *browser*.

# Exemplos de *text boxes*, *passwords* e campos *hidden*

```
<p>Login:  <input type="text" name="login" value="jsilva"></p>  
<p>Senha:  <input type="password" name="senha" value="112358"></p>  
<input type="hidden" name="codigoProduto" value="jr01"><br>
```

Login:

Senha:



# Exercício A1



- ◆ Modificar a Sugestão Musical para incluir o nome do autor da seleção;
- ◆ Este nome será apresentado, pela JSP, junto ao resultado da seleção;
- ◆ Esqueçam a estética!

# Como codificar botões

- ◆ Atributos dos botões:
  - **Type** – especifica o tipo do controle de entrada. Os tipos aceitáveis são Submit, Reset ou Button.
  - **OnClick** – especifica o método JavaScript que será executado quando Button for clicado.

# Tipos válidos para os botões

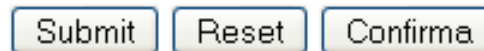
- ◆ O tipo **Submit** ativa o atributo **Action** do formulário.
- ◆ O tipo **Reset** inicia todos os controles do formulário com seus valores originais.
- ◆ O tipo **Button** cria um botão **JavaScript** que quando acionado executa uma função pré-estabelecida.



# Exemplos do uso de botões

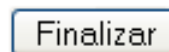
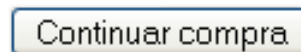
- 3 tipos de botões:

```
<input type="submit" value="Submit">  
<input type="reset" value="Reset">  
<input type="button" value="Confirma" onClick="validate(this.form) ">
```



- 2 botões Submit na mesma página:

```
<form action="compra/index.jsp" method="post">  
  <input type="submit" value="Continuar compra">  
</form>  
<form action="servlet/compra.ServletClient" method="post">  
  <input type="submit" value="Finalizar">  
</form>
```



# Como codificar *checkboxes* e *radiobuttons*

- ◆ Atributos destes botões:
  - **Type** – especifica o tipo de controle. Os tipos aceitáveis são Checkbox ou Radio.
  - **Checked** – seleciona previamente determinado controle.

# Exemplos de *radiobuttons* e *checkboxes*

```
<input type="checkbox" name="addEmail" checked>
```

Sim, me adicione na lista de emails.<br>

```
<br>
```

Entrar em contato por:<br>

```
<input type="radio" name="contatoPor" value="Email">Email
```

```
<input type="radio" name="contatoPor" value="Correios">Correios
```

```
<input type="radio" name="contatoPor" value="Ambos">Ambos<br>
```

```
<br>
```

Me interesse pelos seguintes estilos musicais:<br>

```
<input type="checkbox" name="rock">Rock<br>
```

```
<input type="checkbox" name="classica">Samba<br>
```

```
<input type="checkbox" name="pagode">Pagode<br>
```

☒ Sim, me adicione na lista de emails.

Entrar em contato por:

☐ Email ☐ Correios ☐ Ambos

Me interesse pelos seguintes estilos musicais:

☐ Rock

☐ Samba

☐ Pagode

# Como codificar *comboboxes* e *listboxes*

- ◆ Utiliza dois tipos de *tags*: **Select** e **Option**.
- ◆ Deve haver pelo menos uma *tag* Select e duas *tags* Option.
- ◆ Inicia com a *tag* Select que conterà as *tags* Option.
- ◆ A *tag* Option especifica as diferentes opções disponíveis no *box*.
- ◆ A *tag* Select possui o atributo Multiple que converte um *combox* em um *listbox*.
- ◆ A *tag* Option possui o atributo Selected que seleciona previamente uma opção.

# Exemplos de *comboboxes* e *listboxes*

- Código de um *combobox*:

```
Selecione um país:<br>
<select name="pais">
  <option value="Brasil" selected>Brasil
  <option value="Canada">Canadá
  <option value="Mexico">México
</select>
```

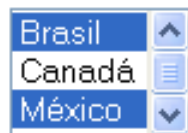
Selecione um país:



- Alterando para um *listbox*:

```
<select name="pais" multiple>
```

Selecione um país:



(Para selecionar mais de um país, pressione e segure a tecla Ctrl)

# Como codificar uma *textarea*

- ♦ Uma *textarea* difere-se de uma *textbox* pelo fato de suportar múltiplas linhas.
- ♦ Usa a tag `<Textarea> </Textarea>`
- ♦ Atributos da *textarea*:
  - Rows – especifica o número de linhas visíveis na *textarea*. Se exceder é utilizado um *scroll bar*.
  - Cols – especifica a largura da *textarea*.

# Exemplo de *textarea*

- Código de uma *textarea*:

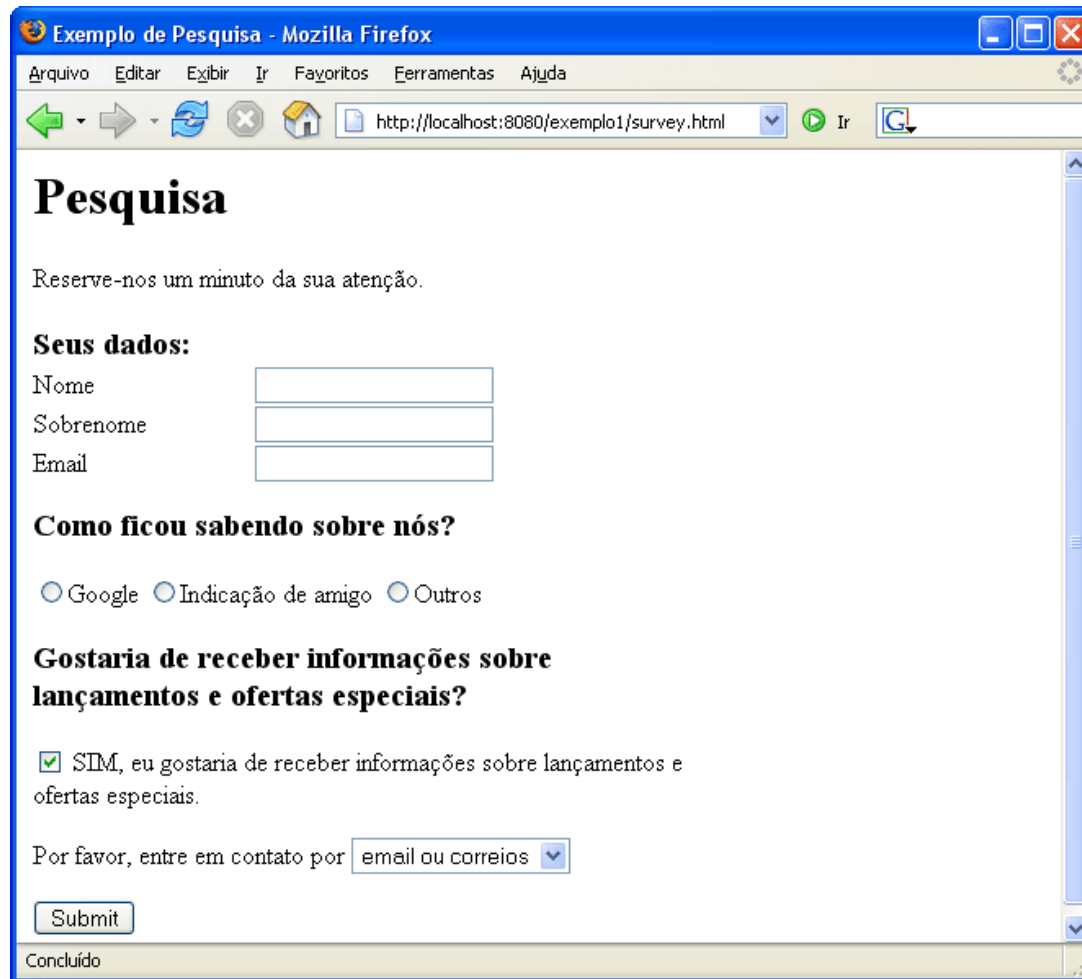
Comentários:<br>

```
<textarea name="comentario" rows="8" cols="60"></textarea>
```

Comentários:

Sim, compatriotas, não esperemos mais, a hora é esta. Vamos cometer um haraquiri coletivo, (...). Pronto, aí tudo fica perfeito. Talvez um pouco esquisito, mas objeto inquestionável de admiração internacional e mais uma vez pioneiro: seremos o primeiro país sem povo e todos os problemas desapareceriam. Por que não pensamos nisso antes? Erram, como sempre, os catastrofistas. O Brasil tem futuro, sim, apesar de que não estaremos aqui para testemunhá-lo, mas não se pode querer tudo neste mundo." (João Ubaldo

# Combinando *tags* - resultado final



The screenshot shows a Mozilla Firefox browser window with the title 'Exemplo de Pesquisa - Mozilla Firefox'. The address bar shows the URL 'http://localhost:8080/exemplo1/survey.html'. The page content includes a title 'Pesquisa', a greeting 'Reserve-nos um minuto da sua atenção.', a section 'Seus dados:' with input fields for 'Nome', 'Sobrenome', and 'Email', a section 'Como ficou sabendo sobre nós?' with radio buttons for 'Google', 'Indicação de amigo', and 'Outros', a section 'Gostaria de receber informações sobre lançamentos e ofertas especiais?' with a checked checkbox and a text input, and a 'Submit' button. The status bar at the bottom says 'Concluído'.

**Exemplo de Pesquisa - Mozilla Firefox**

Arquivo Editar Exibir Ir Favoritos Ferramentas Ajuda

← → ↻ × 🏠  Ir

## Pesquisa

Reserve-nos um minuto da sua atenção.

**Seus dados:**

Nome

Sobrenome

Email

**Como ficou sabendo sobre nós?**

☐ Google ☐ Indicação de amigo ☐ Outros

**Gostaria de receber informações sobre lançamentos e ofertas especiais?**

☒ SIM, eu gostaria de receber informações sobre lançamentos e ofertas especiais.

Por favor, entre em contato por

Concluído



# Combinando *tags* – código HTML

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
  <title>Exemplo de Pesquisa</title>
</head>
<body>

<table border="0" cellpadding="0">
<form action="/exemplo1/servlet/br.ufrj.dcc.poo.SurveyServlet" method="post">
  <tr>
    <td width="410" valign="top" colspan="2">

      <h1>Pesquisa</h1>
      <p>Reserve-nos um minuto da sua atenção.</p>
      <h3>Seus dados:</h3>
    </td>
  </tr>
  <tr>
    <td><p>Nome</p>

    <td><input type="text" name="nome" size="20" tabindex="1"></td>
  </tr>
  <tr>
    <td><p>Sobrenome</p>
    <td><input type="text" name="sobrenome" size="20" tabindex="2"></td>
  </tr>
```

# Combinando *tags* – código HTML

```
<tr>
  <td><p>Email</td>

  <td><input type="text" name="email" size="20" tabindex="3"></td>
</tr>
<tr>
  <td colspan="2" height="12"></td>
</tr>
<tr>
  <td width="410" valign="top" colspan="2">
    <h3>Como ficou sabendo sobre nós?</h3>

    <p>
      <input type="radio" name="heardFrom" value="Google" tabindex="4">Google
      <input type="radio" name="heardFrom" value="Amigo">Indicação de amigo
      <input type="radio" name="heardFrom" value="Outros">Outros
    </p>
    <h3>Gostaria de receber informações sobre lançamentos e ofertas especiais?</h3>
    <p><input type="checkbox" name="querAtualiza" checked> SIM, eu gostaria de receber
informações sobre lançamentos e ofertas especiais.<br>
  </p>
  <p>
```

# Combinando *tags* – código HTML

```
Por favor, entre em contato por
<select name="contatoPor">
  <option value="Ambos" checked>email e correios
  <option value="Email">email apenas
  <option value="Correios">correios apenas
</select>
</p>
<p><input type="submit" value="Submit" tabindex="5"></p>
</td>
</tr>
</form>

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

# A interface ServletRequest

- ◆ Esta interface tem o propósito de entregar os dados do cliente para o servlet;
- ◆ Alguns dos dados fornecidos pelo ServletRequest incluem nomes e valores de parâmetros, atributos e um fluxo de entrada;
- ◆ Métodos do ServletRequest podem fornecer dados adicionais específicos do protocolo;
- ◆ Dados HTTP são fornecidos pela interface HttpServletRequest, que estende ServletRequest;
- ◆ Essa estrutura fornece o único acesso do servlet a esses dados.

# A interface ServletRequest

- ◆ Oferece alguns dos seguintes métodos:
  - **getServerPort()**
    - Obtém a porta onde o servidor está ouvindo o meio.
  - **GetServerName()**
    - Obtém o nome do host do servidor para o qual a solicitação foi enviada.
  - **getProtocol()**
    - Obtém o nome e a versão do protocolo utilizado.
  - **getRemoteAddr()**
    - Obtém o endereço IP do cliente remoto.
  - **getRemoteHost()**
    - Obtém o nome qualificado do cliente remoto.

# A interface ServletRequest

- **getParameter(String)**
  - Obtém o valor de um parâmetro específico.
- **getParameterValues(String)**
  - Obtém um *array* com os valores dos parâmetros passados.
- **getParameterNames()**
  - Obtém os nomes dos parâmetros passados.

Obs.: visitar <https://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html>

# A interface ServletRequest

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletRequestTeste extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
        doPost(request, response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException{
        System.out.println("Porta Servidora: " + request.getServerPort());
        System.out.println("Servidor: " + request.getServerName());
        System.out.println("Protocolo: " + request.getProtocol());
        System.out.println("Cliente: " + request.getRemoteHost());
        System.out.println("Endereço Cliente: " + request.getRemoteAddr());
    }
}
```

# A interface ServletRequest

```
import javax.servlet.*;
import javax.servlet.http.*;

public class ValidaUsuarioSenhaMobile extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {

        String usuario = request.getParameter("usuario");
        String senha = request.getParameter("senha");

        .....

    }
}
```



# A interface ServletRequest

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class ParameterRequest extends HttpServlet {

    public void doGet ( HttpServletRequest request,
                       HttpServletResponse response )
        throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<h2>" + "Separando os Parâmetros" + "</h2>");

        out.println("<b>" + "Nome:" + request.getParameter("nome") + "</b>" + "<BR>");
        out.println("<b>" + "SobreNome:" + request.getParameter("sobrenome") + "</b>" + "<BR>");

    }
}
```

# A interface ServletRequest



## Separando os Parâmetros

Nome: Maria  
SobreNome: Eduarda

- Os nomes dos campos são *case sensitive*.

# A interface ServletRequest

- ◆ Parâmetros com múltiplos valores:
  - Utilizado quando um parâmetro possui diversos valores. Exs: ListBox e CheckBox.
  - O método **getParameter** só fornece o primeiro valor do parâmetro.
  - O método **getParameterValues** retorna um *array* de *strings* contendo todos os valores selecionados.
  - O nome do parâmetro é o argumento para o método **getParameterValues**.

# A interface ServletRequest

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class ParameterRequest extends HttpServlet {

    public void doPost (HttpServletRequest request,
                        HttpServletResponse response )
        throws ServletException, IOException {

        String[] values = request.getParameterValues("musicasFavoritas");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        if (values != null) {
            int length = values.length;
            out.println("<h2>" + "Você Selecionou: " + "</h2>");
            for (int i=0; i<length; i++) {
                out.println("<BR>" + values[i]);
            }
        }
    }
}
```

# A interface ServletRequest

Selecione sua música favorita:

- ☐ Rock
- ☐ Jazz
- ☐ Pagode
- ☐ MPB

Submit

Você selecionou:

Rock  
MPB

## Exercício A2

- ◆ Modificar a Sugestão Musical para utilizar o checkbox com as sugestões musicais;
- ◆ Alterar a recomendação musical para suportar qualquer número de sugestões e apresentá-las;
- ◆ Opcionalmente, podem apresentar dados do protocolo;
- ◆ Esqueçam a estética!

# A interface ServletRequest

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class HttpRequestDemoServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Obtendo Parâmetros com Múltiplos Valores</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");

        out.println("<BR>");
        out.println("<BR>Selecione sua música favorita:");
        out.println("<BR><FORM METHOD=POST>");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=Rock>Rock");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=Jazz>Jazz");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=Pagode>Pagode");
        out.println("<BR><INPUT TYPE=CHECKBOX " +
            "NAME=musicaFavorita VALUE=MPB>MPB");
        out.println("<BR><INPUT TYPE=SUBMIT VALUE=Submit>");
        out.println("</FORM>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

# A interface ServletRequest

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    String[] values = request.getParameterValues("musicaFavorita");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    if (values != null ) {
        int length = values.length;
        out.println("Você selecionou: ");
        for (int i=0; i<length; i++) {
            out.println("<BR>" + values[i]);
        }
    }
}
```



# TomCat – error pages

- ◆ Descritos no arquivo web-xml da aplicação, através da tag `<error-page>`;
- ◆ Define para quais páginas os erros encontrados serão apresentados;

```
<error-page>  
    <exception-type>java.lang.NullPointerException</exception-type>  
    <location>/jsp/VistaPaginaInexiste.jsp</location>  
</error-page>  
<error-page>  
    <error-code>404</error-code>  
    <location>/jsp/VistaPaginaInexiste.jsp</location>  
</error-page>  
<error-page>  
    <error-code>500</error-code>  
    <location>/jsp/VistaPaginaInexiste.jsp</location>  
</error-page>  
<error-page>  
    <error-code>405</error-code>  
    <location>/jsp/VistaPaginaInexiste.jsp</location>  
</error-page>
```

# O ciclo de vida de um *servlet*

## ◆ O método `init()`

- Inicia o *servlet*.
- O *container* chama este método apenas uma vez.
- Pode ser utilizado para iniciar variáveis, carregar o *driver* de um banco de dados etc.
- Recebe, através do objeto `ServletConfig`, os valores especificados no arquivo `web.xml`.
- Assinatura do método:

**`public void init(ServletConfig config) throws ServletException`**

- **Método de uso opcional.**

# O ciclo de vida de um *servlet*

- ◆ O método `Service()`
  - É acionado pelo *container* após o término bem sucedido do método `init()`.
  - Executado a cada chamada do *servlet*.
- ◆ `Destroy()`
  - Remove o *servlet*. Ocorre por falta de uso ou *shutdown* do *server*.

# Como desenvolver *servlets*

- ♦ Um *servlet* herda da classe `HttpServlet` que herda da classe `GenericServlet` que implementa a interface `Servlet`.
- ♦ Necessário importar os pacotes `javax.servlet` e `javax.servlet.http`, contidos em **`servlet.api`**(cópia no diretório `lib` do TomCat).
- ♦ O método `init()` pode ser sobreposto.
- ♦ Pelo menos um método de serviço precisa ser sobreposto.

# Como desenvolver *servlets*

- ◆ O método `doGet` processa todos os HTTP *requests* que usam o método `Get`.
- ◆ O método `doPost` processa todos os HTTP *requests* que usam o método `Post`.
- ◆ Estes métodos recebem os objetos *request* e *response* repassados pelo *container*.
- ◆ O método `setContentType`, do objeto *response*, indica o tipo de resposta retornada ao *browser*.
- ◆ O método `getWriter`, do objeto *response*, é usado para enviar o arquivo HTML para o *web browser*.

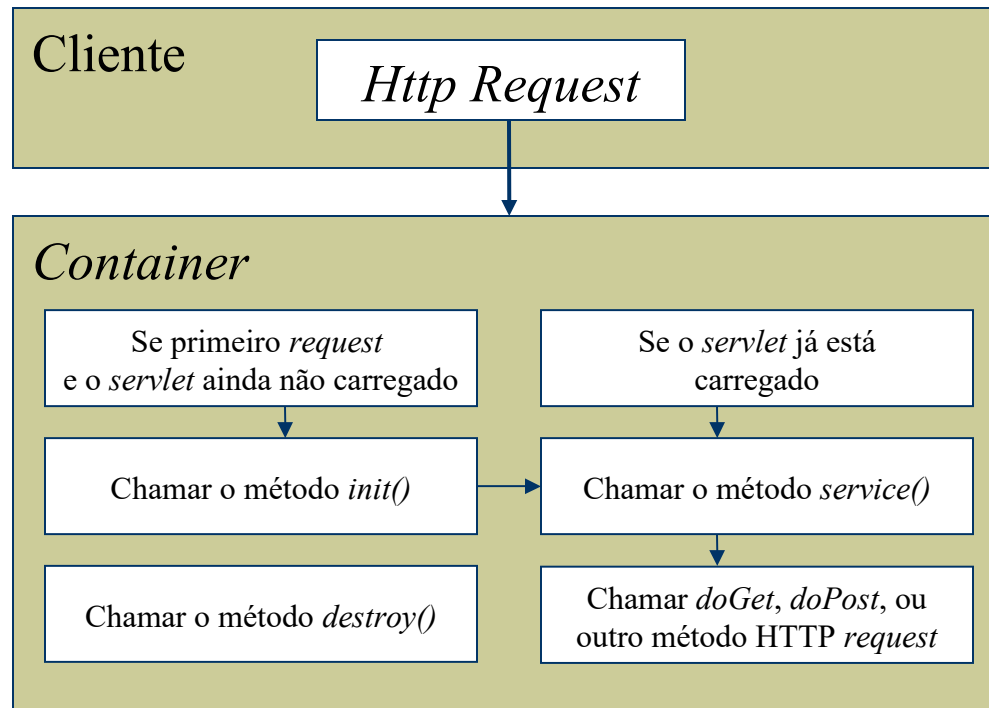


## Alguns privilégios dos *servlets*



- ◆ Capacidade de “*logar*” eventos.
- ◆ Obter referências para outros recursos.
- ◆ Passar atributos para outros *servlets*.

# Como o *container* trata um *request* para um *servlet* ?

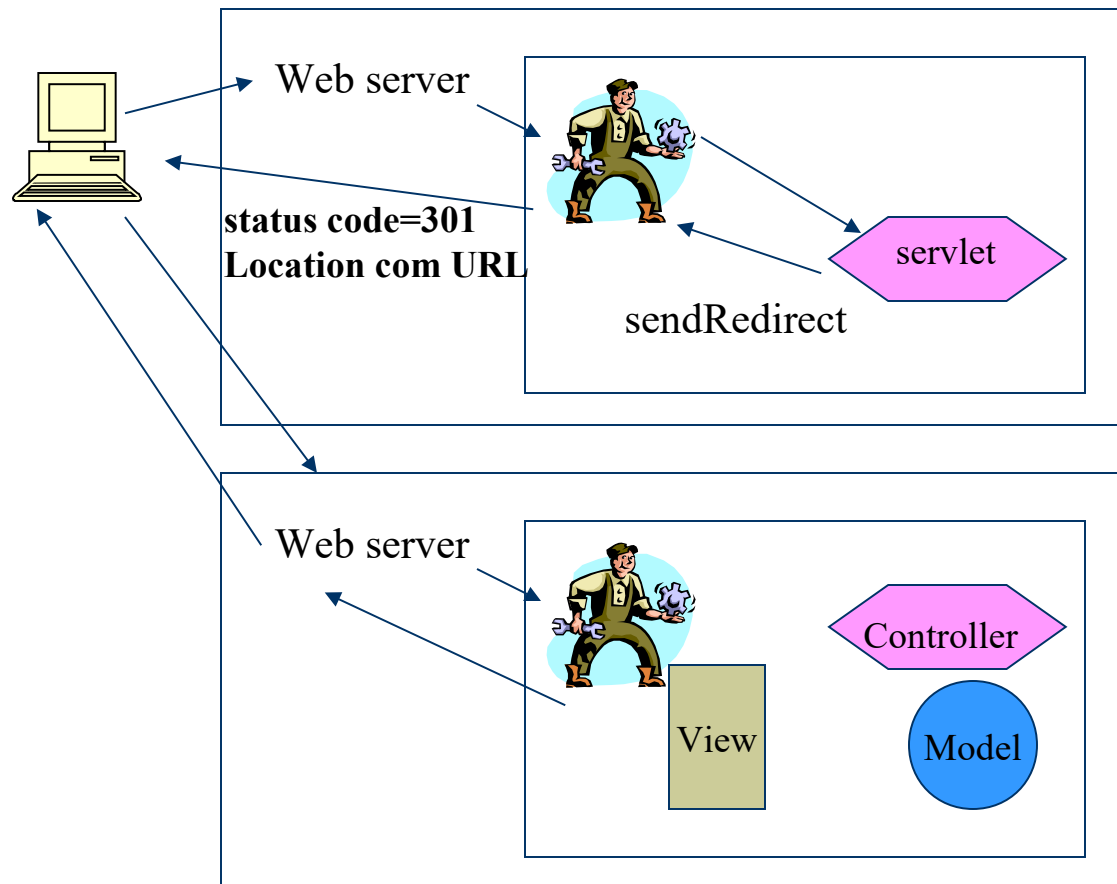


# Redirecionando as respostas

- ♦ O *servlet* pode direcionar uma requisição para outro *servlet* ou para uma JSP.
- ♦ O *servlet* ou JSP destino pode residir em uma URL remota ou no mesmo *container*.
- ♦ O recurso remoto não tem acesso aos objetos *request* e *response* do *servlet* original.



# Redirecionando para outra URL



# Redirecionando para outra URL

## ♦ O que escrevo para redirecionar?

- `response.sendRedirect("http://www.nce.ufrj.br");`

ou

- `response.sendRedirect("/OutraAplicacao/Sugestao")`
  - Desvia para `http://www.nce.ufrj/ OutraAplicacao/Sugestao`;
    - ♦ A "/" significa conectar-se à raiz (outra webapps).

ou

- `response.sendRedirect("FechaCompra/Sugestao");`
  - Conecta-se à webapps original.

♦ Obs.: A URL do novo destino é apresentada no *web browser*.

# Redirecionando para o mesmo local

- ◆ O que escrevo para redirecionar?
  - `RequestDispatcher vista = request.getRequestDispatcher("sugestao.jsp");`
  - `vista.forward(request,response);`
- ◆ *O web browser desconhece este redirecionamento.*

# ServletConfig

- ◆ Objeto criado pelo *container* e utilizado para passar parâmetros de iniciação para um *servlet*.
- ◆ Parâmetros são definidos no web-xml.
- ◆ Evita a inserção de valores, passíveis de alterações, nos *servlets*.
- ◆ Para ativar uma nova versão web-xml é só fazer um *redeploy* ou *reload*, da aplicação, no *container*.
- ◆ Existe apenas um por cada *servlet*.
- ◆ Não pode ser alterado.

# ServletConfig

- ◆ Oferece alguns dos seguintes métodos:
  - **getInitParameter(String)**
    - Retorna o conteúdo de um parâmetro específico.
  - **Enumeration getInitParameterNames()**
    - Retorna um conjunto com os nomes dos parâmetros especificados.

# ServletConfig

## ◆ Especificando no web-xml:

```
<servlet>
  <servlet-name>Musicas</servlet-name>
  <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
  <init-param>
    <param-name>faleConosco</param-name>
    <param-value>centralatendimento@nce.ufrj.br</param-value>
  </init-param>
  <init-param>
    <param-name>areaVendas</param-name>
    <param-value>vendasatendimento@nce.ufrj.br</param-value>
  </init-param>
</servlet>
```

## • Obtendo no *servlet* ou JSP:

```
getServletConfig().getInitParameter("faleConosco");
getServletConfig().getInitParameter("areaVendas");
```

# ServletContext

- ◆ Reflete o ambiente onde o *servlet* é executado.
- ◆ Criado pelo *container* para cada aplicativo *web* existente.
- ◆ Utilizado para os *servlets* compartilharem informações.
- ◆ Independe de sessão.
- ◆ Suporta atributos que podem ser modificados ou recuperados pelos *servlets* ou JSPs.

# ServletContext

- ◆ Permite a declaração de parâmetros no web-xml.
- ◆ Estes parâmetros podem ser recuperados, em qualquer instante, pelos *servlets* ou JSPs.
- ◆ Lembrete: Atributos retornam um *Object* e parâmetros retornam um *String*.



# ServletContext

- ◆ Oferece alguns dos seguintes métodos:
  - **getAttributeNames()**
    - Retorna um conjunto com os nomes dos atributos armazenados.
  - **getAttribute(String)**
    - Retorna um atributo específico do contexto.
  - **setAttribute(String, Object)**
    - Armazena um atributo no contexto
  - **removeAttribute(String)**
    - Remove um atributo do contexto.

# ServletContext

- **getInitParameter(String)**
  - Retorna o conteúdo de um parâmetro específico.
- **Enumeration getInitParameterNames()**
  - Retorna um conjunto com os nomes dos parâmetros especificados.
- **getRequestDispatcher(String)**
  - Desvia para um recurso local.

# ServletContext

## ◆ Especificando no web-xml:

```
<servlet>
```

```
  <servlet-name>Musicas</servlet-name>
```

```
  <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
```

```
  <init-param>
```

```
    <param-name>faleConosco</param-name>
```

```
    <param-value>centralatendimento@nce.ufrj.br</param-value>
```

```
  </init-param>
```

```
  <init-param>
```

```
    <param-name>areaVendas</param-name>
```

```
    <param-value>vendasatendimento@nce.ufrj.br</param-value>
```

```
  </init-param>
```

```
</servlet>
```

```
<context-param>
```

```
  <param-name>enderecoReal</param-name>
```

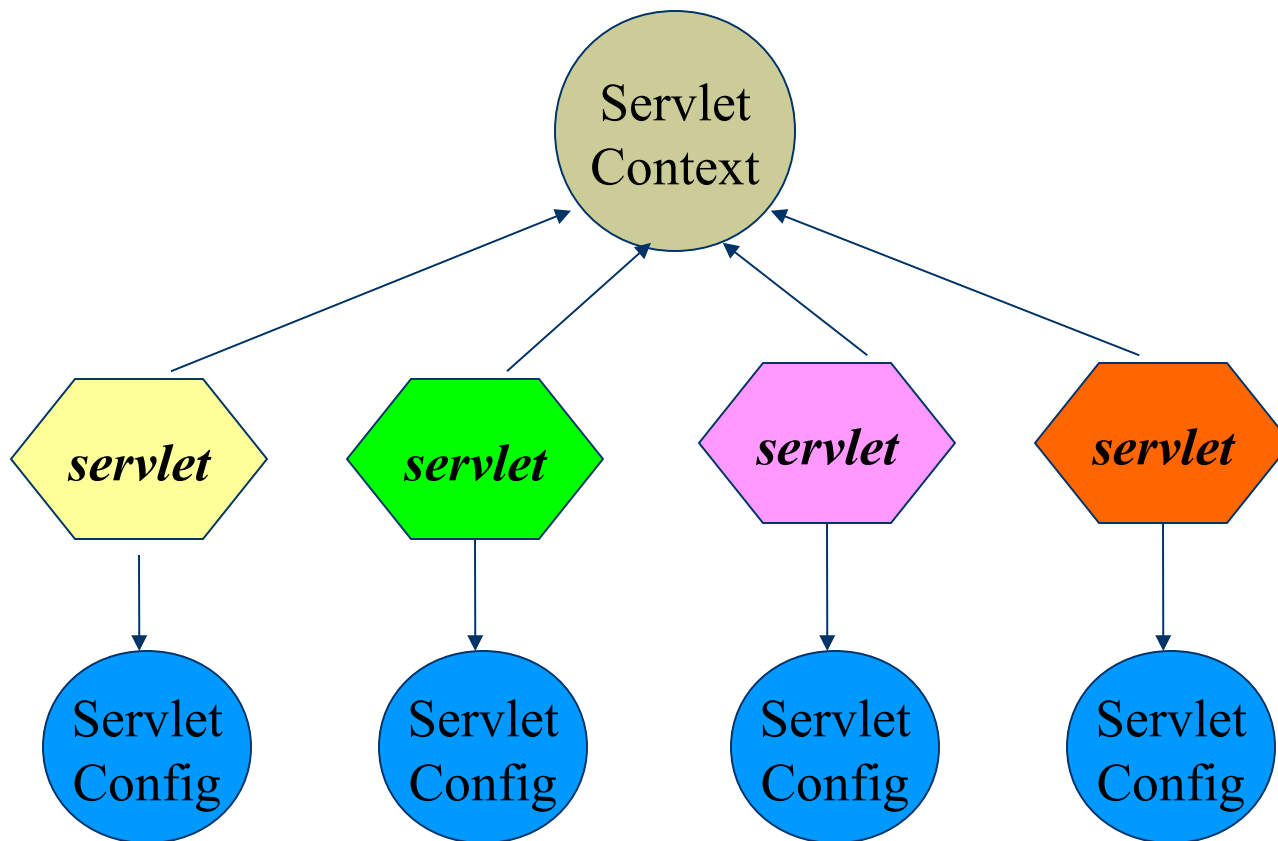
```
  <param-value>Avenida Rio Branco 156</param-value>
```

```
</context-param>
```

# ServletContext

- Obtendo um parâmetro pelo *servlet* ou JSP:  
`getServletContext().getInitParameter("enderecoReal");`
  - Exemplo:
    - `Integer.parseInt(getServletContext().getInitParameter("UltimaQualitec"));`
- Criando um atributo pelo *servlet* ou JSP:  
`getServletContext().setAttribute("endereco", "Avenida Rio Branco 156");`
- Obtendo o atributo pelo *servlet* ou JSP:  
`getServletContext().getAttribute("endereco");`

# ServletConfig e ServletContext



# Java Server Page

- ◆ É uma extensão da tecnologia *servlet*.
- ◆ Criada para suportar a criação de páginas HTML e XML.
- ◆ Combina conteúdo estático com dinâmico.
- ◆ Desonera o programador de se preocupar com os elementos de arte da página.

# Java Server Page

- ◆ Consiste de *tags* HTML e código Java.
- ◆ O código Java fica embutido no código HTML como um *scriptlet* ou uma expressão.
- ◆ Um *scriptlet* é usado para executar um ou mais comandos Java.
- ◆ Uma expressão é usada para apresentar um texto.
- ◆ Para identificar *scriptlets* e expressões são utilizadas *tags* específicas.

# Java Server Page

- ♦ Atos do *container* para processar uma JSP:
  - Quando acionada, a JSP é traduzida para um arquivo .java.
  - Este arquivo .java é compilado tornando-se um arquivo .class.
  - O arquivo .class é carregado e transforma-se, finalmente, em um *servlet*.
- ♦ Obs.: Consultar:  
[Apache Software Foundation/work/Catalina/localhost/aplicacao/org/apache/jsp](http://Apache Software Foundation/work/Catalina/localhost/aplicacao/org/apache/jsp)



# Onde salvar uma JSP ?

- ◆ Precisa ser salva em um diretório visto pelo *web server*.
- ◆ No Tomcat  $\geq 5.0$  pode ser usado qualquer diretório sob o diretório *webapps*.
- ◆ Página deve ter o nome com o sufixo jsp.

# *Scriptlets* e expressões

## ♦ *Scriptlet*

- `<% java statements %>` **não esquecer o “;”**

## ♦ Expressão

- `<% = expressão %>`
  - Atua como um argumento de `out.print()`.

# Scriptlets e expressões

- ♦ `<% String nome=request.getParameter("nome"); %>`

O nome é `<%= nome %>` //ASSIM ou

O nome é `<%= request.getParameter("nome")%>`

```
<%@ page session="false" %>
<% int numeroDeVezes=1;
    while (numeroDeVezes<=5) {
%>
<h3> esta linha é apresentada <%= numeroDeVezes %> vez(es) . </h3>
<%     numeroDeVezes++;
    }
%>
```

# Como codificar *scriptlets* e expressões



**esta linha é apresentada 1 vez(es) .**

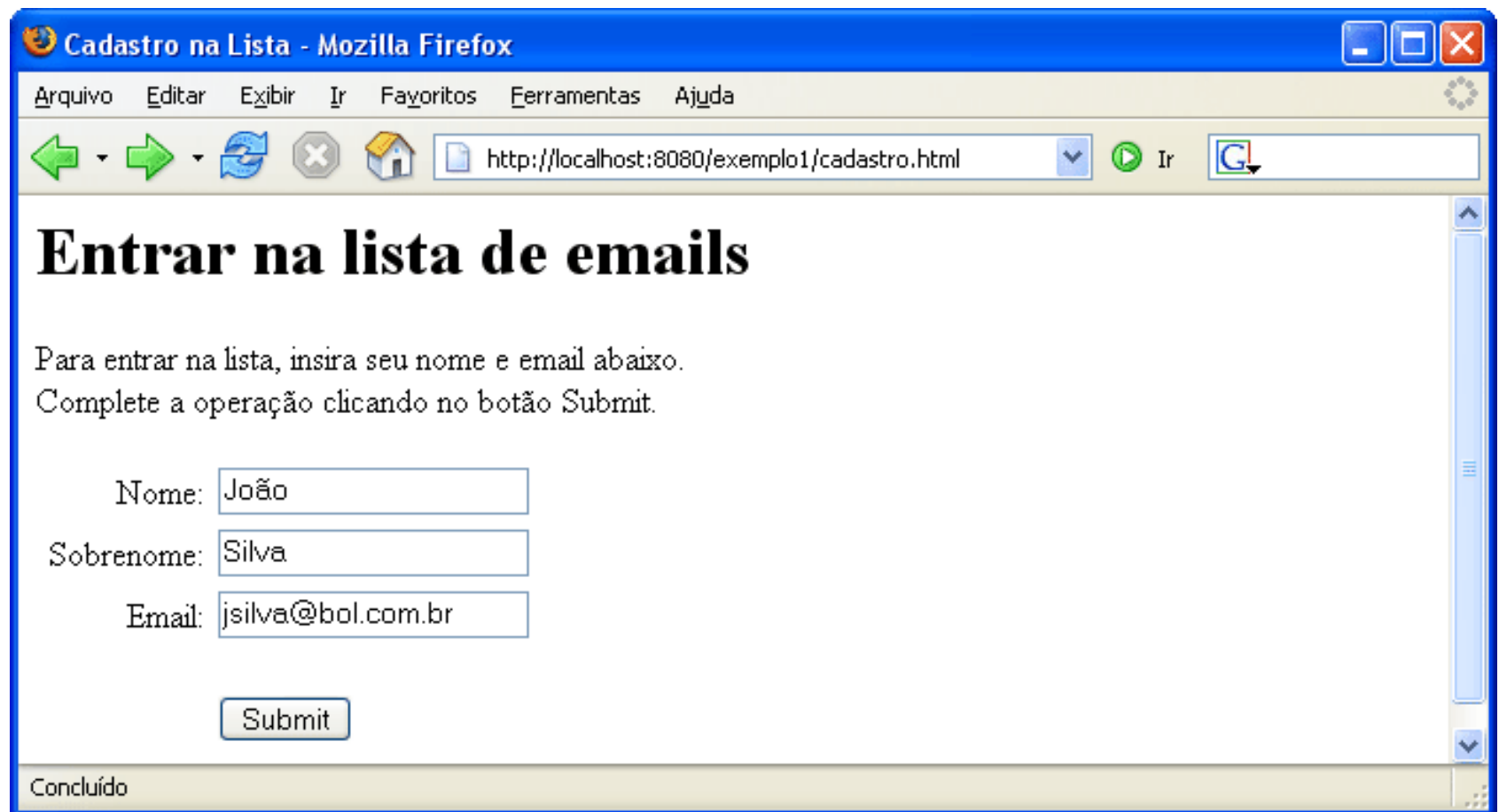
**esta linha é apresentada 2 vez(es) .**

**esta linha é apresentada 3 vez(es) .**

**esta linha é apresentada 4 vez(es) .**

**esta linha é apresentada 5 vez(es) .**

# Como codificar *scriptlets* e expressões



The screenshot shows a Mozilla Firefox browser window with the title "Cadastro na Lista - Mozilla Firefox". The address bar displays "http://localhost:8080/exemplo1/cadastro.html". The page content includes a heading "Entrar na lista de emails", a paragraph "Para entrar na lista, insira seu nome e email abaixo. Complete a operação clicando no botão Submit.", and a form with three input fields: "Nome:" (containing "João"), "Sobrenome:" (containing "Silva"), and "Email:" (containing "jsilva@bol.com.br"). A "Submit" button is located below the email field. The status bar at the bottom indicates "Concluído".

**Cadastro na Lista - Mozilla Firefox**

Arquivo Editar Exibir Ir Favoritos Ferramentas Ajuda

← → ↻ ⓧ 🏠 📄 http://localhost:8080/exemplo1/cadastro.html Ir 🔍

## Entrar na lista de emails

Para entrar na lista, insira seu nome e email abaixo.  
Complete a operação clicando no botão Submit.

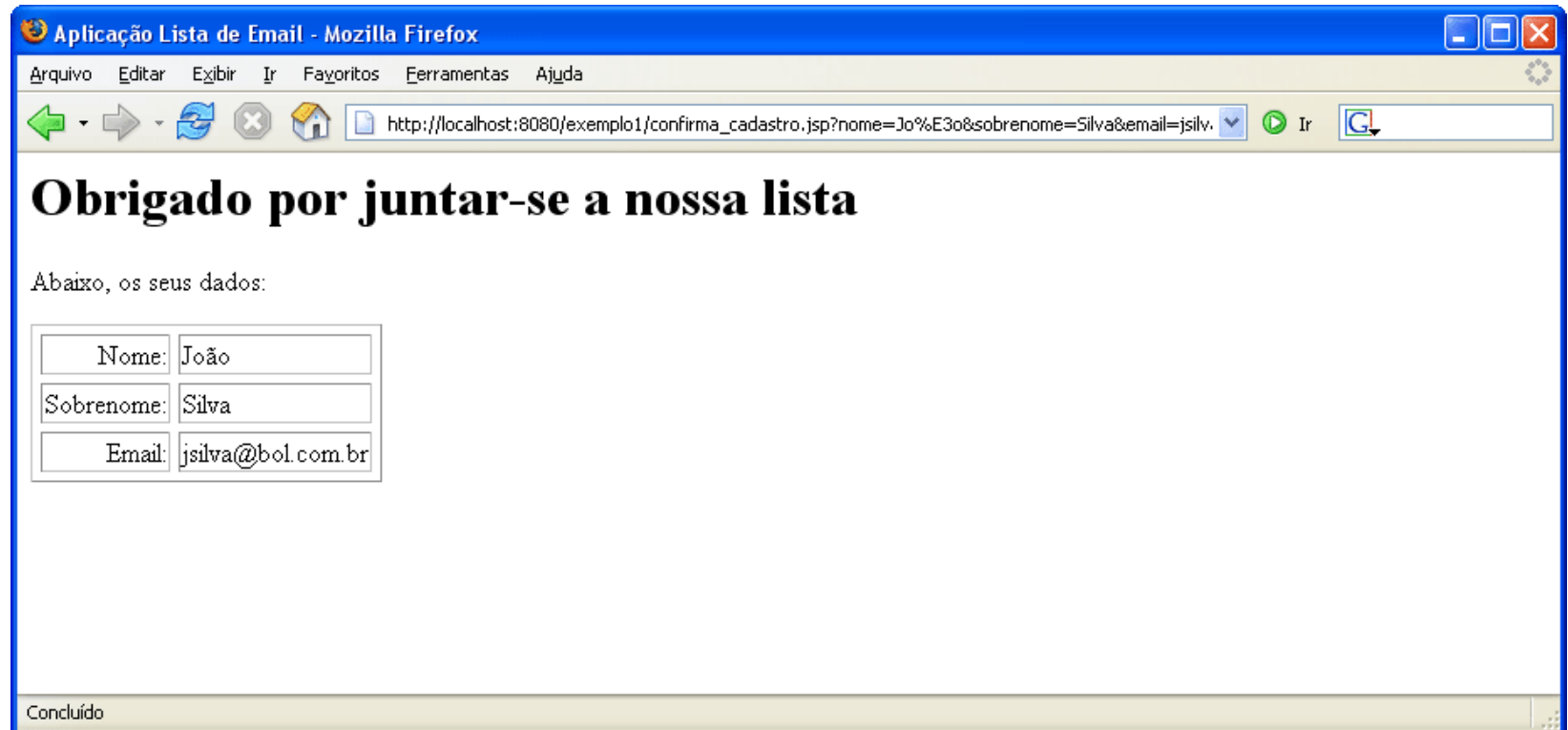
Nome:

Sobrenome:

Email:

Concluído

# Como codificar *scriptlets* e expressões



# Página HTML

```
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
```

```
<html>
```

```
<head>
```

```
<title>Cadastro na Lista</title>
```

```
</head>
```

```
<h1>Entrar na lista de emails</h1>
```

```
<p>Para entrar na lista, insira seu nome e email abaixo. <br>  
Complete a operação clicando no botão Submit.</p>
```

# Página HTML

```
<form action="confirma_cadastro.jsp" method="get">
<table cellpadding="5" border="0">
  <tr>
    <td align="right">Nome:</td>
    <td><input type="text" name="nome"></td>
  </tr>
  <tr>
    <td align="right">Sobrenome:</td>
    <td><input type="text" name="sobrenome"></td>
  </tr>
  <tr>
    <td align="right">Email:</td>
    <td><input type="text" name="email"></td>
  </tr>
  <tr>
    <td></td>
    <td><br><input type="submit" value="Submit"></td>
  </tr>
</table>
</form>
</body>
</html>
```



# A página confirma\_cadastro.jsp

```
<%@ page language="java" %>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Aplicação Lista de Email</title>
</head>
<body>
<%
    String nome = request.getParameter("nome");
    String sobrenome = request.getParameter("sobrenome");
    String email = request.getParameter("email");
%>
<h1>Obrigado por juntar-se a nossa lista</h1>
<p>Abaixo, os seus dados:</p>
<table cellpadding="5" cellspacing="5" border="1">
    <tr>
        <td align="right">Nome:</td>
        <td><%= nome %></td>
    </tr>
    <tr>
        <td align="right">Sobrenome:</td>
        <td><%= sobrenome %></td>
    </tr>
    <tr>
        <td align="right">Email:</td>
        <td><%= email %></td>
    </tr>
</table>
</body>
</html>
```

# JSP – Ciclo de Vida

- ◆ O cliente aciona uma JSP.
- ◆ O *container* tenta traduzir o JSP para um arquivo .java.
- ◆ Se OK, o .java é compilado gerando um arquivo .class.
- ◆ O *container* carrega o .class passando a tratá-lo como um *servlet*.

# JSP – Ciclo de Vida

## ◆ São criados os métodos:

- `jspInit()`.
  - Pode ser sobreposto.
- `jspService()`.
  - Não pode ser sobreposto.
- `jspDestroy()`.
  - Pode ser sobreposto.

# JSP – Ciclo de Vida

- ♦ O *container* instancia o *servlet* acionando o método `jspInit()`.
- ♦ O *container* cria uma nova *thread* e aciona o método `jspService()`.
- ♦ A *thread* é disparada.
- ♦ O *container* aciona o método `jspDestroy()`.

Obs.: A tradução e a compilação só acontecem uma única vez.

# JSP – web.xml

```
<web-app...>
```

```
....
```

```
<servlet>
```

```
  <servlet-name>loginSeradTeste</servlet-name>
```

```
  <jsp-file>/jsp/VistaPaginaPrincipal.jsp</jsp-file>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
  <servlet-name>loginSeradTeste</servlet-name>
```

```
  <url-pattern>/LoginSeradInovUERJ</url-pattern>
```

```
</servlet-mapping>
```

```
...
```

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

# As diretivas de uma JSP

- ◆ São instruções passadas para o *container* em tempo de tradução de uma JSP.
- ◆ Existem 3 tipos de diretivas:
  - *page*.
  - *include*.
  - *taglib*.

# Diretiva *page*

## ◆ Sintaxe:

- `<%@ page atributo1="valor1"... atributon="valorn" %>`

## ◆ Alguns atributos:

- **import**
  - Define os comandos *imports* que devem ser adicionados à classe gerada.
- **isThreadSafe**
  - Define se a JSP deve ser *SingleThreadModel* (*false*).
- **contentType**
  - Define o tipo do MIME.
- **isErrorPage**
  - Define se a página corrente representa uma página de erro.
- **errorPage**
  - Define a URL da página que tratará o erro.

# Diretiva *page*

## ♦ Alguns atributos:

- **language**
  - Define a linguagem de *script*. Por enquanto é Java.
- **extends**
  - Define a superclasse de quem a JSP pode herdar.
- **session**
  - Indica se a página terá um objeto *session* implícito (*default = true*).
- **buffer**
  - Define a buferização para o objeto *out*.
- **isELIgnored**
  - Define se expressões EL devem ser ignoradas quando a página for traduzida.



# Diretivas de página

- ◆ `<%@ page contentType="text/html;charset=GB2312" %>`
- ◆ `<%@ page language="java" %>`
- ◆ `<%@ page import="java.io.*" %>` único replicável
- ◆ `<%@ page buffer="16kb" %>`
- ◆ `<%@ page session="false" %>`
- ◆ `<%@ page errorPage="PaginadeErro.jsp" %>`
- ◆ `<%@ page session="false" buffer="16kb" %>`

# Diretivas de inclusão

- ◆ Permite incluir o conteúdo de outros arquivos na página JSP atual. Feito em tempo de tradução.
- ◆ A página incluída pode ser uma página estática (HTML) ou dinâmica (JSP).
- ◆ A página inserida tem acesso às variáveis de instância da JSP principal.
- ◆ Sintaxe:
  - `<%@ include file="URLrelativa" %>`
- ◆ Exemplo:
  - `<%@ include file="POO/cabecalho.html" %>`

# Diretivas de inclusão - exemplo

- Arquivo Cabecalho.htm

```
<HTML>
```

```
<HEAD>
```

```
<TITLE> Bem-Vindo </TITLE>
```

```
</HEAD>
```

```
<BODY>
```

- Arquivo Rodape.htm

```
</BODY>
```

```
</HTML>
```

# Diretivas de inclusão - exemplo

```
<%@ page session="false" %>
<%@ page import="java.util.Calendar" %>
<%@ include file="Cabecalho.htm" %>
<%
    out.println("Hora atual: " + Calendar.getInstance().getTime());
%>
<%@ include file="Rodape.htm" %>
```

# Diretivas de inclusão - exemplo



Hora atual: Sat Apr 23 00:16:59 BRT 2005

# Resumo dos elementos em uma JSP

- ◆ `<% %>` *scriptlet* JSP.
  - Para inserir comandos em Java.
- ◆ `<%= %>` expressão JSP.
  - Para apresentar o resultado de uma expressão.
- ◆ `<%(%)>` diretiva JSP.
  - Para atribuir condições aplicáveis a toda JSP.

# JSP – Melhorando a carga inicial

```
<servlet>
  <servlet-name>RecepcaoJspServlet</servlet-name>
  <jsp-file>/Recepcao.jsp</jsp-file>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Obs.: Utilizando em servlets também;

# JSP – Tornando a JSP a página inicial

```
<welcome-file-list>
```

```
  <welcome-file>index.jsp</welcome-file>
```

```
</welcome-file-list>
```



## Exercício A3

- ◆ Criar uma aplicação web denominada CadastroPrimario;
- ◆ Sua página index.html possui os campos: nome, cpf, senha e repetição de senha;
- ◆ Estes dados serão enviados, por POST, para o servlet denominado ControllerCadastrarDados;
- ◆ Os dados enviados serão retornados através da JSP cadastrado.jsp;
- ◆ Usar, como base, Exercicio\_A3.html
- ◆ Esqueçam a estética!

# Exercício A3

Nome e Sobrenome

CPF

Senha

Repita a Senha

**Enviar**

# Apache Commons Lang

- ◆ As bibliotecas Java padrão falham em fornecer métodos suficientes para manipulação de suas classes principais;
- ◆ O Apache Commons Lang fornece esses métodos extras;
- ◆ Fornece uma série de utilitários auxiliares para a API `java.lang`;
- ◆ Notadamente métodos de manipulação de strings, métodos numéricos básicos, reflexão de objetos, simultaneidade, criação e serialização e propriedades do sistema;

# Apache Commons Lang

## Classe StringUtils - métodos

**IsEmpty/IsBlank** - verifica se uma String contém texto;

**Trim/Strip** - remove espaços em branco iniciais e finais;

**Equals/Compare** - compara duas strings de maneira segura para nulls;

**startsWith** - verifica se uma String começa com um prefixo de maneira segura para nulls;

**endsWith** - verifica se uma String termina com um sufixo de maneira segura para nulls;

**IndexOf/LastIndexOf/Contains** - verificações de índice seguro null;

**IndexOfAny/LastIndexOfAny/IndexOfAnyBut/LastIndexOfAnyBut** - índice de qualquer um de um conjunto de Strings;

**ContainsOnly/ContainsNone/ContainsAny** - verifica se String contém apenas/nenhum/qualquer um desses caracteres;

**Substring/Left/Right/Mid** - extrações de substring com segurança para null;

**SubstringBefore/SubstringAfter/SubstringBetween** - extração de substring relativa a outras strings;

# Apache Commons Lang

## Classe StringUtils – alguns métodos

Split/Join - divide uma String em uma matriz de substrings e vice-versa;

Remove/Delete - remove parte de uma String;

Replace/Overlay - pesquisa uma String e substitui uma String por outra;

Chomp/Chop - remove a última parte de uma String;

AppendIfMissing - acrescenta um sufixo ao final da String se não estiver presente;

PrependIfMissing - anexa um prefixo ao início da String se não estiver presente;

LeftPad/RightPad/Center/Repeat - preenche uma String;

**UpperCase/LowerCase/SwapCase/Capitalize/Uncapitalize** - muda o *case* de uma String;

CountMatches - conta o número de ocorrências de uma String em outra;

**IsAlpha/IsNumeric/IsWhitespace/IsAsciiPrintable** - verifica os caracteres em uma String;

DefaultString - protege contra uma string de entrada null;

Rotate - girar (deslocamento circular) uma String;

Reverse/ReverseDelimited - inverte uma String;

Abbreviate - abrevia uma string usando reticências ou outra String dada;

Difference - compara Strings e relata suas diferenças;

# ControllerCadastrarDados

```
import org.apache.commons.lang3.StringUtils;
```

```
public class ControllerCadastrarDados extends HttpServlet{

    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
        response.setContentType("text/html;charset=UTF-8");
        String nome = request.getParameter("nome");
        String cpf = request.getParameter("cpf");
        String senha = request.getParameter("senha");
        String confirmasenha = request.getParameter("confirmaSenha");

        if(StringUtils.isNotBlank(nome) & StringUtils.isNotBlank("cpf") & StringUtils.isNotBlank("senha") &
            StringUtils.isNotBlank("confirmaSenha")){

            request.setAttribute("cadastrado" , "Cadastrados: " + nome + " , " + cpf + " , " + senha + " , " +
                confirmasenha);
        }
        else{
            request.setAttribute("cadastrado" ,"Você não preencheu todos os campos!");
        }
        RequestDispatcher vista = request.getRequestDispatcher("/jsp/cadastrado.jsp");
        vista.forward(request, response);
    }
}
```

# Gerenciando sessões

- ♦ Gerenciamento de sessão:
  - É o acompanhamento dos movimentos realizados pelos usuários em um *web site*.
- ♦ O HTTP é um protocolo que não guarda estados:
  - 1o. HTTP *request* – o *browser* requisita a página.
  - 1o HTTP *response* – o servidor retorna a página requisitada e quebra a conexão (*stateless protocol*).
  - HTTP *requests* seguintes – o *browser* requisita a página. O servidor não tem como associar o *browser* com um *request* prévio.

# Técnicas para o gerenciamento de sessões

- ◆ Existem 5 técnicas para o gerenciamento de sessões:
  - Objetos de sessão.
  - Cookies.
  - Reescrita de URL.
  - Campos ocultos.
  - Token.



# Como trabalhar com sessões

- ◆ Quando surge um novo cliente, o *container* cria um objeto sessão que fica associado somente a este cliente.
- ◆ Este objeto fica disponível enquanto o cliente estiver ativo.
- ◆ Funciona como uma *Hashtable* onde pode ser armazenada qualquer quantidade de pares chave/objeto.
- ◆ As sessões terminam por tempo de inatividade ou quando o usuário sai do *browser*.
- ◆ Pode ser acessado por qualquer *servlet* do mesmo aplicativo.
- ◆ Para recuperar um objeto previamente armazenado basta informar a sua chave.

# Como o Java mantém sessões

- ◆ No primeiro *request* do cliente o *container* gera uma identificação única para a sessão.
- ◆ No *response*, esta identificação é retornada para o cliente.
- ◆ Em cada *request* subsequente o *browser* envia a identificação.
- ◆ O *container* recebe e associa a identificação à sessão do cliente.
- ◆ É utilizado um *cookie* para armazenar a identificação.

# Como o Java mantém sessões

- ♦ O *cookie* é enviado no cabeçalho do protocolo HTTP.
- ♦ Cabe ao *servlet* somente informar ao *container* que quer criar, ou usar, o recurso sessão.
- ♦ Cabe ao *container* criar o objeto *cookie*.
- ♦ Se os *cookies* são inibidos no *web browser*, o gerenciamento de sessão não funcionará.

# Como trabalhar com sessões

- ♦ Obtendo um objeto sessão:
  - `HttpSession minhasessao=request.getSession();`
- ♦ Nos *servlets*, sempre deve ser especificado
- ♦ Pode ser obtido no método `init()`.
- ♦ Pode ser utilizado nos métodos **doGet** ou **doPost**.
- ♦ Método **getSession()** definido na interface `HttpServletRequest`.

# Como trabalhar com sessões

- ◆ Verificando se a sessão é nova:
  - **HttpSession session = request.getSession();**
  - **session.isNew()**
    - Retorna *true* se o cliente ainda não retornou com a sua identificação.
- ◆ Outra forma:
  - **HttpSession session = request.getSession(false);**
    - Retorna uma sessão pré-existente ou *null*.

# Métodos da interface HttpSession

- **getAttribute(String)**
  - Retorna um atributo específico da sessão.
- **setAttribute(String, Object)**
  - Armazena um atributo na sessão.
- **removeAttribute(String)**
  - Remove um atributo da sessão.
- **getCreationTime()**
  - Retorna a hora que a sessão foi criada.
- **getId()**
  - Retorna uma string contendo o identificador da sessão.

# Métodos da interface HttpSession

- ◆ **getLastAccessedTime()**
  - Retorna a hora em que o *container* atendeu o último *request* para uma sessão.
- ◆ **setMaxInactiveInterval()**
  - Especifica o tempo máximo , em segundos, de espera entre *requests* de uma sessão.
    - *Default*=30 minutos; Se especificar -1 não sofre *timed out*.
- ◆ **getMaxInactiveInterval()**
  - Retorna o tempo máximo, em segundos, permitido entre *requests*.
- ◆ **invalidate()**
  - Encerra a sessão. Todos os atributos são removidos.

# Especificando o *timeout* no web-xml

```
<session-config>  
  <session-timeout>20</session-timeout>  
</session-config>
```

Obs.: Este tempo é em minutos.



# ControllerCadastrarDados

```
import org.apache.commons.lang3.StringUtils;
```

```
public class ControllerCadastrarDados extends HttpServlet{

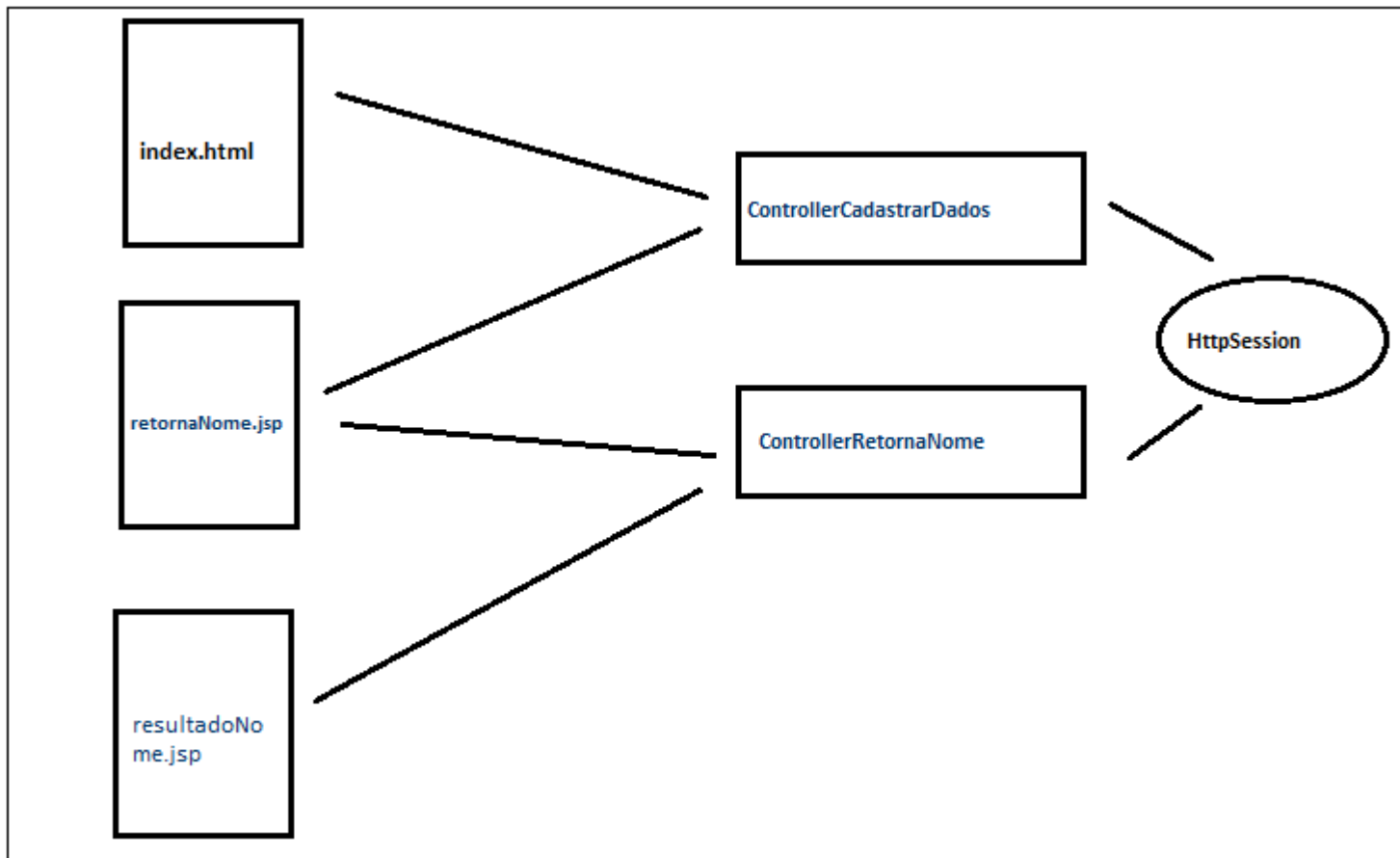
    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {
        response.setContentType("text/html;charset=UTF-8");
        String nome = request.getParameter("nome");
        String cpf = request.getParameter("cpf");
        String senha = request.getParameter("senha");
        String confirmaSenha = request.getParameter("confirmaSenha");

        if(StringUtils.isNotBlank(nome) & StringUtils.isNotBlank("cpf") & StringUtils.isNotBlank("senha") &
            StringUtils.isNotBlank("confirmaSenha")){
            request.setAttribute("cadastrado" , "Cadastrados: " + nome + " , " + cpf + " , " + senha + " , " +
            confirmaSenha);
            HttpSession minhaSessao=request.getSession(false);
            MinhaSessao.setAttribute("nome",nome);
        }
        else{
            request.setAttribute("cadastrado" ,"Você não preencheu todos os campos!");
        }
        RequestDispatcher vista = request.getRequestDispatcher("/jsp/cadastrado.jsp");
        vista.forward(request, response);
    }
}
```

# Exercício C1

- ◆ Alterar o Exercício A3 para fazer o uso de sessões;
- ◆ Caso os dados estejam preenchidos corretamente, o ControllerCadastrarDados:
  - Retornará a jsp retornaNome.jsp cujo HTML possui um botão que invocará o ControllerRetornaNome;
- ◆ O ControllerRetornaNome obterá o nome, gravado no objeto de sessão, e utilizará a jsp resultadoNome.jsp para apresentá-lo;
- ◆ Esqueçam a estética!

# Exercício C1



# *Cookies*

- ◆ São criados pelo *container* ou pelo *servlet*.
- ◆ É um par nome-valor utilizado para o servidor e o cliente estabelecerem uma forma de persistência de dados.
- ◆ No lado cliente o *browser* salva os *cookies* e os envia de volta para o servidor cada vez que solicitar uma página.
- ◆ São transferidos no cabeçalho HTTP.

# *Cookies*

- ◆ Foram especificados pelo NetScape e fazem parte do padrão Internet.
- ◆ *Cookies* podem permanecer em um *browser* por até 3 anos.
- ◆ Normalmente um *browser* aceita até 20 *cookies* por *site* e 300 *cookies* no total, podendo possuir, cada *cookie*, até 4KB de tamanho.

# *Cookies*

- ♦ Criado pela classe **Cookie** que faz parte do pacote `javax.servlet.http`.
- ♦ Construtor:
  - `Cookie(String cookieName, String cookieValue)`
- ♦ Enviado para o *browser* através do objeto *response*.
- ♦ Recuperado pelo objeto *request*.

# *Cookies*

- ◆ Alguns dos métodos
  - **addCookie(nome-do-objeto)**
    - Adiciona um *cookie* ao *response*.
  - **Cookie[] getCookies()**
    - Obtém um string de *cookies* enviados pelo browser.
  - **String getName()**
    - Obtém o nome do *cookie*.
  - **String getValue()**
    - Obtém o valor do *cookie*.

# *Cookies*

## ■ **setMaxAge(int expira)**

- Indica o tempo máximo, em segundos, de vida do *cookie*.
- Para criar um *cookie* persistente basta especificar um número  $>0$ .
- Para criar um *cookie* válido para uma única sessão deve ser especificado o valor  $-1$  (valor *default*).

## ■ **setSecure(boolean flag)**

- Indica ao *browser* que o *cookie* deve somente ser enviado usando um protocolo seguro tal como HTTPS ou SSL.





# *Cookies*

## Exemplo de código para gerar



```
Cookie cookieNome=new Cookie("nome",nome);  
Cookie cookieCPF=new Cookie("cpf",cpf);
```

```
response.addCookie(cookieNome);  
response.addCookie(cookieCPF);
```



# *Cookies*

## Exemplo de código para obter



```
Cookie[] cookies = request.getCookies();  
for(Cookie cookie : cookies){  
    out.println("Cookie Name: " + cookie.getName());  
    out.println("Cookie Value: " + cookie.getValue());  
}
```

# Como apagar *cookies*

- Para apagar *cookies* basta atribuir o valor 0 para a idade do *cookie*.

```
Cookie[ ] cookies = request.getCookies();
```

```
for (int=0; i<cookies.length; i++) {  
    Cookie cookie = cookies[i];  
    cookie.SetMaxAge(0);  
    reponse.addCookie(cookie);
```

```
}
```

# Exercício C2

- ♦ Alterar o Exercício C1 para fazer o uso de cookies;
- ♦ Caso os dados estejam preenchidos corretamente, o ControllerCadastrarDados, gravará o nome em um cookie e:
  - Retornará a jsp retornaNome.jsp cujo HTML possui um botão que invocará o ControllerRetornaNome;
- ♦ O ControllerRetornaNome obterá o nome, gravado no cookie, e utilizará a jsp resultadoNome.jsp para apresentá-lo;
- ♦ Esqueçam a estética!

# Token

- ◆ Forma alternativa de gerenciar sessões;
- ◆ Desnecessário utilizar o objeto HttpSession ou cookies;
- ◆ Pacote `java.security.SecureRandom`, disponível após Java 8, dá apoio;
- ◆ Necessário persistir o token para realizar as validações futuras;
- ◆ Pode ser em uma base de dados ou no `ServletContext`;

# Token

## Fluxo de execução

- ◆ Usuário realiza o “primeiro” acesso;
- ◆ Servlet receptor, após a verificação dos dados, verifica se já existe o token, caso exista, deleta o token antigo e cria um novo;
- ◆ Servlet atende as necessidades do usuário e devolve o token, no *response*;
- ◆ Para cada *request* subsequente, a aplicação front-end envia o token;
- ◆ Token é verificado, pelo servlet receptor, para cada *request* feito;
- ◆ O Servlet de interesse, sempre que acionado, verifica e devolve o token, podendo abortar a operação;

# Token

## Exemplo com servletContext

- ◆ Baixar a pasta CadastroPrimario para o Tomcat;
- ◆ Baixar a pasta CadastroPrimario\_Fontes;
- ◆ Iniciar o TomCat;
- ◆ Comandar:  
<http://localhost:8080/CadastroPrimario/indexC3.html>;
- ◆ Seguir o fluxo da aplicação;

Obs.: Ambas as pastas disponíveis no Google Drive.  
**Sem verificar endereço IP e timeout.**

# Token

## Classes envolvidas

- ◆ Token:
  - Classe do modelo token;
- ◆ CalculaToken:
  - Classe do model, geradora do token;
- ◆ ControllerCadastrarDadosToken:
  - Classe do controller, crítica dos dados de entrada e geração do token, colocando-o em uma Hashtable(key,value);
- ◆ ControllerRetornaNomeToken:
  - Classe controller, verifica a validade do token(key) e retorna o token e o value;



# Token

## Classe Token

```
package model;
import java.util.ArrayList;
import java.sql.Timestamp;
import java.sql.Date;

public class Token {
    private String cpf;
    private String token;
    private String enderecoIP;
    private Timestamp dataHoraUltimoAcesso;

    public String getCpf(){
        return this.cpf;
    }
    public void setCpf(String cpf){
        this.cpf=cpf;
    }
    public String getToken(){
        return this.token;
    }
    public void setToken(String token){
        this.token=token;
    }
}
```

# Token

## Classe Token

```
public String getEnderecoIP(){
    return this.enderecoIP;
}
public void setEnderecoIP(String enderecoIP){
    this.enderecoIP=enderecoIP;
}
public Timestamp getDataHoraUltimoAcesso(){
    return this.dataHoraUltimoAcesso;
}
public void setDataHoraUltimoAcesso(Timestamp dataHoraUltimoAcesso){
    this.dataHoraUltimoAcesso=dataHoraUltimoAcesso;
}
}
```

Obs.: Esta classe foi utilizada para atender a persistência do Token em uma base de dados;

# Token

## Classe CalculaToken

```
package model;

import java.security.SecureRandom;
import java.util.Base64;

public class CalculaToken {

    public static String calculaToken() {

        SecureRandom secureRandom = new SecureRandom();
        Base64.Encoder base64Encoder = Base64.getUrlEncoder();

        byte[] randomBytes = new byte[50];
        secureRandom.nextBytes(randomBytes);
        return(base64Encoder.encodeToString(randomBytes));

    }

}
```

# Token

## Classe SecureRandom

- ◆ Um número aleatório criptograficamente forte atende minimamente aos testes estatísticos de gerador de números aleatórios especificados no FIPS 140-2;
- ◆ Requisitos de segurança para módulos criptográficos, seção 4.9.1;
- ◆ SecureRandom deve produzir saída não determinística;
- ◆ Todas as sequências de saída SecureRandom devem ser criptograficamente fortes, conforme descrito em RFC 1750;

# Token

## Classe ControllerCadastrarDadosToken(trechos)

```
import java.util.HashMap;  
import java.util.Map.Entry;
```

```
public void init() throws ServletException{
```

```
    Hashtable<String,String> listaTokens= new Hashtable<String,String>();  
    getContext().setAttribute("ListaTokens",listaTokens);
```

```
}
```

```
    Hashtable<String,String> listaTokens= new Hashtable<String,String>();  
    String valorToken="";  
    boolean erro=false;
```

```
    if(StringUtils.isNotBlank(nome) & StringUtils.isNotBlank("cpf") & StringUtils.isNotBlank("senha") &  
        StringUtils.isNotBlank("confirmaSenha")){  
        request.setAttribute("cadastrado" , "Cadastrados: " + nome + ", " + cpf + ", " + senha + ", " +  
        confirmaSenha);  
        valorToken = CalculaToken.calculaToken();  
        listaTokens = (Hashtable<String,String>)getContext().getAttribute("ListaTokens");  
        listaTokens.put(valorToken,cpf); // Coloca o token gerado na hashtable  
        request.setAttribute("token",valorToken);  
    }
```

# Token

## Classe ControllerRetornaNomeToken(trechos)

```
import java.util.HashMap;  
import java.util.Map.Entry;
```

```
Hashtable<String,String> listaTokens= new Hashtable<String,String>();  
String token = request.getParameter("token");  
String cpf="";
```

```
listaTokens = (Hashtable<String,String>) getServletContext().getAttribute("ListaTokens");
```

```
if (listaTokens.containsKey(token)){  
    cpf = listaTokens.get(token);    // Obtém o valor associado a key  
}
```

```
request.setAttribute("cpf" , cpf);  
request.setAttribute("token",token);
```

Obs. Observem o conteúdo do token no navegador.

# Invocando Classes nos *servlets*

- ♦ Utilizadas para implementar a funcionalidade Model do MVC;
- ♦ Deve haver uma preocupação com o sincronismo devido aos múltiplos acessos dos *servlets*;
- ♦ *Servlets* invocam estas classes de forma convencional;

# Invocando Classes nos *servlets*

## Exemplo

```
if(StringUtils.isNotBlank(nome) & StringUtils.isNotBlank("cpf") & StringUtils.isNotBlank("senha") &
    StringUtils.isNotBlank("confirmaSenha")){
    request.setAttribute("cadastrado", "Cadastrados: " + nome + ", " + cpf + ", " + senha + ", " +
    confirmaSenha);
    valorToken = CalculaToken.calculaToken();
    listaTokens = (Hashtable<String,String>)getServletContext().getAttribute("ListaTokens");
    listaTokens.put(valorToken,cpf);      // Coloca o token gerado na hashtable
    request.setAttribute("token",valorToken);
}
```



# MySQL

- ♦ Introdução ao MySQL.
- ♦ Interagindo com o MySQL.
- ♦ Iniciando e parando o servidor MySQL.
- ♦ Trabalhando com o programa MySQL.
- ♦ Como criar, selecionar e apagar um *database*.
- ♦ Como criar e apagar uma tabela.
- ♦ Como inserir ou carregar dados em uma tabela.
- ♦ JDBC.

# MySQL

- ◆ *Open source database* que pode ser baixado gratuitamente de [www.mysql.com](http://www.mysql.com).
- ◆ É um dos mais rápidos *databases* relacionais do mercado.
- ◆ Comparado a outros *databases*, é fácil de instalar e utilizar.
- ◆ Roda nos sistemas operacionais Windows, Unix, Solaris e macOS.
- ◆ Suporte à integridade referencial, *subqueries* e transações.

# MySQL

- ◆ Suporta SQL que é a linguagem padrão para trabalhar com *databases* relacionais.
- ◆ Suporta acessos de múltiplos clientes e inúmeras linguagens tais como Java, Perl, PHP, Python, C e JavaScript(node.js).
- ◆ Pode fornecer acesso aos seus dados via intranet ou internet.
- ◆ Pode restringir o acesso a seus dados somente para usuários autorizados.

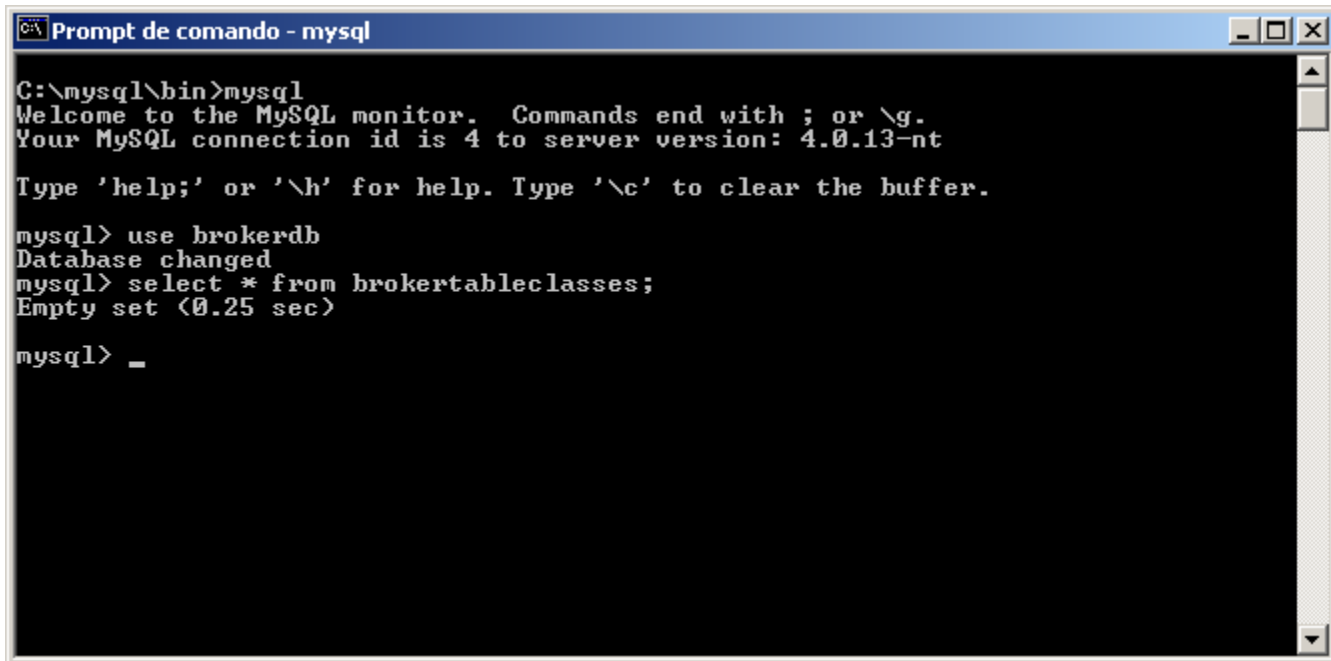
# Iniciando e parando o MySQL

- ♦ O MySQL *database server* pode ser iniciado logo após a entrada do sistema operacional.
- ♦ O MySQL fornece uma GUI que permite ver algumas informações sobre o *database server*.
- ♦ Para parar o *database server* basta ir ao ícone do MySQL e parar o serviço.

# O programa MySQL

- ♦ Utilizado para realizar operações sobre o *database* MySQL(hardcore!).
- ♦ Pode ser conectado, local ou remotamente, ao *database server*.
- ♦ Executa operações MySQL e emite comandos SQL.

# O programa MySQL



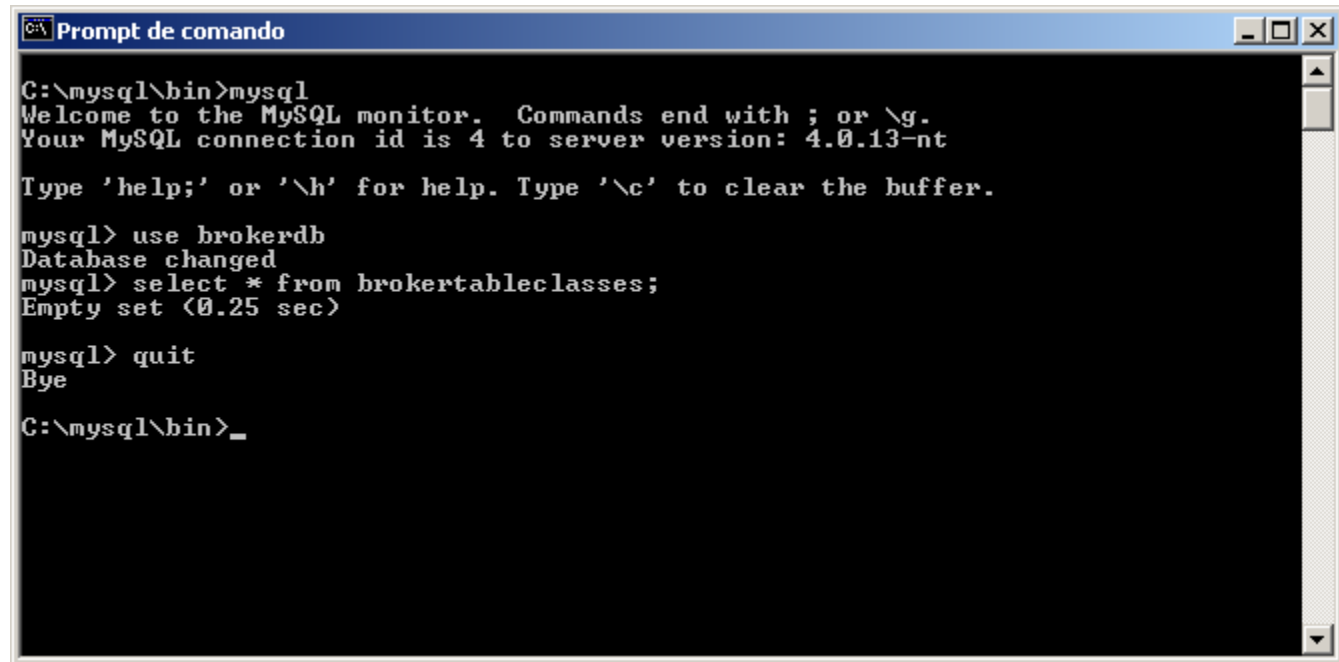
```
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.0.13-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use brokerdb
Database changed
mysql> select * from brokertableclasses;
Empty set (0.25 sec)

mysql> _
```

# Parando o programa MySQL



```
C:\mysql\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 4.0.13-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use brokerdb
Database changed
mysql> select * from brokertableclasses;
Empty set (0.25 sec)

mysql> quit
Bye
C:\mysql\bin>_
```

# Como criar, selecionar e apagar um *database*

- ♦ Criando um database:
  - `mysql> create database dbteste;`
- ♦ Como listar os nomes de todos os databases:
  - `mysql> show databases;`
- ♦ Como selecionar um database para uso:
  - `mysql> use dbteste;`
- ♦ Como apagar um database:
  - `mysql> drop database dbteste;`

**Obs.: Todas as operações podem ser feitas pela GUI Tools.**



# Como criar e apagar uma tabela

## ◆ Criando uma tabela:

```
mysql> create table Funcionario(  
-> ident int not null auto_increment,  
-> nome varchar(50),  
-> endereco varchar(50),  
-> sexo char(1),  
-> primary key(ident));
```

# Como criar e apagar uma tabela

- ♦ Como listar todas as tabelas em um *database*:
  - mysql> **show tables;**
- ♦ Como apagar uma tabela:
  - mysql> **drop table Funcionario;**
- ♦ Apresentando a descrição dos campos de uma tabela:
  - mysql> **describe Funcionario;**

# Como inserir ou carregar dados em uma tabela

## ◆ Inserindo dados:

```
mysql> insert into funcionario(nome, endereco,sexo)
-> values
-> ("Jose", "Rua abc 210", "M"),
-> ("Maria", "Rua def 220","F");
```

## ◆ Carregando dados:

```
mysql> load data local infile "c:/cursoweb/Users.txt" into table User;
```

- A tabela Users.txt deve estar formatada.

# Alguns comandos SQL

## ◆ Seleccionando todas colunas de uma tabela:

```
SELECT *  
FROM tabela-1  
[WHERE critério-de-seleção]  
[ORDER BY campo-1 [ASC|DESC] [, campo-2 [ASC|DESC] ...]]
```

## ◆ Exemplos:

- SELECT \* FROM Funcionario;
- SELECT \* FROM Funcionario WHERE ident<3;

# Alguns comandos SQL

- ◆ Selecionando algumas colunas de uma tabela:

```
SELECT campo-1 [, campo-2] ...  
FROM tabela-1  
[WHERE critério-de-seleção]  
[ORDER BY campo-1 [ASC| DESC] [, campo-2 [ASC| DESC] ...]]
```

- ◆ Exemplo:

- **SELECT nome, sexo FROM Funcionario  
WHERE ident>2 ORDER BY nome ASC;**

# Selecionando dados de múltiplas tabelas

- ◆ Operação conhecida como *join*.
- ◆ Existem 2 tipos de *join*:
  - INNER– somente são selecionadas as linhas daquelas colunas cujos conteúdos são idênticos.
  - OUTER – todas as linhas de uma tabela são selecionadas mesmo se não existir correspondente na outra tabela.
    - **LEFT OUTER** – são incluídos todos os registros da primeira tabela.
    - **RIGTH OUTER** – são incluídos todos os registros da segunda tabela.

# Selecionando dados de múltiplas tabelas

## ◆ Sintaxe:

```
SELECT campo-1 [, campo-2] ...  
FROM tabela-1  
    {INNER | LEFT OUTER | RIGHT OUTER} JOIN tabela-2  
    ON tabela-1.campo-1 {= | < | > | <= | >= | <> } tabela-2.campo-2  
[WHERE critério-de-seleção]  
[ORDER BY campo-1 [ASC| DESC] [, campo-2 [ASC| DESC] ...]]
```

## ◆ Exemplo:

Tabela\_Empregados

Codigo_Empregado	Nome
01	Silva, João
02	Oliveira, Adriano
03	Oliveira, Marcos
04	Costa, Marina

Tabela\_Vendas

Identificação_Produto	Produto	Código_Empregado
234	Impressora	01
657	Mesa	03
865	Cadeira	03

# Selecionando dados de múltiplas tabelas

```
SELECT Tabela_Empregados.Nome, Tabela_Vendas.Produto  
FROM Tabela_Empregados  
INNER JOIN Tabela_Vendas  
ON Tabela_Empregados.Codigo_Empregado=Tabela_Vendas.Codigo_Empregado
```

<b>Nome</b>	<b>Produto</b>
Silva, João	Impressora
Oliveira, Marcos	Mesa
Oliveira, Marcos	Cadeira



# Selecionando dados de múltiplas tabelas

```
SELECT Tabela_Empregados.Nome, Tabela_Vendas.Produto  
FROM Tabela_Empregados  
LEFT JOIN Tabela_Vendas  
ON Tabela_Empregados.Codigo_Empregado= Tabela_Vendas.Codigo_Empregado
```

<b>Nome</b>	<b>Produto</b>
Silva, João	Impressora
Oliveira, Adriano	
Oliveira, Marcos	Mesa
Oliveira, Marcos	Cadeira
Costa, Marina	

# Selecionando dados de múltiplas tabelas

```
SELECT Tabela_Empregados.Nome, Tabela_Vendas.Produto  
FROM Tabela_Empregados  
RIGHT JOIN Tabela_Vendas  
ON Tabela_Empregados.Codigo_Empregado= Tabela_Vendas.Codigo_Empregado
```

<b>Nome</b>	<b>Produto</b>
Silva, João	Impressora
Oliveira, Marcos	Mesa
Oliveira, Marcos	Cadeira

# Como inserir, atualizar e apagar dados

- ◆ Inserindo dados:

- INSERT INTO tabela [(lista-de-campos)]  
VALUES (lista-de-valores)

- ◆ Exemplo:

- INSERT INTO Vendas (UserID,  
CodigoProduto) VALUES (1, 'Mic01')

# Como inserir, atualizar e apagar dados

## ♦ Atualizando dados:

- **UPDATE tabela SET expressão-1 [, expressão-2]...  
WHERE critério-de-seleção**

## ♦ Exemplos:

- **UPDATE Usuario SET Nome = 'Rafael',  
WHERE EmailAddress='rcardozo@globo.com'**
- **UPDATE Produtos SET PrecoProduto=40.95  
WHERE PrecoProduto=36.50**

# Como inserir, atualizar e apagar dados

## ♦ Apagando dados:

- **DELETE FROM tabela**  
**WHERE critério-de-seleção**
- **DELETE FROM tabela (apaga tudo)**
- **DELETE \* FROM tabela (apaga tudo)**

## ♦ Exemplos:

- **DELETE FROM Usuario WHERE**  
**EmailAddress='rcardozo@globo.com'**
- **DELETE FROM Download WHERE DataDownload**  
**< '2005-08-02'**

# Java Database Connectivity (*JDBC*)

- ◆ Tecnologia que permite o acesso e a manipulação de um banco de dados em Java.
- ◆ Faz parte do pacote `java.sql`.
- ◆ Existem vários tipos de *drivers*, um para cada linguagem.
- ◆ Versão corrente para o Java é a versão 5.

# O Java e os *databases*

- ♦ Carregando o *database driver*.
- ♦ Conectando-se a um *database*.
- ♦ Escrevendo uma declaração.
- ♦ Trabalhando com um *result set*.
- ♦ Recuperando dados de um *result set*.
- ♦ Inserindo, atualizando e apagando dados.

# Java Database Connectivity (*JDBC*)

- ◆ Onde obter *database drivers* ?
  - <https://dev.mysql.com/downloads/connector/j/5.1.html>
- ◆ O *database driver* deve ser copiado para:
  - No Tomcat;
    - Diretório lib
  - No Java
    - Diretório jdk → jre → lib → ext

Obs.: Disponibilizei no Google Drive. Copiar agora!



# Carregando o *database driver* do MYSQL

```
try {  
    Class.forName("com.mysql.jdbc.Driver");  
}  
catch (ClassNotFoundException e) {  
    System.out.println(e); /* Driver não encontrado */  
}
```

Obs: Pode ser incluído no método `init()` do *servlet*.

**Melhor opção é delegar a tarefa para o TomCat.**

# Carregando o *database driver* do MySQL pelo TomCat

- Nome do *driver* incluído no context.xml, sob a *tag* Resource;
- Tag <Resource> define também:
  - Login e senha de acesso ao *database*;
  - Nome do database e sua localização;
  - Nome do *resource*, a ser referenciado pela aplicação;

```
<Resource name="jdbc/poolDBCursor" auth="Container" type="javax.sql.DataSource"
  maxActive="90" maxIdle="50" maxWait="20000" validationQuery="Select 1"
  username="xxxxxxxxxxx" password="yyyyyyyyyyy"
  driverClassName="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost/coursouerjfaperj?autoReconnect=true&zeroDateTimeBehavior=convertToNull"/>
```

## Outros métodos do DriverManager

- ◆ `isClosed()` – utilizado para verificar se a conexão está fechada.
- ◆ `createStatement()` – usado para criar um objeto `Statement` que enviará declarações SQL ao banco de dados.
- ◆ `preparedStatement()` – usado para enviar múltiplas declarações para o banco de dados.

# Criando uma declaração

- ♦ Utilizada para enviar declarações SQL para o database.
  - Ex.: `Statement stmt = con.createStatement();`
- ♦ Para fazer **consultas** ao database utiliza-se o método **`executeQuery()`**.
- ♦ Para fazer atualizações (*update, delete ou insert*) utiliza-se o método **`executeUpdate()`**.
- ♦ Esta declaração também deve ser fechada.
  - Ex: `stmt.close()`.

# Recuperando informações

- ◆ É gerado um objeto *result set* contendo as colunas e as linhas com o resultado solicitado.
- ◆ Existem métodos específicos para navegar pelo *result set* e para recuperar os campos da tabela produzida.
  - Ex.: `ResultSet produtos = stmt.executeQuery(“SELECT * FROM tabela_produtos”);`

# Navegando pelo *result set*

## Alguns Métodos

- ◆ `next()` – move o cursor para a próxima linha do *result set*.
- ◆ `last()` – move o cursor para a última linha do *result set*.
- ◆ `first()` – move o cursor para a primeira linha do *result set*.
- ◆ `close()` – fecha o *result set*.
- ◆ `getRow()` – retorna um inteiro que identifica a linha corrente do *result set*.

## Recuperando campos do *result set*

- ◆ `getXXX(intColumnIndex)` – retorna dados de uma coluna identificada pelo seu número.
  - Ex.: `produtos.getString(1)`
- ◆ `getXXX(StringColumnName)` – retorna dados de uma coluna identificada pelo seu nome.
  - Ex.: `produtos.getString(“nomedoproduto”)`

## Recuperando campos do *result set*

- ♦ O método `getXXXpode` ser usado para retornar todos os 8 tipos primitivos dados.
- ♦ O `XXX` indica o tipo primitivo do dado.
- ♦ Também pode ser usado para retornar *strings* (`getString`), datas (`getDate`) e hora (`getTime`).



# *Prepared statements*

- ◆ Quando a aplicação envia uma declaração, o *database server* realiza as seguintes tarefas:
  - Checa se há erros de sintaxe.
  - Prepara um plano para executar a declaração.
  - Executa a declaração.
- ◆ Se a mesma declaração é enviada novamente o *database server* não checa a sintaxe e nem prepara outro plano;
- ◆ Com isto aumenta-se o desempenho das operações no *database*.

# *Prepared statements* consultando

```
PreparedStatement ps;
```

```
ResultSet rs;
```

```
String consulta = "SELECT * from agregadoanimal where id=? " ;
```

```
    ps = con.prepareStatement(consulta);
```

```
    ps.setInt(1, animal.getId());
```

```
    rs = ps.executeQuery();
```

```
    while(rs.next()){
```

```
        animal.setEspecie(rs.getString("especie"));
```

```
        animal.setRaca(rs.getString("raca"));
```

```
        animal.setNome(rs.getString("nome"));
```

```
        animal.setCaracteristica(rs.getString("caracteristica"));
```

```
        animal.setHabilitado(rs.getBoolean("habilitado"));
```

```
    }
```

# *Prepared statements* atualizando

```
String altera = "UPDATE agregadoanimal SET "
```

```
+ " especie = ?, "  
+ " raca = ?, "  
+ " nome = ?, "  
+ " caracteristica = ?, "  
+ " idcolaborador = ?, "  
+ " habilitado= ? "  
+ " WHERE id = ? ";
```

```
ps = con.prepareStatement(altera);  
ps.setString(1,animal.getEspecie());  
ps.setString(2,animal.getRaca());  
ps.setString(3,animal.getNome());  
ps.setString(4,animal.getCaracteristica());  
ps.setString(5,animal.getIdColaborador());  
ps.setBoolean(6,animal.getHabilitado());  
ps.setInt(7,animal.getId());  
ps.executeUpdate();
```

# *Prepared statements* inserindo

```
String insere = "Insert into agregadoanimal(especie, raca, nome, caracteristica, idcolaborador,habilitado) " +  
    "Values (?,?,?,?,?,?)";  
ps = con.prepareStatement(insere);  
ps.setString(1,animal.getEspecie());  
ps.setString(2,animal.getRaca());  
ps.setString(3,animal.getNome());  
ps.setString(4,animal.getCaracteristica());  
ps.setString(5,animal.getIdColaborador());  
ps.setBoolean(6,animal.getHabilitado());  
ps.executeUpdate();
```

# *Prepared statements* excluindo

```
String exclui = "DELETE from agregadoanimal WHERE id = ? ";
```

```
ps = con.prepareStatement(exclui);
```

```
ps.setInt(1,animal.getId());
```

```
ps.executeUpdate();
```

# Tratando Conexões

- ◆ Abrir e fechar conexões a todo instante é custoso.
- ◆ Existe forte recomendação para sempre fechar uma conexão aberta.
- ◆ Uma conexão aberta é utilizada, exclusivamente, por uma única *thread* do *servlet*.
- ◆ Os modernos bancos de dados suportam conexões simultâneas.
- ◆ Solução conciliatória: criação de um *pool* de conexões abertas.

# Conectando-se a um *database*

## Exemplo - Classe AcessoBanco

- ◆ Classe do modelo utilizada para fazer todas as conexões ao *database*.
- ◆ Utiliza-se o método `getConnection`, da classe `DriverManager`.

```
package model;
```

```
import java.sql.*;  
import javax.naming.Context;  
import javax.naming.InitialContext;  
import javax.sql.DataSource;
```

```
public class AcessoBanco{
```

```
    public static Connection conectar() {  
        Connection con = null;  
        try {  
            Context ctx = (Context) new InitialContext().lookup("java:comp/env");  
            con = ((DataSource) ctx.lookup("jdbc/poolIDBCursos")).getConnection();  
            return con;  
        } catch (Exception e) {  
            e.printStackTrace(System.err);  
            return null;  
        }  
    }
```

# Conectando-se a um *database*

## Classe AcessoBanco

```
}  
public static boolean desconectar(Connection con,PreparedStatement ps,ResultSet rs) {  
    try {  
        if (rs!=null){  
            rs.close();  
        }  
        if (ps!=null){  
            ps.close();  
        }  
        if (con!=null){  
            con.close();  
        }  
        return true;  
    } catch (SQLException e) {  
        e.printStackTrace();  
        return false;  
    }  
}  
  
}
```



# MySQL GUI Tools

- ◆ Entrar no Gui Tools com:
  - Execução do comando **mysql query browser**:
    - Senha **admin**
- ◆ Em *tools*, selecionar MySQL Administrator;
- ◆ Comandar o *restore* da base Cursouerjfaperj;
- ◆ Para dar autorização ao acesso à base de dados pela aplicação:
  - `grant all privileges on databasename.* to username@localhost identified by "password";`
  - Dirá que o comando está depreciado, ignore;

# Como depurar problemas em *servlets*

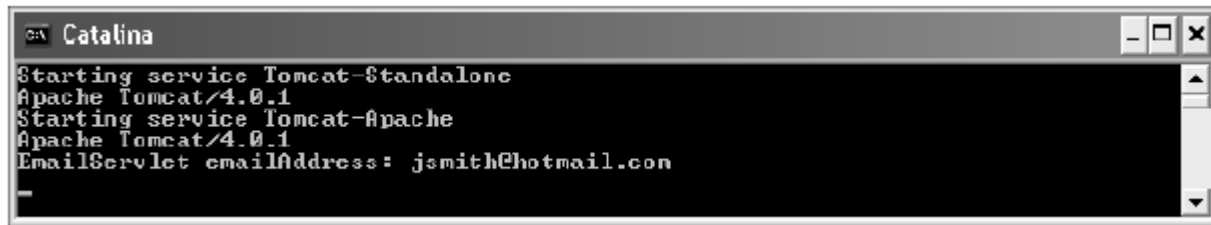
- ♦ O *servlet* não compila:
  - Certifique-se que o compilador tem acesso aos arquivos JAR para todas as APIs necessárias.
  - Certifique-se que o *classpath* está apontando para o diretório que contém o pacote correto.
- ♦ O *servlet* não executa:
  - Certifique-se que o *web server* está executando (Tomcat).
- ♦ As mudanças feitas não estão sendo apresentadas:
  - Certifique-se que a opção de *reloading*(no Tomcat) foi ativada.

# Como depurar problemas em *servlets*

- ◆ Impressão é feita na console do *container* (Tomcat).
- ◆ Podem ser utilizados os métodos *println*, *print* ou *printf* dos objetos *System.out* ou *System.err*.
- ◆ Recomenda-se incluir, na mensagem, o nome da classe e do método que estão sendo depurados.
- ◆ Pode ser usado, também, método *printStackTrace()*, herdado pela classe *Exception*.

# Como depurar problemas em *servlets*

A console do TomCat:

A screenshot of a Windows-style console window titled "Catalina". The window has a standard title bar with minimize, maximize, and close buttons. The text inside the console is as follows:

```
Starting service Tomcat-Standalone
Apache Tomcat/4.0.1
Starting service Tomcat-Apache
Apache Tomcat/4.0.1
EmailServlet emailAddress: jsmith@hotmail.com
```

Código para realizar a impressão:

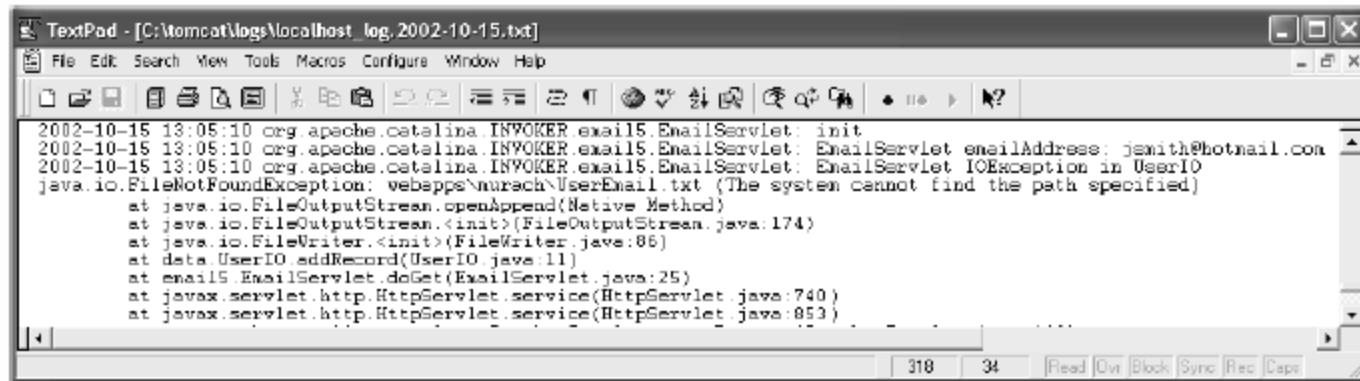
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException{
    // code
    String emailAddress = request.getParameter("emailAddress");
    System.out.println("EmailServlet emailAddress: " + emailAddress);
    // code
}
```

# Como depurar problemas em *servlets* arquivo *log*

- ◆ Para tal, existem dois métodos da classe `HttpServlet`:
  - `log(String mensagem)`
    - Escreve uma mensagem no *log* do *container*.
  - `log(String mensagem, Throwable t)`
    - Escreve uma mensagem no *log* do *container* registrando, também, os métodos chamados para executar determinado comando.

# Como depurar problemas em *servlets* arquivo *log*

Um arquivo log do TomCat:



```
TextPad - [C:\tomcat\logs\localhost_log.2002-10-15.txt]
File Edit Search View Tools Macros Configure Window Help
2002-10-15 13:05:10 org.apache.catalina.INVOKER_email15.EmailServlet: init
2002-10-15 13:05:10 org.apache.catalina.INVOKER_email15.EmailServlet: EmailServlet emailAddress: jsmith@hotmail.com
2002-10-15 13:05:10 org.apache.catalina.INVOKER_email15.EmailServlet: EmailServlet IOException in UserIO
java.io.FileNotFoundException: webapps\nurach\UserEmail.txt (The system cannot find the path specified)
    at java.io.FileOutputStream.openAppend(Native Method)
    at java.io.FileOutputStream.<init>(FileOutputStream.java:174)
    at java.io.FileWriter.<init>(FileWriter.java:86)
    at data.UserIO.addRecord(UserIO.java:11)
    at email15.EmailServlet.doGet(EmailServlet.java:25)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:740)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
```

Código para escrever em um arquivo log:

```
String emailAddress = request.getParameter("emailAddress");
log("EmailServlet emailAddress: " + emailAddress);
User user = new User(firstName, lastName, emailAddress);
try{
    UserIO.addRecord(user, file);
}
catch(IOException ioe){
    log("EmailServlet IOException in UserIO", ioe);
}
```

# AJAX JSON

- ◆ Copiar para o Tomcat, e para o Java JDK, o pacote `json_simple.jar`;
- ◆ No código utilizar:
  - `import org.json.simple.JSONObject;`
  - `import org.json.simple.JSONArray(se for utilizar JSON array);`

# AJAX JSON

- ◆ Exemplo de uso do objeto JSON com servlet:

- Uso de um par chave-valor:

```
JSONObject saidaJSON;
```

```
PrintWriter saida = response.getWriter();
```

```
saidaJSON = new JSONObject();
```

```
saidaJSON.put("erro",erro);
```

```
saidaJSON.put("msg",msg);
```

```
saidaJSON.put("nome",nome);
```

```
saida.print(saidaJSON);
```

```
saida.flush();
```

Obs.: Trecho retirado de ControllerValidaDadosEntrada.java.



# AJAX JSON

- ◆ Exemplo de uso do objeto JSON com servlet:
  - Retorno do JSON caso os dados estejam corretamente preenchidos:

```
{"msg":"Operação realizada com sucesso!","erro":false,"nome":"RAIMUNDO NONATO"}
```

# AJAX JSON

- ◆ Exemplo de uso do array de objetos JSON com servlet:

```
JSONObject saidaJSON;  
PrintWriter saida = response.getWriter();  
  
saidaJSON = new JSONObject();  
JSONArray listaCarrosJSON = new JSONArray();  
  
JSONObject documentoJSON;  
  
listaCarros = DAOCarros.consultarPorMarca(carros);
```

# AJAX JSON

## ◆ Exemplo de uso do array de objetos JSON com servlet:

```
if (listaCarros!=null && listaCarros.size()>0){
    for (int i=0; i<listaCarros.size(); i++){
        documentoJSON = new JSONObject();
        documentoJSON.put("id",listaCarros.get(i).getId());
        documentoJSON.put("marca",listaCarros.get(i).getMarca());
        documentoJSON.put("modelo",listaCarros.get(i).getModelo());
        documentoJSON.put("ano",listaCarros.get(i).getAno());
        documentoJSON.put("cor",listaCarros.get(i).getCor());
        listaCarrosJSON.add(documentoJSON);
    }
    msg="Operação realizada com sucesso!";
}
else{
    msg="Nenhuma ocorrência encontrada para o tipo selecionado!";
}
saidaJSON.put("Carros",listaCarrosJSON);
```

Obs.: Trecho retirado de ControllerEnviarMenu.java.

# AJAX JSON

- ◆ Exemplo de uso do array de objetos JSON com servlet:
  - JSON retornando o array de objetos, Carros, conforme exemplo abaixo:

```
{"msg":"Operação realizada com sucesso!","Carros":  
  [{"marca":"Ford","ano":2011,"cor":"Cinza","id":12,"modelo":"Ka"},  
  {"marca":"Ford","ano":2021,"cor":"Preta","id":13,"modelo":"Taurus"},  
  {"marca":"Ford","ano":2003,"cor":"Vermelha","id":14,"modelo":"EcoS  
port"}], "erro":false}
```

# Aplicação CursoJavaScript

- ◆ Classes do modelo:
  - AcessoBanco, AcessoBancoCeps, Carros e CEP.
- ◆ Classes de Controle:
  - ControllerPegaCEP, ControllerValidaDadosEntrada, ControllerEnviarMenu e ControllerEnviarDetalhes.
- ◆ Classes do DAO(Data Access Object):
  - DAOCarros e DAOCEP.

Obs.: Baixar do Google Drive a pasta [CursoJavaScript\\_Fontes e CursoJavaScript\(para o TomCat\)](#).

# Aplicação CursoJavaScript

- ◆ Digitem: <http://152.92.181.90:8080/CursoJavaScript/indexCursoWeb.html>
- ◆ 1a. classe do Fluxo do negócio:
  - ControllerPegaCEP, que invoca as classes CEP, do modelo, e a DAOCep, do DAO. Cria objeto JSON e retorna;
  - A classe DAOCep invoca a classe AcessoBancoCeps para conectar-se a base de dados correios;
- ◆ 2a. classe do Fluxo do negócio:
  - ControllerValidaDadosEntrada, se OK, cria objeto JSON e retorna;
- ◆ 3a. classe do Fluxo do negócio:
  - ControllerEnviarMenu, que invoca as classes Carros, do modelo, e a DAOCarros, do DAO. Cria objeto JSON e retorna;
  - A classe DAOCarros invoca a classe AcessoBanco para conectar-se a base de dados cursouerjfaperj;

# Aplicação CursoJavaScript

## 4a. classe do Fluxo do negócio:

ControllerEnviarDetalhes, que invoca as classes Carros, do modelo, e a DAOCarros, do DAO. Cria objeto JSON e retorna;

- A classe DAOCarros invoca a classe AcessoBanco para conectar-se a base de dados cursouerjfaperj;

Obs.: Não foi feito nenhum controle de sessão.

**Vamos analisar todas estas classes agora, com carinho!**

**Todas as classes, com exceção das do modelo, com tratamento de erros!**

# Codificando variáveis de instância

- ◆ Uma variável de instância pertence a uma instância de um *servlet*.
- ◆ É compartilhada por qualquer *thread* do *servlet*.
- ◆ Duas *threads* podem conflitar quando tentarem modificar a mesma variável de instância ao mesmo tempo.
- ◆ Para sincronizar o acesso ao bloco de código deve ser usada a palavra-chave *synchronized*.



# Protegendo variáveis de instância

```
package com.exemplo.web;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ContaAcessos extends HttpServlet{

    private int contadorGeral;

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException,
        ServletException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        synchronized(this){
            contadorGeral++;
        }

        out.println("<br> Esta página foi acessada " + contadorGeral + " vezes </br>");
    }
}
```

# Codificando *servlets thread-safe*

- ◆ Um *servlet thread-safe* é aquele que trabalha confiavelmente mesmo quando mais de uma cópia da *thread* está ativa.
- ◆ Para sincronizar o acesso ao método ou ao bloco de código deve ser utilizada a palavra-chave *synchronized*.
- ◆ Implementando a interface `SingleThreadModel` o código inteiro do *servlet* torna-se serializável.

# Codificando *servlets thread-safe*

## Um bloco de código *synchronized*

```
synchronized(this){
    acessCount++;
    if (acessCount == 1000){
        LogUtil.logToFile("Atingimos 1000 usuários dia "
            + new java.util.Date());
    }
}
```

## Um método *synchronized*

```
public static synchronized int addRecord(Connection connection, User user)
    throws SQLException{

    String query =
        "INSERT INTO User " +
        "(EmailAddress, FirstName, LastName) " +
        "VALUES ('" + SQLUtil.encode(user.getEmailAddress()) + "', " +
        "'" + SQLUtil.encode(user.getFirstName()) + "', " +
        "'" + SQLUtil.encode(user.getLastName()) + "')";

    Statement statement = connection.createStatement();
    int status = statement.executeUpdate(query);
    statement.close();
    return status;
}
```

## Um *servlet* que não permite múltiplos acessos de *threads*

```
public class EmailServlet3 extends HttpServlet implements SingleThreadModel {
}
```

# Exercício C3

- ♦ Alterar a Aplicação CursoJavaScript para que suporte o gerenciamento de sessões, uma vez que o usuário poderá comprar o veículo ofertado, necessário lembrar dele(funcionalidade futura!);

Obs. Utilizar o objeto HttpSession;

# Exercício C4

## Praticando o CRUD

- ♦ CRUD, acrônimo de Create, Read, Update e Delete;
- ♦ Alterar a Aplicação CursoJavaScript para que:
  - Faça o gerenciamento de sessões por *token*;
  - Implemente as funcionalidades do CRUD;
  - O CRUD será utilizado para manter a tabela Carros;

Obs. O ControllerLogin(de 1o. acesso) já está escrito e funcionando.  
Passos a seguir no próximo *slide*.

# Exercício C4

## Praticando o CRUD

- ♦ Entrar no MySQL GUI Tools e criar a tabela de *tokens*(colocada no Google Drive):
  - Arquivo=> New Script Tab=> copiar o conteúdo => executar
- ♦ Copiar, para o TomCat, a aplicação CursoJavaScript;
- ♦ Baixar, da *web*, o Postman;
- ♦ Testar o Postman e a geração do *token* pelo ControllerLogin:
  - Selecionar o método POST;
  - Na URL, codificar `http://localhost:8080/CursoJavaScript/Login`;
  - Selecionar Body e x-www-form-urlencoded;
  - Digitar para a *key* => cpf;
  - Digitar para o *value* => o cpf;
  - Clicar em SEND;

# Exercício C4

## Praticando o CRUD

- ◆ No retorno do POSTMAN, virá o token;
- ◆ Em todas as chamadas subsequentes, este *token* deverá ser informado. A *key* é *token*. O *value* é o valor do *token* retornado;
- ◆ Vocês escreverão os *controllers* necessários para inserir, atualizar, apagar e consultar um carro;
- ◆ As classes do modelo e do DAO já estão codificadas;
- ◆ Vamos analisar estas classes e as classes ligadas ao **token**;
- ◆ Terão que fazer sózinhos(sem ajuda do colega!) pois este é o teste final para aqueles que ainda não possuem uma ideia do que fazer;
- ◆ Podem terminar na nossa aula de amanhã, reservada para isto;
- ◆ Na próxima 3a feira(11/04) abordaremos recepção de arquivos, geração de pdfs e geração de excel;

# SQL

## Homologando uma transação

- ◆ `setAutoCommit()` – indica se a homologação se dará por declaração individual(*true*) ou por um conjunto de declarações(*false*).
- ◆ O valor *default* é *true*.
- ◆ `commit()` – utilizado para homologar a transação, caso o `setAutoCommit` seja *false*.
- ◆ `rollback()` – utilizado para retornar a transação a posição anterior.



# Homologando uma transação

## Exemplo

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class CommitAndRollback {

    public static void main(String[] args) {

        Connection connection = null;
        try {
            String driverName = "oracle.jdbc.driver.OracleDriver";
            Class.forName(driverName);
            String serverName = "localhost";
            String serverPort = "1521";
            String sid = "mySchema";
            String url = "jdbc:oracle:thin:@" + serverName + ":" + serverPort + ":" + sid;

            String username = "username";
            String password = "password";
```

# Homologando uma transação

## Exemplo

```
connection = DriverManager.getConnection(url, username, password);
System.out.println("Successfully Connected to the database!");
} catch (ClassNotFoundException e) {
    System.out.println("Could not find the database driver " + e.getMessage());
} catch (SQLException e) {
    System.out.println("Could not connect to the database " + e.getMessage());
}
try {
    connection.setAutoCommit(false);
    // Operações SQL...
    // Operações SQL
    connection.commit();
    System.out.println("Successfully committed changes to the database!");
} catch (SQLException e) {
    try {
        connection.rollback();
        System.out.println("Successfully rolled back changes from the database!");
    } catch (SQLException e1) {
        System.out.println("Could not rollback updates " + e1.getMessage());
    }
} } }
```

# Recepção de Arquivos

## MultipartRequest class

- ◆ Classe utilitária para lidar com solicitações de dados tipo multipart/form-data, para *uploads* de arquivos;
- ◆ Contida no pacote `cos.jar(com.oreilly.servlet)`;
- ◆ Um de seus construtores:
  - `MultipartRequest(javax.servlet.http.HttpServletRequest request, java.lang.String saveDirectory, int maxPostSize)`
    - Objeto request, diretório destino, tamanho máximo permitido do arquivo.
- ◆ Página de descrição completa em:
  - <http://www.servlets.com/cos/javadoc/com/oreilly/servlet/MultipartRequest.html>

**Vamos visitá-la.**

# MultipartRequest class

## métodos

- ♦ **getContentType(nome)** - Retorna o tipo de conteúdo do arquivo especificado (conforme fornecido pelo navegador do cliente) ou nulo se o arquivo não foi incluído no *upload*.
- ♦ **getFile(nome)** - Retorna um objeto File para o arquivo especificado salvo no sistema de arquivos do servidor ou nulo se o arquivo não foi incluído no *upload*.
- ♦ **getFileNames()** - Retorna os nomes de todos os arquivos carregados como uma Enumeração de Strings.
- ♦ **getFileSystemName(nome)** - Retorna o nome do sistema de arquivos do arquivo especificado ou nulo se o arquivo não foi incluído no *upload*.
- ♦ **getOriginalFileName(nome)** - Retorna o nome do sistema de arquivos original do arquivo especificado (antes da aplicação de qualquer política de renomeação) ou nulo se o arquivo não foi incluído no *upload*.
- ♦ **getParameter(nome)** - Retorna o valor do parâmetro nomeado como String, ou nulo se o parâmetro não foi enviado ou foi enviado sem valor.
- ♦ **getParameterNames()** - Retorna os nomes de todos os parâmetros como uma Enumeração de Strings.
- ♦ **getParameterValues(nome)** - Retorna os valores do parâmetro nomeado como um array String, ou nulo se o parâmetro não foi enviado.

# MultipartRequest class

## Exemplo de uso - trechos

```
import java.io.*;
import com.oreilly.servlet.MultipartRequest;
```

```
// Prepara o pathname para o corrente Sistema Operacional
```

```
String SO = System.getProperty("os.name");
```

```
if (SO.indexOf("Windows")>=0){
```

```
    file = new File("C:/Arquivos de programas/Apache Software Foundation/Tomcat  
    7.0/webapps/UDT/qualitec/");
```

```
    pathName="C:/Arquivos de programas/Apache Software Foundation/Tomcat  
    7.0/webapps/UDT/qualitec/";
```

```
}
```

```
else{
```

```
    file= new File("/opt/tomcat-apps/webapps/UDT/qualitec/");  
    pathName="/opt/tomcat-apps/webapps/UDT/qualitec/";
```

```
}
```

```
if (!file.isDirectory()) {  
    file.mkdir();
```

```
}
```

# MultipartRequest class

## Exemplo de uso - trechos

```
String filesystemName="";  
String sufixo="";  
String nomeArquivo="";
```

**// O arquivo já será gravado na pasta, após a instrução abaixo**

```
MultipartRequest mult = new MultipartRequest(request,pathname,50000000);
```

```
String matricula = mult.getParameter("cpf");  
String valorToken = mult.getParameter("token"); // Tem que validar o token!!
```

```
.  
. .
```

```
Enumeration files = mult.getFileNames();
```

```
while (files.hasMoreElements()) {  
    nomeArquivo = (String)files.nextElement();  
    filesystemName = mult.getFilesystemName(nomeArquivo);  
    sufixo = filesystemName.substring(filesystemName.lastIndexOf("."));  
}
```

# MultipartRequest class

## Exemplo de uso - trechos

// Aqui é feita a operação de renomeação do arquivo gravado

```
if (SO.indexOf("Windows")>=0){
    fileOut = new File("C:/Arquivos de programas/Apache Software Foundation/Tomcat
        7.0/webapps/UDT/qualitec/" + valor + ".pdf");
    fileIn = new File("C:/Arquivos de programas/Apache Software Foundation/Tomcat
        7.0/webapps/UDT/qualitec/" + filesystemName );
}
else{
    fileOut = new File("/opt/tomcat-apps/webapps/UDT/qualitec/" + valor + ".pdf");
    fileIn = new File("/opt/tomcat-apps/webapps/UDT/qualitec/" + filesystemName);
}
if (fileOut.exists()){
    fileOut.delete();
}
fileIn.renameTo(fileOut);
```

# Gerando Arquivos pdf classe Document

- ◆ É o elemento raiz padrão para criar um arquivo no formato PDF;
- ◆ Realiza operações de alto nível, podendo definir o tamanho e a rotação da página, adicionar elementos e escrever texto em coordenadas específicas;
- ◆ Contida no pacote itextpdf.jar;
- ◆ Visitar <https://www.baeldung.com/java-pdf-creation>



# Gerando Arquivos pdf

## Exemplo de uso - trechos

```
import com.itextpdf.text.Anchor;
import com.itextpdf.text.BadElementException;
import com.itextpdf.text.BaseColor;
import com.itextpdf.text.Chapter;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.List;
import com.itextpdf.text.ListItem;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.PageSize;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.Section;
import com.itextpdf.text.pdf.PdfPCell;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
```

# Gerando Arquivos pdf

## Exemplo de uso - trechos

```
response.setContentType("application/pdf");  
response.setHeader("Content-disposition","inline; filename='Downloaded.pdf'");
```

```
Font catFont = new Font(Font.FontFamily.TIMES_ROMAN,15,Font.BOLD);  
Font catFontTable = new Font(Font.FontFamily.TIMES_ROMAN,8,Font.BOLD);
```

```
Document document = new Document(PageSize.A4.rotate(),50,10,10,20); // Cria o documento
```

```
PdfWriter.getInstance(document, response.getOutputStream());  
document.open();
```

```
document.add(new Paragraph("Relação de Coleções por UDT",catFont));  
document.add(new Paragraph(" "));  
document.add(new Paragraph(" "));
```

```
PdfPTable table = new PdfPTable(listaObjetos.size()+1);  
float[] columnWidths = new float[]{2f, 2f, 3f, 3f, 2f, 3f, 3f, 3f, 3f,2f, 4f};  
table.setWidths(columnWidths);  
table.setWidthPercentage(90);  
table.setSpacingBefore(0f);  
table.setSpacingAfter(0f);
```

# Gerando Arquivos pdf

## Exemplo de uso - trechos

```
PdfPCell c1;
```

```
c1 = new PdfPCell(new Phrase("UDT",catFontTable));  
c1.setHorizontalAlignment(Element.ALIGN_CENTER);  
table.addCell(c1);
```

```
for (int i=0; i<listaObjetos.size(); i++){  
    c1 = new PdfPCell(new Phrase(listaObjetos.get(i).getTipo(),catFontTable));  
    c1.setHorizontalAlignment(Element.ALIGN_CENTER);  
    table.addCell(c1);  
}  
document.add(table);  
  
document.close();
```

# Gerando Arquivos excel

## Classe Workbook

- ◆ Representação de alto nível de uma pasta de trabalho do Excel;
- ◆ Contida no pacote poi.jar;
- ◆ Visitar

<https://poi.apache.org/apidocs/dev/org/apache/poi/ss/usermodel/Workbook.html>

# Gerando Arquivos excel

## Exemplo de uso - trechos

```
import org.apache.poi.hssf.usermodel.HSSFSheet;  
import org.apache.poi.hssf.usermodel.HSSFWorkbook;  
import org.apache.poi.hssf.usermodel.HSSFRow;
```

```
String filename = "C:/Arquivos de programas/Apache Software Foundation/Tomcat  
7.0/webapps/UDT/qualitec/Resultado_Apuracao.xlsx";
```

```
HSSFWorkbook workbook = new HSSFWorkbook();  
HSSFSheet sheet = workbook.createSheet("Resultado_Apuracao");  
HSSFRow rowhead = sheet.createRow((short)0);  
rowhead.createCell(0).setCellValue("UDT");  
rowhead.createCell(1).setCellValue("NOME");  
  
for (int i=0; i<listaObjetos.size(); i++){  
    k = i + 1;  
    rowhead.createCell(k + 1).setCellValue(listaObjetos.get(i).getTipo());  
}
```

```
int k1=2;
```

# Gerando Arquivos excel

## Exemplo de uso - trechos

```
for (int i=0; i<listaUDTs.size(); i++){
    HSSFRow row = sheet.createRow((short)i+1);
    row.createCell(0).setCellValue(Integer.toString(listaUDTs.get(i).getId()));
    row.createCell(1).setCellValue(uDT.getNome());

    for (int j=0; j<listaObjetos.size(); j++){
        k1 = j + 1;
        row.createCell(k1 +
            1).setCellValue(Integer.toString(quantidadeReal*listaObjetos.get(j).getPontuacao() ));
    }
}

FileOutputStream fileOut = new FileOutputStream(filename);
workbook.write(fileOut);
fileOut.close();
workbook.close();
saidaJSON.put("pathProjeto", "./qualitec/Resultado_Apuracao.xlsx");
saida.print(saidaJSON);
saida.flush();
```

# Filtros

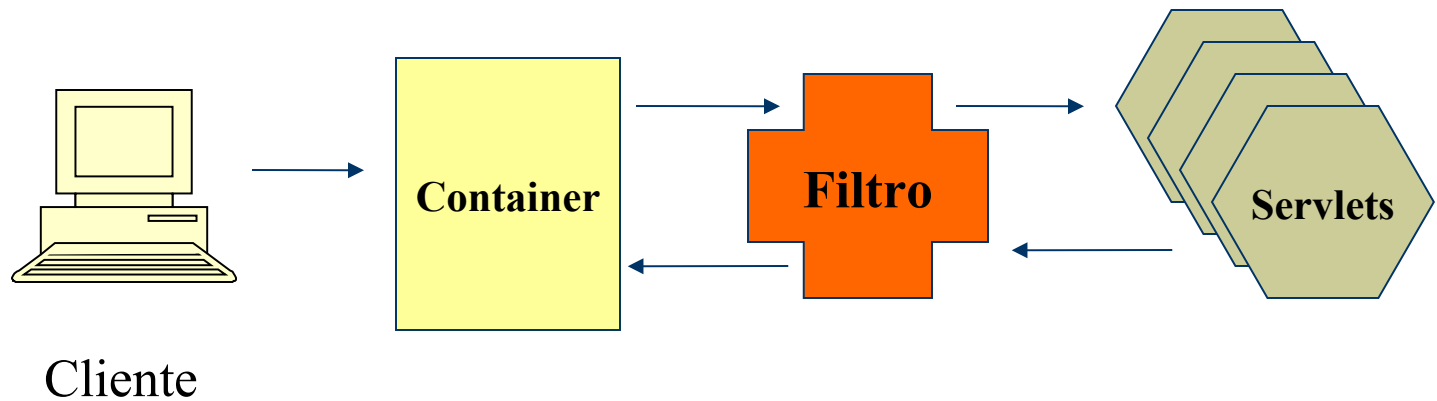
- ◆ Possibilita interceptar um *request* antes de ele chegar ao *servlet*.
- ◆ Possibilita interceptar um *response* antes de ele chegar ao cliente.
- ◆ Tem acesso aos objetos `ServletRequest` e `ServletResponse`.
- ◆ Pode ser usado como dispositivo de criptografia, compressão de dados, validação de entrada de dados etc.

# Filtros

- ♦ Invocados pelo *container* .
- ♦ Declarados no web-xml são desconhecidos pelo desenvolvedor.
- ♦ O web-xml associa o filtro ao *servlet*.
- ♦ Um filtro pode estar associado a mais de um *servlet*.
- ♦ É possível colocar um conjunto de filtros em cadeia. Um filtro apontará para o outro, refinando a filtragem.

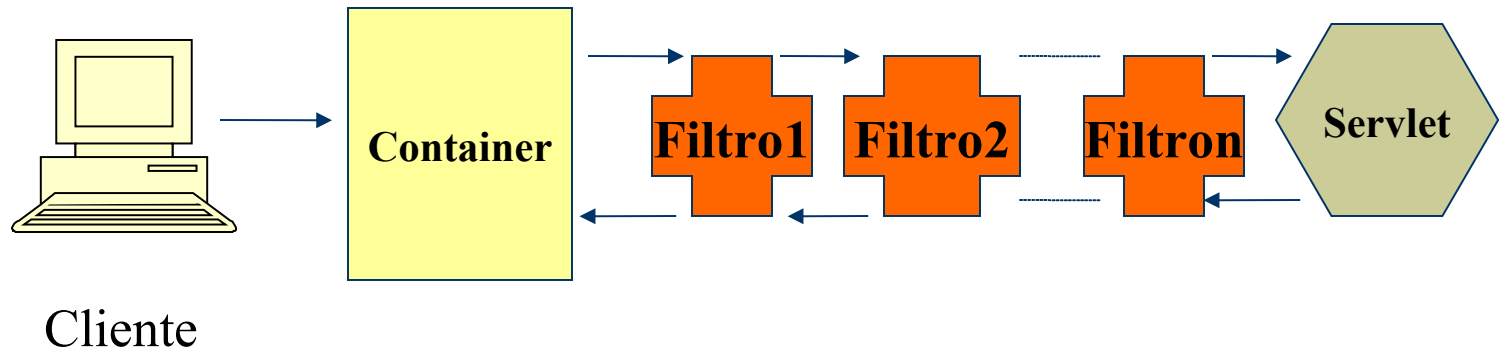


# Filtros



- Os filtros podem:
  - checar segurança(validar token, objeto HttpSession, cookies etc);
  - reformatar *requests* e corpos;
  - comprimir/descomprimir dados;
  - auditar *requests* e *responses*;
  - colocar anexos no *response*;

# Filtros - encadeamento



- Os filtros são interindependentes, um desconhece a existência do outro;

# Filtros interfaces

- ◆ São utilizadas 3 interfaces do pacote `javax.servlet`:
  - `Filter`.
  - `FilterConfig`.
  - `FilterChain`.

# A interface Filter

- ◆ Precisa ser implementada para se escrever um filtro.
- ◆ Cada invocação de um filtro “dispara” uma *thread*.
- ◆ O ciclo de vida de um filtro é representado pelos métodos:
  - `init(FilterConfig filterConfig)`.
  - `doFilter(ServletRequest request,  
                    ServletResponse response,  
                    FilterChain cadeia)`
  - `destroy()`.

# A interface Filter

- ◆ O filtro adquire vida quando o método `init()` é invocado pelo *container*. Este método é executado apenas na primeira chamada do filtro.
- ◆ No método `doFilter()` é onde as operações do filtro são executadas.
- ◆ O *container* chama este método sempre que é solicitado o *servlet* associado ao filtro.
- ◆ Os objetos *request* e *response* podem ser obtidos e modificados pelo método `doFilter()`.

# A interface FilterConfig

- ◆ Passa valores de iniciação para o filtro através dos parâmetros obtidos no web-xml.
- ◆ Possui 4 métodos:
  - `getFilterName()`
  - `getInitParameter(String parameterName)`
  - `getInitParameterName()`
  - `getServletContext()`

# Filtros – web-xml

```
<filter>
  <filter-name>GravaLog</filter-name>
  <filter-class>web.GravaLogFiltro</filter-class>
  <init-param>
    <param-name>ArquivoLog</param-name>
    <param-value>arquivolog.txt</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>GravaLog</filter-name>
  <servlet-name>MelhoresBandas</servlet-name>
</filter-mapping>
```

# Filtros – exemplo

1. Um filtro apresentando os seus ciclos de vida:
  - Nome do filtro – Ciclo Vida
  - Nome da classe do filtro – CicloVida.class
  - Nome do servlet filtrado – FilteredFilter
  - Nome da classe do servlet – FilteredFilter.class



# A distribuição descritiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- Define os filtros -->
  <filter>
    <filter-name>Ciclo Vida</filter-name>
    <filter-class>CicloVida</filter-class>
  </filter>
  <!-- Define o mapeamento dos filtros -->
  <filter-mapping>
    <filter-name>Ciclo Vida</filter-name>
    <servlet-name>FilteredServlet</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>FilteredServlet</servlet-name>
    <servlet-class>FilteredServlet</servlet-class>
  </servlet>
</web-app>
```

# O filtro CicloVida

```
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;

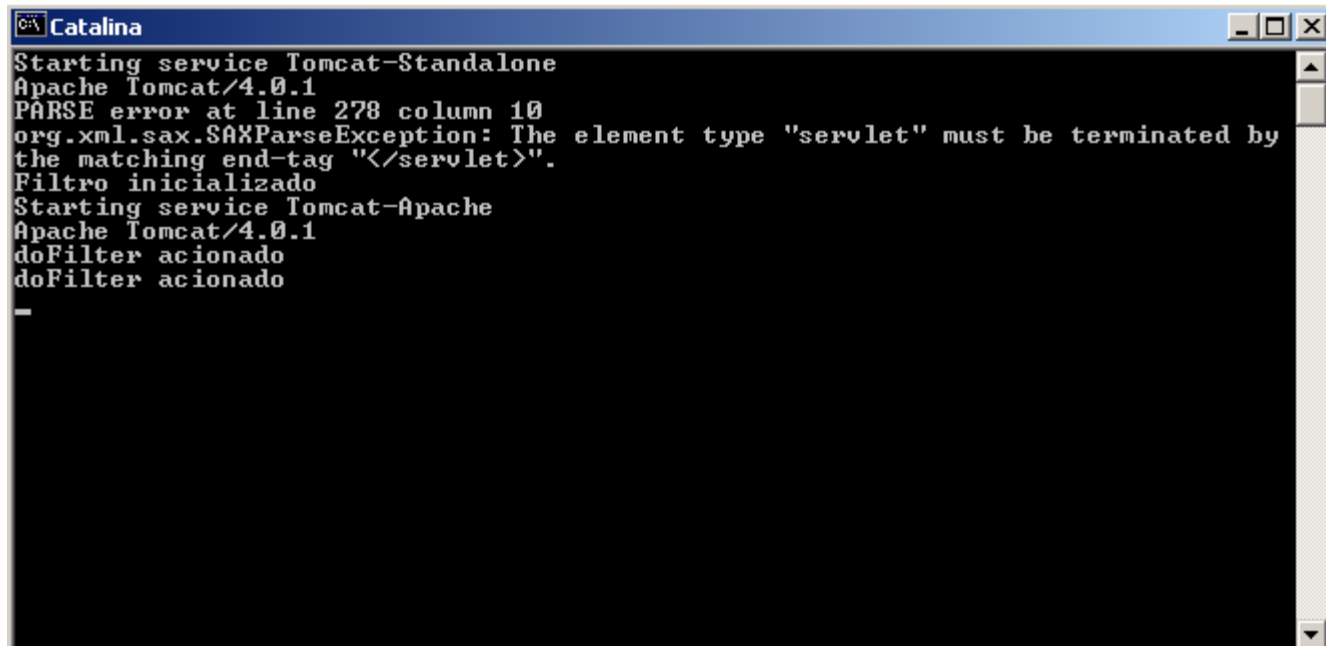
public class CicloVida implements Filter {
    private FilterConfig filterConfig;

    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("Filtro inicializado");
        this.filterConfig = filterConfig;
    }

    public void destroy() {
        System.out.println("Filtro destruído");
        this.filterConfig = null;
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        System.out.println("doFilter acionado");
        chain.doFilter(request, response);
    }
}
```

# Filtros - console TomCat



```
Catalina
Starting service Tomcat-Standalone
Apache Tomcat/4.0.1
PARSE error at line 278 column 10
org.xml.sax.SAXParseException: The element type "servlet" must be terminated by
the matching end-tag "</servlet>".
Filtro inicializado
Starting service Tomcat-Apache
Apache Tomcat/4.0.1
doFilter acionado
doFilter acionado
-
```

# A distribuição descritiva – um filtro para dois servlets

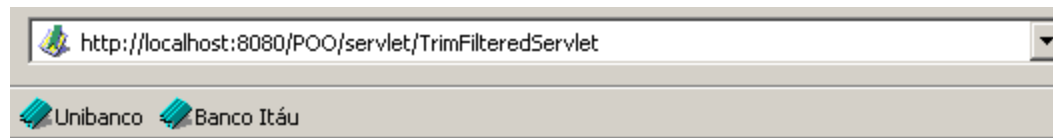
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- Define filtros -->
  <filter>
    <filter-name>Ciclo Vida </filter-name>
    <filter-class>CicloVida</filter-class>
  </filter>
  <!-- Define o mapeamento do filtro para os 2 servlets -->
  <filter-mapping>
    <filter-name>Ciclo Vida</filter-name>
    <servlet-name>FilteredServlet</servlet-name>
  </filter-mapping>
  <filter-mapping>
    <filter-name>Ciclo Vida</filter-name>
    <servlet-name>FilteredServlet2</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>FilteredServlet</servlet-name>
    <servlet-class>FilteredServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>FilteredServlet2</servlet-name>
    <servlet-class>FilteredServlet2</servlet-class>
  </servlet>
</web-app>
```

# Filtros - exemplo

2. Um filtro suprimindo espaços em branco de campos de uma entrada de dados via web:

- Nome do filtro – Trim Filtro
- Nome da classe – TrimFiltro.class
- Nome do servlet – TrimFilteredServlet
- Nome da classe – TrimFilteredServlet.class

# Filtros - exemplo

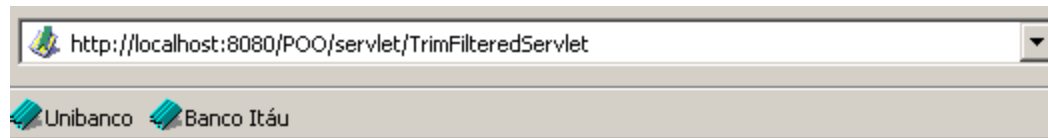


Please enter your details.

First Name:   
Last Name:   
User Name:   
Password:

Login

# Filtros - exemplo



Here are your details.

First Name: rafael

Last Name: cardozo

User Name: rafa

Password: 123456

# A distribuição descritiva

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <!-- Define filtros -->
  <filter>
    <filter-name>Trim Filtro</filter-name>
    <filter-class>TrimFiltro </filter-class>
  </filter>
  <!-- Define o mapeamento dos filtros -->
  <filter-mapping>
    <filter-name>Trim Filtro</filter-name>
    <servlet-name>TrimFilteredServlet</servlet-name>
  </filter-mapping>
  <servlet>
    <servlet-name>TrimFilteredServlet</servlet-name>
    <servlet-class>TrimFilteredServlet</servlet-class>
  </servlet>
</web-app>
```



# O filtro TrimFiltro

```
import java.io.*;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import java.util.Enumeration;

public class TrimFiltro implements Filter {
    private FilterConfig filterConfig = null;

    public void destroy() {
        System.out.println("Filtro destruído");
        this.filterConfig = null;
    }

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        System.out.println("Filtro realizado");
        Enumeration enuma = request.getParameterNames();
        while (enuma.hasMoreElements()) {
            String parameterName = (String) enuma.nextElement();
            String parameterValue = request.getParameter(parameterName);
            request.setAttribute(parameterName, parameterValue.trim());
        }

        chain.doFilter(request, response);
    }

    public void init(FilterConfig filterConfig) throws ServletException {
        System.out.println("Filtro inicializado");
        this.filterConfig = filterConfig;
    }
}
```

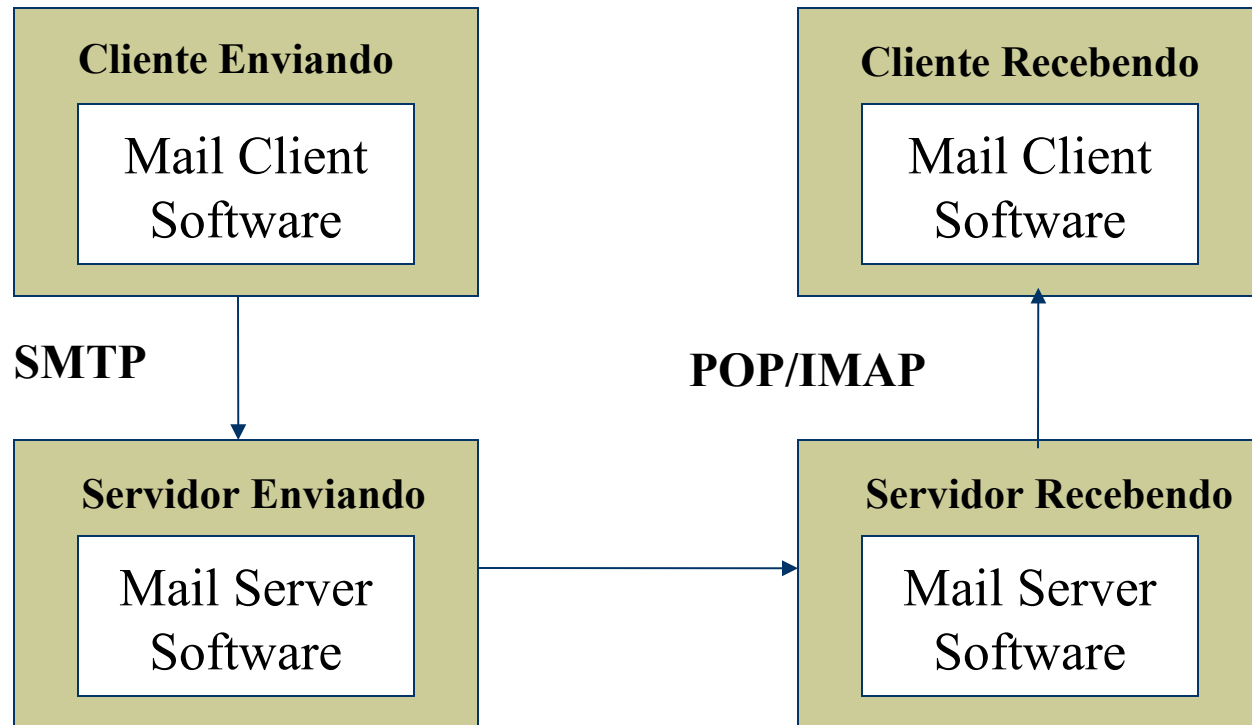
# Distribuindo a aplicação

- ◆ Utiliza-se o **Web Archive**.
- ◆ No TomCat o nome da arquivo WAR torna-se o nome da aplicação.
- ◆ Sob o diretório da aplicação emitir o comando:
  - *jar -cvf nome-da-aplicação.war \**
- ◆ Enviar o arquivo *war* gerado para o diretório *webapps* do TomCat de destino.
- ◆ O TomCat alvo descomprime o arquivo e cria o diretório da aplicação.

# Usando a API JavaMail para enviar *email*

- ♦ A API JavaMail é uma interface que facilita os desenvolvedores escreverem código Java que enviam, automaticamente, e-mails.
- ♦ Contida no pacote javax.mail.jar.

# Enviando *email* - funcionamento



# Usando a API JavaMail para enviar *email* - protocolos

- ♦ SMTP (Simple Mail Transfer Protocol)
  - Utilizado para enviar uma mensagem de um servidor de *email* para outro.
- ♦ POP (Post Office Protocol)
  - Utilizado para recuperar mensagens de um servidor de *email*. Transfere todas as mensagens de um servidor de *email* para o cliente. Está em sua versão 3 e é chamado de POP3.
- ♦ IMAP (Internet Message Access Protocol)
  - Utilizado por serviços de *email* baseados na *web* tais como Yahoo e Hotmail. Permite ao *web browser* ler mensagens armazenadas em um diretório de um servidor de *email*. Está em sua versão 4 e é chamado de IMAP4.
- ♦ MIME (Multipurpose Internet Message Extension)
  - Especifica o tipo de conteúdo que pode ser enviado como mensagem ou como arquivo anexado.

# Usando a API JavaMail para enviar *email*

- ◆ Para utilizarmos as APIs são necessários os seguintes arquivos:
  - javax.mail.jar
    - Contém as classes Java da API JavaMail.
  - activation.jar
    - Contém as classes Java da API JavaBean Activation Framework.
- ◆ Devem residir no diretório jre/lib/ext do Java JDK e no TomCat, diretório lib.

# Os 4 passos para enviar um *email*

- ◆ Criar um sessão de *email*.
- ◆ Criar uma mensagem.
- ◆ Endereçar a mensagem (identificar remetente e destinatário).
- ◆ Enviar a mensagem.

# Criando uma sessão de *email*

- ◆ Atividades necessárias:
  - Identificar o *host* onde reside o servidor de *email*. Usar “localhost” se estiver na mesma máquina da aplicação.
  - Criar um objeto Properties que conterà as propriedades necessárias para enviar um *email*.
  - Especificar uma propriedade para a sessão utilizando-se o método *put* de Properties.
  - Criar um objeto *Session*, que define a sessão de *mail*, chamando o método `getDefaultInstance`.



# Criando uma sessão de *email*

- ♦ A classe Session integra o pacote javax.mail.
- ♦ A classe Properties integra o pacote javax.util.

# Criando uma mensagem

## ◆ Atividades necessárias:

- Utilizar a classe `MimeMessage` para criar uma mensagem. Fica armazenada no pacote `javax.mail.internet`.
- Criar um objeto `MimeMessage` fornecendo uma objeto `Session`.
- Utilizar um `Authenticator` para validar o usuário e a senha.
- Utilizar os métodos `setSubject` e `setText` para especificar o propósito do *email* e seu texto. O texto enviado assume o tipo `text/plain` para o MIME.
- Utilizar o método `setContent` se desejar anexar um documento.

# Endereçando a mensagem

## ◆ Atividades necessárias:

- Definir um endereço de *email* através da classe `InternetAddress`, integrante do pacote `javax.mail.internet`.
- Utilizar o método `setFrom` do objeto `MimeMessage` para atribuir o endereço “From”.
- Utilizar o método `setRecipient` para atribuir o endereço “To”.

# Enviando uma mensagem

- ◆ Atividades necessárias:
  - Utilizar o método *send* da classe Transport para enviar a mensagem para o *email server*.

# Enviando um email

## Código Exemplo

```
package model;
```

```
import java.util.Properties;  
import javax.mail.Message;  
import javax.mail.MessagingException;  
import javax.mail.PasswordAuthentication;  
import javax.mail.Session;  
import javax.mail.Transport;  
import javax.mail.internet.InternetAddress;  
import javax.mail.internet.MimeMessage;
```

```
public class EnviarEmail {
```

```
    public static void enviarEmail(String to, String subject){
```

```
        String from = "email_do_remetente";  
        Properties properties = System.getProperties();  
        properties.put("mail.transport.protocol", "smtp");  
        properties.put("mail.smtp.port", "587");  
        properties.put("mail.smtp.starttls.enable", "true");  
        properties.put("mail.smtp.auth", "true");  
        properties.put("mail.smtp.ssl.trust", "smtp.zoho.com");  
        Session session = Session.getDefaultInstance(properties);
```

# Enviando um email

## Código Exemplo

```
try {  
  
    MimeMessage message = new MimeMessage(session);  
    message.setFrom(new InternetAddress(from));  
    message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));  
    message.setSubject(subject);  
  
    String textoPrimeiraParte = "" +  
        "<!DOCTYPE html>" +  
        "<html>" + // Aqui coloca-se o HTML do email  
        "<head" + "</head>" + "</html>";  
    message.setContent(textoPrimeiraParte,"text/html");  
    Transport transport = session.getTransport();  
    transport.connect("smtp.zoho.com","email_do_remetente", "senha");  
    transport.sendMessage(message, message.getAllRecipients());  
    transport.close();  
  
}  
catch (Exception ex) {  
    ex.printStackTrace();  
}  
}
```

# Enviando um email anexando um arquivo - Código Exemplo

```
BodyPart messageBodyPart = new MimeBodyPart();  
messageBodyPart.setText("Corpo da mensagem");
```

```
Multipart multipart = new MimeMultipart();
```

```
multipart.addBodyPart(messageBodyPart);
```

```
messageBodyPart = new MimeBodyPart();  
String filename = "/UDT/projetos/UDT999.pdf";  
DataSource source = new FileDataSource(filename);  
messageBodyPart.setDataHandler(new DataHandler(source));  
messageBodyPart.setFileName(filename);  
multipart.addBodyPart(messageBodyPart);
```

```
message.setContent(multipart);
```

```
Transport.send(message);
```

# Criptografando com MD5

## Código Exemplo

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class ControllerGeraHashMD5 {

    public static String getHashMd5(String value) {
        MessageDigest md;
        try {
            md = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
        BigInteger hash = new BigInteger(1, md.digest(value.getBytes()));
        return hash.toString(16);
    }

    public static void main(String[] args){
        System.out.println(getHashMd5("12345678925"));
        System.out.println(getHashMd5("12345678925")); // Duplicado para mostrar que é determinístico
    }
}
```





FIM

**Até breve!**