



JavaNororoeste

Grupo de Usuários Java do Nororoeste Paulista

CARLOS FERNANDO GONÇALVES

MINI-CURSO

Introdução a Aplicações Web com Java





Objetivos

- Conteúdo dinâmico na internet;
- Servlet;
- Java Server Pages;
- Aplicações Web - Padrão MVC;
- Um exemplo de projeto em MVC



Conteúdo dinâmico na internet



Quando executamos o navegador Web e solicitamos uma URL, uma solicitação de um documento HTML é realizada. O servidor Web encontra o arquivo correspondente e devolve. Se o documento HTML incluir qualquer imagem, o navegador por sua vez irá submeter solicitações os documentos de imagens também. Como descrito aqui, todas essas solicitações são para arquivos estáticos. Ou seja, os documentos que são solicitados nunca mudam dependendo de quem solicitou, quando estes foram solicitados, ou que(caso haja) parâmetros adicionais foram incluídos com a solicitação.



Conteúdo dinâmico na internet



Conteúdo dinâmico na internet



Conteúdo dinâmico na internet



Para uma solicitação da Web mais simples, um navegador solicita um documento HTML e o servidor Web encontra o arquivo correspondente e devolve. Se o documento HTML incluir qualquer imagem, o navegador por sua vez irá submeter solicitações os documentos de imagens também. Como descrito aqui, todas essas solicitações são para arquivos estáticos. Ou seja, os documentos que são solicitados nunca mudam dependendo de quem solicitou, quando eles foram solicitados, ou que(caso haja) parâmetros adicionais foram incluídos com a solicitação.



Conteúdo dinâmico na internet



No entanto, a maioria dos dados fornecidos através da Web hoje tem uma natureza dinâmica. Como exemplo, extrato bancário, mensagens, cotações de ações, geração de contas de luz e água, boletos bancários, etc.

O conteúdo da Web dinâmico, então, exige que o servidor da Web faça algum processamento adicional para a solicitação correspondente, a fim de gerar uma resposta personalizada.

Os servidores de HTTP mais antigos não incluíam qualquer mecanismo embutido para gerar respostas dinamicamente. Ao invés disso, foram fornecidas interfaces para chamar outros programas para utilizar solicitações no conteúdo runtime.



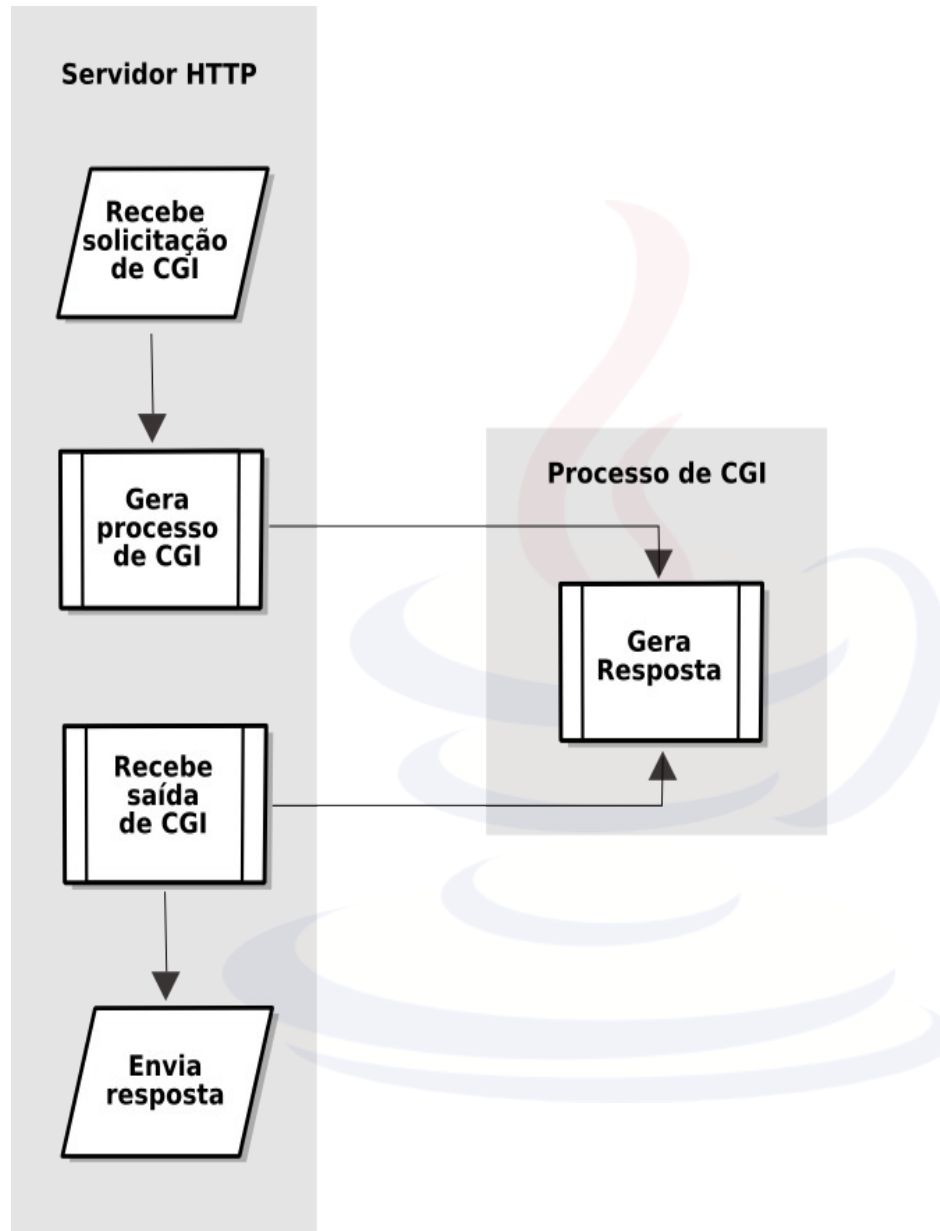
Conteúdo dinâmico na internet



O primeiro padrão para conteúdo da Web dinâmico se baseava na Common Gateway Interface, ou CGI, que especificava um mecanismo para servidores da Web passarem as informações da solicitação para programas externos, que eram então, executados pelo servidor Web para gerar resposta em tempo de execução. A linguagem mais popular para escrever programas de CGI é Perl, mas os códigos de CGI podem ser escritos em qualquer linguagem que possa ser chamada como um programa independente pelo servidor HTTP.



Conteúdo dinâmico na internet



Processo de servidor para rodar programas de CGI.



Conteúdo dinâmico na internet



Outras tecnologias como:

- ColdFusion;
- Active Server Pages;
- Server-Side JavaScript;
- PHP.

Estas tecnologias também são capazes de gerar conteúdo dinâmico para internet.



Servlet



Devido à importância da geração de conteúdo dinâmico para desenvolvimento da Web, foi natural que a SUN Microsystems, propusesse extensões para tecnologia Java em seu domínio. Da mesma forma que a SUN introduziu applets como pequenas aplicações baseadas em Java para adicionar funcionalidade interativa aos navegadores da Web, em 1996, a SUN introduziu **servlets** como pequenas aplicações baseadas em Java para adicionar funcionalidade dinâmica a servidores da Web.

Diferentemente dos programas tradicionais de CGI que necessitam da criação de um novo processo para tratar nova solicitação, todos os **servlets** associados com um servidor da Web rodam dentro de um único processo. Este processo roda uma Java



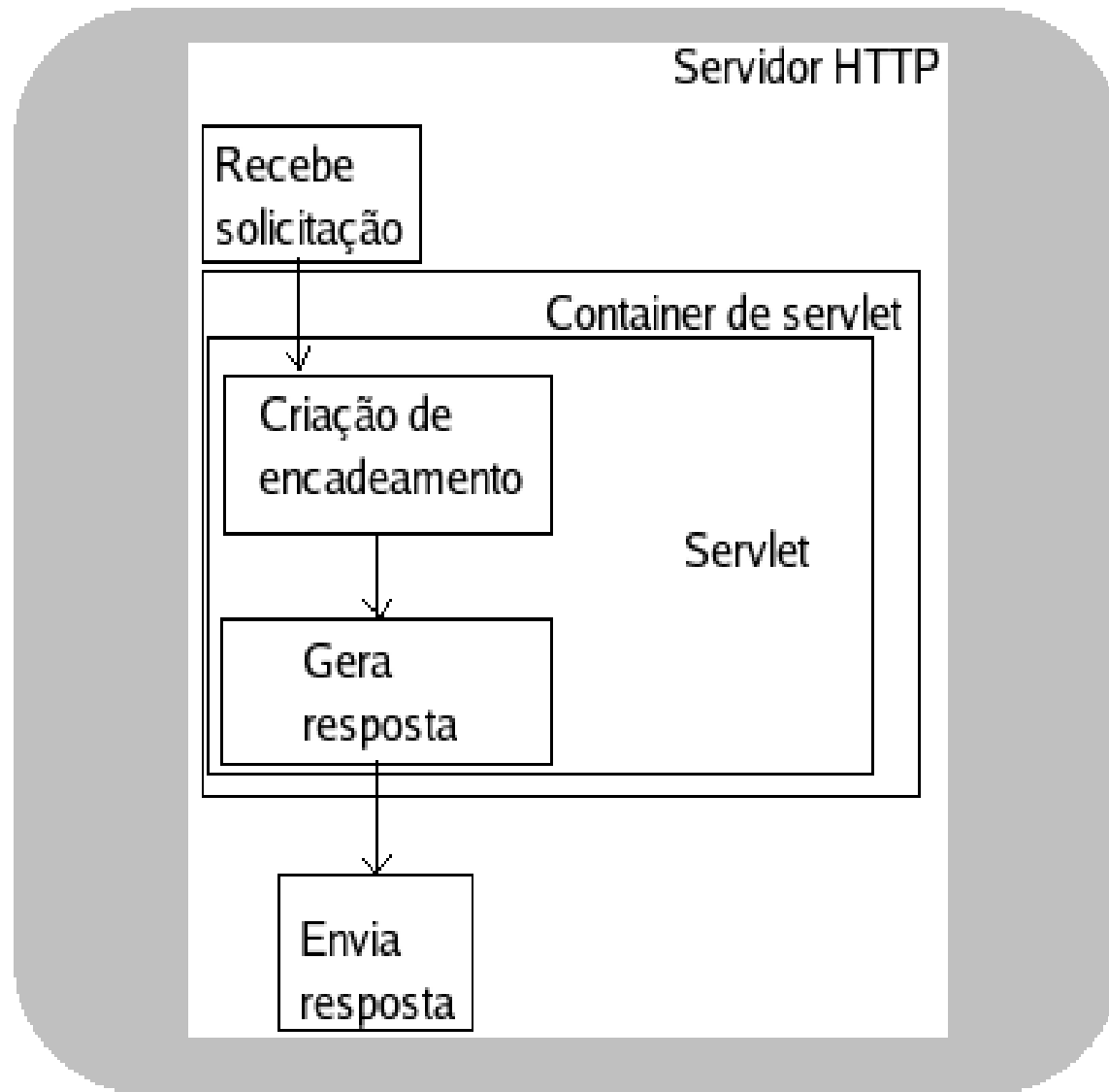
Servlet



Virtual Machine(JVM), que é o programa específico de plataforma para rodar programas de Java compilados (compatíveis com várias plataformas).



Servlet



Servlet



Ao invés de criar um processo para cada solicitação, a JVM, cria um encadeamento para tratar de cada solicitação de servlet. Os encadeamentos de Java tem muito menos overhead do que os processos completos dentro da memória do processador já alocada pela JVM, tornando a execução do servlet consideravelmente mais eficiente do que o processamento de CGI. Já que a JVM persiste além de uma única solicitação, os servlets também podem evitar muitas operações demoradas, como conexão a um banco de dados, ao compartilhá-los entre todas as solicitações. Ao mesmo tempo, pelo fato dos servlets serem escritos em Java, eles se aproveitam de todos os benefícios da plataforma de Java básica; um modelo de programação orientado a objetos, gerenciamento automático de memória, portabilidade compatível



Servlet



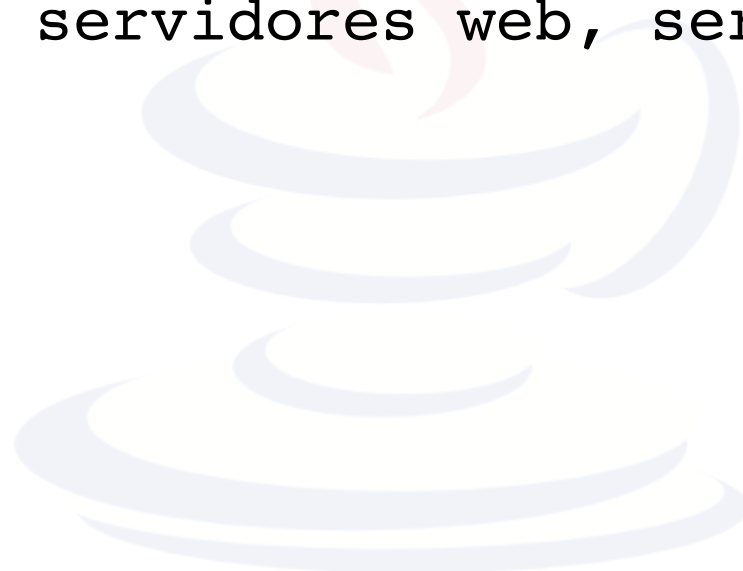
com várias plataformas e acessam a rica coleção de APIs de Java agora disponível para acessar banco de dados, servidores de diretório, recursos de rede e assim por diante.



Servlet



Servlets são classes Java escritas especialmente para rodar dentro de um servidor internet (assim como Applets são classes escritas para rodar dentro de navegadores web). Como tal, não são aplicações autônomas, mas sim extensões de aplicações para servidores web, servidores SMTP (e-mail) e outros.



Servlet



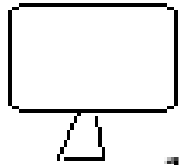
A API de servlets é representada pelos pacotes `javax.servlet` e `javax.servlet.http`, cujas principais classes/interfaces são mostradas na figura a seguir.



Servlet



:Navegador

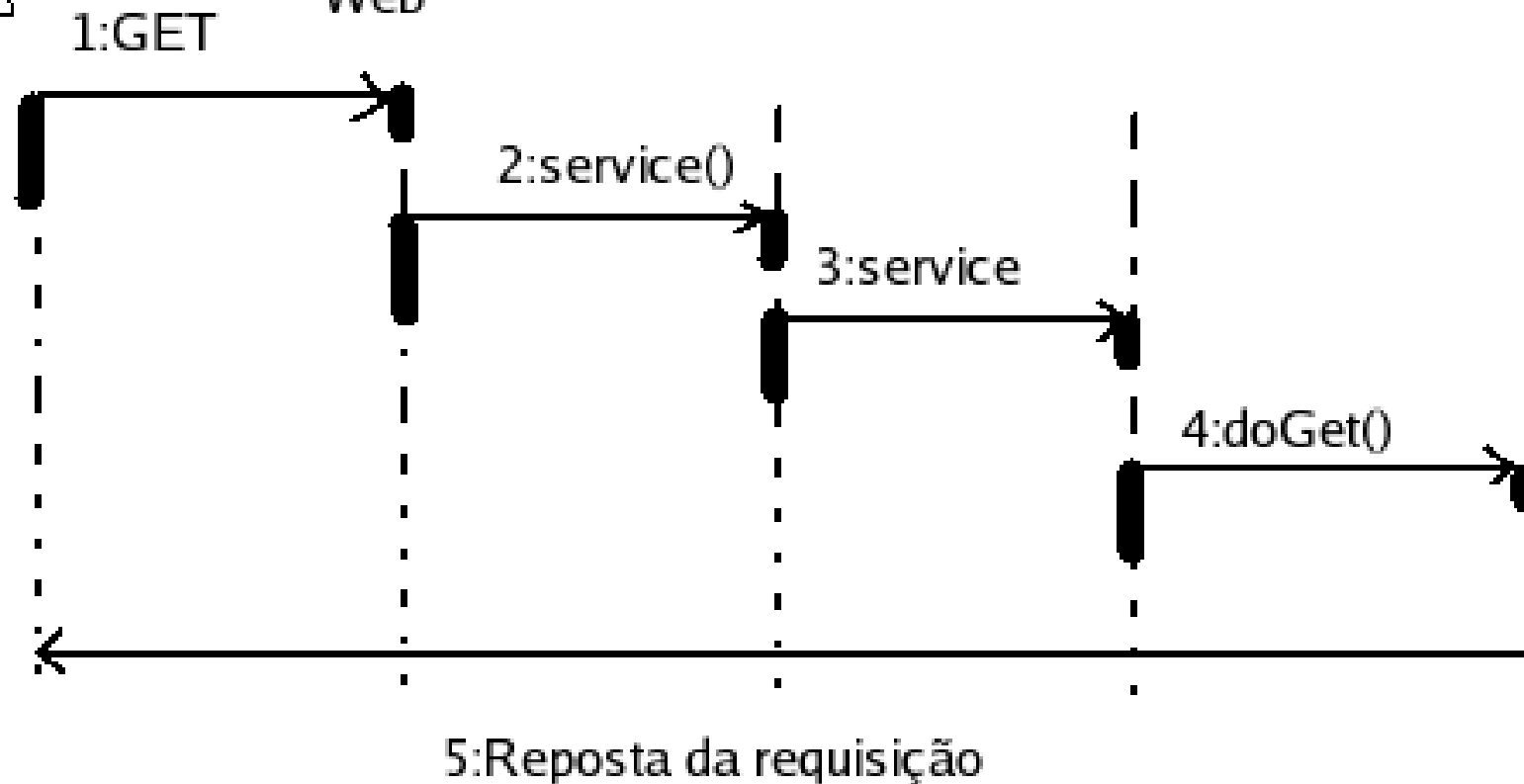


:Container
Web

:Servlet

:HttpServlet

:ServidorHTTP





A interface `javax.servlet.Servlet` é o coração da API. Possui três métodos relacionados com o ciclo de vida de um servlet: `init()`, `service()` e `destroy()`.

O ciclo de vida de um servlet é simples, quando comparado com outras tecnologias Java(ex: EJB):

1. Antes da primeira requisição, o container cria uma instância do servlet e chama seu método `init()`. De forma geral esse método é usado para a alocação de recursos, como conexões com um banco de dados ou referências a EJBs.
2. Quando uma requisição é recebida, o container cria novos objetos `ServletRequest` e `ServletResponse` e os passa para o método `service()` do servlet. Aqui vale notar que comumente múltiplas requisições são processadas de forma simultânea, em threads diferentes, mas por





instâncias compartilhadas do servlet.

3. Quando o servlet não for mais necessário (por exemplo, quando a aplicação Web for removida do container), é chamado o método `destroy()`. Neste devem ser liberados os recursos alocados em `init()`.

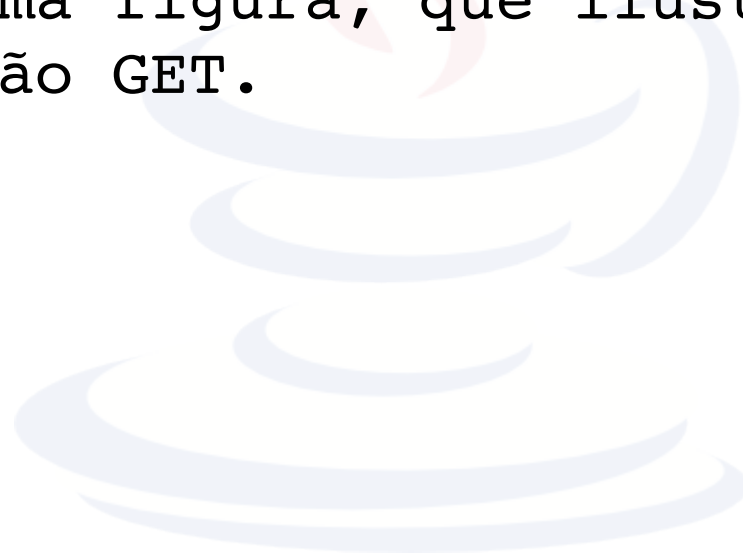
Observe que a interface `Servlet` não define comportamento especificamente associado ao protocolo HTTP. Isso possibilita que um servlet seja utilizado por qualquer protocolo. Porém, como na prática a maioria dos servlets é usada apenas por aplicações web, a API fornece a classe abstrata especializada `javax.servlet.http.HttpServlet`.



Servlet



`HttpServlet` estende `javax.servlet.GenericServlet` que implementa a maior parte dos métodos da interface `Servlet`, e também o método `service()`, que invoca `doGet()`, `doPost()`, `doDelete()` etc. De acordo com o tipo de requisição HTTP recebida. Observe a próxima figura, que ilustra o tratamento de uma requisição GET.



Servlet



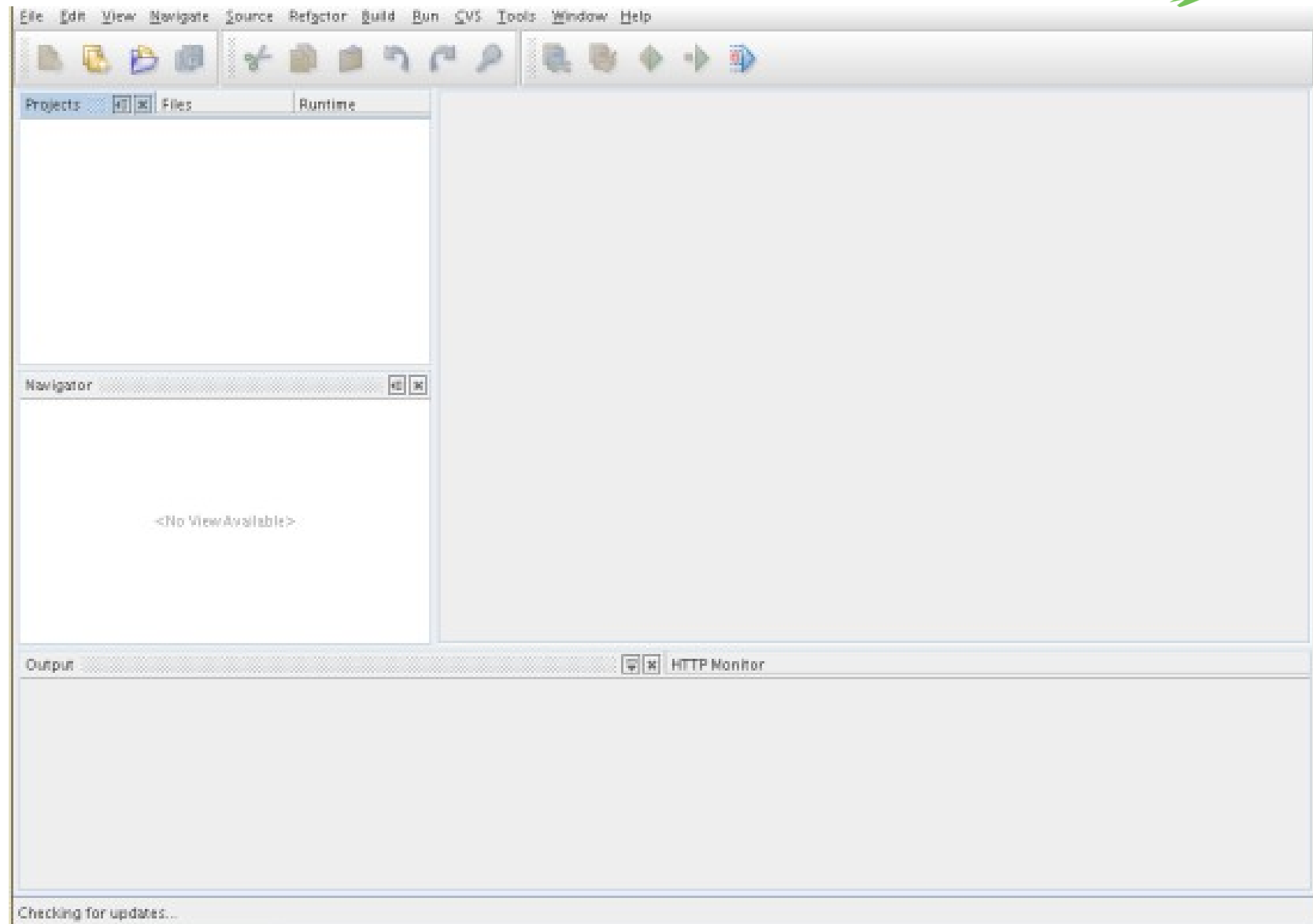
Desta forma, a maneira mais fácil de desenvolver um servlet em aplicações Web, é estender `HttpServlet` e implementar os métodos necessários.



Servlet



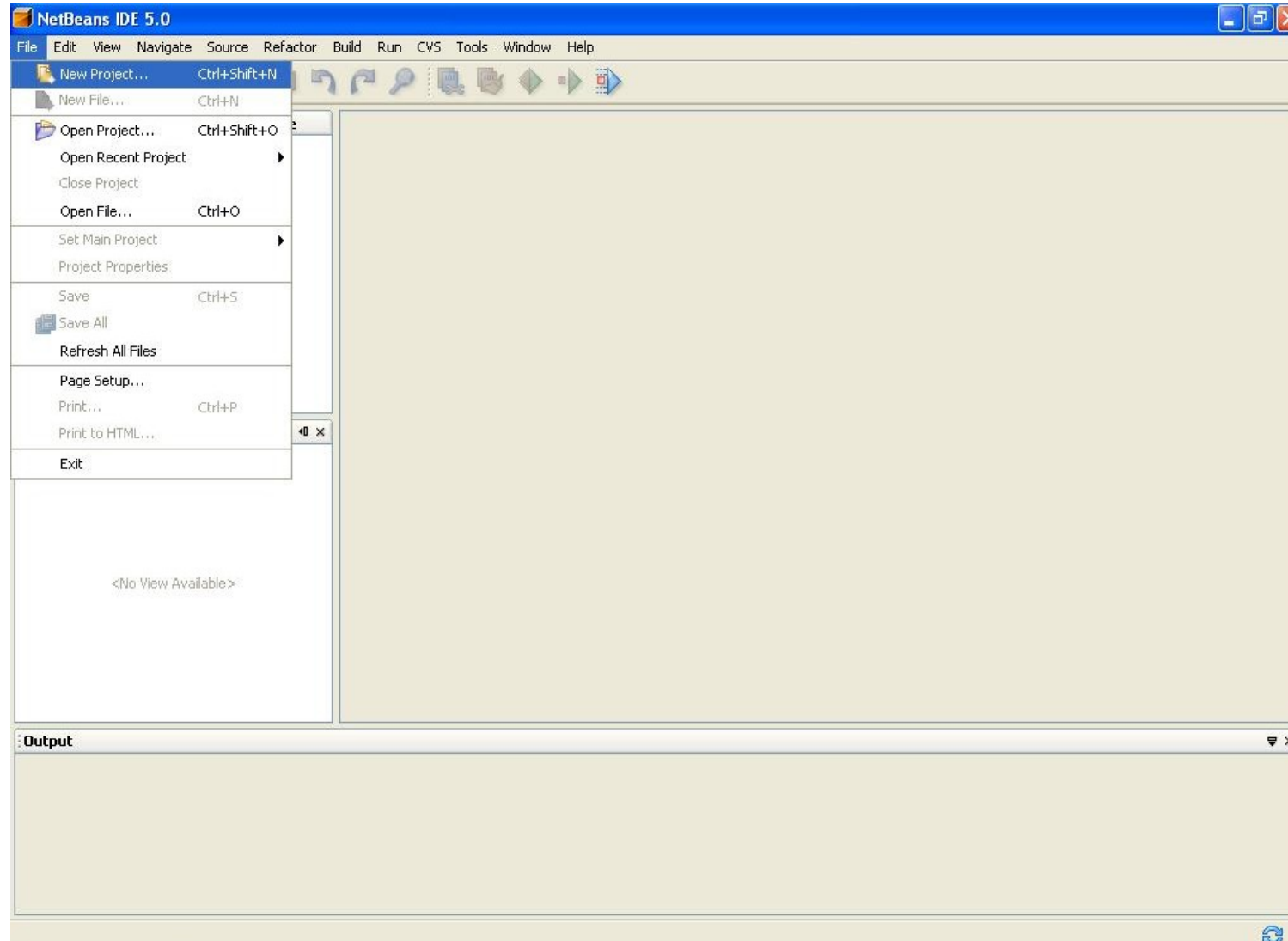
Exemplo:



Servlet



Exemplo:



Servlet



Exemplo:

New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Frameworks

Name and Location

Project Name:

Project Location:

Project Folder:

Source Structure:

Add to Enterprise Application:

Server:

J2EE Version:

Context Path:

Recommendation: Source Level 1.4 should be used in J2EE 1.4 projects.

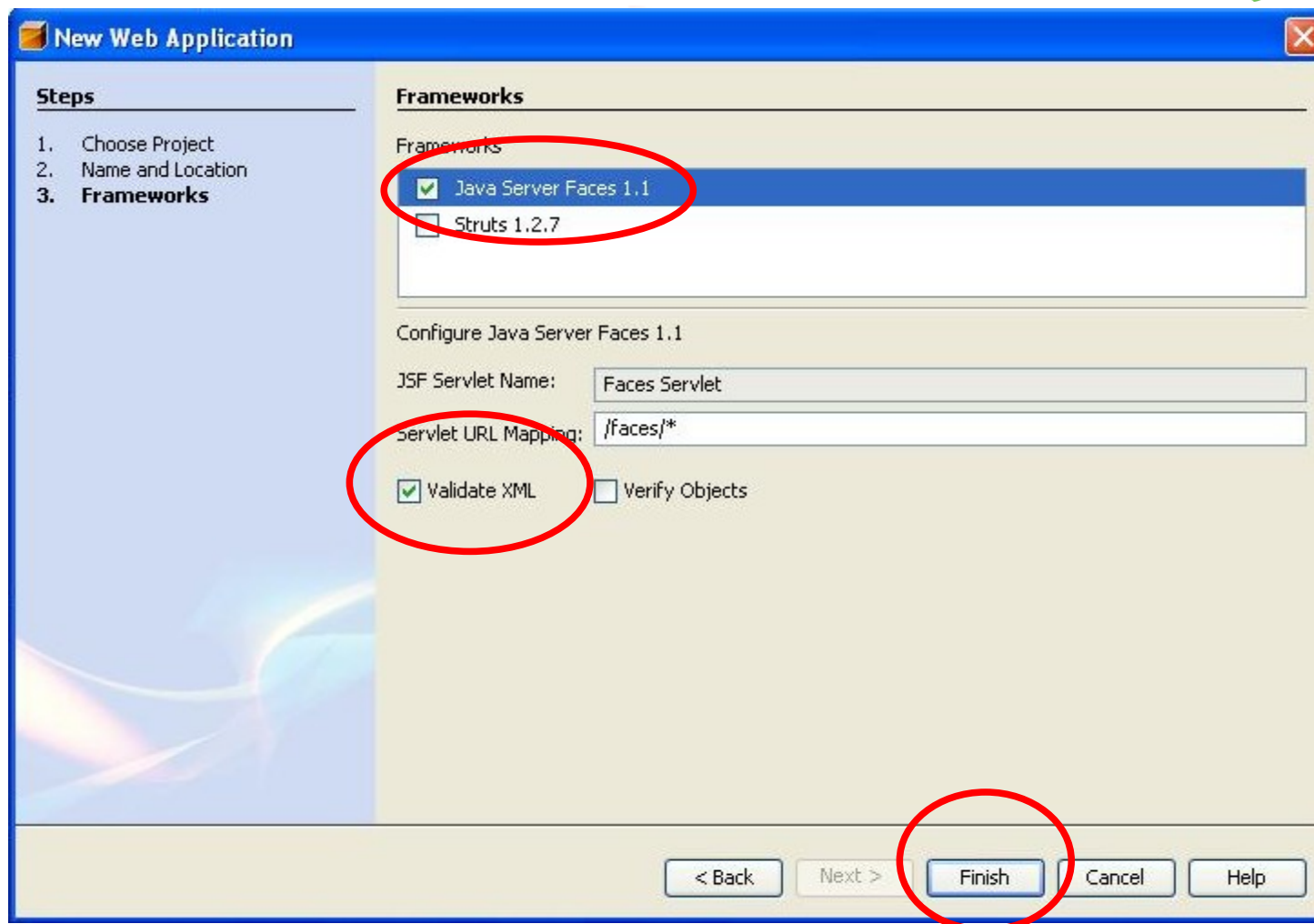
☒ Set Source Level to 1.4

☒ Set as Main Project

Servlet



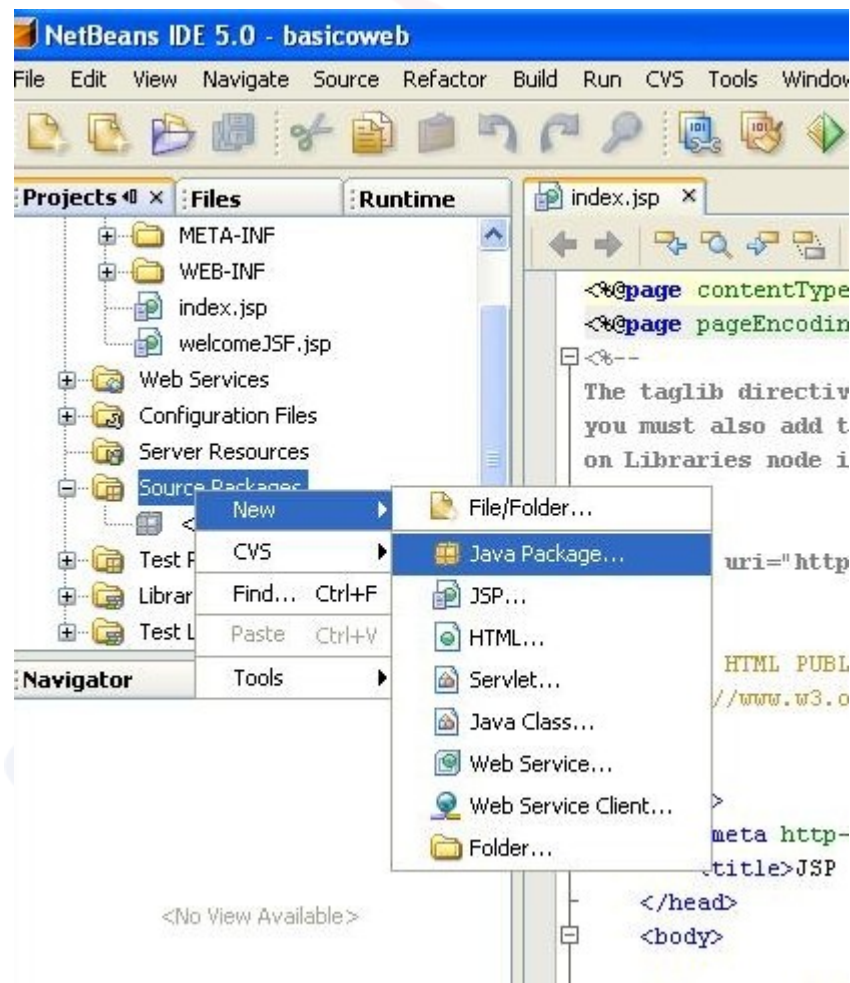
Exemplo:



Servlet



Exemplo:



Servlet



Exemplo:

New Java Package

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Package Name:

Project:

Location:

Created Folder:

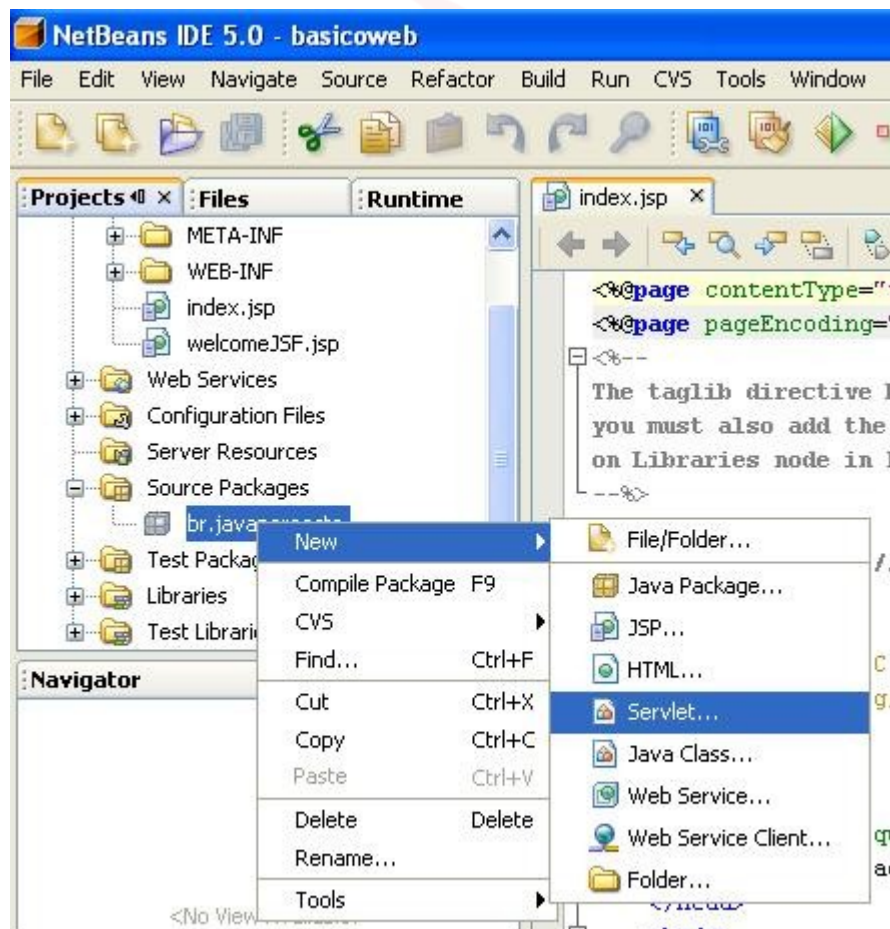
< Back Next > **Finish** Cancel Help



Servlet



Exemplo:



Servlet



Exemplo:

New Servlet

Steps

1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

< Back **Next >** Finish Cancel Help



Servlet



Exemplo:

```
NetBeans IDE 5.0 - basicoweb
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help

index.jsp x AloMundo.java x

import javax.servlet.http.*;

/**
 *
 * @author carlos
 * @version
 */
public class AloMundo extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        /* TODO output your page here
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet AloMundo</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet AloMundo at " + request.getContextPath () + "</h1>");
        out.println("</body>");
        out.println("</html>");
        */
        out.close();
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```



Servlet



Exemplo:

```
NetBeans IDE 5.0 - basicoweb
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help

index.jsp x AloMundo.java * x

out.println("</body>");
out.println("</html>");

out.close();
}

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

/** Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}

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

30:11 INS



Servlet



Exemplo:

```
</context-param>
<context-param>
  <param-name>javax.faces.CONFIG_FILES</param-name>
  <param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>
<context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>client</param-value>
</context-param>
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
  <servlet-name>AloMundo</servlet-name>
  <servlet-class>br.javanoroeste.AloMundo</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>AloMundo</servlet-name>
  <url-pattern>/AloMundo</url-pattern>
</servlet-mapping>
<session-config><session-timeout>
  30
</session-timeout></session-config><welcome-file-list><welcome-file>
  index.jsp
</welcome-file></welcome-file-list></web-app>
```



Conteúdo dinâmico na internet



Exemplo:

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

```
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
```

Alterar para ->

```
//      TODO output your page here
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet AloMundo</title>");
out.println("</head>");
out.println("<body>");
out.println("<h1>Servlet AloMundo at " + request.getContextPath () + "</h1>");
out.println("</body>");
out.println("</html>");
```

```
    out.close();
```

```
}
```



Conteúdo dinâmico na internet



Exemplo:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    String nomePessoa = request.getParameter("nomePessoa");
    // TODO output your page here
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet AloMundo</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Alô " + nomePessoa + "</h1>");
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```



Conteúdo dinâmico na internet



Exemplo:



Bem vindo null



Conteúdo dinâmico na internet



Exemplo:

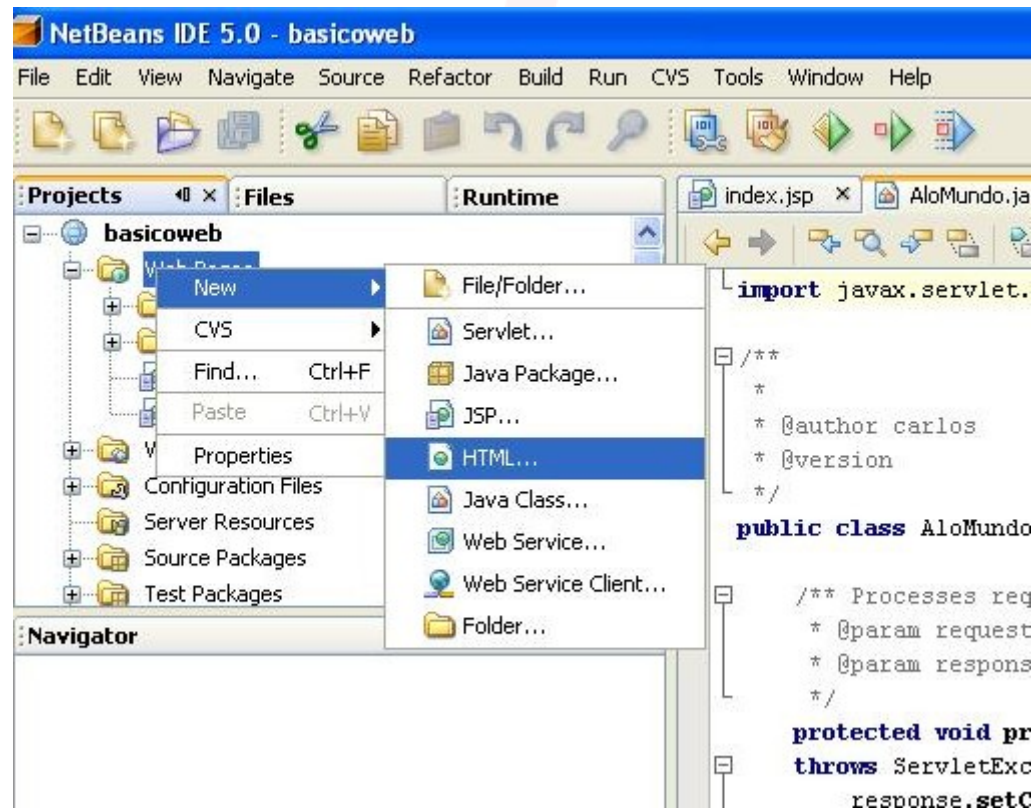


Bem vindo Carlos



Conteúdo dinâmico na internet

Criando a página formulario.html:



Conteúdo dinâmico na internet

Criando a página formulario.html:



New HTML File

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

HTML File Name:

Project:

Location: ▼

Folder:

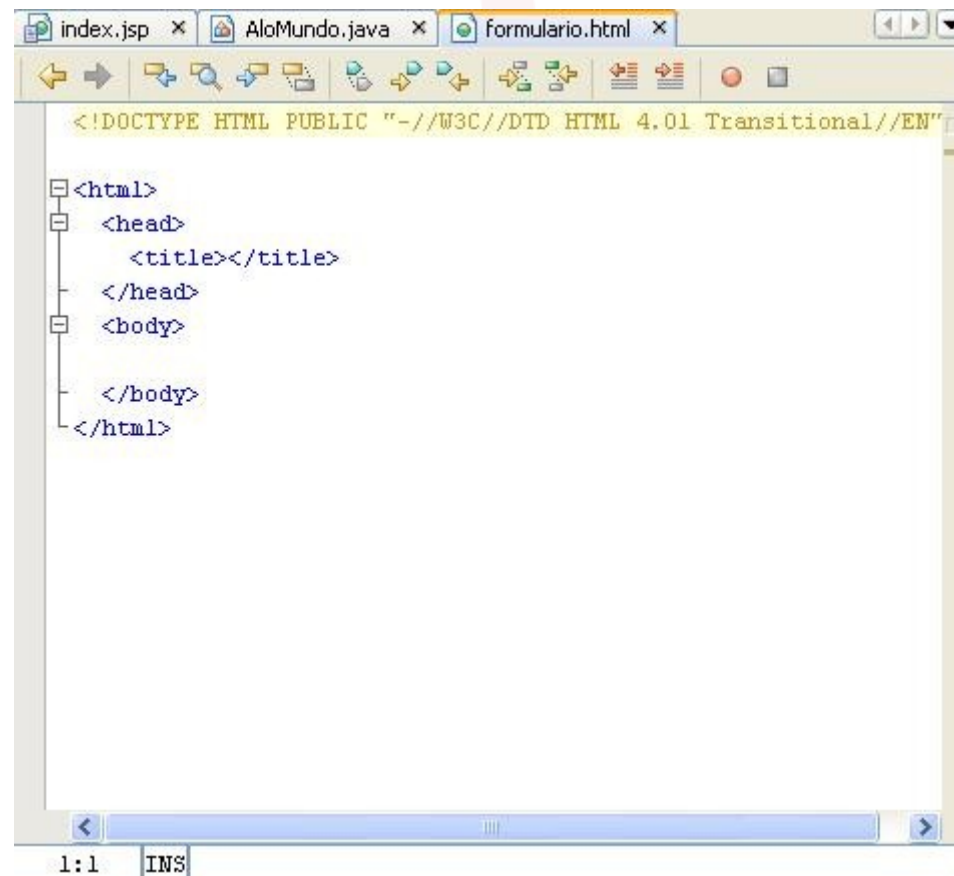
Created File:

< Back Next > **Finish** Cancel Help



Conteúdo dinâmico na internet

Criando a página formulario.html:

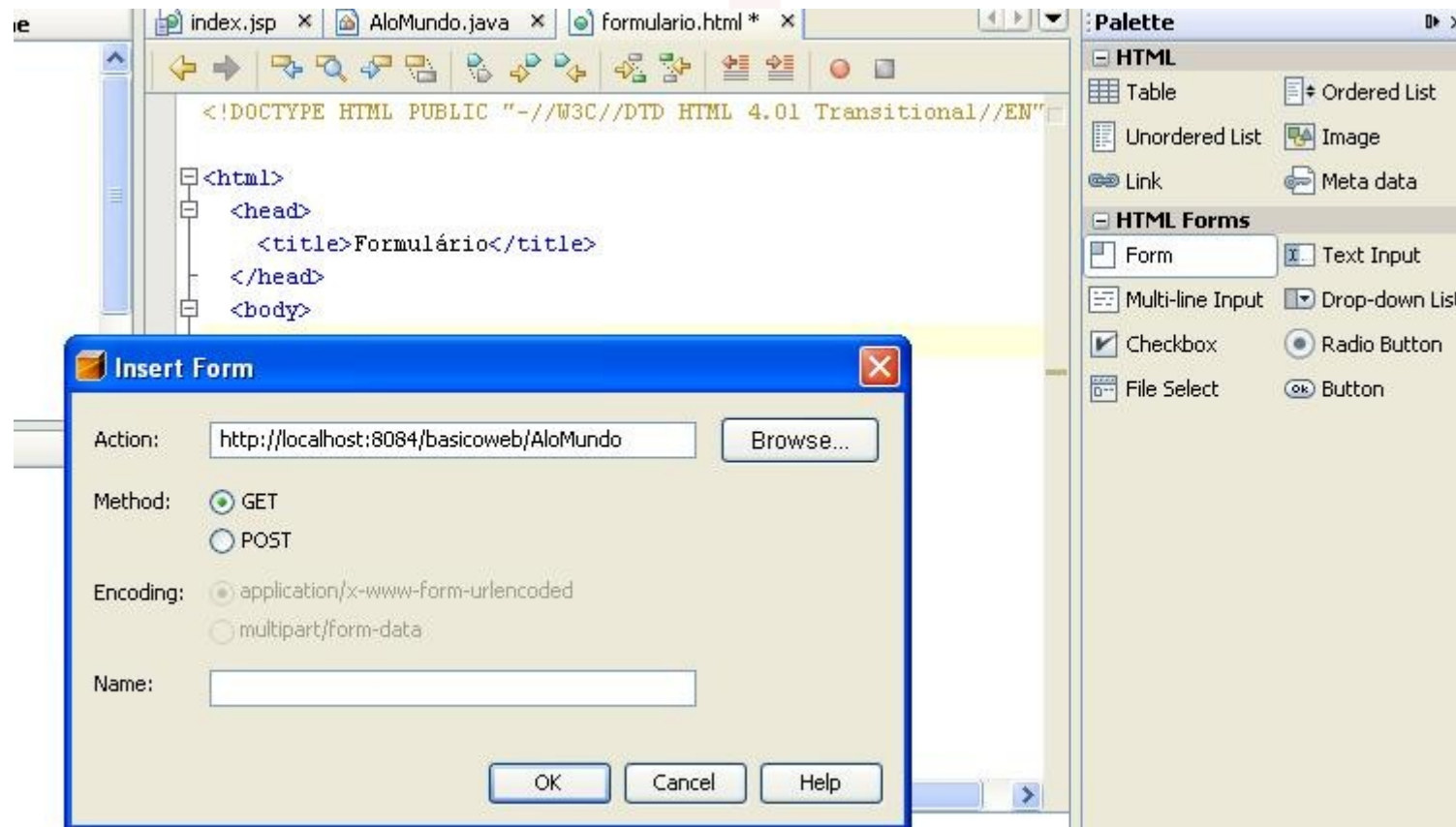


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
<html>
<head>
<title></title>
</head>
<body>
</body>
</html>
```



Conteúdo dinâmico na internet

Criando a página formulario.html:



Conteúdo dinâmico na internet



Criando a página formulario.html:

The screenshot shows an IDE with three tabs: `index.jsp`, `AloMundo.java`, and `Formulario.html *`. The `Formulario.html` file is open, displaying the following HTML code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Formulário</title>
</head>
<body>
  <form action="http://localhost:8084/basicoweb/AloMundo">
  </form>
</body>
</html>
```

The `<form>` tag is highlighted with a red rectangle. On the right side, the **Palette** window is open, showing the **HTML Forms** section. The **Form** element is highlighted with a red rectangle, indicating it was used to create the form in the code.



Conteúdo dinâmico na internet



Criando a página formulario.html:

The screenshot shows an IDE with three tabs: index.jsp, AloMundo.java, and formulario.html *. The formulario.html file is open, displaying the following HTML code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
<html>
  <head>
    <title>Formulário</title>
  </head>
  <body>
    <form action="http://localhost:8084/basicweb/AloMundo">
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

The code is displayed in a tree view on the left. The right side of the IDE shows the 'Palette' window with the 'HTML Forms' section expanded. The 'Text Input' option is highlighted with a red box. Other options in the 'HTML Forms' section include Form, Multi-line Input, Drop-down List, Checkbox, Radio Button, File Select, and Button. The status bar at the bottom shows the time 12:2 and the cursor position INS.



Conteúdo dinâmico na internet



Criando a página formulario.html:

formulario.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 T
```

```
<html>
<head>
  <title></title>
</head>
<body>
  <form action="http://localhost:8084/basicowe"
    <input type="text" name="nomePessoa" value
    <input type="submit" value="Enviar" />
  </form>
</body>
</html>
```

Palette

- Ordered List
- Unordered List
- Image
- Link
- Meta data
- HTML Forms**
 - Form
 - Text Input
 - Multi-line Input
 - Drop-down List
 - Checkbox
 - Radio Button
 - File Select
 - Ok Button**



Conteúdo dinâmico na internet



Exemplo:

```
NetBeans IDE 5.0 - basicoweb
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help

AloMundo.java x AloServletRepasse.java x formulario.html x view.jsp x

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

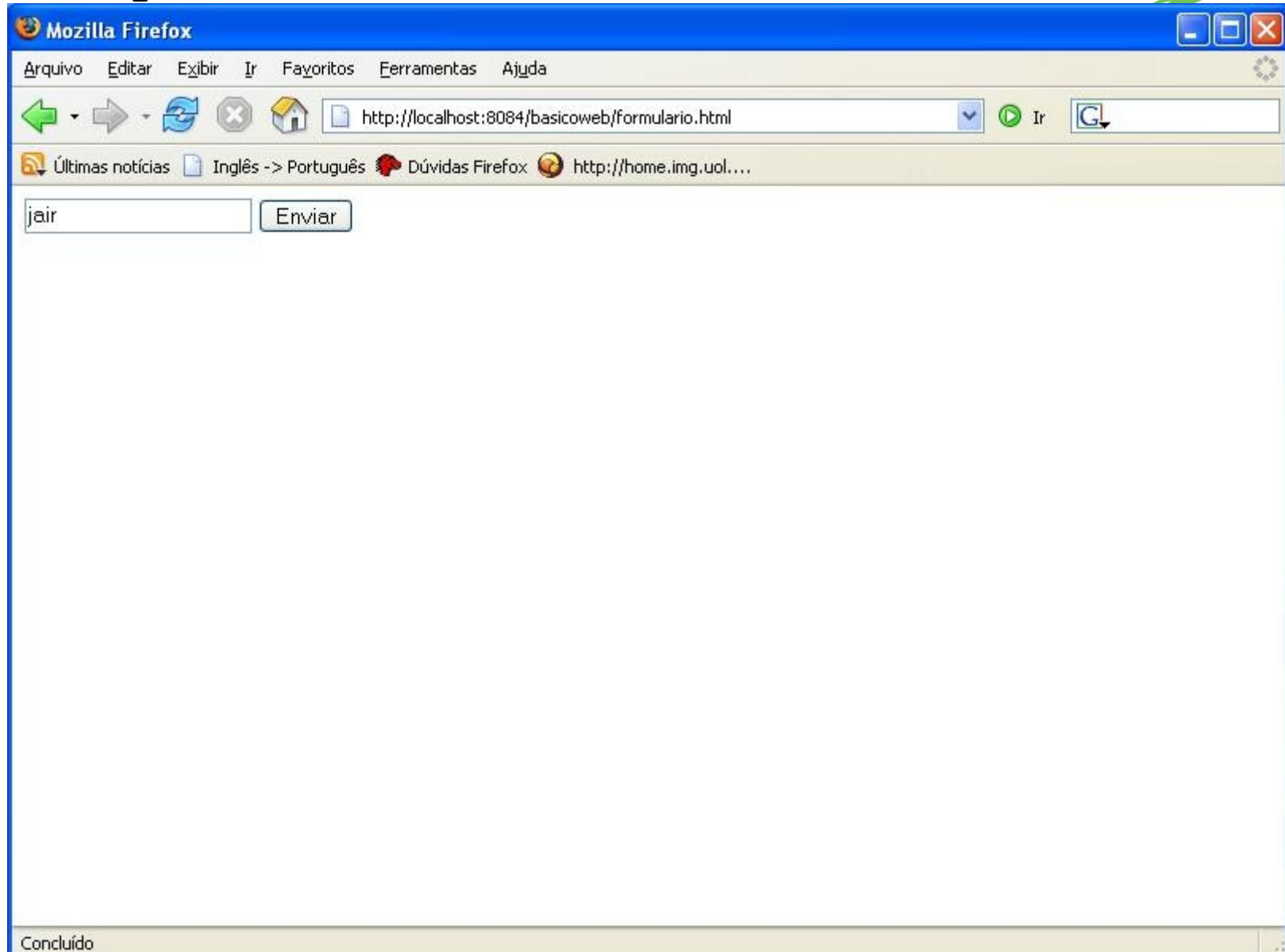
<html>
<head>
<title></title>
</head>
<body>
<form action="http://localhost:8084/basicoweb/AloServletRepasse">
<input type="text" name="nomePessoa" value="" />
<input type="submit" value="Enviar" />
</form>
</body>
</html>
```



Conteúdo dinâmico na internet



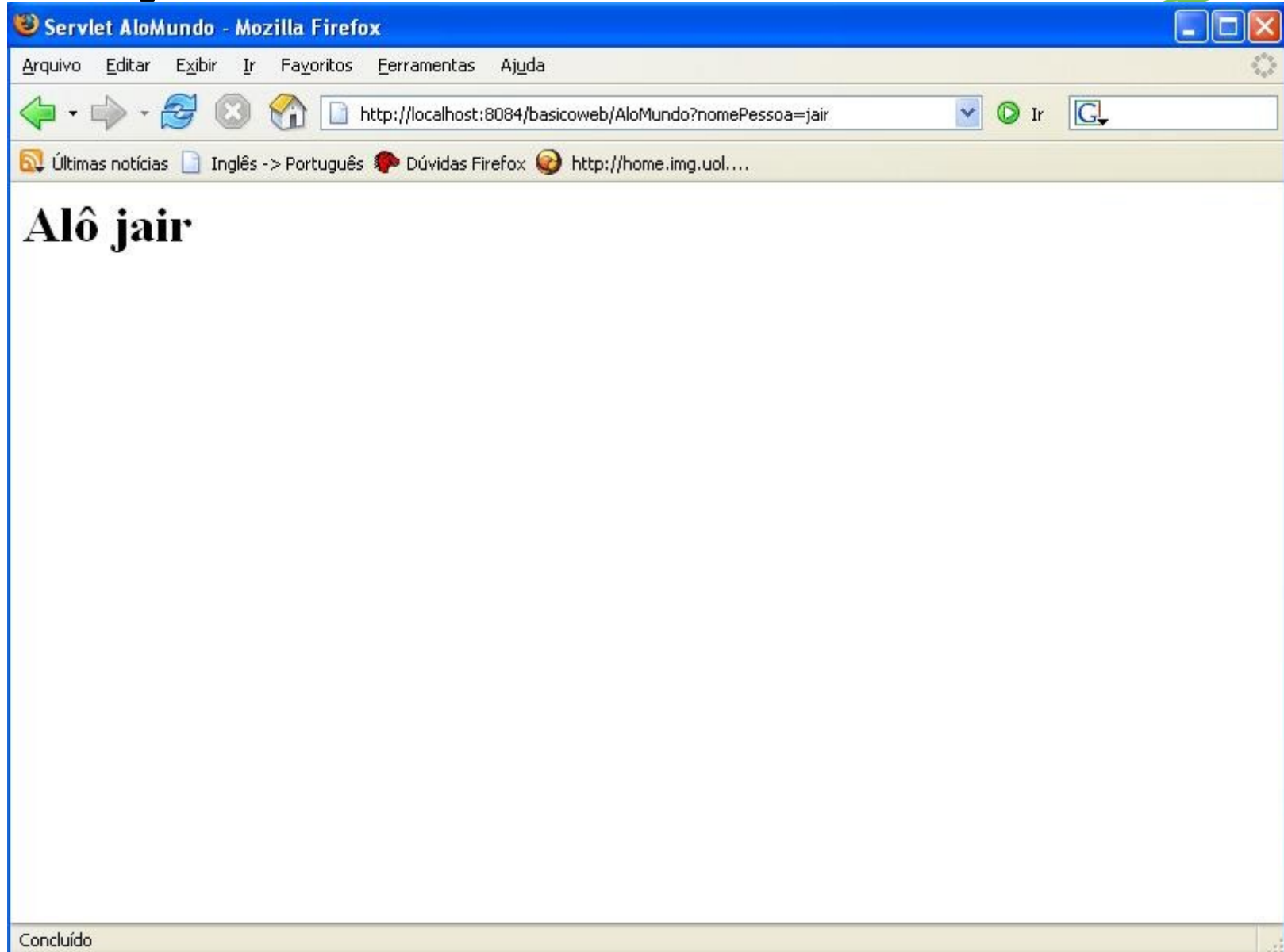
Exemplo:



Conteúdo dinâmico na internet



Exemplo:



Servlet



No método `processReques()` existe dois objetos como argumentos (`HttpServletRequest request`, `HttpServletResponse response`), necessários para a requisição e a resposta, que em qualquer informação que seja necessária para gerar resposta de ser obtida por meio da requisição, em qualquer que seja a resposta, ela deve ser entregue a métodos do objeto de resposta.



Servlet



Toda aplicação Web é configurada através de um descritor (web descriptor), representado pelo arquivo web.xml, contido no diretório WEB-INF da aplicação. É no descritor que são definidos os servlets presentes na aplicação, o mapeamento dos servlets e URLs, as página de erros da aplicação, os filtros e listeners disponíveis e vários outros aspectos de uma aplicação Web.





Exemplo de descritor:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<!-- elementos de web-app, em qualquer ordem -->
</web-app>
```



Servlet



No caso de servlets, estamos interessados em dois elementos:

`<servlet>` e `<servlet-mapping>`

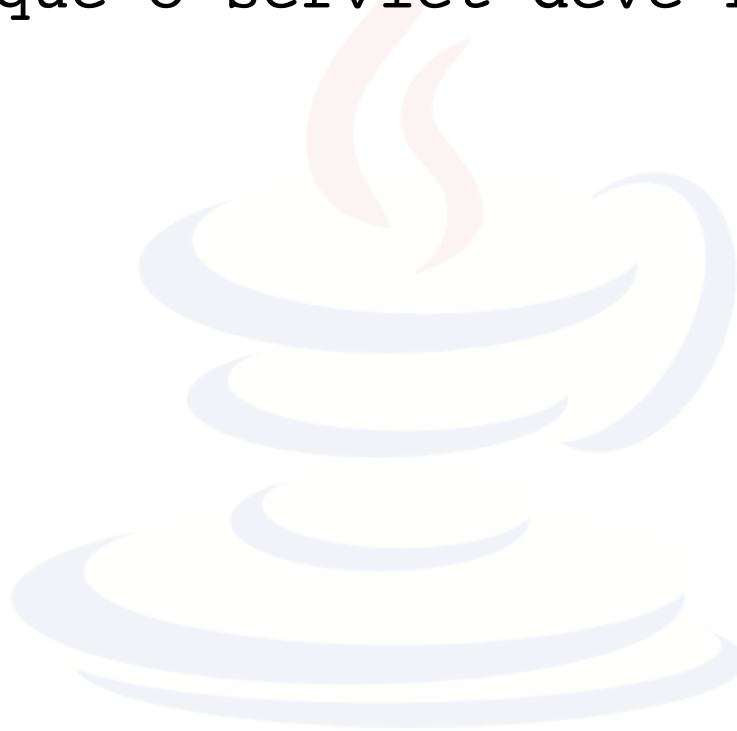
O descritor deve conter um elemento `<servlet>` para cada servlet disponível na aplicação, com os sub-elementos `<servlet-name>` e `<servlet-class>` definindo o nome do servlet (que deve ser único dentro da aplicação) e a classe que implementa. Tomando como exemplo o servlet do exemplo.



Servlet



Já em relação aos mapeamentos, é preciso usar um elemento `<servlet-mapping>` para cada um, incluindo o nome do servlet (definido em `<servlet>`) e o padrão da URL que o servlet deve responder.

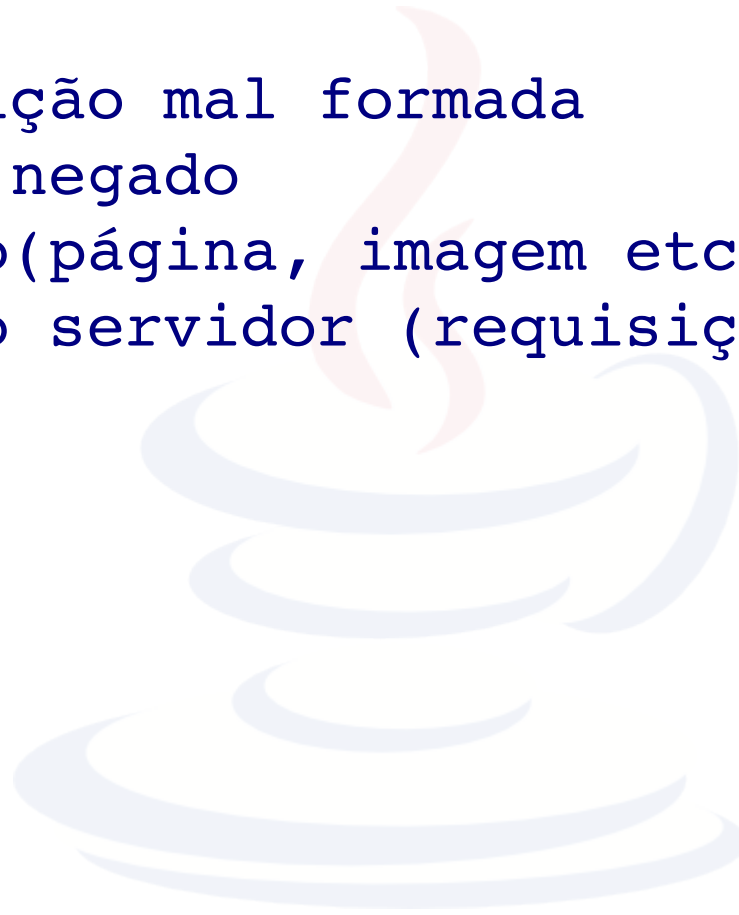


Servlet



Código Significado

| | |
|-----|--|
| 200 | OK |
| 400 | Requisição mal formada |
| 403 | Acesso negado |
| 404 | Recurso(página, imagem etc.) inexistente |
| 500 | Erro no servidor (requisição não pode ser tratada) |





JSP é uma página da Web que contém código Java junto com html.

Está funciona quase como qualquer outra página, sendo normalmente acessada através de um cliente navegador mas com a única diferença que o código Java será executado no servidor.

Essa idéia de colocar código de uma linguagem de programação junto com html não é tão nova. Temos os exemplos como ASP(Active Server Pages) enquanto o Netscape o SSJS(Server-Side Javascript), usando código baseado em visual basic e javascript respectivamente.

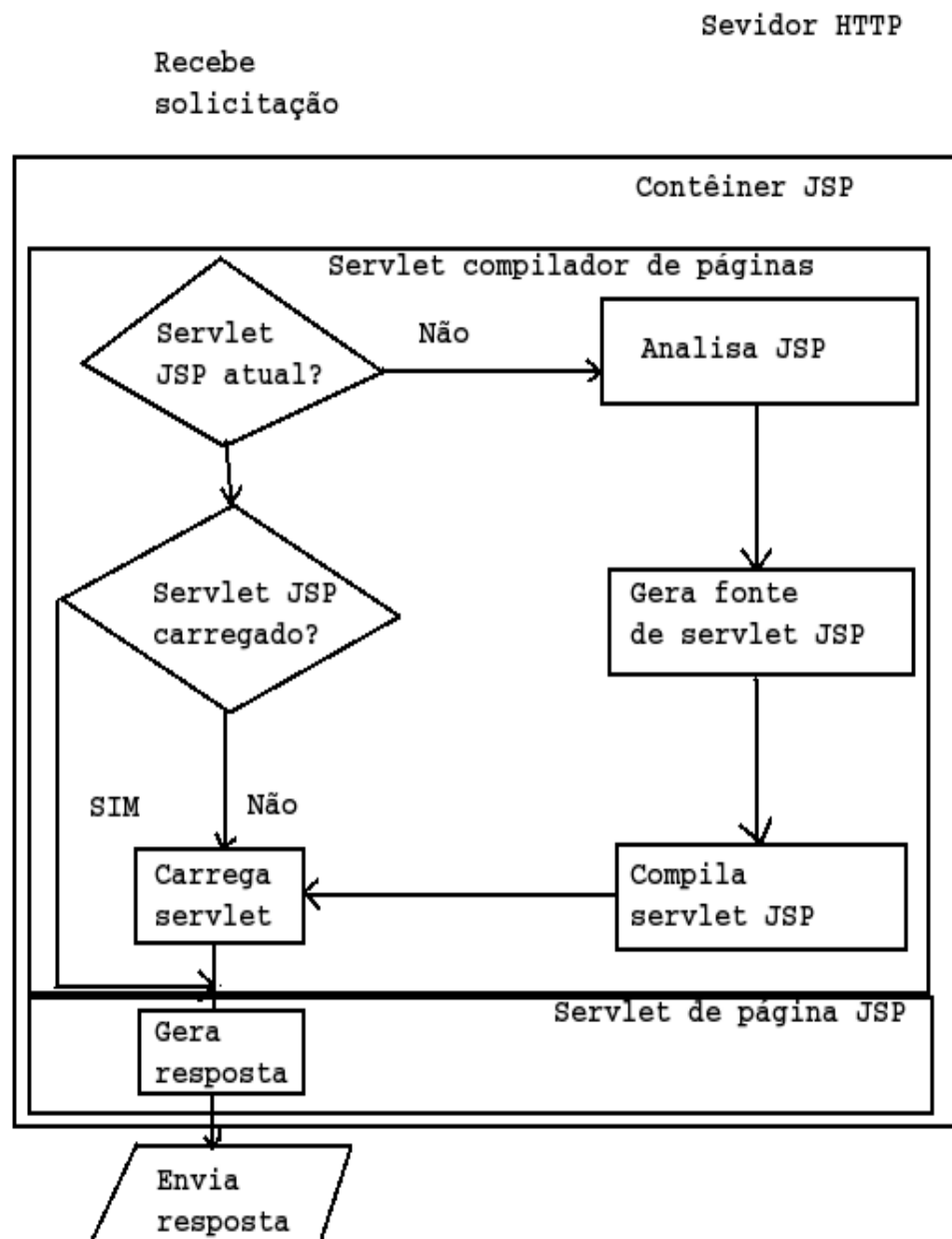




O Web contêiner interpreta o arquivo JSP, o compila e transforma em um servlet. Assim sendo, logo que o arquivo JSP é chamado pela primeira vez por um cliente, um servlet que representa é criado, aplicando todos os benefícios do mesmo para uma página JSP.



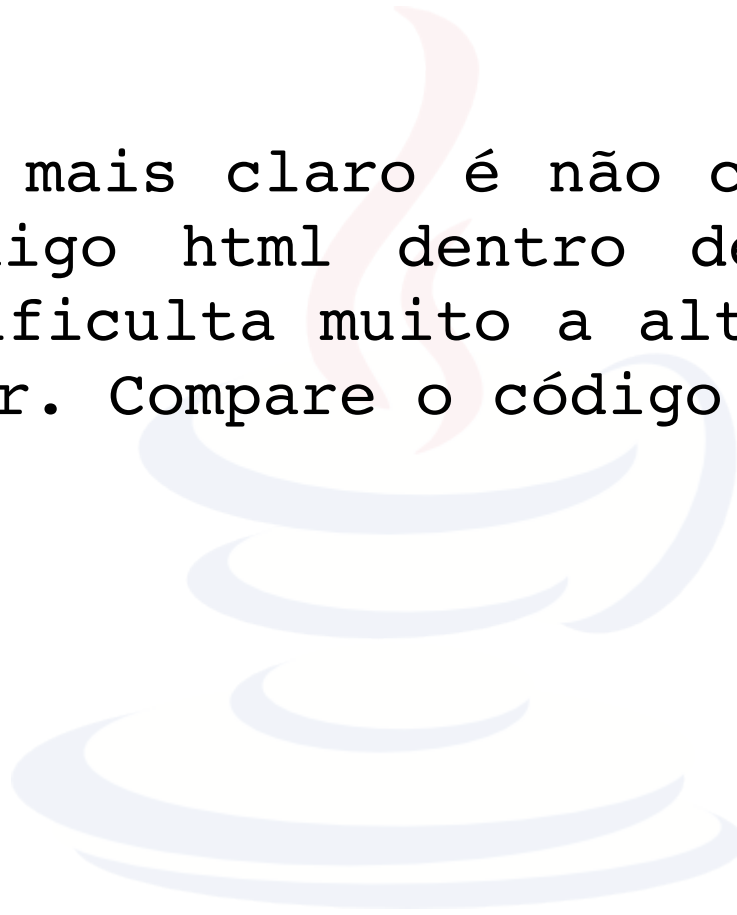
Java Server Pages (JSP)





Vantagens

O benefício mais claro é não colocar uma série imensa de código html dentro de uma classe em java, o que dificulta muito a alteração da página por um designer. Compare o código do servlet.





Objetos implícitos:

| IDENTIFICADOR | Descrição |
|---------------|-------------------------------------|
| APPLICATION | O contexto |
| SESSION | A sessão |
| REQUEST | O objeto de requisição |
| RESPONSE | O objeto de resposta |
| OUT | O stream writer de saída |
| PAGE | A instância da servlet a ser criada |
| PAGECONTEXT | Contexto da página |





Tudo que estiver entre as tags `<%--` e `--%>` é considerado comentário.

As formas normais de comentários em Java também são aceitas dentro do código Java em uma página jsp. Esse código incluso na página é chamado scriptlet.

Para imprimir o valor de uma variável no writer de saída existe o atalho `<%= valor %>`



Encaminhamento de requisições



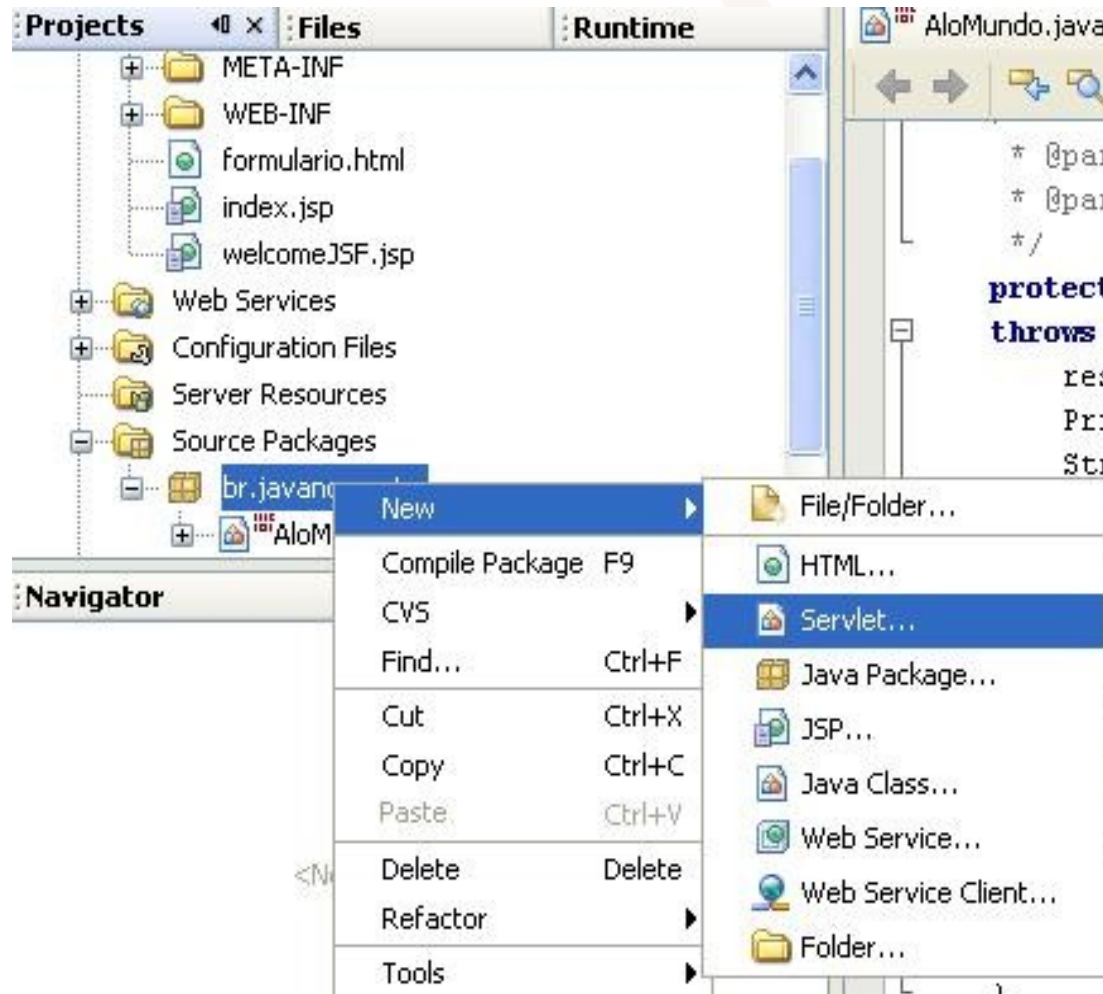
Uma técnica interessante da API de Servlets é o encaminhamento (forwarding) de requisições entre servlets. Um servlet recebe a requisição, faz algum pre-processamento e repassa a requisição para outro servlet. Este fica responsável por enviar a resposta final ao navegador (ou por encaminhá-la a outro servlet).



Encaminhamento de requisições



Exemplo:



Encaminhamento de requisições



Exemplo:

The screenshot shows the 'New Servlet' wizard with the following configuration:

| Steps | Name and Location |
|---------------------------------|--|
| 1. Choose File Type | Class Name: AloServletRepasse |
| 2. Name and Location | Project: basicoweb |
| 3. Configure Servlet Deployment | Location: Source Packages |
| | Package: br.javanoroeste |
| | Created File: C:\projetos\basicoweb\src\br\javanoroeste\AloServletRepasse.java |

Navigation buttons at the bottom: < Back, Next >, Finish, Cancel, Help.



Encaminhamento de requisições



Exemplo:

New File

Steps

1. Choose File Type
2. Name and Location
3. **Configure Servlet Deployment**

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name:

Servlet Name:

URL Pattern(s):

Initialization Parameters:

| Name | Value |
|------|-------|
|------|-------|

New Edit... Delete

< Back Next > Finish Cancel Help



Encaminhamento de requisições



Exemplo:

```
public class AloServletRepassa extends HttpServlet {  
  
    /** Processes requests for both HTTP GET and POST methods.  
    * @param request servlet request  
    * @param response servlet response  
    */  
  
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)  
    throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        /* TODO output your page here  
        out.println("<html>");  
        out.println("<head>");  
        out.println("<title>Servlet AloServletRepassa</title>");  
        out.println("</head>");  
        out.println("<body>");  
        out.println("<h1>Servlet AloServletRepassa at " + request.getContextPath () + "</h1>");  
        out.println("</body>");  
        out.println("</html>");  
        */  
        out.close();  
    }  
  
    HttpServlet methods. Click on the + sign on the left to edit the code.  
}
```



Encaminhamento de requisições



Exemplo:

```
import java.io.*;
import java.net.*;

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

/**
 *
 * @author carlos
 * @version
 */
public class AloServletRepassa extends HttpServlet {

    /** Processes requests for both HTTP GET and POST methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        String nomePessoa = request.getParameter("nomePessoa");
        request.setAttribute("nome", nomePessoa);
        RequestDispatcher dispatcher = request.getRequestDispatcher("/view.jsp");
        dispatcher.forward(request, response);
    }

    HttpServlet methods. Click on the + sign on the left to edit the code.
}
```

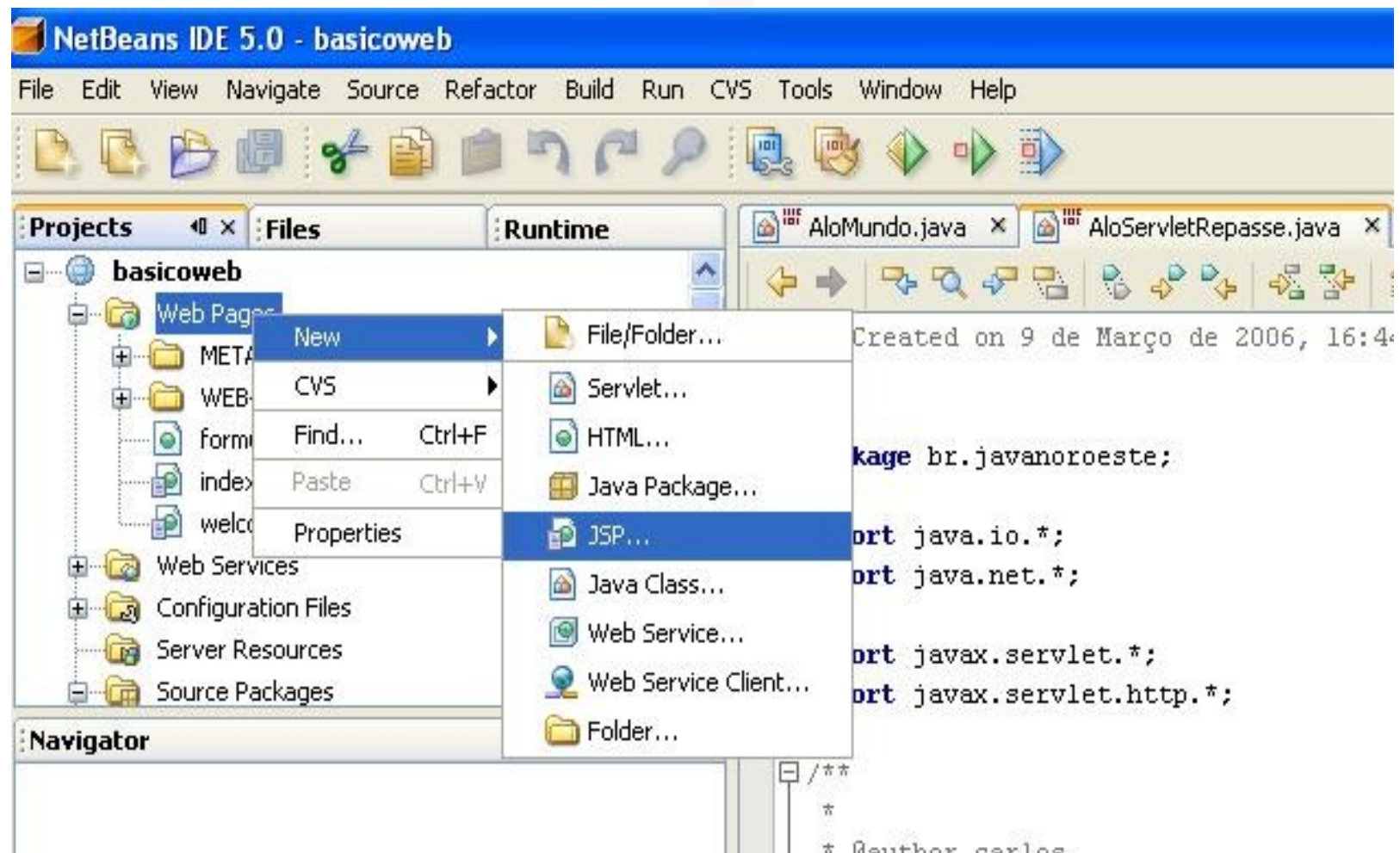
**Código
Alterado**



Encaminhamento de requisições



Exemplo:



Encaminhamento de requisições



Exemplo:

New JSP File

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

JSP File Name:

Project:

Location:

Folder:

Created File:

Options: ☒ JSP File (Standard Syntax) ☐ JSP Document (XML Syntax)

☐ Create as a JSP Segment

Description:

Encaminhamento de requisições



Exemplo:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<!--
The taglib directive below imports the JSTL library. If you uncomment it,
you must also add the JSTL library to the project. The Add Library... action
on Libraries node in Projects view can be used to add the JSTL 1.1 library.
--%>
<!--
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
--%>

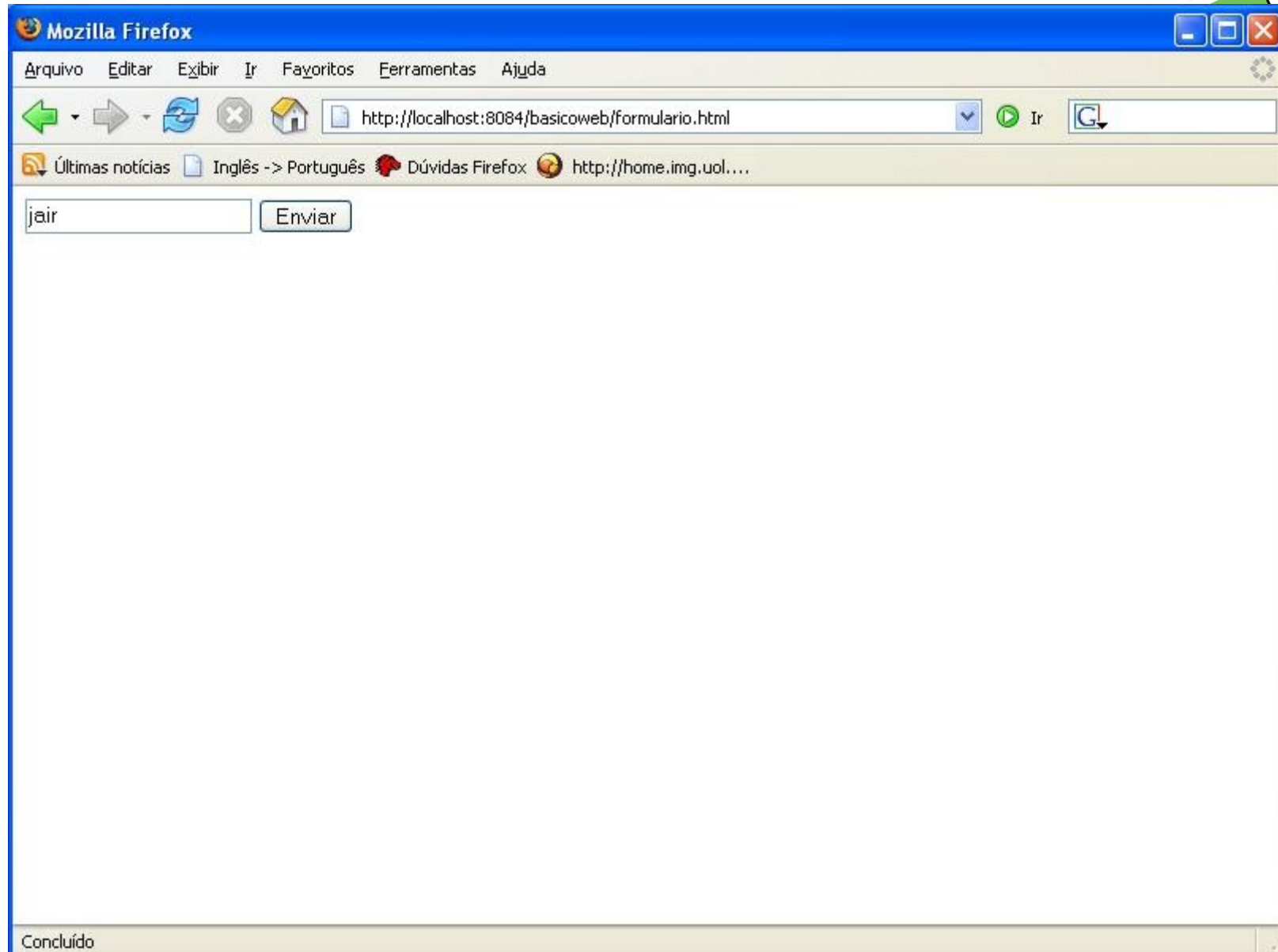
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>

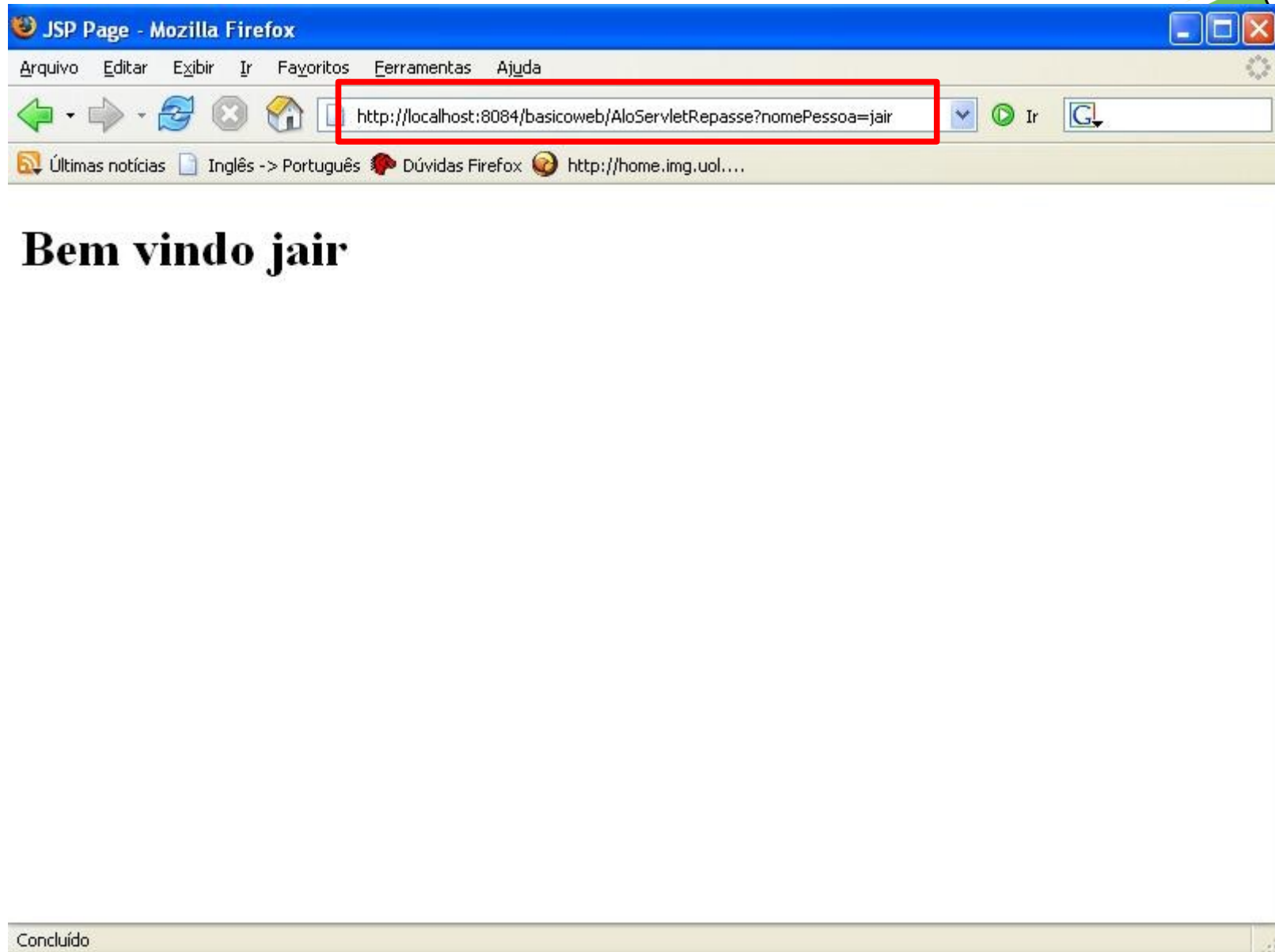
<hl>Bem vindo <%= request.getAttribute("nome")%></hl>

<!--
This example uses JSTL, uncomment the taglib directive above.
To test, display the page like this: index.jsp?sayHello=true&name=Murphy
--%>
<!--
```

Encaminhamento de requisições



Encaminhamento de requisições

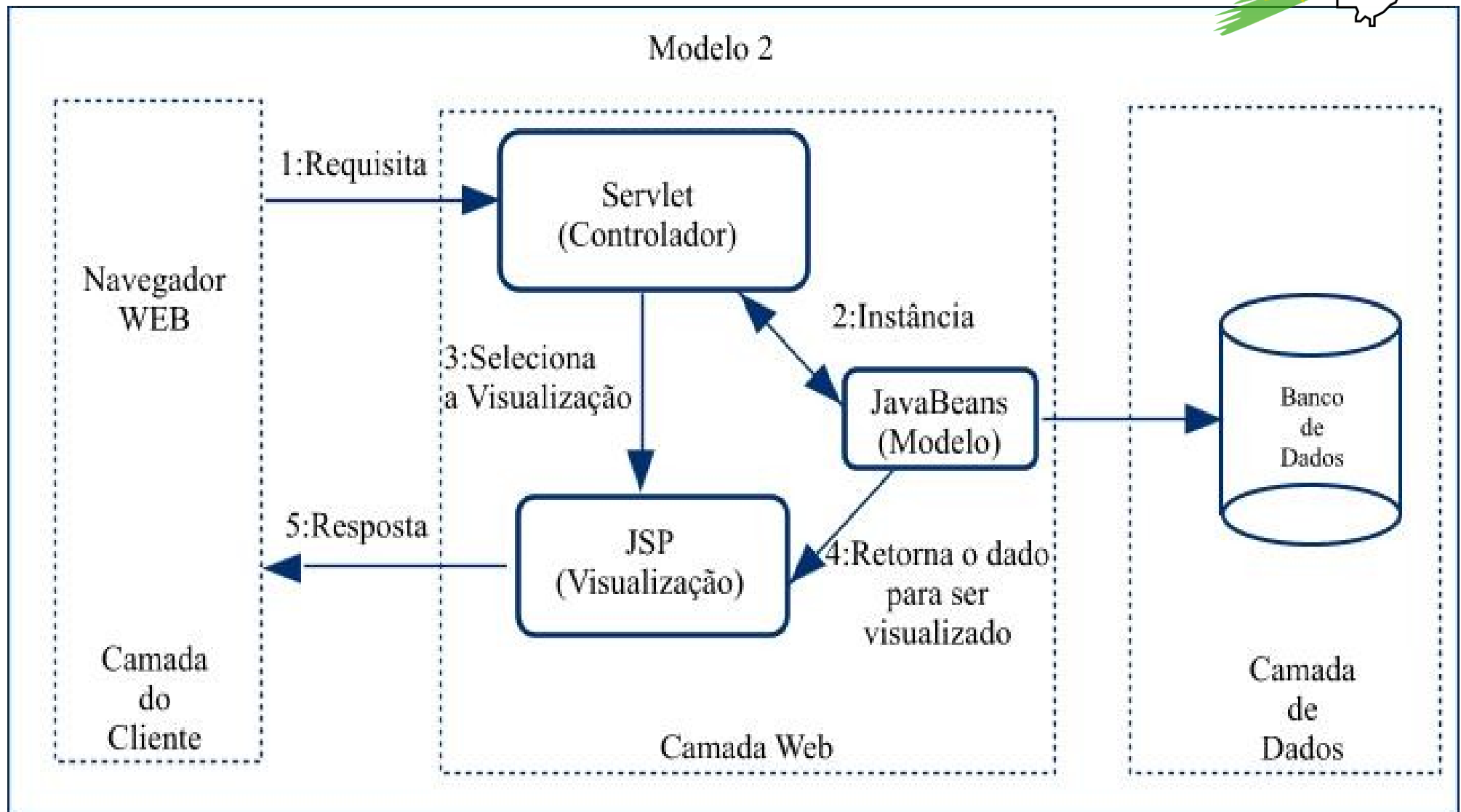




A sigla MVC vem do inglês Model-View-Controller (Modelo-Visualização-Controle), é um padrão que oferece uma boa separação entre as responsabilidades do sistema nas tarefas de dados (modelo), o gerenciamento das tarefas (controle) e as relações entre os dados e a forma (visualização) no qual serão exibidas.

MVC é o padrão para desenvolvimento Java para Web da SUN Microsystems, criado originalmente na linguagem Smalltalk para desenvolvimento de aplicações gráficas (GUI).







O ganho para o desenvolvimento é separar melhor as responsabilidades, diminuir código JSP e reusabilidade. Neste padrão MVC, é utilizada a arquitetura de implementação com servlet, JavaBeans e JSP. O Controle fica a cargo de um servlet que gerencia o fluxo da aplicação Web, o Modelo é delegado para um JavaBeans e a Visualização é de responsabilidade de uma JSP, que produz uma saída em HTML para o navegador Web do cliente.





JavaBeans são objetos escritos em Java, cujas implementações estão de acordo com um conjunto de convenções projetadas para promover modularidade e reusabilidade. **JavaBeans** usa um estilo de programação que conduz a partes de código de programa independentes, pelo qual encapsulam comportamento, funcionalidade ou dados relacionados, podendo ser usados e reutilizados em múltiplos contextos sem que seja preciso conhecer detalhes de sua operação interna.





**Isto é só o início...
Bem vindo ao mundo Java**

