

**INSTITUTO
FEDERAL**

Paraíba

Campus
Cajazeiras

PROGRAMAÇÃO P/ WEB 1

Recapitulando . . .

PROF. DIEGO PESSOA

✉ DIEGO.PESSOA@IFPB.EDU.BR

🐱 [@DIEGOEP](https://github.com/DIEGOEP)



**CST em Análise e
Desenvolvimento
de Sistemas**

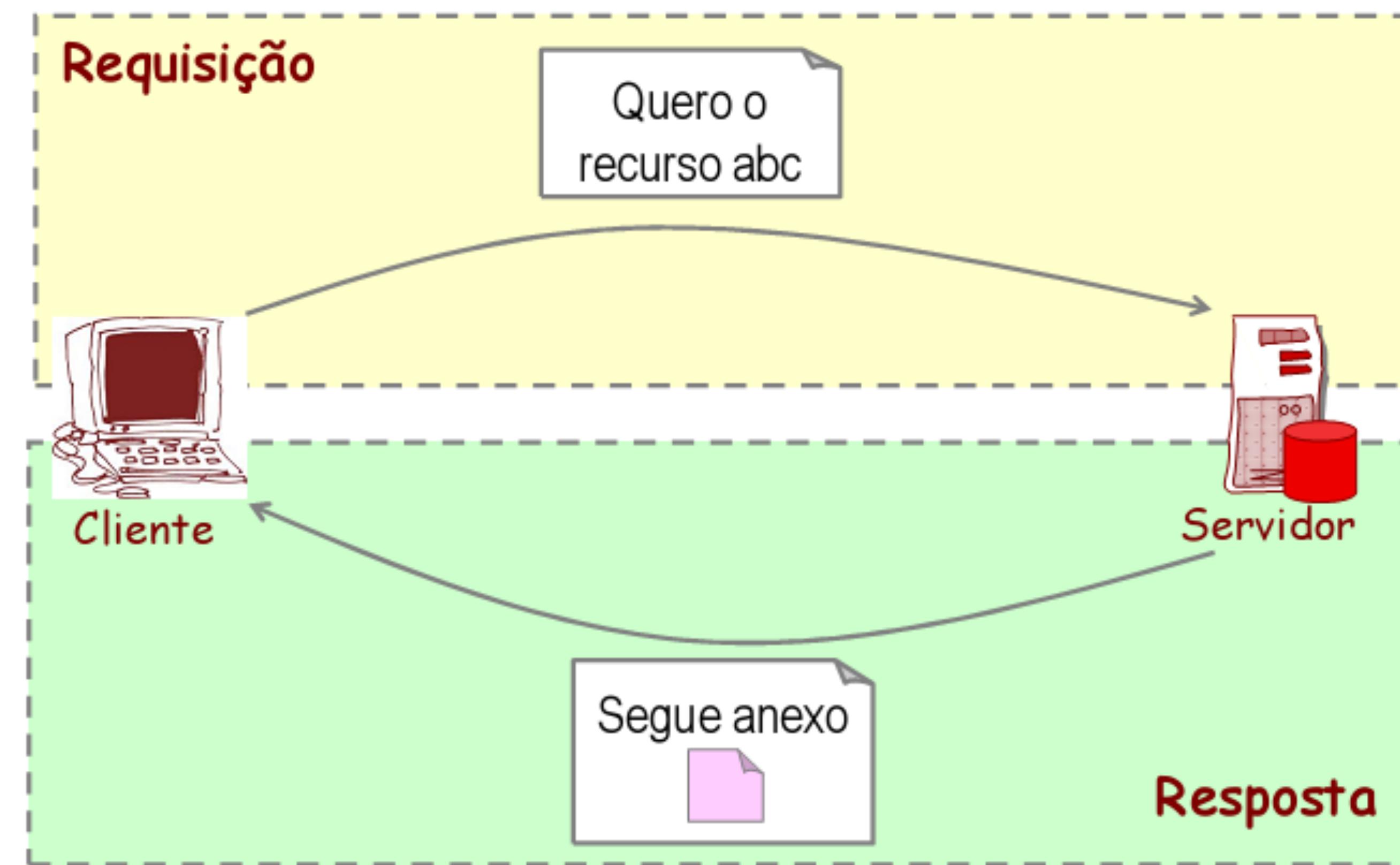
Roteiro

- ▶ Aplicações Web
- ▶ Padrão MVC
- ▶ JDBC
- ▶ Padrão DAO
- ▶ Introdução a Servlets
- ▶ Servlets para o gerenciamento de aplicações

Aplicações Web

Protocolo HTTP - Atuação

- ▶ Atua na forma de requisição/resposta, na qual é estabelecida uma conexão que depois é fechada.



Protocolo HTTP - Atuação

► Requisição HTTP GET

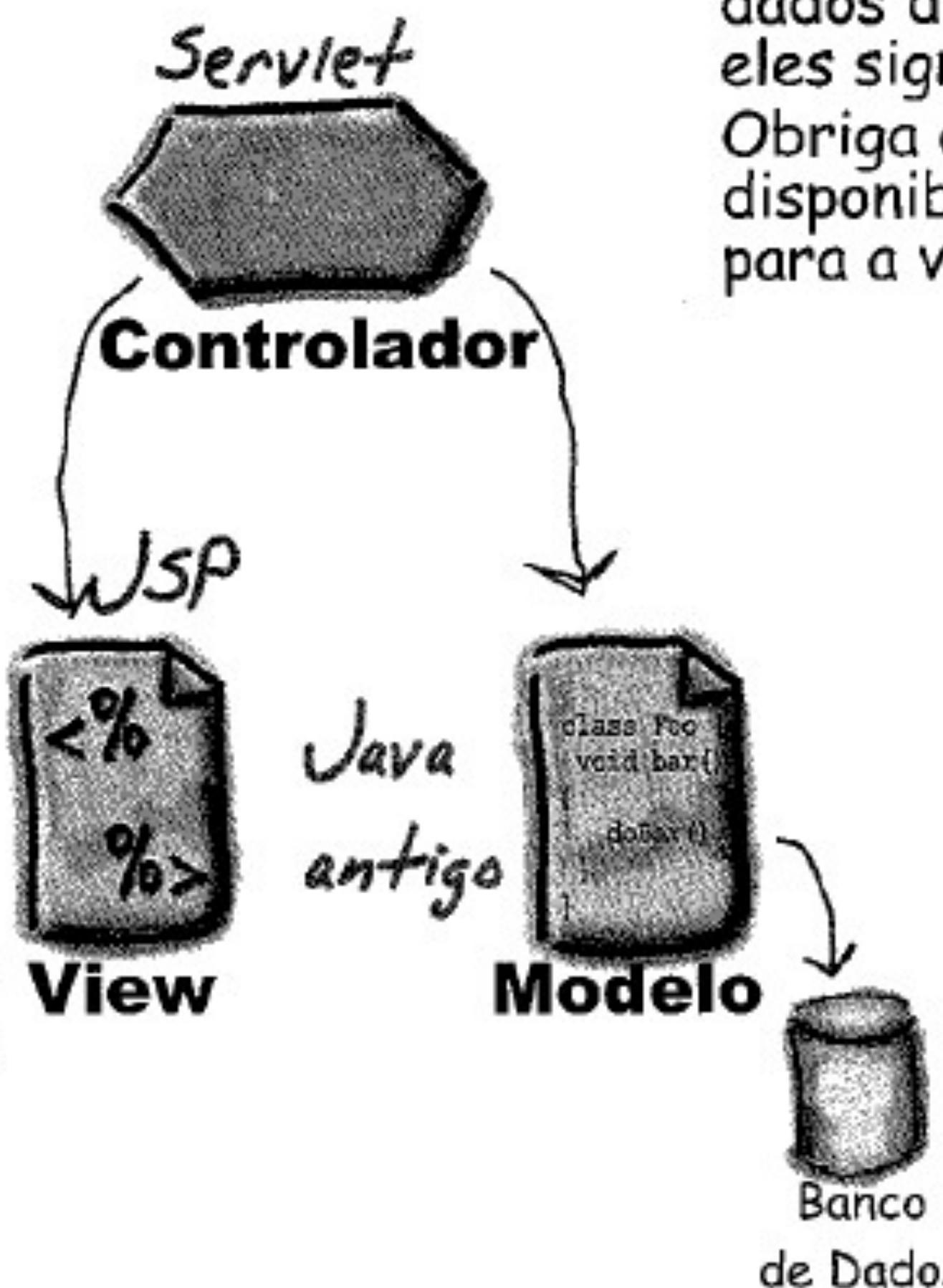
método	path do recurso	dados	versão do protocolo
GET	/loja/consulta.jsp?id=123&tipo=10		HTTP/1.1
Host: www.lojatal.com.br			
User-Agent: IE/7.0			
Accept: text/html,application/xml,text/html,text/plain,image/gif,image/jpeg			
Connection: keep-alive			
Cookie: jsessionid=12abd4f67cc34			
parâmetros do request			

Padrão MVC

O MVC no mundo Servlet & JSP

VIEW

Responsável pela apresentação. Ela recebe o estado do modelo do Controlador (embora não diretamente; o Controlador põe os dados do modelo em um lugar onde a View possa encontrá-lo). Também é a parte que recebe os dados de entrada do usuário que volta ao Controlador.



CONTROLADOR

Retira da solicitação do usuário os dados de entrada e interpreta o que eles significam para o modelo.

Obriga o modelo a se atualizar e disponibiliza o estado do novo modelo para a view (o JSP).

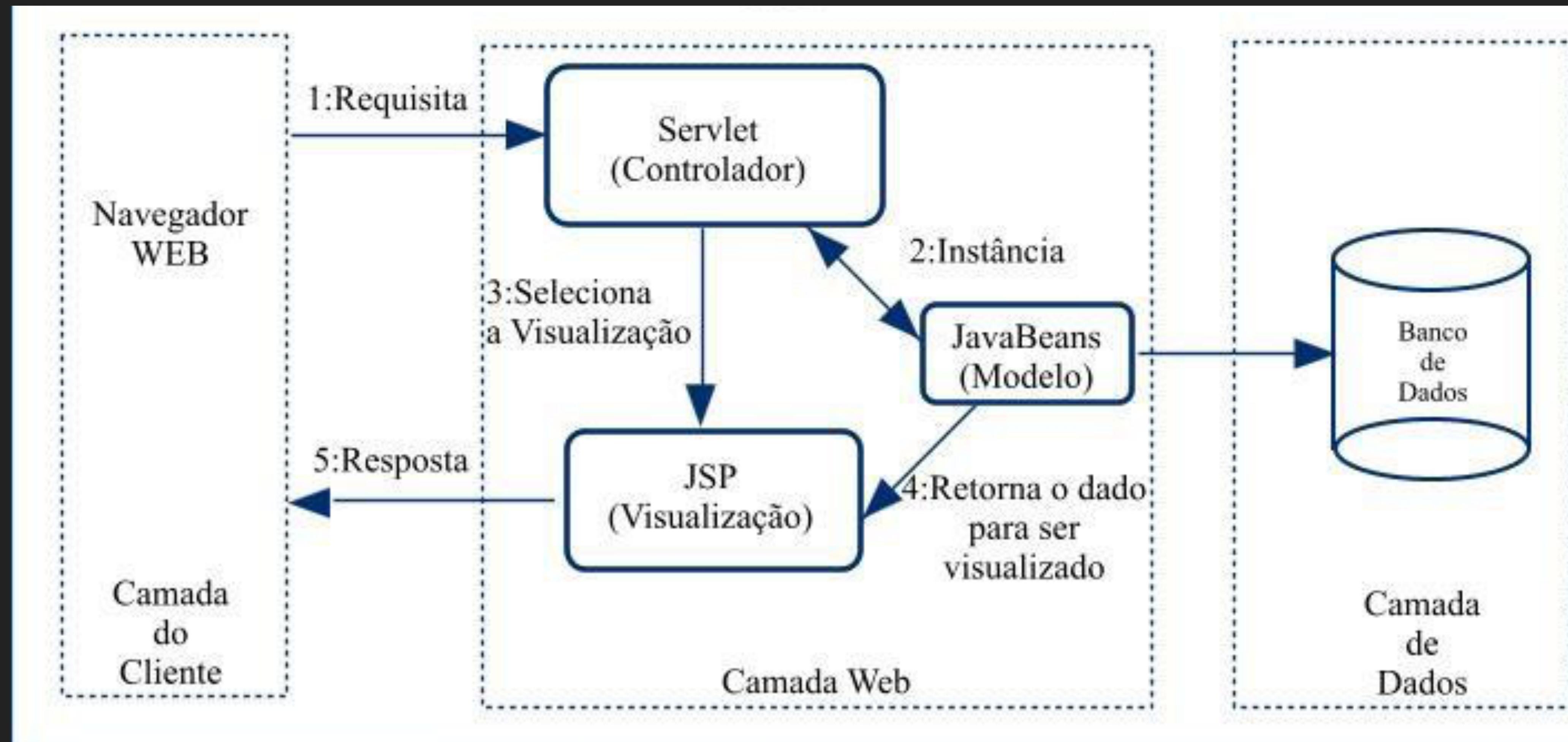
MODELO

Abriga a verdadeira lógica e o estado do modelo. Em outras palavras, ele conhece as regras para obtenção e atualização do estado.

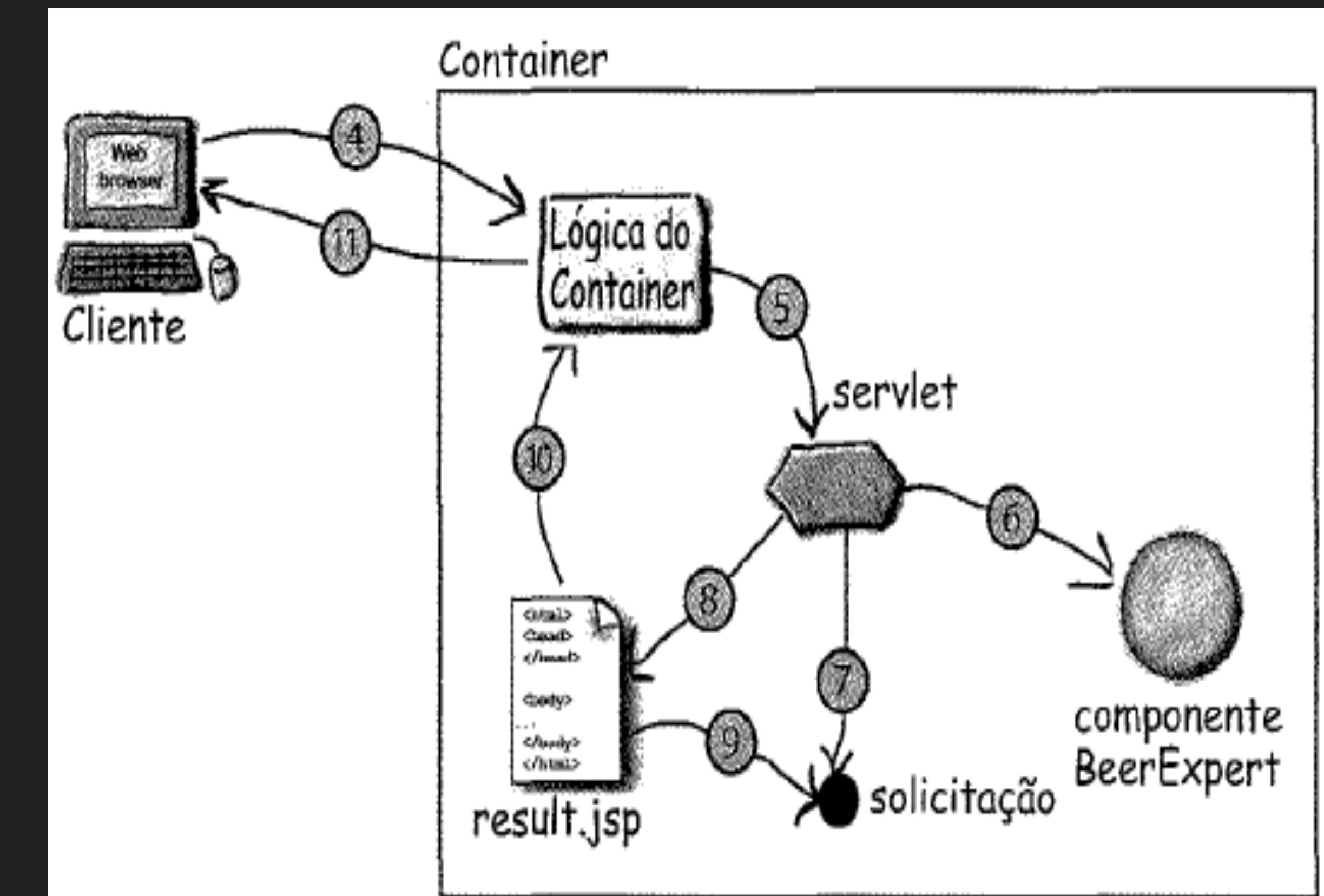
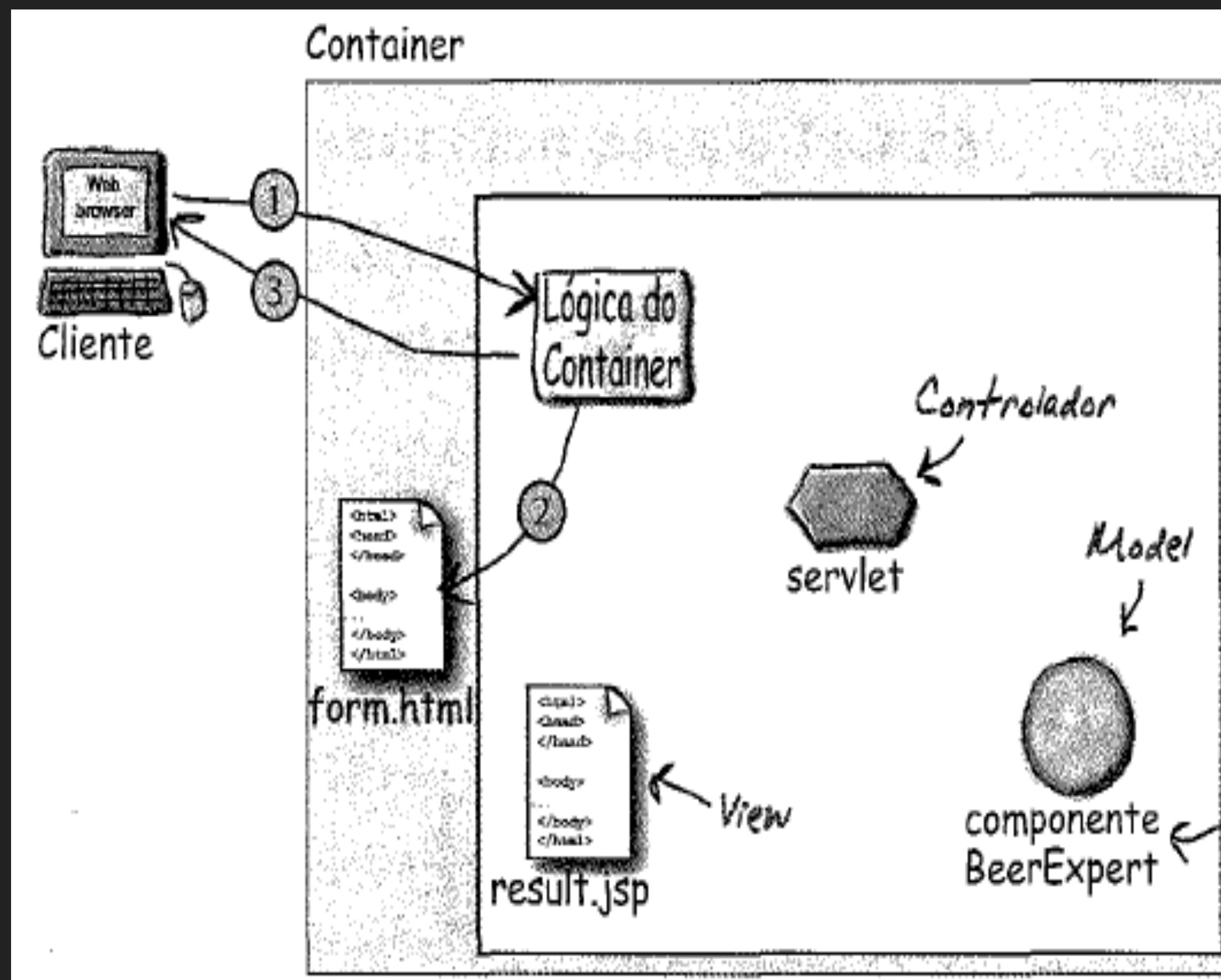
O conteúdo de um Carrinho de Compras (e as regras sobre o que fazer com isso) seria parte do Modelo no MVC.

E a única parte do sistema que se comunica com o banco de dados (embora ele provavelmente use outro objeto para a verdadeira comunicação com o DB, mas guardaremos este padrão para mais tarde...).

MVC NA CAMADA WEB EM JAVA

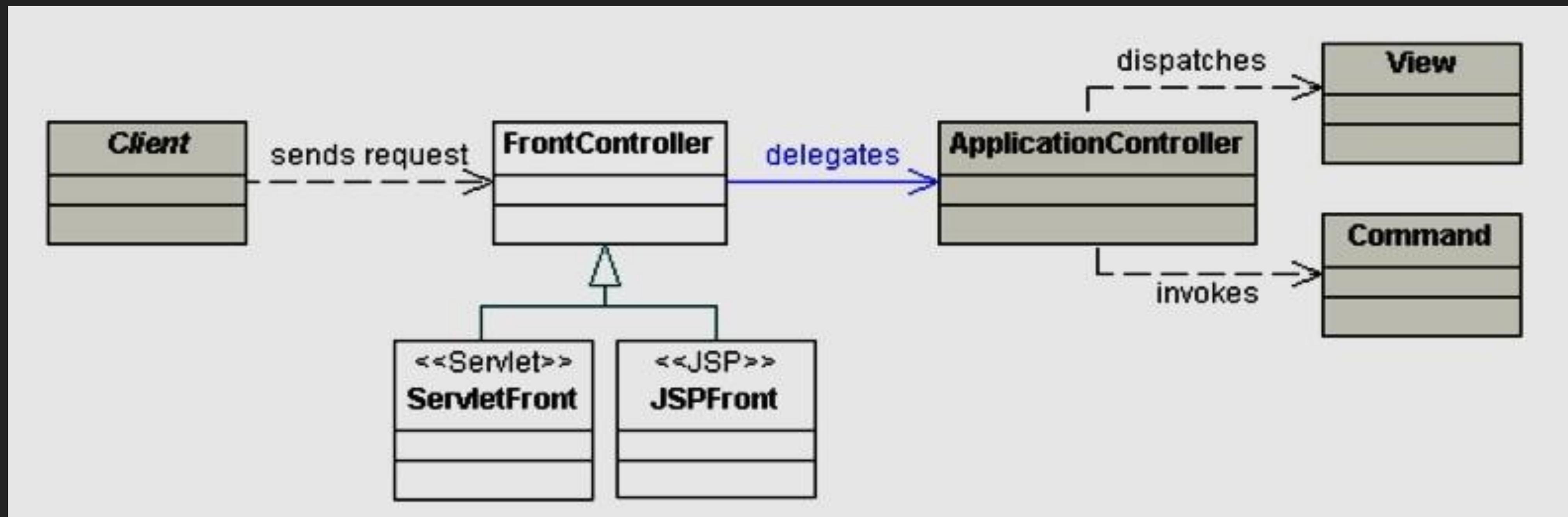


APLICANDO MVC NO PROJETO DA APLICAÇÃO WEB



O PADRAO FRONT CONTROLLER

- ▶ Diagrama de Classes:



O PADRAO COMMAND

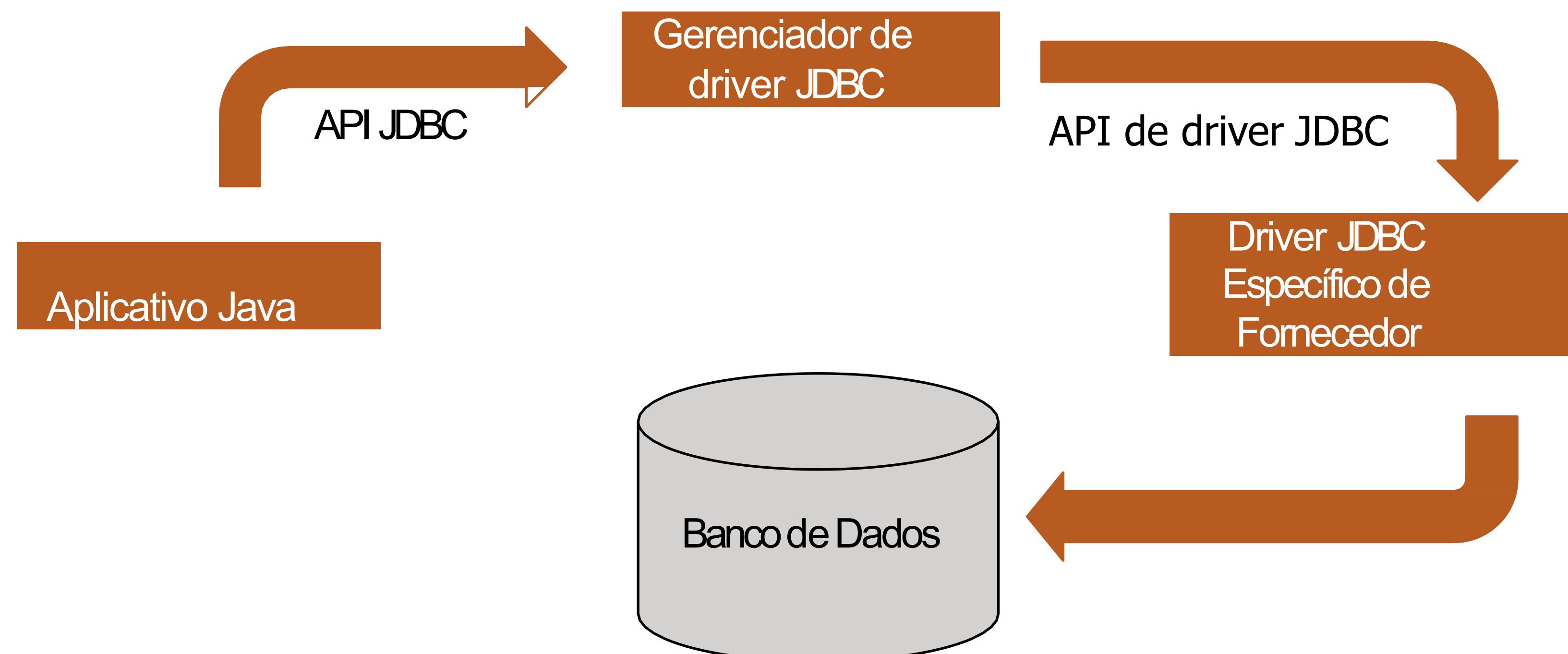
- ▶ Definindo a interface dos comandos:

```
public interface Command {  
    public void execute(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, BookApplicationException, IOException;  
}
```

JDBC

Carregando o Driver JDBC

- ▶ O driver é a parte do software que sabe como conversar com o servidor de banco de dados.



Estabelecendo uma Conexão

- ▶ Exemplo de um fragmento de código que carrega o driver JDBC e estabelece uma conexão com um banco de dados via JDBC.

```
Class.forName ("org.postgresql.Driver");
String url = "jdbc:postgresql://localhost:5432/BancoTeste";
String login = "mylogin";
String password = "mypasswd";
Connection connection = DriverManager.getConnection(url, login, password);
```

Estabelecendo uma Conexão

- ▶ A classe Connection estabelece alguns métodos úteis:
- ▶ **prepareStatement:** cria consultas pré-compiladas para submissão ao banco de dados.
- ▶ **prepareCall:** Acessa procedimento armazenados no banco de dados.
- ▶ **rollback/commit:** Controla o gerenciamento de transação.
- ▶ **close:** Encerra a conexão aberta.
- ▶ **isClosed:** Determina se a conexão expirou ou foi fechada explicitamente.

Executando Comandos SQL

- ▶ Exemplo de um fragmento de código que recupera o nome de todos os alunos:

[...]

```
String query = “Select Nome FROM Aluno”;  
ResultSet result = stat.executeQuery(query);  
Collection nomeDosAlunos = new Vector();  
while(result.next())  
    nomeDosAlunos .add(result.getString(“Nome”));
```

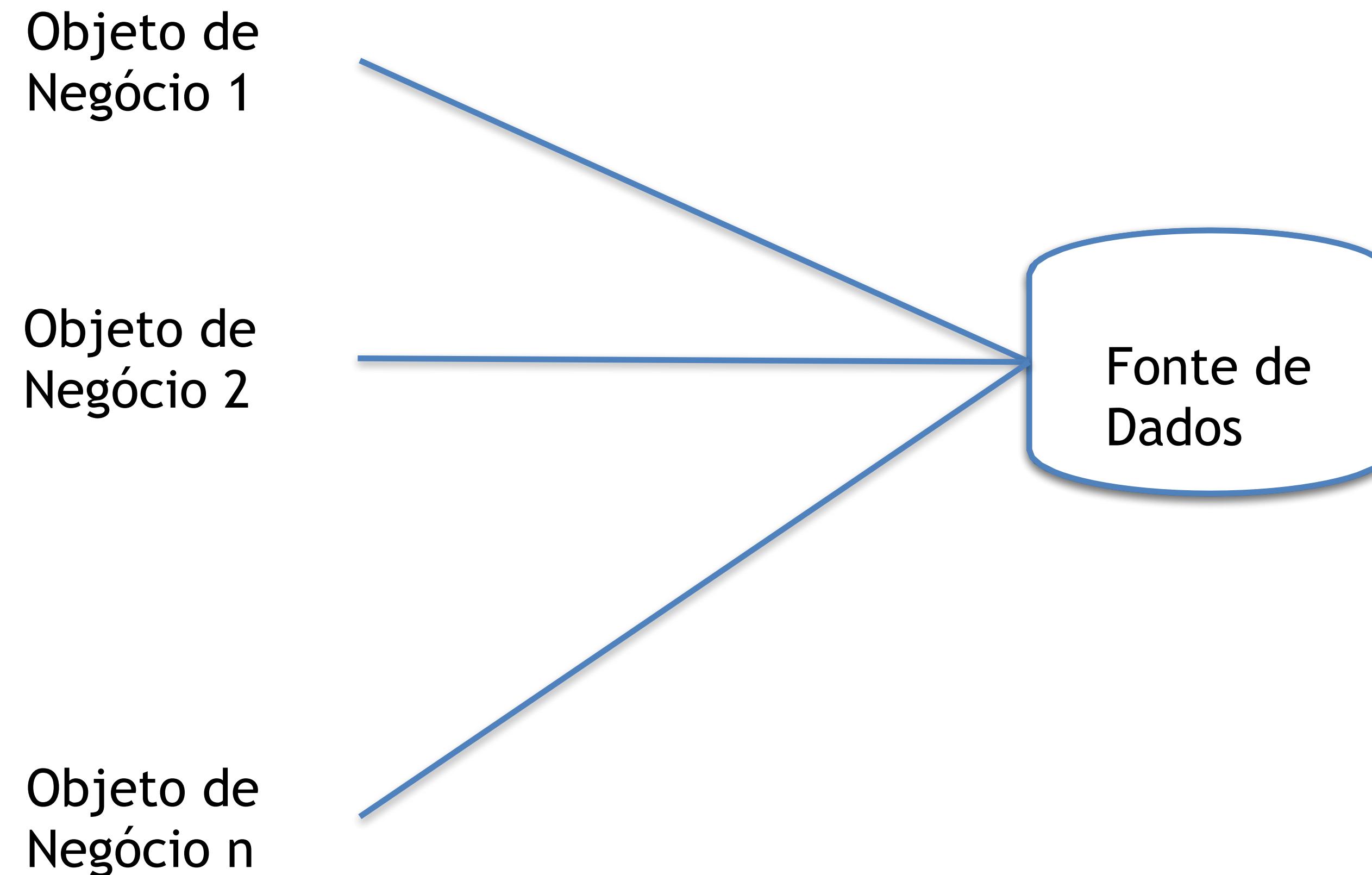
Executando Comandos SQL

- Exemplo (continuação):

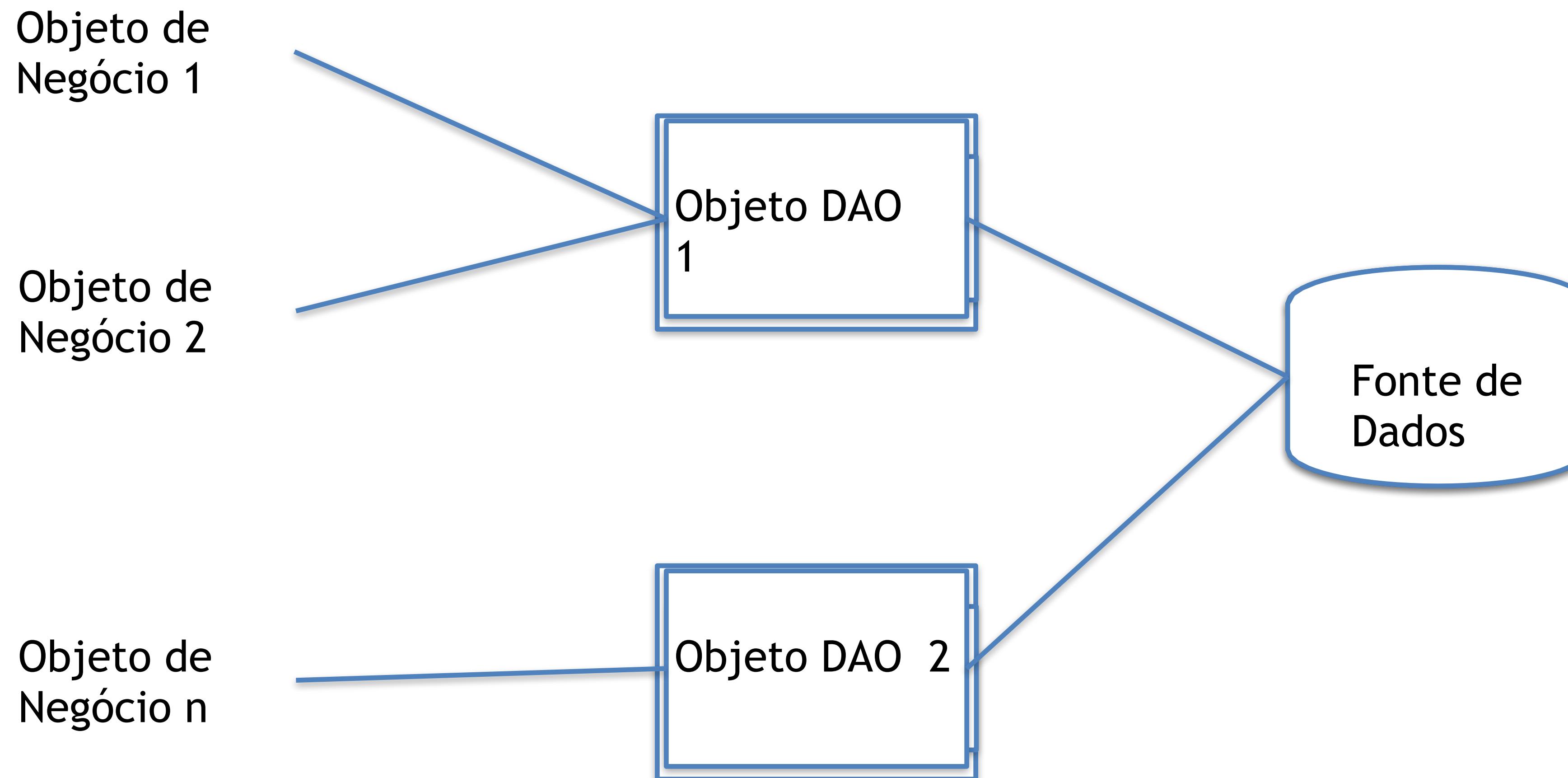
```
public Iterator getAlunos(int minIdade, int maxIdade) throws SQLEXception{  
    String query = "SELECT Nome FROM Aluno WHERE Idade >= ? AND Idade <= ?";  
    PreparedStatement stat = connection.prepareStatement(query);  
    stat.setInt (1, minIdade);  
    stat.setInt(2, maxIdade);  
    Collection result = new Vector();  
    ResultSet rs = stat.executeQuery();  
    while(rs.next())  
        result.add(rs.getString("Nome"));  
    return result.iterator();  
}
```

Padrão DAO

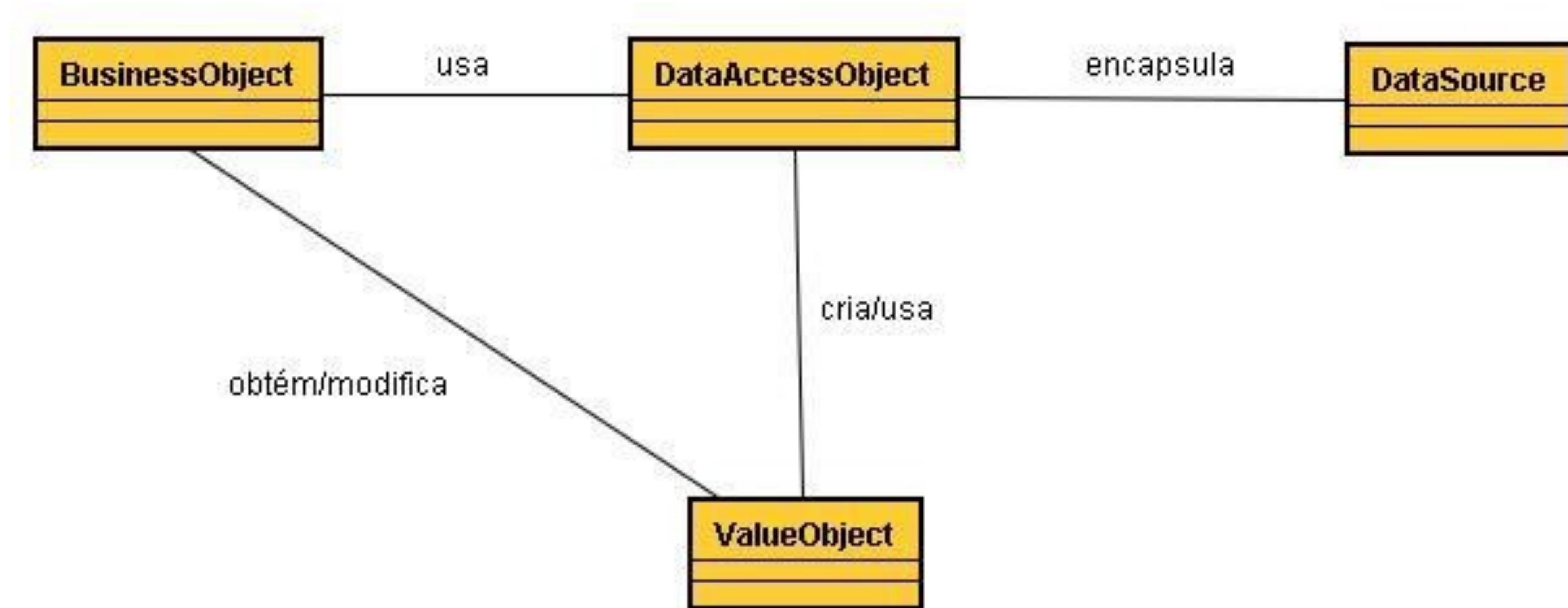
- ▶ Na prática, temos um cenário parecido com este:



- Com o padrão DAO, temos agora o seguinte cenário:



O Padrão DAO - Estrutura



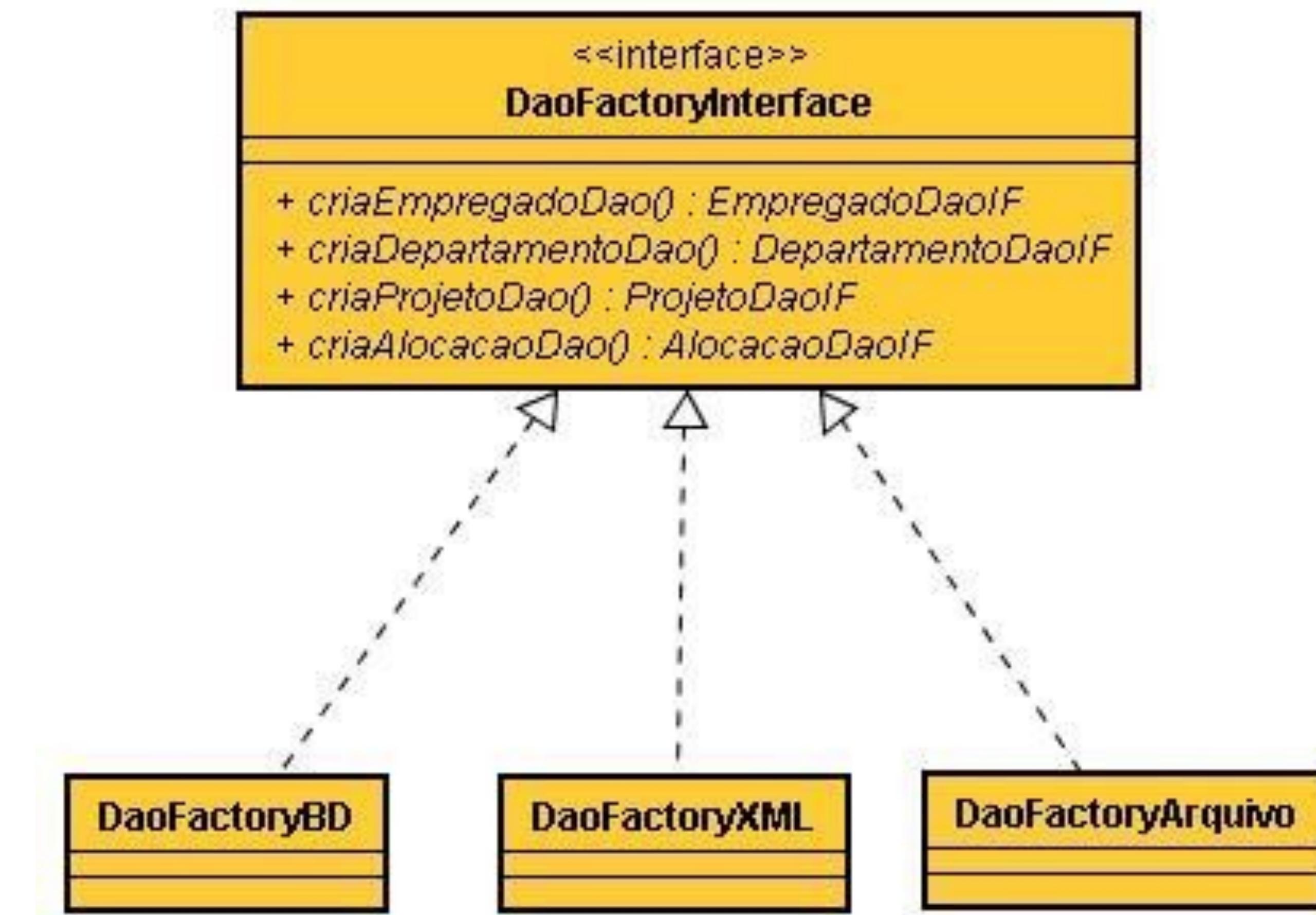
Implementação - DAO

- ▶ Criando a classe concreta que implementa a interface:

```
public void persiste(Empregado empregado) throws PersistenciaException{
    String sql = "INSERT INTO EMPREGADO (Matricula, Nome, Salario, " +
                 "Supervisor, CodDepartamento) VALUES (?, ?, ?, ?, ?) ";
    try{
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setString(1, empregado.getMatricula());
        statement.setString(2, empregado.getNome());
        statement.setFloat(3, empregado.getSalario());
        statement.setString(4, empregado.getSupervisor().getSupervisor().getMatricula());
        statement.setInt(5, empregado.getDepartamento().getCodigo());
        statement.execute();
        statement.close();
    }
    catch(Exception e){
        throw new PersistenciaException(e);
    }
}
```

Implementação - Fábrica de DAOs

- ▶ Estrutura de implementação do DAO:



Implementação - Fábrica de DAOs

- ▶ Instanciando a fábrica de DAOs:

```
public class DaoFactory {  
  
    public static DaoFactoryIF createFactory() {  
        return new DaoFactoryBD();  
    }  
  
}
```

Implementação - Objetos de Negócio

- ▶ Note que o objeto de negócio não sabe nada da implementação da fonte de dados;
- ▶ Ele também não conhece detalhes sobre a implementação da fábrica;
- ▶ Ele conhece apenas a interface que ele precisa utilizar;
- ▶ Assim, podemos alterar a implementação das fábricas sem ter de alterar o objeto de negócio;
- ▶ O mesmo acontece para os objetos que implementam os DAOs.

► Implementando o objeto de negócio para múltiplas fábricas

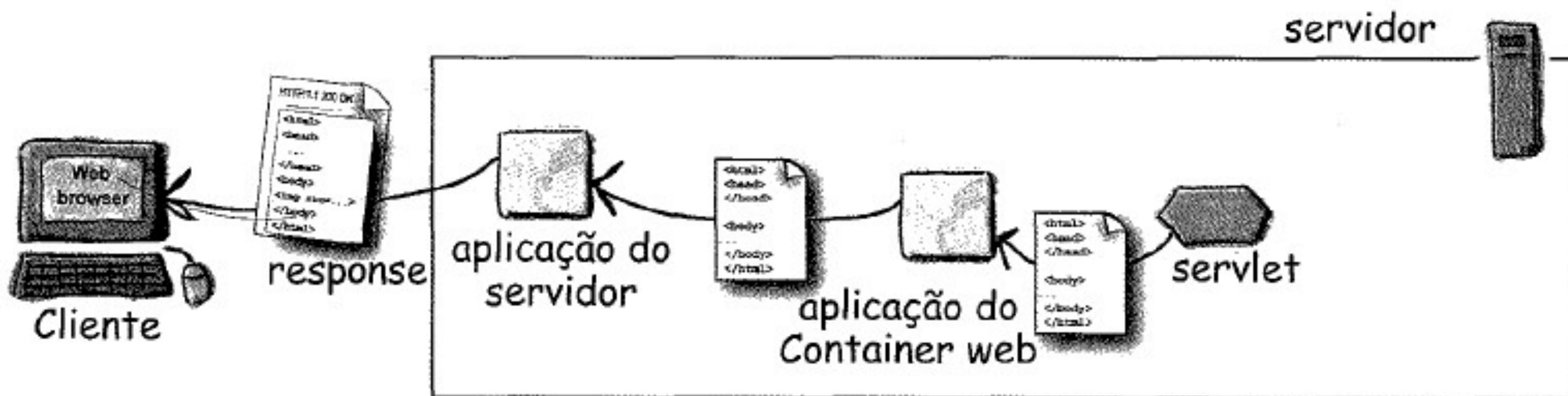
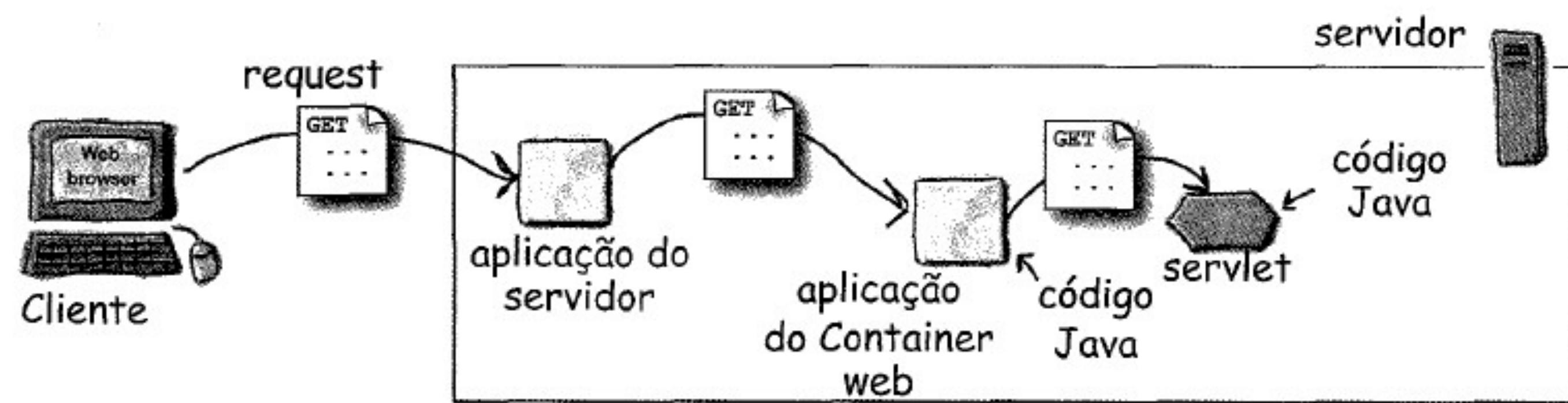
```
public class GerenciadorDeEmpregado{  
  
    public void adicionaEmpregado(String matricula, String nome, float salario,  
        Departamento departamento, Empregado supervisor) throws PersistenciaException{  
        Empregado novoEmp = new Empregado();  
        novoEmp.setMatricula(matricula);  
        novoEmp.setNome(nome);  
        novoEmp.setSalario(salario);  
        novoEmp.setDepartamento(departamento);  
        novoEmp.setSupervisor(supervisor);  
  
        DaoFactoryIF fabrica = DaoFactory.createFactory(DaoFactory.DAO_BD);  
        EmpregadoDaoIF empDao = fabrica.criaEmpregadoDao();  
        empDao.persiste(novoEmp);  
    }  
  
    public void removeEmpregado(String matricula) throws PersistenciaException{  
        DaoFactoryIF fabrica = DaoFactory.createFactory();  
        EmpregadoDaoIF empDao = fabrica.criaEmpregadoDao();  
        empDao.exlcui(matricula);  
    }  
}
```

Conclusão

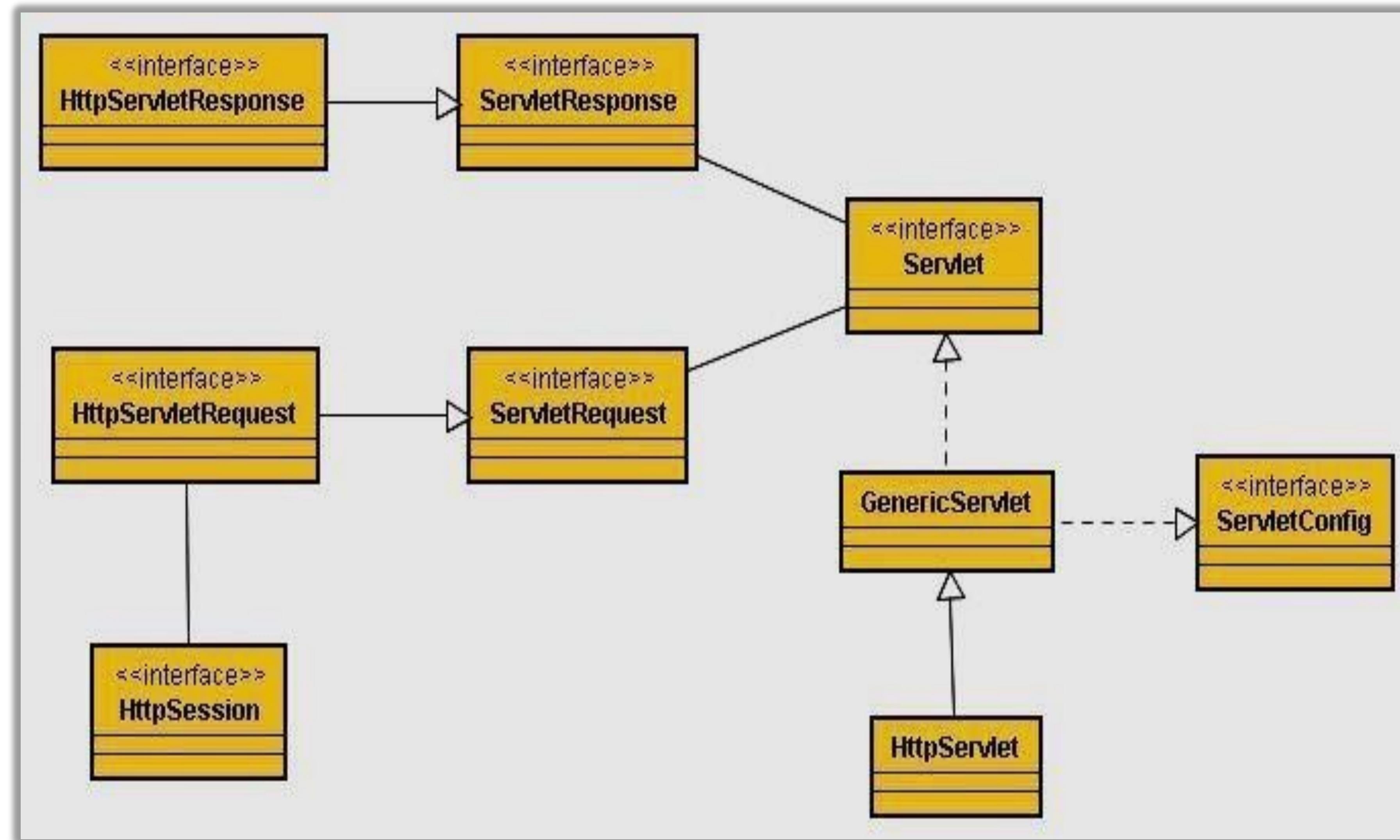
- ▶ O padrão DAO:
 - ▶ Provê transparência para o processo de acesso aos dados;
 - ▶ Permite alterar facilmente a implementação da fonte de dados do sistema;
 - ▶ Reduz a complexidade da implementação dos objetos de negócio;
 - ▶ Centraliza todo o acesso aos dados numa camada separada;
 - ▶ Adiciona uma camada extra de objetos entre os objetos de negócio e a fonte de dados;
 - ▶ Aumenta a complexidade do projeto;

Introdução a Servlets

O Tomcat

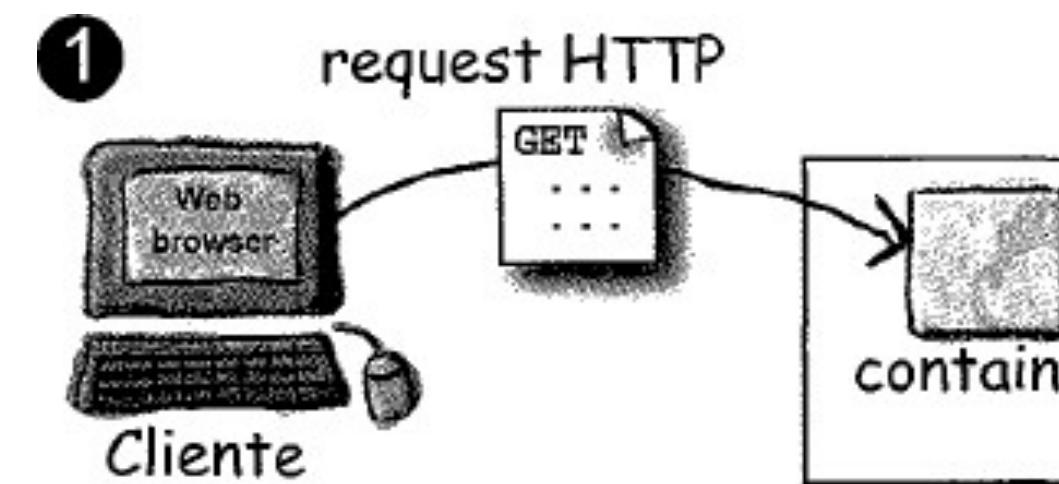


Servlets

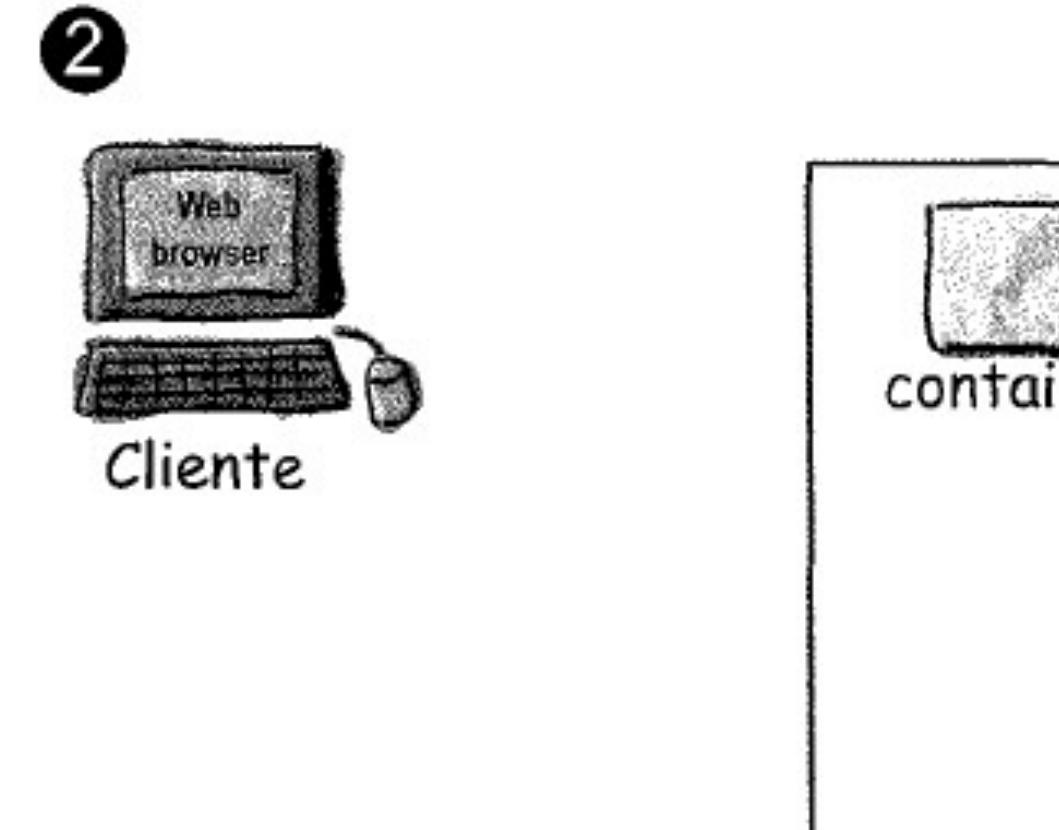


Servlets

- Como container trata uma solicitação



O usuário clica em um link que contém uma URL para um servlet, em vez de uma página estática.

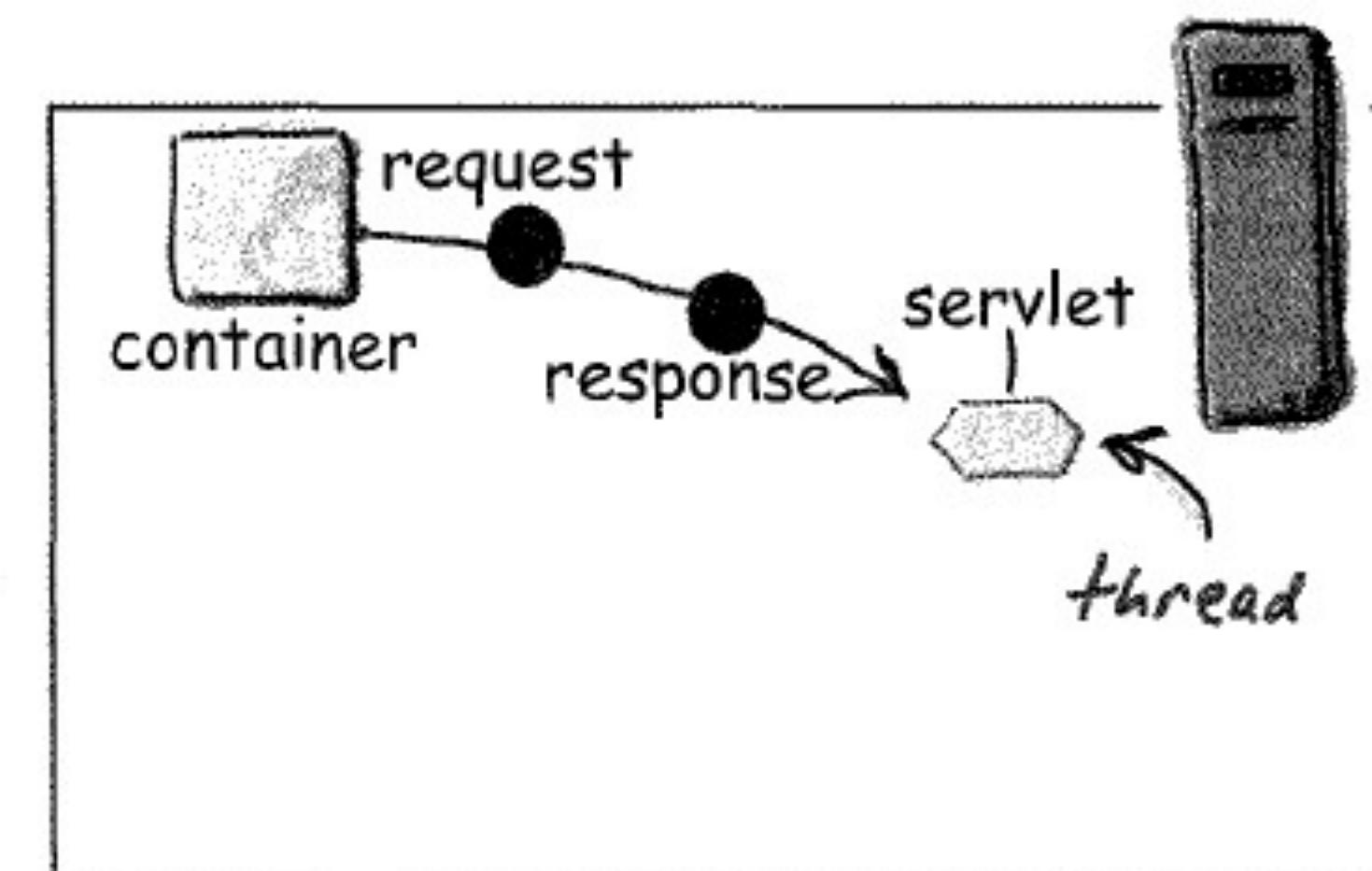


O container “vê” que a request é para um servlet e então ele cria dois objetos:

- 1) HttpServletResponse
- 2) HttpServletRequest

Servlets

- Como container trata uma solicitação

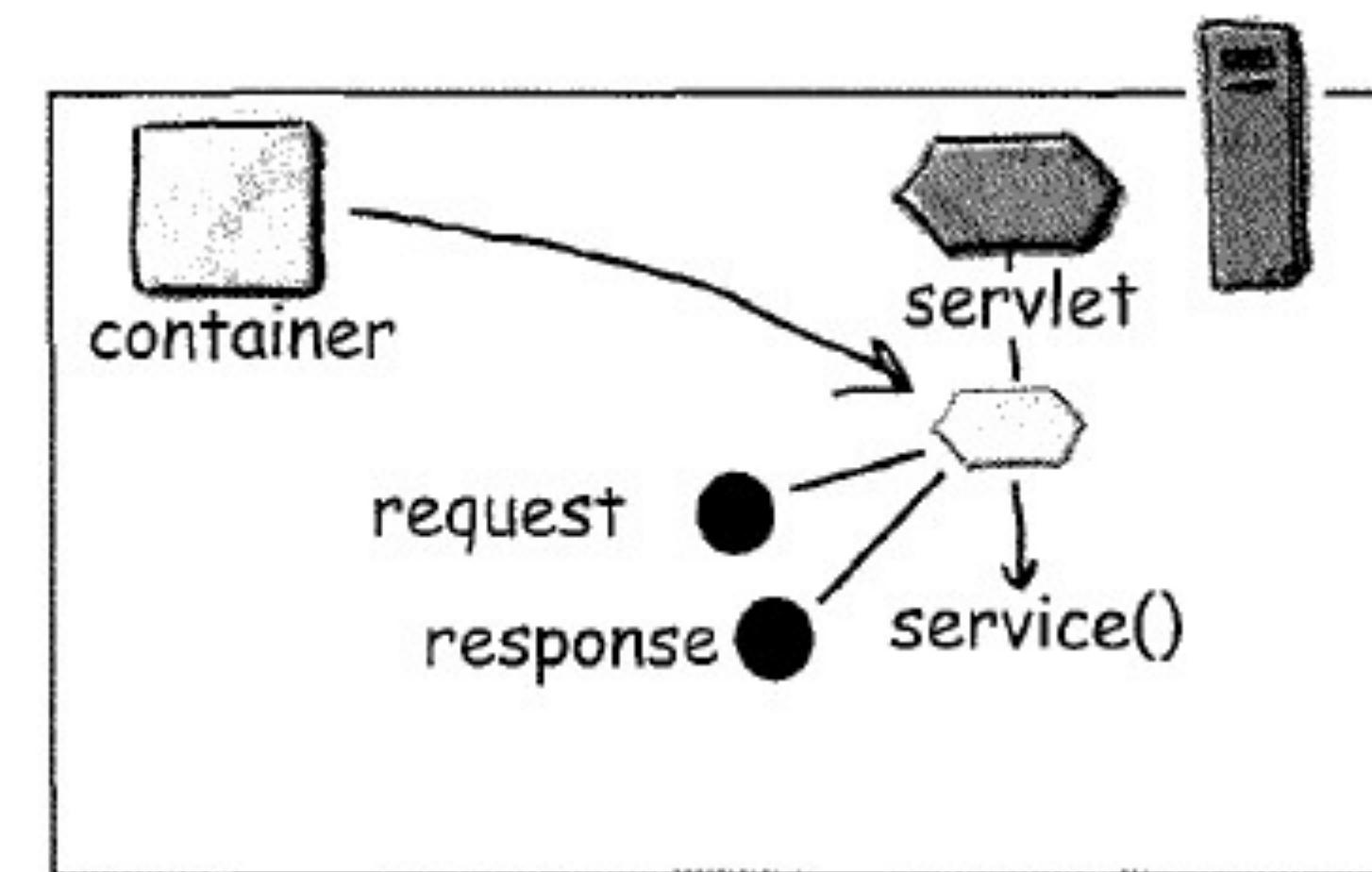


O container encontra o servlet correto baseado na URL da request, cria ou aloca uma thread para essa request, e passa os objetos request e response para a thread do servlet.

Servlets

- Como container trata uma solicitação

4

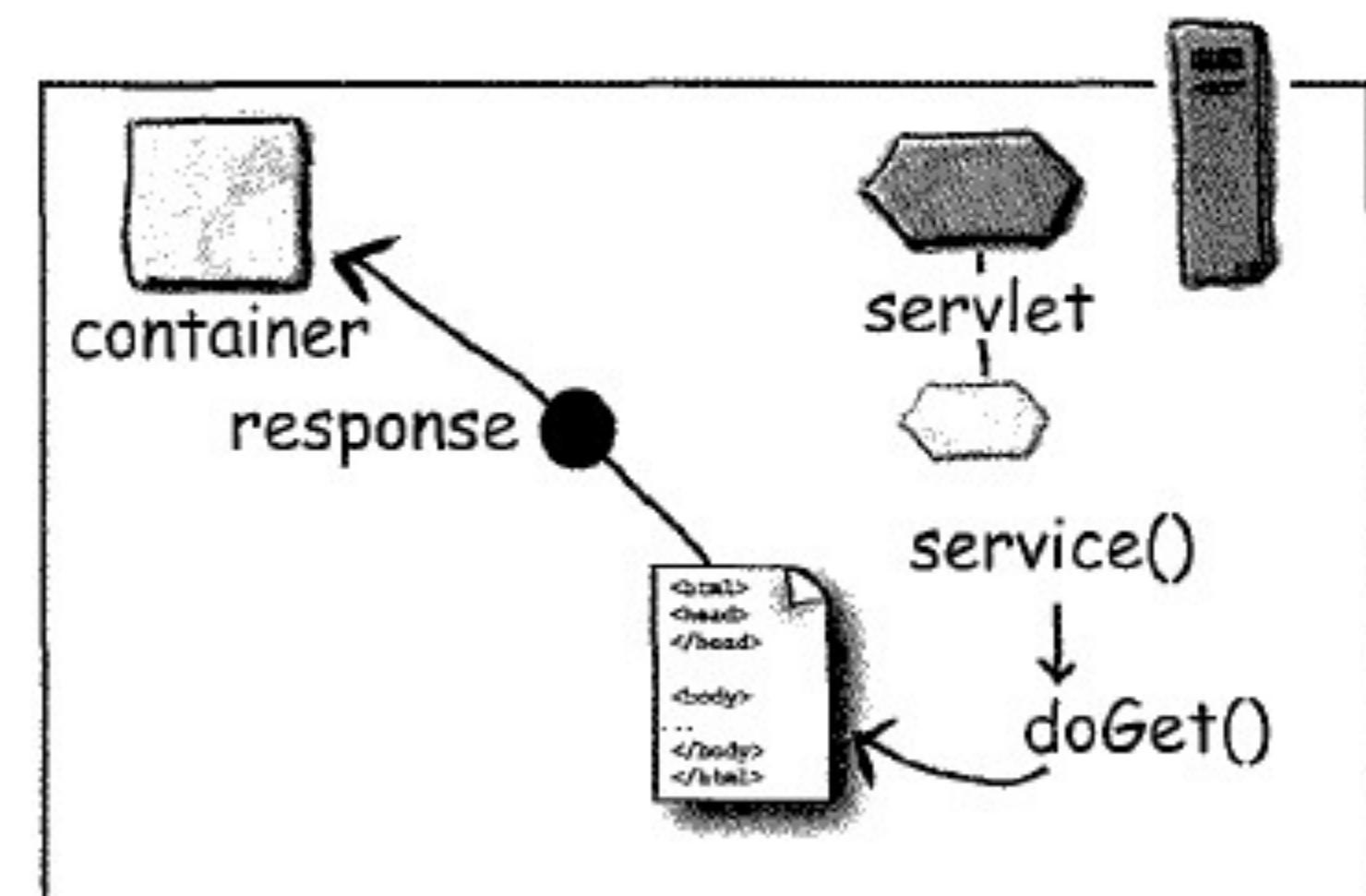


O container chama o método `service()` do servlet. Dependendo do tipo de request, o método `service()` chama ou o método `doGet()`, ou o método `doPost()`.

Para este exemplo, consideraremos que a request foi um HTTP GET.

Servlets

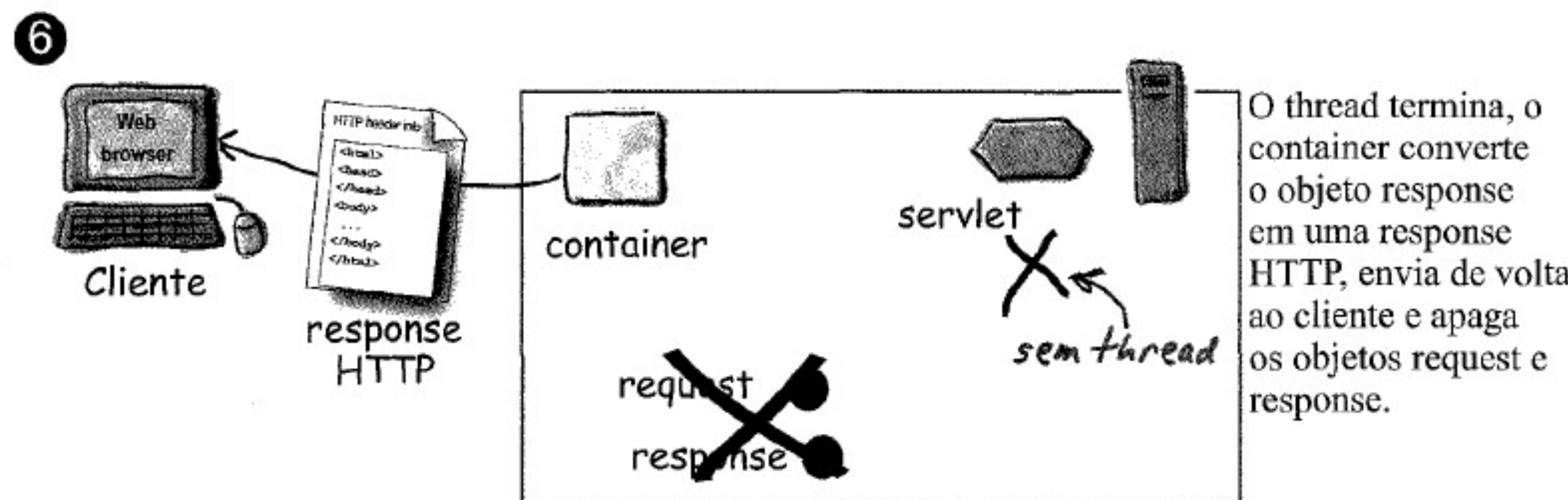
- Como container trata uma solicitação



O método doGet gera uma página dinâmica e a insere no objeto response. Lembre-se, o container ainda tem uma referência do objeto response!

Servlets

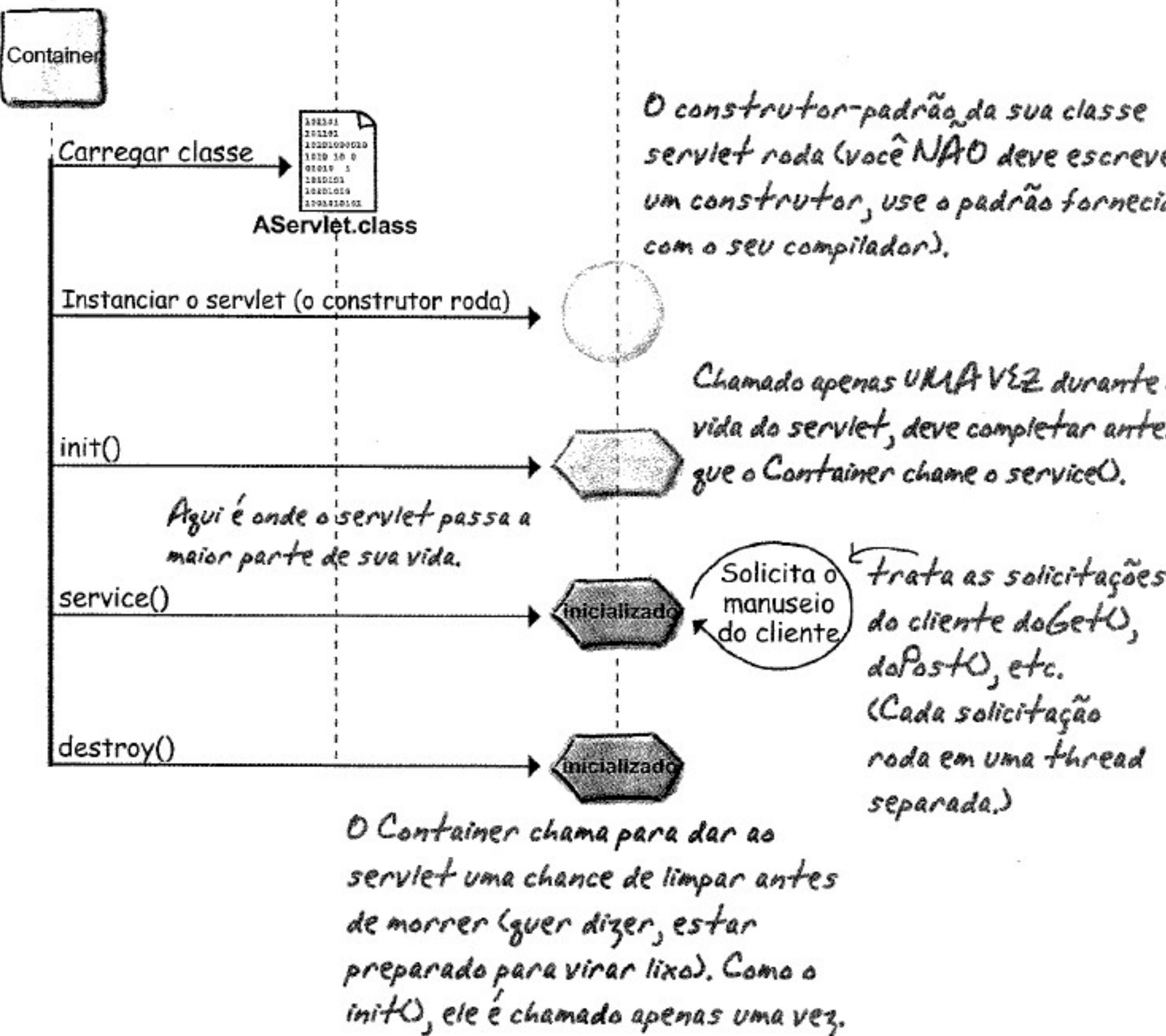
- Como container trata uma solicitação



O Ciclo de Vida de um Servlet

- Etapas na execução de um servlet:
 - Inicialização do servlet;
 - Solicitação HTTP do cliente;
 - Geração da requisição e da resposta;
 - Alocação da thread;
 - Geração da resposta;
 - Resposta ao cliente;
 - Finalização do servlet;

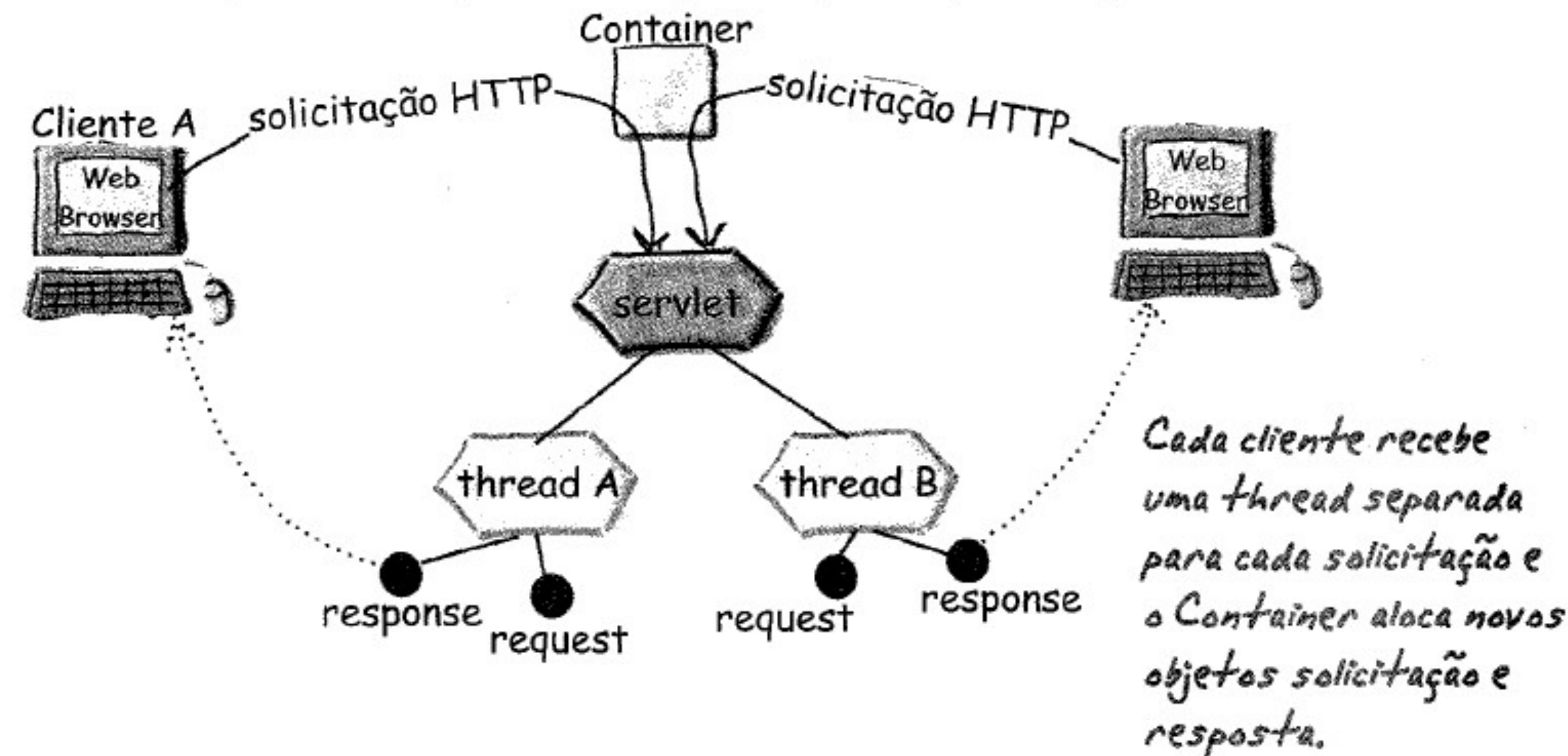
Container web **Classe servlet** **Objeto servlet**



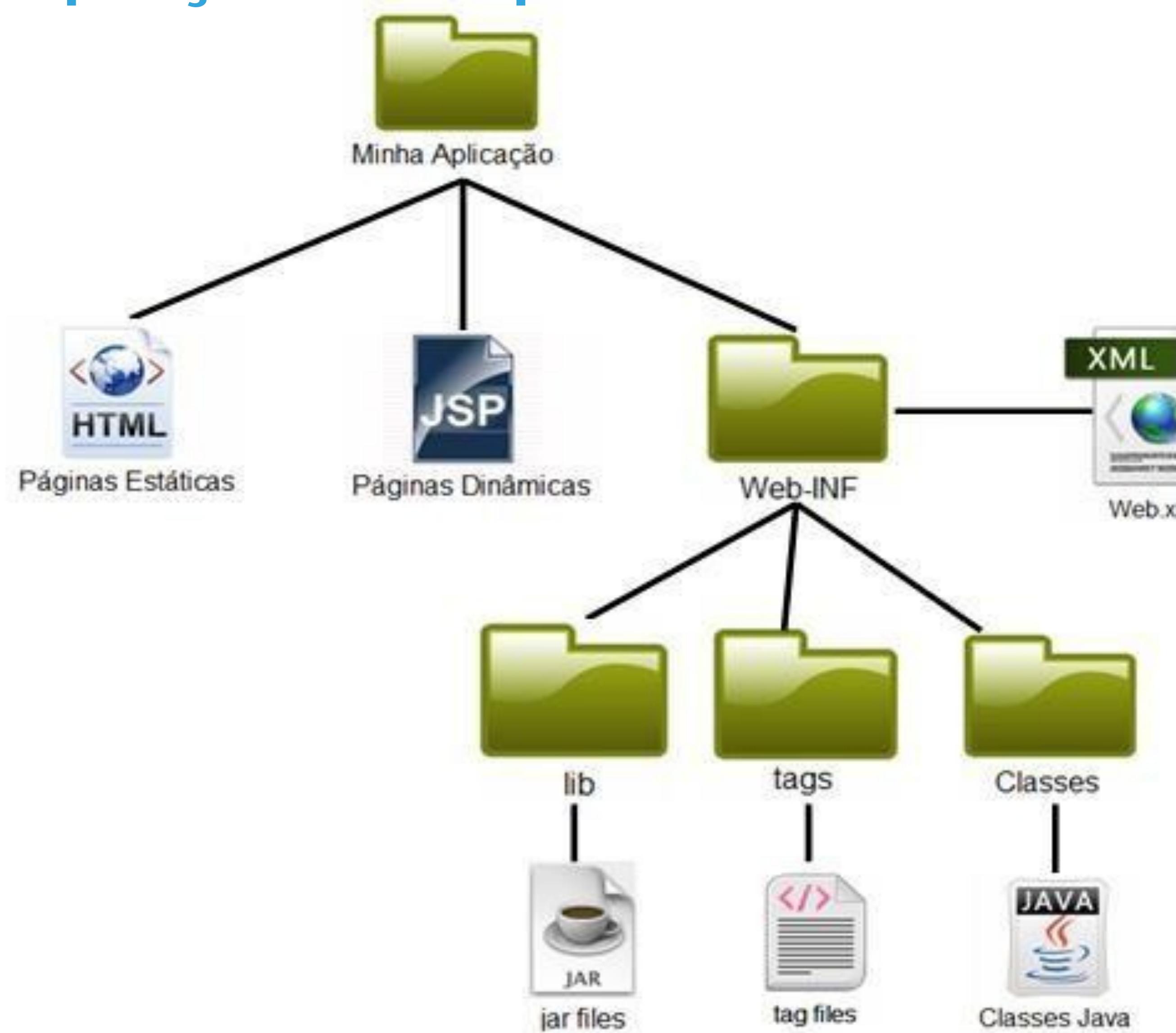
O Ciclo de Vida de um Servlet

O Container roda várias *threads* para processar as várias solicitações para um único servlet.

E cada solicitação do cliente gera um novo par de objetos request e response..



Criando uma Aplicação Web no Apache Tomcat



Tomcat



Servlet - Exemplo

```
@WebServlet("/mostraHeaders")
public class MostraHeadersServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        Enumeration<String> headers = request.getHeaderNames();
        out.println("<html><head><title> Cabeçalho da requisição </title></head>");
        out.println("<body>");
        while(headers.hasMoreElements()) {
            String headerName = headers.nextElement();
            String headerValue = request.getHeader(headerName);
            out.println("<p> " +
                "<h1> <b>" +headerName+ "</b> " +headerValue+ " </h1></p>");
        }
        out.println("</body></html>");
    }

}
```

Servlet - Exemplo Utilizando Outros Tipos de Conteúdo

```
@WebServlet("/retornaXml")
public class RetornaXmlServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        response.setContentType("application/xml");
        PrintWriter out = response.getWriter();
        Enumeration<String> headers = request.getHeaderNames();
        out.println("<xml>");
        while(headers.hasMoreElements()){
            String headerName = headers.nextElement();
            String headerValue = request.getHeader(headerName);
            out.println("<header>");
            out.println("  <headerName>" +headerName+"</headerName> ");
            out.println("  <headerValue>" +headerValue+"</headerValue> ");
            out.println("</header>");

        }
        out.println("</xml>");

    }
}
```

127.0.0.1:8080/psd/retornaXML

127.0.0.1:8080/psd/retornaXML

This XML file does not appear to have any style information associated with it. The document tree is shown below.

Resultado gerado pelo servlet

```
<?xml>
  <header>
    <headerName>host</headerName>
    <headerValue>127.0.0.1:8080</headerValue>
  </header>
  <header>
    <headerName>connection</headerName>
    <headerValue>keep-alive</headerValue>
  </header>
  <header>
    <headerName>accept</headerName>
    <headerValue>
      text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
    </headerValue>
  </header>
  <header>
    <headerName>user-agent</headerName>
    <headerValue>
      Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
    </headerValue>
  </header>
  <header>
    <headerName>referer</headerName>
    <headerValue>http://127.0.0.1:8080/psd/</headerValue>
  </header>
  <header>
    <headerName>accept-encoding</headerName>
    <headerValue>gzip,deflate,sdch</headerValue>
  </header>
  <header>
    <headerName>accept-language</headerName>
    <headerValue>pt-BR,pt;q=0.8,en-US;q=0.6,en;q=0.4</headerValue>
  </header>
</xml>
```

Fazendo Uploads de Arquivos

- A versão atual da API nos permite fazer uploads de arquivos enviados na requisição;
- O servlet responsável pelo upload deve ter a anotação `@MultipartConfig`;
- A requisição deve ser codificada como um multipart/form-data;

Fazendo Uploads de Arquivos

- A anotação `MultipartConfig` pode ter alguns atributos de configuração:
 - `fileSizeThreshold`;
 - `maxFileSize`;
 - `maxRequestSize`
 - `location`;

Fazendo Uploads de Arquivos

- Dois métodos da requisição permitem ao servlet manipular os arquivos enviados:
 - `Collection<Part> getParts();`
 - `Part getPart(String partName);`

Fazendo Uploads de Arquivos

- Alguns métodos definidos na interface Part:
 - `delete();`
 - `write(String fileName);`
 - `getName();`
 - `getSize();`
 - `getContentType();`
 - `getInputStream();`
 - `getSubmittedFileName();`

Fazendo Uploads de Arquivos

- Definindo a requisição como um multipart/form-data:

```
<form action="uploader" method="post" enctype="multipart/form-data" name="form1">

    <input type="file" name="path" id="path">
    <input type="submit" value="Enviar">

</form>
```

Declaração de um servlet para a realização de uploads de arquivos

```
@WebServlet("/uploader")
@MultipartConfig(
    fileSizeThreshold = 1024 * 1024 * 1, // 1 MB
    maxFileSize      = 1024 * 1024 * 10, // 10 MB
    maxRequestSize   = 1024 * 1024 * 15, // 15 MB
    location         = "D:/Uploads"
)
public class FileUploader extends HttpServlet{

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

    protected void doPost
        (HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {...}
}
```

Implementação do método doPost

```
protected void doPost
    (HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException{
    PrintWriter out = response.getWriter();
    out.println("<html><head><title> Upload file example </title></head>");
    out.println("<body>");
    Part path = request.getPart("path");
    out.println("<p> <b> Name: </b> "+path.getName()+"</p>");
    out.println("<p> <b> Content Type: </b> "+path.getContentType()+"</p>");
    out.println("<p> <b> Size: </b> "+path.getSize()+" bytes</p>");
    InputStreamReader isr = new InputStreamReader(path.getInputStream());
    BufferedReader bufReader = new BufferedReader(isr);
    out.println("<p><b>Content</b></p><br>");
    String line = bufReader.readLine();
    while(line!=null){
        out.println("<p>"+line+"</p>");
        line = bufReader.readLine();
    }
    out.println("</body></html>");
}
```

Servlet - Exemplo De Logs

Exemplo de servlet que realiza logs

```
@WebServlet("/logger")
public class LoggerServlet extends HttpServlet{

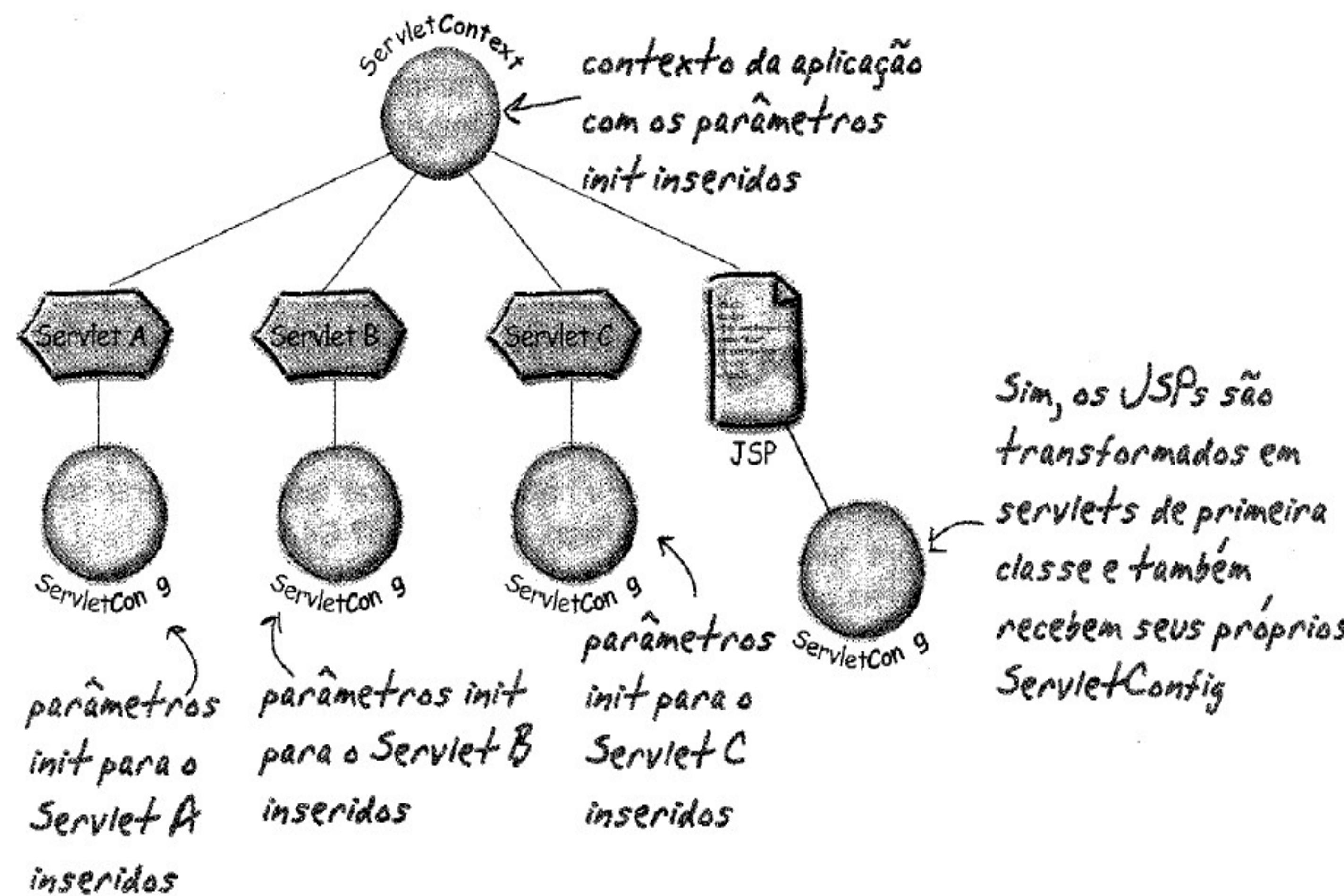
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        String userName = request.getParameter("name");
        this.log("LoggerServlet UserName is "+userName);
        out.println("<html><head><title> Exemplo de logs </title></head>");
        out.println("<body>");
        out.println("<p> Log efetuado com sucesso </p>");
        out.println("</body>");
        out.println("</html>");
    }

}
```

Gerenciamento de Aplicações

O Contexto da Aplicação



O Container lê o DD e cria um par de String nome/valor para cada <context-param>.

O Container cria uma nova instância do ServletContext.

O Container dá ao ServletContext uma referência a cada par nome/valor dos parâmetros init do contexto.

Cada servlet e JSP distribuído como parte de uma única aplicação tem acesso àquele mesmo ServletContext.

A Interface ServletContext

Obtém os parâmetros init e obtém/especifica atributos

Obtém informações sobre o servidor/container.

Falaremos adiante neste capítulo sobre o RequestDispatcher.

<code><<interface>></code>
<code>ServletContext</code>
<code>getInitParameter(String)</code>
<code>getInitParameterNames()</code>
<code>getAttribute(String)</code>
<code>getAttributeNames()</code>
<code>setAttribute(String)</code>
<code>removeAttribute(String)</code>
<hr/>
<code>getMajorVersion()</code>
<code>getServerInfo()</code>
<hr/>
<code>getRealPath(String)</code>
<code>getResourceAsStream(String)</code>
<code>getRequestDispatcher(String)</code>
<hr/>
<code>log(String)</code>
<code>// mais métodos</code>

Obtém os parâmetros init e obtém/especifica atributos

← Escreve no arquivo de log do servidor (específico por fabricante) ou no System.out.

Parâmetros de Inicialização

- Métodos da interface ServletConfig:

javax.servlet.ServletConfig

<<interface>>

ServletConfig

getInitParameter(String)

Enumeration getInitParameterNames()

getServletContext()

getServletName()



*A maioria das pessoas
nunca usa este método*

Parâmetros de Inicialização

- Métodos da interface ServletContext para a manipulação de parâmetros:

<<interface>> ServletContext	
Obtém os parâmetros init e obtém/especifica atributos	<code>getInitParameter(String)</code> <code>getInitParameterNames()</code>
Obtém informações sobre o servidor/container.	<code>getAttribute(String)</code> <code>getAttributeNames()</code> <code>setAttribute(String)</code> <code>removeAttribute(String)</code>
Faremos adiante neste capítulo sobre o RequestDispatcher.	<code>getMajorVersion()</code> <code>getServerInfo()</code>
	<code>getRealPath(String)</code> <code>getResourceAsStream(String)</code> <code>getRequestDispatcher(String)</code>
	<code>log(String)</code> // mais métodos

← Escreve no arquivo de log do servidor (específico por fabricante) ou no System.out.

Compartilhando Informações

- Parâmetros de inicialização são muito úteis, mas possuem algumas limitações:
 - Só podem receber valores do tipo String;
 - Os seus valores precisam ser conhecidos no momento em que a aplicação é inicializada;
 - Os seus valores não podem ser alterados em tempo de execução;

Compartilhando Informações

- Existem situações nas quais precisamos inicializar um objeto que será usado pelos componentes da aplicação;
 - Que podem eventualmente alterar o estado destes objetos;
- Nestes casos, parâmetros de inicialização não resolvem o problema;

Compartilhando Informações

- Tampouco, não podemos delegar esta tarefa a um servlet;
 - Como garantir que este *servlet* seria executado antes de todos os outros?

Compartilhando Informações

Precisamos de um objeto à parte que possa:

- Ser notificado quando o contexto é inicializado (a aplicação está sendo distribuída).
 - Conseguir os parâmetros init do contexto através do ServletContext.
 - Usar o nome de lookup do parâmetro init para fazer uma conexão com o banco de dados.
 - Armazenar a conexão com o banco de dados como um atributo, para que todas as partes da aplicação possam acessá-la.
- Ser notificado quando o contexto é destruído (a aplicação é retirada do ar ou cai).
 - Encerrar a conexão com o banco de dados.

Inicializando Objetos

- O **ServletContextListener**:
 - Os objetos que implementam esta interface são notificados sobre eventos ocorridos na aplicação;
 - Usamos este tipo de objeto para programar ações que devem ser executadas no momento de inicialização ou finalização da aplicação;

Inicializando Objetos

- Cadastrando um listener no descriptor da aplicação:

```
<listener>
    <listener-class>servlets.ApplicationLogger</listener-class>
</listener>
```

Inicializando Objetos

- Listeners também podem ser adicionados à aplicação através de uma anotação;
- Para isso, usamos a anotação `@WebListener`;

Inicializando Objetos

- O ServletContextListener:

Uma classe ServletContextListener:

```
import javax.servlet.*;  
  
public class MyServletContextListener implements ServletContextListener {  
  
    public void contextInitialized(ServletContextEvent event) {  
        //código para inicializar a conexão com o banco de dados  
        //e armazená-la como um atributo do contexto  
    }  
  
    public void contextDestroyed(ServletContextEvent event) {  
        //código para encerrar a conexão com o banco de dados  
    }  
}
```

O ServletContextListener
está no pacote javax.servlet

O listener do contexto
é simples: implementa o
ServletContextListener

Estas são as duas notificações
que você recebe. Ambas exibem
um ServletContextEvent

```
package com.example;
```

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

```
public class MyServletContextListener implements ServletContextListener {
```

```
    public void contextInitialized(ServletContextEvent event) {
```

```
        ServletContext sc = event.getServletContext(); ← Solicita o ServletContext  
        ao evento.
```

```
        String dogBreed = sc.getInitParameter("breed");
```

```
        Dog d = new Dog(dogBreed); ← Cria um  
        novo Dog
```

```
        sc.setAttribute("dog", d); ← Usa o contexto para obter o  
        parâmetro init
```

```
}
```

Uso o contexto para especificar um atributo
(um par nome/objeto) Dog. Agora, outros
trechos da aplicação serão capazes de receber
o valor do atributo (o Dog)

```
    public void contextDestroyed(ServletContextEvent event) {
```

```
        // nada a fazer aqui
```

```
}
```



Não precisamos de nada aqui. O Dog não precisa
ser limpo... quando o contexto termina, significa
que toda a aplicação vai finalizar, incluindo o Dog

Inicializando Objetos

```
package com.example;

public class Dog {
    private String breed;

    public Dog(String breed) {
        this.breed = breed;
    }

    public String getBreed() {
        return breed;
    }
}
```

Nada de especial aqui. Só uma classe Java simples

(Usaremos os parâmetros init do contexto como argumento para o construtor Dog.)

Nosso servlet receberá o Dog do contexto (o Dog, que o listener configura como atributo), chamará o método `getBreed()` do Dog e copiará a raça na resposta, para que possamos vê-la no browser

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

public class ListenerTester extends HttpServlet {

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

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

        out.println("test context attributes set by listener<br>");

        out.println("<br>");

        Dog dog = (Dog) getServletContext().getAttribute("dog");
    }
}
```

↑
Se algo não funcionar, é AQUI que
descobriremos... vamos receber um enorme
NullPointerException se tentarmos chamar
o getBreed() e não houver nenhum Dog

Nada de especial até aqui... apenas um
servlet comum

Aqui obtemos o
Dog através do
ServletContext. Se
o listener funcionou,
o Dog vai estar ali
ANTES que o método
service seja chamado
pela primeira vez

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <servlet>
        <servlet-name>ListenerTester</servlet-name>
        <servlet-class>com.example.ListenerTester</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>ListenerTester</servlet-name>
        <url-pattern>/ListenTest.do</url-pattern>
    </servlet-mapping>

    <context-param>
        <param-name>breed</param-name>
        <param-value>Great Dane</param-value>
    </context-param>

    <listener>
        <listener-class>
            com.example.MyServletContextListener
        </listener-class>
    </listener>
</web-app>
```

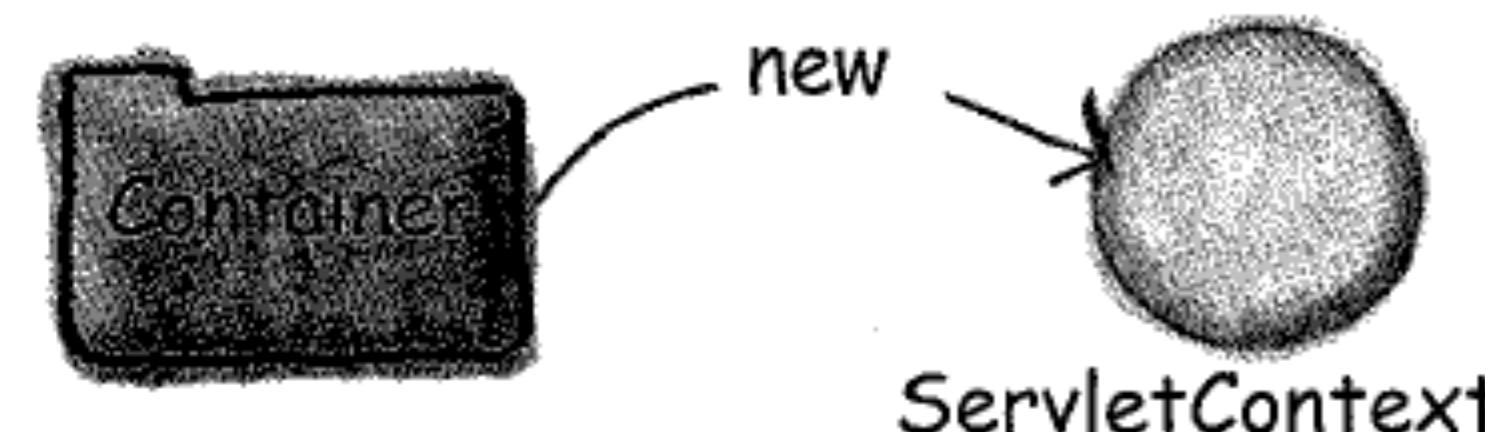
Precisamos de um parâmetro init
do contexto para a aplicação.
O listener precisa dele para
construir o Dog

Registre esta classe como um listener. IMPORTANTE:
elemento <listener> NÃO vai dentro de um elemento <servlet>. Isto não funcionaria, pois o listener do contexto é para um evento ServletContext (que significa para toda a aplicação). A questão toda é inicializar a aplicação ANTES que qualquer servlet seja inicializado.

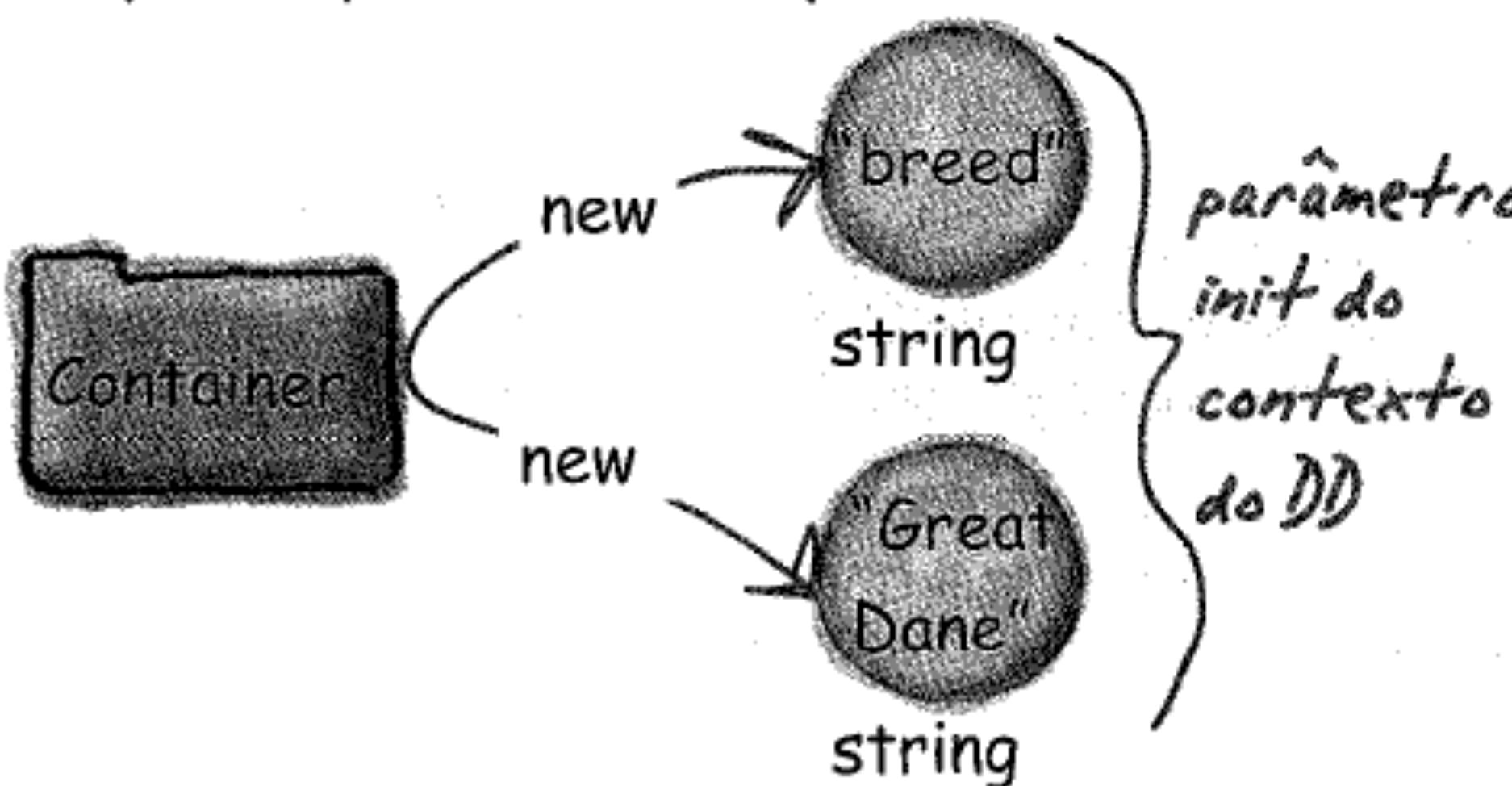
- 1** O Container lê o Deployment Descriptor para esta aplicação, inclusive os elementos <listener> e <context-param>.



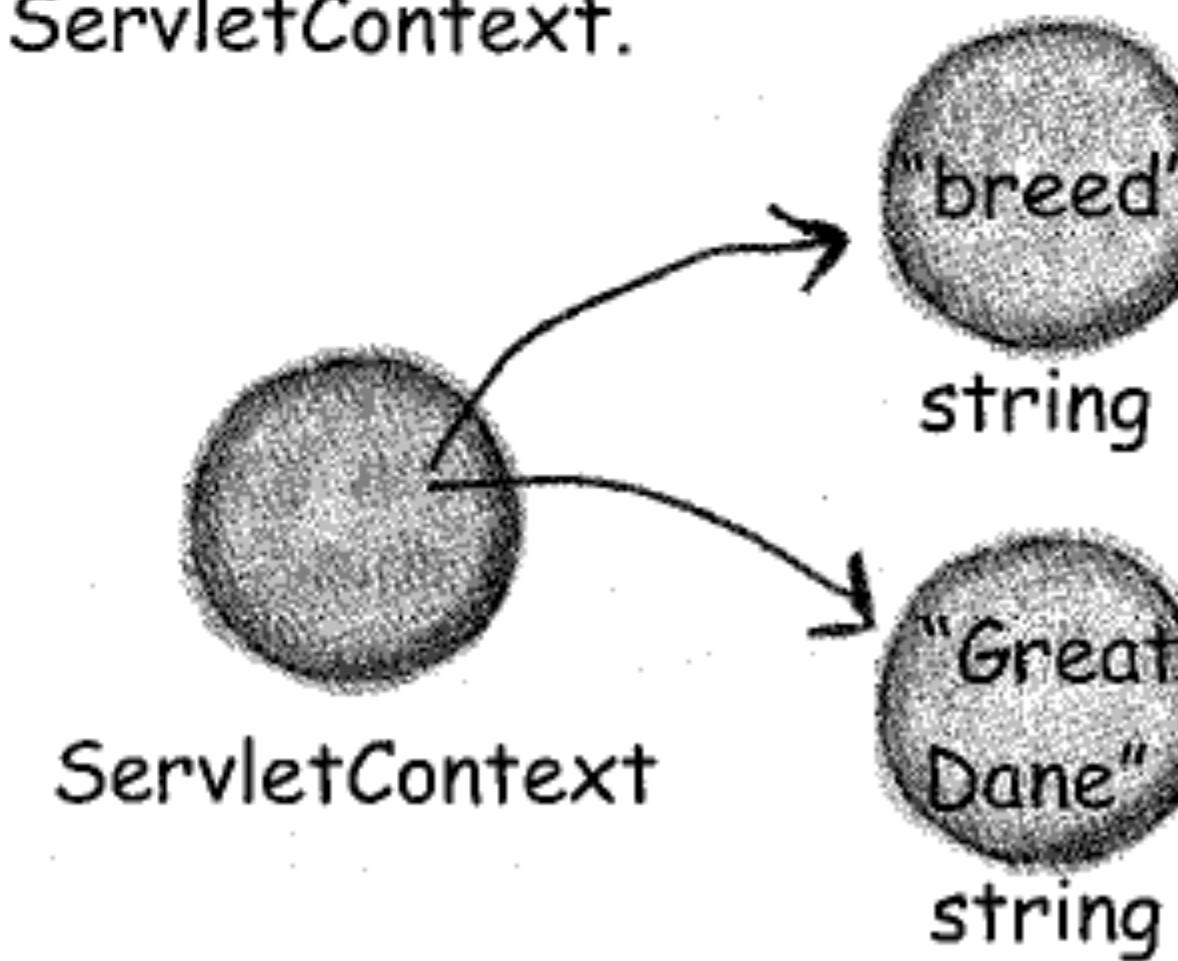
- 2** O Container cria um novo ServletContext, que todas as partes da aplicação compartilharão.



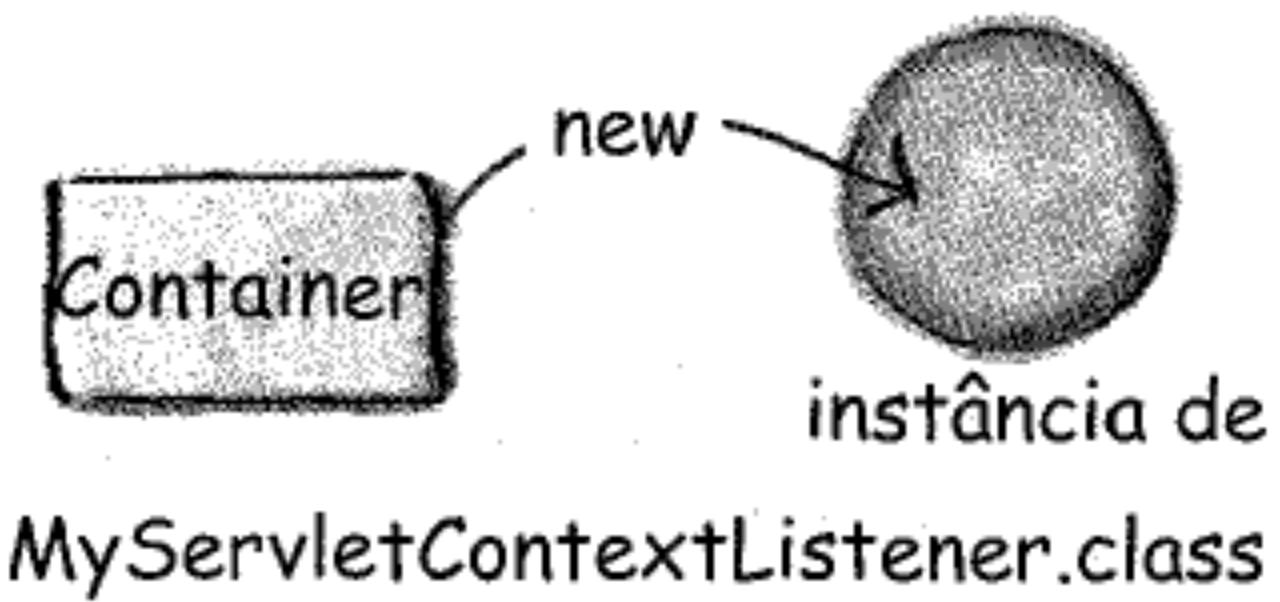
- 3** O Container cria um par de Strings nome/valor para cada parâmetro init do contexto. Suponha que tenhamos apenas um.



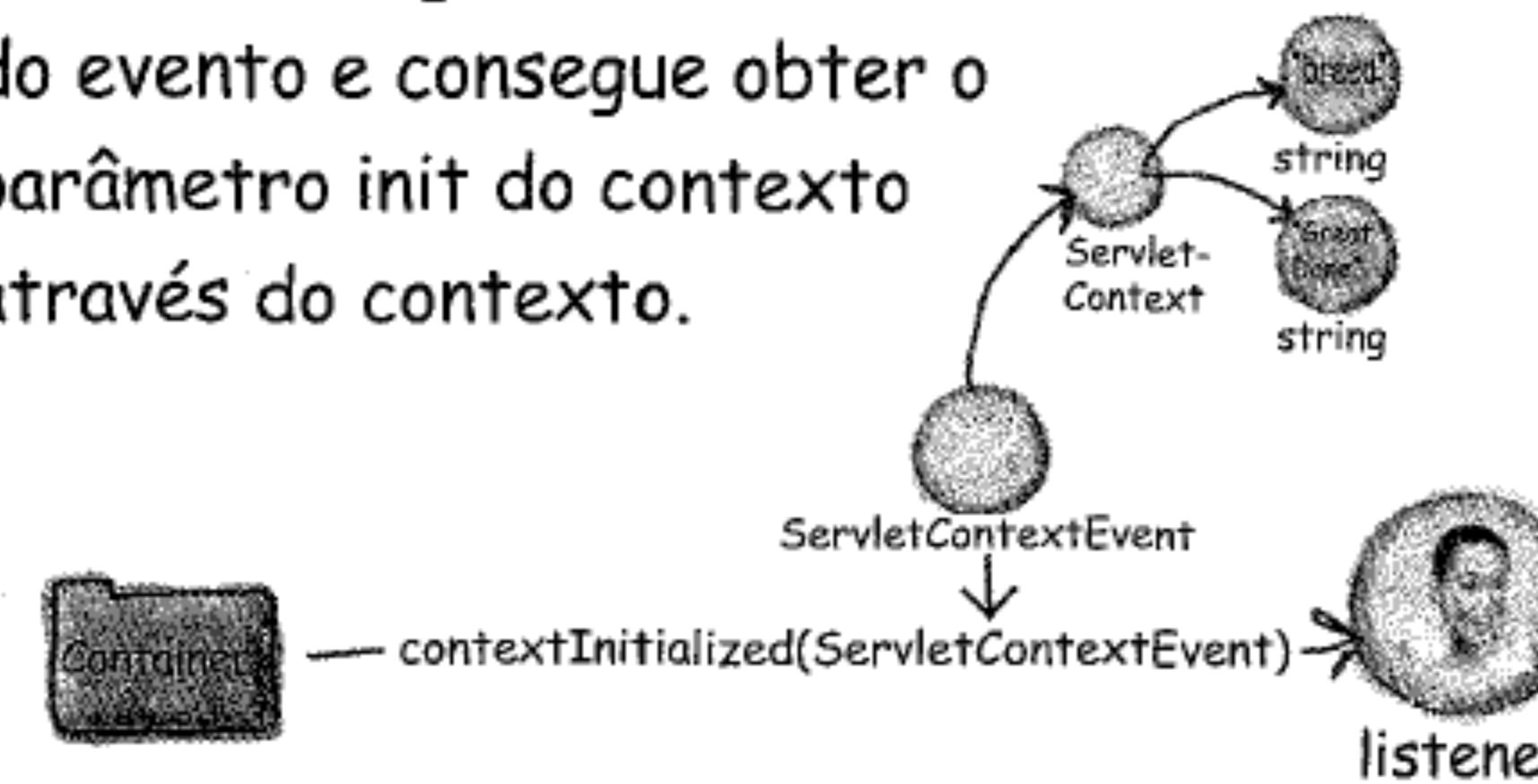
- 4** O Container dá aos parâmetros nome/valor as referências do ServletContext.



- 5 O Container cria uma nova instância da classe MyServletContextListener.



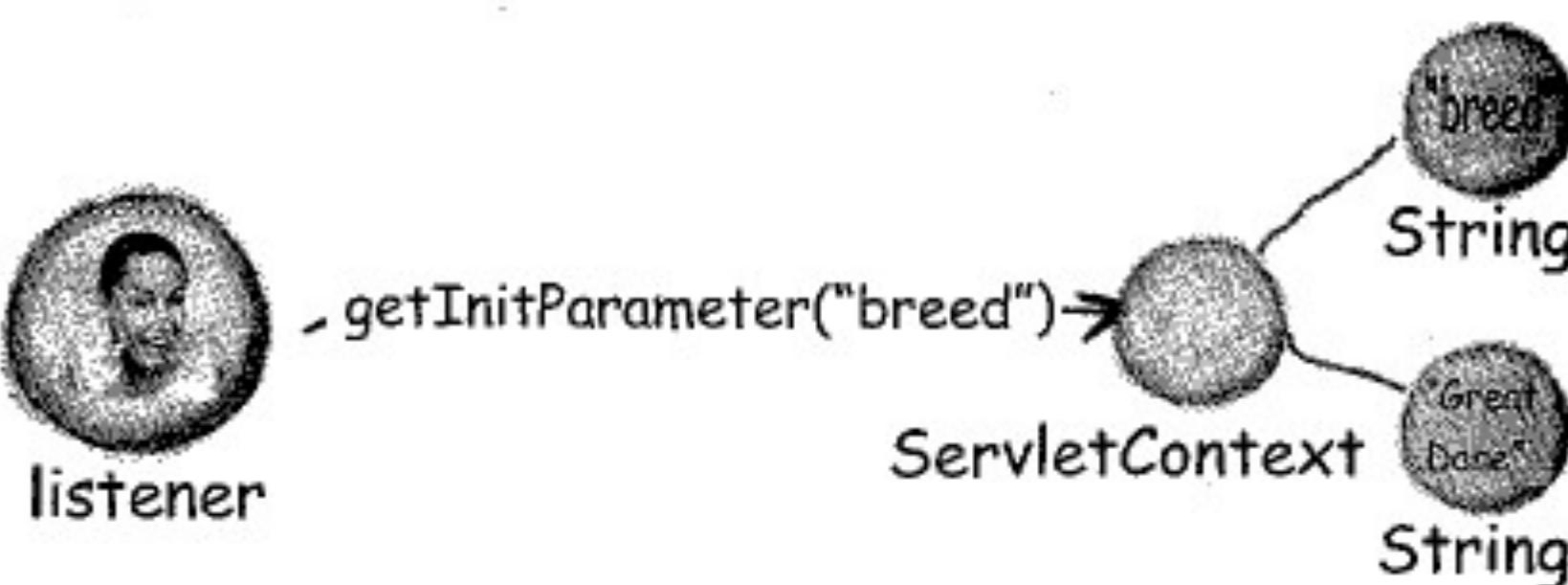
- 6 O Container chama o método contextInitialized() do listener, passando em um novo ServletContextEvent. O objeto de evento tem uma referência para o ServletContext, então o código que trata o evento consegue obter o contexto através do evento e consegue obter o parâmetro init do contexto através do contexto.



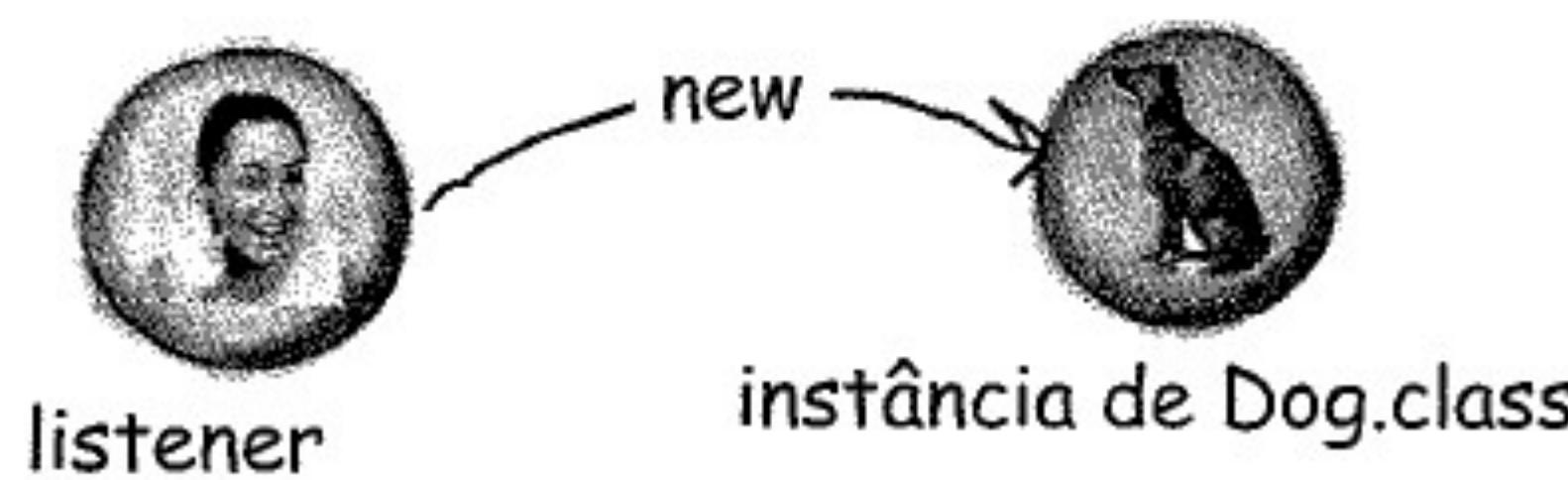
- 7** O listener solicita ao ServletContextEvent uma referência para o ServletContext.



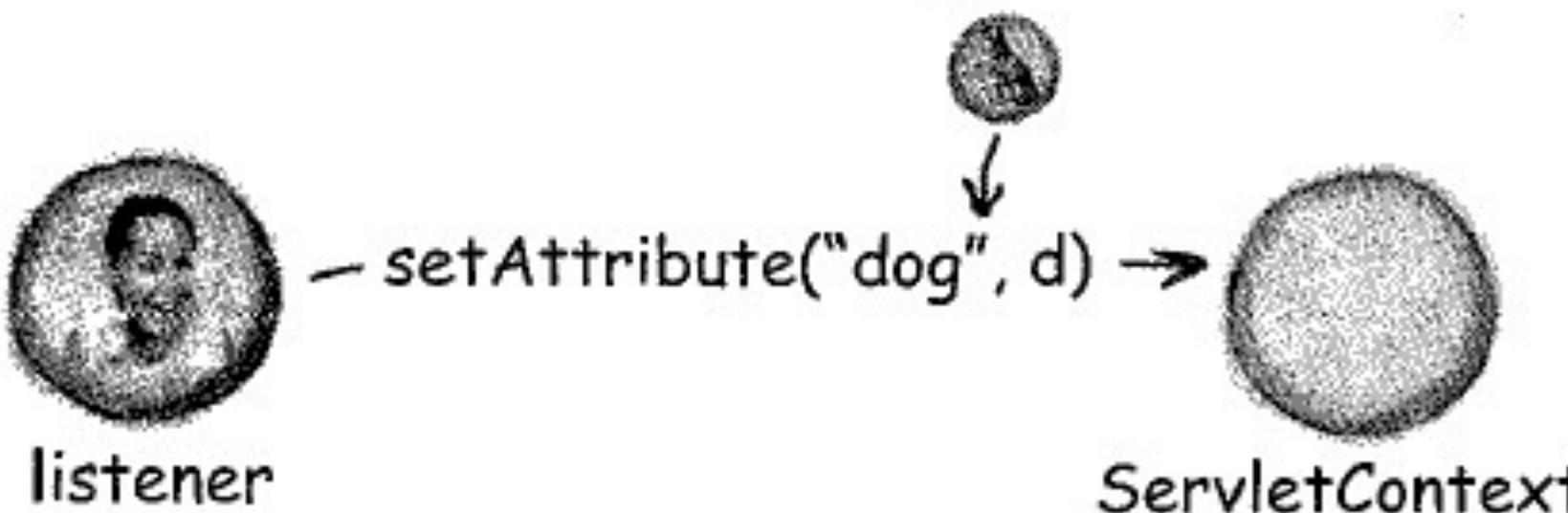
- 8** O listener solicita ao ServletContext o parâmetro init do contexto "breed".



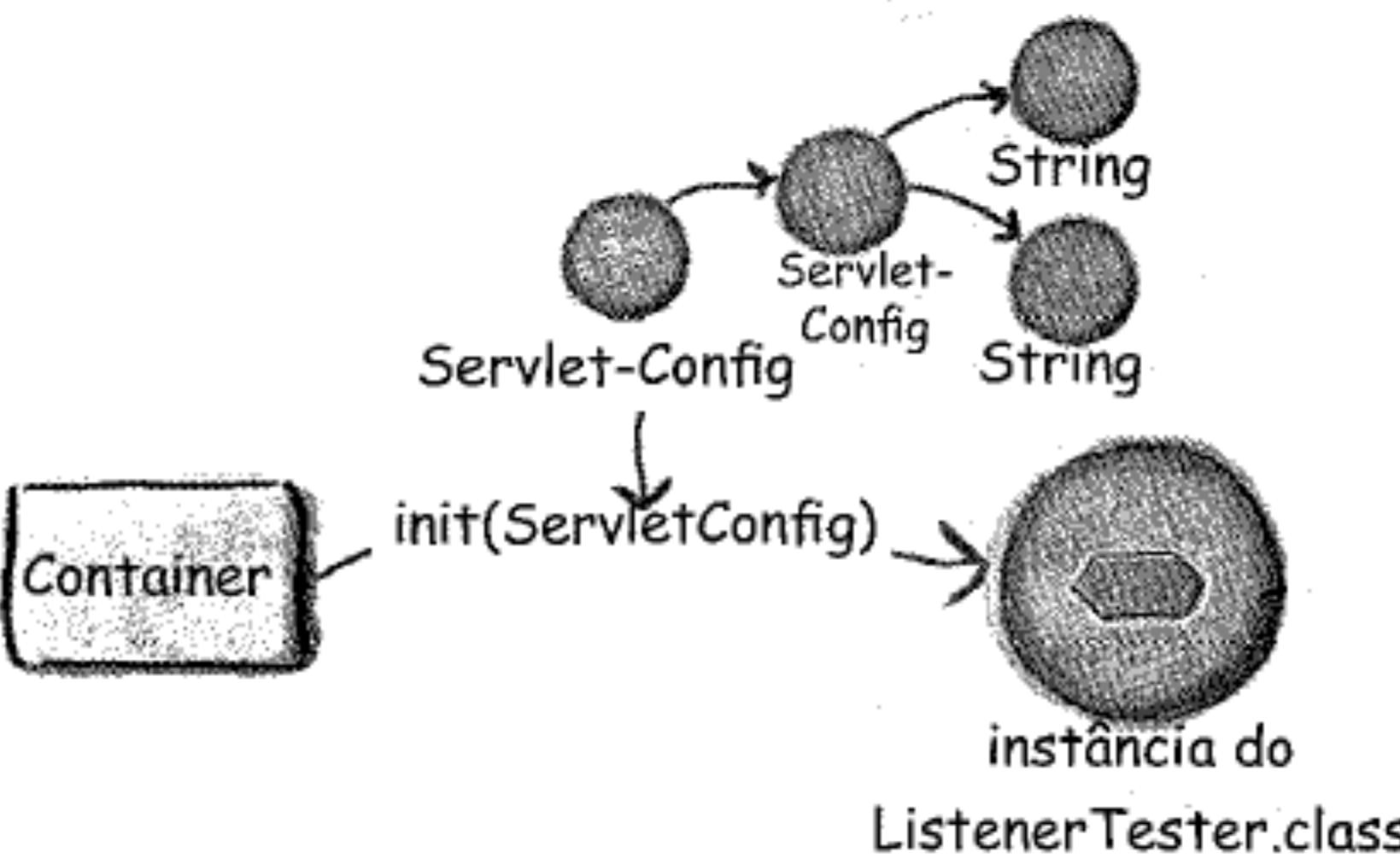
- 9** O listener usa o parâmetro init para construir um novo objeto Dog.



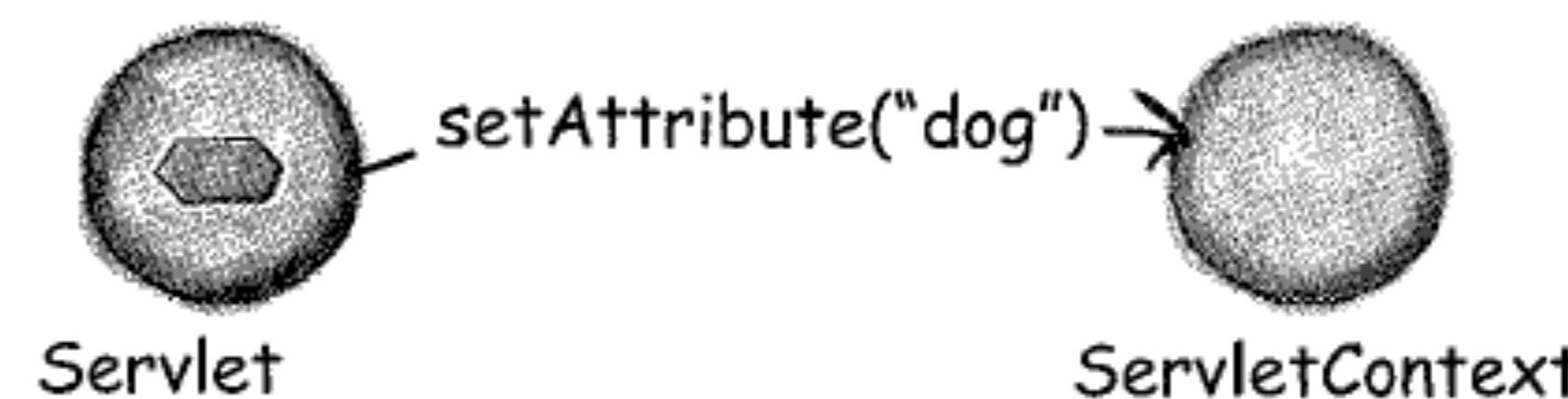
- 10** O listener configura o Dog como um atributo no ServletContext.



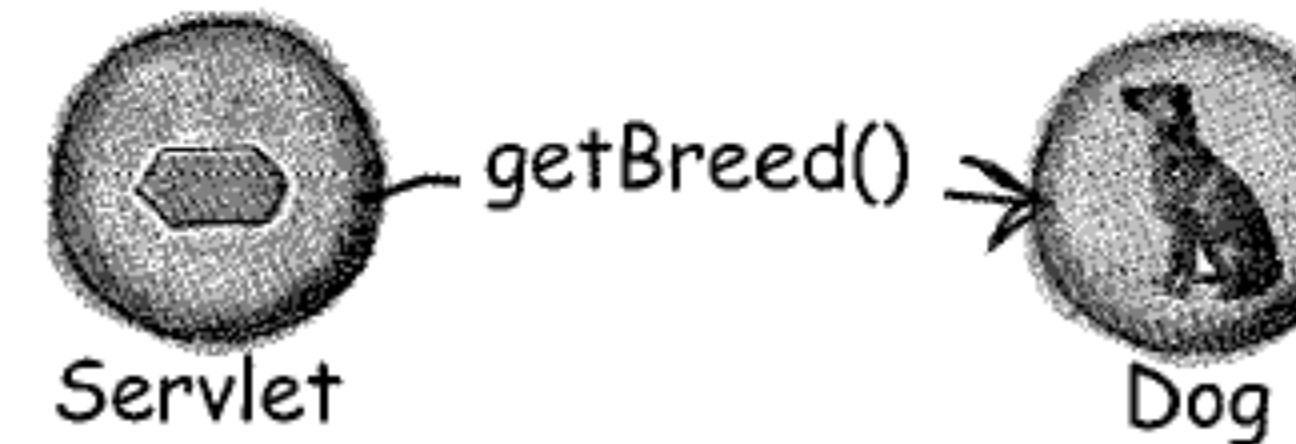
⑪ O Container cria um novo Servlet (isto é, cria um novo ServletConfig com parâmetros init, dá ao ServletConfig uma referência para o ServletContext e chama o método init() do Servlet).



⑫ O servlet recebe uma solicitação e pede ao ServletContext o atributo "dog".



⑬ O servlet chama o getBreed() no Dog (e exibe-o no HttpResponse).



Inicializando Objetos

- Outros tipos de listeners oferecidos pela API de Servlets:
 - `AttributeListener`;
 - `HttpSessionListener`;
 - `ServletRequestListener`;
 - `ServletRequestAttributeListener`;
 - `HttpSessionBindingListener`;
 - `HttpSessionAttributeListener`;
 - `HttpSessionActivationListener`;

Compartilhando Informações



*Quem pode ver este
quadro de avisos?*

*Quem pode receber
e configurar os
atributos?*

Um atributo é como um objeto preso em um quadro de avisos. Alguém o afixou no quadro para que outros possam pegá-lo.

As grandes perguntas são: quem tem acesso ao quadro de avisos e quanto tempo ele permanece lá? Em outras palavras, qual é o escopo do atributo?

Compartilhando Informações

- Todo atributo é identificado através de um nome;
- Todo atributo é armazenado na forma de um objeto;

		Atributos	Parâmetros
Tipos	<p>Application/context Request Session</p> <p><i>Não há nenhum atributo específico para o servlet (basta usar uma variável da instância)</i></p>		<p>Application/parâmetros init do contexto Parâmetros da solicitação Parâmetros init do <u>servlet</u></p> <p><i>Não existem parâmetros da sessão!</i></p>
Método para configuração	setAttribute(nome da String, valor do Objeto)		Você NÃO PODE configurar os parâmetros init da Application e do Servlet – eles são configurados no DD, lembra? (Com os parâmetros da Solicitação, você pode ajustar a query String, mas é diferente.)
Tipo de retorno	Objeto		<p>String</p>  <p><i>Grande diferença!</i></p>
Método para obtenção	<p>ServletContext.getAttribute (String name)</p> <p><i>Não se esqueça de que os atributos devem ser convertidos, já que o tipo de retorno é Objeto</i></p>		ServletContext.getAttribute(String name)

Compartilhando Informações

- Podemos criar atributos com três diferentes escopos:
 - Requisição, sessão e aplicação;
- O escopo do atributo depende do objeto no qual o mesmo é adicionado;

Compartilhando Informações

- Atributos de requisição:



Acessível apenas para aqueles com acesso a um *ServletRequest* específico

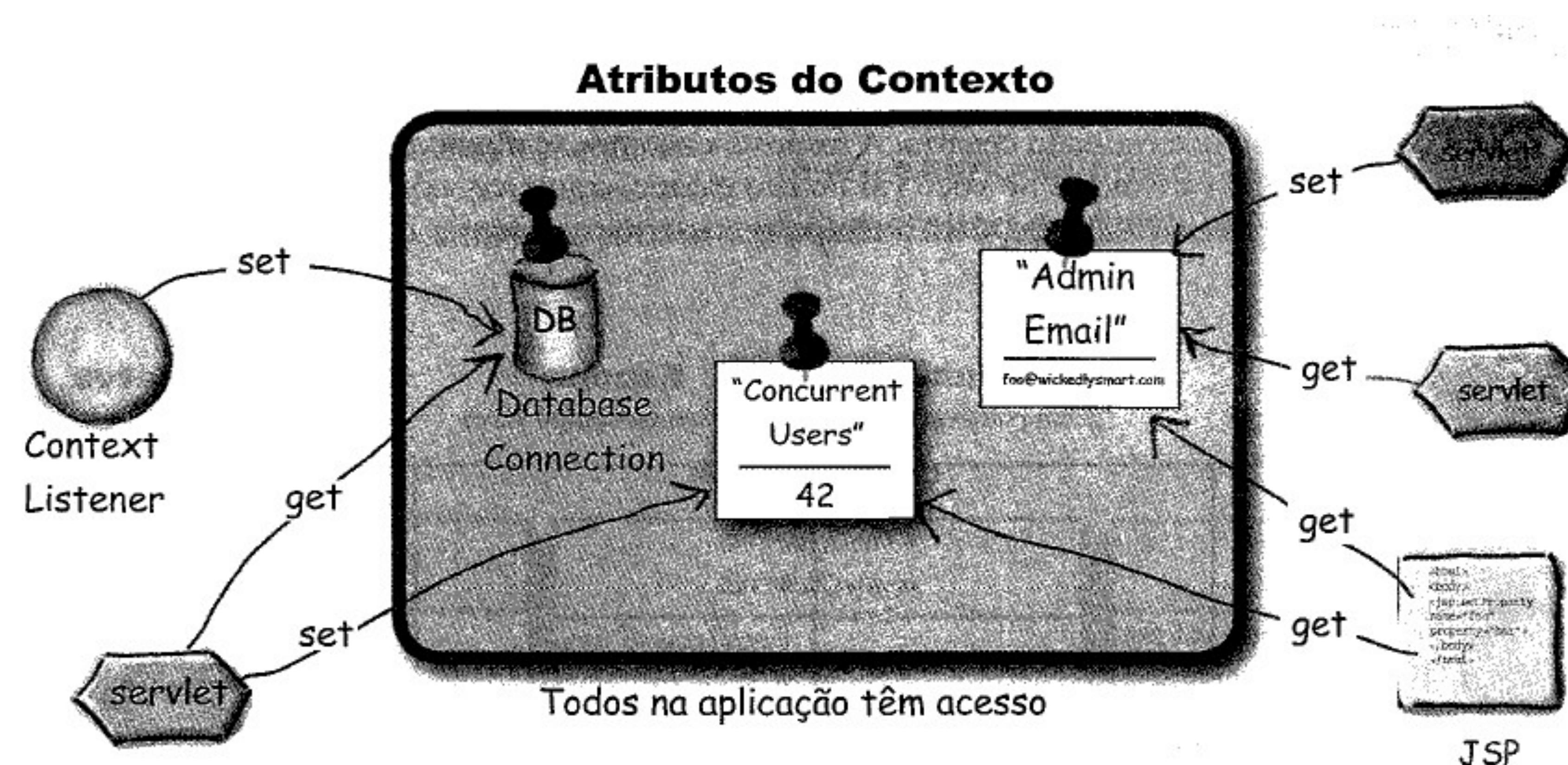
Compartilhando Informações

- Atributos de sessão:



Compartilhando Informações

- Atributos de aplicação:



Compartilhando Informações

- Métodos usados para a manipulação de atributos:



Exemplo

Listener que inicializa um atributo

```
@WebListener
public class InicializaObjetoListener implements ServletContextListener{

    @Override
    public void contextInitialized(ServletContextEvent event) {
        //Instanciando o objeto
        Calendar rightNow = Calendar.getInstance();

        //Compartilhando o objeto com os demais componentes da aplicação
        event.getServletContext().setAttribute("startTime", rightNow);
    }

    @Override
    public void contextDestroyed(ServletContextEvent event) {
        event.getServletContext().removeAttribute("startTime");
    }
}
```

Exemplo

Servlet que recupera e imprime o valor do atributo

```
@WebServlet("/mostraHora")
public class MostraHoraInicializacaoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Exemplo </title></head>");
        out.println("<body>");
        //Recuperando o valor do atributo
        Calendar startTime = (Calendar) getServletContext().getAttribute("startTime");
        //Imprimindo a hora de inicialização
        out.println("<p> Horário de inicialização da aplicação: "+startTime.getTime()+"</p>");

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

}
```

Compartilhando Informações

- Cuidado: atributos de contexto podem ser utilizados simultaneamente pela aplicação
- Problemas de sincronização e consistência podem acontecer;



Compartilhando Informações

- Vamos supor a seguinte situação:
 - Uma thread A cria no contexto o atributo “numero”, e atribui a ele o valor 25;
 - Uma thread B, da mesma aplicação, cria o mesmo atributo no contexto e atribui a ele o valor 50;
 - Mais tarde, a thread A recupera o valor do atributo número, obtendo o valor 50;
 - **Resultado: O valor encontrado é inconsistente;**

```
public class SynchronizedServlet extends HttpServlet{

    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        ServletContext context = this.getServletContext();
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Meu primeiro Servlet </title></head>");
        out.println("<body>");
        synchronized (context){          Sincronizando o acesso a atributos do contexto
            context.setAttribute("numero", "15");
            context.setAttribute("cliente", "John");
            out.println("<p> <b>Numero:</b> "+context.getAttribute("numero")+"</p>");
            out.println("<p> <b>Cliente:</b> "+context.getAttribute("cliente")+"</p>");
        }
        out.println("</body>");
        out.println("</html>");
    }

}
```

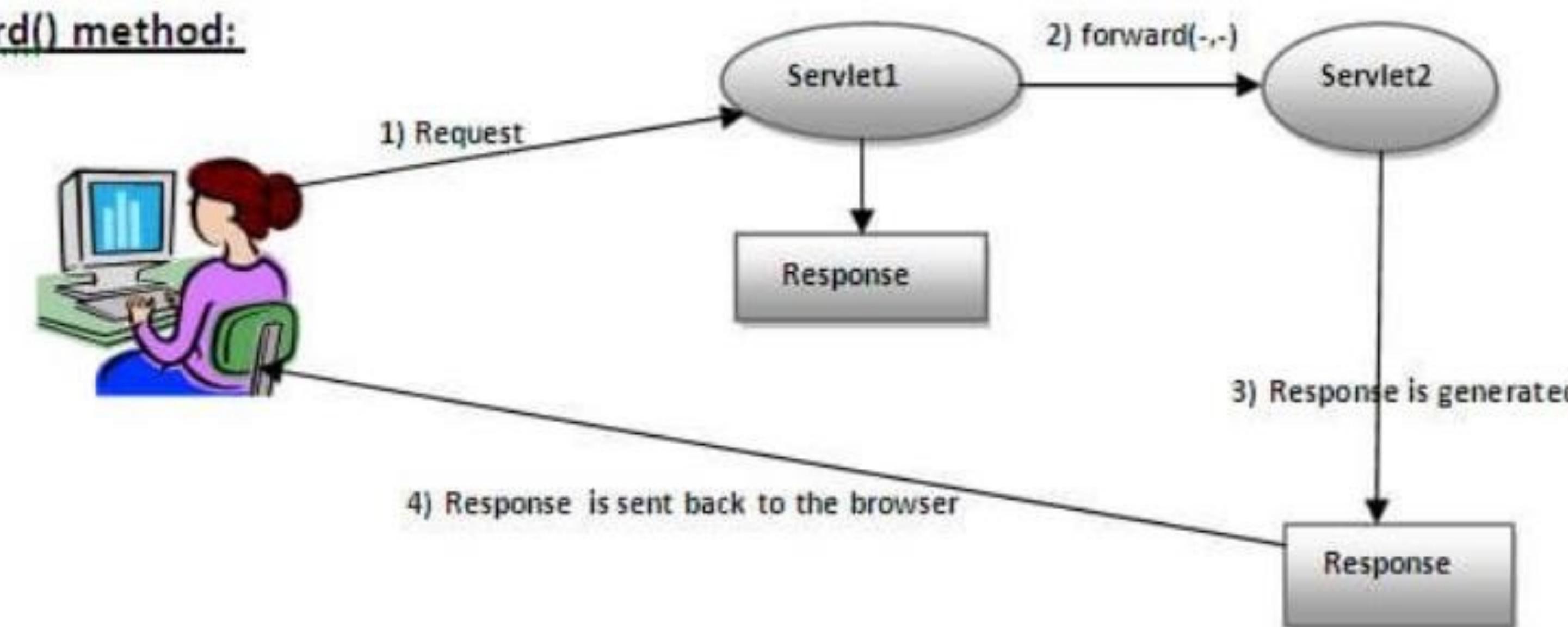
Despachando Requisições

- Um servlet pode transferir o controle da requisição recebida para outros componentes da aplicação;
- Esta transferência pode ser feita de forma temporária ou definitiva;
- Requisições são transferidas através do **RequestDispatcher**;

Despachando Requisições

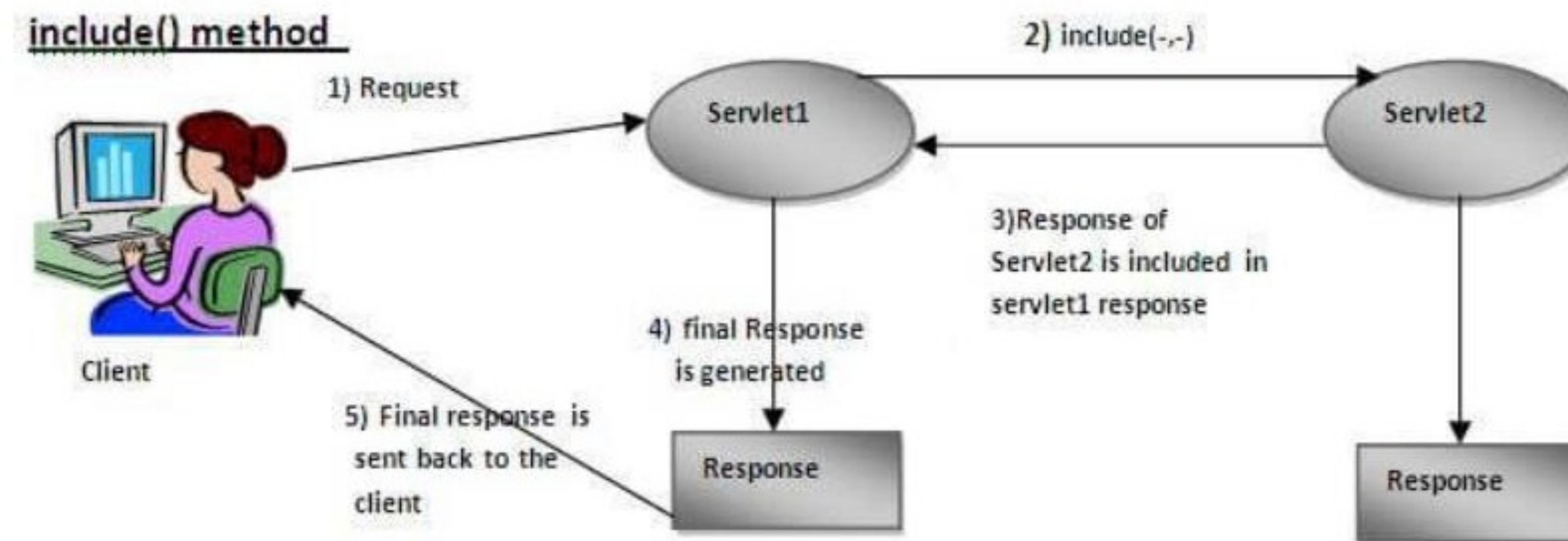
- **public void forward(ServletRequest request, ServletResponse response)**

forward() method:

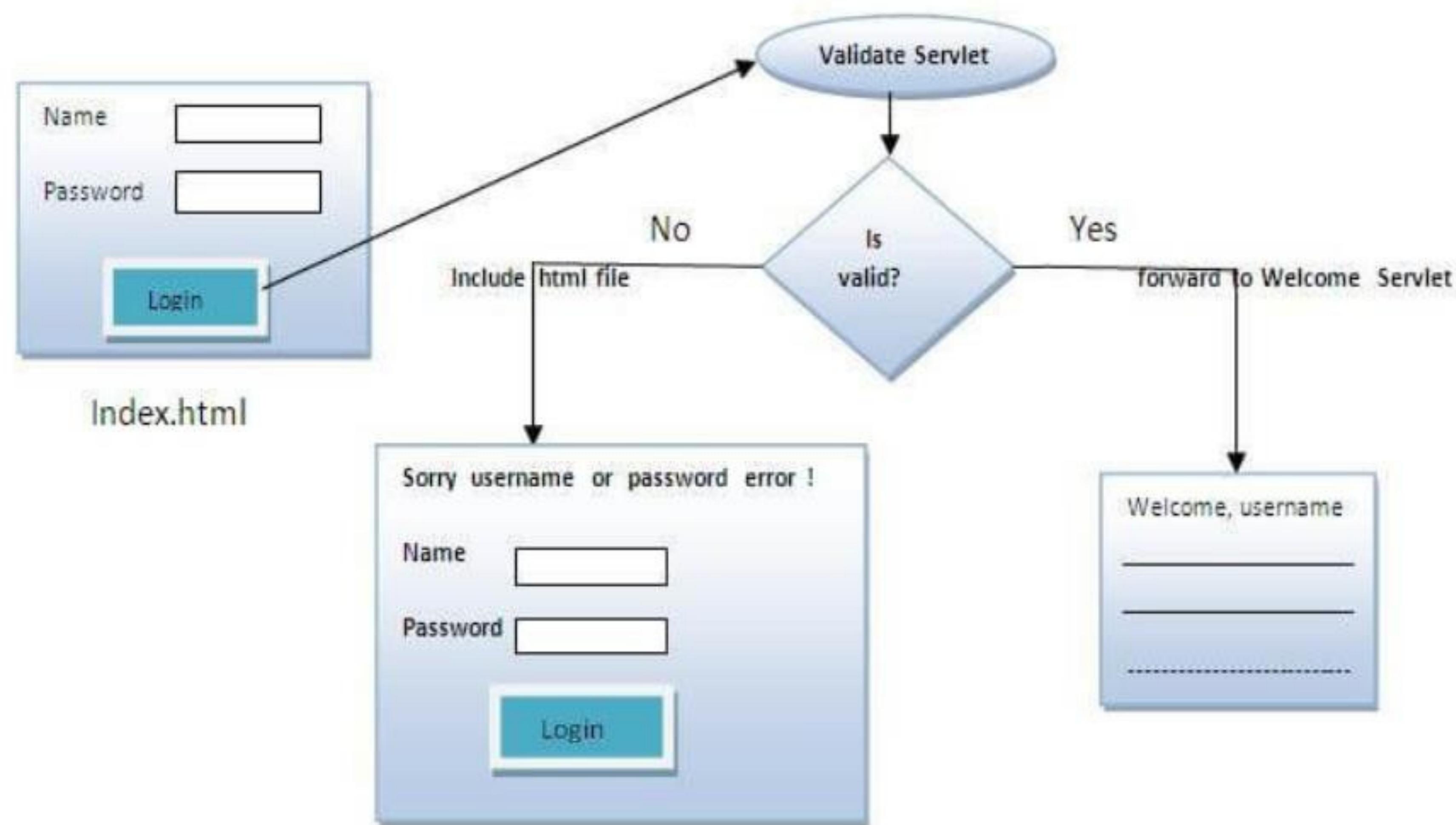


Despachando Requisições

- **public void include(ServletRequest request, ServletResponse response)**



Despachando Requisições



Despachando Requisições - Exemplo

- Neste caso, vamos gerar a página inicial através de três servlets:
 - Um para gerar apenas o banner;
 - Um para gerar apenas os copyrights;
 - Um para gerar o conteúdo principal e controlar a geração da página;

O servlet que gera a página principal, com a ajuda dos outros servlets

```
@WebServlet("/inicial")
public class SantaMariaServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Lojas Santa Maria </title></head>");
        out.println("<body>");

        //Obtendo um despachante para o servlet que gera o banner e despachando para o mesmo
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner");
        dispatcher.include(request, response);

        out.println("<p align=\"center\"> <h1> Esta é a página inicial das lojas Santa Maria. "
            + "</h1></p>");

        //Despachando para o servlet que gera o copyright
        dispatcher = getServletContext().getRequestDispatcher("/copyright");
        dispatcher.include(request, response);

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

Despachando Requisições

- O servlet que gera o copyright:

```
@WebServlet("/copyright")
public class CopyrightServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        out.print("<p align=\"right\"><h2> Copyritghs: Lojas Santa Maria Ltda</h2></p>");
    }
}
```

Despachando Requisições

- O servlet que gera o banner:

```
@WebServlet("/banner")
public class BannerServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        doPost(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException{
        PrintWriter out = response.getWriter();
        out.print("<p align = \"center\"> Lojas Santa Maria. "
            + "Aqui o cliente sempre tem prioridade. </p>");
    }

}
```

Despachando Requisições

- Vamos refazer o exemplo da página inicial da loja;
- Agora, os conteúdos do banner e do copyright serão definidos em arquivos HTML;

Despachando Requisições

- O arquivo banner.html:

```
<p align = "center">  
    Lojas Santa Maria. Aqui o cliente sempre tem prioridade.  
</p>
```

- O arquivo copyright.html:

```
<p align="right\">  
    <h2> Copyritghs: Lojas Santa Maria Ltda</h2>  
</p>
```

O servlet que gera a página principal, com a ajuda dos outros servlets

```
@WebServlet("/ inicialHtml")
public class SantaMariaServletHtml extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Lojas Santa Maria </title></head>");
        out.println("<body>");

        //Obtendo um despachante para o arquivo HTML que contém o banner
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/banner.html");
        dispatcher.include(request, response);

        out.println("<p align=\"center\"> <h1> Esta é a página inicial das lojas Santa Maria."
            + "</h1></p>");

        //Obtendo um despachante para o arquivo HTML que contém o copyright
        dispatcher = getServletContext().getRequestDispatcher("/copyright.html");
        dispatcher.include(request, response);

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

Despachando Requisições

- O método **forward**:
 - Este método concede ao componente que recebe a requisição o controle definitivo da mesma;
 - Depois que o método é chamado a requisição não retorna mais para o servlet que a despachou;
 - O componente que recebe a requisição é responsável por finalizar seu processamento ou encaminhá-la para outro componente da aplicação;

```
@WebServlet("/busca")
public class RealizaBuscaServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

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

        //Realizando a consulta
        String key = request.getParameter("key");
        List<Book> bookList = searchByKey(key);

        //Colocando o resultado como atributo da requisição
        request.setAttribute("result", bookList);

        //Despachando para o servlet que vai mostrar o resultado
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/mostraResultado");
        dispatcher.forward(request, response);
    }

    private List<Book> searchByKey(String key) {...19 linhas }

}
```

```
public class MostraResultadoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

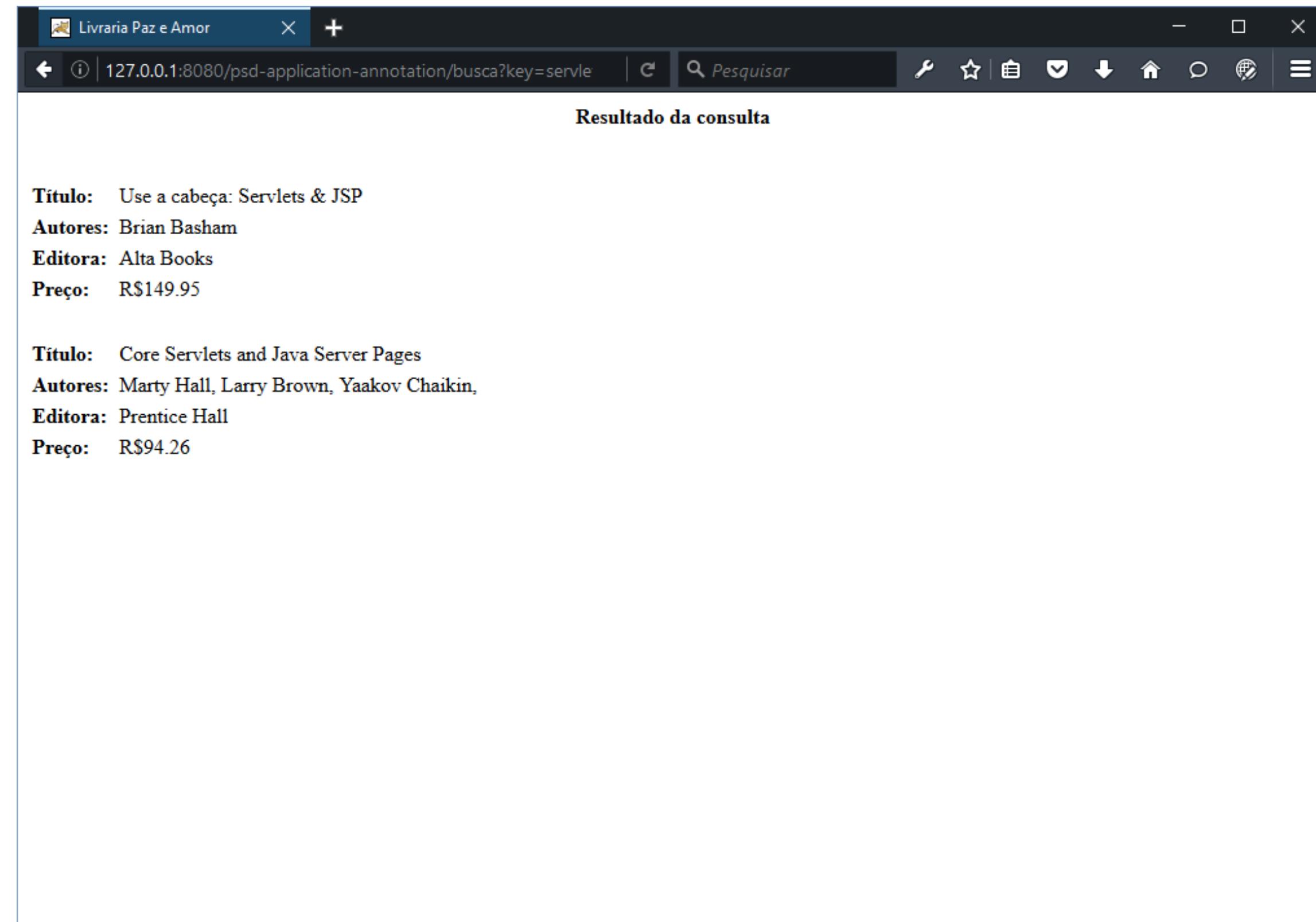
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{
        PrintWriter out = response.getWriter();
        out.println("<html><head><title> Livraria Paz e Amor </title></head>");
        out.println("<body> <p align=\"center\"><b>Resultado da consulta </b></p>");
        List<Book> bookList = (List<Book>) request.getAttribute("result");
        out.println("<table>");

        for(Book book : bookList){
            out.println("<tr>");
            out.println("<tr><td><b> Título: </b><td/><td>" +book.getTitle()+"</td></tr>");
            out.println("<tr><td><b> Autores: </b><td/><td>" +book.getAuthors()+"</td></tr>");
            out.println("<tr><td><b> Editora: </b><td/><td>" +book.getPublisher()+"</td></tr>");
            out.println("<tr><td><b> Preço: </b><td/><td>R$" +book.getPrice()+"</td></tr>");
            out.println("<tr><td> &ampnbsp<td>&ampnbsp</td><td>&ampnbsp</td></tr>");
        }
        out.println("</table>");
        out.println("</body></html>");
    }

}
```

Despachando Requisições

- Resultado final:



Revisão

- Métodos da interface RequestDispatcher interface
 - **public void forward(ServletRequest request,ServletResponse response)**
 - Encaminha uma solicitação de um servlet para outro recurso (servlet, arquivo JSPou arquivo HTML) no servidor.
 - **public void include(ServletRequest request,ServletResponse response)**
 - Inclui o conteúdo de um recurso (servlet, página JSPou arquivo HTML) na resposta.

Redirecionando Requisições

- A classe `HttpServletResponse` oferece métodos que nos permitem redirecionar o cliente da aplicação;
- Estes métodos são:
 - `sendRedirect(String url);`
 - `sendError(int code);`

Servlet que redireciona o cliente para outros locais

```
@WebServlet("/redirecionamento")
public class RedirecionamentoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

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

        PrintWriter out = response.getWriter();
        //Obtendo o parâmetro informado pelo usuário
        String option = request.getParameter("option");
        //Redirecionando o cliente para sites externos
        if(option.equals("sports"))
            response.sendRedirect("http://www.globoesporte.com");
        if(option.equals("news"))
            response.sendRedirect("http://www.globo.com");
        if(option.equals("movies"))
            response.sendRedirect("http://netflix.com");

        //Redirecionando o cliente para um servlet da aplicação local
        response.sendRedirect("servletLocal");
    }

}
```

Servlet que realiza o redirecionamento de erro

```
@WebServlet("/autentica")
public class AutenticacaoServlet extends HttpServlet{

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException{...3 linhas }

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

        //Obtendo os dados para autenticação
        String user = request.getParameter("user");
        String password = request.getParameter("password");

        boolean result = check(user,password);

        if(result)
            response.sendRedirect("acesso.html");
        else response.sendError(999);

    }

    private boolean check(String user, String password) {...5 linhas }

}
```