



Curso de **Java8** para **Web**

Professor
Antonio Benedito Coimbra Sampaio Jr

abc  | Treinamentos

www.abctreinamentos.com.br

Quarta Disciplina

JEE – Java Servlets e JSP

- **UNIDADE 1:** Introdução à Internet, WEB e HTML
- **UNIDADE 2: Java Servlets**
- **UNIDADE 3:** JSP
- **UNIDADE 4:** Padrão de Projeto MVC (Integração Servlet e JSP)

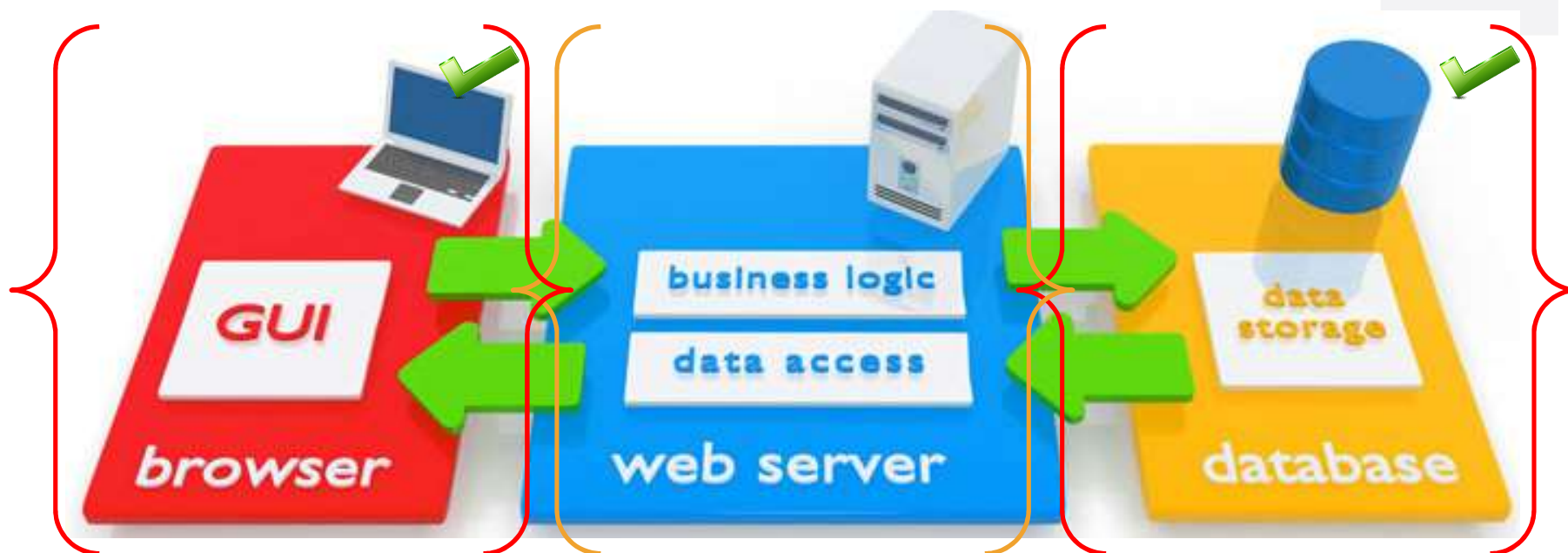
UNIDADE 2

JAVA SERVLETS

Componentes WEB JEE

Modelo de Aplicação JEE

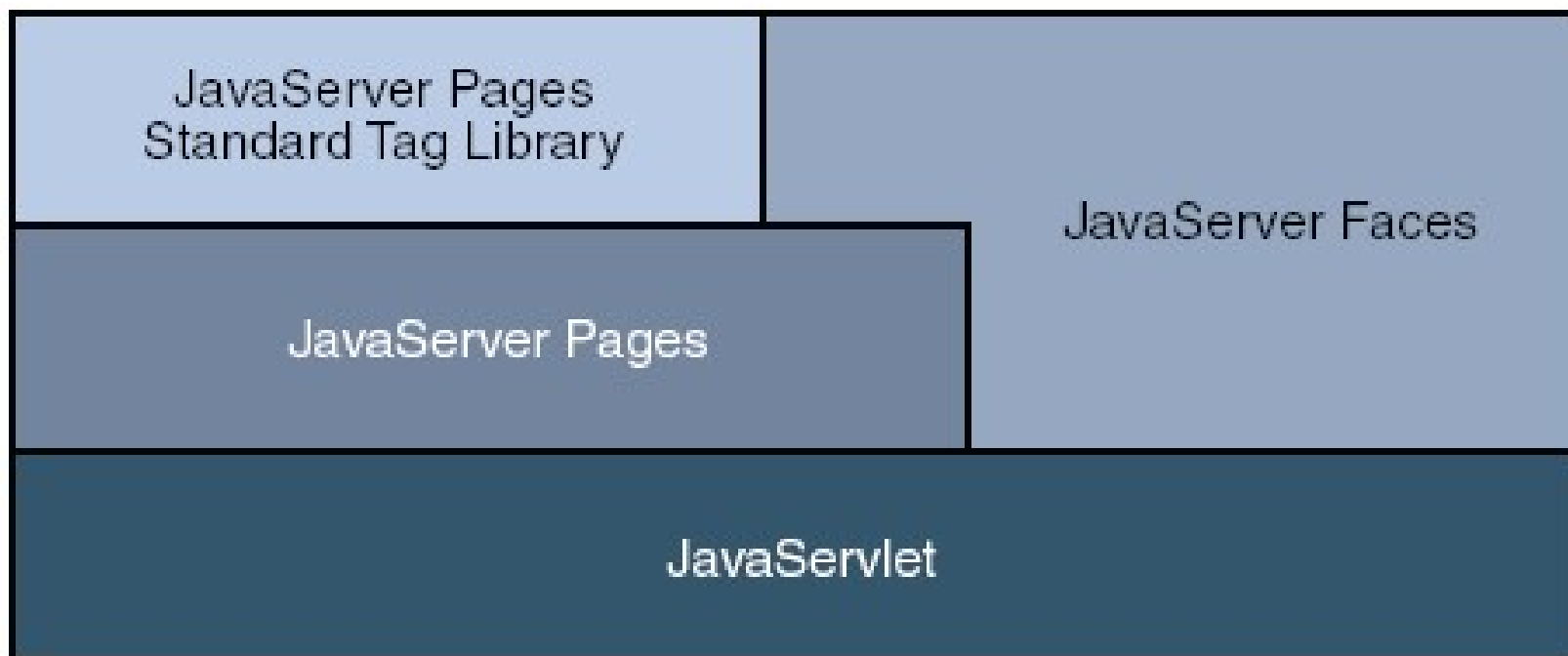
- A plataforma JEE utiliza um modelo de aplicação distribuída multicamada.



- A lógica da aplicação é dividida em componentes de acordo com a sua função.
- Os vários componentes que constituem uma aplicação JEE são instalados em diferentes equipamentos.

COMPONENTES WEB JEE

- A Plataforma JEE define quatro tecnologias básicas para a construção de Aplicações WEB:
 - **Java Servlets 3.1**
 - **JavaServer Pages 2.2**
 - **JavaServer Faces 2.2**
 - **JavaServer Pages Standard Tag Library 1.2.1**



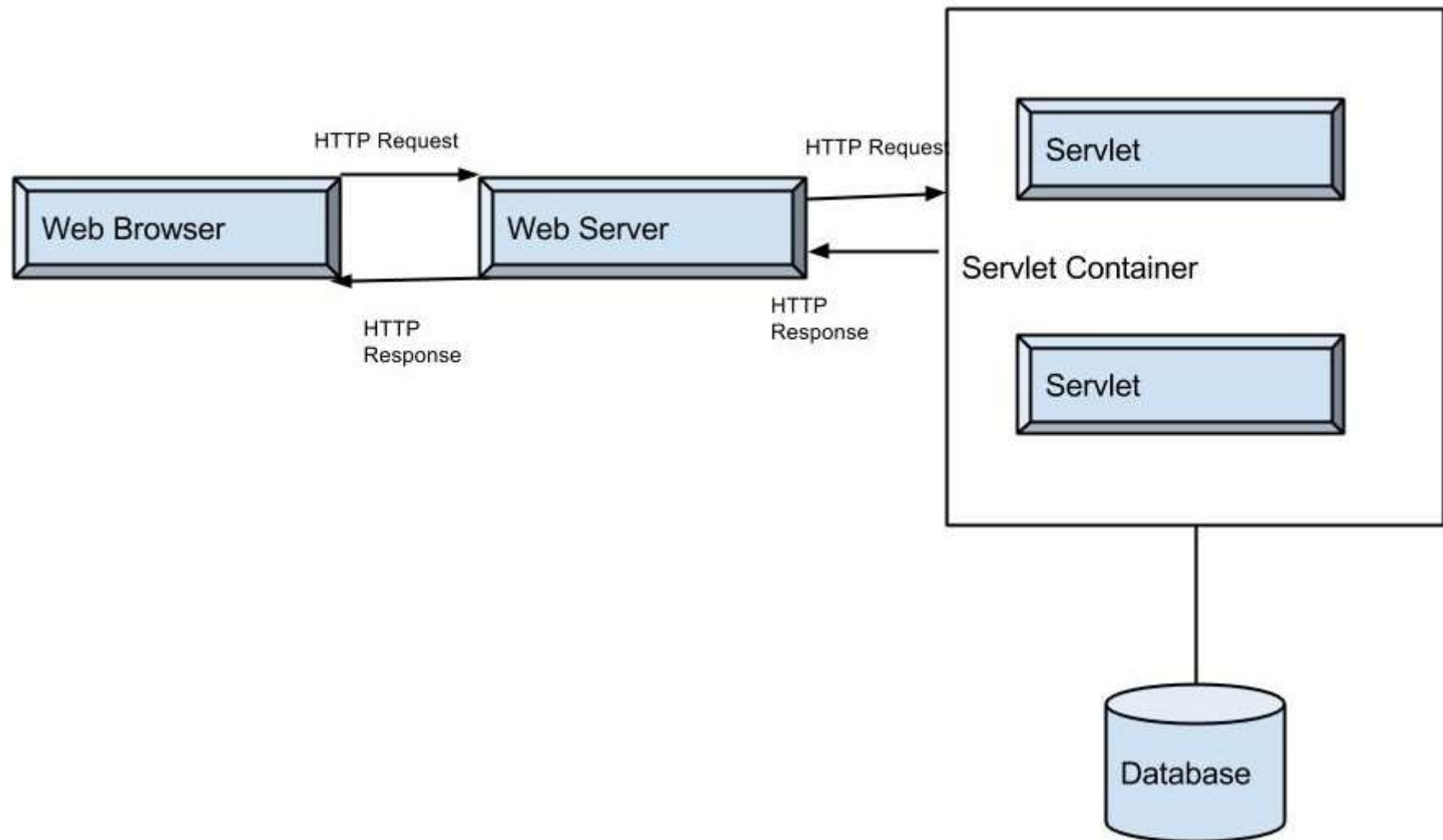
JAVA SERVLETS

Definição

- Servlets são módulos que estendem servidores orientados a requisição/resposta, como servidores Web JEE.
- Um servlet é similar a extensão proprietária de um servidor, sendo executado na JVM do mesmo.
- Esta tecnologia surgiu em 1997, tendo como objetivo ser o novo paradigma de programação para servidores WEB.
- Servlets são utilizados para a criação de aplicações WEB com java.

JAVA SERVLETS

Arquitetura

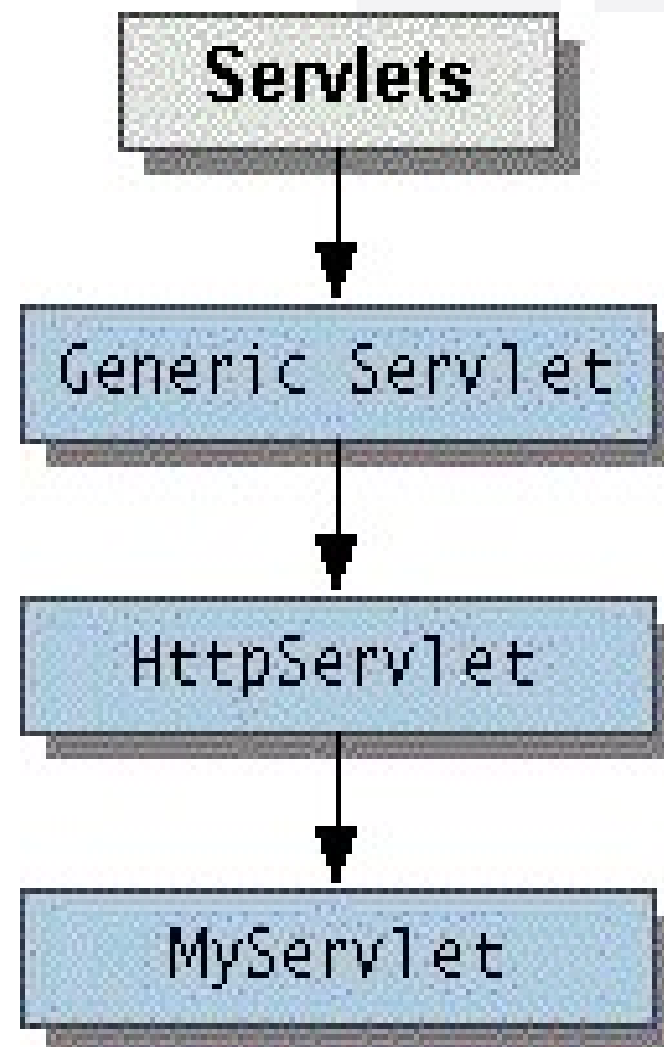


API SERVLET

- A API Servlet é composta por dois pacotes: **javax.servlet** e **javax.servlet.http**.
- O pacote **javax.servlet** define um conjunto de classes e interfaces utilizados na comunicação entre os servlets e o seu ambiente de execução (web container).
 - Exemplo: Servlet, ServletConfig, ServletContext, ServletContextListener, ServletRequest, ServletResponse, etc.
- O pacote **javax.servlet.http** define um conjunto de classes e interfaces utilizados na comunicação - via protocolo HTTP - entre os servlets e o seu ambiente de execução (web container).
 - Exemplo: HttpSession, HttpServletRequest, HttpServletResponse, etc.

MODELO SERVLET

- Todo servlet criado pelo desenvolvedor deverá implementar a interface **Servlet** e ser subclasse de **HttpServlet**.
- A interface **javax.servlet.Servlet** define todos os métodos que um servlet deverá implementar.
- A classe abstrata **javax.servlet.http.HttpServlet** define um servlet para utilizar o protocolo de comunicação hipermídia HTTP.
- A subclasse de **javax.servlet.http.HttpServlet** (**MyServlet**) deverá anular pelo menos 01 método daquela superclasse.

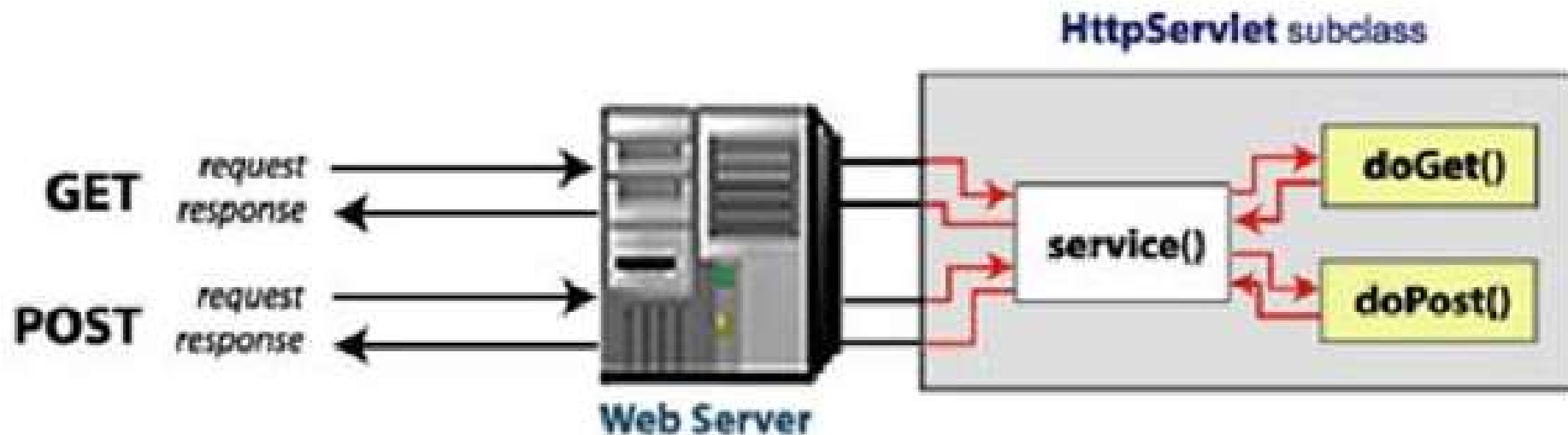


CLASSE HTTPServlet

- Principais métodos dessa classe:
 - **doGet(...)**, para oferecer suporte aos métodos GET do protocolo HTTP;
 - **doPost(...)**, para oferecer suporte aos métodos POST do HTTP;
 - **doPut(...)**, para oferecer suporte aos métodos PUT do HTTP;
 - **doDelete(...)**, para oferecer suporte aos métodos DELETE do HTTP;
 - **init(...)** e **destroy(...)**, para gerenciar o ciclo de vida de um servlet;
 - **getServletInfo(...)**, para fornecer informações do servlet.

HTTP GET e POST

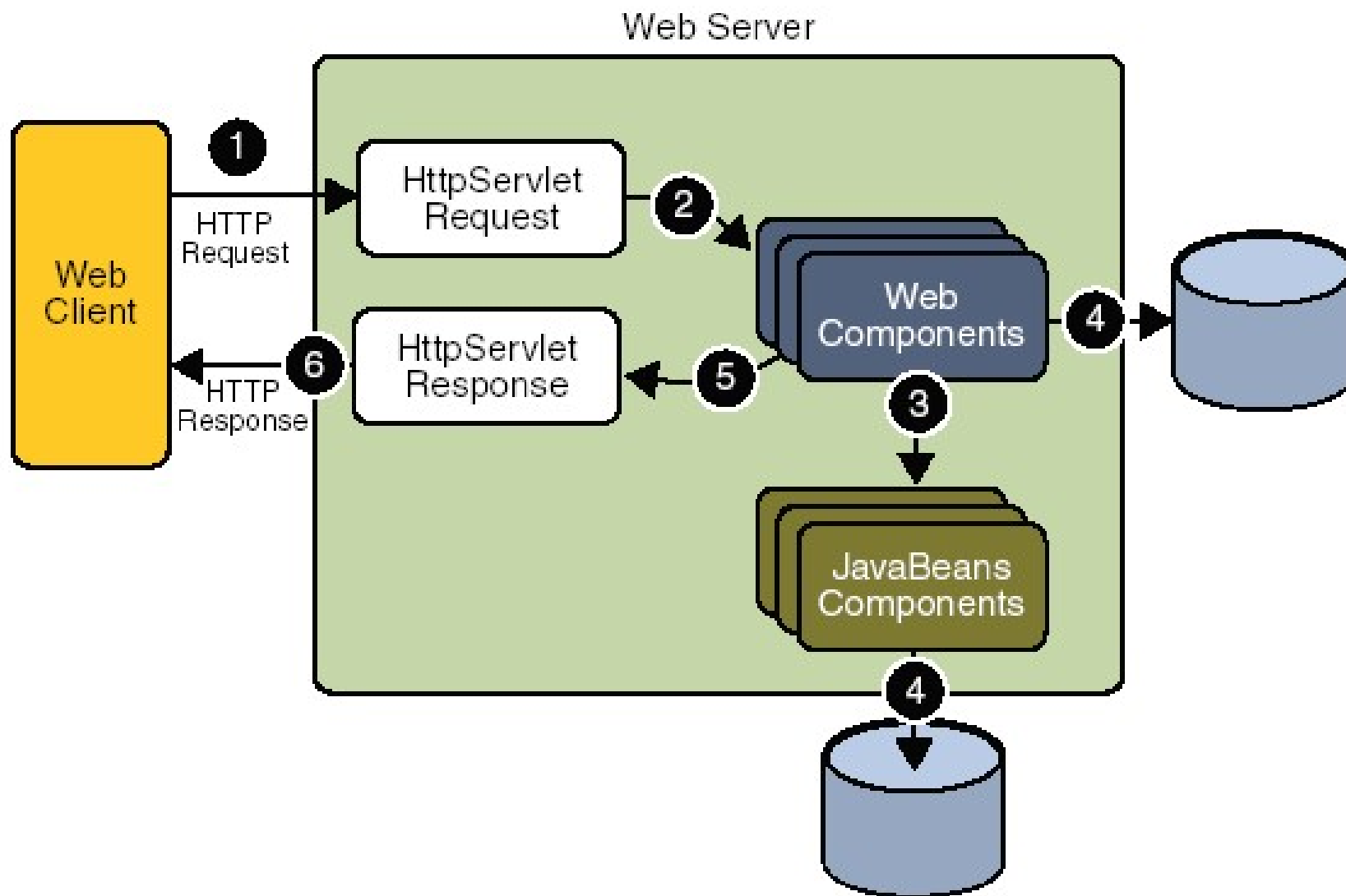
- Métodos de requisição definidos no protocolo HTTP
 - **GET (requisições normais)**
 - **POST (requisições através de formulários)**
 - PUT
 - DELETE



<http://javawebaction.blogspot.com.br/2011/07/what-is-java-servlet-life-cycle-and-how.html>

PRINCIPAIS INTERFACES

HttpServletRequest e HttpServletResponse



HTTPSERVLETREQUEST

- Esta interface fornece informações de requisição aos servlets.
- Os seus principais métodos estão definidos abaixo:
- **String getParameter(String parameter)** → Retorna o valor de um parâmetro.
- **String [] getParameterValues(String parameter)** → Retorna um vetor de String contendo todos os valores que aquele dado parametro possui, ou null se não houver.
- **String getRequestedURI()** → Retorna a URI requisitada.
- **String getQueryString()** → Retorna uma string contendo todos os parâmetros e valores associados em uma consulta.

HTTPSERVLETRESPONSE

- Esta interface fornece informações de resposta dos servlets.
- Os seus principais métodos estão definidos abaixo:
- **PrintWriter getWriter() throws IOException** → Retorna um objeto `PrintWriter` que pode enviar texto como resposta a uma requisição.
- **ServletOutputStream getOutputStream() throws IOException** → Retorna um objeto `ServletOutputStream` que pode enviar dados binários como resposta a uma requisição.
- **public void setContentType(String tipo)** → Define o formato da resposta (text/html;charset=UTF-8;etc.) que será enviado pelo servlet.

Exercício

1) [AOCP - 2012 – BRDE] Sobre Servlets, analise as assertivas e assinale a alternativa que aponta as corretas.

- I. Servlets são implementadas como arquivos de classe da Linguagem Java.
- II. Servlets são independentes de plataforma, de modo que podem ser executadas em diferentes servidores, em diferentes sistemas operacionais.
- III. As Servlets podem acessar qualquer uma das APIs Java. Uma Servlet pode usar a API JDBC para acessar e armazenar dados ou para acessar objetos remotos.
- IV. Ao criar uma Servlet, somos obrigados a reescrever nove métodos presentes à interface que foi implementada.

- a) Apenas I e II.
- b) Apenas I e III.
- c) Apenas II e III.
- d) Apenas I, II e III.
- e) I, II, III e IV.

Exercício

1) [AOCP - 2012 – BRDE] Sobre Servlets, analise as assertivas e assinale a alternativa que aponta as corretas.

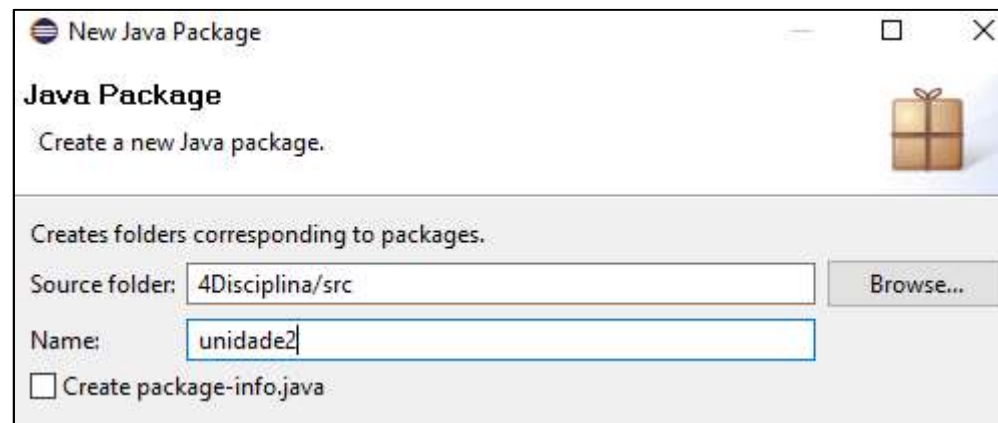
- I. Servlets são implementadas como arquivos de classe da Linguagem Java.
- II. Servlets são independentes de plataforma, de modo que podem ser executadas em diferentes servidores, em diferentes sistemas operacionais.
- III. As Servlets podem acessar qualquer uma das APIs Java. Uma Servlet pode usar a API JDBC para acessar e armazenar dados ou para acessar objetos remotos.
- IV. Ao criar uma Servlet, somos obrigados a reescrever nove métodos presentes à interface que foi implementada.

- a) Apenas I e II.
- b) Apenas I e III.
- c) Apenas II e III.
- d) Apenas I, II e III.**
- e) I, II, III e IV.

Primeiro Servlet

CRIAÇÃO DO PRIMEIRO SERVLET

- Criar o pacote **unidade2** no projeto **4Disciplina**.



- Selecionar a opção **“New”** ⇒ **“Servlet”**

CRIAÇÃO DO PRIMEIRO SERVLET

Create Servlet
Specify class file destination.

Project: 4Disciplina

Source folder: /4Disciplina/src Browse...

Java package: unidade2 Browse...

Class name: PrimeiroServlet

Superclass: javax.servlet.http.HttpServlet Browse...

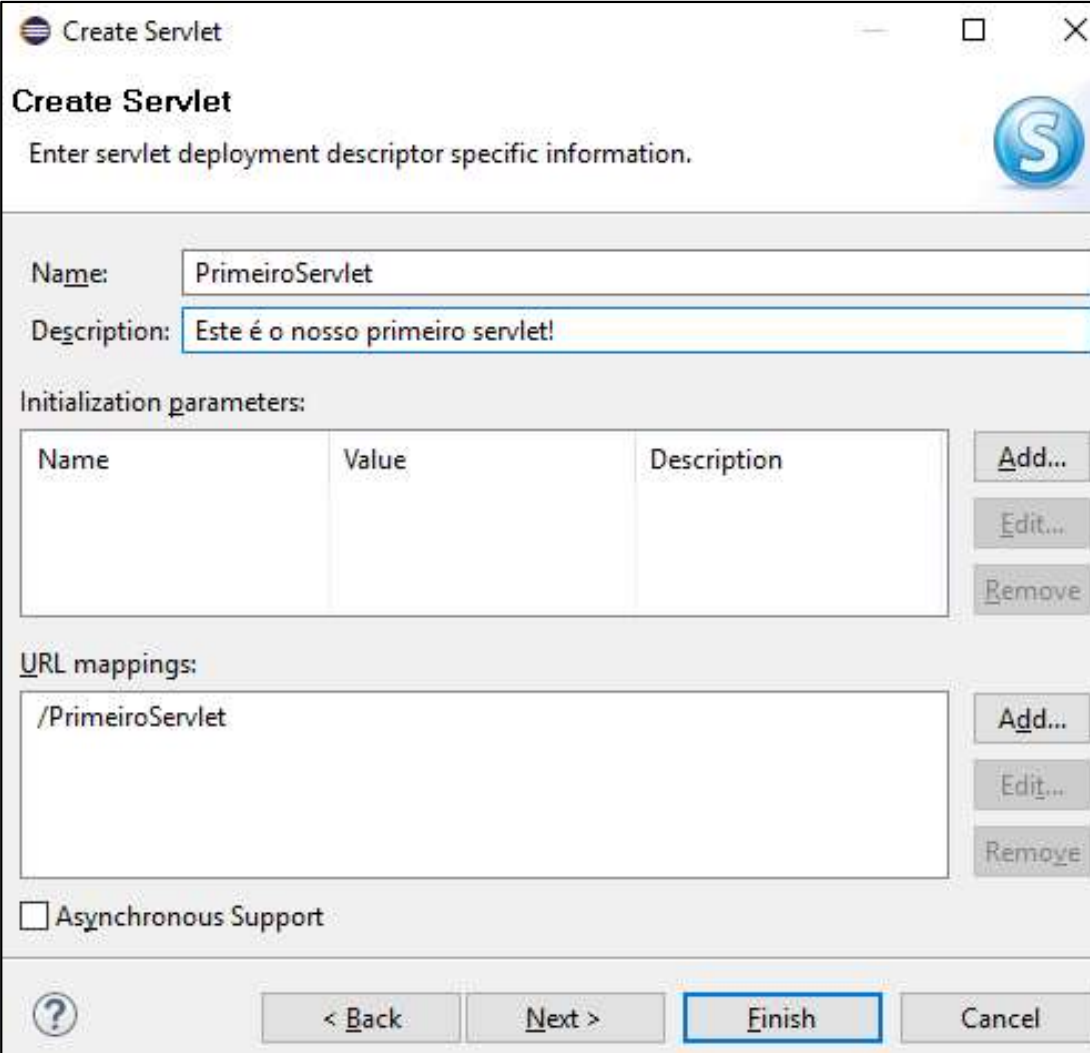
☐ Use an existing Servlet class or JSP

Class name: PrimeiroServlet Browse...

? < Back Next > Finish Cancel

- Selecionar a opção **Next**.

CRIAÇÃO DO PRIMEIRO SERVLET



The image shows a 'Create Servlet' dialog box from an IDE. It contains fields for 'Name' (PrimeiroServlet) and 'Description' (Este é o nosso primeiro servlet!). Below these are sections for 'Initialization parameters' and 'URL mappings', each with a table and 'Add...', 'Edit...', and 'Remove' buttons. At the bottom, there is an 'Asynchronous Support' checkbox and a set of navigation buttons: '?', '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

Create Servlet

Enter servlet deployment descriptor specific information.

Name: PrimeiroServlet

Description: Este é o nosso primeiro servlet!

Initialization parameters:

Name	Value	Description
------	-------	-------------

URL mappings:

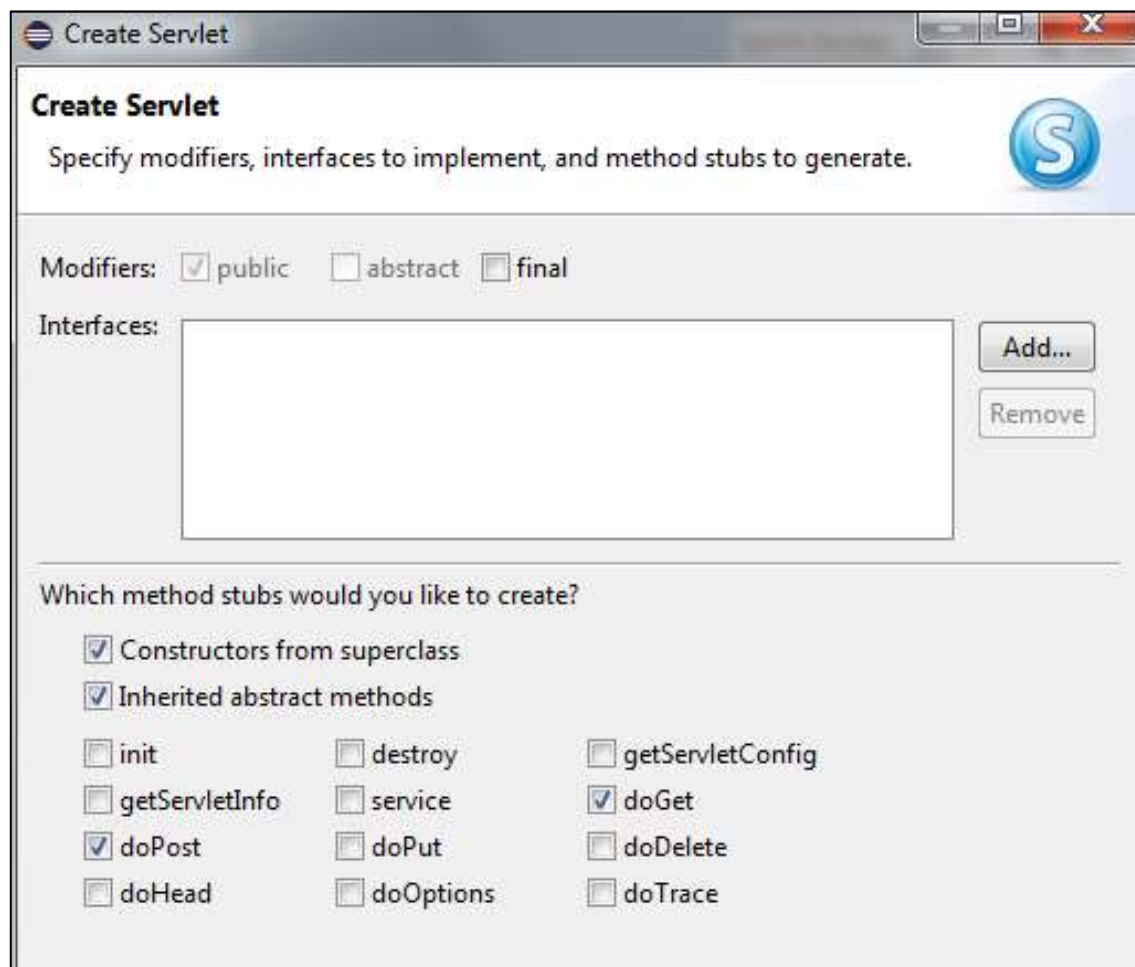
/PrimeiroServlet

☐ Asynchronous Support

? < Back Next > **Finish** Cancel

- Selecionar a opção **Next**.

CRIAÇÃO DO PRIMEIRO SERVLET



- Selecionar a opção **Finish**.

CÓDIGO DO PRIMEIRO SERVLET

```
package unidade2;

...
@WebServlet(description = "Este é o nosso primeiro servlet!",
urlPatterns = { "/PrimeiroServlet" })
public class PrimeiroServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public PrimeiroServlet() {
        super();
    }

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

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

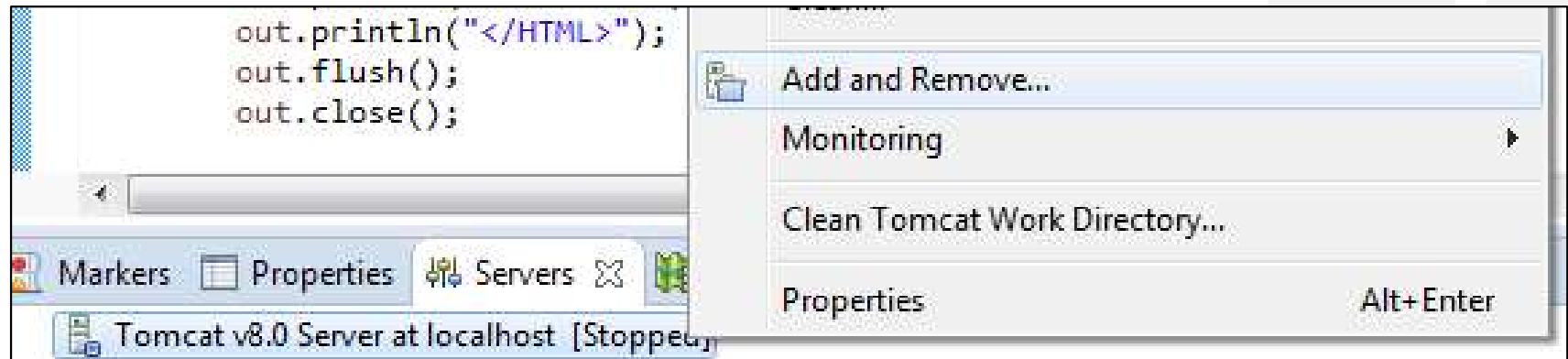
CÓDIGO DO PRIMEIRO SERVLET

- Incluir no método **doGet(...)** o trecho de código abaixo:

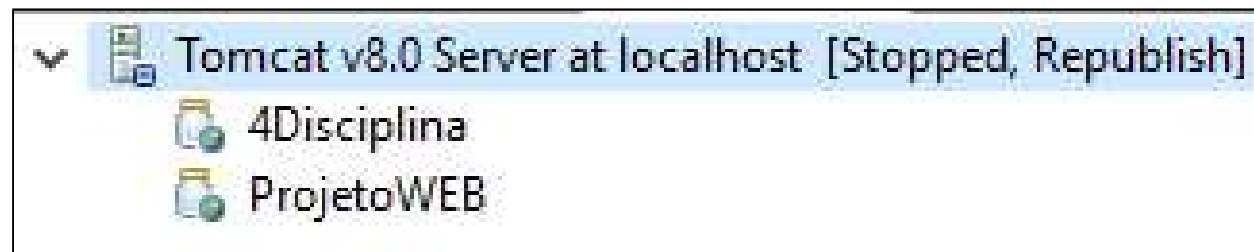
```
...  
protected void doGet(...) {  
    response.setContentType("text/html");  
    PrintWriter out = response.getWriter();  
    out.println("<HTML>");  
    out.println("<HEAD><TITLE>Servlet</TITLE></HEAD>");  
    out.println("<BODY>");  
    out.print("Esta é a ");  
    out.print(this.getClass());  
    out.println(" usando o método GET");  
    out.println("    </BODY>");  
    out.println("</HTML>");  
    out.flush();  
    out.close();  
}
```


DEPLOY DO PROJETO

- Na aba “**Servers**” do Eclipse, clicar com o botão direito no servidor JEE Tomcat v8.0.

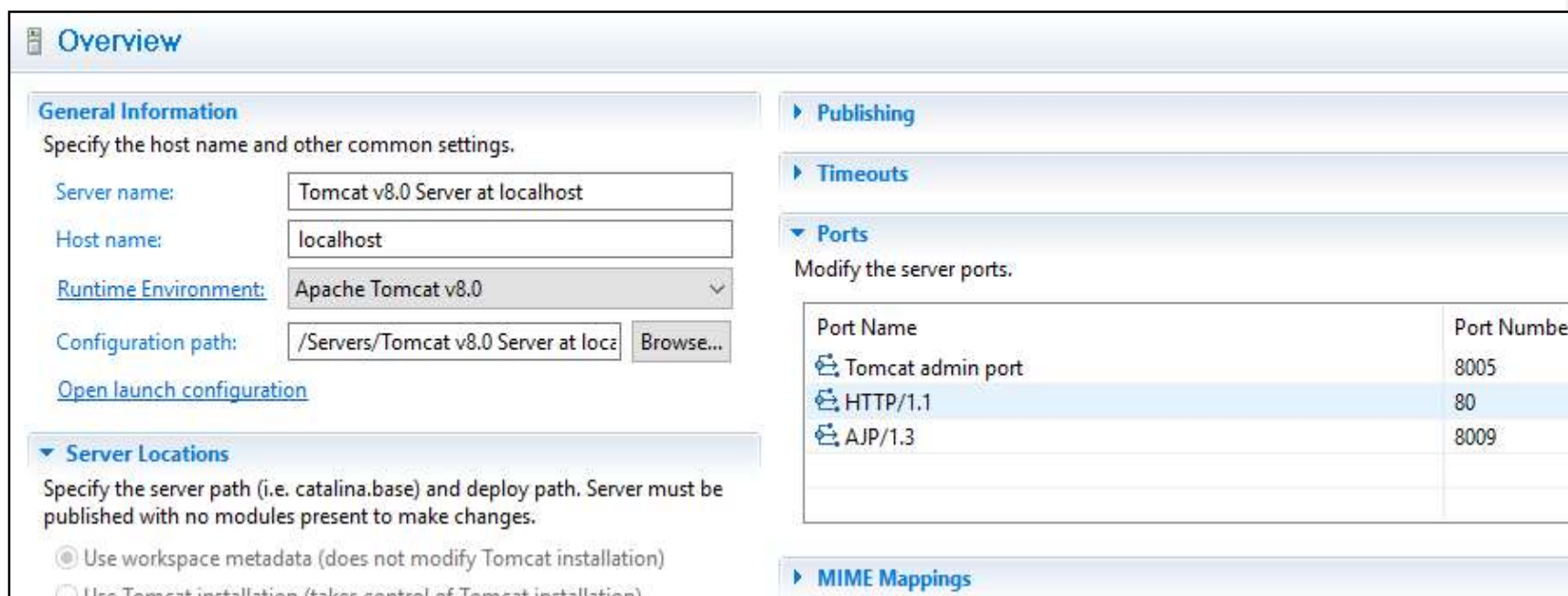


- Escolha a opção “**Add and Remove...**”. Selecionar “**Add**” para o “**4Disciplina**”. Clicar no botão “**Finish**”. Clique no botão para iniciar o servidor.



EXECUÇÃO DO SERVLET

- Sugestão: alterar a porta padrão do servidor JEE Tomcat de 8080 para 80. É possível fazer a alteração no arquivo **server.xml** ou no editor do Eclipse.



Overview

General Information
Specify the host name and other common settings.

Server name: Tomcat v8.0 Server at localhost

Host name: localhost

Runtime Environment: Apache Tomcat v8.0

Configuration path: /Servers/Tomcat v8.0 Server at localhost Configuration/Browser... Browse...

[Open launch configuration](#)

Server Locations
Specify the server path (i.e. catalina.base) and deploy path. Server must be published with no modules present to make changes.

☒ Use workspace metadata (does not modify Tomcat installation)

☐ Use Tomcat installation (takes control of Tomcat installation)

Publishing

Timeouts

Ports
Modify the server ports.

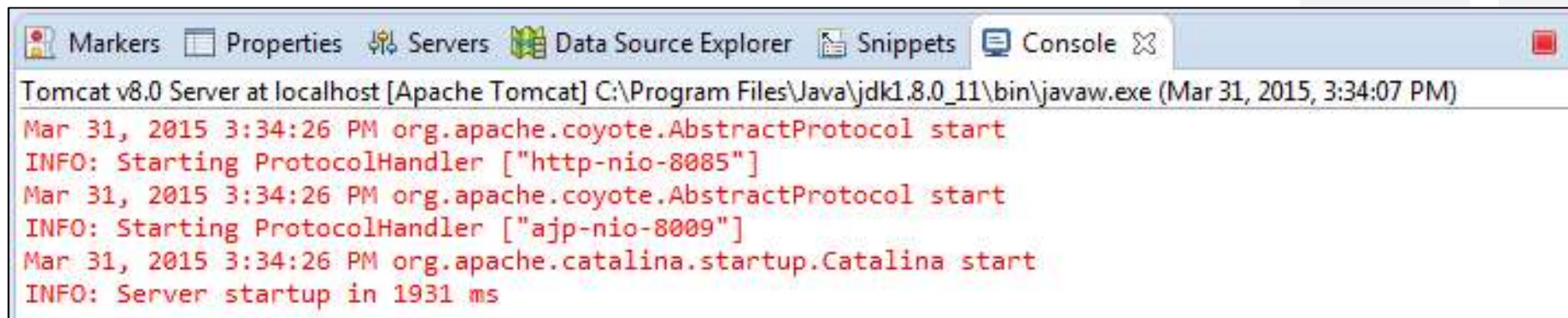
Port Name	Port Number
Tomcat admin port	8005
HTTP/1.1	80
AJP/1.3	8009

MIME Mappings

- Tendo iniciado corretamente o servidor Tomcat, abrir um navegador Web e digitar o seguinte endereço:

http://máquina:porta/projeto/Classe

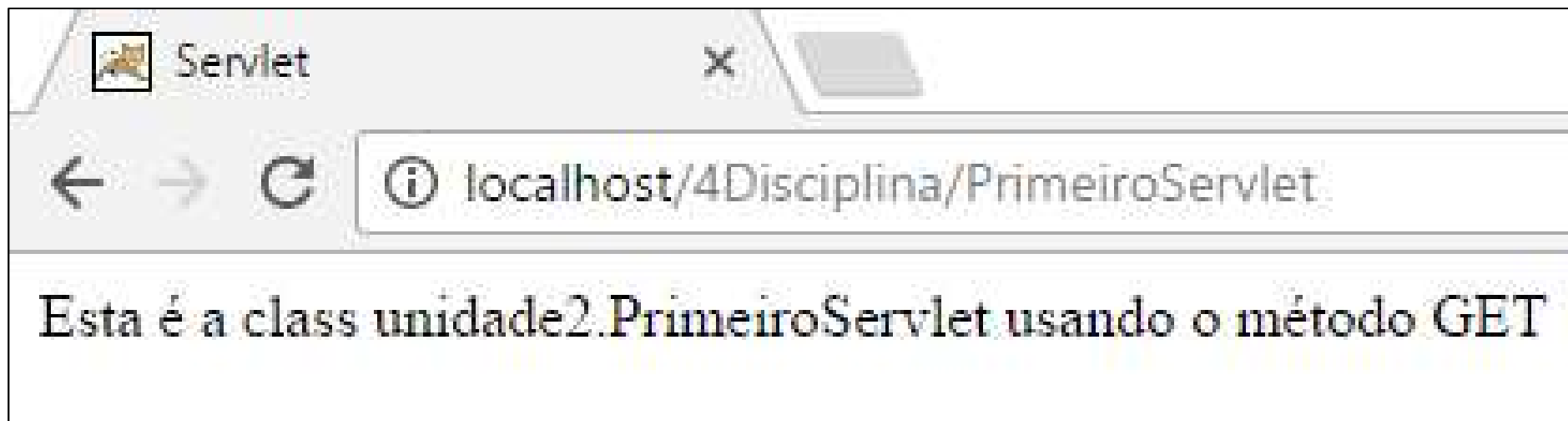
EXECUÇÃO DO SERVLET



Markers Properties Servers Data Source Explorer Snippets Console

Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_11\bin\javaw.exe (Mar 31, 2015, 3:34:07 PM)

```
Mar 31, 2015 3:34:26 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8085"]
Mar 31, 2015 3:34:26 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Mar 31, 2015 3:34:26 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1931 ms
```



ANATOMIA DO SERVLET

- Todo servlet deve importar classes e interfaces dos seguintes pacotes:
 - **import java.io.*;**
 - **import javax.servlet.*;**
 - **import javax.servlet.http.*;**
- Todo Servlet é sub-classe de HttpServlet.
 - **public class PrimeiroServlet extends HttpServlet**
- Diferente do que ocorre com uma aplicação Java desktop, o servlet não possui o método main(). Deve implementar um dos seguintes métodos: **service()**, **doGet()** e **doPost()**.
 - **public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException**

ANATOMIA DO SERVLET

- Os objetos **request** (**HttpServletRequest**) e **response** (**HttpServletResponse**) representam, respectivamente, a requisição do cliente para o servidor WEB e a resposta do servlet a esta requisição.
- Os métodos **service()**, **doGet()** e **doPost()** sempre “lançam” as exceções **ServletException** e **IOException**.
- O método **setContentType(...)** especifica o formato da resposta que a será enviado pelo servlet via objeto **response**. O padrão MIME padrão para páginas HTML é “text/html”.
 - **res.setContentType("text/html");**
- O método **println(...)** é utilizado para retornar dados do servlet para o cliente.
 - **PrintWriter out = response.getWriter();**
 - **out.println("<HTML>");**

Exercícios

1) [VUNESP - 2013 – FUNDUNESP] Considere o Servlet a seguir:

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

public class ClasseServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) {
        response.write("<html>");
        response.write("<body>");
        response.write("Servlet em operação!");
        response.write("</body>");
        response.write("</html>");
    }
}
```

2) Escrever o Servlet **PrimeiroServlet** que informe quantas vezes o mesmo é acessado pelo usuário.

Exercícios

- 1) Sobre o código do Servlet, é possível afirmar que:
 - a) ao ser executado por um contêiner de Servlet, será exibida uma tela em branco no navegador.
 - b) ao ser executado por um contêiner de Servlet, será exibida a mensagem “Servlet em operação!” na tela do navegador.
 - c) não pode ser compilado, pois a classe `HttpServletResponse` não possui o método `write`.
 - d) não pode ser compilado, pois `HttpServlet` é uma interface e, portanto, não pode ser estendida por uma classe.
 - e) o conteúdo exibido na tela do navegador não será codificado corretamente, pois a codificação da página não foi informada.
- 2) Escrever o Servlet **PrimeiroServlet** que informe quantas vezes o mesmo é acessado pelo usuário.

Exercícios

1) Sobre o código do Servlet, é possível afirmar que:

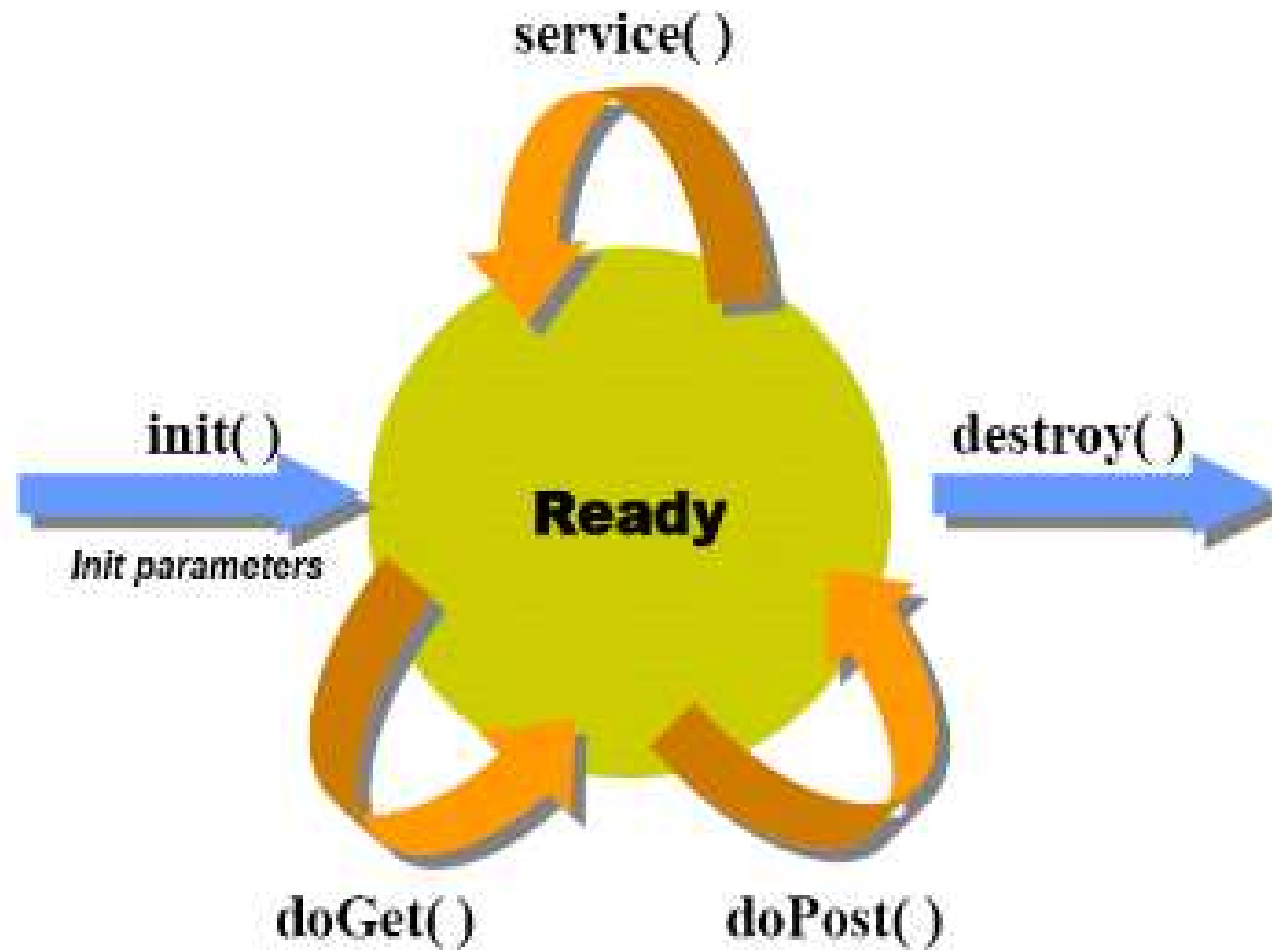
- a) ao ser executado por um contêiner de Servlet, será exibida uma tela em branco no navegador.
 - b) ao ser executado por um contêiner de Servlet, será exibida a mensagem “Servlet em operação!” na tela do navegador.
 - c) não pode ser compilado, pois a classe HttpServletResponse não possui o método write.**
 - d) não pode ser compilado, pois HttpServlet é uma interface e, portanto, não pode ser estendida por uma classe.
 - e) o conteúdo exibido na tela do navegador não será codificado corretamente, pois a codificação da página não foi informada.
- 2) Escrever o Servlet **PrimeiroServlet** que informe quantas vezes o mesmo é acessado pelo usuário.

Ciclo de Vida

CICLO DE VIDA

- Os servlets possuem um ciclo de vida bem definido que é gerenciado pelo servidor JEE.
- **Este ciclo é dividido em 05 etapas:**
 - 1) Carregar em memória o servlet especificado;**
 - 2) Criar a instância dele;**
 - 3) Chamar o método init() do servlet;**
 - 4) Chamar o método service(...) dele;**
 - 5) Chamar o método destroy() do servlet.**

CICLO DE VIDA



MÉTODO INIT()

- Este método é executado apenas uma vez, após o servlet ter sido carregado em memória e ter sido criado uma instância dele.
- Muito útil para realizar a leitura de dados de configuração.
- **Sintaxe:**

```
public void init () throws ServletException {  
    //code  
}
```

MÉTODO SERVICE()

- Após o servlet ter sido inicializado, o servidor WEB envia requisições a ele utilizando o método **service(...)**.
- Este método só é chamado quando há uma requisição.
- Ao receber uma requisição HTTP, o método **service(...)** é responsável por enviar a requisição recebida para o método 'equivalente' **doXXX(...)**.
- **Sintaxe:**

```
public void service (ServletRequest request, ServletResponse response)
    throws ServletException, IOException {
}
```

MÉTODO DOGET()

- Após o método **service(...)** ter sido executado, ele aciona o método **doGet(...)** quando receber dados passados pelo usuário via HTTP GET, isto é, parâmetros passados via URL.
- **Sintaxe:**

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

MÉTODO DOPOST()

- Após o método **service(...)** ter sido executado, ele aciona o método **doPost(...)** quando receber dados passados pelo usuário via HTTP POST, isto é, uma grande quantidade de dados é enviada por uma forma mais segura, no corpo da mensagem HTTP (invisível ao usuário comum), e não na sua URL (visível a qualquer usuário).
- É o padrão adotado para a passagem de dados de autenticação (login e senha) e dados fornecidos via formulários.
- **Sintaxe:**

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

MÉTODO DESTROY()

- Após uma aplicação WEB ter sido finalizada, o método **destroy()** é chamado.
- Este chamamento é para eliminar o servlet da memória do servidor de aplicação.
- Ao definir este método, as ações de fechamento de arquivos, fechamento de conexões de banco de dados, etc. deverão ser realizadas.
- **Sintaxe:**

```
public void destroy(){  
    //code  
}
```


Exercícios

1) [FCC - 2013 - DPE-SP] Um Servlet Contêiner controla o ciclo de vida de uma servlet onde são invocados três métodos essenciais: um para inicializar a instância da servlet, um para processar a requisição e outro para descarregar a servlet da memória. Os itens a seguir representam, nessa ordem, o que ocorre quando um usuário envia uma requisição HTTP ao servidor:

I. A requisição HTTP recebida pelo servidor é encaminhada ao Servlet Contêiner que mapeia esse pedido para uma servlet específica.

II. O Servlet Contêiner invoca o método `init` da servlet. Esse método é chamado em toda requisição do usuário à servlet não sendo possível passar parâmetros de inicialização.

III. O Servlet Contêiner invoca o método `service` da servlet para processar a requisição HTTP, passando os objetos `request` e `response`. O método `service` não é chamado a cada requisição, mas apenas uma vez, na primeira requisição do usuário à servlet.

Exercícios

1) [FCC - 2013 - DPE-SP]

IV. Para descarregar a servlet da memória, o Servlet Contêiner chama o método `unload`, que faz com que o garbage collector retire a instância da servlet da memória.

Está correto o que se afirma em

- a) I, II, III e IV.
- b) I, apenas.
- c) I e IV, apenas.
- d) II, III e IV, apenas.
- e) II e III, apenas.

2) Criar o servlet '**CicloVida**' que mostre no browser os seus 'estados'.

Exercícios

1) [FCC - 2013 - DPE-SP]

IV. Para descarregar a servlet da memória, o Servlet Container chama o método unload, que faz com que o garbage collector retire a instância da servlet da memória.

Está correto o que se afirma em

a) I, II, III e IV.

b) I, apenas.

c) I e IV, apenas.

d) II, III e IV, apenas.

e) II e III, apenas.

2) Criar o servlet '**CicloVida**' que mostre no browser os seus 'estados'.

Passagem de Parâmetros

PASSAGEM DE PARÂMETROS

- São 03 as principais formas que um cliente WEB tem para enviar parâmetros a um servlet:

- 1) Via Solicitações Comum (método 'get')
- 2) Via Formulário (método 'post')
- 3) Via Contexto ou Sessão do servidor JEE

- O tipo (1) não necessita de formulário, sendo que os parâmetros são enviadas como **Query String**.

URL = `http://localhost:8080/curso/SeuServlet?param1=valor1¶m2=valor2`



- A **Query String** inicia após o nome do recurso, seguido do '?'. Cada parâmetro é separado por '&'.

PASSAGEM DE PARÂMETROS

- O tipo (2) necessita de formulário, sendo que os parâmetros são enviados no corpo da mensagem, sem fazer uso da **Query String**.
- No tipo (1), o cliente não pode enviar uma QS maior que 240 caracteres. No tipo (2), não existe esta limitação.
- O tipo (2) é mais seguro, visto que os parâmetros não serão 'facilmente' visualizados.
- O método **getParameter(...)** é utilizado para receber os parâmetros enviados ao servlet.
 - **getParameter(String nomeParâmetro)**

PASSAGEM DE PARÂMETROS

- A sintaxe padrão para o **tipo (1)** está destacada abaixo:

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    String login = request.getParameter("login");
    String senha = request.getParameter("senha");
```

- A sintaxe padrão para o **tipo (2)** está destacada abaixo:

```
public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    String login = request.getParameter("login");
    String senha = request.getParameter("senha");
```

Exercício

1) Criar o servlet **ServletControlador** que recebe os dados **CPF** e **Senha** informados pelo arquivo **formulario.html**.



O formulário é uma caixa cinza com um ícone de três pessoas no topo. Abaixo do ícone, o texto "Área do aluno" é exibido em uma fonte maior, seguido por "Favor informar os dados abaixo" em uma fonte menor. Há dois campos de entrada: o primeiro é rotulado "CPF:" e o segundo "Senha:". Abaixo dos campos, há um botão cinza escuro com o texto "Entrar".

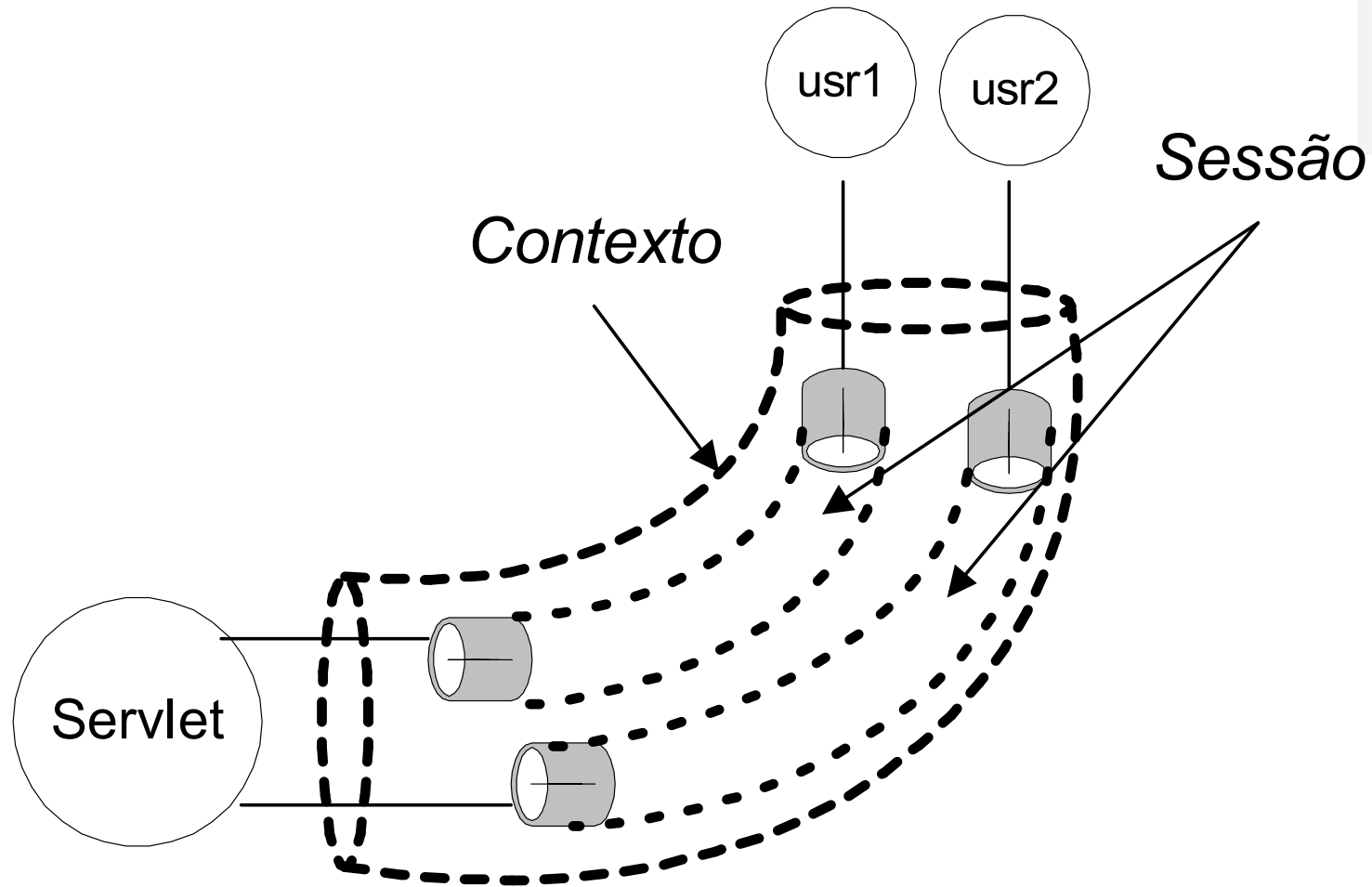
formulario.html

Contexto e Sessão

MODELO SERVLET CONTAINER

- Todo servlet faz parte de um único contexto e utiliza várias sessões (uma para cada usuário).
- O objeto contexto é uma instância da interface **ServletContext**. Para toda aplicação Web, existe um único objeto Contexto!
- Todo cliente WEB de um servlet possui a sua própria sessão. Uma sessão pode ser compreendida como o local onde todas as ações realizadas por um determinado usuário podem ser acessadas.
- Uma sessão é criada quando o usuário envia a sua primeira requisição para um servlet. O objeto sessão é uma instância da interface **HttpSession**.

MODELO SERVLET CONTAINER



MODELO SERVLET CONTAINER

ATRIBUTOS

- Um atributo é um objeto que é incorporado a um contexto ou sessão de um servlet.
- Os métodos associados a um atributo são:
 - **void setAttribute(String nome_atributo, Object valor)**
 - **Object getAttribute(String nome_atributo)**
 - **void removeAttribute(String nome_atributo)**
- Um atributo é incluído no contexto/sessão pelo uso do método **setAttribute(...)**.
- O valor de um atributo é obtido de um contexto/sessão pelo uso do método **getAttribute(...)**.

ATRIBUTOS NO CONTEXTO

INCLUSÃO DE UM ATRIBUTO

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    ServletContext context = getServletContext();
    context.setAttribute("cpf", cpf);
```

OBTENÇÃO DE UM ATRIBUTO

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    ServletContext context = getServletContext();
    String cpf = context.getAttribute("cpf");
```

ATRIBUTOS NA SESSÃO

INCLUSÃO DE UM ATRIBUTO

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    HttpSession session = request.getSession();
    session.setAttribute("cpf", cpf);
```

OBTENÇÃO DE UM ATRIBUTO

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    HttpSession session = request.getSession();
    String cpf = session.getAttribute("cpf");
```

DESTRUIÇÃO DE UMA SESSÃO

- Após um período de inatividade, a sessão é destruída. Essa destruição pode ser feita de duas formas:
 - Via arquivo de configuração **web.xml**
 - Via uso dos métodos **public void setMaxInactiveInterval (int sec)** ou **public void invalidate()**

```
<web-app>
...
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
</web-app>
```

```
HttpSession.setMaxInactiveInterval(15);
HttpSession.invalidate();
```

Exercícios

- 1) Realizar as seguintes operações no Servlet **ServletControlador**:
 - Armazenar o nome do componente (**this.getClass()**) no contexto da aplicação;
 - Armazenar os valores de **CPF** e **Senha** passados na sessão do usuário.
- 2) Criar o Servlet **ServletLeitor** para obtenção desses dados armazenados no contexto da aplicação e na sessão do usuário.

Encaminhamento/ Redirecionamento de Requisições e Tratamento de Exceção

ENCAMINHAMENTO

- Uma tarefa realizada com frequência em uma aplicação WEB é o encaminhamento de uma requisição para ser atendida por outro recurso. Utiliza-se o método **forward(...)** da interface **RequestDispatcher**.
- O **forward(...)** serve para transferir o controle para outro componente web (servlet/JSP/JSF). Existe outra opção (**include(...)**), que não encaminha mas inclui o conteúdo informado.

```
String uri = "servlet/ServletTwo";  
RequestDispatcher dispatcher =  
request.getRequestDispatcher(uri);  
dispatcher.include(request, response);  
dispatcher.forward(request, response);
```

- O método **include(...)** mantém o controle do servlet “chamador” na saída de informações. O método **forward(...)** não mantém este controle.

REDIRECIONAMENTO

- É possível redirecionar uma requisição fazendo uso do método **sendRedirect(String URL)**.
- Este método está definido na interface **HttpServletResponse**.

```
response.sendRedirect("pagina.html");
```

- No caso do **sendRedirect(...)**, o cliente receberá uma resposta http em cujo cabeçalho haverá a informação de que ele deve requisitar outra página, e o browser fará esta requisição. Ou seja, o redirecionamento ocorre no lado no cliente.
- No caso do **forward(...)**, o servidor irá encaminhar a requisição do usuário para ser atendida por outro recurso (outro servlet). Este outro servlet eventualmente devolverá outra página para o usuário.
- A diferença é bem grande principalmente no que tange à experiência do usuário.

TRATAMENTO DE EXCEÇÃO

- No mundo real, todas as aplicações podem falhar. Neste caso, o servidor WEB envia mensagens de erro. Essas mensagens podem ser tratadas das seguintes formas:
 - 1) Mensagem de erro enviada pelo servidor
 - 2) Mensagem de erro enviada pelo servidor com os métodos `sendError(...)` e `setStatus(...)`
 - 3) Criação de uma página de erro padrão
 - 4) Criação de uma página de erro dinâmica
- Abaixo, um exemplo do **tipo (1)**.

```
try{
    arq = getFile(file);
}
catch(FileNotFoundException e){
    out.println("arquivo não encontrado");}
```

TRATAMENTO DE EXCEÇÃO

- Abaixo, um exemplo do **tipo (2)**.

```
try{
    arq = getFile(file);
}
catch(FileNotFoundException e){
    response.sendError(res.SC_NOT_FOUND,
        "O arquivo não foi encontrado");}
```

- Outro exemplo do **tipo (2)**.

```
try{
    arq = getFile(file);
    response.setStatus(res.SC_OK);
} catch(FileNotFoundException e) {
    response.sendError(res.SC_NOT_FOUND,
        "O arquivo não foi encontrado");}
```

TRATAMENTO DE EXCEÇÃO

- Abaixo, um exemplo do **tipo (3)** 'erro404.html'.

```
<HTML>
  <HEAD>
    <TITLE>Página de Erro</TITLE>
  </HEAD>
  <BODY>
    <H1>Documento não encontrado</H1>
  </BODY>
</HTML>
```

- Alteração (necessária) no arquivo **web.xml**.

```
<web-app>
  <error-page>
    <error-code>404</error-code>
    <location>/erro/404.html</location>
  </error-page>
</web-app>
```

TRATAMENTO DE EXCEÇÃO

- No **tipo (4)** é necessário utilizar dois atributos de erro pré-definidos:
 - **javax.servlet.error.status_code**
 - **javax.servlet.error.message**
- Além disso, é necessário alterar o arquivo web.xml.
 - **<location>/erro/ErrorServlet</error-code>**
- Exemplo do **ErrorServlet**

```
public void doGet(HttpServletRequest req,
    HttpServletResponse res) throws ServletException,
    IOException
{
    Integer code = (Integer) req.getAttribute
        ("javax.servlet.error.status_code");
    String msg = (String) req.getAttribute
        ("javax.servlet.error.message");
    out.println("<BODY><H1>" + code + "</H1>");
    out.println("<BODY><H1>" + msg + "</H1>");
}
```

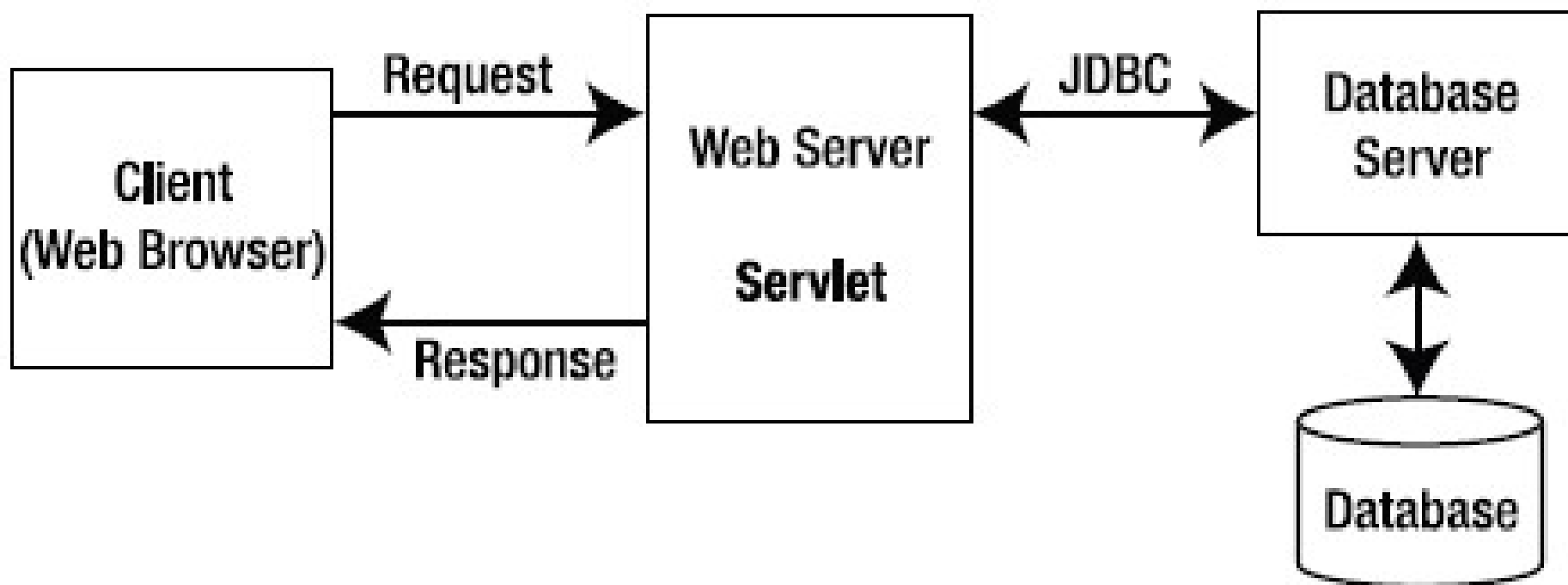
Exercícios

- 1) Utilizar o método **forward(...)** no **ServletControlador** para chamar o componente JEE **ServletLeitor**.
- 2) Criar as páginas erro **404.html** e **500.html** no projeto **4Disciplina**. Simular a chamada desses erros.

Servlet e JDBC

Servlet e JDBC

- Todo componente WEB (Servlet, JSP, JSF) irá fazer uso das informações armazenadas em um SGBD.



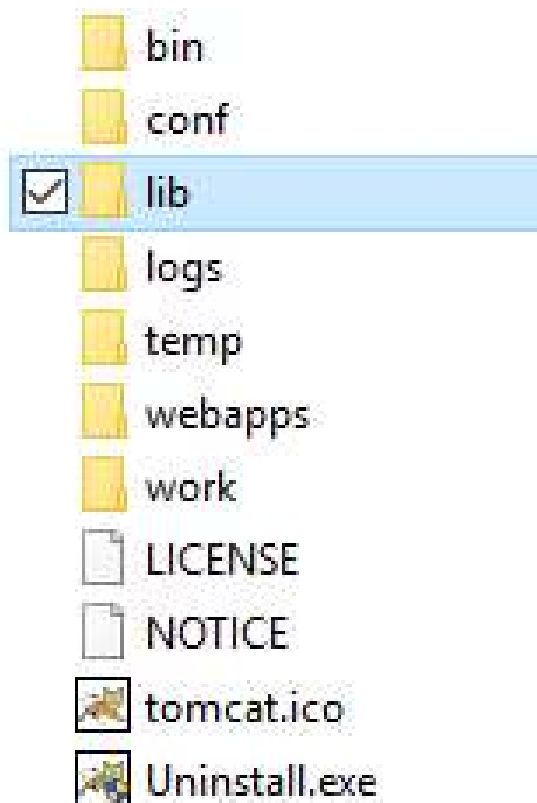
- Todos os conceitos vistos nas aplicações Java Desktop (JSE) para fazer uso de um SGBD, via JDBC 'puro' ou via frameworks de persistência ORM (JPA/Hibernate), serão utilizados de forma idêntica nos componentes WEB.

Servlet e JDBC

- Para executar qualquer operações no SGBD escolhido, é requisito indispensável de publicar o driver jdbc (no nosso caso, **ojdbc6.jar**) no projeto WEB (pasta **lib**) ou no servidor JEE (também na pasta **lib**).



Projeto WEB



Servidor Tomcat

Exercício

- 1) Renomear a página **formulariov2.html** para **login.html** e fazer a autenticação do usuário via **ServletAutenticador**.
 - Criar no esquema CursoJava a tabela Login (#cpf,senha);
 - Fazer a checagem nessa tabela via JDBC.

Formatos de Compressão

FORMATOS DE COMPRESSÃO

- O Java oferece três formatos para a compressão de arquivos:
 - **1) JAR (Java ARchive)**
 - **2) WAR (Web ARchive)**
 - **3) EAR (Enterprise ARchive)**
- **JAR**
- O Java adotou o formato JAR como padrão para facilitar a distribuição e utilização de bibliotecas de classes. O JDK fornece o aplicativo **jar** que gera arquivos neste formato.
- Um JAR é um arquivo compactado no formato ZIP que contém um conjunto de classes (".class") e arquivos de configuração.

Criar arquivo: `jar cvf arq.jar *.class`

Extrair arquivo: `jar xvf arq.jar`

FORMATOS DE COMPRESSÃO

- **WAR**
- Para a distribuição de aplicações WEB, o formato utilizado é o WAR. Este contém as classes necessárias, os arquivos de configuração e a estrutura de diretórios.

Criar arquivo: `jar -cfv ShoppingCart.war *`

Extrair arquivo: `jar -xvf ShoppingCart.war`

- **EAR**
- Para a distribuição de aplicações JEE, o formato utilizado é o EAR. Este contém um conjunto de arquivos (.jar) e (.war).

Criar arquivo: `jar -cfv ShoppingCart.war *`

Extrair arquivo: `jar -xvf ShoppingCart.war`

Exercícios

- 1) [CESGRANRIO - 2013 – BNDES] Ao instalar uma aplicação Java EE, a ferramenta de instalação deve ler o descritor de instalação de aplicação do arquivo .ear da aplicação, que é encontrado em
 - a) META-INF/application.xml
 - b) META-INF/application-client.xml
 - c) META-INF/deployment.xml
 - d) META-INF/ejb-jar.xml
 - e) META-INF/MANIFEST.ML.
- 2) Exportar o projeto **4Disciplina** para o formato .WAR (**4Disciplina.war**).

Exercícios

- 1) [CESGRANRIO - 2013 – BNDES] Ao instalar uma aplicação Java EE, a ferramenta de instalação deve ler o descritor de instalação de aplicação do arquivo .ear da aplicação, que é encontrado em
 - a) **META-INF/application.xml**
 - b) META-INF/application-client.xml
 - c) META-INF/deployment.xml
 - d) META-INF/ejb-jar.xml
 - e) META-INF/MANIFEST.ML.
- 2) Exportar o projeto **4Disciplina** para o formato .WAR (**4Disciplina.war**).

Projeto Prático (Parte 2)

Projeto Prático

Criar um Servlet que gerencie todas as Operações realizadas em **Clientes, **Cursos** e **Pagamentos****

>> CLIENTES <<

Consultar Todos os Clientes

Consultar um Cliente Específico

Cadastrar um Novo Cliente

Alterar um Cliente

Excluir um Cliente

>> CURSOS <<

Consultar Todos os Cursos

Consultar um Curso Específico

Cadastrar um Novo Curso

Alterar um Curso

Excluir um Curso

>> PAGAMENTOS <<

Consultar Todos os Pagamentos

Consultar um Pagamento Específico

Cadastrar um Novo Pagamento

Alterar um Pagamento

Excluir um Pagamento

Exercícios

- 1) Criar as páginas de erro padrão **404** e **500** no **ProjetoWEB**.
- 2) Criar o Servlet **Controlador** no **ProjetoWEB** que trate de todas as operações de **CRUD** realizadas nos **Clientes**, **Cursos** e **Pagamentos**.

RESUMO

TÓPICOS APRESENTADOS

- Nesta aula nós estudamos:
 - **Componentes WEB JEE**
 - **Primeiro Servlet**
 - **Ciclo de Vida**
 - **Passagem de Parâmetros**
 - **Contexto e Sessão**
 - **Encaminhamento/Redirecionamento de Requisições e Tratamento de Exceção**
 - **Formatos de Compressão**
 - **Projeto Prático (Parte 2)**

ATIVIDADES PARA SE APROFUNDAR

- **1) Criar e Aplicar no projeto WEB visto nesta unidade, os outros três formatos de tratamento de exceção apresentados.**
 - **1) Mensagem de erro enviada pelo servidor**
 - **2) Mensagem de erro enviada pelo servidor com os métodos `sendError(...)` e `setStatus(...)`**
 - **3) Criação de uma página de erro dinâmica**
- **2) Faça um estudo e identifique quais são as principais páginas de erro de um servidor WEB que precisam ser modificadas.**