

# Programação de Aplicações Corporativas

Arquitetura MVC em Java Legado  
sem a utilização de Frameworks

# Agenda

- Arquitetura dos Laboratórios;
- Instalação e configuração da infraestrutura
- Introdução ao MVC
- Criar, configurar e testar um novo projeto no VSCode
- JSP
- Servlets
- Java Beans

# Arquitetura do Laboratório INFO1

- Windows 7
- JDK 21
- Maven 3.9.9
- TomCat 10.1.39
- **MySQL (XAMP)**
- VSCode 1.70.2 (Última versão para Windows 7)
  - Extension Pack for Java
- Subdiretório de instalação dos aplicativos
  - Geral: c:\dev\pac2025
  - Maven: c:\dev\pac2025\maven
  - TomCat c:\dev\pac2025\tomcat

# Configurando Variáveis - Windows 10

## Configurar Variáveis de Ambiente

Acessar Propriedades do Sistema

Acessar **Variáveis de Ambiente**

## Configurar as Variáveis

JAVA\_HOME c:\Program Files\Java\jdk-21

MAVEN\_HOME c:\dev\pac\2025\maven

CATALINA\_HOME c:\dev\pac\2025\tomcat

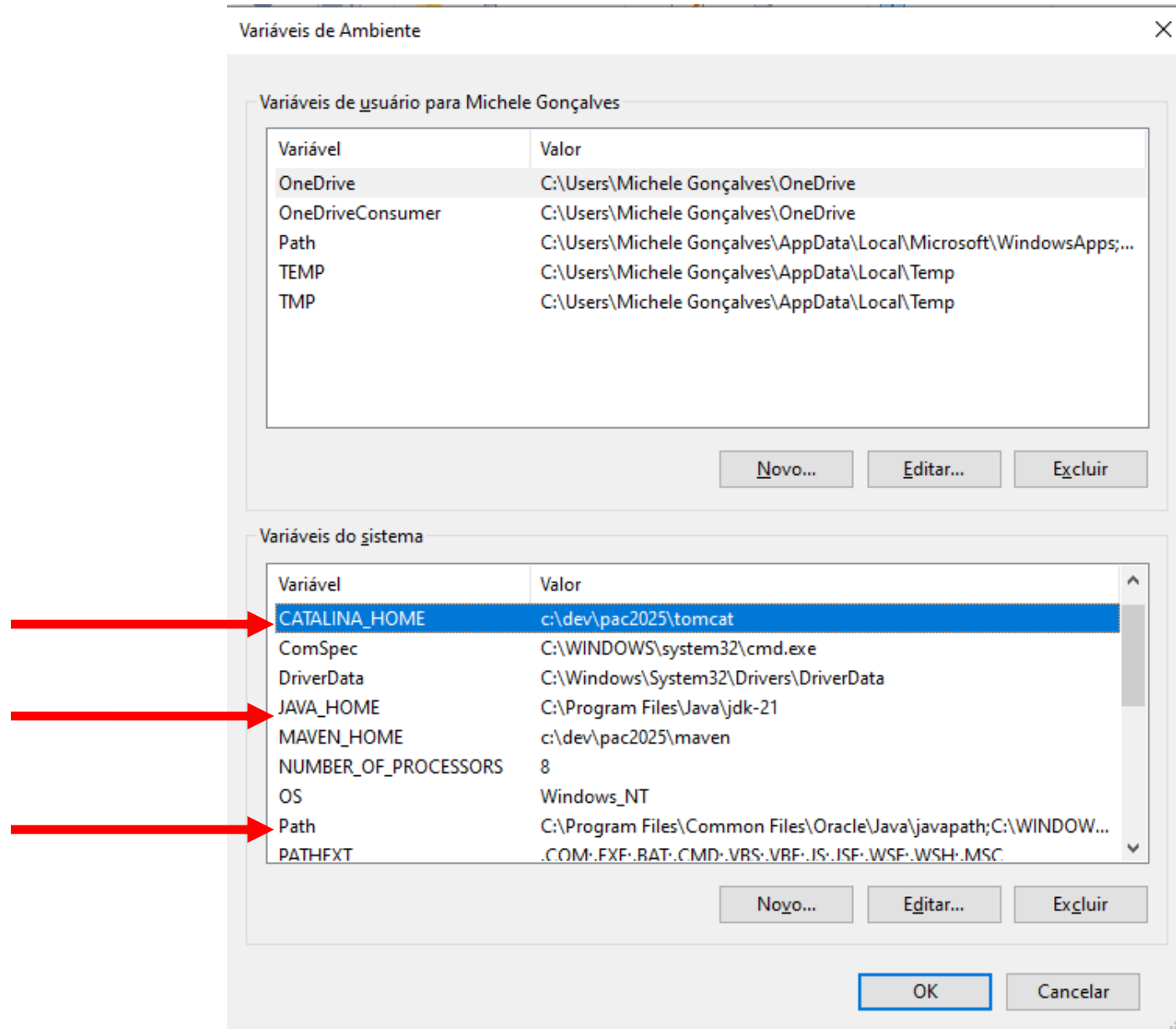
Path

c:\Program Files\Java\jdk-21\bin

c:\dev\pac\2025\maven\bin

c:\dev\pac\2025\tomcat\bin

# Configurando Variáveis - Windows 10



# Instalando a infraestrutura

## **Instalar o Java - Executar o aplicativo de instalação**

Adicionar o path C:\Program Files\Java\jdk-21\bin as variáveis do sistema

Adicionar a variável JAVA\_HOME C:\Program Files\Java\jdk-21

## **Descompactar o Maven em c:\dev\pac2025\**

Adicionar o path c:\dev\pac\2025\maven\bin as variáveis do sistema

Adicionar a variável MAVEN\_HOME c:\dev\pac\2025\maven

## **Descompactar o Tomcat em c:\dev\pac2025\**

Adicionar o path c:\dev\pac\2025\tomcat\bin as variáveis do sistema

Adicionar a variável CATALINA\_HOME c:\dev\pac\2025\tomcat

**Você terá que renomear os subdiretórios para**

**c:\dev\pac2025\maven**

**c:\dev\pac2025\tomcat**

# Testando a infraestrutura “Executar Individualmente”

**No terminal, digite:**

**Java**

`java -version`

`javac -version`

Verificar se a versão está na 21

**Maven**

`mvn -version`

Verificar se a versão está na 3.9.9

**Levantar o TOMCAT**

No Terminal, execute

`startup.bat`

**Parar a execução**

`shutdown.bat`

**Parar a execução de qualquer aplicativo no terminal  
ctrl + c**

# Maven

- É uma ferramenta para gerenciar projetos Java e outras linguagens;
- É responsável por configurar e baixar todas as dependências externas utilizadas no projeto;
- Ele automatiza a compilação/deploy/publicação do sistema;
- O arquivo pom.xml contém todas as dependências utilizadas no projeto.



# Maven (pom.xml)

Em <properties>, configure a codificação e a versão do Java.

```
<properties>  
<project.build.sourceEncoding>UTF8</project.build.sourceEncoding>  
  <maven.compiler.source>21</maven.compiler.source>  
  <maven.compiler.target>21</maven.compiler.target>  
</properties>
```

# MAVEN (pom.xml)

Incluir as dependências dentro da TAG <dependencies> para Servlet, JSTL, MySQL;

```
<dependency>
  <groupId>jakarta.servlet</groupId>
  <artifactId>jakarta.servlet-api</artifactId>
  <version>6.0.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>3.0.1</version>
</dependency>
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>9.2.0</version>
</dependency>
```

# MAVEN (Comandos Básicos - Terminal)

`mvn -version` (Verifica a versão do Maven)

`mvn clean` (Remove arquivos compilados)

`mvn compile` (Compila o código-fonte)

`mvn test` (Executa os testes)

**`mvn package` (Gera um JAR ou WAR\*)**

`mvn install` (Instala no repositório local)

`mvn dependency:tree` (Exibe dependências)

\*Um **arquivo WAR (Web Application Archive)** é um pacote compactado no formato **.war** que contém todos os arquivos necessários para executar uma aplicação web Java.

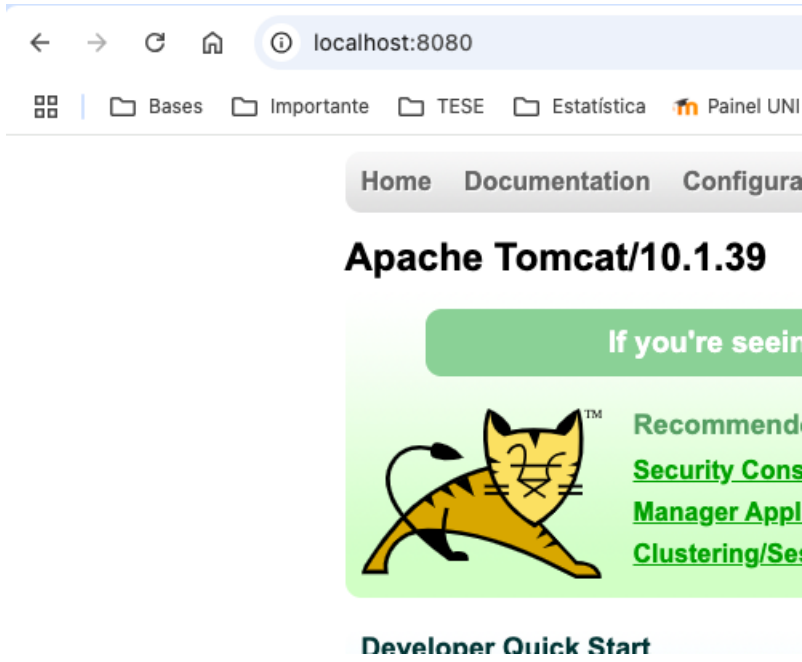
# Tomcat

- O **Apache Tomcat** é um servidor web e contêiner de servlets open-source desenvolvido pela Apache Software Foundation. Ele é usado principalmente para executar aplicações web Java baseadas nas especificações **Servlet**, **JavaServer Pages (JSP)** e **WebSockets**.

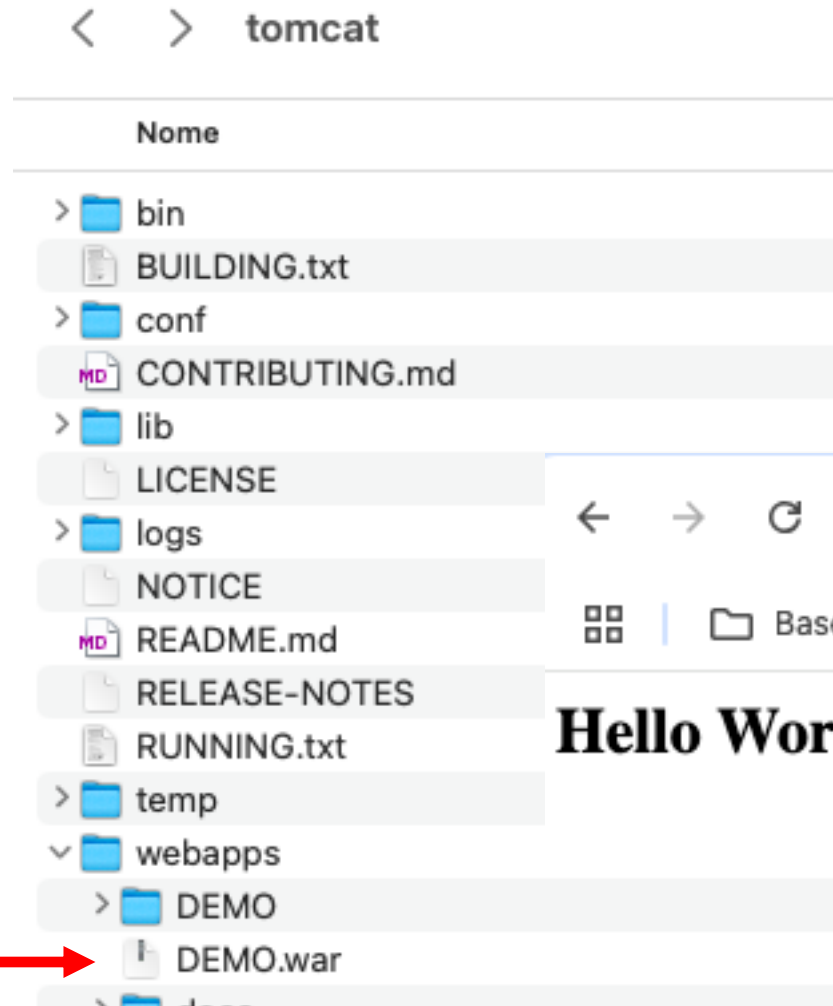
## Principais Características do Tomcat

- **Servidor Web para Java:** Processa requisições HTTP e executa código Java em aplicações web.
- **Contêiner de Servlets e JSP:** Permite a execução de servlets (código Java que processa requisições) e páginas JSP (Java Server Pages).
- **Leve e Rápido:** Comparado a servidores mais completos como o JBoss/WildFly, o Tomcat é mais leve e focado no essencial.
- **Suporte a WebSockets:** Possui suporte nativo para comunicação assíncrona via WebSockets.
- **Integração com Spring Boot:** O Tomcat pode ser embutido em aplicações **Spring Boot**, permitindo que rodem como um JAR executável.

# Tomcat



Tomcat está no iniciado.



**Hello World!**

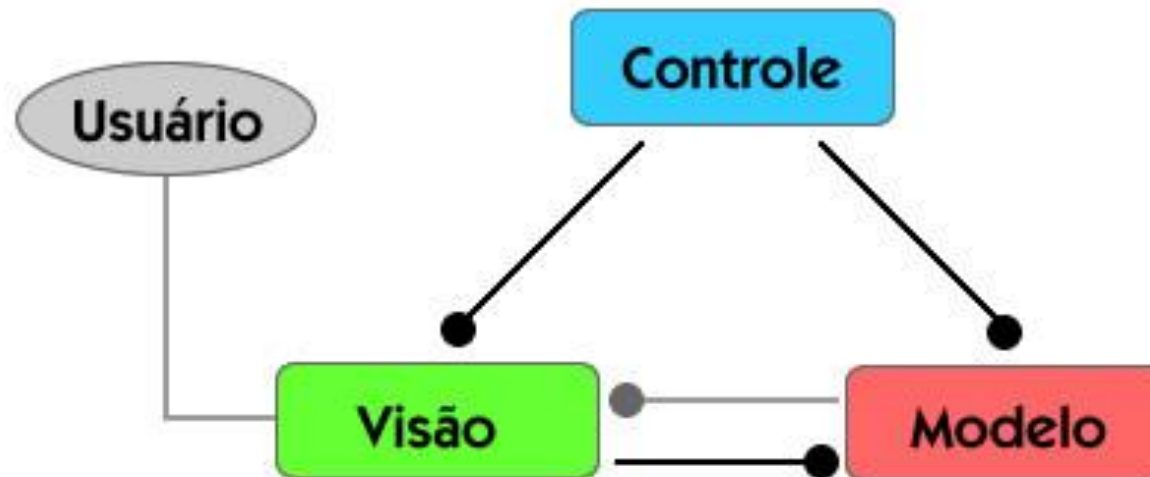
Quando copiamos um arquivo .war para o subdiretório webapps, o site é publicado.

# MVC – O que é?

- Em um sistema de fácil manutenção as responsabilidades estão implementadas em arquivos separados.

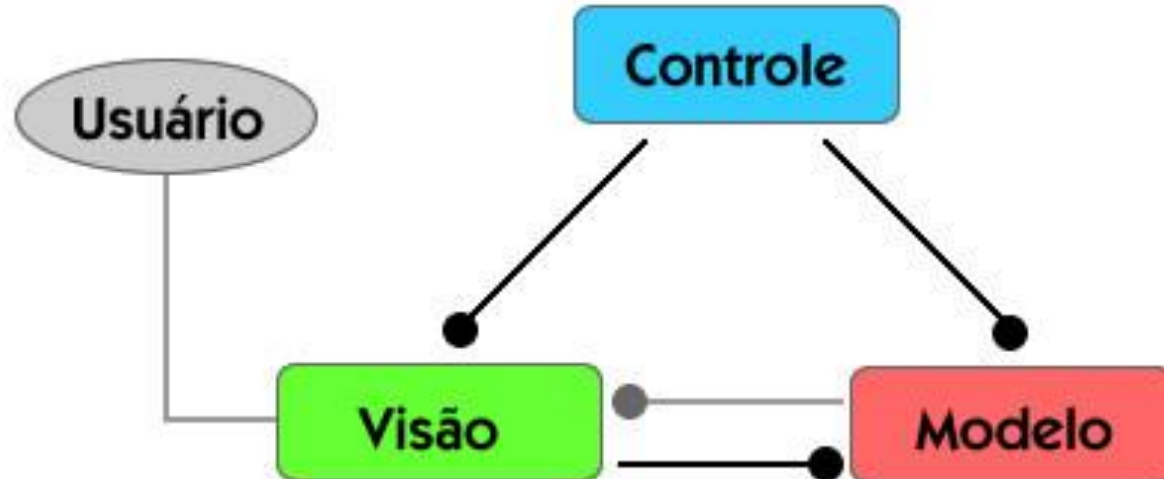
Ex.:

- O JSP/HTML (View) implementa funcionalidades de tela (entrada e saída);
- O acesso ao banco está separado de uma lógica que testa se o usuário pode excluir ou digitar uma informação (Controller).



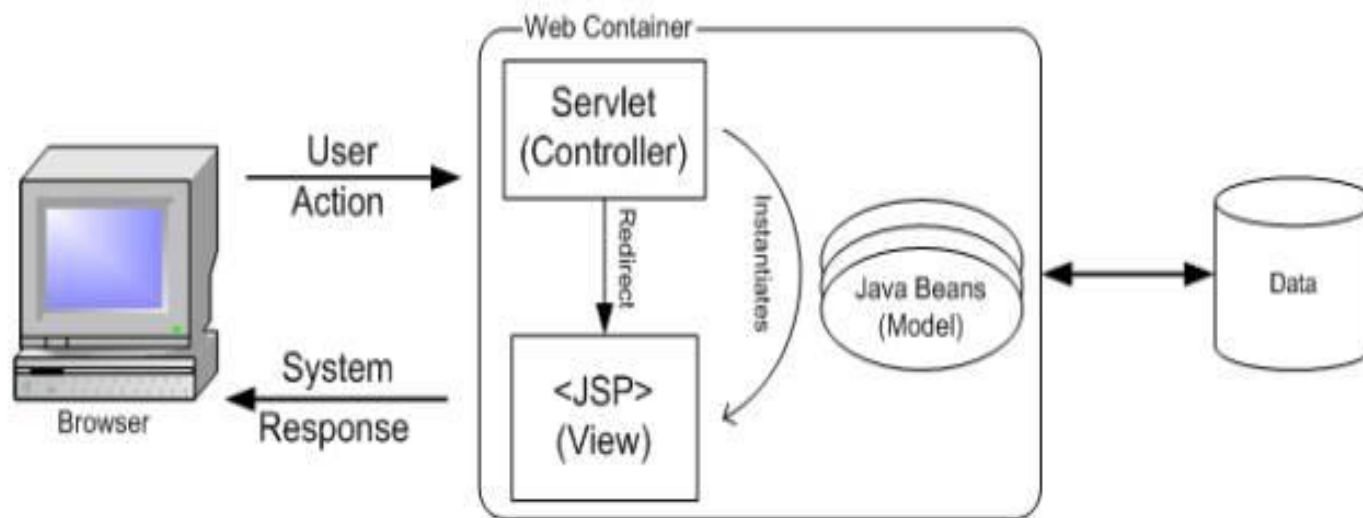
# MVC – O que é?

- O MVC (Model – View – Controller) é um padrão de arquitetura muito utilizado no desenvolvimento de projetos WEB;
- O objetivo é facilitar a separação da layer visual da de negócio;
- Um sistema com MVC está separado em três camadas. Cada camada é responsável por implementar códigos em seu nível de atuação;
- Cada camada prepara as informações para outra.
- Cada camada deve interagir com o nível mais próximo.



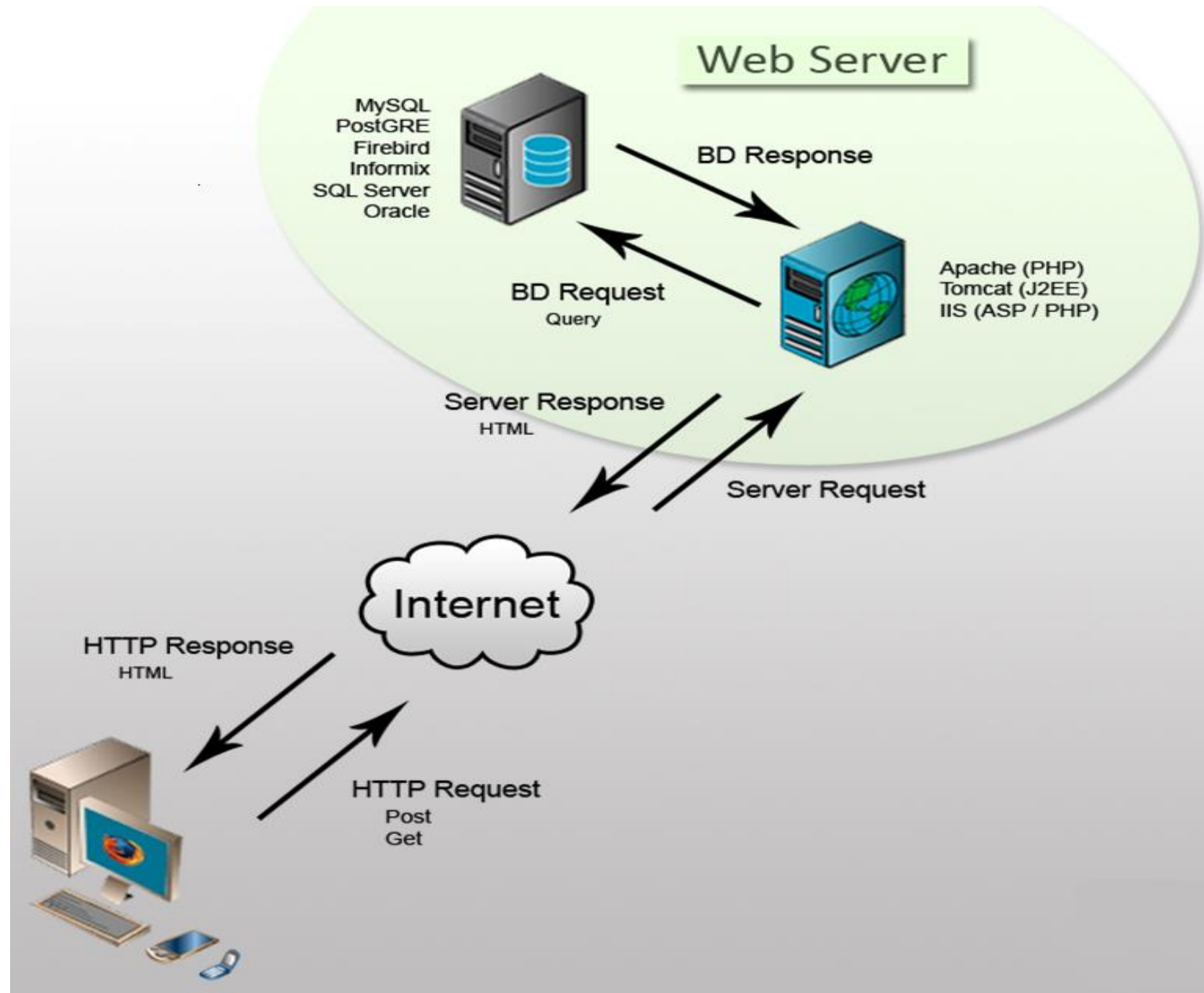
# MVC – O que é?

- **Model:** modelo de dados, camadas de acesso e algumas regras do negócio;
- **View:** representa tudo que envolve a interface do sistema (HTML, JSP e JS);
- **Controller:** Conecta a view a model.
  - Responsável por receber dados da view, processá-los e enviá-los a model;
  - Responsável por receber dados da model e enviar a view;

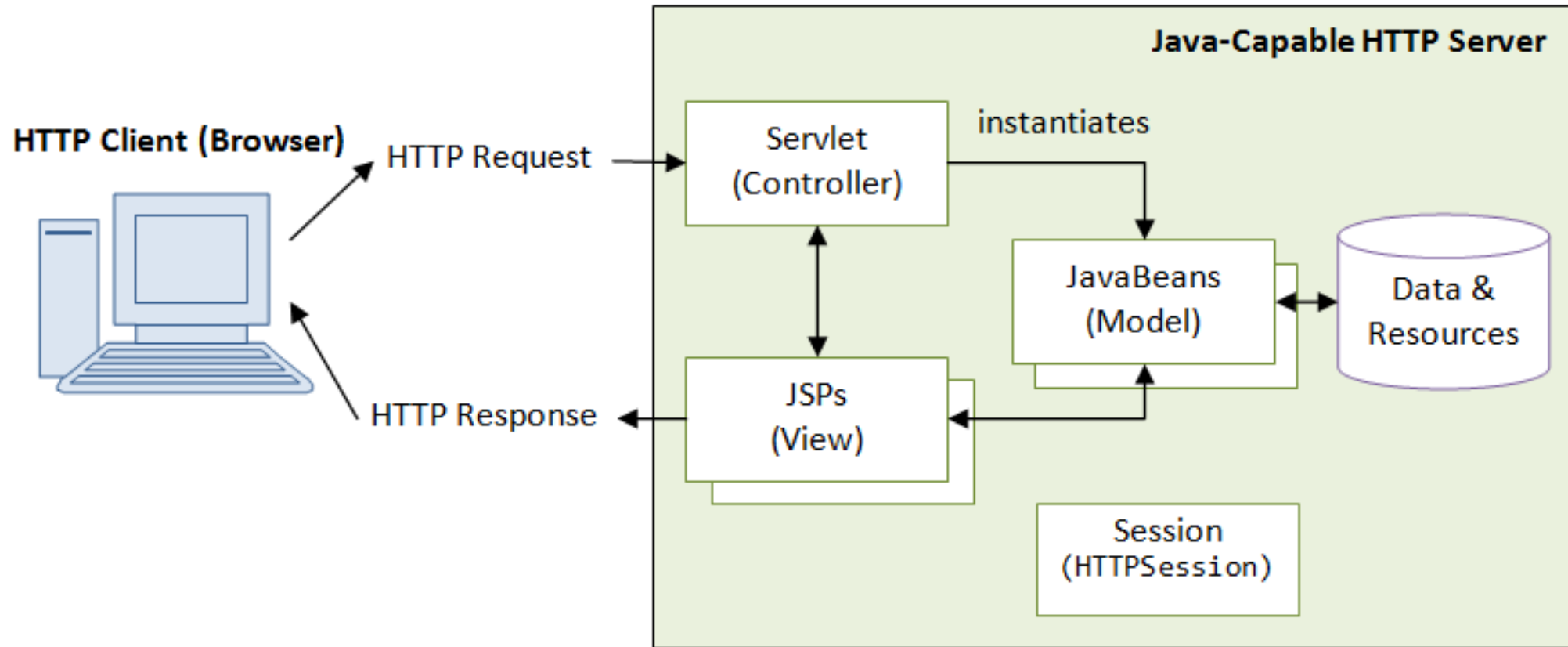




# Arquitetura



# Arquitetura Básica Java EE (Legado)



**JSP (Java Server Page):** Páginas HTML + Scriptlet + Tags Java;

**Servlet (requisições servidor):** Gera conteúdo para páginas dinâmicas;

**JavaBean:** Classe Java que representa uma entidade (Construtores, gets e sets);

# Estrutura padrão de subdiretórios e arquivos

```
meu-projeto-javaee/  
|— src/  
|   |— main/  
|   |   |— java/  
|   |   |   |— com.exemplo.servlet/  
|   |   |   |   |— MeuServlet.java  
|   |   |— webapp/  
|   |   |   |— WEB-INF/  
|   |   |   |   |— web.xml  
|   |   |   |— index.jsp  
|— pom.xml
```

**src (Source):** Contém as pastas do Código Java e HTML;

**java:** subdiretório para código Java

**com.exemplo.servlet:** Pacote

**MeuServlet.java:** Arquivo WEB de controller

**webapp:** Pasta dos arquivos WEB (HTML, JSP, Servlet)

**index.jsp:** HTML + Script Java

```
meu-projeto-javaee
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── com.exemplo.servlet (Pacote para Servlets)
│   │   │   │   ├── MeuServlet.java
│   │   │   ├── resources (Arquivos de configuração, se necessário)
│   │   │   └── webapp (Raiz da aplicação web)
│   │   └── WEB-INF (Arquivos internos do servidor, não acessíveis diretamente pelo usuário)
│   │       ├── web.xml (Arquivo de configuração da aplicação)
│   │       ├── lib (Bibliotecas externas)
│   │       └── views (Arquivos JSP)
│   │           ├── index.jsp
│   │           └── login.jsp
│   └── assets (Imagens, CSS, JavaScript)
│       ├── css
│       │   ├── styles.css
│       │   └── js
│       │       └── scripts.js
└── target (Gerado pelo Maven após a compilação)
    └── pom.xml (Configuração do Maven)
```

# Estrutura de diretórios e arquivos

src

**src/main/java/com.exemplo.servlet**  
→ Contém as classes Java, incluindo os Servlets.

**src/main/webapp** → Pasta raiz da aplicação web.

**WEB-INF** → Contém arquivos privados, como web.xml e bibliotecas externas.

**views** → Contém os arquivos JSP usados na camada de apresentação.

**assets** → Contém arquivos estáticos como CSS, JS e imagens.

**pom.xml** → Arquivo do Maven para gerenciamento de dependências.

# Habilidades essenciais no VSCode

# Habilidades essenciais no VSCode

Criar, configurar e testar um novo projeto

1. Criar o novo projeto;
2. Configurar o pom.xml;
3. Configurar o web.xml;
4. Criar um arquivo HTML/JSP;
5. Criar um pacote;
6. Criar um servlet e
7. Realizar o deploy.



Repita as  
operações 5x.

# 1. Criar um Novo Projeto

- Criar uma pasta em c:\Users\Aluno\Documents\Seu\_nome
- Localizar o arquivo “code.exe”;
- File / Open Folder
- File / New File
  - New Java Project
  - Maven
  - Maven-archetype-webapp
  - version 1.4
- No terminal,
  - com.exemplo: Package/url invertida da empresa (Altere para **com.cefet**)
  - demo: Artifact id (nome do projeto) (Crie e selecione para **exemplo1**)
  - crie\selecione um subdiretório (pasta) para o seu projeto; (Crie **exemplo1**)

# 1. Criar um Novo Projeto (OU)

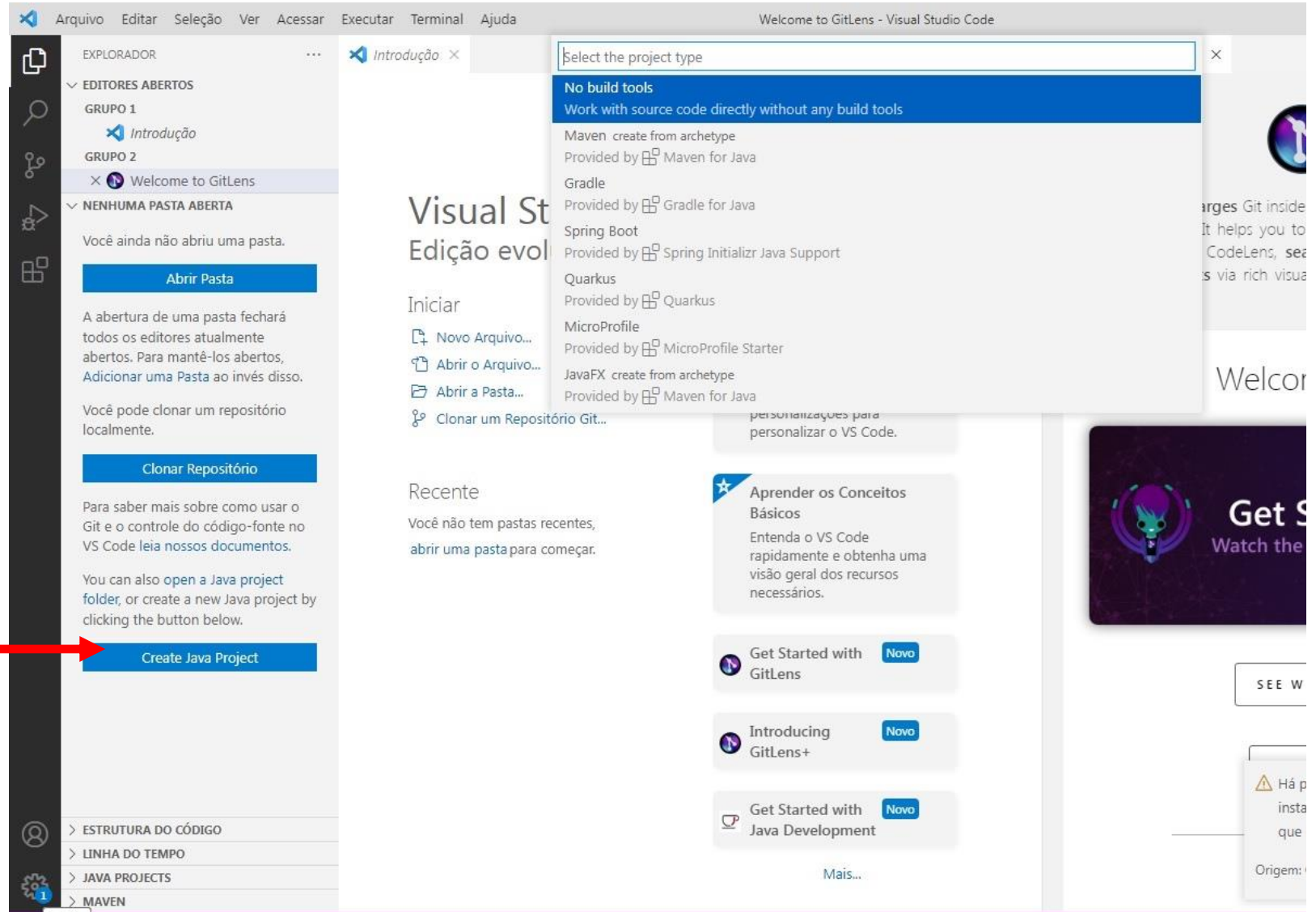
**Ctrl + Shift + P**

Digitar “Java”

Create Java Project  
Maven....

**Ou**

Create Java Project





# 1. Criar um Novo Projeto

```
[INFO]
[INFO]
[INFO] --- archetype:3.1.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] Archetype repository not defined. Using the one from [org.apache.maven.archetypes:maven-archetype-webapp:1.4]
[INFO] Using property: groupId = br.cefet
[INFO] Using property: artifactId = exemplo1
Define value for property 'version' 1.0-SNAPSHOT: :
[INFO] Using property: package = br.cefet
Confirm properties configuration:
groupId: br.cefet
artifactId: exemplo1
version: 1.0-SNAPSHOT
package: br.cefet
Y: : Y
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-webapp:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: br.cefet
```

No terminal, <<enter>> para confirmar a estrutura e “Y” para confirmar.

## 2. Configurar o pom.xml

Em <properties>, configure a codificação e a versão do Java.

```
<properties>  
<project.build.sourceEncoding>UTF8</project.build.sourceEncoding>  
<maven.compiler.source>21</maven.compiler.source>  
<maven.compiler.target>21</maven.compiler.target>  
</properties>
```

No plugin de compilação do Maven, inclua a release 21.

```
<plugin>  
<artifactId>maven-compiler-plugin</artifactId>  
<version>3.8.0</version>  
<configuration>  
    <release>21</release>  
</configuration>  
</plugin>
```

## 2. Configurar o pom.xml

Incluir as dependências dentro da TAG <dependencies> para Servlet, JSTL e MySQL;

```
<dependency>
  <groupId>jakarta.servlet</groupId>
  <artifactId>jakarta.servlet-api</artifactId>
  <version>6.0.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>jakarta.servlet.jsp.jstl</groupId>
  <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
  <version>3.0.0</version>
</dependency>
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>3.0.1</version>
</dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>9.2.0</version>
  </dependency>
```

# 3. Configurar o web.xml\*

Selecione o arquivo src/webapp/WEB-INF/web.xml;

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

<web-app
xmlns="https://jakarta.ee/xml/ns/jakartaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
version="6.0">
</web-app>
```

**\* Configuração que faz o projeto aceitar as JSTL na versão JAKARTA.**

## 4. Criar um HTML

Selecione a pasta webapp;

Botão direito, New File;

Nomeie para exemplo1.html;

Dentro do arquivo, digite “!” e espaço;

Verifique o lang “pt-br” e o charset “utf-8”;

No body, digite

```
<p> Olá mundo! Exemplo HTML </p>
```

## 5. Criar um JSP

Selecione a pasta webapp;

Botão direito, New File

Nomeie para exemplo1.jsp

Dentro do arquivo, digite “!” e espaço

Verifique o lang “pt-br” e o charset “utf-8”;

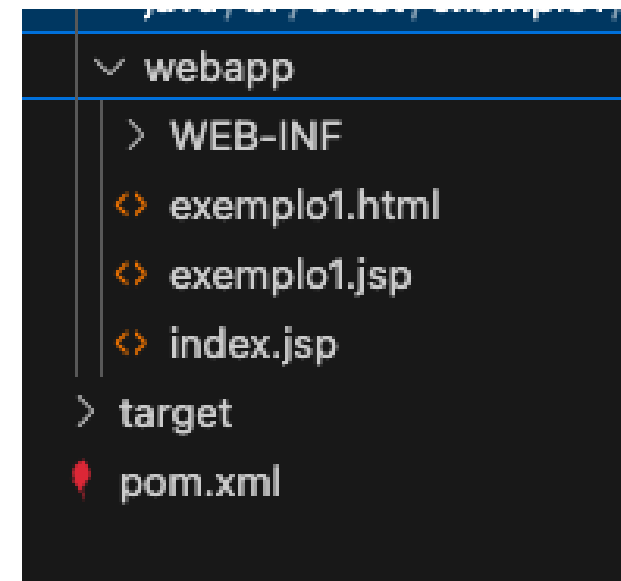
Insira as linhas antes da “<!DOCTYPE html>”

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
```

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
```

No body, digite

```
<p> Olá mundo! Exemplo1 JSP </p>
```



## 6. Criar um pacote

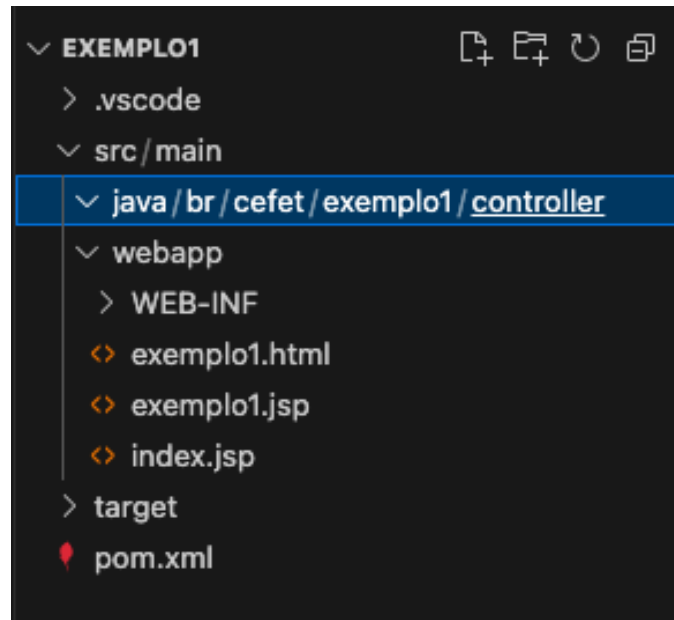
Selecione a pasta src/main

Crie a pasta **java** (botão direito, New folder)

**Selecione a pasta src/main/java**

**Botão direito, New Java Package (ou pastas)**

**Nomeie para br.cefet.exemplo1.controller**



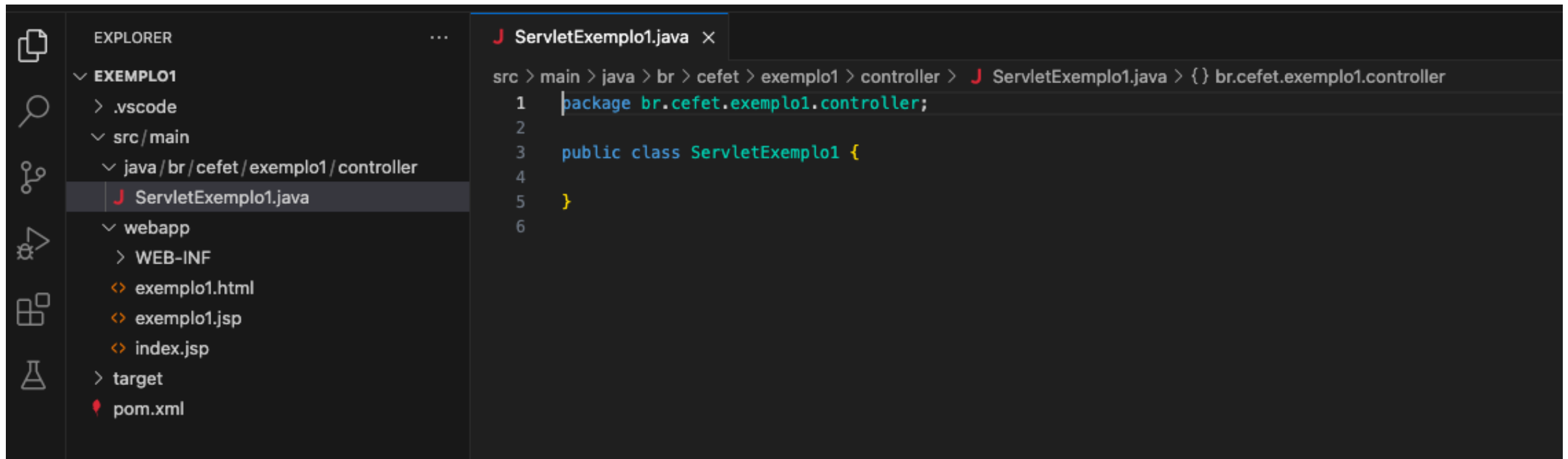
# 7. Criar um Servlet

Selecione a pasta src/main/java

Selecione o para br.cefet.demo1.controller

Botão direito, New Java File / Class

ServletExemplo1.java





# 7. Servlet Padrão

```
package br.cefet.exemplo1.controller;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.annotation.WebServlet;  
import jakarta.servlet.http.HttpServlet;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import java.io.IOException;
```

```
@WebServlet("/exemplo")
```

```
public class Exemplo extends HttpServlet {
```

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

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

```
}
```

# 7. Criar um Servlet - Exemplo

```
package br.cefet.exemplo1.controller;
```

```
import java.io.IOException;
```

```
import jakarta.servlet.annotation.WebServlet;
```

```
import jakarta.servlet.http.HttpServlet;
```

```
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http.HttpServletResponse;
```

```
@WebServlet("/ServletExemplo1")
```

```
public class ServletExemplo1 extends HttpServlet {
```

```
    private static final long serialVersionUID = 1L;
```

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

```
        response.setContentType("text/html");
```

```
        response.setCharacterEncoding("UTF-8");
```

```
        response.getWriter().println("<html><body>");
```

```
        response.getWriter().println("<h1>Olá, mundo! Este é um Servlet.</h1>");
```

```
        response.getWriter().println("</body></html>");
```

```
    }
```

```
}
```

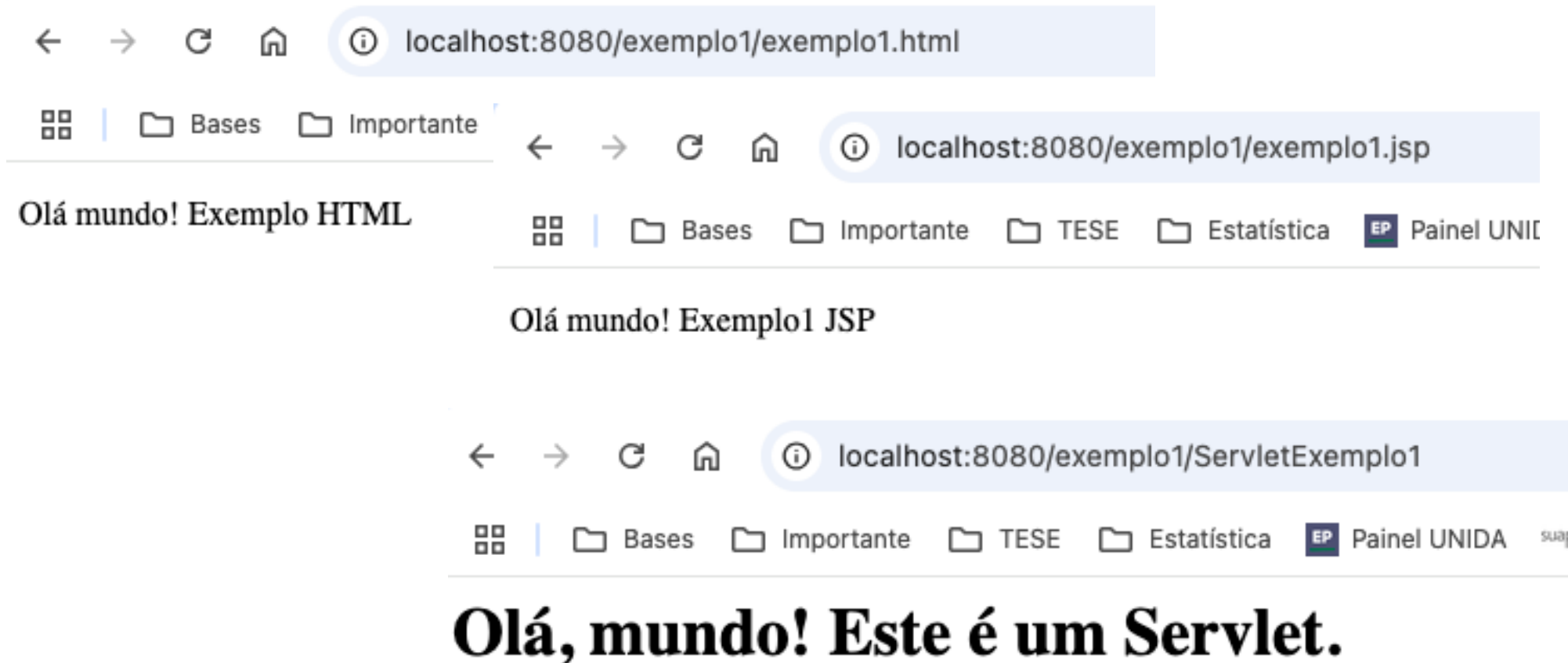
# 8. Realizar o Deploy e testar a aplicação

No terminal,

**Compilar:** mvn package

**Copiar o arquivo:** copy .\target\exemplo1.war c:\dev\pac2025\tomcat\webapps

**Executar o TomCat:** startup.bat



# EXERCÍCIO

Criar, configurar e testar **CINCO\*** novos projetos

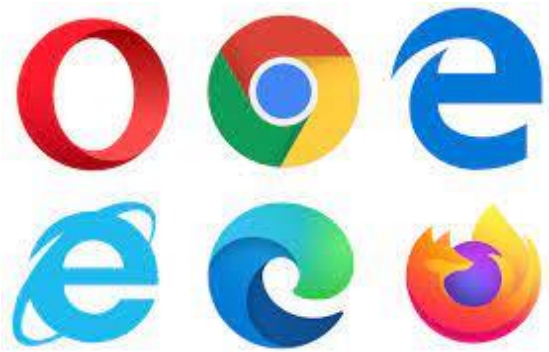
- Criar o novo projeto (em um novo workspace);
- Configurar o pow.xml;
- Configurar o web.xml;
- Criar um arquivo HTML e JSP;
- Criar um pacote;
- Criar um servlet e
- Realizar o deploy e testar a aplicação.

**\*mude o nome de tudo em cada operação.**



**Repita as  
operações 5x.**

# Java Server Page (JSP)



*request*



*response*



**JSP**

- HTML
- Scriptlet `<% %>`

*Apache TomCat*



# Java Server Page

- A tecnologia Java Server Page (JSP) permite gerar conteúdo da Web dinâmico (HTML);
- Arquivos JSP são uma forma de implementar o conteúdo da página dinâmica do lado do servidor.
- Em páginas JSP o servidor (Apache Tomcat) inclui dinamicamente conteúdo nas páginas HTML antes que elas sejam enviadas para um navegador solicitante.

# Java Server Page - Scriptlet

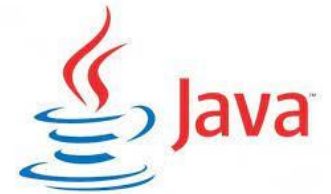
- Scriptlet é um pedaço de código Java embutido em uma página JSP semelhante a um código HTML.
- O código deve estar dentro das tags `<% %>`.
- Entre `<% %>` o DEV pode adicionar código Java.

# Exercícios / Exemplos



## Exemplo 1 – Exibir mensagem na tela

```
<html>
<head>
    <title>Bem-vindo!</title>
</head>
<body>
    <%
        out.println("<h1>Seja bem-vindo!</h1>");
    %>
</body>
</html>
```



## Exemplo 2 – Somar dois números e apresentar o resultado na tela

```
<body>
```

```
<%
```

```
    float n1 = 2;
```

```
    float n2 = 3;
```

```
    float soma = n1 + n2;
```

```
    out.println("<p> A soma é " + soma);
```

```
%>
```

```
</body>
```

# Exercício 1

Faça um JSP que atribua um número inteiro a uma variável e mostre uma mensagem indicando o número, se ele é par ou ímpar, e se é positivo ou negativo.

**Exemplo:**

**O número -21 é ímpar e negativo.**

# Exercício 1 - Resposta

<%

```
int n = -21;  
if (n % 2 == 0)  
    out.println(n + " é par");  
else  
    out.println(n + " é impar");  
    if (n > 0)  
        out.println(" e positivo");  
    else  
        if (n < 0)  
            out.println(" e negativo");  
        else  
            out.println(" e neutro");
```

%>

## Exercício 2

Faça um JSP que atribua um número inteiro positivo a uma variável. Você deverá mostrar todos os números predecessores até 0 (for). No mesmo JSP, exiba os números de 0 até o número inteiro (while).

Exemplo - For: (n = 4)

4  
3  
2  
1  
0

Exemplo - while: (n = 4)

0  
1  
2  
3  
4

## Exercício 2 - Resposta

<%

```
int n = 4;  
for (int i = n; i >= 0; i--)  
    out.println(i);
```

%>

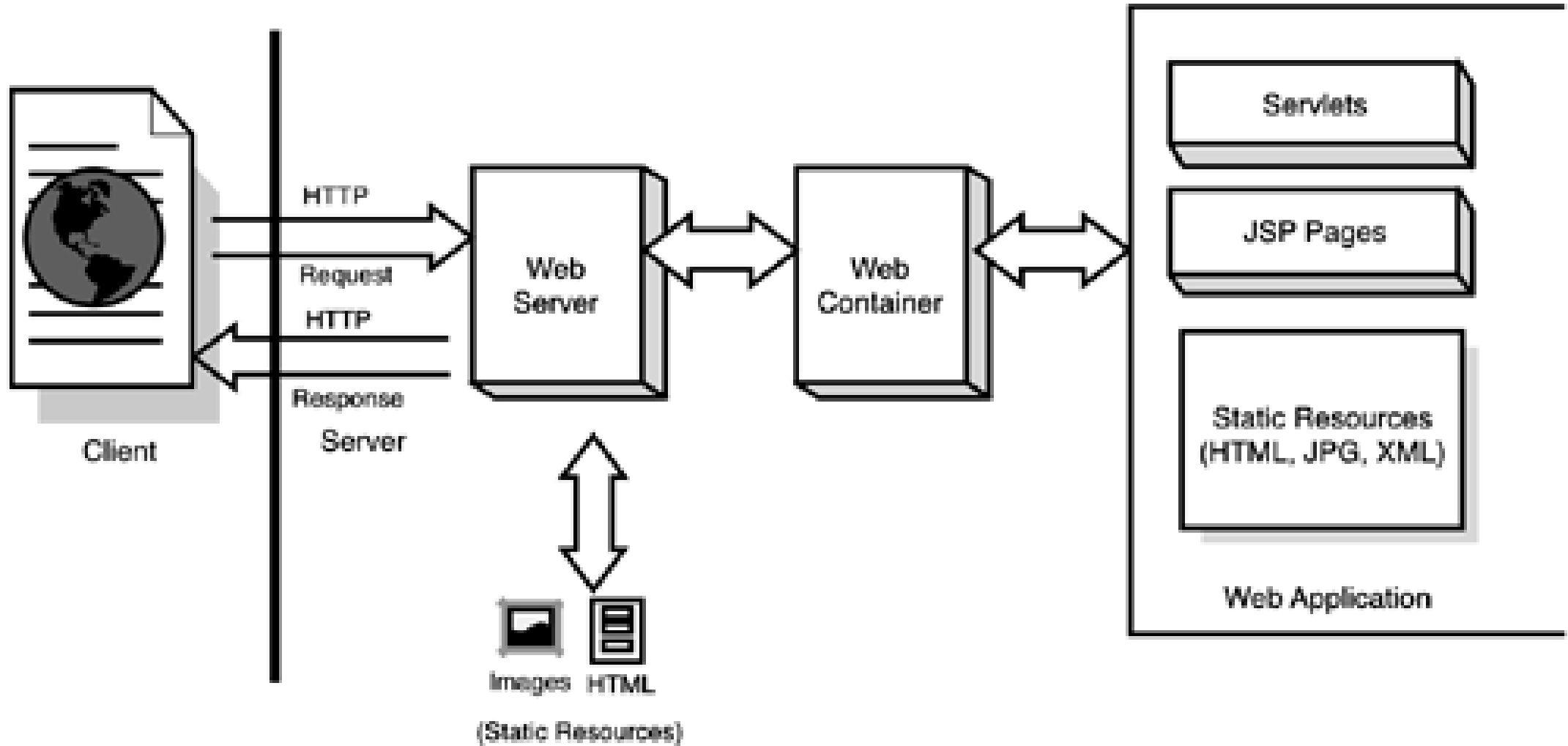
<%

```
int j = 0;  
while (j <= n){  
    out.println(j);  
    j++;
```

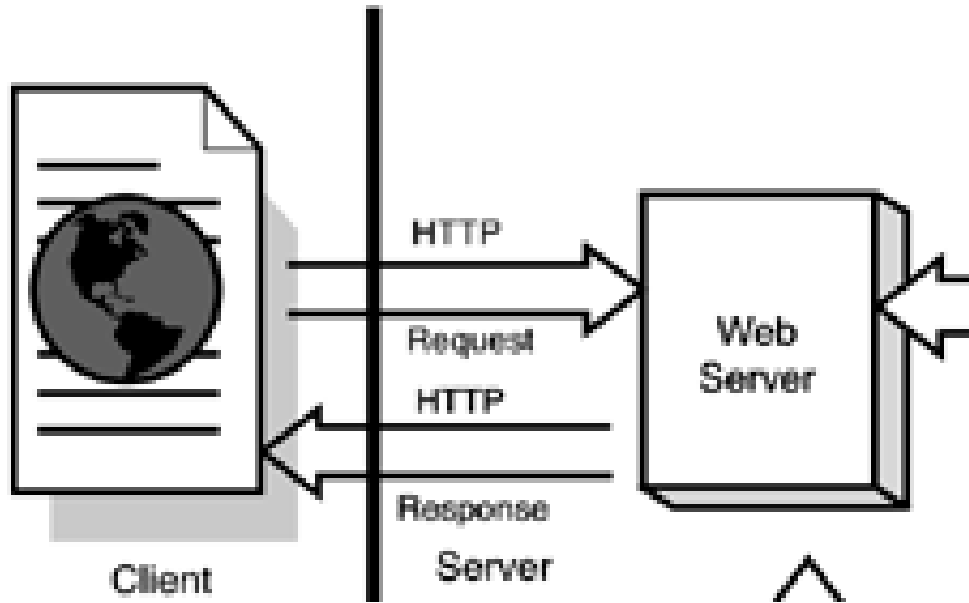
```
}
```

%>

# Arquitetura Java EE



# Arquitetura Java EE



**HTTP** é um protocolo de comunicação. Por meio dele o cliente e o servidor conseguem se comunicar;

## **Request**

A requisição é o pedido que um cliente realiza ao servidor;

## **Response**

Resposta do servidor web ao cliente;



## Exemplo 3 – Extraíndo um valor de uma variável do REQUEST (exemplo3.jsp)

```
<body>
<form action="exemplo3.jsp" method="GET">
  <input type="text" name="nome">
  <input type="submit"><br/>
</form>
<%
    String nome = request.getParameter("nome");
    out.print("Bem-vindo " + nome);
%>
</body>
```

## Exemplo 4 – JSP (Somando dois números)

```
<form method="post" action="exemplo4.jsp">
  Número 1: <input type="number" name="n1">
  Número 2: <input type="number" name="n2">
  <input type="submit">
</form>
<%
String s1 = request.getParameter("n1");
String s2 = request.getParameter("n2");

if ((s1!=null ) && (s2!=null)){
    float n1 = Float.parseFloat(s1);
    float n2 = Float.parseFloat(s2);
    float soma = n1 + n2;
    out.println("<p> A soma é " + soma);
}
%>
```

# Exercício 3

1. Crie o formulário proposto (exercicio3.jsp).
2. Crie um JSP (extracao3.jsp) que extrairá todos os valores do formulário por meio do request e exiba os valores na tela.

## exercicio3.jsp

Nome:

Preço:

## extracao3.jsp

Nome: Banana

R\$: 3.99

# Exercício 3 - Resposta

## exercicio3.jsp

```
<form action="extracao3.jsp" method="GET">
  <label for="nome">Nome: </label><br>
  <input type="text" id="nome" name="nome"><br>
  <label for="preco"> Preço:</label><br>
  <input type="number" id="preco" name="preco"
  placeholder="0.00" step="0.01" min="0"> <br>
  <input type="submit">
</form>
```

## extracao3.jsp

```
<%
    String nome = request.getParameter("nome");
    float preco = Float.valueOf(request.getParameter("preco"));
    out.println("Nome: " + nome + "<br>");
    out.println("Preço: " + preco);
%>
```

# Exercício 4

1. Crie o formulário proposto (exercicio4.jsp).
2. Crie um JSP (extracao4.jsp) que extrairá todos os valores do formulário por meio do request. Transforme a data de admissão de String para date. Além disso, calcule um aumento de 10% no salário. Ao final, exiba todos os valores.

## exercicio4.jsp

Nome:

Salario:

Data de Admissão:



Salario:



## extracao4.jsp

Nome: Maria

Salário: 12245.05

Aumento: 13469.555

UF: CE

Data inserida: 2023-12-01

Data convertida (java.sql.Date): 2023-12-01

Data convertida (formato dd/MM/yyyy): 01/12/2023

# Exercício 4

Trabalhar com data....

<%

// Capturando a data de admissão

String inputDateStr = request.getParameter("datadm");

//Fazendo a conversão

java.time.LocalDate localDate = java.time.LocalDate.parse(inputDateStr);

java.sql.Date sqlDate = java.sql.Date.valueOf(localDate);

// Define o formato de saída da data (dd/MM/yyyy)

java.time.format.DateTimeFormatter outputFormatter =

java.time.format.DateTimeFormatter.ofPattern("dd/MM/yyyy");

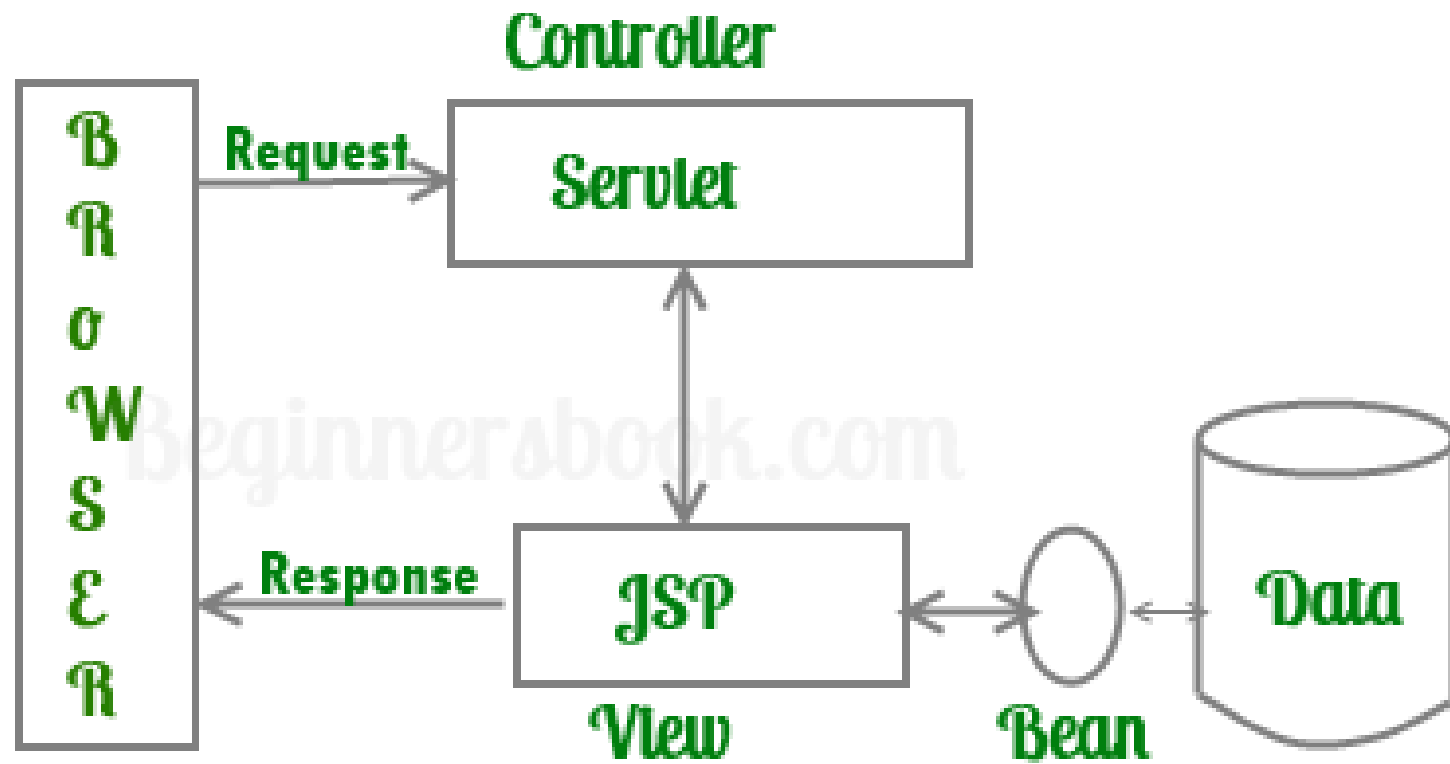
out.println("Data inserida: " + inputDateStr + "<br>");

out.println("Data convertida (java.sql.Date): " + sqlDate + "<br>");

out.println("Data convertida (formato dd/MM/yyyy): " + localDate.format(outputFormatter) + "<br>");

%>

# Servlets



# Servlets

- Servlet é uma classe Java que é usada para estender a funcionalidade de servidores web.
- Servlet é responsável por processar as solicitações HTTP recebidas pelo servidor web e gerar uma resposta apropriada.
- Em Java consideraremos as classes (`HttpServletRequest request`, e `HttpServletResponse response`).
- Seus principais métodos são `doGet`, `doPost`, `doPut` e `doDelete`;
- Ciclo de Vida:
  - o método `init()` é executado na primeira vez que o servlet é executado e
  - O método `destroy()` é executado quando o servidor está sendo derrubado.



## Exemplo – Servlet Completo

```
package br.cefet.sisvenda.controller;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
```

**@WebServlet("/hello")**

```
public class HelloServlet extends HttpServlet {
```

**// Construtor do servlet**

```
public HelloServlet() {
    super();
    System.out.println("Servlet instanciado.");
}
```

**// Chamado quando o servlet é inicializado**

```
@Override
public void init() throws ServletException {
    System.out.println("Servlet inicializado.");
}
```

**@WebServlet = Mapeamento da rota externa do servlet**

# Exemplo – Servlet Completo

## // Chamado quando o servlet é destruído

```
@Override  
public void destroy() {  
    System.out.println("Servlet destruído.");  
}
```

## // Lida com requisições HTTP GET

```
@Override  
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,  
IOException {  
    response.setContentType("text/plain");  
    response.getWriter().write("Olá via GET!");  
}
```

## // Lida com requisições HTTP POST

```
@Override  
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,  
IOException {  
    response.setContentType("text/plain");  
    response.getWriter().write("Olá via POST!");  
}
```

## Acesso via Curl

curl -X GET http://localhost:8080/demo/hello  
Olá via GET!

curl -X POST http://localhost:8080/demo/hello  
Olá via POST!

## Exemplo – Servlet Completo

### // Responde com requisições HTTP PUT

@Override

```
protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    response.setContentType("text/plain");
    response.getWriter().write("Olá via PUT!");
}
```

### // Responde com requisições HTTP DELETE

@Override

```
protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    response.setContentType("text/plain");
    response.getWriter().write("Olá via DELETE!");
}
```

## Exemplo – Servlet Completo

**// Responde com requisições HTTP HEAD (sem corpo na resposta)**

```
@Override
protected void doHead(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    response.setContentType("text/plain");
    // Sem corpo na resposta
}
```

**// Responde com requisições HTTP OPTIONS (verbo usado para descobrir os métodos permitidos)**

```
@Override
protected void doOptions(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    response.setHeader("Allow", "GET, POST, PUT, DELETE, OPTIONS, HEAD");
}
}
```

## Exemplo 5 – Gerando HTML (Servlet)

```
@WebServlet("/Exemplo5")
```

```
public class Exemplo5 extends HttpServlet {  
    private static final long serialVersionUID = 1L;
```

```
    public Exemplo6() {  
        super();  
    }
```

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

```
        PrintWriter out = response.getWriter();  
        out.println("<h1>Seja bem-vindo!</h1>");  
    }
```

## Exemplo 6 – Soma de dois números (Servlet)

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

```
    float n1 = 2;
```

```
    float n2 = 3;
```

```
    float soma = n1 + n2;
```

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

```
    response.setCharacterEncoding("UTF-8");
```

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

```
    out.println("<p> A soma é " + soma + "</p>");
```

```
}
```

## Exemplo 7 – Formulário - Extraíndo um valor de uma variável do REQUEST

---

Número 1:

Número 2:

exemplo7.jsp

```
<form method="GET" action="Exemplo7">  
  Número 1: <input type="number" name="n1">  
  Número 2: <input type="number" name="n2">  
  <input type="submit">  
</form>
```

## Exemplo 7– Servlet - Extraíndo um valor de uma variável do REQUEST

```
protected void doGet(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
    String s1 = request.getParameter("n1");  
    String s2 = request.getParameter("n2");  
  
    float n1 = Float.parseFloat(s1);  
    float n2 = Float.parseFloat(s2);  
    float soma = n1 + n2;  
  
    PrintWriter out = response.getWriter();  
    out.println("<p> A soma é " + soma + "</p>");  
    }  
}
```



# JSP - Expression Language (EL)

- **Expression Language (EL)** possibilita acessar variáveis, a session, request e objetos Java no JSP.
- O valor da variável/objeto deve ser **despachado** (no servlet) antes de entrar no arquivo jsp.

Ex.: Formulário

Número 1: `<input type="number" name="n1" value="${valor1}">`

Nome: `<input type="text" name="nome" value="${aluno.nome}">`

# Servlet - Despachando um objeto do Servlet para o JSP

```
// 1
request.setAttribute("soma",soma);
// 2
RequestDispatcher rd = request.getRequestDispatcher("/exemplo8.jsp");
// 3
rd.forward(request,response);
```

1. Empacote a variável ao request (objeto).
2. Prepare o despacho da variável (caminho).
3. Despache a variável (envio do objeto ao destino);

## Exemplo 8 – Formulário – Acessando valores despachados

exemplo8.jsp

```
<form method="GET" action="Exemplo8">
```

```
  Número 1: <input type="number" name="n1" value="{n1}">
```

```
  Número 2: <input type="number" name="n2" value="{n2}">
```

```
<input type="submit">
```

```
<p> A soma é {soma}
```

## Exemplo 8 – Servlet –Despachando uma variável para um JSP (Exemplo8)

```
String s1 = request.getParameter("n1");
```

```
String s2 = request.getParameter("n2");
```

```
float n1 = Float.parseFloat(s1);
```

```
float n2 = Float.parseFloat(s2);
```

```
float soma = n1 + n2;
```

```
// Despachando
```

```
request.setAttribute("soma",soma);
```

```
request.setAttribute("n1",n1);
```

```
request.setAttribute("n2",n2);
```

```
RequestDispatcher rd = request.getRequestDispatcher("exemplo8.jsp");
```

```
rd.forward(request,response);
```

# Exercício 5


1. Crie o formulário proposto (exercicio5.jsp).
2. Crie um Servlet (ServletFuncionario) que extrairá todos os valores do formulário por meio do request. Além disso, será calculado um aumento do salário de 10%. Ao final, despache todos as variáveis para a tela (exercicio5.jsp).
3. O arquivo exercicio5.jsp deverá utilizar de Expression language para preencher todos os campos dos campos.

Nome:


Salario:

Aumento (10%):

Admissão:

Estado

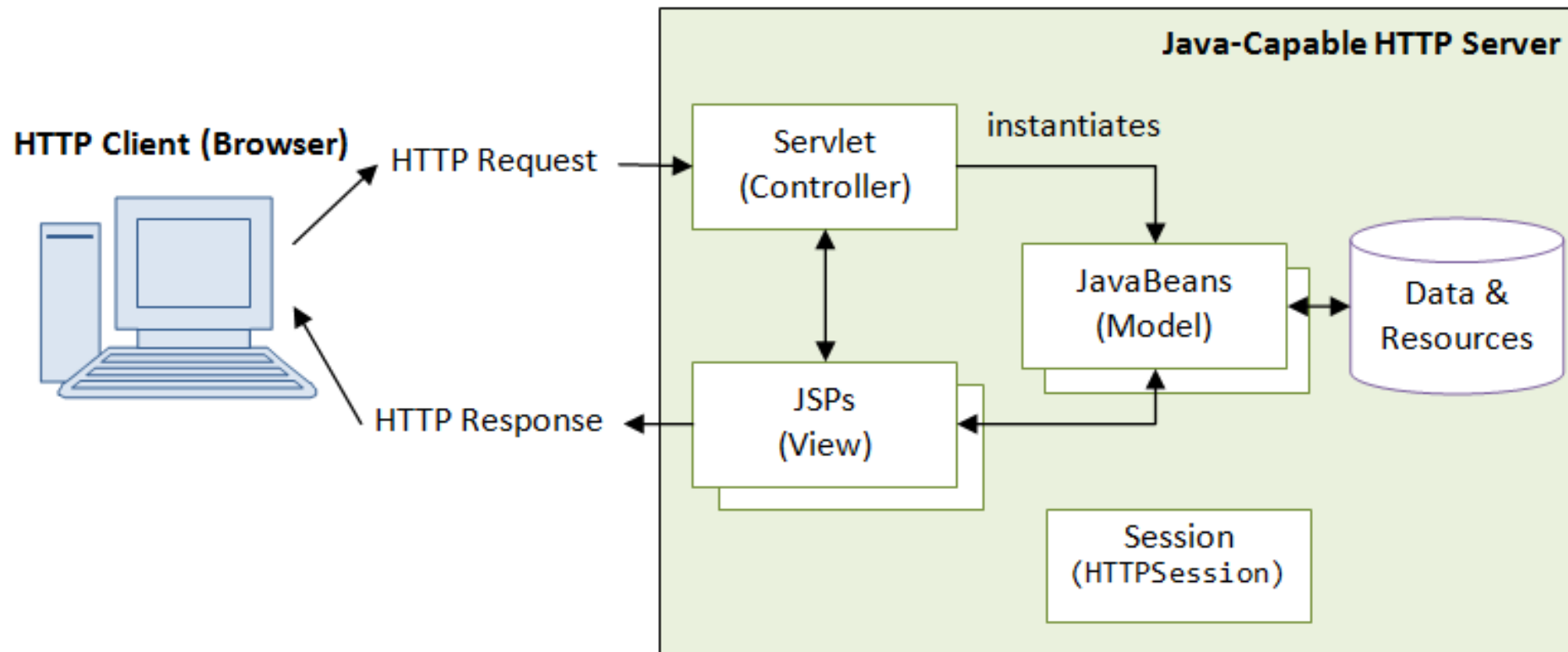
 

Enviar

**exercicio5.jsp**

# Operações essenciais em Java WEB

## 9. Criar uma classe Beans;



## Exemplo 10 – Bean / Model Funcionário

Funcionario
<ul style="list-style-type: none"><li>- nome : String</li><li>- salario : float</li><li>- admissao : Date</li><li>- uf : String</li></ul>


Crie a classe com seus atributos, construtores e métodos Gets e Sets.

## Exercício 6

1. Crie o formulário proposto (exercicio6.jsp).
2. Crie um Servlet (Funcionario6) que extrairá todos os valores do formulário por meio do request. Transforme a data de admissão de String para Date. Instancie um objeto funcionario. Ao final, despache o objeto funcionário para página (exercicio6.jsp).
3. O arquivo exercicio6.jsp deverá utilizar de Expression Language para preencher todos os campos (objeto funcionario) e apresentar na parte inferior todas as informações do funcionário.

Nome:

Salario:

Admissão:  
 

Estado

Funcionario
<ul style="list-style-type: none"><li>- nome : String</li><li>- salario : float</li><li>- admissao : Date</li><li>- uf : String</li></ul>



## Exercício 6

4. Na classe funcionário, implemente o método:

4.1. Que apresente a data no formato (dd/mm/yyyy);


4.2. Que apresente o aumento salarial.

4.3. Que liste por por extenso a unidade da federação.


Nome:

Salario:

Admissão:

Estado

Enviar

### Funcionario

- nome : String
- salario : float
- admissao : Date
- uf : String