

An aerial photograph of a university campus. The image shows a large green lawn in the center, surrounded by various buildings, some with red roofs and others with white roofs. There are several ponds and a winding path. The text "WEB ACADEMY" is overlaid in a large, bold, dark blue font.

# WEB ACADEMY

## Fundamentos de Programação Back-end

Daniel Augusto Nunes da Silva

# **Apresentação**

# Ementa

- Linguagens de programação server-side. Arquitetura em camadas. **Servlets** e Jakarta Server Pages (**JSP**). Acesso à bases de dados com **JDBC** (Java Database Connectivity). Implementação de operações **CRUD** (Create, Read, Update, Delete). Segurança.



# Objetivos

- **Geral:** Capacitar o aluno na utilização de **procedimentos e técnicas básicas** de desenvolvimento de aplicações para a WEB, com ênfase nos fundamentos dos **recursos nativos da linguagem Java** aplicados ao desenvolvimento **back-end**.
- **Específicos:**
  - Compreender a estrutura de uma aplicação web construída com recursos nativos da linguagem Java;
  - Apresentar uma visão geral do funcionamento de aplicações web baseadas em Servlets e as vantagens da utilização de JSP;
  - Permitir ao aluno conhecer e aplicar os recursos básicos necessários para construção de aplicações web com acesso a banco de dados utilizando as tecnologias JDBC e JSP;
  - Demonstrar a execução de tarefas relacionadas ao processo de implantação de aplicações web.

# Conteúdo programático

## Introdução

- Linguagens de programação server-side
- Revisão da linguagem Java e POO;
- Arquitetura em camadas e MVC.

## Servlets

- Visão geral do funcionamento de Servlets;
- Ciclo da vida;
- Tratamento de solicitações HTTP.
- Servidores de aplicação (Tomcat), empacotamento (WAR) e implantação de aplicações web Java em ambiente de produção.

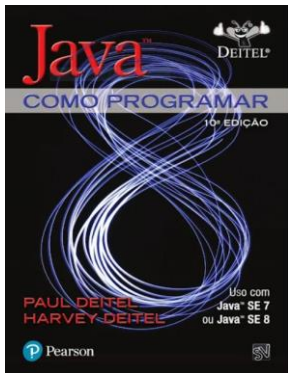
## JDBC

- Principais classes e métodos da API do JDBC;
- Configuração e gerenciamento de conexão com bases de dados;
- Drivers e fontes de dados
- Sintaxe das principais instruções SQL usadas em operações CRUD;
- Execução de instruções SQL (Statements e Result Sets).

## JSP

- Elementos, ações-padrão e diretivas;
- Objetos implícitos;
- Tratamento de exceções;
- Segurança de aplicações web em Java.

# Bibliografia



## Java: Como Programar.

Paul Deitel e Harvey Deitel

10ª Edição – 2016

Editora Pearson

ISBN 9788543004792



## Engenharia de Software Moderna

Marco Tulio Valente

<https://engsoftmoderna.info/>



# Sites de referência

- Jakarta Server Pages Specification.
  - <https://jakarta.ee/specifications/pages/3.0/jakarta-server-pages-spec-3.0.html>
- Jakarta Servlet Specification.
  - <https://jakarta.ee/specifications/servlet/5.0/jakarta-servlet-spec-5.0.html>

# Ferramentas: JDK e Maven

- **JDK 11**

- <https://www.oracle.com/br/java/technologies/javase/jdk11-archive-downloads.html>
- Criar a variável de ambiente JAVA\_HOME configurada para o diretório de instalação do JDK. Exemplo: “C:\Program Files\Java\jdk-11.0.13”.
- Adicionar “%JAVA\_HOME%\bin” na variável de ambiente PATH.
- Tutorial de configuração: [https://mkyong.com/java/how-to-set-java\\_home-on-windows-10/](https://mkyong.com/java/how-to-set-java_home-on-windows-10/)

- **Maven**

- <https://maven.apache.org/download.cgi>
- Adicionar o diretório de instalação do Maven na variável de ambiente PATH. Exemplo: “C:\apache-maven\bin”.
- Tutorial de instalação: <https://mkyong.com/maven/how-to-install-maven-in-windows/>



# Ferramentas

## Apache Tomcat 10

### Tomcat Web Application Manager

Message:	OK
----------	----

<b>Manager</b>			
<a href="#">List Applications</a>	<a href="#">HTML Manager Help</a>	<a href="#">Manager Help</a>	<a href="#">Server Status</a>

<b>Applications</b>					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/> <input type="button" value="Expire sessions"/> with idle ≥ 30 <input type="text" value=""/> minutes
/docs	None specified	Tomcat Documentation	true	0	Start <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/> <input type="button" value="Expire sessions"/> with idle ≥ 30 <input type="text" value=""/> minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/> <input type="button" value="Expire sessions"/> with idle ≥ 30 <input type="text" value=""/> minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/> <input type="button" value="Expire sessions"/> with idle ≥ 30 <input type="text" value=""/> minutes
/manager	None specified	Tomcat Manager Application	true	1	Start <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/> <input type="button" value="Expire sessions"/> with idle ≥ 30 <input type="text" value=""/> minutes

<b>Deploy</b>	
Deploy directory or WAR file located on server	
Context Path (required): <input type="text"/>	
XML Configuration file URL: <input type="text"/>	
WAR or Directory URL: <input type="text"/>	
<input type="button" value="Deploy"/>	
<b>WAR file to deploy</b>	
Select WAR file to upload <input type="button" value="Choose File"/> No file chosen	
<input type="button" value="Deploy"/>	

<https://tomcat.apache.org/>

# Ferramentas

MySQL

```
mysql> SET GLOBAL log_output = 'TABLE';
Query OK, 0 rows affected (0.00 sec)

mysql> SET GLOBAL general_log = 'ON';
Query OK, 0 rows affected (0.00 sec)

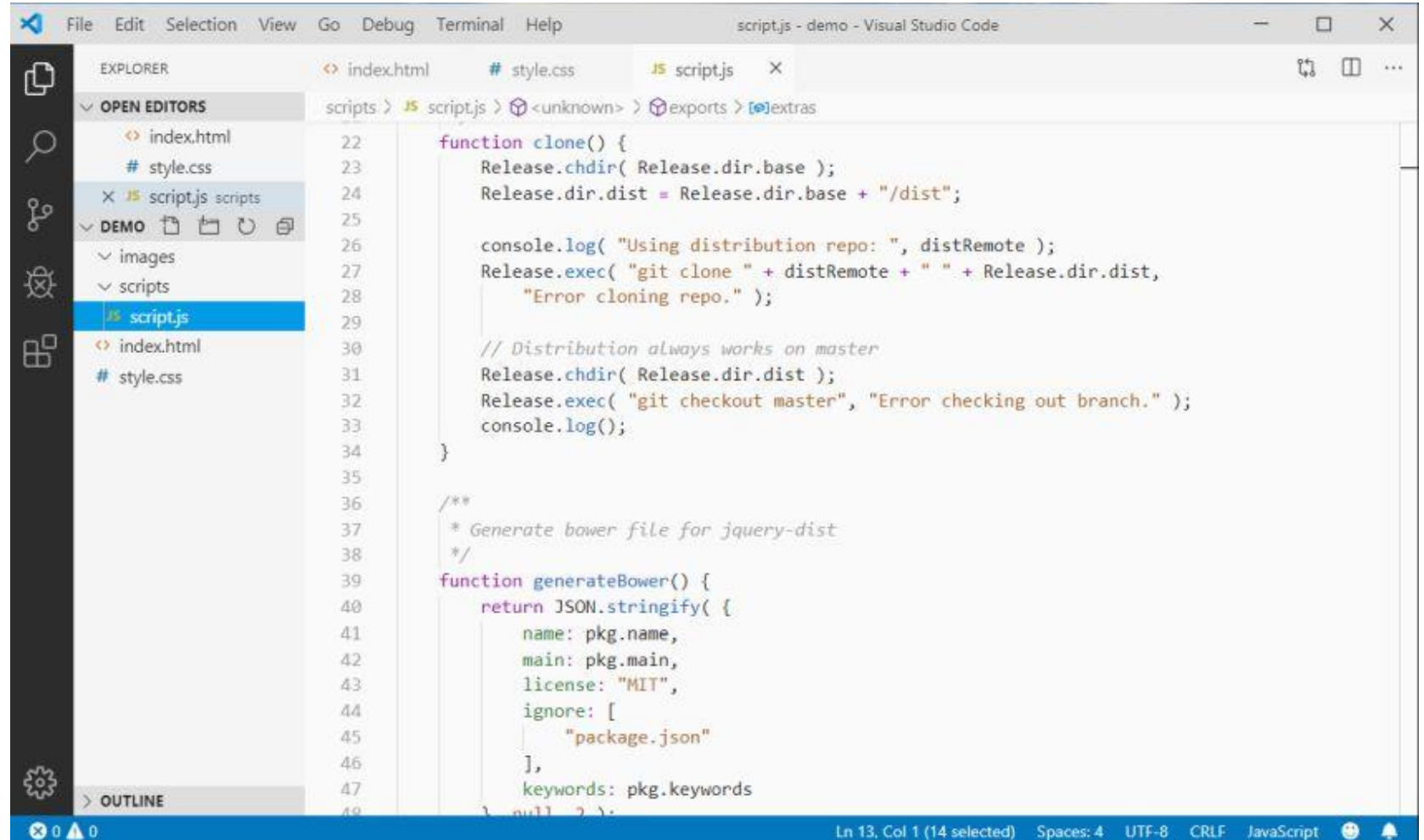
mysql> SELECT COUNT(*) FROM mysql.general_log WHERE command_type = 'Query' AND argument LIKE 'INSERT %' AND event_time > '2017-04-16 11:22:00';
+-----+
| COUNT(*) |
+-----+
|         2 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT event_time, server_id, COUNT(server_id) FROM mysql.general_log WHERE command_type = 'Query' GROUP BY server_id;
+-----+-----+-----+
| event_time          | server_id | COUNT(server_id) |
+-----+-----+-----+
| 2017-05-18 10:58:07 |         0 |             1282 |
+-----+-----+-----+
```

<https://dev.mysql.com/downloads/windows/installer/8.0.html>

# Ferramentas

Visual Studio Code



```
File Edit Selection View Go Debug Terminal Help
script.js - demo - Visual Studio Code

EXPLORER
  OPEN EDITORS
    index.html
    style.css
    script.js scripts
  DEMO
    images
    scripts
    script.js
    index.html
    style.css

OUTLINE

22 function clone() {
23   Release.chdir( Release.dir.base );
24   Release.dir.dist = Release.dir.base + "/dist";
25
26   console.log( "Using distribution repo: ", distRemote );
27   Release.exec( "git clone " + distRemote + " " + Release.dir.dist,
28     "Error cloning repo." );
29
30   // Distribution always works on master
31   Release.chdir( Release.dir.dist );
32   Release.exec( "git checkout master", "Error checking out branch." );
33   console.log();
34 }
35
36 /**
37  * Generate bower file for jquery-dist
38  */
39 function generateBower() {
40   return JSON.stringify( {
41     name: pkg.name,
42     main: pkg.main,
43     license: "MIT",
44     ignore: [
45       "package.json"
46     ],
47     keywords: pkg.keywords
48   } );
49 }
```

<https://code.visualstudio.com/docs/java/java-tutorial>

# Ferramentas: Extensões do VS Code

- **Extension Pack for Java**

- <https://marketplace.visualstudio.com/items?vscjava.vscode-java-pack>

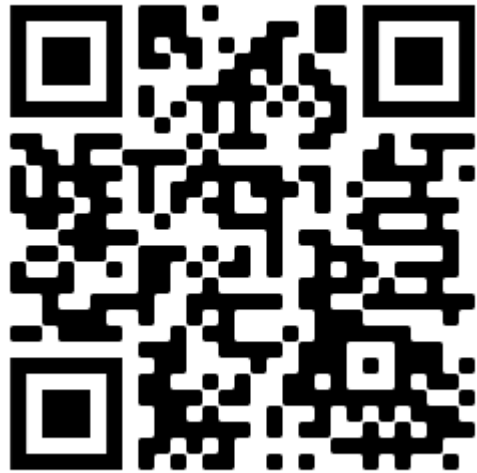
- **Log Viewer**

- <https://marketplace.visualstudio.com/items?itemName=berublan.vscode-log-viewer>

- **Java Server Pages (JSP)**

- <https://marketplace.visualstudio.com/items?itemName=pthorsson.vscode-jsp>

# Contato



<https://linkme.bio/danielnsilva/>

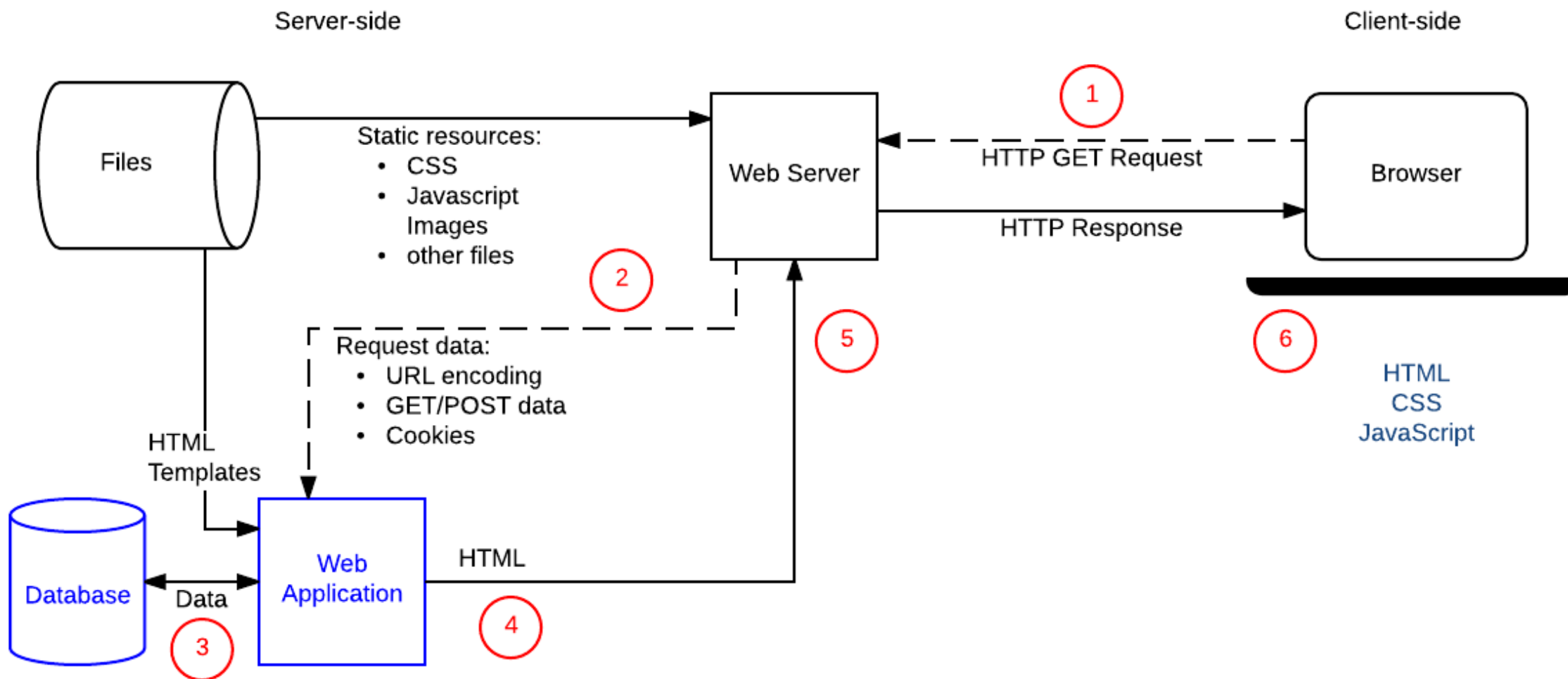


# Introdução

# Programação server-side

- Em **aplicações web** os navegadores (lado cliente) se comunicam com os servidores por meio do **protocolo HTTP**.
- Sempre que uma ação como a chamada de um link ou envio de formulário é realizada, uma **requisição HTTP** é feita ao servidor.
- Linguagens **client-side** estão ligadas a aparência e comportamento da página no navegador, enquanto que linguagens **server-side** estão relacionadas a tarefas como manipular os dados que serão retornados ao cliente.
- Exemplos de linguagem server-side: Java, PHP, Python, C#, JavaScript (Node.js).

# Programação server-side



Fonte: [https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First\\_steps/Introduction](https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Introduction)

# Java

- O processo criação e execução de um aplicativo Java pode ser resumido normalmente em 5 passos:
  1. Escrita do código-fonte (arquivo .java);
  2. Compilação do programa Java em **bytecodes**, gerando os arquivos .class;
  3. Carregamento do programa na memória pela **JVM** (Máquina Virtual Java);
  4. Verificação de bytecode pela JVM;
  5. Execução do programa pela JVM.

```
public class Exemplo {  
    public static void main(String[] args) {  
        System.out.println("WEB ACADEMY");  
    }  
}
```

```
>javac Exemplo.java  
  
>java Exemplo  
  
WEB ACADEMY
```

# Java

- Java é uma linguagem de **tipagem forte e estática** e, portanto, requer que todas as variáveis tenham um tipo.
- Tipos primitivos: boolean, char, byte, short, int, long, float, double.

```
public class Exemplo {  
    public static void main(String[] args) {  
        int x = 10;  
        x = "WEB ACADEMY";  
        mensagem = "WEB ACADEMY";  
        String mensagem = "WEB ACADEMY";  
        System.out.println(mensagem);  
    }  
}
```

```
>javac Exemplo.java  
Exemplo.java:4: error: incompatible types: String  
cannot be converted to int  
        x = "WEB ACADEMY";  
          ^  
Exemplo.java:5: error: cannot find symbol  
        mensagem = "WEB ACADEMY";  
          ^  
    symbol:   variable mensagem  
    location: class Exemplo  
2 errors
```



# Programação orientada a objetos (POO)

- **Classe:**

- Estrutura que abstrai um conjunto de objetos com características semelhantes.

- **Objeto:**

- Instância ou modelo derivado de uma classe, que pode ser manipulado pelo programa.

```
1.  public class Pessoa { // Classe
2.      private String nome;
3.      private String email;
4.      public String getNome() {}
5.      public void setNome(String nome) {}
6.      public String getEmail() {}
7.      public void setEmail(String email) {}
8.  }
9.  public class Exemplo {
10.     public static void main(String[] args) {
11.         Pessoa p = new Pessoa(); // Objeto
12.     }
13. }
```

# Programação orientada a objetos (POO)

- **Herança:**

- Mecanismo que permite criar novas classes, aproveitando as características da classe
- Promove reaproveitamento do código existente.

```
1.  public class Pessoa { // Superclasse
2.      private String nome;
3.      private String email;
4.      public String getNome() {}
5.      public void setNome(String nome) {}
6.      public String getEmail() {}
7.      public void setEmail(String email) {}
8.  }
9.  public class Aluno extends Pessoa { // Subclasse
10.     private int matricula;
11.     public int getMatricula() {}
12.     public void setMatricula(int matricula) {}
13. }
```

# Programação orientada a objetos (POO)

- **Encapsulamento:**

- Conceito voltado para **organização de informações que sejam relacionadas em um mesmo objeto** (classe).
- Não é sinônimo de ocultar informações, pois a restrição de acesso é apenas parte do conceito.

```
1. public class Pessoa {  
2.     private String nome;  
3.     private String email;  
4.     public String getNome() {}  
5.     public void setNome(String nome) {}  
6.     public String getEmail() {}  
7.     public void setEmail(String email) {}  
8. }
```

# Programação orientada a objetos (POO)

- **Polimorfismo:**

- Permite que os programas processem objetos que compartilham a mesma superclasse como se todos fossem objetos da superclasse.
- Uma das formas de implementar o polimorfismo é através de uma classe abstrata, cujos métodos são declarados mas não são definidos.

```
1. public abstract class Quadrilatero {  
2.     public abstract double calculaArea();  
3. }  
4. public class Quadrado extends Quadrilatero {  
5.     private double lado;  
6.     public Quadrado(double lado) {  
7.         this.lado = lado;  
8.     }  
9.     public double calculaArea() {  
10.         return this.lado * this.lado;  
11.     }  
12. }
```

# Programação orientada a objetos (POO)

- **Polimorfismo:**

- Outra forma de implementar o polimorfismo é por meio de interfaces.
- Uma interface define as operações que uma classe será obrigada a implementar.

```
1.  public interface Quadrilatero {  
2.      double calculaArea();  
3.  }  
4.  public class Quadrado implements Quadrilatero {  
5.      private double lado;  
6.      public Quadrado(double lado) {  
7.          this.lado = lado;  
8.      }  
9.      public double calculaArea() {  
10.         return this.lado * this.lado;  
11.     }  
12. }
```



# Programação orientada a objetos (POO)

- Parte significativa dos padrões de projeto de software são sustentados pelo uso de polimorfismo.
- **Strategy:** evita excesso de estruturas de controle aninhadas (dificulta manutenção).

```
1.  public class Desconto {  
2.      public double calcula(double valor, String tipo) {  
3.          double desconto = 0;  
4.          if (tipo.equals("ALUNO")) desconto = valor * 0.3;  
5.          else if (tipo.equals("PROFESSOR")) desconto = valor * 0.2;  
6.          else desconto = valor * 0.1;  
7.          return desconto;  
8.      }  
9.  }  
10. public class Exemplo {  
11.     public static void main(String[] args) {  
12.         Desconto d = new Desconto();  
13.         System.out.println(d.calcula(100, "PROFESSOR"));  
14.     }  
15. }
```

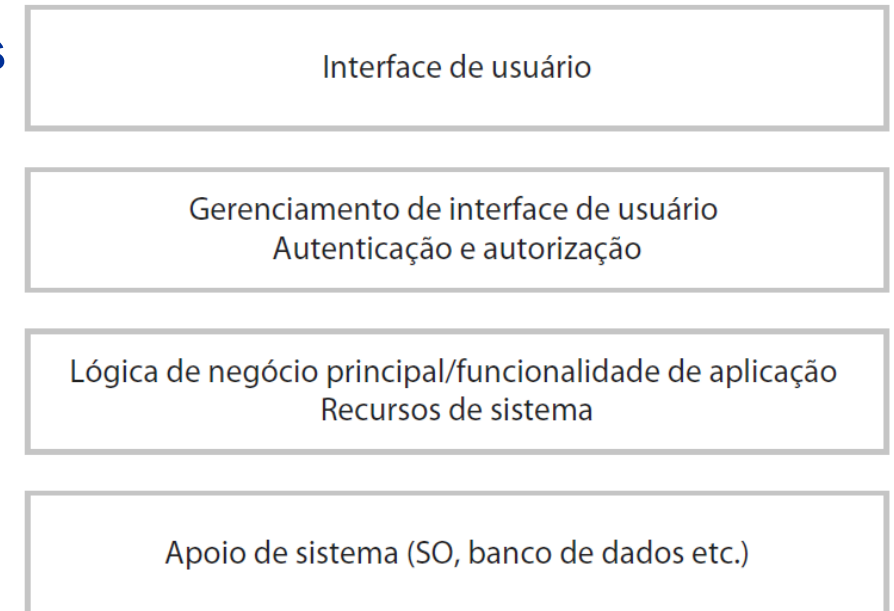
# Padrão de Projeto Strategy

```
1. public interface Desconto {
2.     double calcula(double valor);
3. }
4. public class Aluno implements Desconto {
5.     public double calcula(double valor) {
6.         return valor * 0.3;
7.     }
8. }
9. public class Professor implements Desconto {
10.    public double calcula(double valor) {
11.        return valor * 0.2;
12.    }
13. }
```

```
14. public class Outros implements Desconto {
15.     public double calcula(double valor) {
16.         return valor * 0.1;
17.     }
18. }
19. public class Exemplo {
20.     public static void main(String[] args) {
21.         Desconto d = new Professor();
22.         System.out.println(d.calcula(100));
23.     }
24. }
```

# Arquitetura em camadas

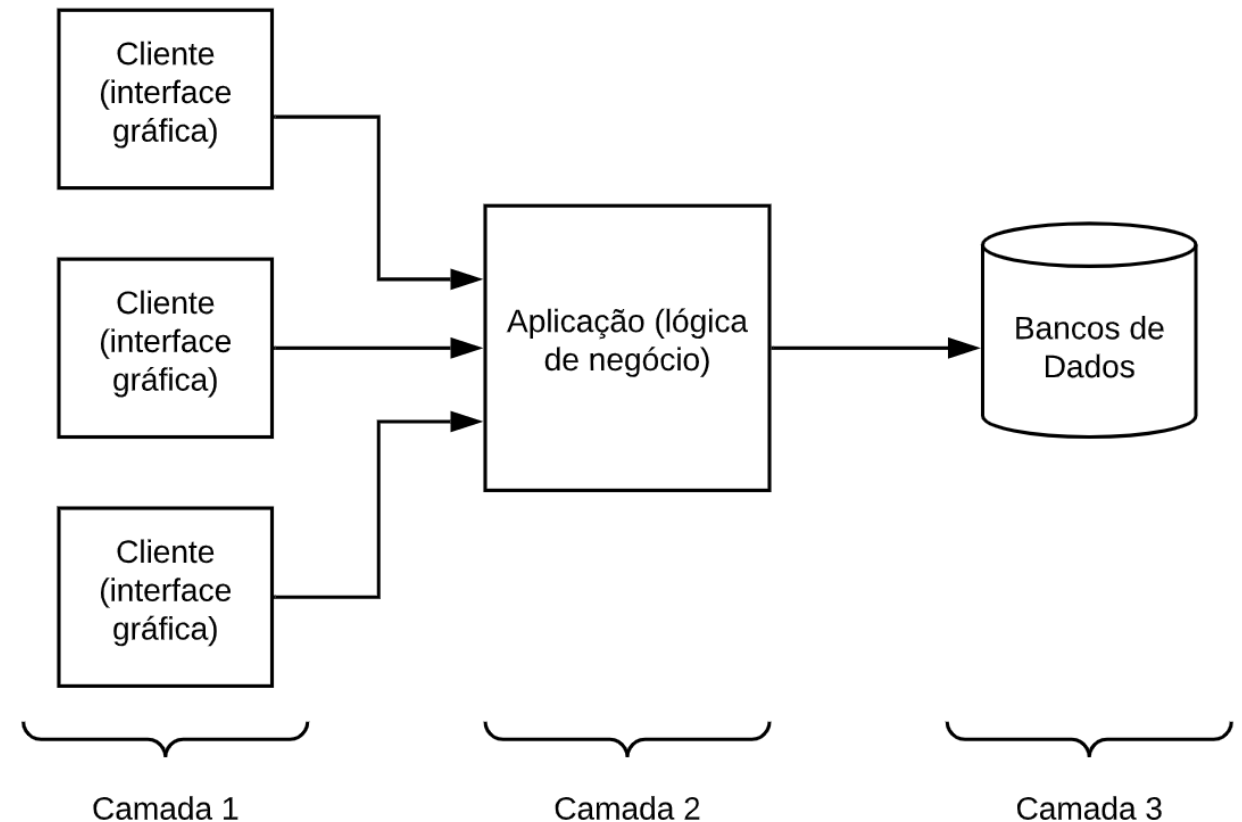
- Arquitetura em camadas é **um dos padrões arquiteturais mais usados**.
- As **classes são organizadas em módulos** de maior tamanho, chamados de camadas.
- As camadas são **dispostas de forma hierárquica**, onde uma camada somente pode usar serviços da camada imediatamente inferior.
- **Particiona a complexidade** envolvida no desenvolvimento de um sistema **em componentes menores** (as camadas), e disciplina as dependências entre essas camadas.



Fonte: SOMMERVILLE, 2011.

# Arquitetura em três camadas

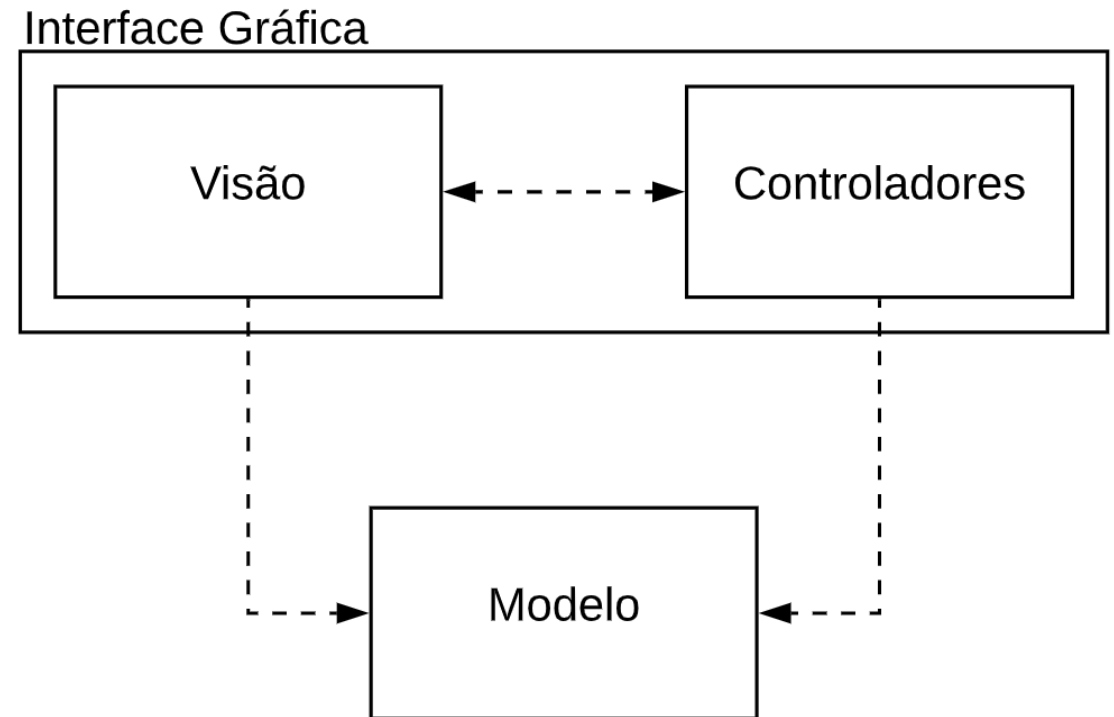
- Tipo de arquitetura **comum na construção de sistemas de informação corporativos**.
1. **Interface com o Usuário**, responsável por toda interação com o usuário;
  2. **Lógica de Negócio**, que implementa as regras de negócio do sistema;
  3. **Banco de Dados**, que armazena os dados manipulados pelo sistema.



Fonte: VALENTE, 2020.

# Arquitetura MVC (*Model-View-Controller*)

- **Visão**: responsável pela apresentação da interface gráfica do sistema, incluindo janelas, botões, menus, barras de rolagem, etc.
- **Controladores**: tratam e interpretam eventos gerados por dispositivos de entrada.
- **Modelo**: armazenam os dados manipulados pela aplicação, sem qualquer dependência com as outras camadas.

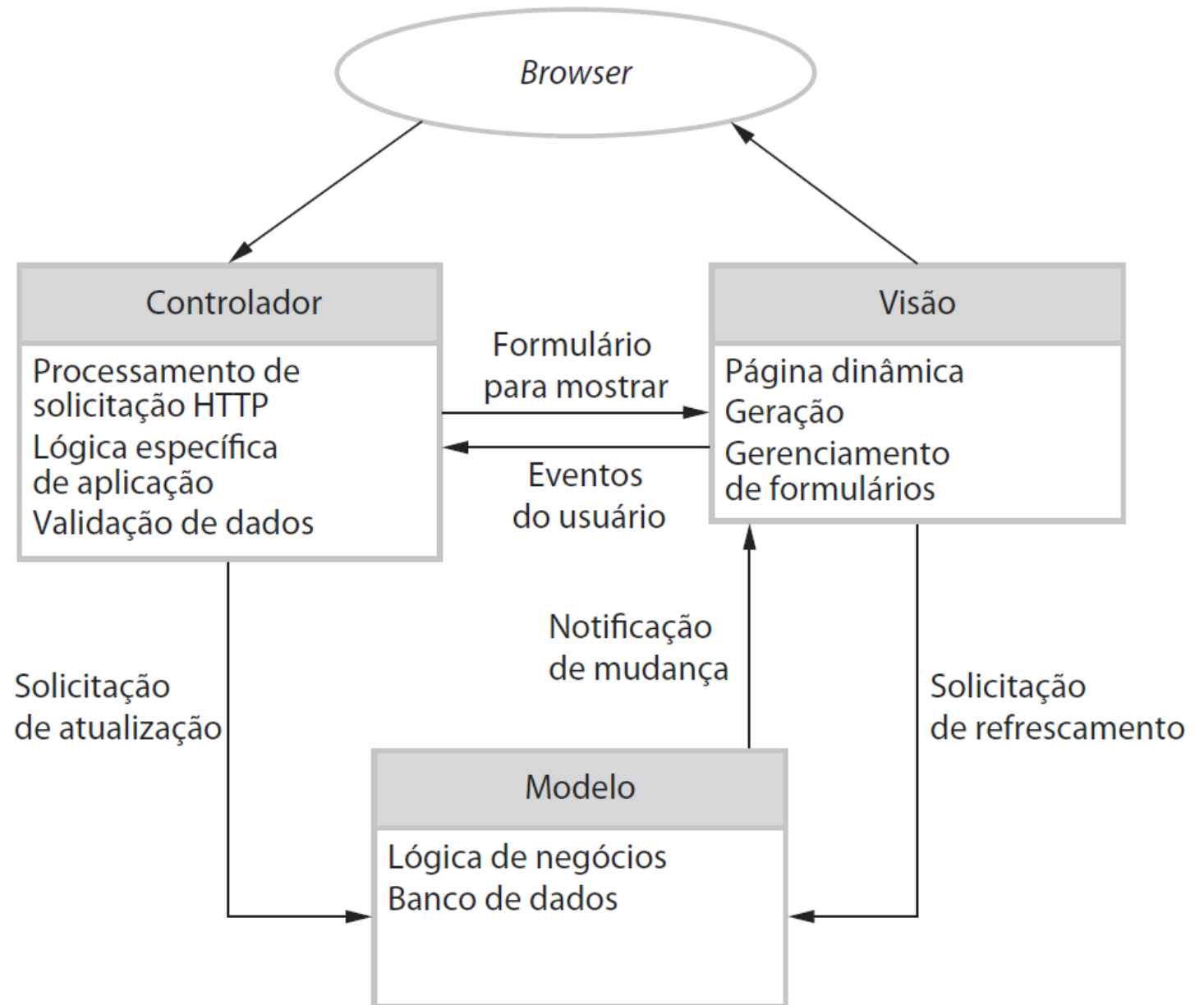


Fonte: VALENTE, 2020.



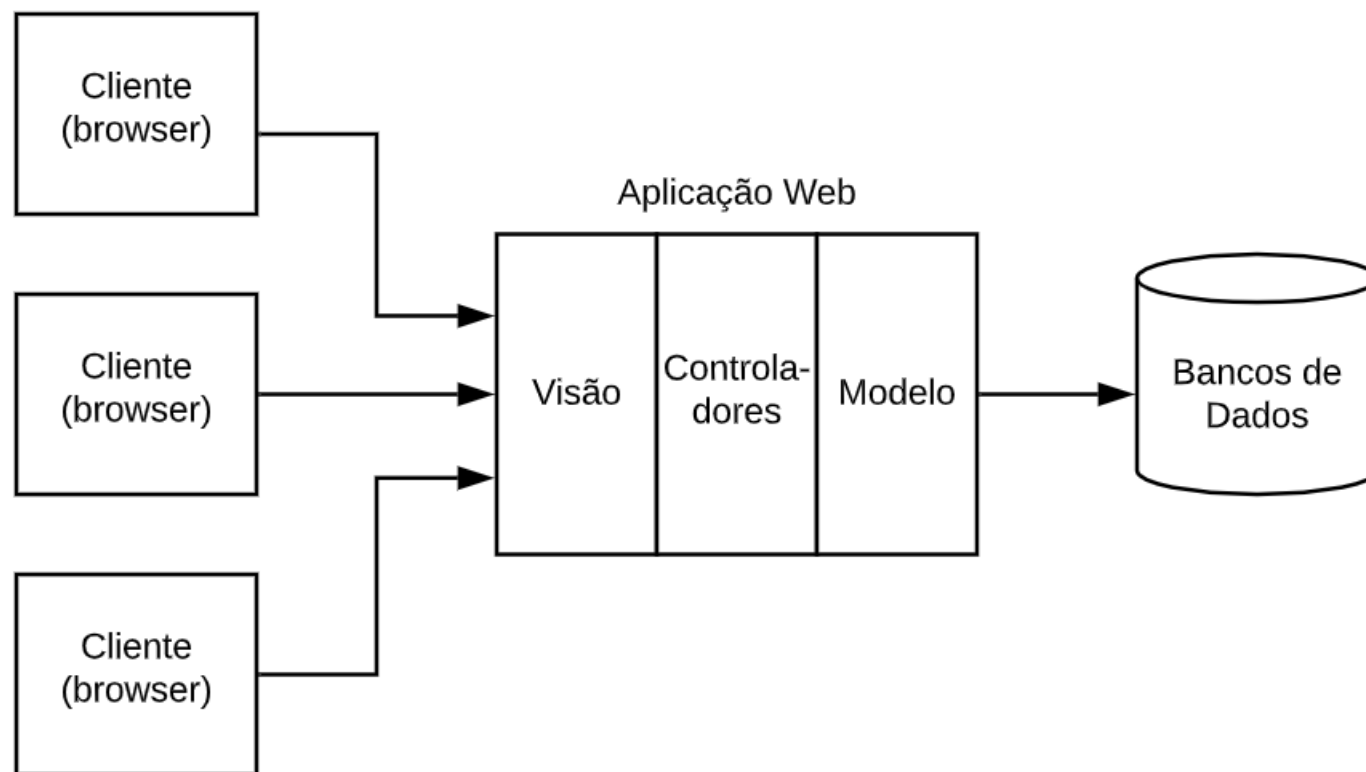
# Arquitetura MVC

Exemplo de arquitetura de aplicações Web usando MVC



Fonte: SOMMERVILLE, 2011.

# Qual a diferença entre MVC e três camadas?



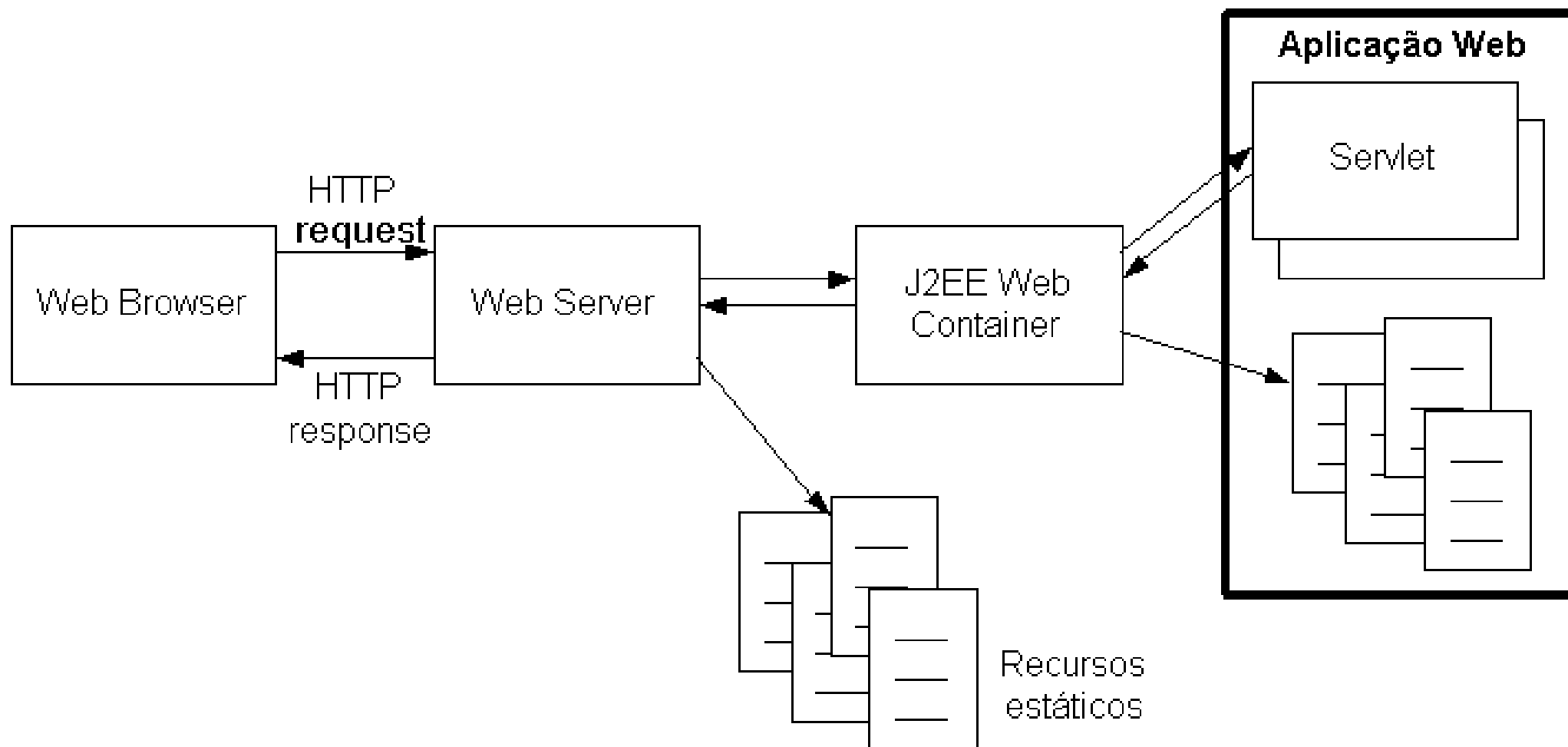
Fonte: VALENTE, 2020.

# Vantagens de arquiteturas MVC

- **Favorece a especialização do trabalho de desenvolvimento.** Por exemplo, pode-se ter desenvolvedores trabalhando na interface gráfica, e desenvolvedores de classes de Modelo que não precisam lidar com aspectos da interface gráfica.
- **Permite que classes de Modelo sejam usadas por diferentes Visões.** Uma mesma informação tratada nas classes de Modelo pode ser apresentada de formas (visões) diferentes.
- **Favorece testabilidade.** É mais fácil testar objetos não relacionados com a implementação de interfaces gráficas.

# Servlets

# Visão geral do funcionamento de servlets



Fonte: <http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/servlet/html/intro.htm>

# Estrutura de um projeto web em Java

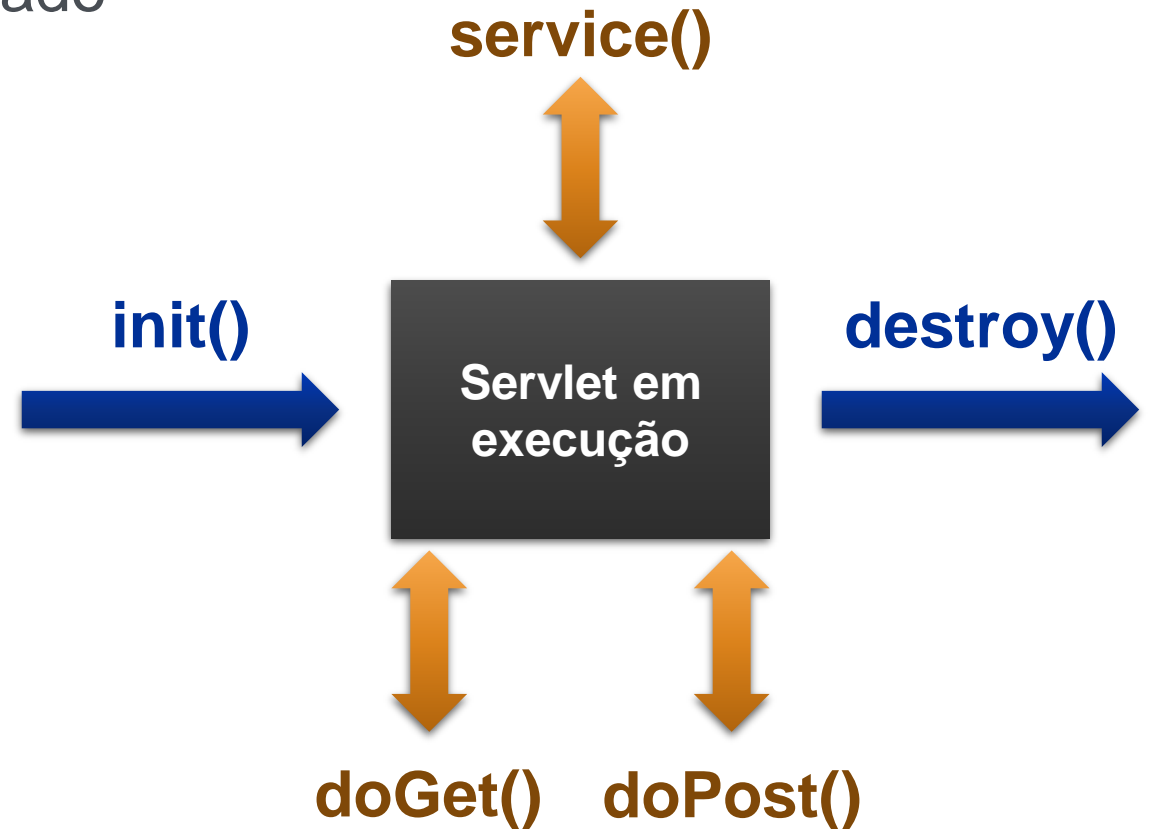
- **src/** - código-fonte Java que gera os servlets, beans, e outras classes (.java);
- **target/** - armazenamento temporário da classes compiladas (.class);
- **webapp/** - conteúdo acessível pelo cliente (html, jsp, imagens, css, etc.);
- **webapp/WEB-INF/** - arquivos de configuração do projeto;
- **webapp/WEB-INF/lib/** - bibliotecas necessárias para a aplicação web (.jar);
- **webapp/WEB-INF/classes/** - armazena arquivos compilados (.class);

# Exemplo de Servlet

```
public class PrimeiroServlet extends HttpServlet {  
    @Override  
    public void service(ServletRequest req, ServletResponse res)  
        throws ServletException, IOException {  
        PrintWriter saida = res.getWriter();  
        saida.println("<html>");  
        saida.println("<head>");  
        saida.println("<title>Primeiro Servlet</title>");  
        saida.println("</head>");  
        saida.println("<body>");  
        saida.println("<h1>Exemplo de Servlet</h1>");  
        saida.println("</form>");  
        saida.println("</body>");  
        saida.println("</html>");  
    }  
}
```

# Ciclo de vida de servlets

- O ciclo de vida de um servlet é determinado por três métodos principais:
  - **init()**: executado quando o container inicia o servlet;
  - **service()**: utilizado para gerenciar as requisições (em conjunto com outros métodos como o **doGet** , **doPost**);
  - **destroy()**: chamado quando o container encerra o servlet.





# Deployment da aplicação web em Java

- Aplicações web em Java são distribuídas no formato **WAR** (**W**eb **AR**chive).
- O arquivo contém todos os componentes necessários para o funcionamento da aplicação.
- O **servidor de aplicação** (Tomcat) identifica todos os servlets presentes no pacote WAR e **faz a chamada do método init() para cada servlet**.
- Um arquivo de configuração **descritor** (web.xml) é necessário para **indicar ao servidor de aplicação a existência de servlets**.

# Descritor web.xml

- Documento XML que armazena informações de configuração e de implantação de uma aplicação web Java.
- Localizado no diretório **WEB-INF**.

```
<?xml version="1.0" encoding="utf-8" ?>
<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_5_0.xsd"
         version="5.0">
  <display-name>Primeiro Servlet</display-name>
  <description>Exemplo de um servlet.</description>
  <servlet>
    <servlet-name>PrimeiroServlet</servlet-name>
    <servlet-class>br.ufac.webacademy.PrimeiroServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>PrimeiroServlet</servlet-name>
    <url-pattern>/primeiroServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

# Deploy com Maven

## pom.xml

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <url>http://localhost:8080/manager/text</url>
    <server>Tomcat</server>
    <path>/${project.artifactId}</path>
  </configuration>
</plugin>
```

## %USERPROFILE%\m2\settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <servers>
    <server>
      <id>Tomcat</id>
      <username>tomcat</username>
      <password>tomcat</password>
    </server>
  </servers>
</settings>
```

## Tomcat: conf\tomcat-users.xml

```
<user username="tomcat" password="tomcat"
roles="admin-gui,manager-gui,manager-script" />
```

## Comandos

```
>mvn tomcat7:deploy
>mvn tomcat7:undeploy
>mvn tomcat7:redploy
```

**JDBC**

# Operações CRUD

- CRUD é um acrônimo para quatro **operações básicas de manipulação de dados**.
- Essas operações são **essenciais para qualquer aplicação que utilize banco de dados**, mesmo que o acesso não seja direto.

	Operação	Instrução SQL
C	Create	INSERT
R	Read	SELECT
U	Update	UPDATE
D	Delete	DELETE

# SQL para operações CRUD

- Create:
  - **INSERT INTO** nome\_tabela (coluna1, coluna2, ...) **VALUES** (valor1, valor2, ...);
- Read:
  - **SELECT \* FROM** nome\_tabela;
- Update:
  - **UPDATE** nome\_tabela **SET** coluna1 = valor1, coluna2 = valor2, ... **WHERE** condição;
- Delete:
  - **DELETE FROM** nome\_tabela **WHERE** condição;

# JDBC

- O JDBC (**J**ava **D**ata**B**ase **C**onnectivity) consiste de um conjunto de classes e interface com suporte a vários comando **SQL**;
- Aumentou mais ainda portabilidade de aplicações Java, que eram independentes de plataforma agora poderiam ser também independentes de **SGBD**;
  - Aplicativos que usavam um **SGBD** poderia ter seu **SGBD** trocado sem modificar uma linha de código.
- A API **JDBC** fornece um mecanismo para:
  - carregar (em tempo de execução) o driver de um determinado SGDB;
  - registrar esse driver no gerenciador de drivers (JDBC Driver Manager);
  - criar conexões;
  - executar instruções SQL;

# Usando a API JDBC

- Uma aplicação JDBC acessa a fonte de dados usando um *DriverManager*,
  - Esta classe requer uma aplicação para carregar um driver específico, usando uma URL para a classe que contém o driver;
- A conexão é criada usando o método estático *getConnection* do *DriverManager*, passando três parâmetros: a URL para o Banco, o usuário e a senha;
  - `Connection con = DriverManager.getConnection();`
- Formato da URL depende do fabricante.
- As chamadas dos métodos devem usar blocos protegidos (try...catch), pois geram exceções.



# Exemplos de URLs

- **MySQL**

- `com.mysql.cj.jdbc.Driver`
- `jdbc:mysql://nomeDoHost/nomeDoBanco`

- **Oracle**

- `oracle.jdbc.driver.OracleDriver`
- `jdbc:oracle:thin:@nomeDoHost:numeroDaPorta:nomeDoBanco`

# Execução de instruções SQL

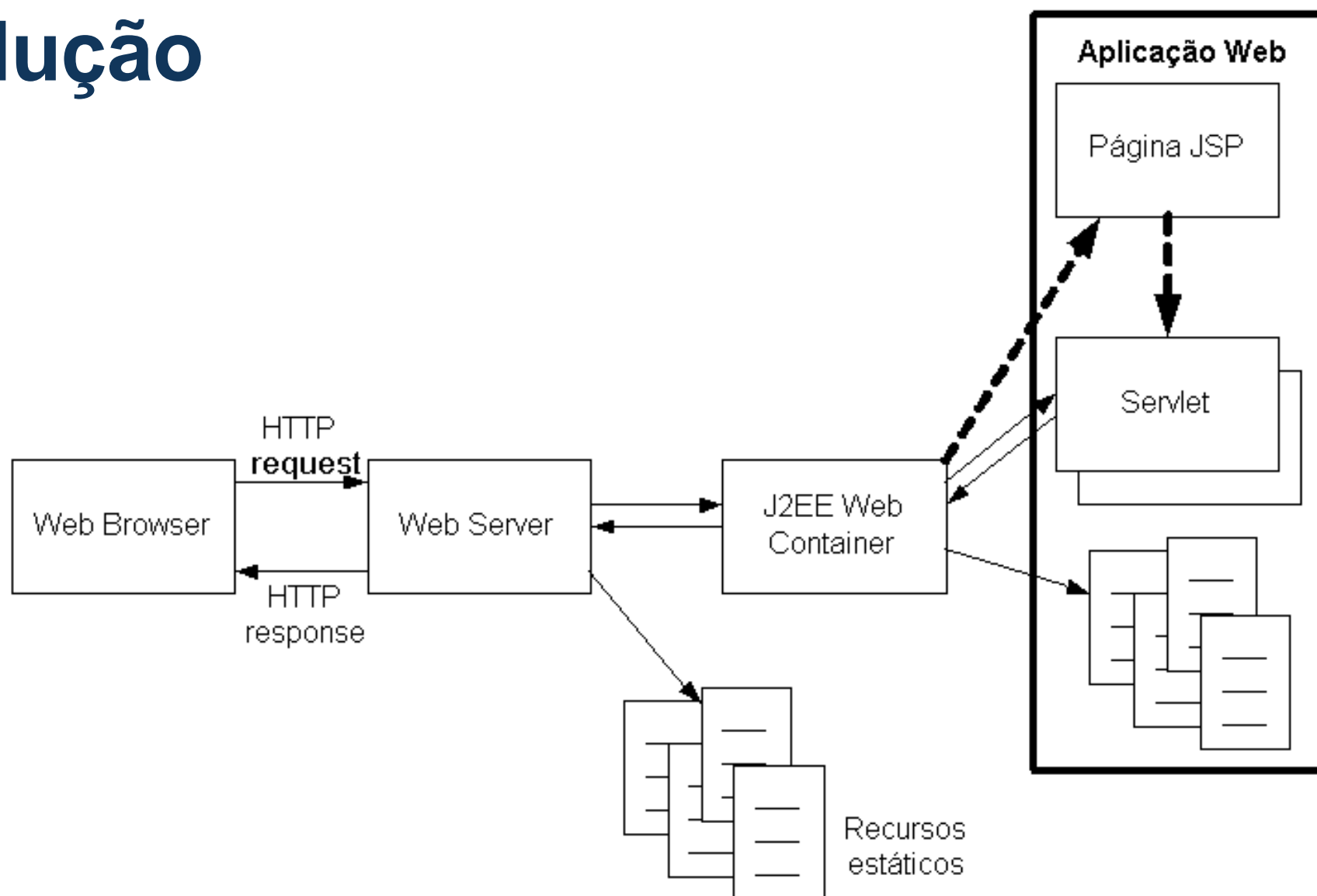
Método	Descrição	Retorna
<code>execute()</code>	Executa qualquer instrução SQL	TRUE/FALSE
<code>executeQuery()</code>	Normalmente usado para instruções SELECT	ResultSet
<code>executeUpdate()</code>	Usado para as demais instruções (INSERT, UPDATE, DELETE, CREATE, DROP, etc.)	Número de registros afetados

**JSP**

# Introdução

- **Jakarta Server Pages (JSP)** é a tecnologia que facilita a criação de conteúdo dinâmico para Web utilizando a linguagem Java;
- Separa a apresentação da lógica de negócio, responsável pela produção do conteúdo;
- Isso facilita a separação da aplicação web em camadas, onde:
  - Programadores front-end concentram-se na interface de usuário (HTML, CSS, JavaScript);
  - Programadores back-end voltam-se para a escrita de código Java.
- O mesmo código Java pode ser utilizado com um front-end feito em Swing (desktop), por exemplo, e também em JSP (web).

# Introdução



Fonte: <http://www.dsc.ufcg.edu.br/~jacques/cursos/daca/html/servlet/html/intro.htm>

# HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título</title>
  </head>
  <body>
    <p>Conteúdo</p>
  </body>
</html>
```

# JSP

```
<!DOCTYPE html>
<html>
  <head>
    <title>Título</title>
  </head>
  <body>
    <%
      String nome = "Daniel";
    %>
    <p><%= nome %></p>
  </body>
</html>
```

# Diretivas

- Diretivas são utilizadas para enviar mensagens ao contêiner que controla as páginas JSP, e podem ser de 3 tipos:

1. **page**: define um conjunto de propriedades de uma página JSP.

```
<%@ page pageEncoding="UTF-8"%>
```

2. **taglib**: amplia o conjunto de tags que o JSP pode interpretar.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

3. **include**: insere o conteúdo de um arquivo na página JSP.

```
<%@ include file="pagina.jsp"%>
```

# Ações-padrão

- Usadas para manipular páginas:
  - **<jsp:include>**
    - inclui dinamicamente algum recurso no JSP
  - **<jsp:forward>**
    - encaminha o processamento para outro recurso
  - **<jsp:param>**
    - especifica algum parâmetro para as outras ações
- Usadas para manipular JavaBean:
  - **<jsp:useBean>**
    - permite o JSP usar uma instância de um JavaBean
  - **<jsp:setProperty>**
    - define uma propriedade na instância do JavaBean
  - **<jsp:getProperty>**
    - obtém o valor de uma propriedade na instância do JavaBean



# Objetos implícitos

Objeto	Tipo	Descrição
request	<code>jakarta.servlet.HttpServletRequest</code>	Dados da requisição (incluindo os parâmetros)
response	<code>jakarta.servlet.HttpServletResponse</code>	Dados da resposta a uma requisição.
pageContext	<code>jakarta.servlet.jsp.PageContext</code>	Informações de contexto de uma página JSP.
session	<code>jakarta.servlet.http.HttpSession</code>	Dados da sessão criada para cada cliente.
application	<code>jakarta.servlet.ServletContext</code>	Dados compartilhadas por todas as páginas JSP da aplicação.
out	<code>jakarta.servlet.jsp.JspWriter</code>	Controle o fluxo de saída (escrever na página JSP).
config	<code>jakarta.servlet.ServletConfig</code>	Acesso as configurações do servlet.
page	<code>java.lang.Object</code>	Instância da página que processa a requisição atual.
exception	<code>java.lang.Throwable</code>	Erros (ou exceções) não capturados.

# Tratamento de exceção

- Há 3 maneiras de tratar exceções em JSP:
  1. Por meio de blocos **try...catch** na própria página;
  2. Utilizando as diretivas **errorPage** e **isErrorPage**;
  3. Configurando elementos do tipo `<error-page>` no **web.xml**.

# Tratamento de exceção: bloco try...catch

<%

```
try {  
    throw new RuntimeException("Erro!");  
} catch (Exception e) {  
    e.printStackTrace(  
        response.getWriter()  
    );  
}
```

%>

# Tratamento de exceção: diretivas

Página onde a exceção é lançada

```
<%@ page errorPage="error.jsp" %>  
<%  
    throw new RuntimeException("Erro!");  
%>
```

Página que trata a exceção

```
<%@ page isErrorPage="true" %>  
<%  
    exception.printStackTrace(  
        response.getWriter()  
    );  
%>
```

# Tratamento de exceção: web.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<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_5_0.xsd"
  version="5.0">
  <display-name>Primeiro Servlet</display-name>
  <description>Exemplo de um servlet.</description>
  <error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/erro.jsp</location>
  </error-page>
</web-app>
```

# Segurança

Usuário e senha sendo capturados no Wireshark

http									
	Time	Source	Destination	Protocol	Length	Info			
81	10.751746988	10.0.2.15	176.28.50.165	HTTP	549	[TCP Prev]			
87	11.023756075	176.28.50.165	10.0.2.15	HTTP	71	HTTP/1.1 2			
89	11.112734635	10.0.2.15	176.28.50.165	HTTP	480	GET /Flash			
92	104.364660492	10.0.2.15	192.16.58.8	OCSP	485	Request			
94	104.405805104	192.16.58.8	10.0.2.15	OCSP	842	Response			
33	104.866913950	10.0.2.15	177.69.134.249	HTTP	342	GET /succe			
35	104.906092022	177.69.134.249	10.0.2.15	HTTP	438	HTTP/1.1 2			
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n									
Accept-Language: en-US,en;q=0.5\r\n									
Accept-Encoding: gzip, deflate\r\n									
Referer: http://testphp.vulnweb.com/login.php\r\n									
Content-Type: application/x-www-form-urlencoded\r\n									
0120	65 6e 3b 71 3d 30 2e 35 0d 0a	41 63 63 65 70 74	en;q=0.5 ..Accept						
0130	2d 45 6e 63 6f 64 69 6e 67 3a	20 67 7a 69 70 2c	-Encodin g: gzip,						
0140	20 64 65 66 6c 61 74 65 0d 0a	52 65 66 65 72 65	deflate ..Refere						
0150	72 3a 20 68 74 74 70 3a 2f 2f	74 65 73 74 70 68	r: http: //testph						
0160	70 2e 76 75 6c 6e 77 65 62 2e	63 6f 6d 2f 6c 6f	p.vulnwe b.com/lo						
0170	67 69 6e 2e 70 68 70 0d 0a 43	6f 6e 74 65 6e 74	gin.php · ·Content						
0180	2d 54 79 70 65 3a 20 61 70 70	6c 69 63 61 74 69	-Type: a pplicati						
0190	6f 6e 2f 78 2d 77 77 77 2d 66	6f 72 6d 2d 75 72	on/x-www -form-ur						
01a0	6c 65 6e 63 6f 64 65 64 0d 0a	43 6f 6e 74 65 6e	lencoded · ·Conten						
01b0	74 2d 4c 65 6e 67 74 68 3a 20	32 30 0d 0a 43 6f	t-Length : 20 · ·Co						
01c0	6f 6b 69 65 3a 20 6c 6f 67 69	6e 3d 74 65 73 74	okie: lo gin=test						
01d0	25 32 46 74 65 73 74 0d 0a 43	6f 6e 6e 65 63 74	%2Ftest · ·Connect						
01e0	69 6f 6e 3a 20 6b 65 65 70 2d	61 6c 69 76 65 0d	ion: kee p-alive·						
01f0	0a 55 70 67 72 61 64 65 2d 49	6e 73 65 63 75 72	·Upgrade -Insecur						
0200	65 2d 52 65 71 75 65 73 74 73	3a 20 31 0d 0a 0d	e-Request: 1···						
0210	0a 75 6e 61 6d 65 3d 74 65 73	74 26 70 61 73 73	·uname=t est&pass						
0220	3d 74 65 73 74		=test						
eth0: <live capture in progress>									

Fonte: <https://tavernalinix.com/wireshark-capturando-pacotes-de-login-e-senha-do-telnet-e-http-com-wireshark-3180f7bd2f9>

# SSL/TLS

- **SSL** (**S**ecure **S**ockets **L**ayer) permite o tráfego de dados pela rede de forma segura, estabelecendo um **canal de comunicação entre aplicações onde as informações são criptografadas**.
- **TLS** (**T**ransport **L**ayer **S**ecurity) é o **successor do SSL** e funciona de forma semelhante.
  - Apesar do termo SSL ser mais popular, na maioria das vezes o termo correto que deveria ser utilizado é TLS.
- O protocolo **HTTPS** é uma implementação do HTTP com uma camada adicional de segurança (HTTPS = HTTP + SSL/TLS).

# Segurança: Habilitar SSL no Tomcat

## Criar o keystore

```
> keytool -genkey -alias tomcat -keyalg RSA -keystore  
"c:\Program Files\Apache Software Foundation\Tomcat  
10.0\conf\keystore.jks"
```

## Outras opções de configuração

<https://tomcat.apache.org/tomcat-10.0-doc/ssl-howto.html>

## Configuração Tomcat: conf/sever.xml

```
<Connector  
  port="8443"  
  protocol="org.apache.coyote.http11.Http11NioProtocol"  
  maxThreads="150"  
  SSLEnabled="true">  
  <SSLHostConfig>  
    <Certificate  
      certificateKeystoreFile="conf/keystore.jks"  
      certificateKeystorePassword="tomcat"  
      type="RSA" />  
    </SSLHostConfig>  
  </Connector>
```



# Segurança: Forçar utilização de SSL

```
<security-constraint>
```

```
<web-resource-collection>
```

```
<web-resource-name>AcessoSeguro</web-resource-name>
```

```
<url-pattern>/*</url-pattern>
```

```
</web-resource-collection>
```

**Indica área da aplicação que será afetada**

```
<user-data-constraint>
```

```
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

```
</user-data-constraint>
```

**Exige utilização de SSL**

```
</security-constraint>
```

**Fim!**



# Referências

- DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar**. 10. ed. São Paulo: Pearson, 2016. 968 p.
- ORACLE; ECLIPSE FOUNDATION (ed.). **Jakarta Server Pages Specification**. [S. l.], 2022. Disponível em: <https://jakarta.ee/specifications/pages/3.0/jakarta-server-pages-spec-3.0.html>
- ORACLE; ECLIPSE FOUNDATION (ed.). **Jakarta Servlet Specification**. [S. l.], 2022. Disponível em: <https://jakarta.ee/specifications/servlet/5.0/jakarta-servlet-spec-5.0.html>
- MARCO TULIO VALENTE. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**, 2020. Disponível em: <https://engsoftmoderna.info/>
- SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Addison-Wesley, 2011.