WEB ACADEMY

Fundamentos de Programação Back-end

Daniel Augusto Nunes da Silva

Apresentação

Ementa

- Linguagens de programação server-side. Arquitetura em camadas. **Java**, **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 JSP;
  - Permitir ao aluno conhecer e aplicar os recursos básicos necessários para construção de aplicações web com acesso a banco de dados utilizando JDBC;
  - Demonstrar a execução de tarefas relacionadas ao processo de implantação de aplicações web.

Conteúdo programático

Introdução

- Programação server-side;
- Java: sintaxe, modificadores de acesso, estruturas de controle, tipos básicos e arrays;
- Depuração de apps Java no VS Code;
- Arquitetura em camadas e MVC.
- Arquitetura em camadas e pacotes Java.

Java e POO

- Programação orientada a objetos (POO): classes e objetos;
- Encapsulamento, herança e polimorfismo;
- Sobrescrita e sobrecarga de métodos.

JDBC

- Java Beans;
- API do JDBC;
- Sintaxe das principais instruções SQL usadas em operações CRUD;
- Execução de instruções SQL (Statements e Result Sets).
- SQL Joins.

Servlets e JSP

- Visão geral de Servlets;
- Servidores de aplicação, empacotamento e implantação de aplicações web Java em ambiente de produção;
- Depuração de webapps Java;
- JSP: elementos, ações-padrão, diretivas e objetos implícitos;
- Segurança.

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/3.1/jakarta-server-pages-spec-3.1.html
- Jakarta Servlet Specification
  - https://jakarta.ee/specifications/servlet/6.0/jakarta-servlet-spec-6.0.html
- Apostila Java e Orientação a Objetos (Caelum/Alura)
  - https://www.alura.com.br/apostila-java-orientacao-objetos
- Apostila Java para Desenvolvimento Web (Caelum/Alura)
  - https://www.alura.com.br/apostila-java-web
- Baeldung
  - https://www.baeldung.com/
- Java Tutorial (VS Code)
  - https://code.visualstudio.com/docs/java/java-tutorial

Ferramentas

- **Visual Studio Code:**
  - https://code.visualstudio.com/Download
- **Extension Pack for Java (Extensão do VS Code):**
  - https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack
- **Java Server Pages - JSP (Extensão do VS Code):**
  - https://marketplace.visualstudio.com/items?itemName=pthorsson.vscode-jsp
- **XML (Extensão do VS Code):**
  - https://marketplace.visualstudio.com/items?itemName=redhat.vscode-xml

Ferramentas: JDK 17

- Verificar versão do JDK instalada: **javac -version**
- https://download.oracle.com/java/17/archive/jdk-17.0.6_windows-x64_bin.msi
- Criar a variável de ambiente JAVA_HOME configurada para o diretório de instalação do JDK. Exemplo: “C:\Program Files\Java\jdk-17”.
- Adicionar “%JAVA_HOME%\bin” na variável de ambiente PATH.
- Tutorial de configuração: https://mkyong.com/java/how-to-set-java_home-on-windows-10/

Ferramentas: Maven

- Verificar versão do Maven instalada: **mvn -version**
- 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

- Verifique se o Tomcat está instalado e funcionando:
  - Localize o aplicativo Monitor Tomcat.
  - Acesse a URL http://localhost:8080, que deve exibir uma página indicando que o Tomcat está funcionando.
- Link para download: https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.7/bin/apache-tomcat-10.1.7.exe
- Tutorial de instalação: https://github.com/webacademyufac/tutoriais/blob/main/tomcat/tomcat.md

Ferramentas: MySQL

- Verificar se o MySQL está funcionando:
  - **mysql -u root -p**
  - Tentar acessar com senha em branco ou senha igual ao nome de usuário (root).
  - Tutorial para resetar a senha de root: https://dev.mysql.com/doc/mysql-windows-excerpt/8.0/en/resetting-permissions-windows.html
- Link para download: https://dev.mysql.com/downloads/file/?id=512698
- Tutorial de instalação: https://github.com/webacademyufac/tutoriais/blob/main/mysql/mysql.md
- Para criação do banco e importação de dados, a partir do diretório **sql**, executar os comandos:
  - **mysql -u root -p < sgcm.sql**
  - **mysql -u root -p sgcm < dados.sql**

Contato

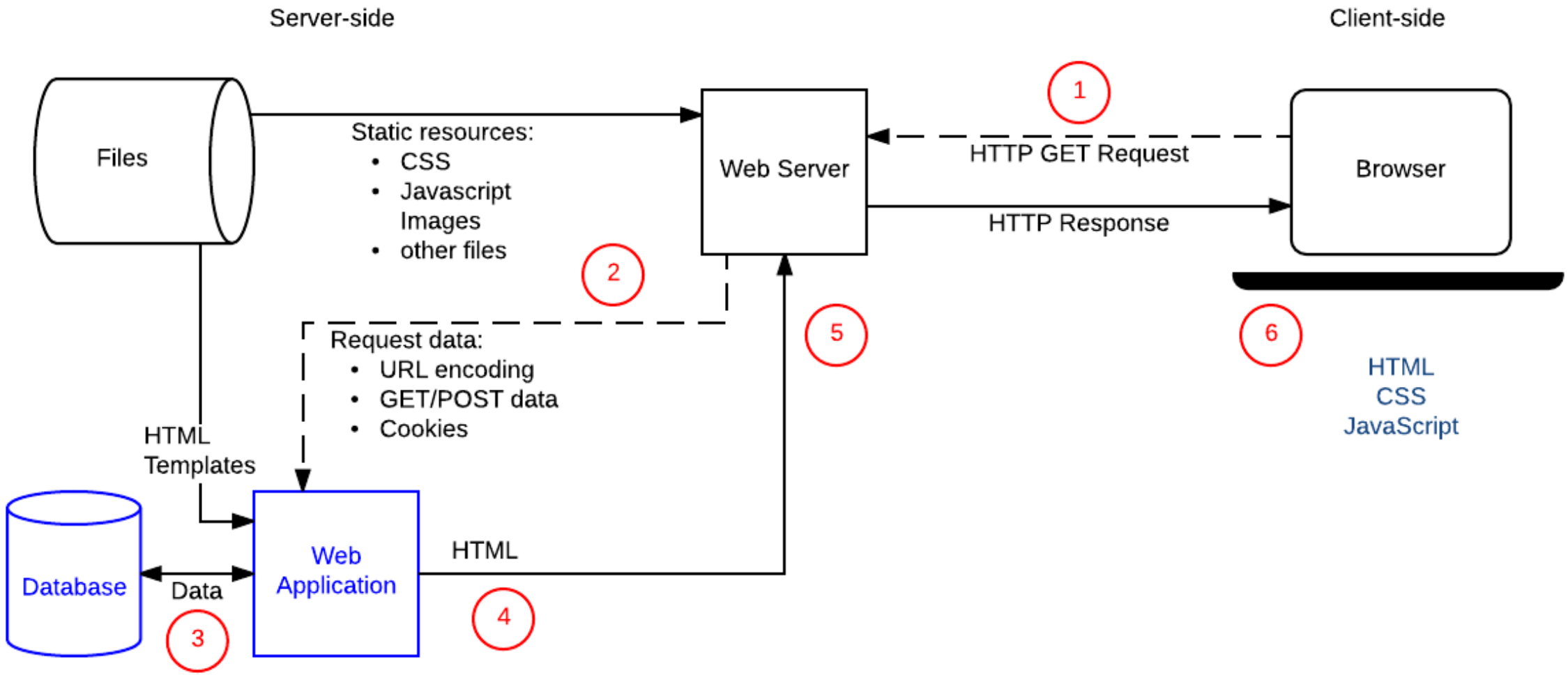
https://linkme.bio/danielnsilva/

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 aos aspectos visuais 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

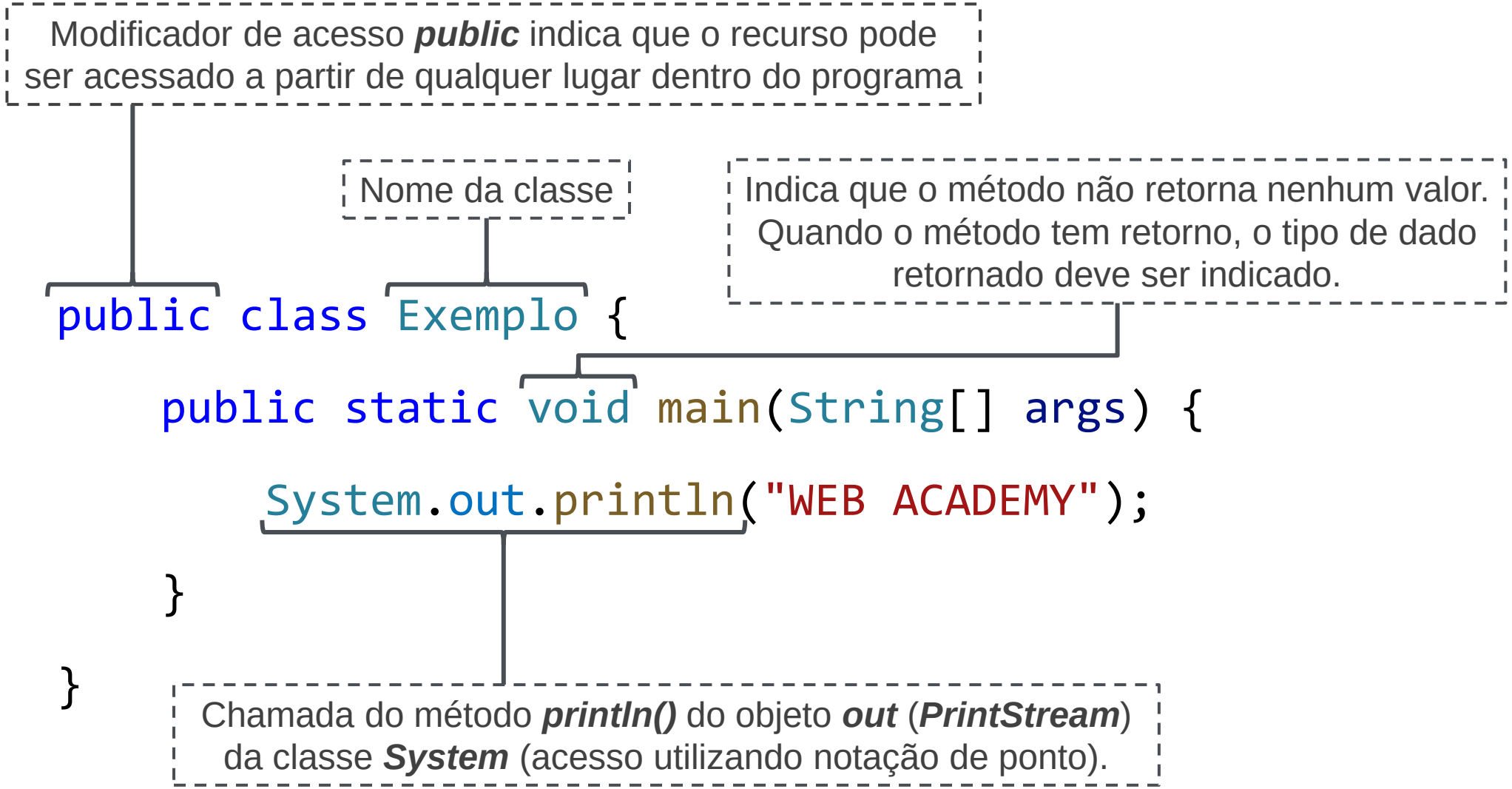
Java

- 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


Modificador de acesso *public* indica que o recurso pode ser acessado a partir de qualquer lugar dentro do programa
Nome da classe
Indica que o método não retorna nenhum valor. Quando o método tem retorno, o tipo de dado retornado deve ser indicado.
public class Exemplo {
public static void main(String[] args) {
System.out.println("WEB ACADEMY");
}
}
Chamada do método *println()* do objeto *out* (*PrintStream*) da classe *System* (acesso utilizando notação de ponto).

Java: modificadores de acesso

- **public**: permite que a classe, método ou variável seja acessado por qualquer código em **qualquer lugar**.
- **private**: permite que a classe, método ou variável seja acessado somente dentro da própria **classe onde foi definido**.
- **protected**: permite que a classe, método ou variável seja acessado dentro da própria **classe, subclasses e outras classes no mesmo pacote**.
- **default** (ou package-private): permite que a classe, método ou variável seja acessado somente dentro do **mesmo pacote**.

Java: estruturas de controle

```java
int numero = 1;
String mensagem;

// if/else
if (numero == 1) {
    mensagem = "Igual a 1";
} else {
    mensagem = "Maior ou igual 2";
}
System.out.println(mensagem);

// Operador ternário
mensagem = (numero > 3) ? "Maior que 3" : "Menor ou igual a 3";

System.out.println(mensagem);
```

Java: estruturas de controle

```java
int numero = 1;
String mensagem;

// if/else
if (numero == 1) {
    mensagem = "Igual a 1";
} else {
    mensagem = "Maior ou igual 2";
}
System.out.println(mensagem);

// Operador ternário
mensagem = (numero > 3) ? "Maior que 3" : "Menor ou igual a 3";
System.out.println(mensagem);
```

Condição **true** **false**

Java: tipos básicos

- Java é uma linguagem de **tipagem forte e estática** e, portanto, requer que todas a 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
```

Java: arrays

- **Arrays** são estruturas de dados que permitem **armazenar e manipular coleções de elementos do mesmo tipo**.
- Tipos de arrays dinâmicos: ArrayList, LinkedList, Vector, Stack, Queue, Deque.

```java
// Declaração de array estático de 5 posições
int[] numeros = new int[5];

// Declaração de array dinâmico
List<Integer> numeros = new ArrayList<Integer>();

// Acessando um elemento do array estático
int numero = numeros[1];

// Acessando um elemento do array dinâmico
int numero = numeros.get(1);
```

```java
// Percorrendo arrays pelo índice
for (int i = 0; i < numeros.length; i++) {
    System.out.println(numeros[i]);
}


// Percorrendo arrays com loop for-each
for (int numero : numeros) {
    System.out.println(numero);
}
```

Depuração de apps Java no VS Code

- **Breakpoints**: pontos definidos no código onde a execução do programa é interrompida para permitir a depuração.

Step Out: executa o código até o final da função atual e retorna para o ponto onde a função foi chamada.

Continuar/Interromper a execução normal do programa.

Stop: encerra a execução atual do programa e sai do modo de depuração.

Step Over: executa uma linha de código e passa para a próxima linha sem entrar em funções chamadas na linha atual.

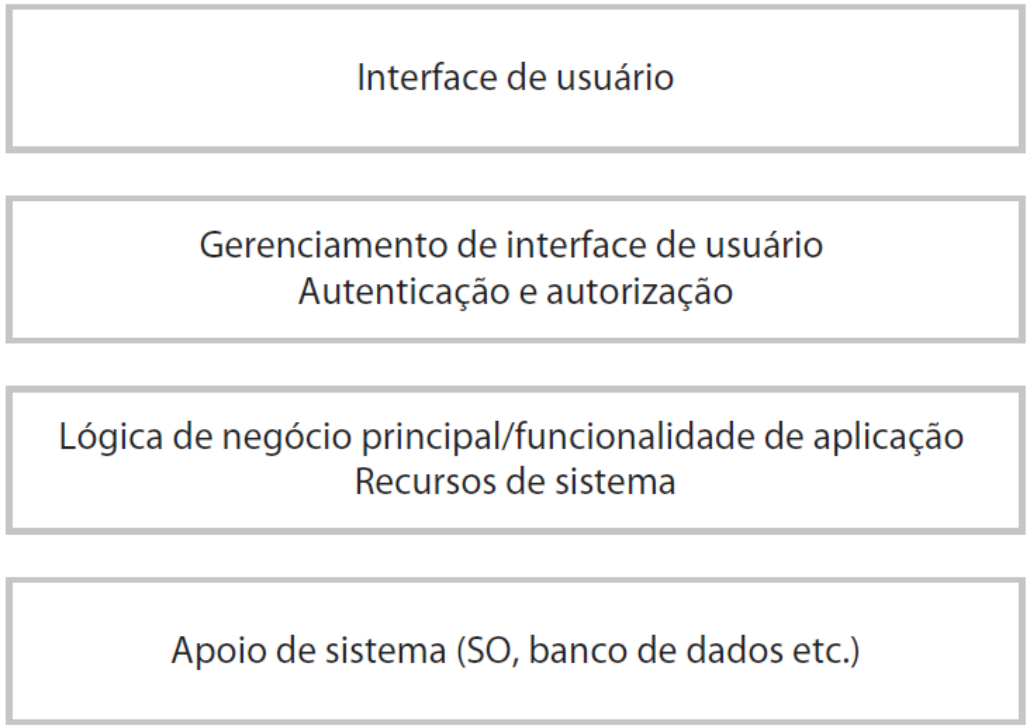
Step Into: executa uma linha de código e entra em funções chamadas na linha atual.

Depuração de apps Java no VS Code

- Referências úteis:
  - https://code.visualstudio.com/docs/editor/debugging
  - https://code.visualstudio.com/docs/java/java-debugging

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.



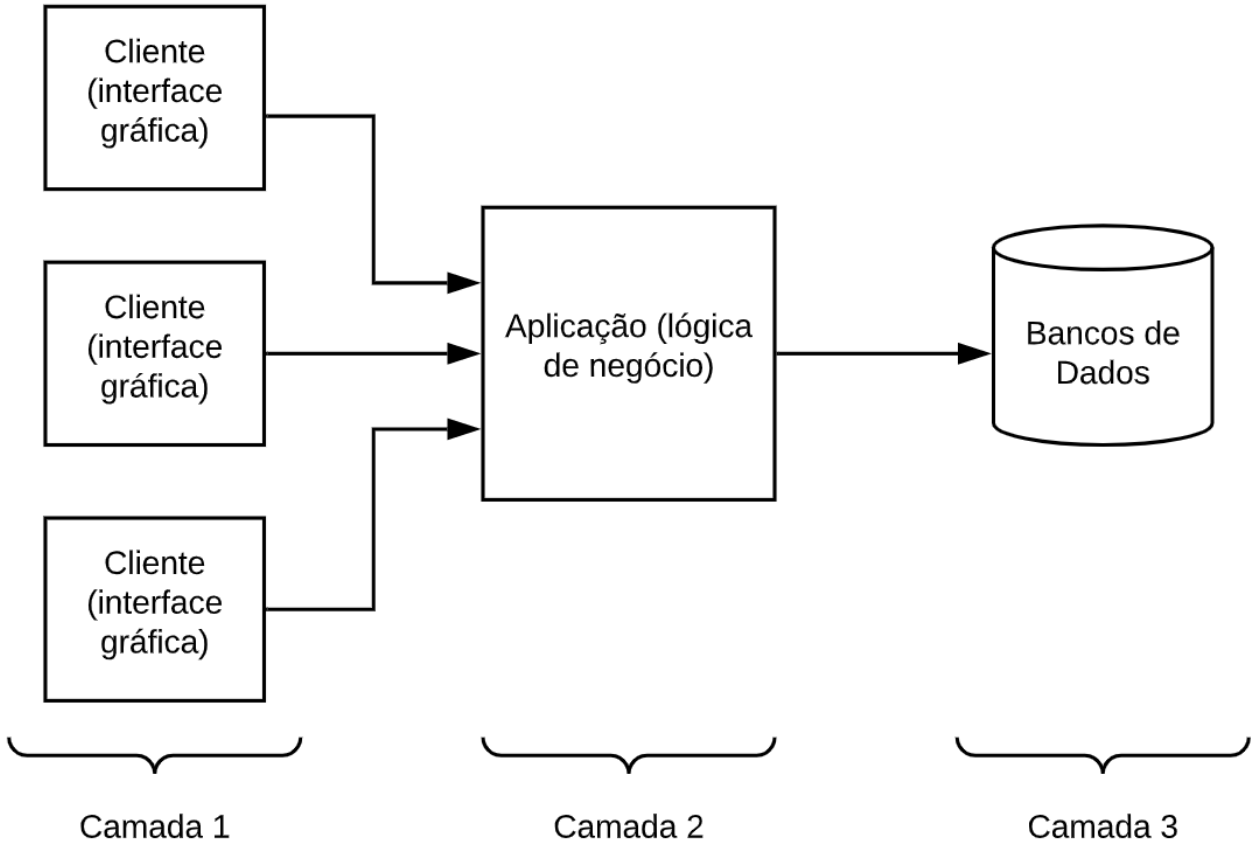
Fonte: SOMMERVILLE, 2011.

- **Particiona a complexidade** envolvida no desenvolvimento de um sistema **em componentes menores** (as camadas), e disciplina as dependências entre essas camadas.

Arquitetura em três camadas

- Tipo de arquitetura **comum na construção de sistemas de informação corporativos**.

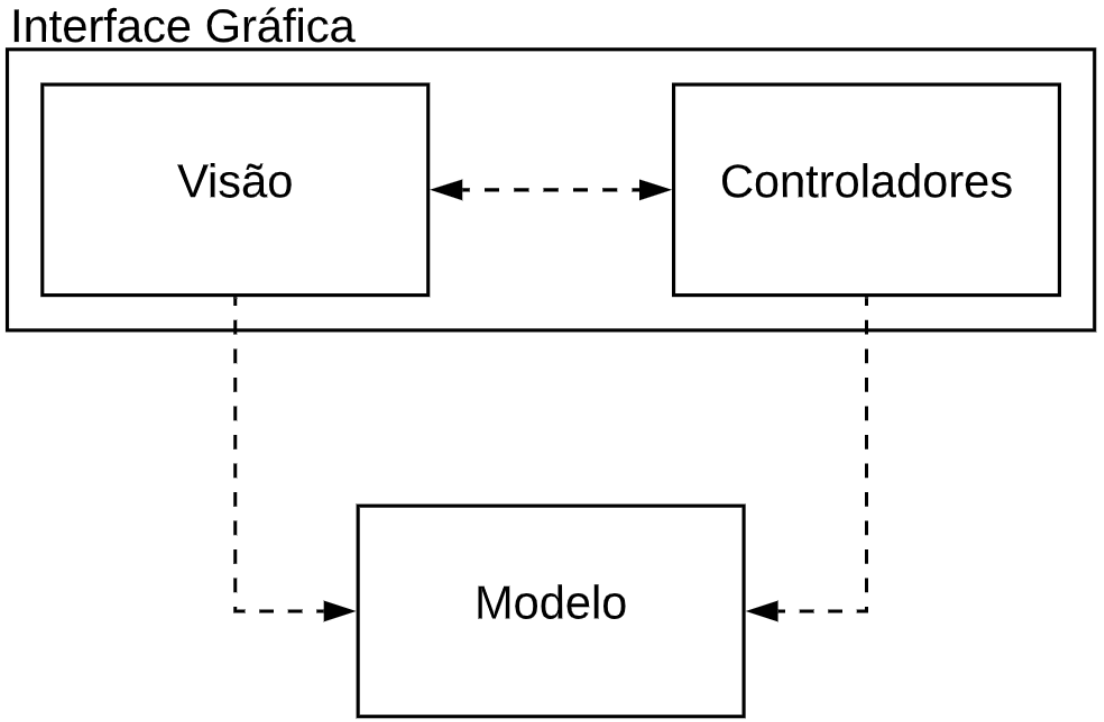
1. **Interface com o Usuário**, responsável por toda interação com o usuário;
2. **Lógica de Negócio**, que implementa as regras de negócio do sistema;
3. **Banco de Dados**, que armazena os dados manipulados pelo sistema.



Fonte: VALENTE, 2020.

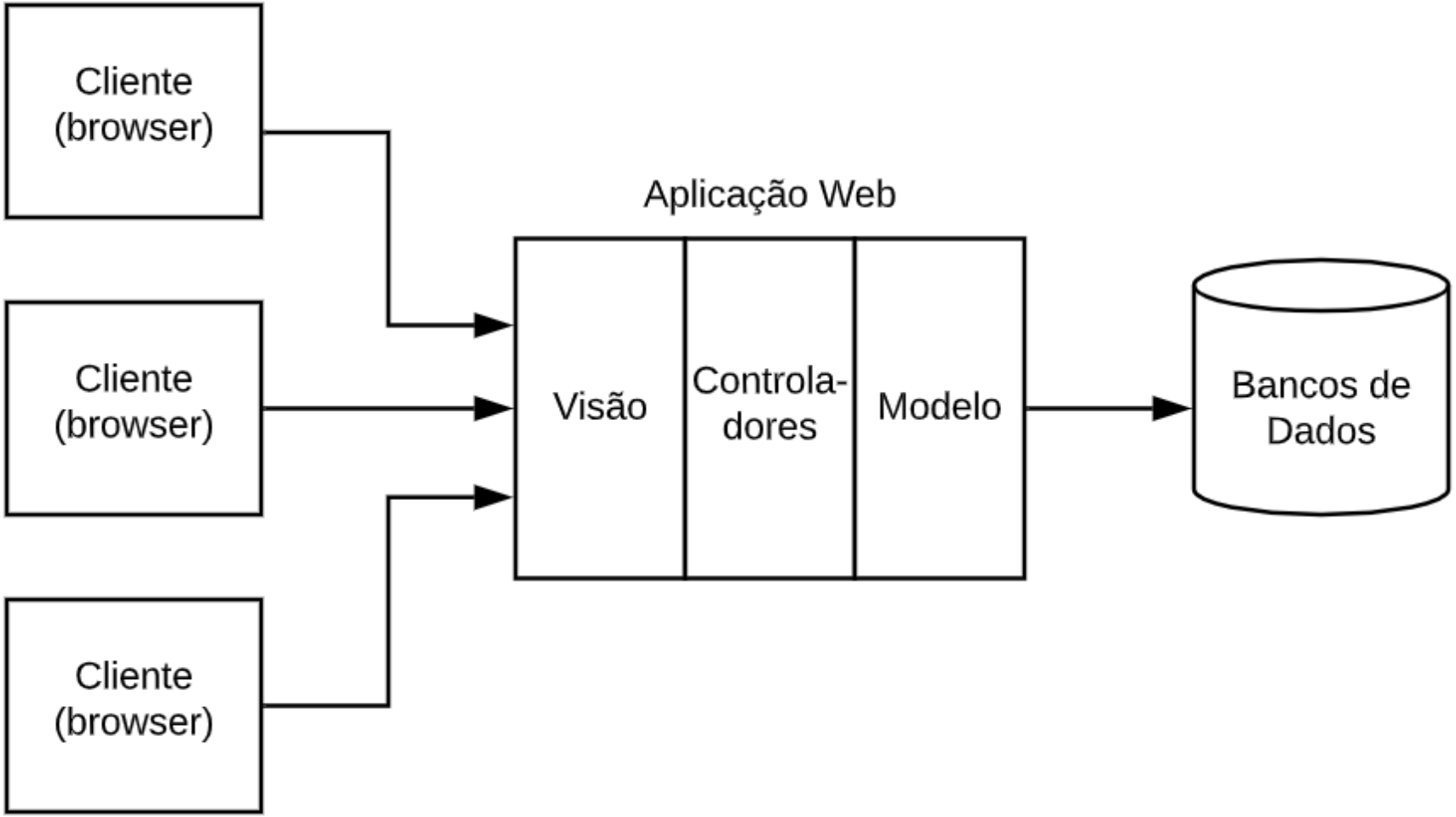
Arquitetura MVC (*Model-View-Controller*)

- **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.

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.

Arquitetura em camadas e pacotes Java

- **Pacotes organizam classes relacionadas**, dividindo o código em módulos lógicos que tornam mais fácil gerenciar projetos complexos.
- O nome do pacote corresponde ao caminho relativo à raiz do diretório que armazena os arquivos fonte. Exemplo: se a raiz é "**/src**", o pacote "**br.ufac.sgcm**" pode ser armazenado no diretório "**/src/br/ufac/sgcm**".

```java
package br.ufac.sgcm;

public class Exemplo {
    // corpo da classe
}
```

```java
import br.ufac.sgcm.Exemplo;

public class OutroExemplo {
    public static void main(String[] args) {
        Exemplo objeto = new Exemplo();
    }
}
```

Não é necessário usar a instrução **import** para acessar classes do mesmo pacote.

Arquitetura em camadas e pacotes Java

Camada (pacote)	Descrição
src\main**java\br\ufac\sgcm\model**	modelos de objetos
src\main**java\br\ufac\sgcm\dao**	acesso a dados e operações de banco de dados
src\main**java\br\ufac\sgcm\controller**	controladores de interface do usuário (lógica de negócio)
src\main**webapp**	recursos da interface do usuário

Java e POO

Programação orientada a objetos (POO)

- **Classe:**
  - Estrutura que abstrai um conjunto de objetos com **características semelhantes**.
  - **POJO** (**P**lain **O**ld **J**ava **O**bject): define uma classe simples, sem recursos especiais.
- **Objeto:**
  - Instância ou modelo **derivado de uma classe**, que pode ser manipulado pelo programa.

```java
public class Pessoa { // Classe
    private String nome;
    private String email;
    public String getNome() {}
    public void setNome(String nome) {}
    public String getEmail() {}
    public void setEmail(String email) {}
}
public class Exemplo {
    public static void main(String[] args) {
        Pessoa p = new Pessoa(); // Objeto
    }
}
```

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.

```
public class Pessoa {

    private String nome;
    private String email;

    public String getNome() {}
    public void setNome(String nome) {}
    public String getEmail() {}
    public void setEmail(String email) {}

}
```

POO: Herança

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

```
public class Pessoa { // Superclasse
    private String nome;
    private String email;
    public String getNome() {}
    public void setNome(String nome) {}
    public String getEmail() {}
    public void setEmail(String email) {}
}
public class Aluno extends Pessoa { // Subclasse
    private int matricula;
    public int getMatricula() {}
    public void setMatricula(int matricula) {}
}
```

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**, que não pode ser instanciada, servindo de base para outras classes.

```
public abstract class Quadrilatero {
    public abstract double calcularArea();
}
public class Quadrado extends Quadrilatero {
    private double lado;
    public Quadrado(double lado) {
        this.lado = lado;
    }
    public double calcularArea() {
        return this.lado * this.lado;
    }
}
```

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.

```
public interface Quadrilatero {
    double calcularArea();
}
public class Quadrado implements Quadrilatero {
    private double lado;
    public Quadrado(double lado) {
        this.lado = lado;
    }
    public double calcularArea() {
        return this.lado * this.lado;
    }
}
```

Sobrescrita e sobrecarga de métodos

- **Sobrescrita**: um método na subclasse possui o **mesmo nome, tipo de retorno e parâmetros** que um método na superclasse.
- **Sobrecarga**: ocorre quando dois ou mais métodos na mesma classe têm o **mesmo nome e tipo de retorno, mas parâmetros diferentes**.

```
public abstract class Quadrilatero {
    public abstract double calcularArea();
}
public class Quadrado extends Quadrilatero {
    // Sobrescrita do método calcularArea()
    @Override
    public double calcularArea() {
        return this.lado * this.lado;
    }
    // Sobrecarga do método calcularArea()
    public double calcularArea(double diagonal) {
        return (diagonal * diagonal) / 2;
    }
}
```

JDBC

Java Beans

- Classes padronizadas que **encapsulam características de objetos seguindo um conjunto de convenções**, podendo ser utilizadas para **representar entidades do banco de dados** em projetos Java.
  - Atributos privados.
  - Acesso por meio dos métodos *getters* e *setters*.
  - Método construtor sem argumentos.
  - Implementa a interface *Serializable*.

```java
// Classe (Java Bean)
public class Pessoa implements Serializable {
  private String nome; // Atributo privado
  public Pessoa() {} // Construtor
  public String getNome() { // Getter
    return nome;
  }
  public void setNome(String nome) { // Setter
    this.nome = nome;
  }
}
```

JDBC

- O **JDBC** (**J**ava **D**ata**B**ase **C**onnectivity) consiste de um conjunto de classes e interfaces que dão suporte a execução de comandos **SQL**;
- Favorece a portabilidade de aplicações Java, que podem ser independentes de plataforma e **SGBD**;
- Um aplicação poderia trocar o **SGBD** sem necessidade de mudanças significativas no 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;

Configuração do projeto – pom.xml

```
<dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.32</version>
  </dependency>
</dependencies>
```

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 = DriverManange.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

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**.

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

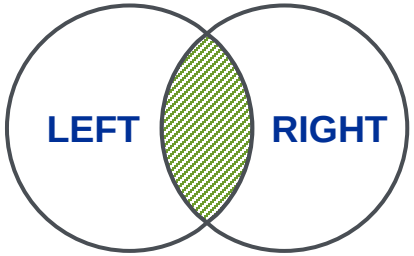
SQL para operações CRUD

- 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;**

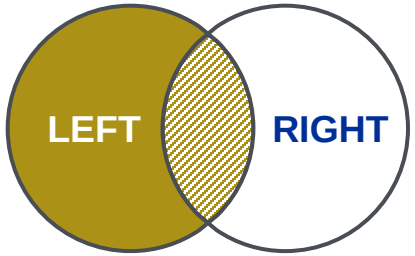
Execução de instruções SQL

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

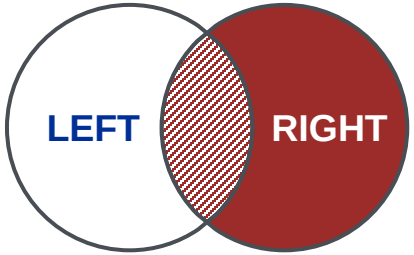
SQL Joins



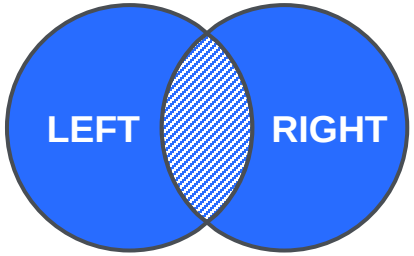
INNER JOIN retorna apenas as linhas que têm correspondência em ambas as tabelas (ou seja, onde a condição de junção é verdadeira).



LEFT JOIN retorna todas as linhas da tabela à esquerda e as correspondentes da tabela à direita. Valores nulos são retornados se não houver correspondência na tabela à direita. As linhas da tabela à esquerda são sempre incluídas no resultado.



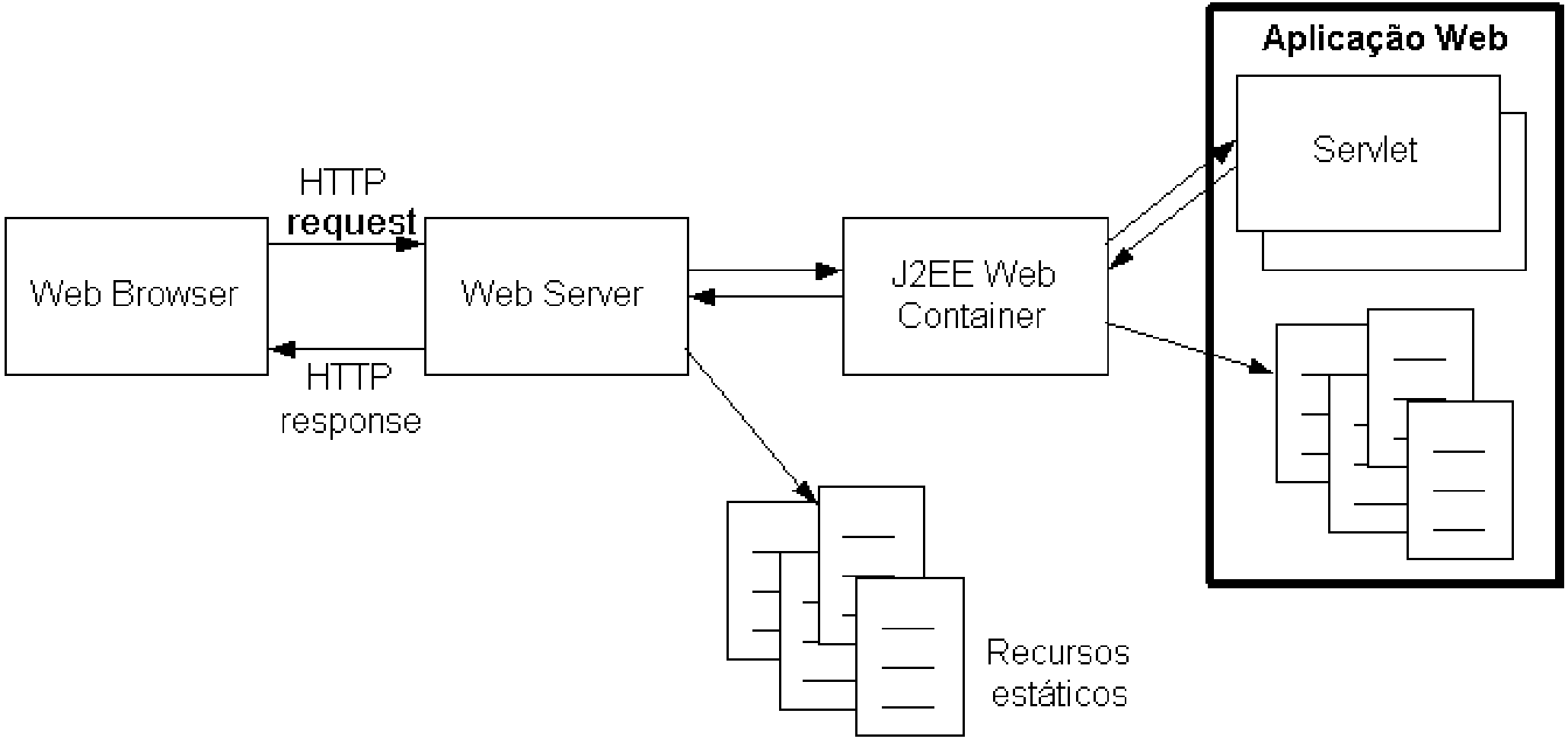
RIGHT JOIN é semelhante ao LEFT JOIN, mas retorna todas as linhas da tabela à direita e as correspondentes da tabela à esquerda. Se não houver correspondência na tabela à esquerda, o resultado contém NULL nos valores da tabela à esquerda.



FULL JOIN retorna todas as linhas de ambas as tabelas, incluindo aquelas sem correspondência em uma ou em ambas as tabelas. Se não houver correspondência em uma tabela, o resultado conterá NULL para as colunas daquela tabela.

Servlets e JSP

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 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);

Configuração do projeto – pom.xml

- O arquivo **pom.xml** (**POM** - **P**roject **O**bject **M**odel) contém informações do projeto e informações de configuração (como dependências e plugins) para o **maven** compilar o projeto.

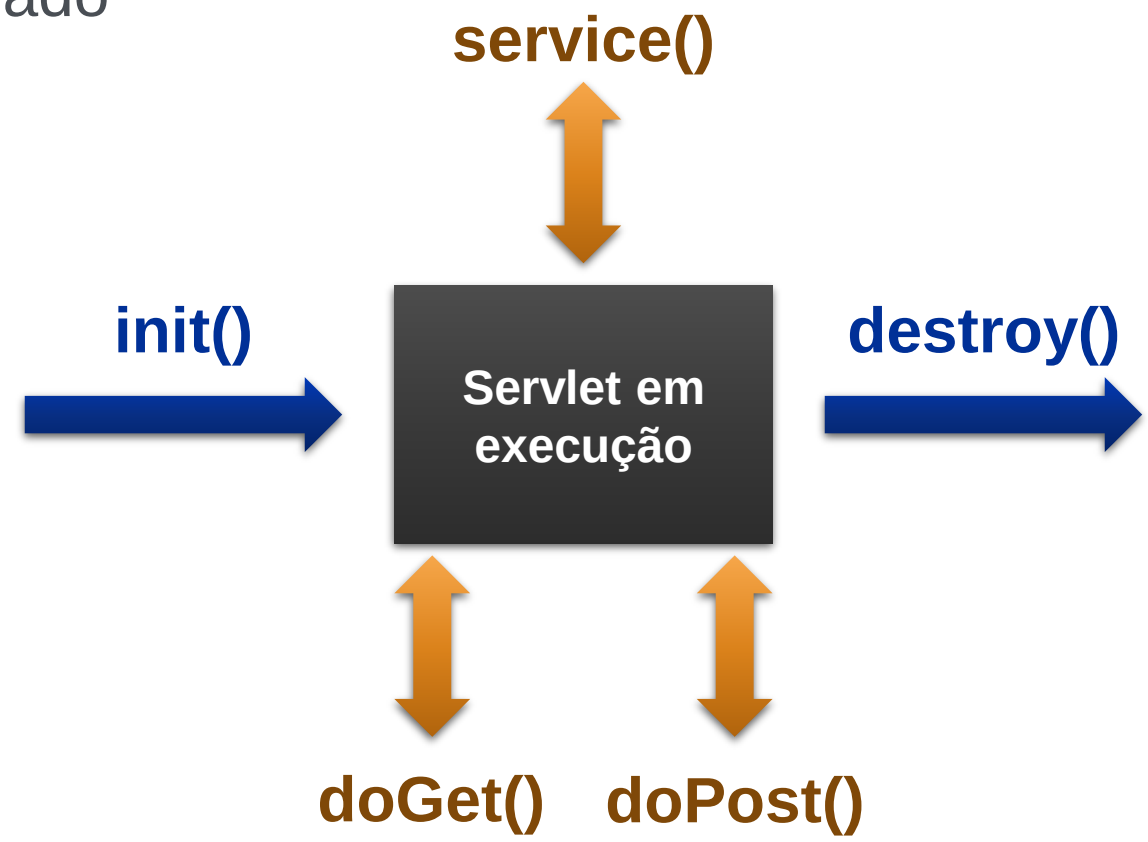
```
<dependencies>
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>5.0.0</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

Exemplo de Servlet

```java
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("</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** e **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_6_0.xsd“
         version="6.0">
    <display-name>Primeiro Servlet</display-name>
    <description>Exemplo de um servlet.</description>
    <servlet>
        <servlet-name>PrimeiroServlet</servlet-name>
        <servlet-class>br.ufac.sgcm.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" />
```

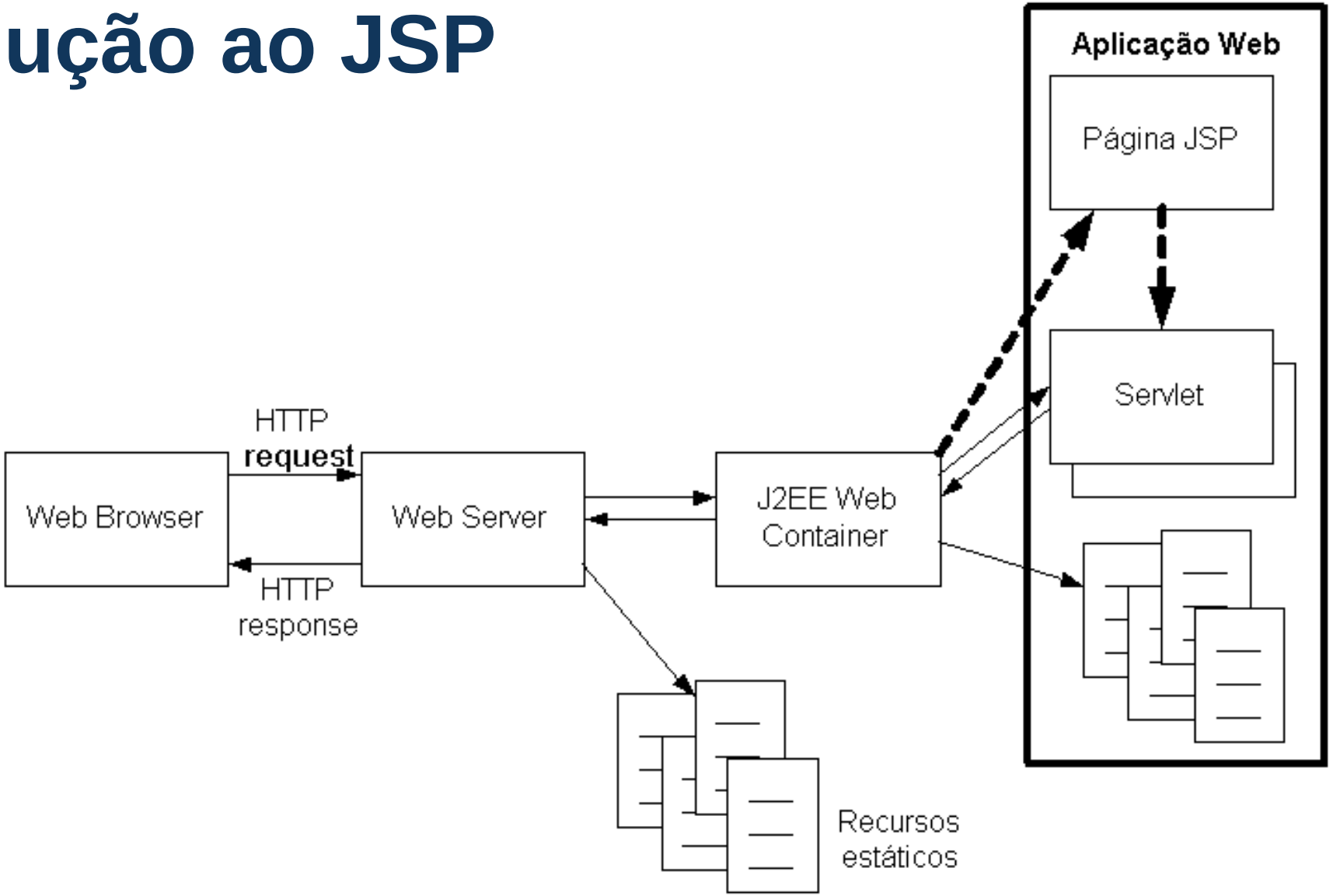
Comandos

```
# mvn tomcat7:deploy
# mvn tomcat7:undeploy
# mvn tomcat7:redeploy
```

Introdução ao JSP

- **Jakarta Server Pages (JSP)** é a tecnologia que a facilita a criação de conteúdo dinâmico para Web utilizando a linguagem Java;
- Separa a apresentação da lógica de negócio, funcionando como um mecanismo de template;
- Permite a separação da aplicação web em camadas, o que facilita a manutenção e evolução do código.
- O mesmo código Java pode ser utilizado com um front-end feito em Swing (desktop), por exemplo, e também em JSP (web).

Introdução ao JSP



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

HTML

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

JSP

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

HTML

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

JSP

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

Scriplets

Exibe valor na página

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" %>
<%@ page import="java.util.List" %>
```

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 classes *Bean*:
  - **<jsp:useBean>**
    - permite o JSP usar uma instância de um *Bean*
  - **<jsp:setProperty>**
    - define uma propriedade na instância do *Bean*
  - **<jsp:getProperty>**
    - obtém o valor de uma propriedade na instância do *Bean*

Objetos implícitos

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

Continua…

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.], 2023. Disponível em: https://jakarta.ee/specifications/pages/3.1/jakarta-server-pages-spec-3.1.html
- ORACLE; ECLIPSE FOUNDATION (ed.). **Jakarta Servlet Specification**. [S. l.], 2023. Disponível em: https://jakarta.ee/specifications/servlet/6.0/jakarta-servlet-spec-6.0.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.