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
- Apostila Java e Orientação a Objetos (Caelum/Alura)
  - https://www.alura.com.br/apostila-java-orientacao-objetos
- 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
- **Log Viewer (Extensão do VS Code):** https://marketplace.visualstudio.com/items?itemName=berublan.vscode-log-viewer
- **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 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/
- **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

- 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/v10.0.23/bin/apache-tomcat-10.0.23.exe
- Tutorial de instalação: https://github.com/webacademyufac/fundamentos-back-end/blob/main/tutoriais/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/fundamentos-back-end/blob/main/tutoriais/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

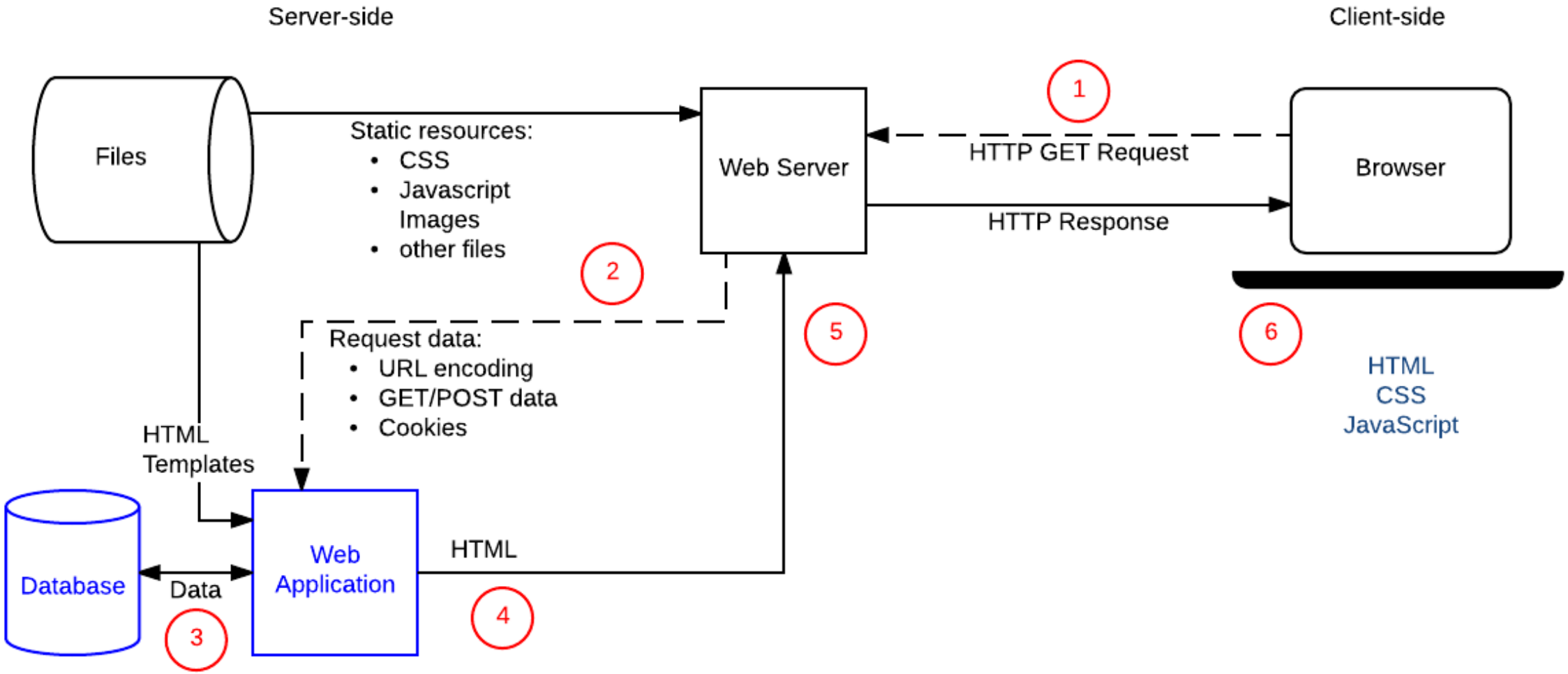
https://linkme.bio/danielnsilva/

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.

```java
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 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
```

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. }
```

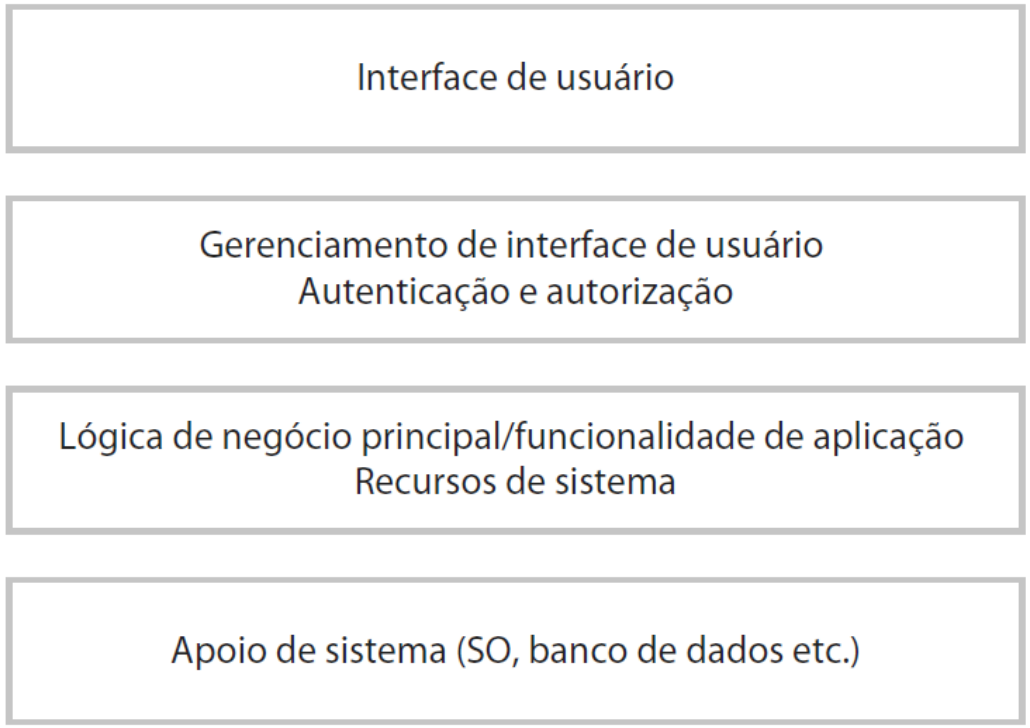
Programação orientada a objetos (POO)

- **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. }
```

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.



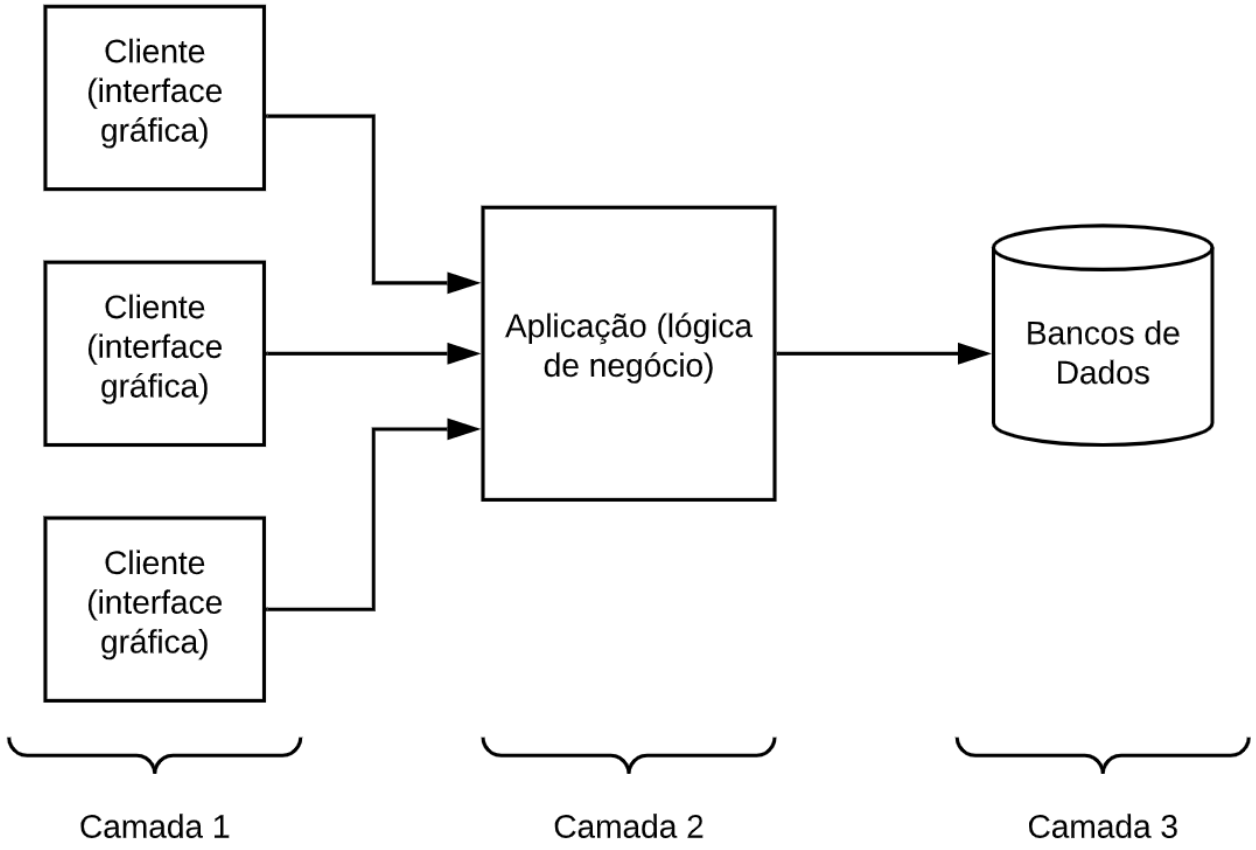
Fonte: SOMMERVILLE, 2011.

- **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**.

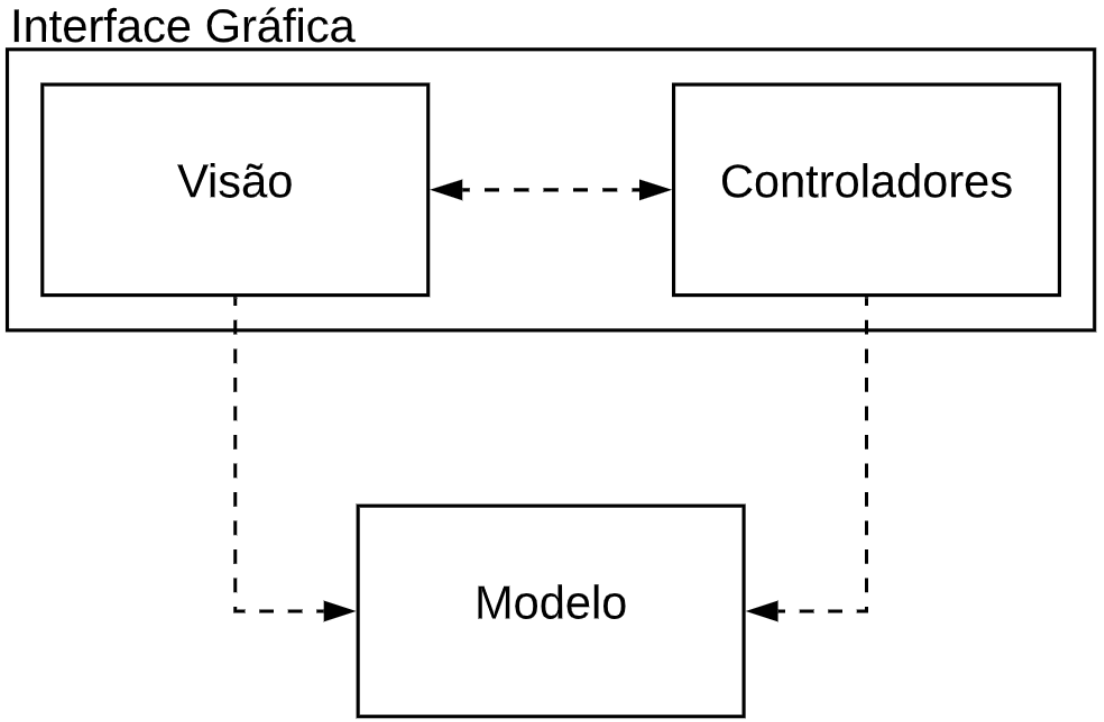
1. **Interface com o Usuário**, responsável por toda interação com o usuário;
2. **Lógica de Negócio**, que implementa as regras de negócio do sistema;
3. **Banco de Dados**, que armazena os dados manipulados pelo sistema.



Fonte: VALENTE, 2020.

Arquitetura MVC (*Model-View-Controller*)

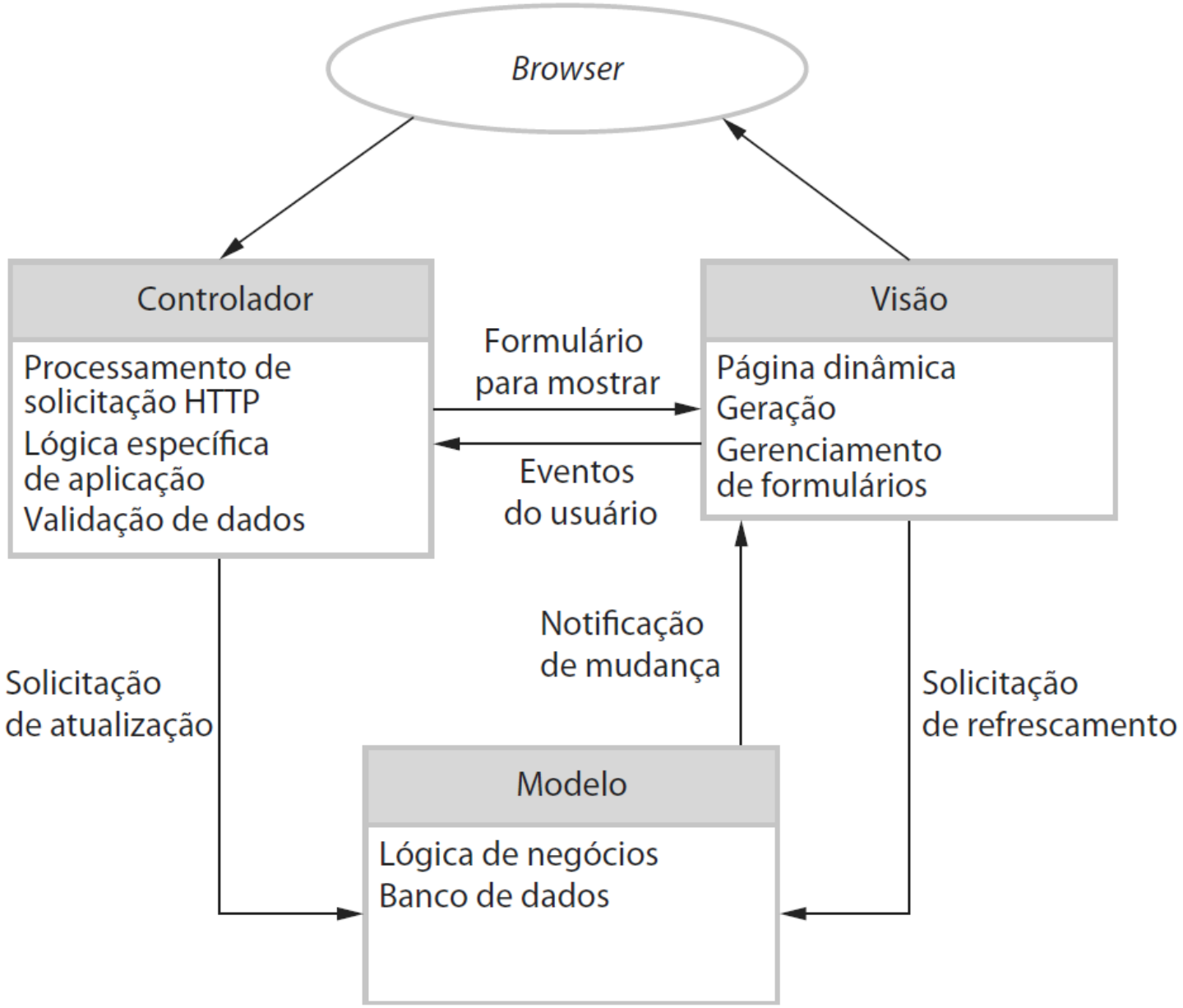
- **Visão**: responsável pela apresentação da interface gráfica do sistema, incluindo janelas, botões, menus, barras de rolagem, etc.
- **Controladores**: tratam e interpretam eventos gerados por dispositivos de entrada.
- **Modelo**: armazenam os dados manipulados pela aplicação, sem qualquer dependência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



Fonte: VALENTE, 2020.

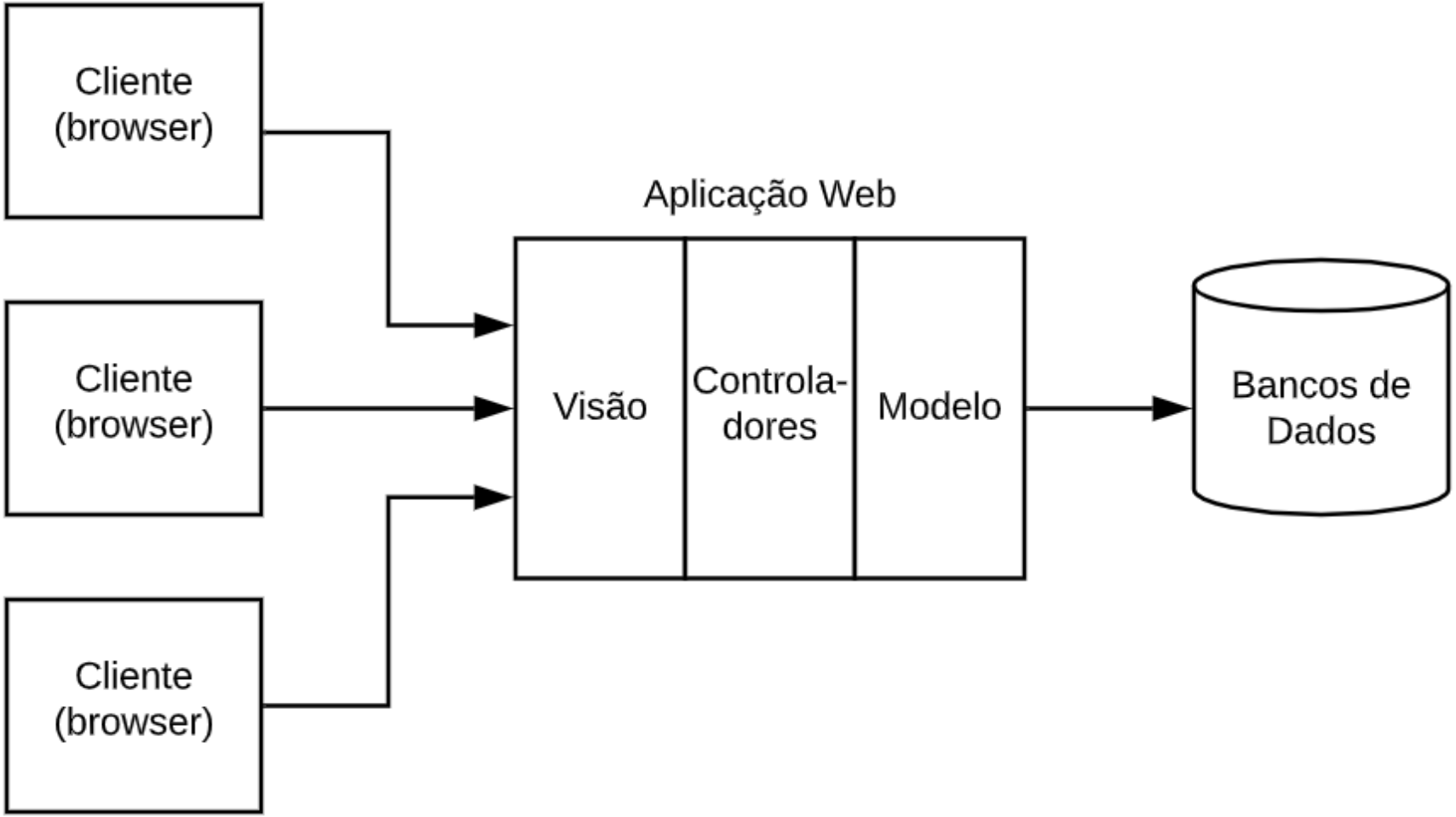
Arquitetura MVC

Exemplo de arquitetura de aplicações Web usando MVC



Fonte: SOMMERVILLE, 2011.

Qual a diferença entre MVC e três camadas?



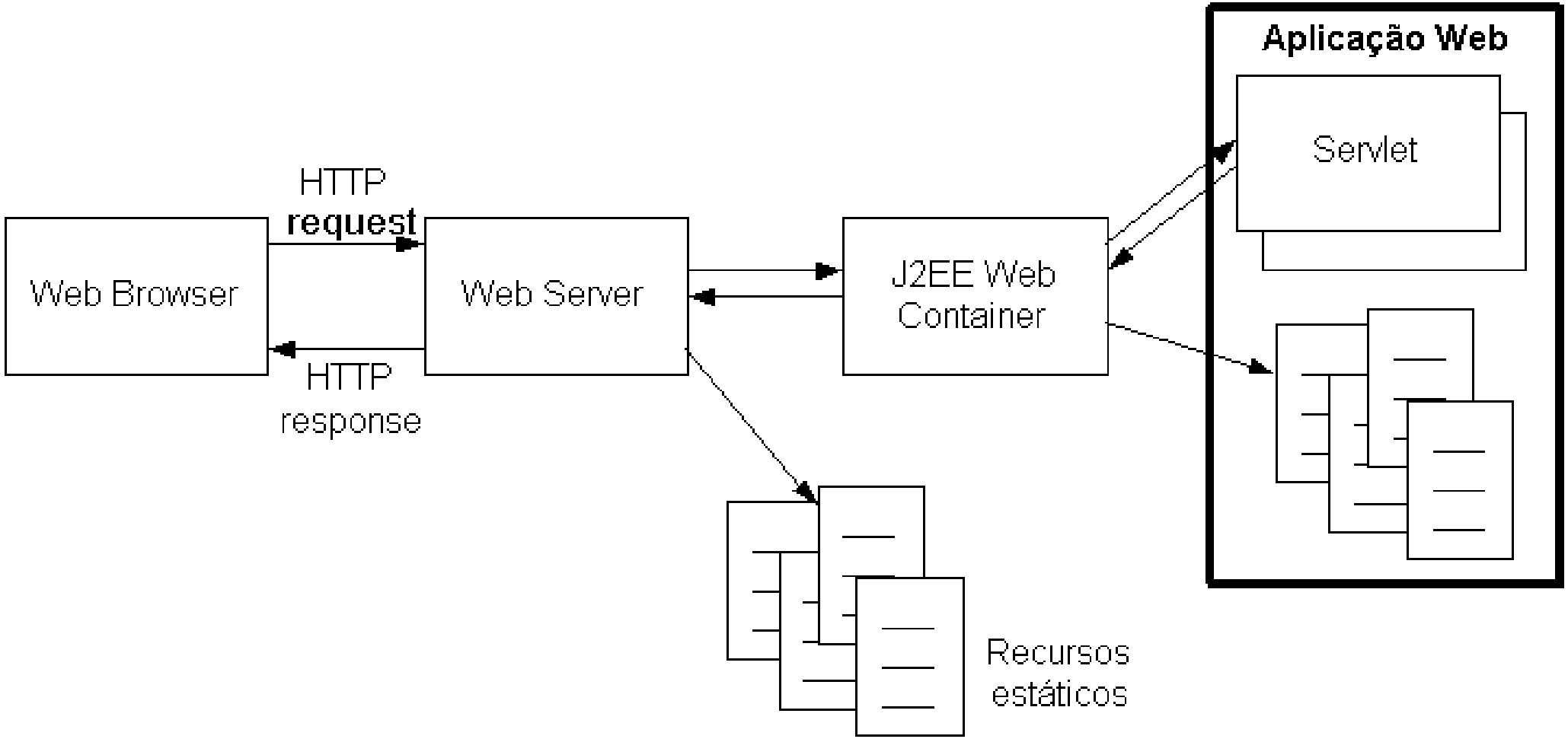
Fonte: VALENTE, 2020.

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 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.

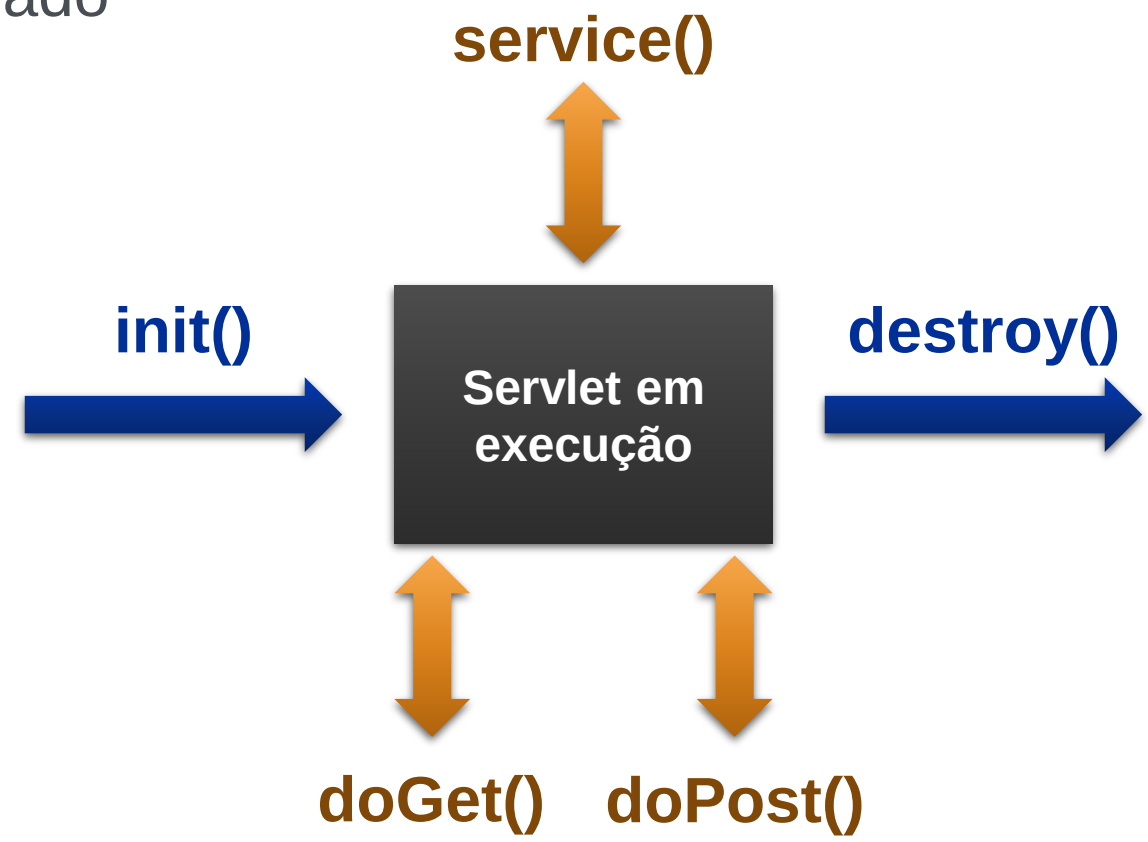
```
1.  <dependencies>
2.    <dependency>
3.      <groupId>jakarta.servlet</groupId>
4.      <artifactId>jakarta.servlet-api</artifactId>
5.      <version>5.0.0</version>
6.      <scope>provided</scope>
7.    </dependency>
8.  </dependencies>
```

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** 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_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:redeploy
```

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;
  }
}
```

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;**

JDBC

- O JDBC (**Java DataBase Connectivity**) 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 = 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

Configuração do projeto – pom.xml

```
1.    <dependencies>
2.      <dependency>
3.        <groupId>mysql</groupId>
4.        <artifactId>mysql-connector-java</artifactId>
5.        <version>8.0.28</version>
6.      </dependency>
7.    </dependencies>
```

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

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.