



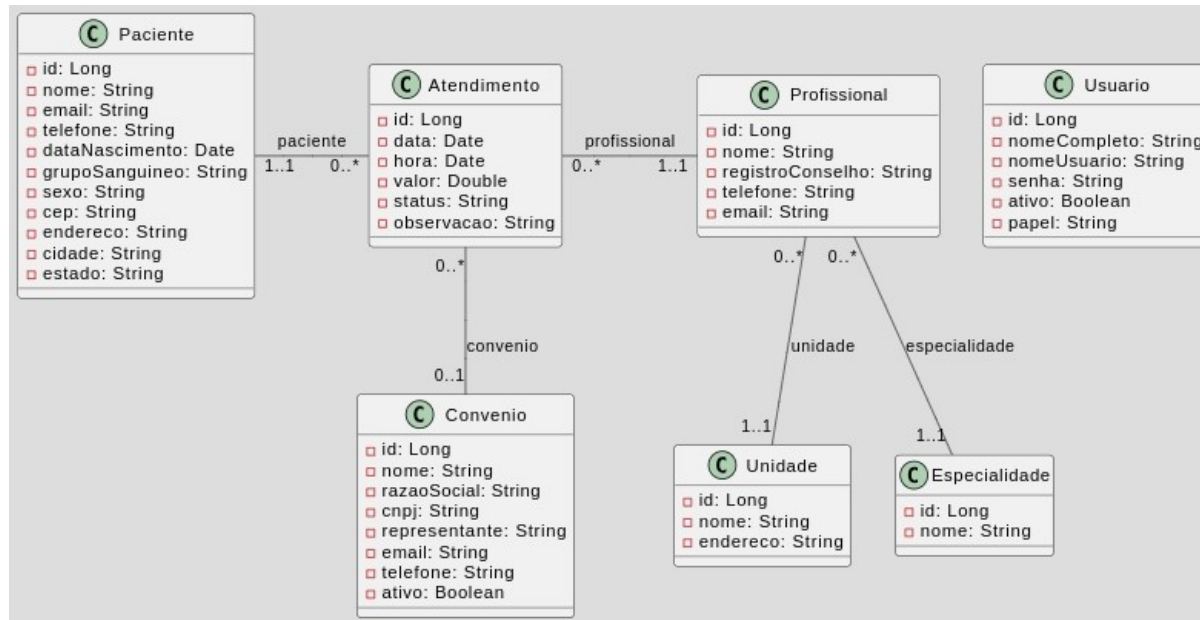
Web Academy



Apresentação

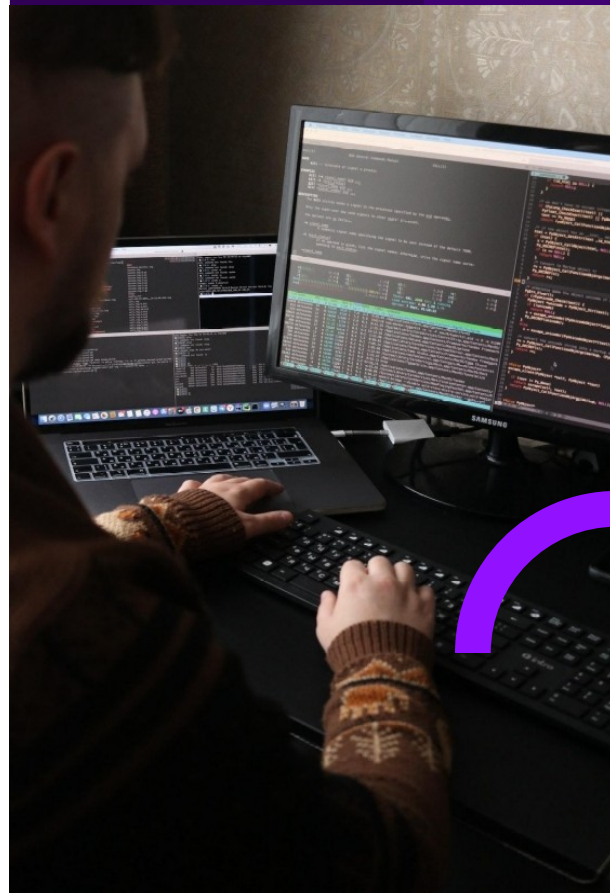
SGCM - Sistema de Gerenciamento de Consultas Médicas

- Documentação: <https://github.com/webacademyufac/s'gcmdocs>
 - Diagrama de classes



Ementa

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

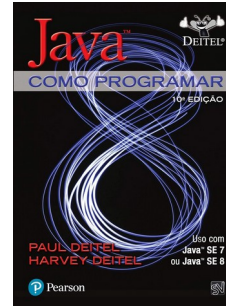
Servlets

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

Bibliografia



Web



Java: como programar
10ª Edição - 2016
Editora Pearson
ISBN 9788543004792



Engenharia de Software Moderna
Marco Tulio Valente
<https://engsoftmoderna.info/>

Academy

-

Sites de conteúdo

- **Java e Orientação a Objetos (Caelum/Alura)**
 - <https://www.alura.com.br/apostila-java-orientacao-objetos>
- **Java para Desenvolvimento Web (Alura)**
 - <https://www.alura.com.br/apostila-java-web>
- **Java Tutorial (VS Code)**
 - <https://code.visualstudio.com/docs/java/java-tutorial>
- **Baeldung**
 - <https://www.baeldung.com/>

Ferramentas

- **Visual Studio Code**
 - <https://code.visualstudio.com/Download>
- **Extension Pack for Java (Extensão do VS Code)**
 - <https://marketplace.visualstudio.com/items?vscjava.vscode-java-pack>
- **Java Server Pages - JSP (Extensão do VS Code)**
 - <https://marketplace.visualstudio.com/items?pthorsson.vscode-jsp>
- **XML (Extensão do VS Code)**
 - <https://marketplace.visualstudio.com/items?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 configuração:
 - <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/v10.1.30/bin/apache-tomcat-10.1.30.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 reiniciar a senha de root:
<https://dev.mysql.com/doc/mysql-windows-excerpt/8.0/en/resetting-permissions-window-s.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**



Web Academy

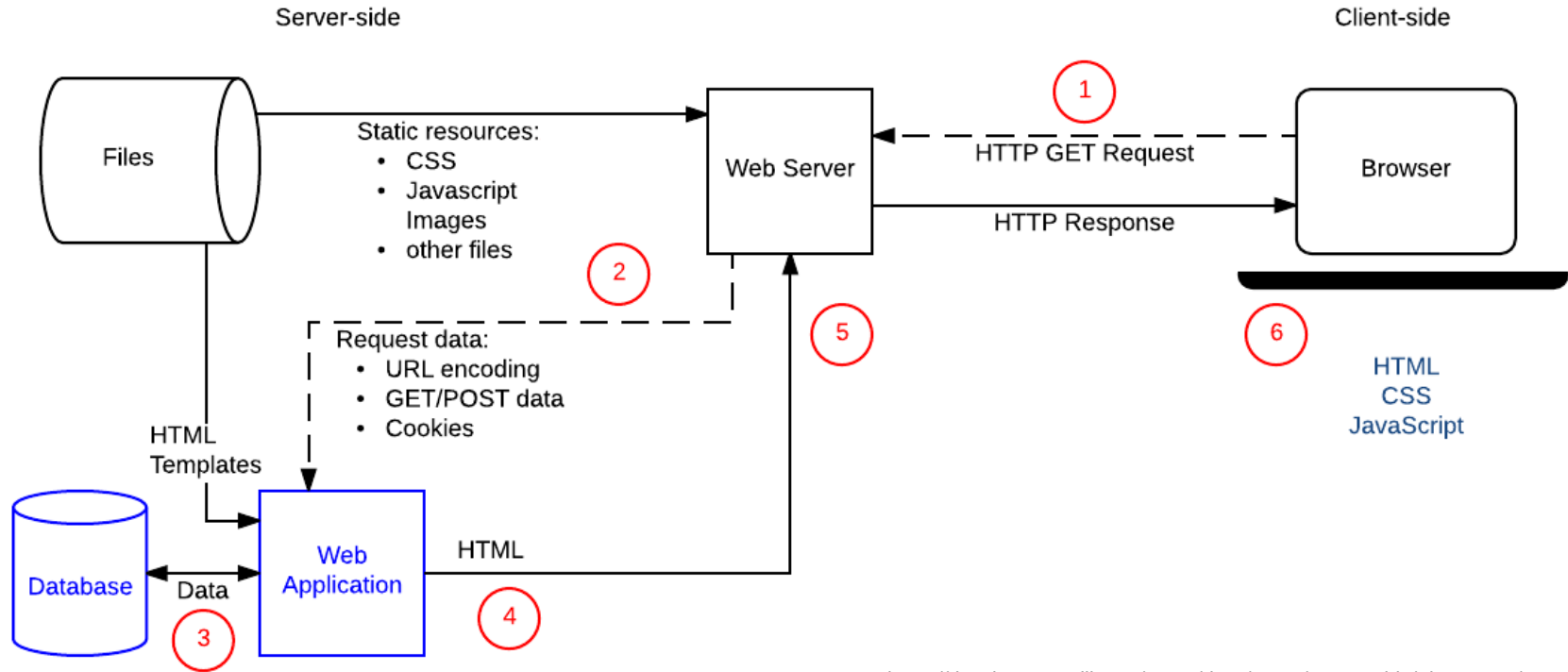


Introdução

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: História

- Interessada em dispositivos eletrônicos inteligentes, a **Sun Microsystems** financiou em 1991 o projeto Green.
- Linguagem baseada em C e C++, criada por **James Gosling**, inicialmente chamada de Oak (carvalho).
- Em **1995**, no evento conhecido como SunWorl'95, a Sun apresentou o navegador HotJava e a linguagem Java. No ano seguinte, a NetScape Corp lançou a versão 2 do seu navegador (Navigator), que incorporou a funcionalidade de executar aplicações Java conhecidas como **applets**.
- Em 1996, a Sun liberou de forma gratuita para a comunidade um conjunto de ferramentas para desenvolvimento usando Java (**JDK**).



Java: Plataformas

- A Sun continuou detentora dos direitos até 2009, quando a empresa foi comprada pela Oracle (US\$ 7,4 bilhões) que continuou com a evolução da linguagem e da plataforma.
- A aquisição da Sun não gerou impacto para os desenvolvedores Java, pois a linguagem continua gratuita.
- Java **Standard Edition** ou **JavaSE**
 - Ambiente para o desenvolvimento de aplicações de pequeno e médio porte, além de um conjunto de APIs (Swing) e a JVM padrão.
- Java **Enterprise Edition** ou **JavaEE**
 - Componente baseado no desenvolvimento de aplicações empresariais multicamadas de grande porte e provê serviços adicionais, ferramentas e APIs (JPA, JSP) para simplificar a criação de aplicações complexas.





Java: ambiente de desenvolvimento

- Java entrega um ambiente para o desenvolvimento de programas composto por:
 - Uma linguagem de programação de alto nível orientada a objetos;
 - Máquina Virtual (**Java Virtual Machine** ou **JVM**), que garante independência de plataforma, pois o código executa na máquina virtual e essa pode ser portada para outras plataformas como Windows ou Linux;
 - Java Runtime Environment ou **JRE**, que agrega a máquina virtual e alguns recursos para a execução de aplicações Java; e
 - Java Development Kit ou **JDK**, que é um conjunto de utilitários que oferece suporte ao desenvolvimento de aplicações.

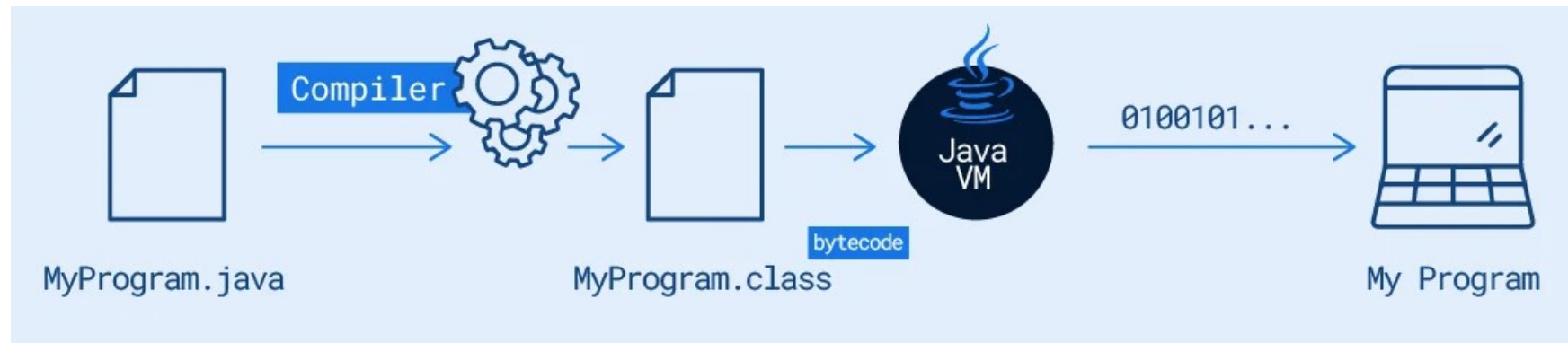


Java: características

- Independência de plataforma (portabilidade)
- Orientação a Objetos
- Não usa ponteiros
- Multithread
- Segurança
- Recursos de rede
- Gerência automática de memória (*Garbage Collection*)
- Sintaxe similar a C/C++
- Método híbrido de implementação (Compilação + Interpretação)

Java: programas

- Em Java, os programas são escritos em um arquivo com a extensão **.java**, que em um processo posterior serão compilados para arquivos com a extensão **.class**. Esses, por sua vez, contêm os códigos a serem executados na máquina virtual, os **bytecodes**.



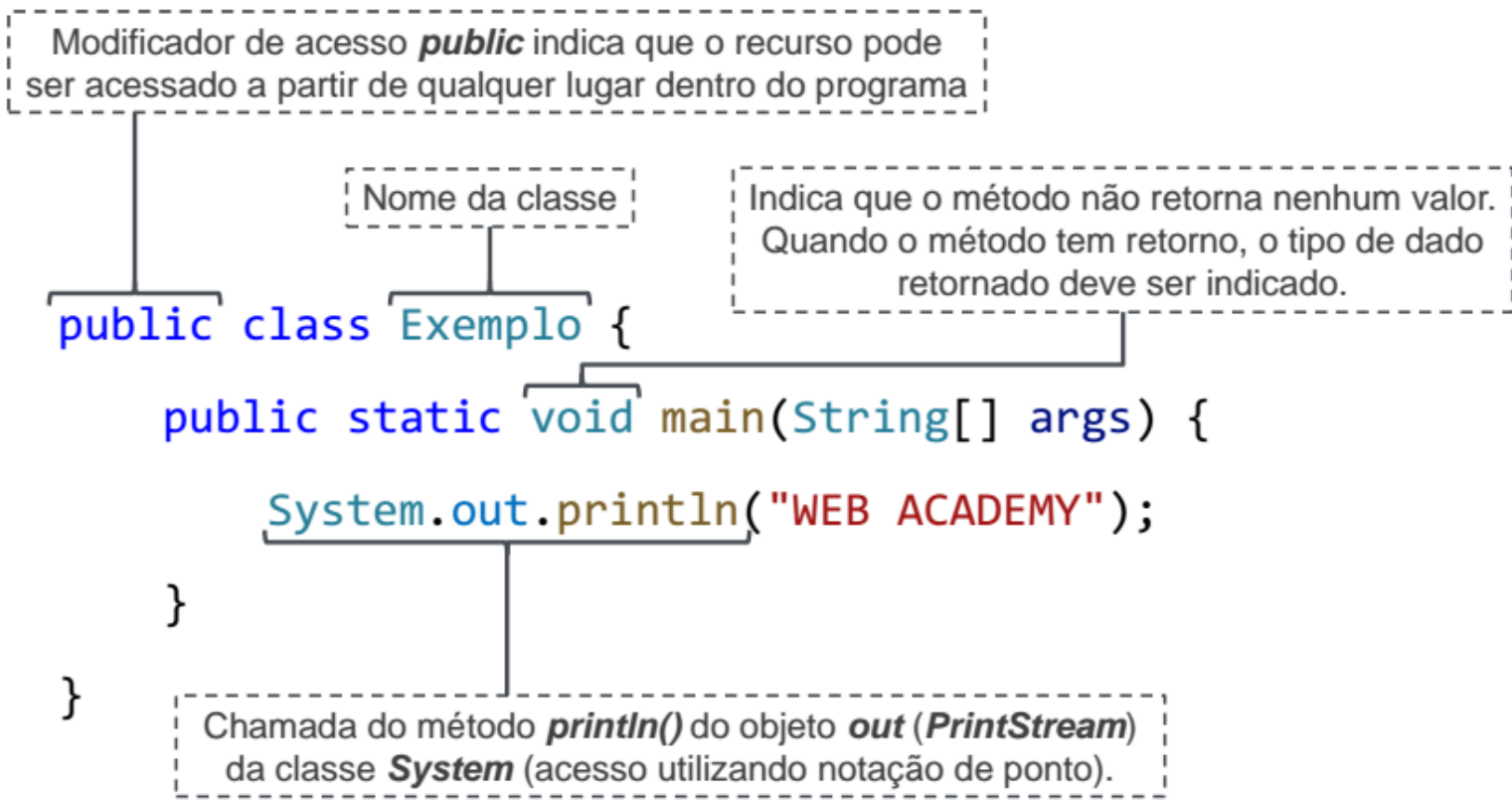
Java: exemplo

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



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: tipos de dados

- Java é uma linguagem de tipagem **forte e estática**, portanto, não permite operações diretas entre tipos diferentes e requer que todas as variáveis tenham um tipo.
- Tipos primitivos: boolean, char, byte, short, int, long, float, double.
- O tipo cadeia de caracteres (string) é um objeto da classe String.

```
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: casting

PARA:	byte	short	char	int	long	float	double
DE:							
byte	----	<i>Impl.</i>	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
short	(byte)	----	(char)	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
char	(byte)	(short)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
int	(byte)	(short)	(char)	----	<i>Impl.</i>	<i>Impl.</i>	<i>Impl.</i>
long	(byte)	(short)	(char)	(int)	----	<i>Impl.</i>	<i>Impl.</i>
float	(byte)	(short)	(char)	(int)	(long)	----	<i>Impl.</i>
double	(byte)	(short)	(char)	(int)	(long)	(float)	----

Java: estruturas de controle

```
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);
```

Diagram illustrating the ternary operator logic:

Condição	true	false
(numero > 3)	"Maior que 3"	"Menor ou igual a 3"

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.

```
// 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];

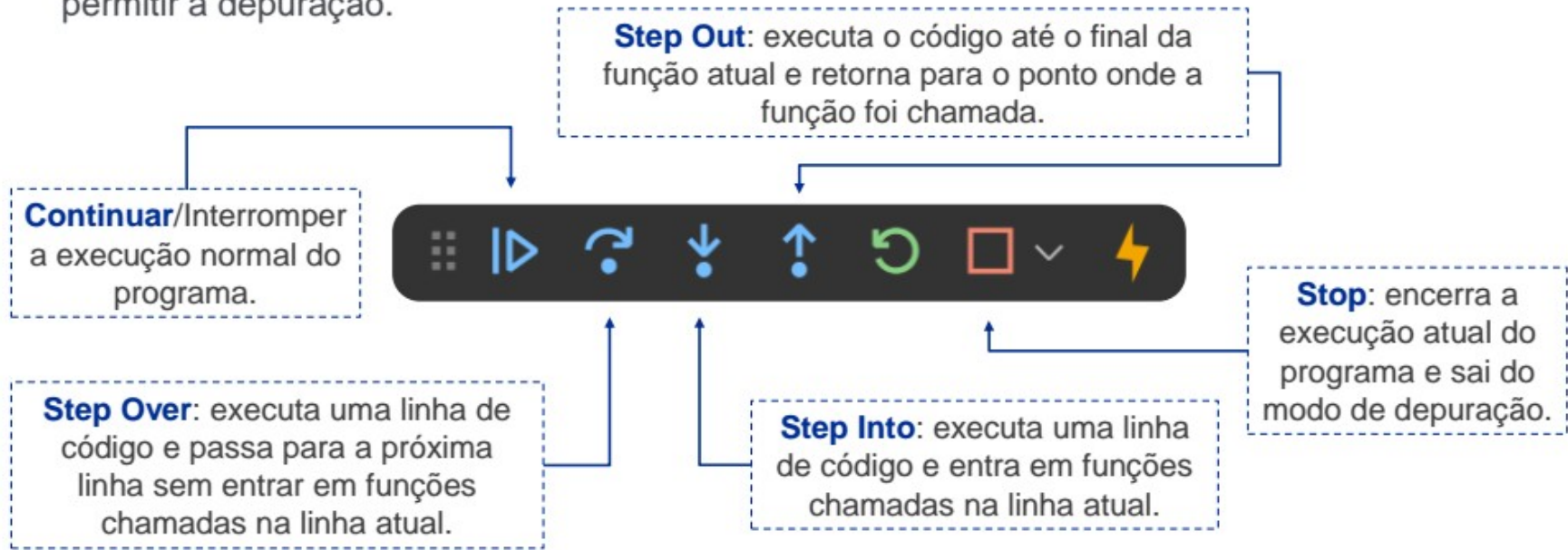
// Acessando um elemento do array dinâmico
int numero = numeros.get(1);
```

```
// 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.





Depuração de apps Java no VS Code

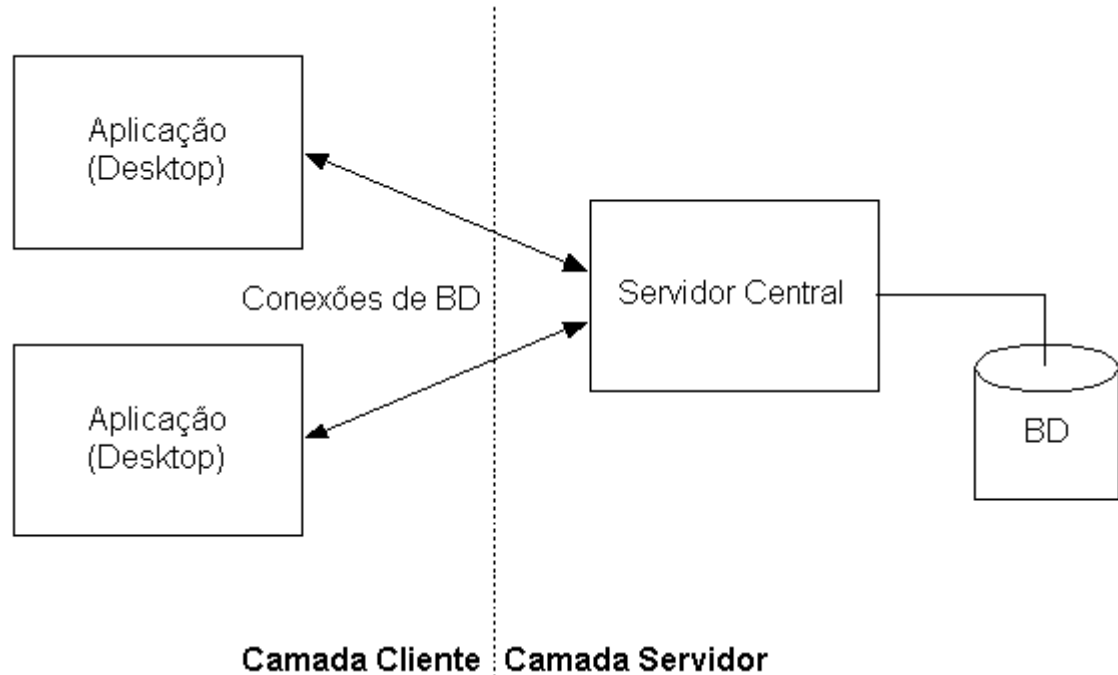
- Referências úteis:
 - <https://code.visualstudio.com/docs/editor/debugging>
 - <https://code.visualstudio.com/docs/java/java-debugging>

Arquitetura Cliente-Servidor

- É uma estrutura de aplicação (**arquitetura**) que distribui as tarefas e cargas de trabalho entre **fornecedores** de um recurso ou serviço (**servidores**) e **requerentes** (**clientes**).
- O **cliente** é uma aplicação que o **usuário final** interage e o **servidor** é uma aplicação que **processa as solicitações e retorna os dados ou serviços solicitados**, através de uma rede em computadores distintos, mas tanto o cliente quanto o servidor podem residir no mesmo computador.
- O **servidor compartilha recursos** com os clientes. Um **cliente não compartilha** qualquer de seus recursos. Os clientes iniciam sessões de comunicação com os servidores que aguardam requisições de entrada.

Arquitetura Cliente-Servidor (2 camadas)

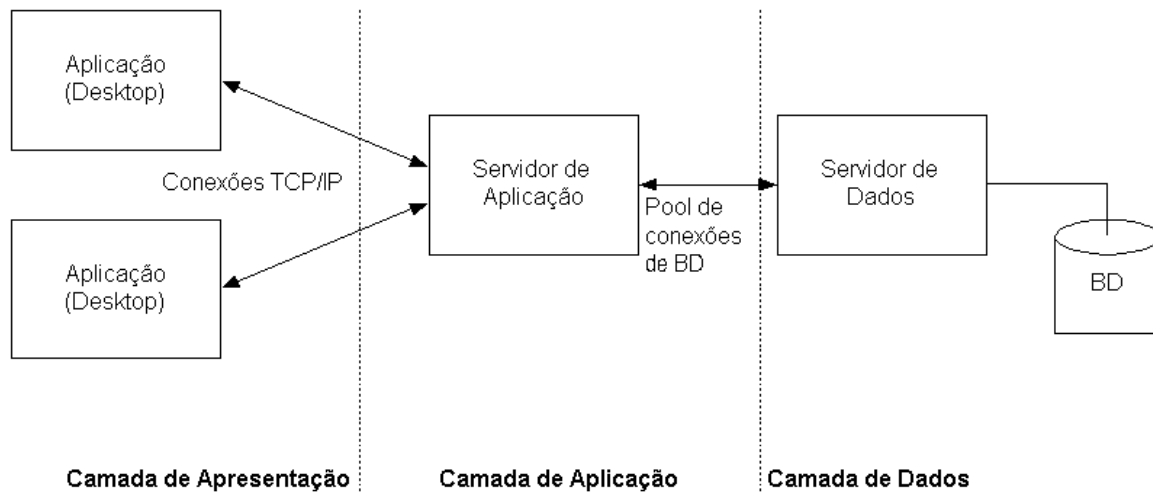
- Melhor aproveitamento dos Desktops
- Oferecer sistemas com **interfaces gráficas amigáveis**
- Integrar o desktop e os dados corporativos
- Aumentar a escalabilidade de uso de Sistemas de Informação
- Camada **cliente** trata da **lógica de negócio e da UI (User Interface)**
- Camada **servidor** trata dos dados (usando um SGBDs)



Fonte: <http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/intro/intro.htm>

Arquitetura Cliente-Servidor (3 camadas)

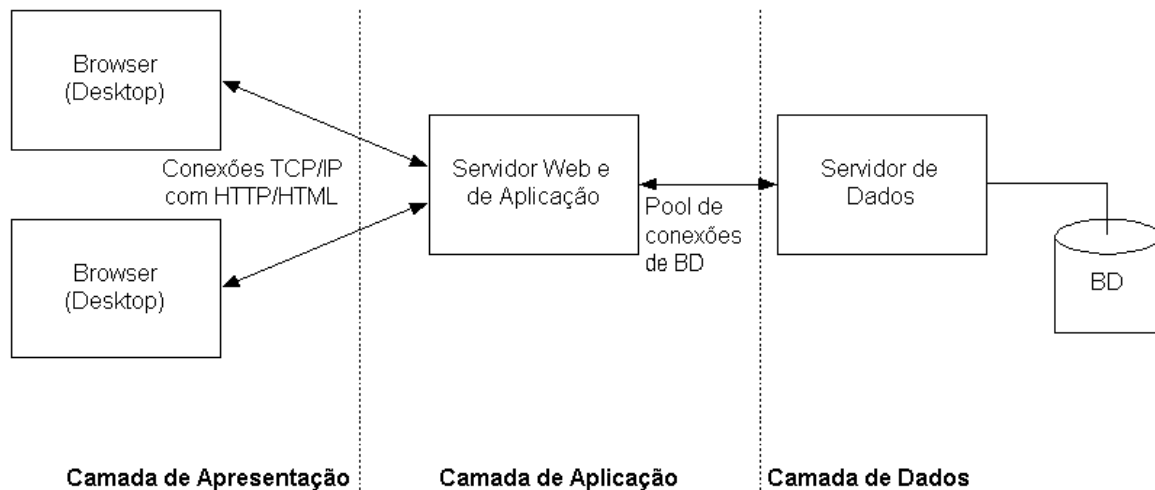
- Problemas de manutenção foram reduzidos, pois mudanças nas camadas de aplicação e de dados não necessitam de novas instalações no desktop
- Observe que as **camadas são lógicas**
- Fisicamente, **várias camadas podem executar na mesma máquina**
- Quase sempre, há separação física de máquinas



Fonte: <http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/intro/intro.htm>

Arquitetura Cliente-Servidor Web (3 camadas)

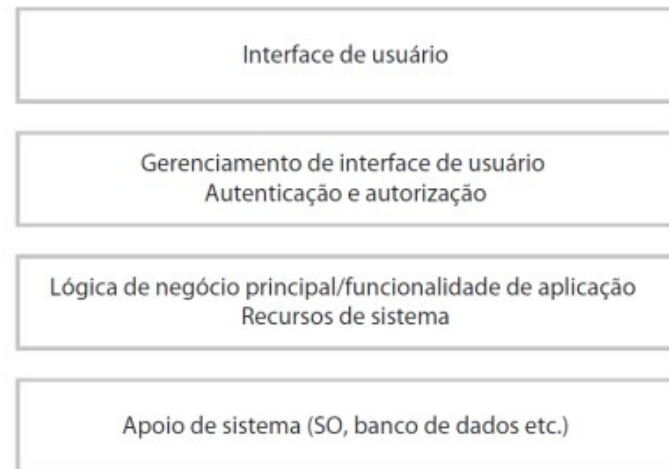
- Conceito de Intranet
- A camada de **aplicação se quebra em duas: Web e Aplicação**
- Evitamos instalar qualquer software no desktop e portanto, problemas de manutenção
- Evitar instalação em computadores de clientes, parceiros, fornecedores, etc.



Fonte: <http://www.dsc.ufcg.edu.br/~jacques/cursos/j2ee/html/intro/intro.htm>

Modelo em camadas

- Modelo 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.

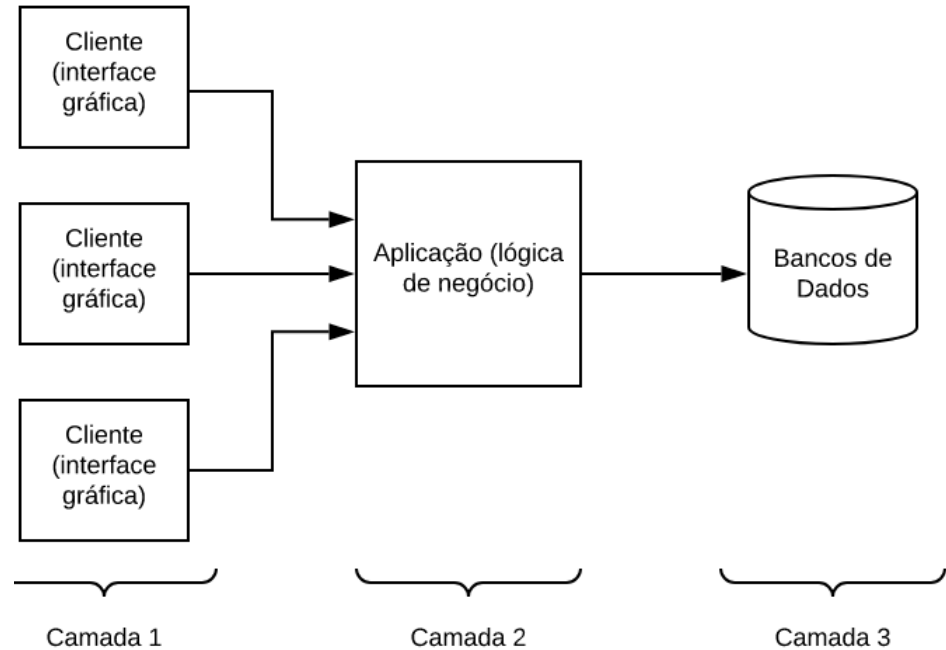


Fonte: SOMMERVILLE, 2011.

Modelo em três camadas

- Comum na construção de **sistemas de informação corporativos**.

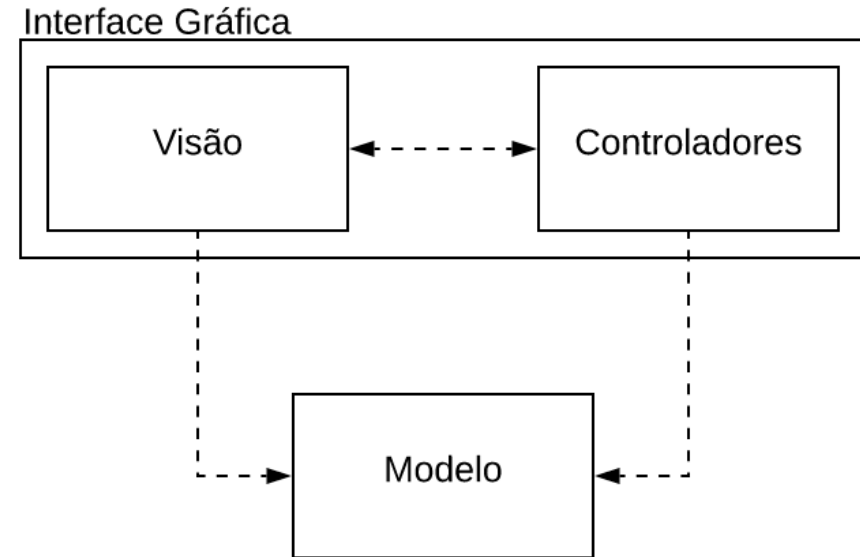
- Interface com o Usuário**, responsável por toda interação com o usuário;
- Lógica de Negócio**, que implementa as regras de negócio do sistema;
- Banco de Dados**, armazena os dados manipulados pelo sistema.



Fonte: (VALENTE, 2020)

Arquitetura MVC (Model-View-Controller)

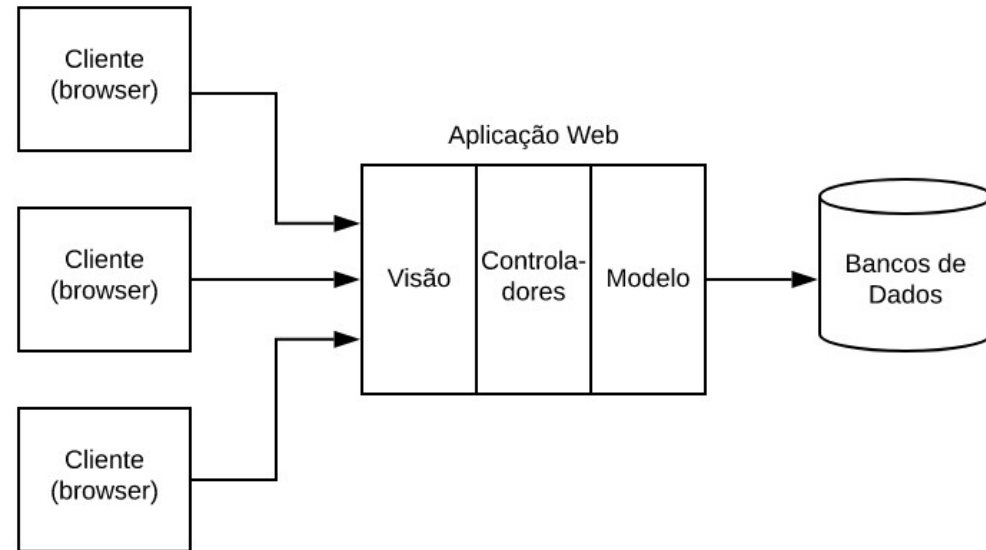
- **Visão:** responsável pela apresentação da interface gráfica do sistema, incluindo janelas, botões, menus, barras de rolagem, etc.
- **Controladores:** tratam e interpretam eventos gerados por dispositivos de entrada.
- **Modelo:** armazenam os dados manipulados pela aplicação, sem qualquer dependência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



Fonte: (VALENTE, 2020)

Diferença entre MVC e três camadas

- MVC surgiu no final da década de 70, para ajudar na construção de interfaces gráficas, pode ser usado na implementação da camada de interface.
- O modelo em três camadas surgiu na década de 90. A camada de interface executa na máquina dos clientes, a de negócio em um servidor de aplicação. E, por fim, temos o banco de dados.
- No início dos anos 2000, os sistemas Web se popularizaram e o termo MVC começou a ser usado por *frameworks*: **visão**, composta por páginas HTML; **controladores**, que processam uma solicitação e geram uma nova visão como resposta e **modelo**, que é a camada que persiste os dados em um banco de dados.



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"**.

```
package br.ufac.sgcm;  
  
public class Exemplo {  
    // corpo da classe  
}
```

```
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



Web Academy



Java e Programação Orientada a Objetos



Programação Orientada a Objetos (POO)

- É um método de programar que ajuda na organização e resolve muitos problemas enfrentados pela programação procedural.
- Usa **tipos de dados personalizados**.
- Em vez de programar apenas com tipos de dados primitivos, podemos construir **novos tipos de dados**.
- Baseia-se fundamentalmente no conceito de **classes e objetos**.
- Os objetos que se comunicam por **troca de mensagens** enviadas e recebidas pelos métodos.



POO - Vantagens

- Fornece estrutura modular para a construção de programas.
- O software se torna mais fácil de manter.
- **Reutilização** de código
 - Desenvolver mais rápido
 - Objetos podem ser reutilizados em aplicação diferentes
- **Encapsulamento**
 - Não é necessário conhecer a implementação interna de um objeto para poder usá-lo



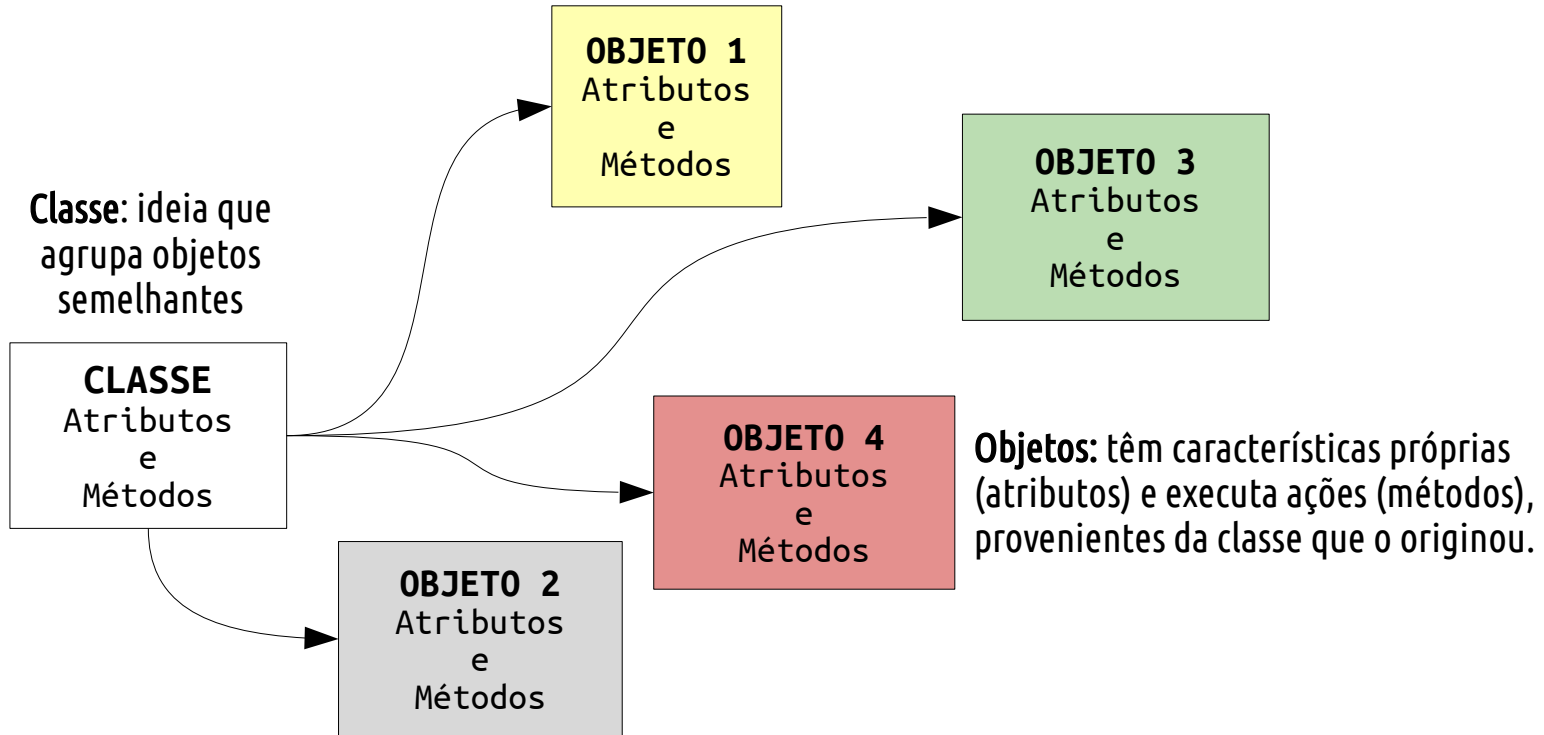
POO - Abstração

- Abstrair em orientação a objetos é selecionar objetos que queremos representar a partir do contexto em que se situam e representar somente as **características que são relevantes** para o problema em questão.
- Tais abstrações se comunicam entre si, por meio de troca de **mensagens**.

POO - Mensagens

- Mensagem é um **sinal de um objeto a outro**, requisitando um serviço, usando uma operação programada no objeto chamado.
- Quando uma mensagem é recebida, uma operação é invocada no objeto chamado.
- Podem ser resultados de fórmulas matemáticas, acionamento de eventos, regra de negócio, etc.

PОО - Classes e Objetos



POO - Classes e Objetos

- **Classe:**
 - Estrutura que abstrai um conjunto de objetos com **características e comportamentos semelhantes**.
 - **POJO (Plain Old Java Object)**: define uma classe simples, sem recursos especiais.
 - **Tipo personalizado de dados**, ou seja, molde para a criação de objetos.
- **Objeto:**
 - Instância ou **modelo derivado de uma classe**, que pode ser manipulado pelo programa.
 - Representam **entidades do mundo real**, como: carros, contas, pessoas, recursos computacionais, etc.

```
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

- A ideia de encapsular é, **'proteger' de forma organizada** todos os membros de uma classe: os atributos e os métodos (*getters* e *setters*) do sistema.
- 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 - Encapsulamento

- Encapsular é **fundamental para mudanças**: não precisamos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada.
- O conjunto de métodos públicos de uma classe é também chamado de **interface da classe**, pois esta é a única maneira a qual você se comunica com seus objetos.
- As mesmas regras de acesso aos atributos valem para os métodos. É comum, e faz sentido, que **os atributos sejam privados e quase todos seus métodos sejam públicos**. Desta forma, toda conversa de um objeto com outro é feita por troca de mensagens, isto é, acessando seus métodos.

POO - Herança

- Mecanismo que permite criar novas classes, **aproveitando as características da classe.**
- Promove **reaproveitamento** do código existente.
- Java não permite herança múltipla apenas **herança simples.**

```
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

- É capacidade de um objeto poder ser referenciado de várias formas.
- Permite que os programas processem objetos que compartilham a mesma superclasse **como se todos fossem objetos da superclasse**.
- Uma forma de implementar polimorfismo é através de **classes abstratas**, que não podem ser instanciadas, 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 - Classes Abstratas

- Uma **Classe Abstrata** é considerada um **projeto** para outras classes, ou seja, é um tipo especial de classe que **não pode ser instanciada**.
- Permite **especificar um conjunto de métodos** que devem ser implementados em qualquer classe filha construída a partir da classe abstrata.
- Uma classe abstrata deve conter **um ou mais métodos abstratos**.
- Um **método abstrato** é um método que possui uma declaração, mas **não possui uma implementação**.

POO - Classes Abstratas

- Caso os **métodos abstratos** não sejam implementados nas **classes filhas**, um erro será lançado durante a execução do programa
- Para implementar um método abstrato em uma classe filha, basta definir o método **com o mesmo nome e assinatura na classe filha**. A implementação do método deve seguir a lógica específica da classe filha
- Além dos métodos abstratos, **uma classe abstrata também pode ter métodos concretos**, ou seja, métodos que já possuem uma implementação padrão. Esses métodos podem ser sobrescritos nas classes filhas, se necessário.

POO - Polimorfismo

- Em Java, outra forma de implementar o polimorfismo é por meio de **interfaces**.
- Uma interface define as **operações que uma classe será obrigada a implementar**.
- Uma classe pode implementar **várias interfaces**.

```
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;  
    }  
}
```


PОО - 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;  
    }  
}
```



Web Academy



JDBC (Java DataBase Connectivity)



Maven

- O Apache Maven é uma ferramenta de **gerenciamento de projetos Java**.
- O Maven organiza todas as informações do projeto em um único arquivo: o **pom.xml**.
- O Maven realiza o **build** do projeto, ou seja, conforme dependências são requisitadas, o POM é atualizado. O projeto Maven pode possuir módulos e cada módulo pode ter seu respectivo POM sem perder a organização e hierarquia do projeto principal.

Maven

- O Maven conta com um recurso chamado **archetype**, que permite criar toda a estrutura de um projeto automaticamente. Existem vários tipos de archetypes disponíveis, entre eles:
 - **maven-archetype-webapp**: cria a estrutura de uma aplicação web básica;
- Para criar o projeto precisamos informar:
 - **groupId**: O nome da organização ao qual pertence esse projeto, possui o mesmo padrão de nomenclatura de pacotes (br.ufac.sgcm)
 - **artifactId**: O nome do projeto (sgcm)
 - **version**: A versão do nosso projeto, caso não seja colocado nenhum valor o Maven irá utilizar o valor padrão que é 1.0-SNAPSHOT (1.0)

Maven - pom.xml

- O arquivo pom.xml contém todas as configurações que o Maven necessita para interagir corretamente com o projeto.
- Conteúdo do arquivo pom.xml
 - Coordenadas do projeto, ou seja, os dados que identificam o projeto, como groupId, artifactId e version.
 - Propriedades do projeto, ou seja, informações de *encondig* e também a versão do Java.
 - Dependências de nosso projeto.
 - Informações de *build* que dizem como o projeto deve ser compilado pelo Maven.

Maven - configuração de bibliotecas

```
<dependencies>
```

```
  <dependency>
```

```
    <groupId>mysql</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
```

```
    <version>8.0.32</version>
```

```
  </dependency>
```

```
</dependencies>
```

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*.
 - Construtor sem argumentos.
 - Implementa a interface *Serializable*.

```
// Classe (Java Bean)
public class Pessoa implements Serializable {
    private String nome; // Atributo privado
    public Pessoa() {} // Construtor
    public String getName() { // Getter
        return nome;
    }
    public void setName(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 mecanismos 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.

JDBC - Usando a API

- 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.

JDBC - Exemplos de URLs

- MySQL
 - `com.mysql.cj.jdbc.Driver` (ClassName)
 - `jdbc:mysql://nomeDoHost/nomeDoBanco`
- Oracle
 - `oracle.jdbc.driver.OracleDriver` (ClassName)
 - `jdbc:oracle:thin:@nomeDoHost:numeroDaPorta:nomeDoBanco`

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.

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

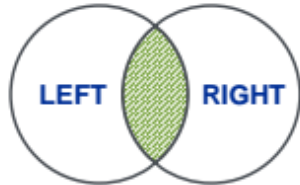
JDBC - SQL para operações CRUD

- 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 - 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

JDBC - Execução de instruções SQL



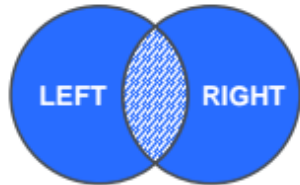
INNER JOIN retorna apenas as linhas que têm correspondência em ambas as tabelas (ou seja, onde a condição de junção é verdadeira).



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.