Web Academy

Fundamentos de Programação Back-end

Ufac

Manoel Limeira de Lima Júnior

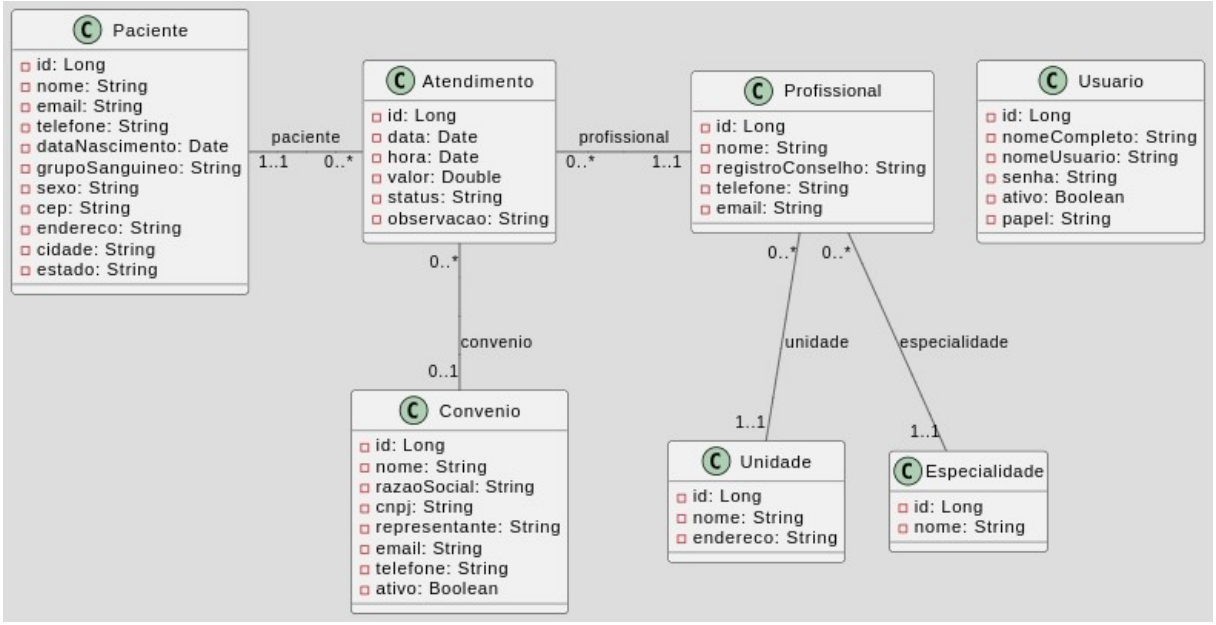
Web Academy

Ufac

Apresentação

SGCM – Sistema de Gerenciamento de Consultas Médicas

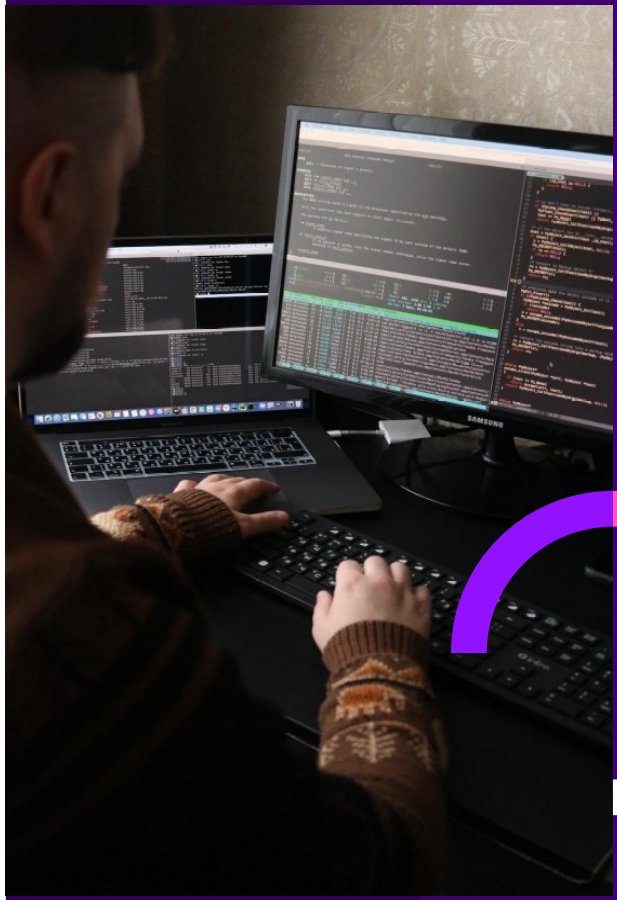
- Documentação: https://github.com/webacademyufac/sgcmdocs
  - Diagrama de classes

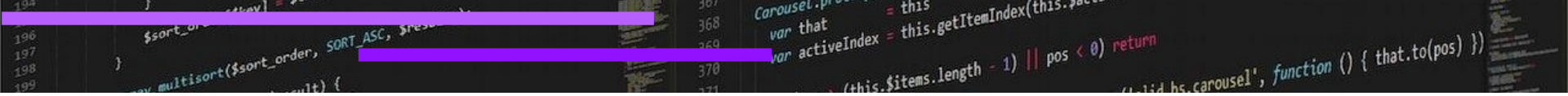


Web Academy

Ementa

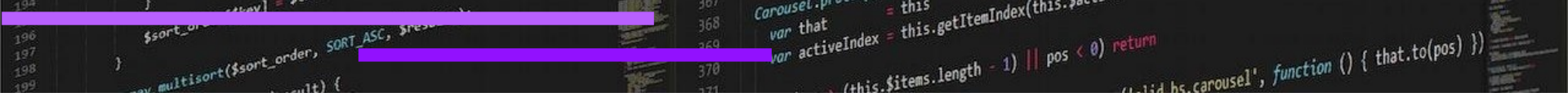
1. Linguagens de programação **server-side**.
2. Arquitetura em **camadas**.
3. Java, Servlets e Jakarta Server Pages (**JSP**)
4. Acesso à bases de dados com **JDBC** (Java Database Connectivity).
5. Implementação de operações **CRUD** (Create, Read, Update, Delete)
6. Segurança.





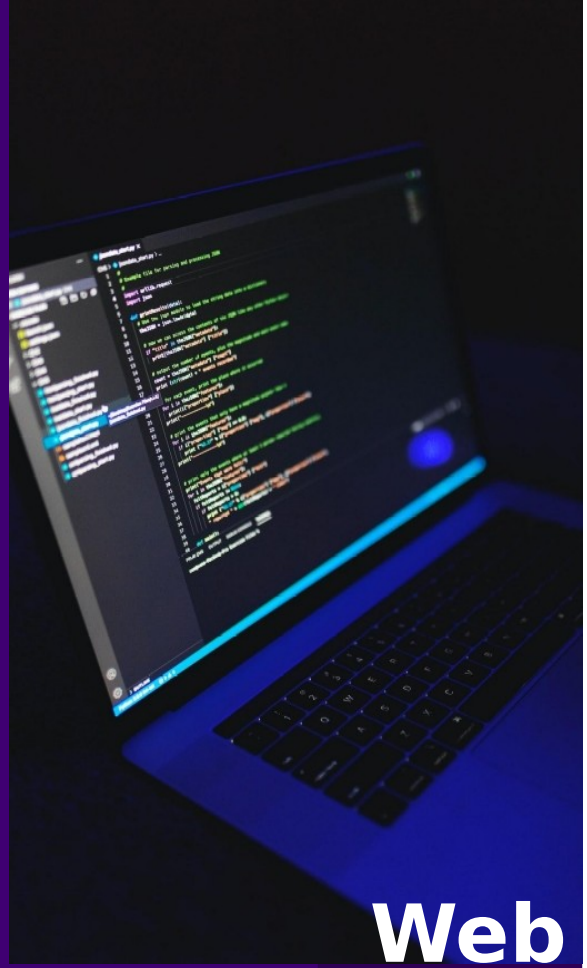
Objetivos

- **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.

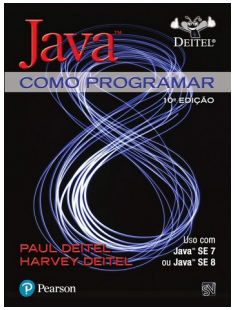


Conteúdo programático

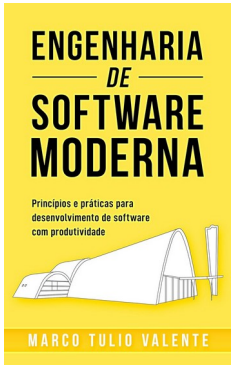
Introdução	Java e POO	JDBC	Servlets
Programação server-side; Java: sintaxe, modificadores de acesso, estruturas de controle, tipos básicos e arrays; Depuração de apps Java no VS Code; Arquitetura em camadas, MVC e pacotes Java;	Programação orientada a objetos (POO): classes e objetos; Encapsulamento, herança e polimorfismo; Sobrescrita e sobrecarga de métodos;	Java Beans; API do JDBC; Sintaxe das principais instruções SQL usadas em operações CRUD; Execução de instruções SQL (Statements e Result Sets); SQL Joins.	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



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



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

Academy

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

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?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-windows10/

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/v10.1.7/bin/apache-tomcat-10.1.7.exe
- Tutorial de instalação:
  - https://github.com/webacademyufac/tutoriais/blob/main/tomcat/tomcat.md

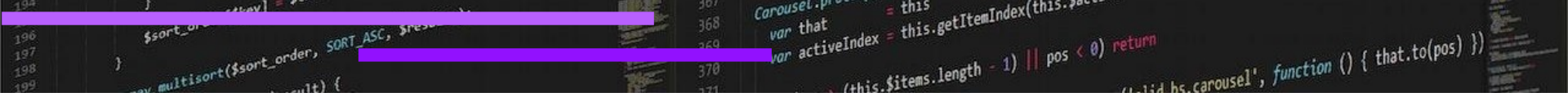
Ferramentas: MySQL

- 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 reiniciar 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**

Web Academy

Ufac

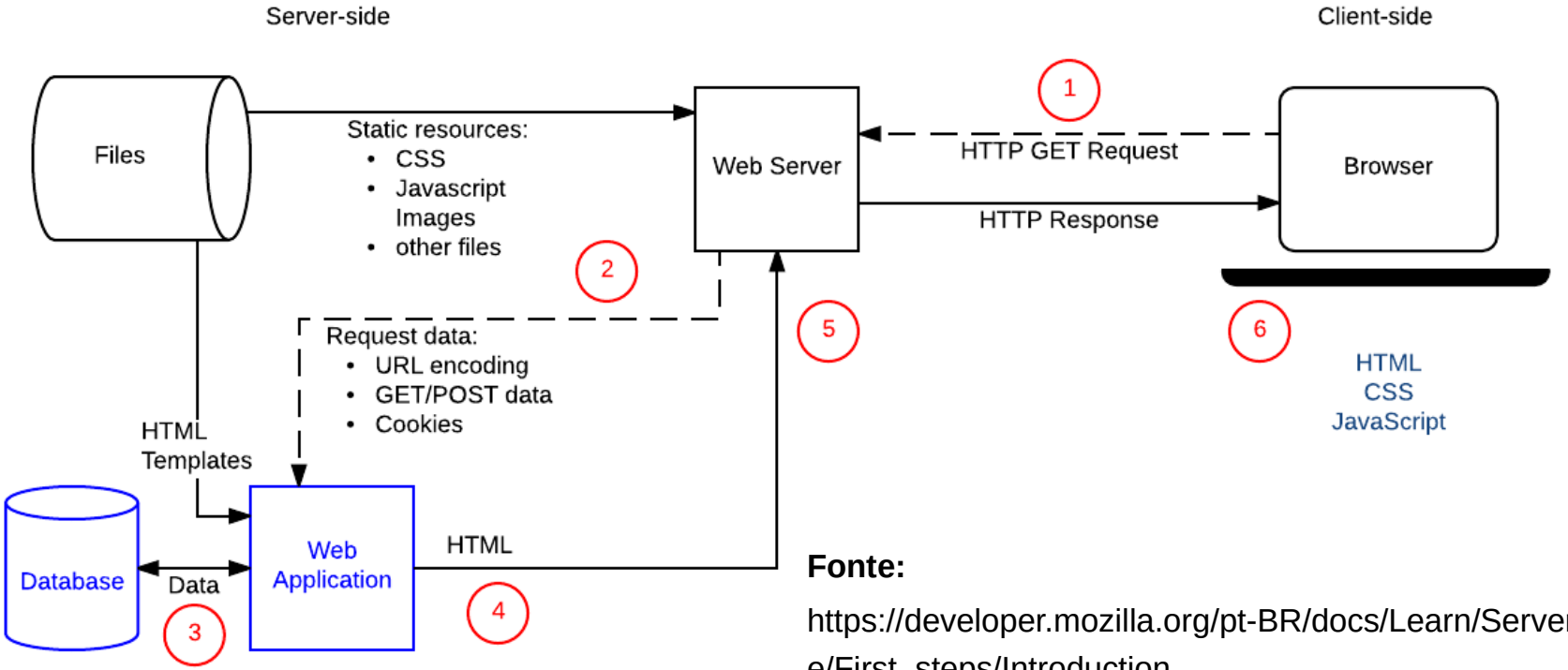
Introdução



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

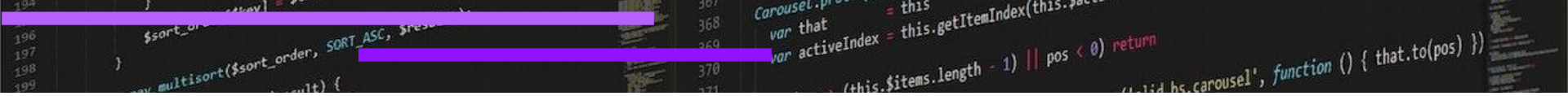


Fonte:

https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Introduction

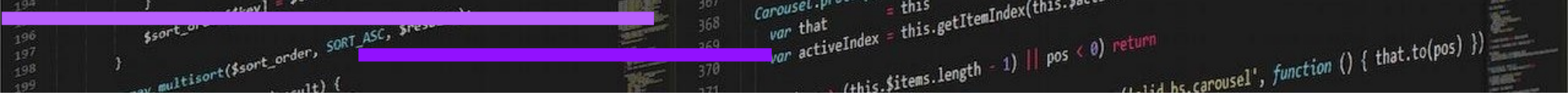
Java: História

- 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**).



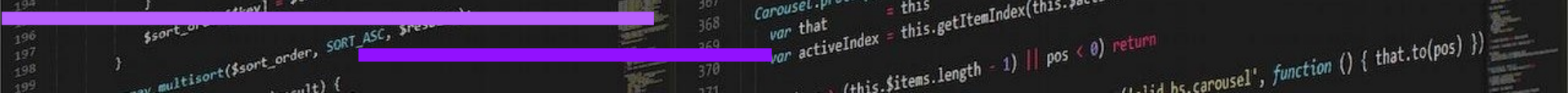
Java: Plataformas

- 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.

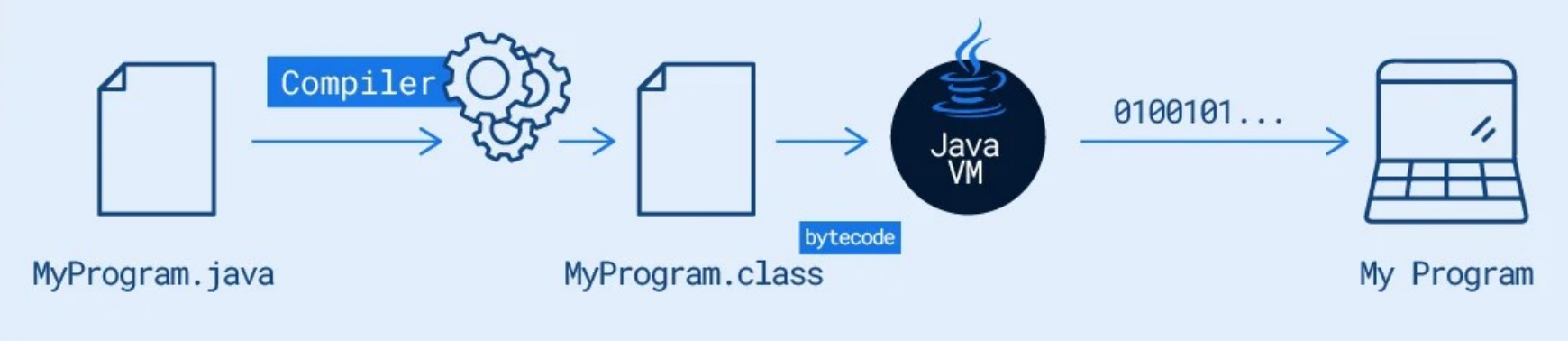


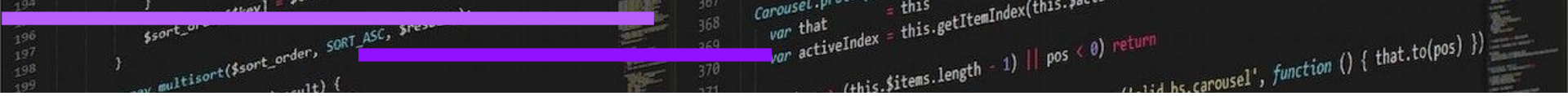
Java: características

- Independência de plataforma (portabilidade)
- Orientação a Objetos
- Não usa ponteiros
- Multithread
- Segurança
- Recursos de rede
- Gerência automática de memória
- Sintaxe similar a C/C++

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**.





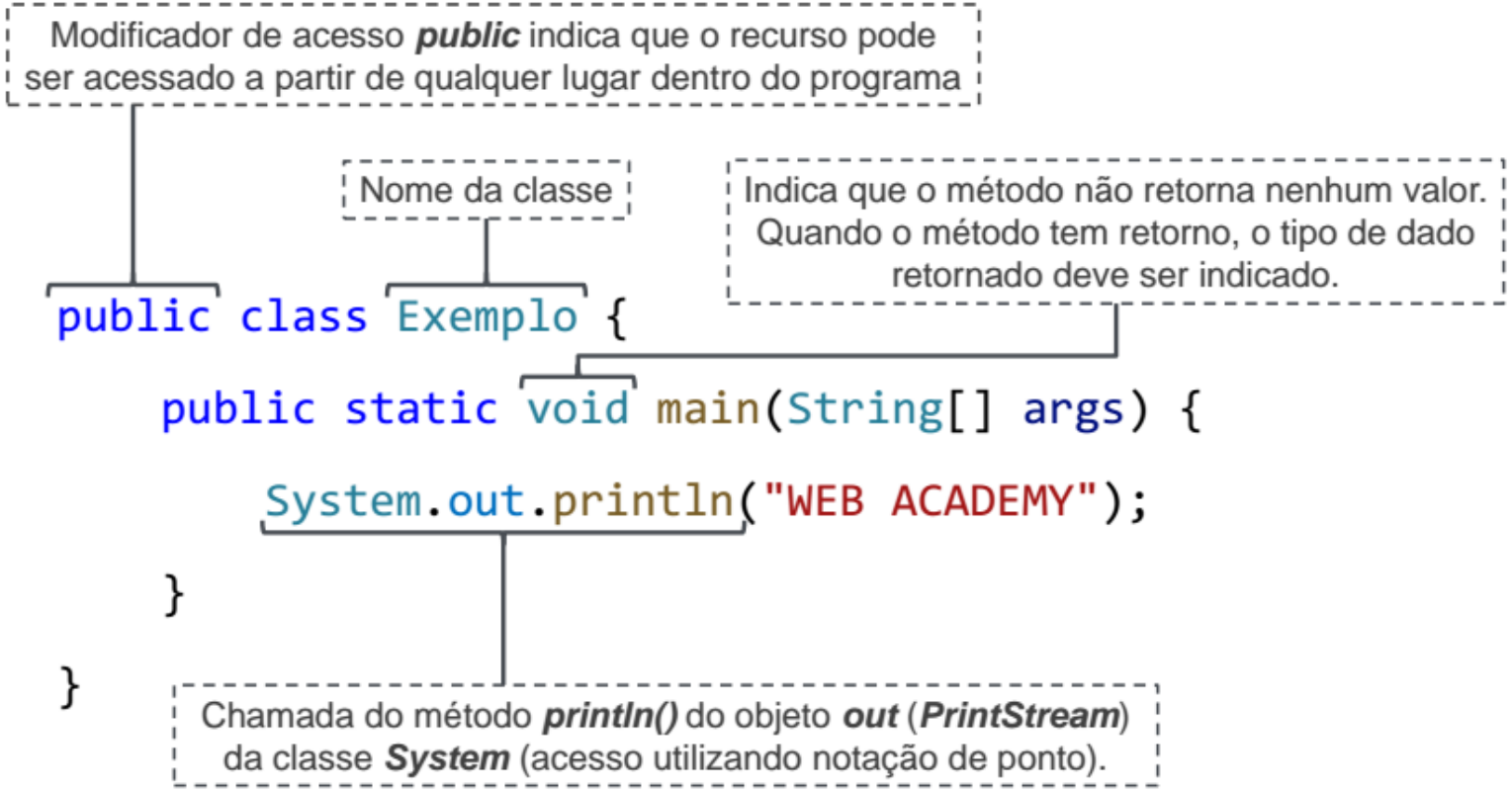
Java: exemplo

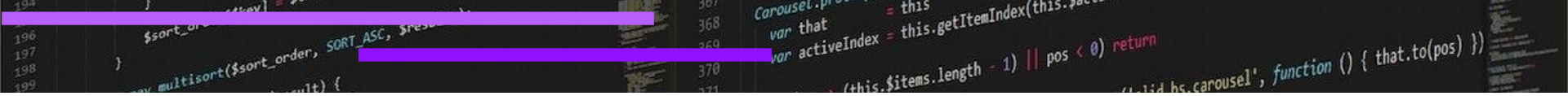
- O processo criação e execução de um aplicativo Java pode ser resumido normalmente nos seguintes passos:
  - Escrita do código-fonte (arquivo .java);
  - Compilação do programa Java em bytecodes, gerando os arquivos .class;
  - Carregamento do programa na memória pela JVM (Máquina Virtual Java);
  -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

```java
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

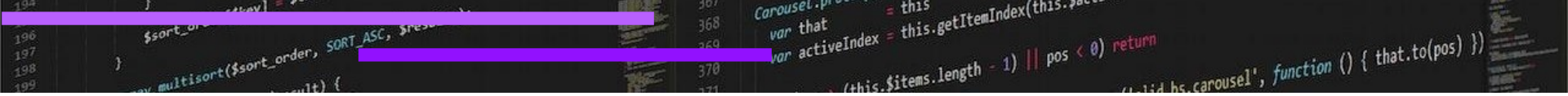
- **public**: permite que a classe, método ou variável seja acessado por qualquer código em **qualquer lugar**.
- **private**: permite que a classe, método ou variável seja acessado somente **dentro da própria classe** onde foi definido.
- **protected**: permite que a classe, método ou variável seja 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, requer que todas a variáveis tenham um tipo e não permite operações diretas entre tipos diferentes.
- Tipos primitivos: boolean, char, byte, short, int, long, float, double.

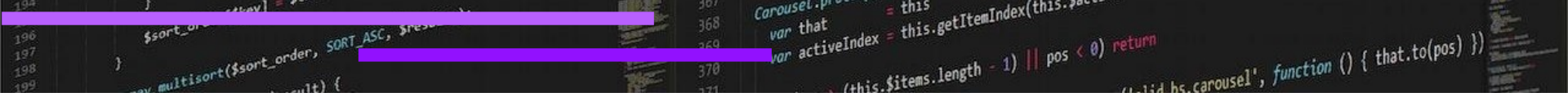
```
public class Exemplo {
    public static void main(String[] args) {
        int x = 10;
        x = "WEB ACADEMY";
        mensagem = "WEB ACADEMY";
        String mensagem = "WEB ACADEMY";
        System.out.println(mensagem);
    }
}
```

```
# 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: casting

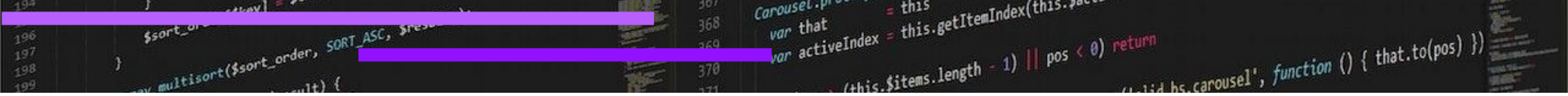
DE: \ PARA:	byte	short	char	int	long	float	double
byte	----	*Impl.*	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
short	(byte)	----	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
char	(byte)	(short)	----	*Impl.*	*Impl.*	*Impl.*	*Impl.*
int	(byte)	(short)	(char)	----	*Impl.*	*Impl.*	*Impl.*
long	(byte)	(short)	(char)	(int)	----	*Impl.*	*Impl.*
float	(byte)	(short)	(char)	(int)	(long)	----	*Impl.*
double	(byte)	(short)	(char)	(int)	(long)	(float)	----



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: 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.

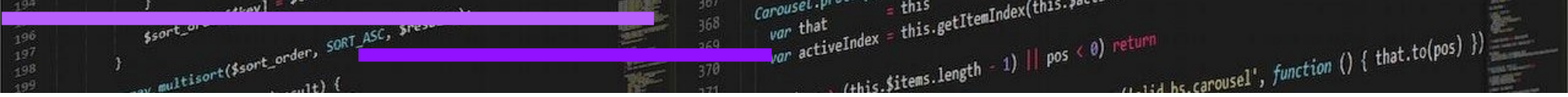
Step Out: executa o código até o final da função atual e retorna para o ponto onde a função foi chamada.

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.

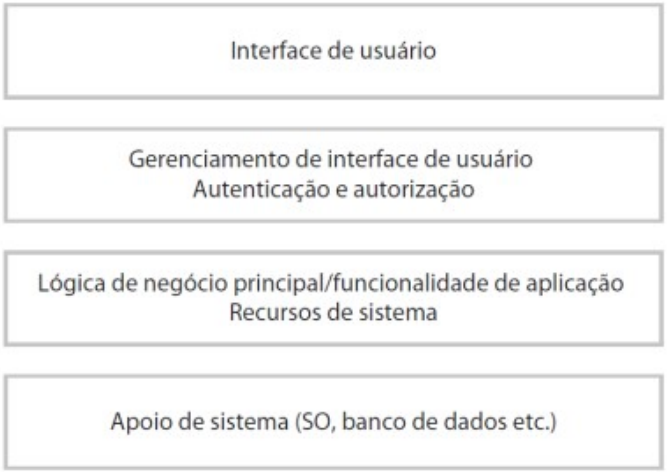


Depuração de apps Java no VS Code

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

Arquitetura em camadas

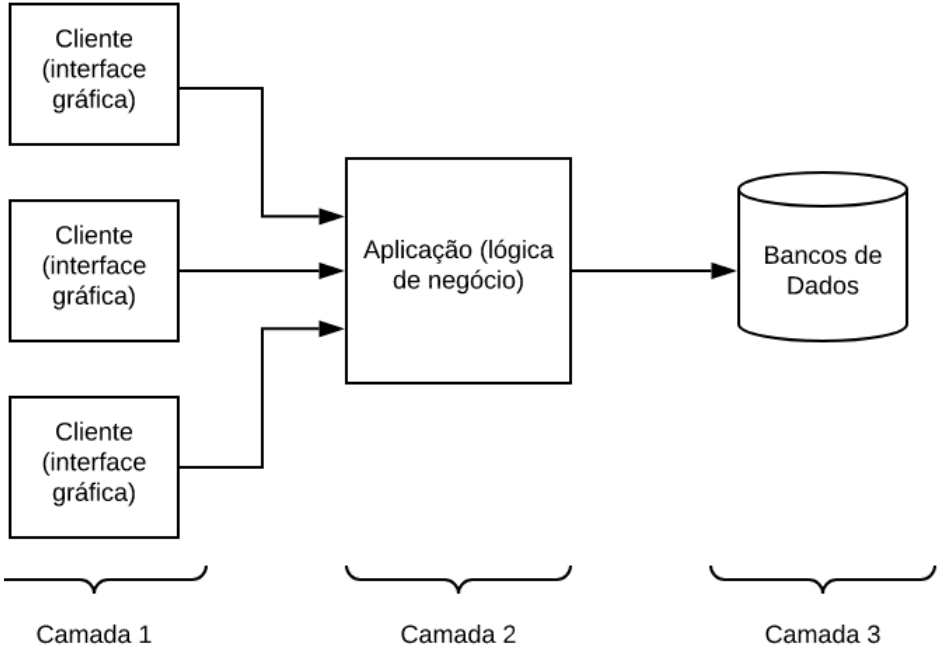
- Arquitetura em camadas é um dos **padrões arquiteturais mais usados**.
- As **classes são organizadas em módulos** de maior tamanho, chamados de camadas.
- As camadas são **dispostas de forma hierárquica**, onde uma camada somente pode usar serviços da
- camada imediatamente inferior.
- **Particiona 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

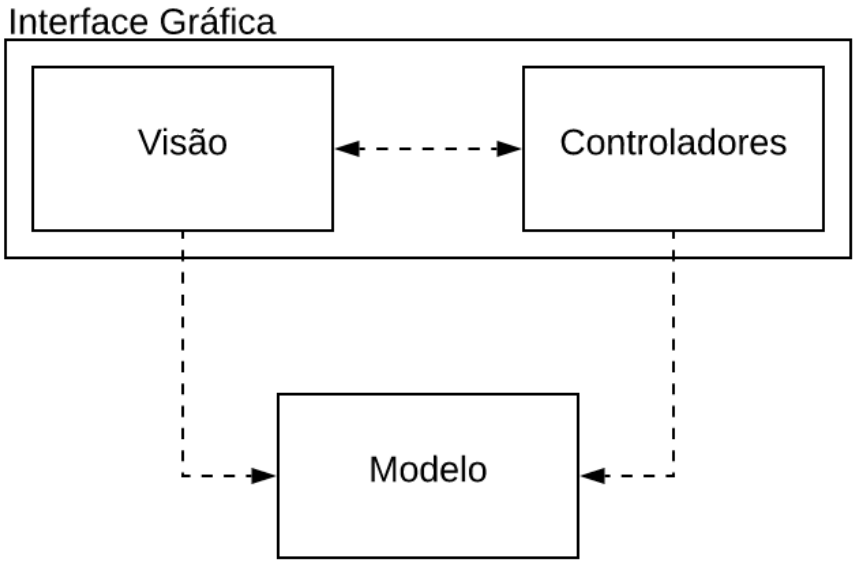
- Tipo de arquitetura comum na construção de **sistemas de informação corporativos**.
  1. **Interface com o Usuário**, responsável por toda interação com o usuário;
  2. **Lógica de Negócio**, que implementa as regras de negócio do sistema;
  3. **Banco de Dados**, que armazena os dados manipulados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



Fonte: (VALENTE, 2020)

Arquitetura MVC (Model-View-Controller)

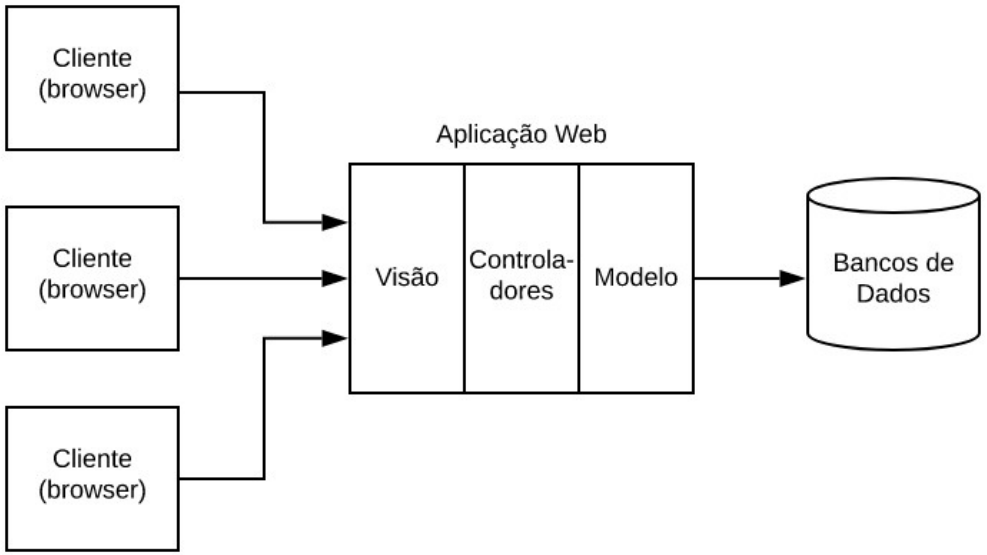
- **Visão**: responsável pela apresentação da interface gráfica do sistema, incluindo janelas, botões, menus, barras de rolagem, etc.
- **Controladores**: tratam e interpretam eventos gerados por dispositivos de entrada.
- **Modelo**: armazenam os dados manipulados pela aplicação, sem qualquer dependência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



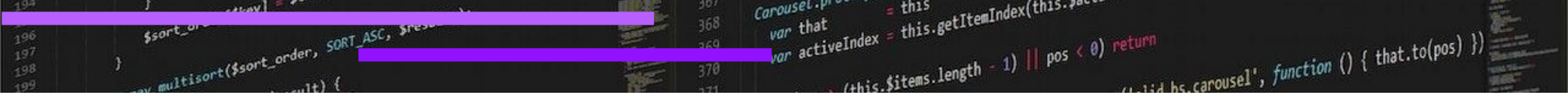
Fonte: (VALENTE, 2020)

Diferença entre MVC e três camadas

- Arquitetura em três camadas é distribuída. 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.
- MVC surgiu no final da década de 70, para ajudar na construção de interfaces gráficas, pode ser usado na implementação da camada de interface. Em um sistema Web com MVC: 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.

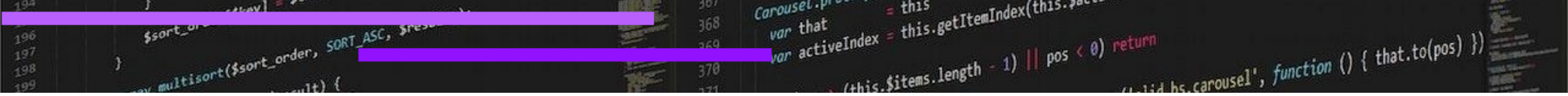


Fonte: (VALENTE, 2020)



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.



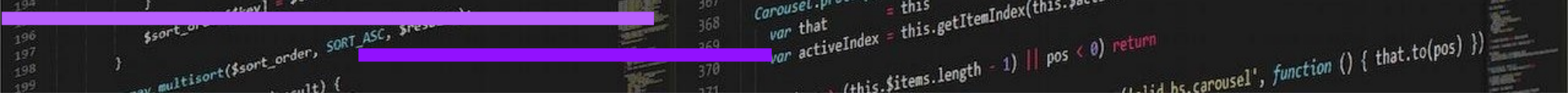
Arquitetura em camadas e pacotes Java

- **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**".

```java
package br.ufac.sgcm;

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

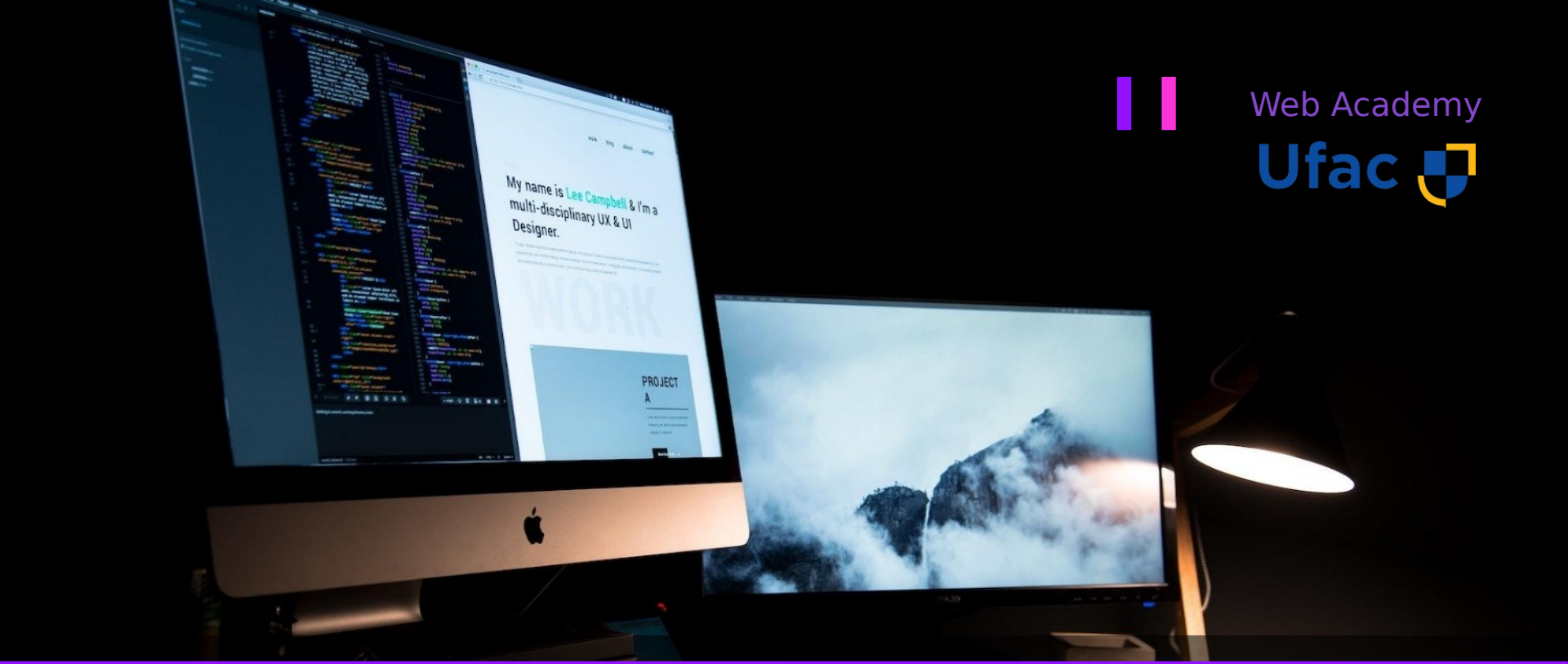
```java
import br.ufac.sgcm.Exemplo;

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

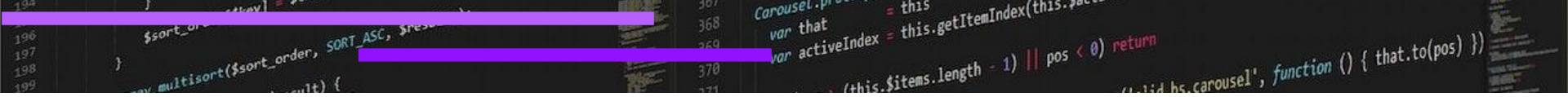
Camada (pacote)	Descrição
src\main\java\br\ufac\sgcm\model	modelos de objetos
src\main\java\br\ufac\sgcm\dao	acesso a dados e operações de banco de dados
src\main\java\br\ufac\sgcm\controller	controladores de interface do usuário (lógica de negócio)
src\main\webapp	recursos da interface do usuário



Web Academy

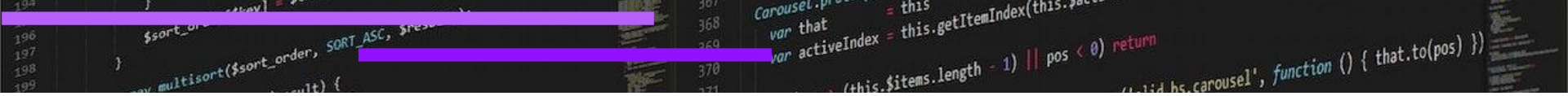
Ufac

Java e Programação Orientada a Objetos



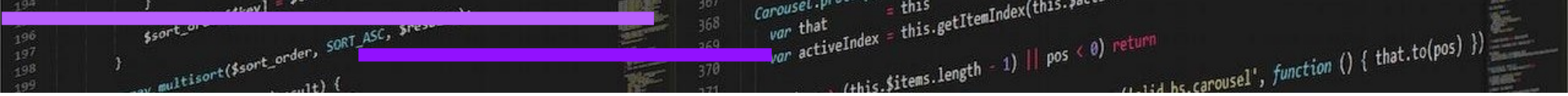
Programação Orientada a Objetos (POO)

- **É uma método de programar** que ajuda na organização e resolve muitos problemas enfrentados pela programação procedural.
- Usa **tipos de dados personalizados.**
- Em vez de operar 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.



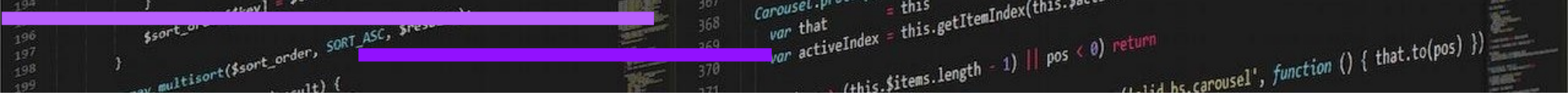
POO – Vantagens

- 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.**

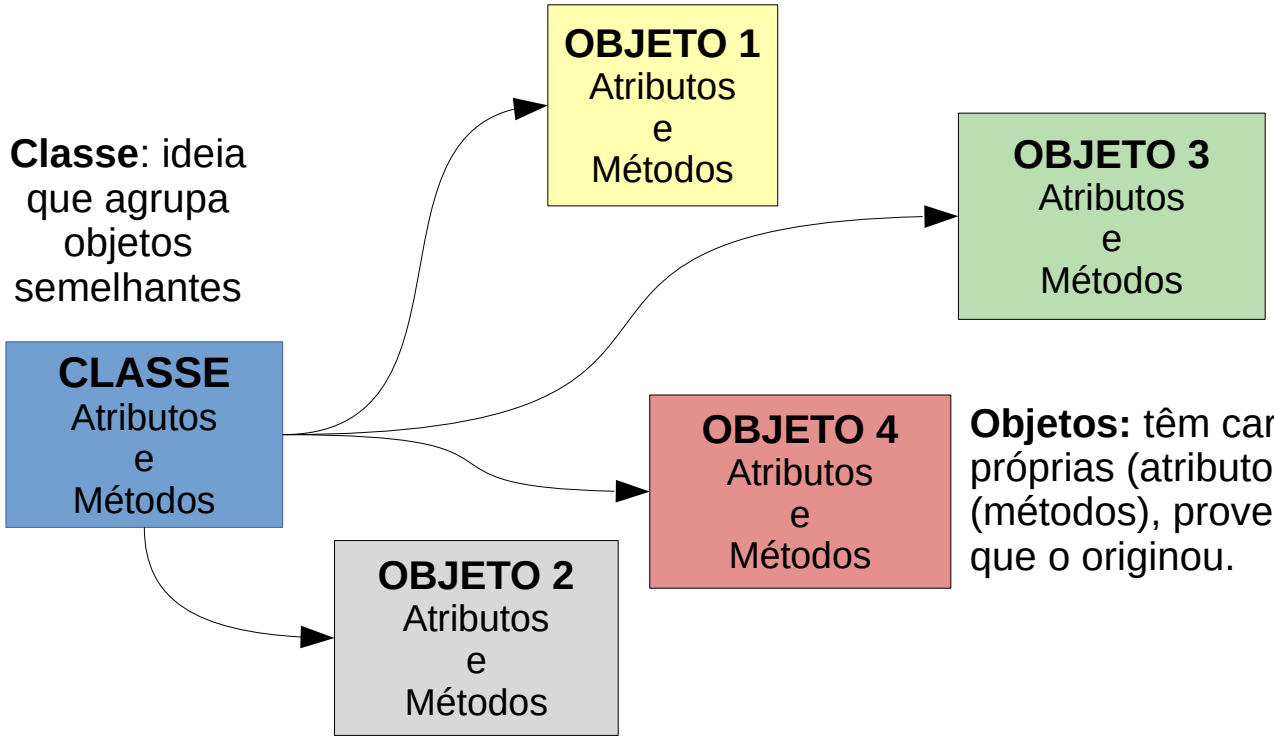


POO – Mensagens

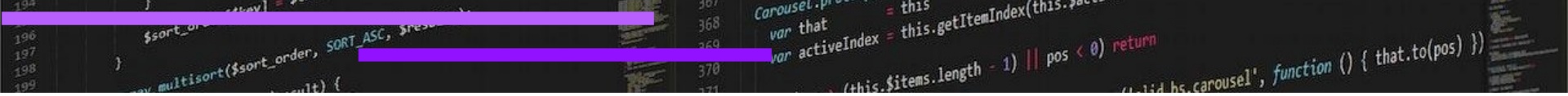
- Mensagem é um **sinal de um objeto a outro**, requisitando um serviço, usando uma operação programada no objeto chamado.
- Quando uma mensagem é recebida, uma operação é invocada no objeto chamado.
- Podem ser resultados de fórmulas matemáticas, acionamento de eventos, regra de negócio, etc.

POO – Classes e Objetos

Classe: ideia que agrupa objetos semelhantes



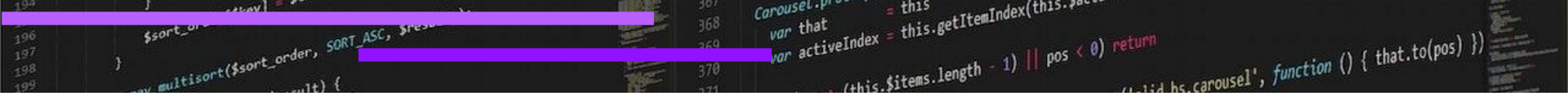
Objetos: têm características próprias (atributos) e executa ações (métodos), provenientes da classe que o originou.



POO – Classes e Objetos

- **Classe**:
  - Estrutura que abstrai um conjunto de objetos com **características e comportamentos semelhantes**.
  - **POJO** (**P**lain **O**ld **J**ava **O**bject): 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.

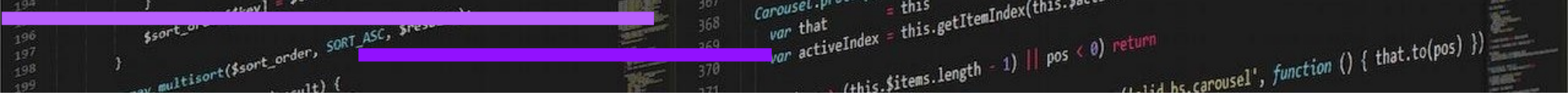
```
public class Pessoa { // Classe
    private String nome;
    private String email;
    public String getNome() {}
    public void setNome(String nome) {}
    public String getEmail() {}
    public void setEmail(String email) {}
}
public class Exemplo {
    public static void main(String[] args) {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



POO – Encapsulamento

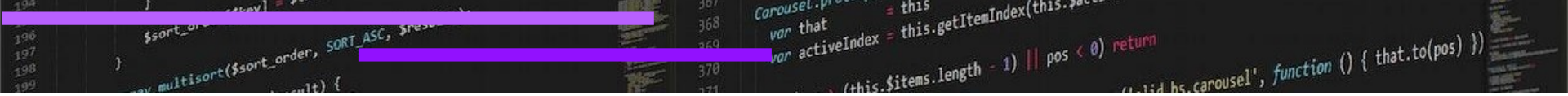
- A ideia de encapsular é, **'proteger' de forma organizada** todos os membros de uma classe: os atributos e os métodos (*getters* e *setters*) do sistema.
- Não é sinônimo de ocultar informações, pois a restrição de acesso é apenas parte do conceito.

```
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

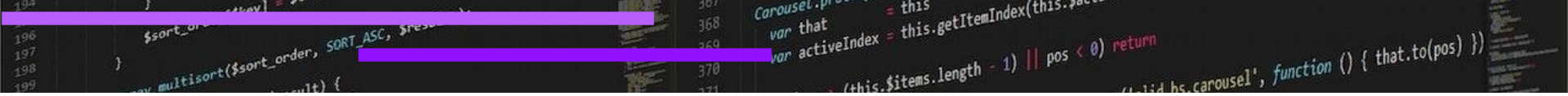
- Encapsular é **fundamental para mudanças**: não precisamos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada.
- O conjunto de métodos públicos de uma classe é também chamado de **interface da classe**, pois esta é a única maneira a qual 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**.

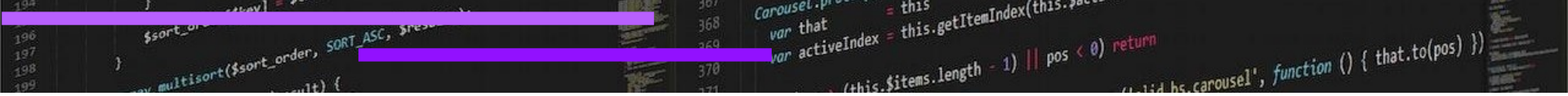
```
public class Pessoa { // Superclasse
    private String nome;
    private String email;
    public String getNome() {}
    public void setNome(String nome) {}
    public String getEmail() {}
    public void setEmail(String email) {}
}
public class Aluno extends Pessoa { // Subclasse
    private int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



POO – Polimorfismo

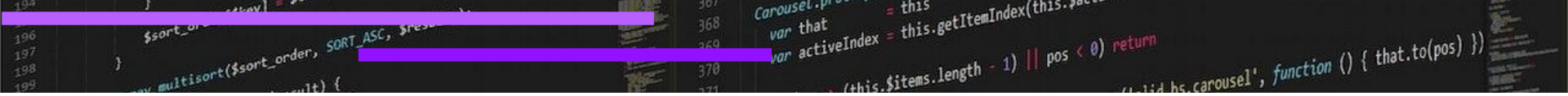
- É capacidade de um objeto poder ser referenciado de várias formas. Permite que os programas processem objetos que compartilham a mesma superclasse **como se todos fossem objetos da superclasse**.
- Uma forma de implementar polimorfismo é através de **classes abstratas**, que não podem ser instanciadas,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



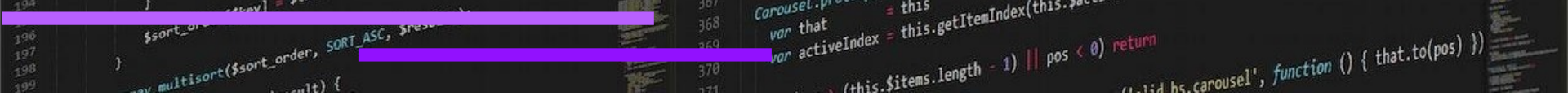
POO – Classes Abstratas

- 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

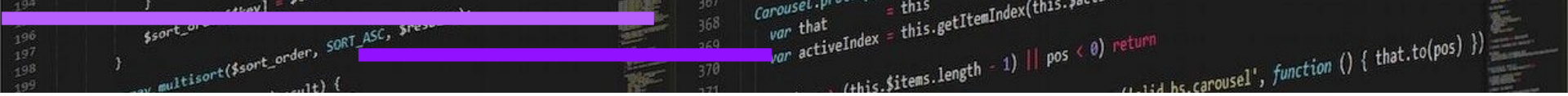
- Caso os **métodos abstratos não sejam implementados nas classes filhas**, um erro será lançado durante a execução do programa
- Para implementar um método abstrato em uma classe filha, basta definir o método **com o mesmo nome e assinatura na classe filha**. A implementação do método deve seguir a 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.

```java
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;
    }
}
```



POO – 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;
    }
}
```