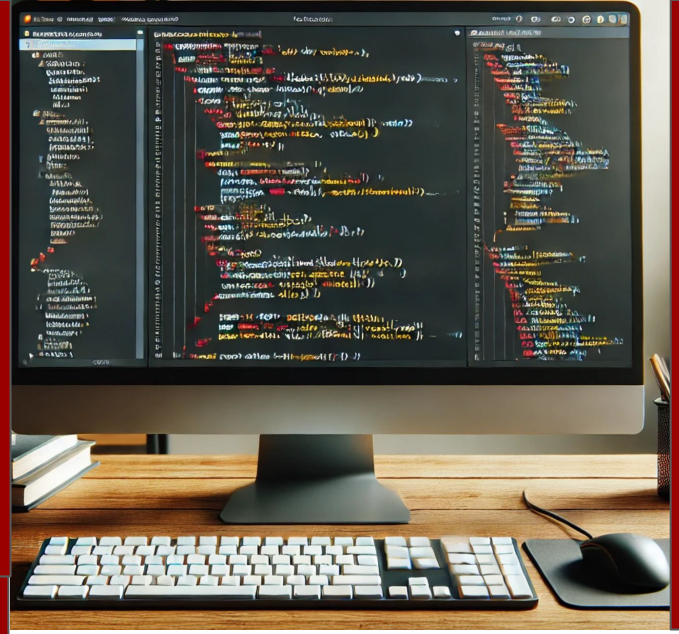
TypeScript



JS

Desenvolvimento Backend
Aula 04

Prof. Me Daniel Vieira

SENAI

Agenda

1- Paradigmas de programação
2 - Orientação a objeto
3 - Classes e atributos
4 - Herança
5- Exercícios

Paradigmas de linguagem de programação

O que é um paradigma ?

Orientação a objetos

Dart

Funcional

Javascript

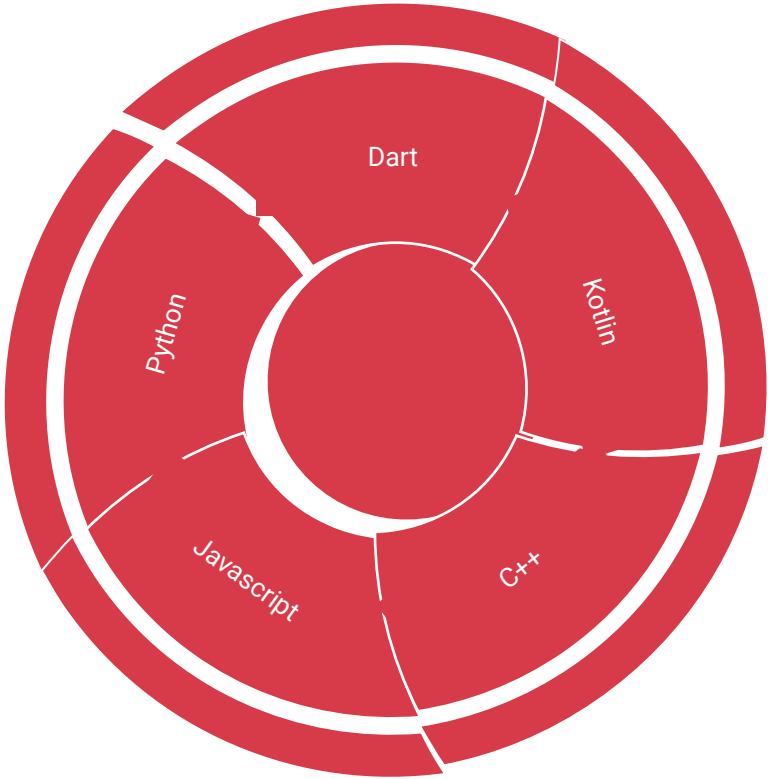
+

Procedural

Python

Concorrente

Java



Paradigmas de linguagem de programação

Paradigma imperativo

No paradigma imperativo, como o nome já revela, o desenvolvedor cria uma instrução para que a máquina processe as execuções de uma **determinada** maneira. Dentro dessa categoria existem:

Orientado ao Objeto

Esse paradigma é um dos mais aplicados por conta das vantagens que ele traz para o processo, como a **modularidade do código** e a função de criar relações entre problemas reais dentro dos termos de código.

Procedural
Perfeita para programação geral; consiste em uma lista de instruções para o computador executar as tarefas, uma de cada vez.

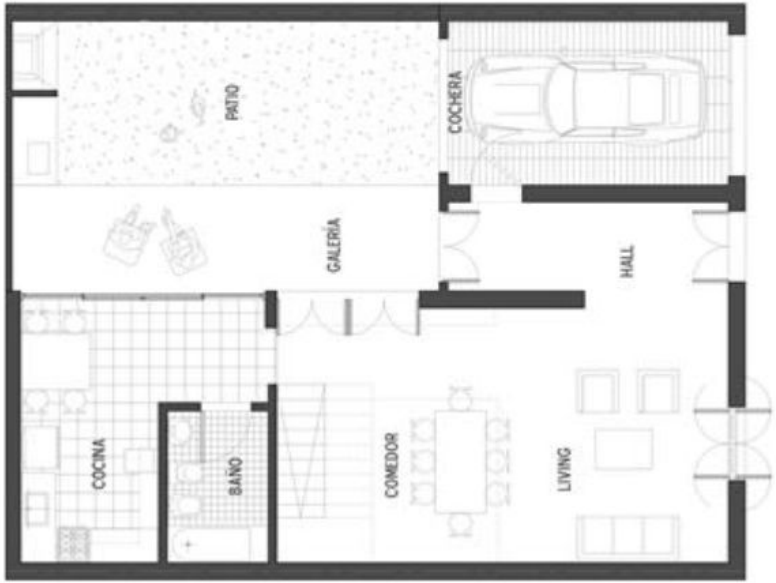
A maioria das linguagens de programação que um desenvolvedor aprende na faculdade são procedurais, como C, C++ e Java, por exemplo

Paradigmas de linguagem de programação

Programação estruturada

Paradigmas de linguagem de programação

Classe - Planta



Objeto - Casa

Paradigmas de linguagem de programação

Atributos

Casa (objeto)

Cor
Tipo de janela
Vagas na garagem
...

Paradigmas de linguagem de programação

Métodos

Casa (objeto)

Abrir portão
Abrir janelas
Ligar luzes
...

Paradigmas de linguagem de programação

```
class Casa {
  // Atributos definem características
  constructor() {
    this.cor = null; // Inicializa o atributo 'cor'
  }
}

// Função principal para execução do código
function main() {
  const nome = "Daniel";
  const minhaCasa = new Casa(); // Instancia a classe em um objeto
  minhaCasa.cor = "Blue";
  console.log(minhaCasa.cor);
}
// Chama a função principal para executar o código
main();
```

1. **Classe e Construtor**: Em JavaScript, você usa a palavra-chave `class` para definir uma classe e o método `constructor` para inicializar os atributos. No exemplo, o atributo `cor` é inicializado como `null`.
2. **Instanciação de Objetos**: O operador `new` é usado para criar uma nova instância da classe `Casa`.
3. **Métodos e Propriedades**: Em JavaScript, as propriedades dos objetos são acessadas e modificadas da mesma forma que no Dart, usando a notação de ponto (`.`).
4. **Função Principal**: Em vez de `main()` como no Dart, usamos uma função normal em JavaScript e a chamamos explicitamente para executar o código.
5. **Impressão**: `print()` em Dart é substituído por `console.log()` em JavaScript para exibir informações no console

Paradigmas de linguagem de programação

```
class Casa {
  // Atributos definem características
  constructor() {
    this.cor = null; // Inicializa o atributo 'cor'
  }

  // Métodos definem o que os objetos podem fazer
  abrirJanela(qtdeJanelas) {
    console.log(`Abrir Janela, qtde janelas ${qtdeJanelas}`);
  }

  abrirPorta() {
    console.log(`Abrir porta da casa ${this.cor}`);
  }

  abrirCasa() {
    this.abrirJanela(2);
    this.abrirPorta();
  }
}
```

```
// Função principal para execução do código
function main() {
  const minhaCasa = new Casa(); // Instancia a classe em
um objeto
  minhaCasa.cor = "Blue";
  minhaCasa.abrirCasa();
}

// Chama a função principal para executar o código
main();
```

Paradigmas de linguagem de programação

Exemplo :
Criar uma classe usuario com dois atributos:
email e senha e criar a autenticação do usuário

```
// Exemplo de uso
const usuario = new Usuario();
usuario.usuario = "Senai";
usuario.senha = "senai@2023";
usuario.autentica(); // Espera-se que "Login correto" seja impresso
```

```
class Usuario {
  constructor() {
    this.usuario = null;
    this.senha = null;
  }

  autentica() {
    const usuario = "Senai";
    const senha = "senai@2023";
    if (this.usuario === usuario && this.senha === senha) {
      console.log("Login correto");
    } else {
      console.log("Erro, tente novamente");
    }
  }
}
```

Construtor

O construtor na orientação a objeto permite a passagem de parâmetros para uma classe

```
class Fruta {
  // Criar construtor
  constructor(sabor, nome, cor, peso, diasdesdecolheita) {
    this.sabor = sabor;
    this.nome = nome;
    this.cor = cor;
    this.peso = peso;
    this.diasdesdecolheita = diasdesdecolheita;
    this.isMadura = null; // O atributo pode ser opcional, inicializado como null
  }

  // Criar método
  madura(diasParaMadura) {
    if (diasParaMadura >= this.diasdesdecolheita) {
      console.log(`A ${this.nome} está madura`);
    } else {
      console.log(`A ${this.nome} não está madura`);
    }
  }
}
```

Construtor

```
// Exemplo de uso
const fruta = new Fruta('Doce', 'Manga', 'Amarela', 0.5, 7);
fruta.madura(10); // Espera-se que "A Manga está madura" seja impresso
```

Herança

Herança é um dos pontos chave de **programação orientada a objetos** (POO). A ideia de **herança** é facilitar a **programação**. Uma classe A deve herdar de uma classe B quando podemos dizer que A é um B.

```
class Cachorro {
  String nome;
  double peso;

  comer(){
    console.log("$nome comeu");
  }
 fazerSom(){
    console.log("$nome fez dom!");
  }
```

```
class Gato {
  String nome;
  double peso;

  comer(){
    console.log("$nome comeu");
  }
  fazerSom(){
    console.log("$nome fez dom!");
  }
```

Herança

Herança é um dos pontos chave de **programação orientada a objetos** (POO). A ideia de **herança** é facilitar a **programação**. Uma classe A deve herdar de uma classe B quando podemos dizer que A é um B.

```
class Animal {
    constructor() {
      this.nome = null;
      this.idade = null;
    }

    // Método da classe base
    fazerSom() {
      console.log(`${this.nome} faz um som.`);
    }
  }
```

```
// extends herança, a classe cachorro herda da classe animal algumas
  // características como nome e idade
  class Cachorro extends Animal {
    constructor() {
        super(); // passa parametros da classe mae para a classe filha
        this.raca = null;
      // Chama o construtor da classe base

    }
```

Herança

Herança é um dos pontos chave de **programação orientada a objetos** (POO). A ideia de **herança** é facilitar a **programação.** Uma classe A deve herdar de uma classe B quando podemos dizer que A é um B.

```
  // Método sobrescrito
   fazerSom() {
     console.log(`${this.nome} late.`);
   }
   // Método adicional
   exibirInfo() {
     console.log(`Nome: ${this.nome}, Idade: ${this.idade}, Raça: ${this.raca}`);
   }
}
const Rocky = new Cachorro();
Rocky.nome = "Rocky";
Rocky.idade = "2";
Rocky.raca = "Amstaff";
Rocky.exibirInfo();
```

Exercícios

1 - Criar uma classe chamada carro com os seguintes atributos: marca, modelo, ano, motor ligado
Métodos da classe:
ligar_motor(): Um método que liga o motor do carro e atualiza o atributo motor_ligado para True.
desligar_motor(): Um método que desliga o motor do carro e atualiza o atributo motor_ligado para False.
status_motor(): Um método que retorna uma mensagem indicando se o motor está ligado ou desligado. Teste sua classe criando um objeto Carro, ligando e desligando o motor, e verificando o status do motor.

Exercícios

1 - Criar uma classe chamada carro com os seguintes atributos: marca, modelo, ano, motor ligado
Métodos da classe:
ligar_motor(): Um método que liga o motor do carro e atualiza o atributo motor_ligado para True.
desligar_motor(): Um método que desliga o motor do carro e atualiza o atributo motor_ligado para False.
status_motor(): Um método que retorna uma mensagem indicando se o motor está ligado ou desligado. Teste sua classe criando um objeto Carro, ligando e desligando o motor, e verificando o status do motor.

Exercícios

2) Criar uma classe chamada Pessoa com os seguintes atributos:
Nome, idade, profissão, salário.
Método exibetrabalho(String nomeempresa, int tempo de trabalho)
Print(nome da empresa, tempo de trabalho)
3) Criar uma classe chamada automóvel, essa classe deve ser a classe mãe e deve ter os seguintes parâmetros:
Cor do automóvel,Modelo, tipo de combustível, quantidade de rodas.
Criar classes filhas denominadas carro, moto, caminhão, herdando características da classe automóvel.
Métodos
Ligar carro, Desligar carro, abrir vidro, descer vidro.
4) Criar um programa que receba informações digitadas pelo usuário e realize transações bancárias com opções digitadas pelo usuário com uma classe clientes
Nome, profissão, saldo
Métodos Pix(double valor) Empréstimo(double valor) Saque(double valor) Extrato(double valor)

Exercícios

5- Criar uma classe denominada Máquinas
Com os seguintes atributos:
Nome da máquina
Quantidade de eixos
Rotações por minuto
Consumo de energia elétrica
Essa classe deve ser mae de outras classes.
Criar classe denominada furadeira herdando o nome da máquina, rotações por minuto, consumo de energia elétrica.
Criar métodos para ligar, desligar a máquina e um método para ajustar a velocidade de rotação da máquina.

Exercícios

6- Criar uma classe denominada Produtos e deve ter os seguintes parâmetros:
Nome do produto
Quantidade
Preço do produto
Tipo de comunicação
Consumo de energia elétrica
Essa classe produtos deve ser mãe de outras classes como fritadeira, televisão, ar-condicionado.
As classes filhas devem possuir métodos:
Ligar, desligar, ajuste de temperatura com passagem de parâmetros para setpoint.

Exercícios

7- Criar uma classe denominada Livros e deve ter os seguintes parâmetros:
Nome do livro
Quantidade
Preço do livro
Nome do autor
Nº da Edição
métodos emprestar, devolver

Obrigado!

Prof. Me Daniel Vieira

Email: danielvieira2006@gmail.com

Linkedin: Daniel Vieira

Instagram: Prof daniel.vieira95