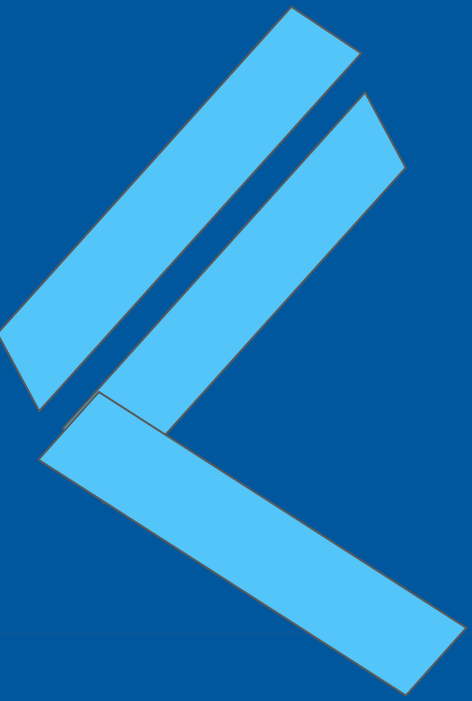


Desenvolvimento Mobile 1
Aula 05

Prof. Me Daniel Vieira

SENAI

Agenda

1- Paradigmas de Programação
2-Programação orientada a objeto
3 - Classe e objeto
4 -Herança
5 - Construtor
6- Exercício

1-Paradigmas de programação

1- Paradigmas de programação

- **Paradigma imperativo**

No paradigma imperativo, como o próprio nome sugere, o desenvolvedor descreve passo a passo as instruções que a máquina deve seguir para executar uma tarefa. Dentro do paradigma imperativo, existem duas abordagens principais:

- **Programação Procedural** – Este paradigma é ideal para tarefas gerais de programação, pois se baseia em uma sequência de instruções que o computador executa de forma linear, uma de cada vez.

A maioria das linguagens de programação tradicionais, como C, C++ e Java, são procedurais.

1- Paradigmas de programação

- **Programação Orientada a Objetos (POO) –** 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.

2- Programação orientada a objeto

A Programação Orientada a Objetos (POO) é um paradigma de programação que organiza o software em objetos. Esses objetos são entidades que combinam dados (atributos) e comportamentos (métodos), representando elementos do mundo real ou conceitual no código.

2- Programação orientada a objeto

Programação estruturada

Programação orientada a objeto

2- Programação orientada a objeto

Na programação estruturada, temos os seguintes conceitos:

Dados globais – Informações que podem ser acessadas em qualquer parte do código, ou seja, têm um escopo global.

2- Programação orientada a objeto

Variáveis – Espaços de memória para armazenar valores. Elas permitem que o programa guarde e manipule dados durante sua execução.

Condições – Verificações de valores ao longo do código, que ajudam a tomar decisões e direcionar o fluxo de execução, dependendo do resultado da comparação.

2- Programação orientada a objeto

Ação – Refere-se à execução de comportamentos específicos, como exibir informações na tela, salvar itens em um banco de dados, entre outras operações.

2- Programação orientada a objeto

Na programação orientada a objetos, cada objeto é composto por dados (também chamados de variáveis ou atributos) e métodos (funções associadas ao objeto). Isso significa que cada objeto possui seus próprios atributos e métodos, o que contribui para a modularidade e a organização do código.

2- Programação orientada a objeto

Os dados de um objeto representam os valores associados ao objeto específico que foi instanciado a partir de uma classe. Já os métodos são as funções definidas na classe, que podem ser executadas pelos objetos criados a partir dessa classe.

2- Programação orientada a objeto

Porque utilizar programação orientada a objetos ?

- Complexidade do Software:

Com o crescimento do software, tornou-se difícil organizar e manter programas estruturados, especialmente os com muitos trechos de código interdependentes.

A POO oferece uma abordagem modular, onde o software é dividido em objetos, cada um com sua responsabilidade.

2- Programação orientada a objeto

- Reutilização de Código:

Em paradigmas anteriores, o código muitas vezes precisava ser reescrito para ser reutilizado.

Com a POO, conceitos como herança e polimorfismo permitem reaproveitar funcionalidades existentes.

- Manutenção e Escalabilidade:

2- Programação orientada a objeto

A modularidade da POO facilita encontrar e corrigir erros.

Classes e objetos podem ser adicionados ou modificados sem afetar outras partes do sistema.

- Modelagem do Mundo Real:

2- Programação orientada a objeto

A POO permite mapear entidades do mundo real diretamente para o código.

Um carro, por exemplo, pode ser representado como uma classe com atributos como marca e modelo, e métodos como acelerar.

- Segurança e Controle de Acesso:

2- Programação orientada a objeto

- O encapsulamento protege os dados do objeto, permitindo acesso apenas por métodos específicos.
- Isso reduz erros causados por alterações não controladas nos dados.
- Colaboração em Equipes:
- Projetos grandes envolvem múltiplos desenvolvedores.

2- Programação orientada a objeto

- A POO facilita a divisão do trabalho em módulos independentes (classes e objetos), promovendo a colaboração.

Aplicações da POO

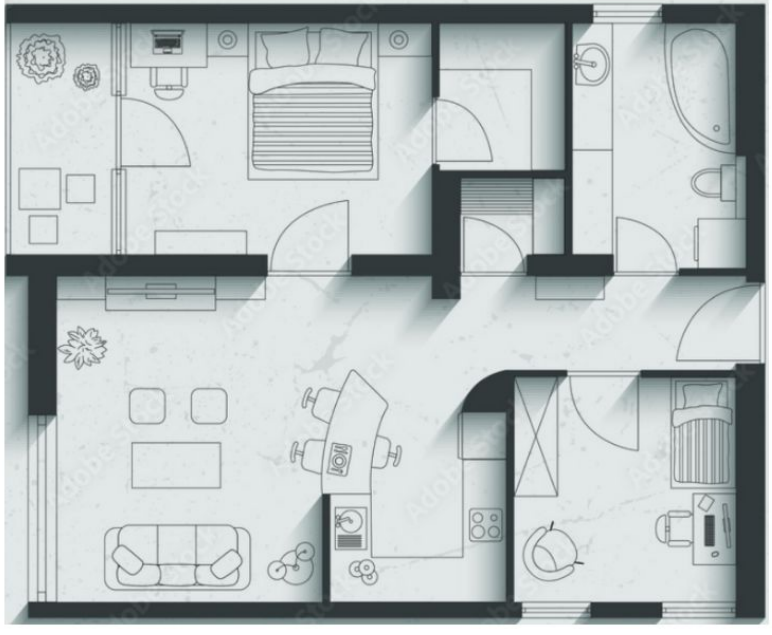
- Desenvolvimento de Software Corporativo:
- Sistemas de gestão (ERP, CRM) frequentemente modelam entidades como clientes, produtos e pedidos.
- Jogos:

2- Programação orientada a objeto

- Jogos digitais usam objetos como personagens, inimigos, armas, e ambientes.
- Desenvolvimento Mobile:
- Em Flutter, o conceito de widgets é baseado na POO.
- Sistemas Orientados a Serviços:
- APIs e serviços web modelam entidades e suas interações usando objetos.

3- Classe e objeto

Classe: Casa



Objeto: Casa

3- Classe e objeto

```
1. /*
 2. Cria uma classe chamada carro, essa classe possui dois atributos
que é a marca e o ano e um método chamado acelerar que irá escrever
no terminal o carro está acelerando.
 3. */
 4. class Casa {
 5.   String cor;
 6.   int qtde_p;
 7.
 8.   void abrirporta() {
 9.     print("A porta esta aberta");
10.   }
11. }
12.
```

3- Classe e objeto

```
1. Casa minhaCasa = Casa();
2. minhaCasa.cor = "azul";
3. minhaCasa.qtde_p = 2;
4. minhaCasa.abrirporta();
```

3- Classe e objeto

```
1. /*
 2. Cria uma classe chamada carro, essa classe possui dois atributos
que é a marca e o ano e um método chamado acelerar que irá escrever
no terminal o carro está acelerando.
 3. */
 4. class Carro {
 5.   String marca;
 6.   int ano;
 7.
 8.   void acelerar() {
 9.     print("O carro está acelerando!");
10.   }
11. }
12.
```

3- Classe e objeto

```
1. Carro meuCarro = Carro();
2. meuCarro.marca = "Toyota";
3. meuCarro.ano = 2020;
4. meuCarro.acelerar();
```

3- Classe e objeto

```
1. /*
 2. Classe pessoa com atributos privados que garantem segurança para
proteger os dados
 3. */
 4. class Pessoa {
 5.   String _nome; // Atributo privado
 6.
 7.   void setNome(String nome) {
 8.     _nome = nome;
 9.   }
10.
11.   String getNome() {
12.     return _nome;
13.   }
14. }
15.
```

Exemplo criar um usuário com parâmetros

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

```
class Usuario
{
   String ? usuario;
   String? senha;
   void autentica()
   {
      var usuario = "Senai";
      var senha = "senai@2025";
      if(this.usuario == usuario && this.senha == senha)
      {
        print("Login correto");
      }
      else{
            print("Erro, tente novamente");
      }
   }
}
```

Exemplo criar um usuário com parâmetros

```
void main()
{
  Usuario usuario= Usuario();
  usuario.usuario= "Daniel";
  usuario.senha= "senai@2023";
  usuario.autentica();


}
```

4- Herança

- Herança:

Permite que uma classe herde atributos e métodos de outra classe.

Promove a reutilização de código.

4- Herança

```
1. /*Cria uma classe denominada animal com o método dormir.
 2. Cria uma classe denominada cachorro herdando o método dormir da
classe animal.
 3. */
 4. class Animal {
 5.   void dormir() {
 6.     print("Animal dormindo...");
 7.   }
 8. }
 9. class Cachorro extends Animal {
10.   void latir() {
11.     print("Cachorro latindo...");
12.   }}
13.
14.
```

5- Construtor

Construtor: Na programação orientada a objetos em Dart, um **construtor** é um método especial usado para inicializar uma classe. Ele permite passar parâmetros ao criar um objeto e configura os atributos da classe com os valores fornecidos. Assim, garante que os objetos sejam instanciados com valores consistentes.

- Como funciona o Construtor em Dart?

Definição: O construtor tem o mesmo nome da classe.

Parâmetros: Os atributos da classe podem ser inicializados diretamente dentro do construtor usando a sintaxe this.atributo.

5- Construtor

Uso do this: O this refere-se aos atributos da instância atual da classe, diferenciando-os de variáveis ou parâmetros com o mesmo nome.

5- Construtor

```
1. /*
 2. Construtor com this:
 3.
 4. Fruta(this.sabor, this.nome, this.cor, this.peso, this.diasDesdeColheita);
 5. O uso de this associa os parâmetros do construtor diretamente aos atributos da
classe, reduzindo a necessidade de inicializá-los manualmente.
 6.
 7. Atributo Opcional:
 8. O atributo isMadura é marcado como nulo opcional (bool?), o que significa que ele
pode ou não ter um valor atribuído.
 9.
10. Métodos:
11. A classe possui um método verificaMaturidade que realiza uma ação com base nos
valores dos atributos.
12. */
```

5- Construtor

```
13. class Fruta {
14.   // Atributos da classe
15.   String sabor;
16.   String nome;
17.   String cor;
18.   double peso;
19.   int diasDesdeColheita;
20.   bool? isMadura; // Opcional (pode ser nulo)
21.
22.   // Construtor: inicializa os atributos da classe
23.   Fruta(this.sabor, this.nome, this.cor, this.peso, this.diasDesdeColheita);
```

5- Construtor

```
25.    // Método para verificar se a fruta está madura
26.    void verificaMaturidade(int diasParaMaturidade) {
27.      if (diasDesdeColheita >= diasParaMaturidade) {
28.        print("A $nome está madura.");
29.      } else {
30.        print("A $nome ainda não está madura.");
31.      }
32.    }
33. }
34.
```

5- Construtor

```
35. void main() {
36.   // Criando um objeto da classe Fruta
37.   Fruta manga = Fruta("Doce", "Manga", "Amarela", 1.2, 5);
38.
39.   // Chamando o método para verificar maturidade
40.   manga.verificaMaturidade(4); // Saída: "A Manga está madura."
41.   manga.verificaMaturidade(6); // Saída: "A Manga ainda não está
madura."
42. }
```

6- Exercícios

1. Crie uma classe mãe chamada “Animal” com os atributos:
String nome, int idade, String cor, String raça.

2. Crie uma classe “Filha” com o tipo de animal pássaro, cachorro, tigre, peixe e o atributo: peso, métodos acordou, dormiu.

3. Crie 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 a mãe 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.

6- Exercícios

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

Obrigado!

Prof. Me Daniel Vieira

Email: danielvieira2006@gmail.com

Linkedin: Daniel Vieira

Instagram: Prof daniel.vieira95