Instituto Federal de Ciência, Educação e Tecnologia do Piauí – IFPI

Aluno: Vinícius Gomes Araújo Costa

Disciplina: Programação Orientada a Objetos (POO) – Turma Especial

Professor: Ely Miranda

Curso: Análise e Desenvolvimento de Sistemas (ADS) – Módulo III – Tarde

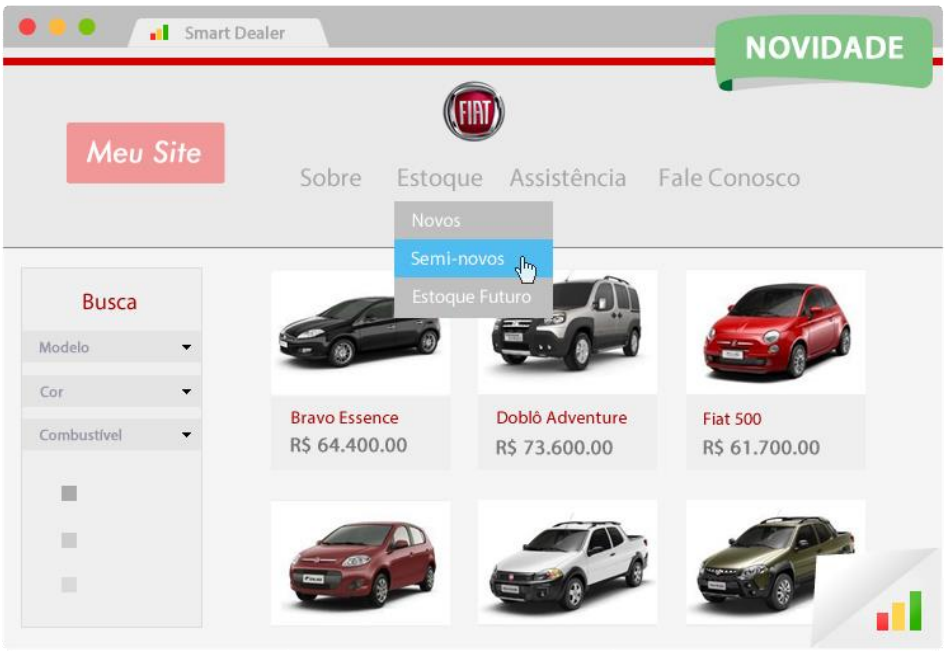
Atividade Extra

1.

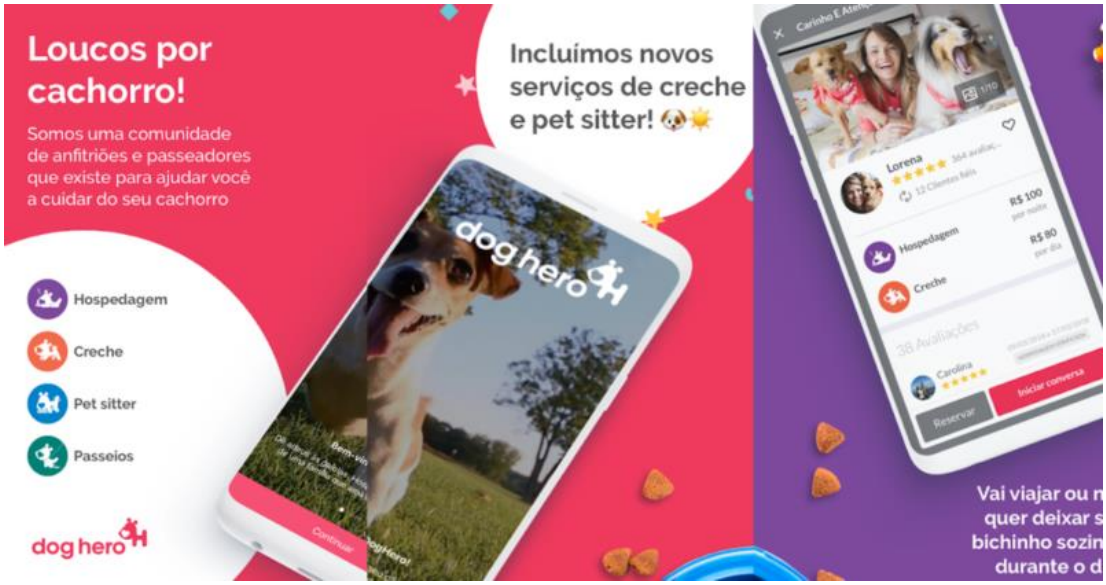
a) **Abstração**. Pois a capacidade de controlar a complexidade baseia-se em uma espécie de "**design**", focar no essencial em Programação Orientada a Objetos (POO) é fundamental para o funcionamento de um sistema, eliminar dados irrelevantes a um sistema e focar na supressão ou desconsideração de detalhes que não são essenciais prioriza o que é realmente importante, tornando o software mais compreensível, gerenciável e mais adaptável.

Exemplificação:

Sistema de gestão para concessionárias de veículos, um site com preços e modelos de carros, vulgo um sistema hierárquico de classes para um objeto carro em Programação Orientada a Objetos (POO).



Aplicativo para pets, como o da foto abaixo, o **dog hero** é mais um exemplo de serviço aplicado a marcação de banho, tosas e passeio com seu animal de estimação.



b) Encapsulamento. Pois isso significa que nenhum acesso direto é concedido aos **dados** (**atributos**) de um objeto; em vez disso, o acesso é fornecido apenas através de métodos (funções) definidos na classe. Em resumo, **o encapsulamento** permite que você controle o acesso aos dados e proteja a integridade do objeto, garantindo que ele só possa ser modificado ou acessado de acordo com as regras definidas na classe. Isso ajuda a manter a consistência dos objetos e reduzir a probabilidade de erros de programação.

Principais aspectos do encapsulamento:

1) Acesso restrito: Os atributos de uma classe são declarados como privados (**private**), o que significa que eles só podem ser acessados internamente pela própria classe;
2) Métodos de acesso: A classe define métodos públicos (**public**) que permitem interações controladas com os atributos privados. Esses métodos são conhecidos como “métodos de acesso” ou “**getters**” e “**setters**”;
3) **Getters (métodos de acesso)**: São métodos públicos que permitem a obtenção dos valores dos atributos privados. Eles permitem a leitura dos dados, mas não permitem a modificação direta;
4) **Setters (métodos de acesso)**: São métodos públicos que permitem a alteração dos valores dos atributos privados. Eles fornecem uma forma controlada de modificar os dados, permitindo que a classe aplique validações ou restrições;
5) Proteção contra acesso não autorizado: Ao usar o encapsulamento, você pode proteger os atributos de uma classe contra acessos não autorizados ou alterações acidentais, garantindo que apenas os métodos aprovados permitam a manipulação dos dados.

O encapsulamento é uma das práticas fundamentais em POO e ajuda a melhorar a segurança, modularidade e manutenibilidade do código. Ao ocultar a implementação interna dos objetos, você pode alterá-la sem afetar o restante do programa, desde que a interface pública (métodos) permaneça inalterada.

c) Construtor. Pois o conceito dado, o “**constructor**” em **Programação Orientada a Objetos(POO)**; é um método especial dentro de uma classe que é automaticamente

executado sempre que um novo objeto é criado a partir dessa classe. Sua principal finalidade é inicializar os atributos (propriedades) do objeto e realizar quaisquer outras ações necessárias para configurá-lo corretamente.

Ex:.

```
class Pessoa {
    private nome: string;
    private idade: number;

    constructor (nome: string, idade: number) {
        this.nome = nome;
        this.idade = idade;
    }
}

// Métodos de acesso podem ainda serem criados (getters e setters)
// Podem seguir o constructor

// Outros métodos da classe
// ...

// Criando objetos usando o constructor
const pessoa1 = new Pessoa("Alice", 25);
const pessoa2 = new Pessoa("Bob", 30);
```

/* **O constructor da classe** recebe dois parâmetros: **"nome" e "idade"**, e usa-se a palavra-chave **"this"** para atribuir esses valores aos atributos correspondentes da classe. */

/* A partir desse constructor, podemos criar objetos da classe **"Pessoa"** passando os valores para **"nome" e "idade"**. Quando usamos a palavra-chave **"new" para criar um novo objeto, o constructor é automaticamente invocado, e o objeto é criado e inicializado com os valores fornecidos.** */

/* Com o **TypeScript**, ele nos permite criar classes com constructores que **facilitam a criação e inicialização de objetos com valores específicos para seus atributos**. Isso é particularmente útil quando precisamos garantir que os objetos criados estejam em um estado válido e consistente desde o início. */

2) Assinale V ou F:

(F) Classes são instâncias de objetos;

(F) Classes são apenas agrupamentos de métodos;

(V) Atributos definem o estado de um objeto;

(V) Métodos podem ser análogos às funções e em TypeScript podem ser escritos dentro ou fora da classe, como em C++;

(F) Podemos ignorar o uso de tipos em TypeScript.

Justificativa:

- Na verdade, é o contrário: **objetos são instâncias de classes**. Classes e objetos são termos importantes na Programação Orientada a Objetos (POO). A POO é um paradigma de programação que organiza o código em torno de "objetos" que encapsulam dados e comportamentos relacionados. As classes fornecem um modelo para criar objetos.

Uma classe é como um **"plano" ou "blueprint"** que define a estrutura, comportamento e atributos que os objetos de um determinado tipo terão. Ela especifica quais propriedades (atributos) e métodos (comportamentos) os objetos a partir dela terão. Por exemplo, imagine uma classe chamada "Carro". Essa classe pode ter atributos como "cor", "marca", "modelo" e métodos como "ligar", "acelerar", "frear".

Quando você cria um objeto em POO, você está instanciando uma classe, ou seja, criando uma cópia da classe definida. O objeto é uma instância específica dessa classe, com seus próprios valores para os atributos e comportamentos definidos pelos métodos da classe. Voltando ao exemplo do carro, você pode criar vários objetos diferentes a partir da classe "Carro", cada um com sua própria cor, marca e modelo específicos.

Portanto, os objetos são instâncias concretas e individuais criadas a partir de uma classe, enquanto as classes são apenas uma descrição ou modelo que define como esses objetos devem ser estruturados e se comportar.

- Na verdade, essa afirmação está incorreta. Classes são muito mais do que simples agrupamentos de métodos. Em Programação Orientada a Objetos (POO), as classes são uma parte fundamental do paradigma, e têm um papel crucial na organização do código e na modelagem de objetos.

Uma classe é uma estrutura de programação que encapsula dados (atributos) e comportamentos (métodos) relacionados em uma única unidade. Além dos métodos, ela também pode conter atributos (também chamados de propriedades ou campos) que representam o estado do objeto. Os métodos definem as operações que o objeto pode realizar e como ele interage com outros objetos.

Características importantes de uma classe:

1) **ATRIBUTOS**: são variáveis que armazenam o estado de um objeto. Eles representam as características do objeto. Por exemplo, em uma classe "Pessoa", os atributos podem ser "nome", "idade" e "altura";
2) **MÉTODOS**: são funções associadas à classe que definem o comportamento do objeto. Os métodos permitem que o objeto realize operações e interaja com o mundo externo. Por exemplo, em uma classe "Pessoa", os métodos podem ser "andar", "falar" e "comer";

3) **ENCAPSULAMENTO**: as classes permitem encapsular seus atributos e métodos, o que significa que o acesso a eles pode ser controlado. Isso ajuda a proteger os dados internos da classe e permite que o código seja mais organizado e seguro;
4) **ABSTRAÇÃO**: classes fornecem um nível de abstração, permitindo que você modele objetos do mundo real em termos de suas características e comportamentos essenciais, ignorando detalhes irrelevantes;
5) **HERANÇA**: classe podem herdar características de outras classes, o que significa que uma classe pode ser derivada de outra classe, aproveitando seus atributos e métodos e estendendo ou personalizando seu comportamento;
6) **POLIMORFISMO**: esse conceito permite que diferentes classes possam ser tratadas de maneira uniforme quando têm uma interface comum, mesmo que cada uma delas implemente essa interface de forma diferente;

Portanto, as **classes** não são apenas agrupamentos de métodos, mas uma estrutura essencial para criar objetos, definir suas características e comportamentos, e aplicar princípios importantes da POO, como encapsulamento, abstração, herança e polimorfismo.

- Afirmação corretíssima! Pois os atributos definem o estado de um objeto. Em Programação Orientada a Objetos, os atributos (também conhecidos como campos ou propriedades) de uma classe são responsáveis por definir o estado de um objeto. O estado de um objeto representa os valores dos atributos que o objeto contém em um determinado momento.

Os atributos de uma classe representam as características ou informações que um objeto dessa classe pode conter. Por exemplo, se tivermos uma classe “Círculo”, ela pode ter atributos como “raio” e “cor”. Para cada objeto criado a partir dessa classe, o atributo “raio” pode ter um valor específico que define o tamanho do círculo, e o atributo “cor” pode ter um valor específico que define a cor do círculo.

Quando um objeto é criado a partir de uma classe, ele adquire o estado inicial definido pelos valores iniciais dos atributos da classe. À medida que o programa é executado, o estado do objeto pode ser alterado, ou seja, os valores dos atributos podem ser modificados ao longo do tempo.

Por exemplo, suponha que temos um objeto chamado “meuCirculo” criado a partir da classe “Circulo”. Inicialmente, “meuCirculo” pode ter um raio de 5 unidades e uma cor azul. Se, posteriormente, o raio for alterado para 7 unidades e a cor for alterada para vermelho, o objeto “meuCirculo” estará em diferentes estados em momentos diferentes durante a execução do programa.

Portanto, os atributos de uma classe são fundamentais para definir e representar o estado de um objeto, tornando possível armazenar e gerenciar informações específicas de cada instância da classe criada.

- Alternativa verdadeira, pois em Programação Orientada a Objetos (POO), os métodos são análogos a funções, pois eles representam blocos de código que contém instruções para realizar operações específicas. A principal diferença é que os métodos estão associados a uma classe e operam em objetos dessa classe, enquanto funções geralmente são independentes e podem ser chamadas sem estar associados a um objeto específico.

Em TypeScript, uma linguagem que adiciona recursos de tipagem estática ao JavaScript, é possível definir métodos tanto dentro quanto fora da classe, assim como em C++.

Dentro da classe, os métodos são declarados como parte da definição da classe e têm acesso aos atributos e outros métodos da mesma classe. Esses métodos são geralmente usados para realizar operações específicas nos atributos do objeto.

Por exemplo, em TypeScript:

```
class Circulo {
   raio: number;
   cor: string;

      constructor (raio: number, cor: string) {
      this.raio = raio;
      this.cor = cor;
     }
}

calcularArea(): number {
    return Math.PI * this.raio * this.raio;
}
```

Nesse exemplo, a classe “Circulo” possui um método chamado “calcularArea()”, que calcula a área do círculo com base no atributo “raio” do objeto.

Por outro lado, também é possível definir métodos fora da classe. Esses métodos são independentes da classe, mas ainda podem ser usados para operar em objetos dessa classe, desde que recebam uma instância do objeto como argumento.

Por exemplo, em TypeScript:

```
        class Circulo {
            raio: number;
            cor: string;

           constructor (raio: number, cor: string) {
            this.raio = raio;
            this.cor = cor;
            }
}
```

```
function calcularAreaDoCirculo (circulo : Circulo) : number {
    return Math.PI * circulo.raio * circulo.raio;
}
```

Nesse exemplo, temos um método chamado “calcularAreaDoCirculo()”, que recebe uma instância de “Circulo” como argumento e calcula a área do círculo com base no atributo “raio” do objeto passado.

Em ambos os casos, esses métodos são usados para realizar operações específicas relacionadas a objetos de uma classe “Circulo”, mas sua definição e utilização podem variar dependendo do contexto do código.

- Alternativa falsa, pois em TypeScript; uma das principais vantagens é a adição de tipagem estática do JavaScript, permitindo especificar os tipos de variáveis, parâmetros de função, valores de retorno e até mesmo atributos e métodos de classes. No entanto, é possível ignorar o uso de tipos e escrever o código de forma mais similar ao JavaScript padrão.

Entretanto, quando você ignora o uso de tipos em TypeScript, o compilador não fará verificações de tipo estático no seu código, e você perde um dos principais benefícios que a linguagem oferece em termos de segurança, detecção de erros em tempo de compilação e autocompletar inteligente em ambientes de desenvolvimento.

Para ignorar o uso de tipos em TypeScript, você pode:

1) Não especificar tipos para variáveis e parâmetros de função;
2) Não especificar tipos para atributos e métodos de classes;
3) Não usar interfaces e tipos personalizados para definir formatos de dados;

Lembrando que, embora seja possível ignorar o uso de tipos em TypeScript, é recomendado aproveitar o poder da tipagem estática para melhorar a qualidade do código, facilitar a manutenção e evitar erros comuns. A utilização adequada dos tipos torna o código mais robusto e ajuda a evitar comportamentos inesperados em tempos de execução. Portanto, é aconselhável aproveitar os benefícios da tipagem estática sempre que possível.

3. Teresina Equipamentos Hidráulicos, Empresa e Nome da Empresa.

Letra c: classe, objeto e atributo.

4. let conta : Conta = new Conta();

Pela análise da questão, o let é o indicador de variável, conta (minúsculo) é minha classe, Conta (maiúsculo) é meu objeto, new é meu constructor e Conta() é o meu operador de instanciação.

(3) **Constructor**; (1) **Classe**;

(4) **Operador de Instanciação**; (2) **Objeto**;

5. Demonstrada no VS Code, em Linguagem TypeScript;

6. Demonstrada no Code Blocks, em Linguagem C++, a partir do exercício anterior;

7. Demonstrada no VS Code, em Linguagem TypeScript, a partir do exercício anterior;

8. Demonstrada no VS Code, em Linguagem TypeScript, a partir do exercício anterior.

Fim da atividade!