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

Aluno: Vinícius Gomes Araújo Costa

Disciplina: Programação Orientada a Objetos (POO)

Professor: Ely Miranda

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

**Semestre: 2023.2**

**Exercício 01**

1. **Objetos** são instâncias de classes. Classes e objetos são conceitos 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.

As classes também especificam 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”. Já 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.

Entretanto, 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.

**Exemplificando**: 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.

**E*m 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;

}

1. 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. 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.

No exemplo da 1ª questão, 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.

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.

|  |  |
| --- | --- |
| **Atributo** | **Sistema em que é**  **importante** |
| **Peso** | Supermercado, frigorífico, academia; |
| **Tipo de CNH** | DETRAN, STRANS, um banco de dados para buscar empregos como habilidade; |
| **Tipo sanguíneo** | Hospitais e clínicas, laboratórios; |
| **Habilidade destra** | Escolas e instituições de ensino infantil (creches), arremesso de cargas em esportes olímpicos; |
| **Percentual de gordura** | Na Biologia, pode ser identificado no DNA/RNA, alimentação Fast Food e de alimentação saudável, como Mac Donald’s e Subway, respectivamente; |
| **Saldo em conta** | Bancos, lotéricas, transações financeiras que mecham com dados tangíveis; |
| **Etnia** | Instituto Brasileiro de Geografia e Estatística (IBGE), Sistema de Seleção Unificado (SISU), Exame Nacional do Ensino Médio (ENEM), Sistema Unificado de Administração Pública (SUAP-EDU), etc... |

Obs: **ABSTRAÇÃO –** Outrossim, 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;

Priorizando assim, o que vale a pena considerar nos **Sistemas de Informação** referenciados acima na tabela é que cada atributo possui seu contexto/sistema aonde se encaixa perfeitamente o software usado para cada modelagem, sua respectiva manutenção, correção, princípios de maturidade e se realmente possui uma **solidez OO**, **Orientada a Objetos**.

1. Justificando as letras **a** e **b** da 4ª questão, um atributo interno pessoa pode ser **sim** representado pelo titular da conta se o programador que utiliza TypeScript criar uma nova classe “ContaBancaria”, a partir dos métodos ou atributos chamados no exemplo citado no VS Code: “idConta”, “saldo” e “titular”. Os quais são separados/individualizados pela classe “Pessoa”, que também possui primeiramente os métodos ou atributos chamados no VS Code como: “nomeCliente” e “cpf”.

Com isso, os objetos que são instanciados pelas suas classes **“Pessoa”** e **“Conta”** podem possuir **sim** mais de uma conta também como atributo. Neste 2º caso, para realizar operações na minha conta bancária, necessitaria de mais funcionalidades como “sacar()”, “depositar()”, “consultarSaldo()”, entre outras. Exemplo mais atual: o **uso de várias contas PIX** para o mesmo titular da conta.

*Elemento de programação estruturada que mais representa o conjunto de contas:*

**Uma possível abstração ou simplificação para uma agência bancária seria um “array”/conjunto de contas** cada uma com:

* Número: inteiro;
* CPF do titular : string;
* Saldo : real;
* Operações/funções de crédito e débito;

**Em C, o array teria “structs” com os dados** e haveria funções implementando as operações.

Um paradigma de programação também ajuda **abstrair problemas**:

* Simplifica elementos do mundo real;
* Transforma-os em analogias/peças aplicáveis a Algoritmos e soluções.

1. Notas, Código da Matrícula, CPF, Nome, E-mail, Código da Disciplina, Professor, etc...
2. Nome do jogo: **Ultra Street Fighter IV**;

Gênero: luta torneio de artes marciais mistas modo História.

Quem seriam meus objetos: meus personagens (avatares) movidos por uma sequência de movimentos em um ambiente virtual.

Atributos dos meus personagens ou métodos: socarDireita(), chutarDireita(), comboEspecial(), socarEsquerda(), chutarEsquerda(), comboEspecial(), pularDireita(), pularEsquerda(), etc.

Minhas classes: socar(), chutar(), poderesEspecial(), modoDeLuta(), tempoDeVida(), etc...

Funcionalidades dos meus objetos: possuem para cada luta combinada com dois ou mais avatares: uma cena de entrada (paisagem), saudação do personagem como causa de honra da luta e respeito ao companheiro, porcentagemEnergia(), escolhaRoupa(), nacionalidadePersonagem(), escolhaDaFala(), Xp(), poder(), help(), listaDeComandos(), modoUniplayer(), modoMultiplayer();

Stakeholders: crianças de 12 a 14 anos, entretenimento para jovens de 18 a 30 anos que curtem luta e estilos diferentes de nacionalidades e histórias marcantes. Público infanto-juvenil também.

1. Feita no VS Code.
2. Feita no VS Code.
3. Feita no VS Code.
4. Na 10ª questão:

Representar as classes das questões 8ª e 9ª no formato **UML** (Unified Modeling Language) ou Linguagem de Modelagem Unificada – o qual seria uma linguagem de notação padrão para uso em projetos de sistema. O **UML** é utilizado para construir, especificar, visualizar e documentar um software. Os diagramas da **UML** fornecem um **“desenho” do sistema** que se pretende desenvolver, centralizando nos diagramas um determinado conceito de fácil entendimento aos envolvidos no projeto.

**Diagramas de Classe**:

|  |  |
| --- | --- |
| **Classe Círculo** | **Objeto Área** |
| Nome: String; | Nome: terreno do Vinícius; |
| Ano: Number; | Ano: 2023; |
| Preço: Number; | Preço: R$ 80.000,00 |
| Raio : Number | Raio: 2metros quadrados; |
| Pi: Number; | Pi: 3.14 |

|  |  |
| --- | --- |
| **Classe Círculo** | **Objeto Perímetro** |
| Nome: String; | Nome: terreno do Vinícius; |
| Ano: Number; | Ano: 2023; |
| Preço: Number; | Preço: R$ 80.000,00 |
| Raio: Number; | Raio: 2 metros quadrados; |
| Pi: Number; | Pi: 3.14 |

|  |  |
| --- | --- |
| **Classe Situação Financeira** | **Objeto Saldo** |
| nomeBanco: String; | nomeBanco: vini Banking |
| valorCredito: Number; | valorCredito: 300; |
| valorDebito: Number; | valorDebito: 500; |
| saldo: Number; | saldo: -200; |

Fim da atividade!