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 III – Turma Especial

**Semestre: 2024.1**

**Exercício 01**

1. De acordo com o conteúdo ministrado em sala de aula pelo professor Ely, e pelo real entendimento do assunto, define-se que os **Objetos** são instâncias de classes e as **Classes** são instanciadaspelos objetos. E, em Programação Orientada a Objetos (POO), não teria como compreender estes conceitos sem entender primeiro o conceito de paradigma: um estilo, um modelo, um padrão a ser seguido a fim de focar uma maneira de pensar, abstraindo detalhes essenciais a resolver problemas, onde os **Objetos** são elementos de estudo neste paradigma Orientado a Objetos.

O professor da disciplina explana bem este conteúdo quando dar por exemplo que **Objetos** são elementos em arranjo, isto é, um conjunto de matrizes conhecidas como arrays bidimensionais. As matrizes são estruturas de dados que representam um conjunto de valores do mesmo tipo. Elas são acessíveis com um único nome e armazenadas contiguamente na memória. Cada variável de um vetor é individualizada através do uso de índices, no caso de POO, colchetes.

Com isso, quebrando problemas em pequenas partes vai diminuindo a complexidade dos mesmos em resolvê-los. Como dito anteriormente, a ideação de focar nos elementos primordiais do problema irá levar a desenvolvermos e a capacitar as linguagens de programação, no caso, TypeScript. Ademais, **classes** serão nossas variáveis de escopo em TypeScript: 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) ou métodos (comportamentos) os **objetos** criados a partir dela terão.

**Exemplos**: disponíveis **soma de matrizes** e **demonstração da área do círculo** no VS Code, em **Linguagens de Programação** **TypeScript e Java**.

1. No paradigma **Programação Orientado a Objetos**, parte do princípio de que o mundo é formado por elementos (**objetos**) que interagem ente si. E tudo relacionado ao paradigma são objetos. Portanto, quem possui características/estado são as variáveis conhecidas como **atributos** e quem possui comportamentos são as funções, também denominadas de **métodos**. Os métodos alteram os valores dos atributos.

Além disso, outra definição para **atributos e métodos: 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 às 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.

**Exemplificação da questão**: disponível do VS Code.

Detalhando mais um pouco sobre o exemplo dado da 2ª questão, eu elaborei, em TypeScript, um exemplo de objeto que possui atributos e métodos (notação livre) para cálculo de área de um círculo.

Com isso, o código disponível no VS Code mostra um exemplo simples, em TypeScript, de um objeto que representa um círculo, contendo atributos (raio) e métodos para se calcular a área.

Neste exemplo, a classe ‘Circulo’ tem um atributo privado ‘raio’ e métodos públicos para obter, definir o raio e calcular a área do círculo. O método ‘calcularArea’ utiliza a fórmula matemática pi \* raio ^ 2 para calcular a área do círculo.

O código do exemplo cria uma instância do objeto ‘Circulo’ com um raio de 5 unidades e imprime o raio e a área calculada. Lembre-se de que TypeScript é superset de JavaScript, então compila-se este código TypeScript para JavaScript para ser executado em ambientes compatíveis.

1. Em diversas situações: sistema bancário, rede de esportes olímpicos, escola de modelos, joalheria, algumas informações (atributos) são de crucial importância para um determinado objeto. Então, a abstração visa focar no essencial e esquecer os detalhes irrelevantes para cada sistema. Com isso, atentar à tabela abaixo:

|  |  |  |  |
| --- | --- | --- | --- |
| **Atributo** | **Sistema em que não é importante** | **Sistema em que é mais ou menos importante** | **Sistema em que é importante** |
| **Peso** | Sistema bancário, Igrejas, escolas ou universidades | Exame Nacional do Ensino Médio (ENEM), Logística, Parques Aquáticos, Sistemas de Elevação, como elevadores. | Supermercado, frigorífico, academia, hospitais, a ciência natural Física. |
| **Tipo de CNH** | Sistema de Banco de Dados aonde a habilidade de dirigir não convém: escolinhas de futebol, esportes olímpicos, exército. | Baladas e festas, aniversários, passaporte para viagens. Irá depender da situação conveniente. | DETRAN, STRANS, Sistema Nacional de Empregos e Estágios (SINE), blitz de trânsito para Lei Seca. |
| **Tipo Sanguíneo** | Censo do IBGE para qualidade/expectativa de vida, escolas bilíngues. | Escolha de pessoas com deficiência no Mercado de Trabalho, academias, manutenção de smartphones ou computadores. | Hospitais e clínicas, laboratórios, exército, HEMOPI, por exemplo, detecção de doenças, como câncer. |
| **Habilidade Destra** | Situações de fuga, como corrida, certas emergências para lotações em shows e/ou espaço aberto, teatro, comer pipoca e tomar refrigerante com uma mão só. | Resolver cálculos matemáticos com maior precisão, estímulo cerebral, potência de chute na perna direita com o futebol, barco a remo, ganho de massa magra em academias. | Escolas e instituições de ensino infantil (creches), educação familiar, Igreja, arremesso de cargas em esportes olímpicos. |
| **Percentual de gordura** | Na Biologia, pode ser trivial, como a simples detecção a olho nu de DNA/RNA em um ribossomo, entre outros casos. | Relacionamentos amorosos, herança, joalherias e escolas ou universidades. | Alimentação em Redes Fast Food, seleção de frutas, ou também em começo de dietas alimentares, percentual de gordura em exames de sangue nos hospitais e clínicas. |
| **Saldo em conta** | Bitcoins, pois são criptomoedas não reguladas pelo Bacen, golpes financeiros aplicados a agiotas, assaltos. | Sistemas de segurança em shoppings centers, portas lacradas com biometria em bancos de acesso a moedas circuladas. A depender da situação conveniente. | Bancos, lotéricas, transações financeiras que mecham com dados tangíveis. |
| **Etnia** | Discriminação racial, crimes hediondos, tortura e tráfico de armas, drogas e atos ilícitos designados pela cor da pele. | Em casos de raça animal, Petshops são a melhor escolha, IBAMA – proteção de tartarugas marinhas, animas em extinção. Para seres humanos, cotas para universidades públicas. | 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.. |

1. Considerando dois objetos **Pessoa** e **Conta**:
2. **Sim**. De acordo com os ensinamentos do professor Ely em sala de aula, uma instituição bancária (tradicional) – em um banco, possui um array de clientes vinculados a n contas ou uma conta pode unicamente está vinculada a um e somente um cliente. Outro exemplo: em uma sala de aula, podem existir n mesas e em cada mesa m computadores. Não sendo levado em consideração, no caso do engenheiro/arquiteto as dimensões da sala, muito menos o acabamento de dimensões n x m dadas em uso.

Portanto, como objetos são instâncias das classes, eu (Vinícius) irei trabalhar com **TypeScript**, com a seguinte orientação: pessoas com contas em um banco possuem como atributos: nome, CPF, saldo, etc. 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.

1. Uma possível **abstração ou simplificação** para uma **agência bancária** possuir mais de uma conta como atributo seria interessante no sentido inverso que 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 a **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, Turma.
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. Disponível e demonstrada no VS Code.
2. Disponível e demonstrada no VS Code.
3. Disponível e demonstrada 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. Exemplo de ferramenta online: **Lucid Chart**.

**Diagramas de Classe**:

|  |  |
| --- | --- |
| **Classe Círculo** | **Objeto Área** |
| Nome: String; | Nome: terreno do Vinícius; |
| Ano: Number; | Ano: 2024; |
| Preço: Number; | Preço: R$ 80.000,00 |
| Raio : Number | Raio: 5 metros quadrados; |
| Pi: Number; | Pi: 3.1415; |

|  |  |
| --- | --- |
| **Classe Círculo** | **Objeto Perímetro** |
| Nome: String; | Nome: terreno do Vinícius; |
| Ano: Number; | Ano: 2024; |
| Preço: Number; | Preço: R$ 80.000,00 |
| Raio: Number; | Raio: 5 metros quadrados; |
| Pi: Number; | Pi: 3.1415; |

|  |  |
| --- | --- |
| **Classe Situação Financeira** | **Objeto Saldo** |
| nomeBanco: String; | nomeBanco: vini Banking |
| valorCredito: Number; | valorCredito: 5000; |
| valorDebito: Number; | valorDebito: 3000; |
| saldo: Number; | saldo: R$2000; |

**Fim da atividade!**