

Correção da atividade 01, de POO, disponibilizado pelo professor Ely no Classroom e submetido no Git Hub por Vinícius Gomes.

1. Classes são modelos, moldes para objetos. Já os objetos são classes “vivas”, instâncias de uma classe.

Ex1: Classe Pessoa (id, nome, CPF, dataDeNascimento)

Objeto (1, 'Ely', '825...', 29/06/1979)

Objeto (2, 'Mateus', '4449...', 20/05/2020)

2. Atributos são as características, estado. São análogos às variáveis;

Métodos: são ações ou comportamentos dinâmicos. São análogos às funções;

Sempre que possível, pensar em objetos e classes de jogos.

Jogo de Tabuleiro – atributos : arrays de peças, arrays de posições preenchidas, time1, time2;

Métodos: mover peça, descartar peça, avaliar jogada, encerrar partida.

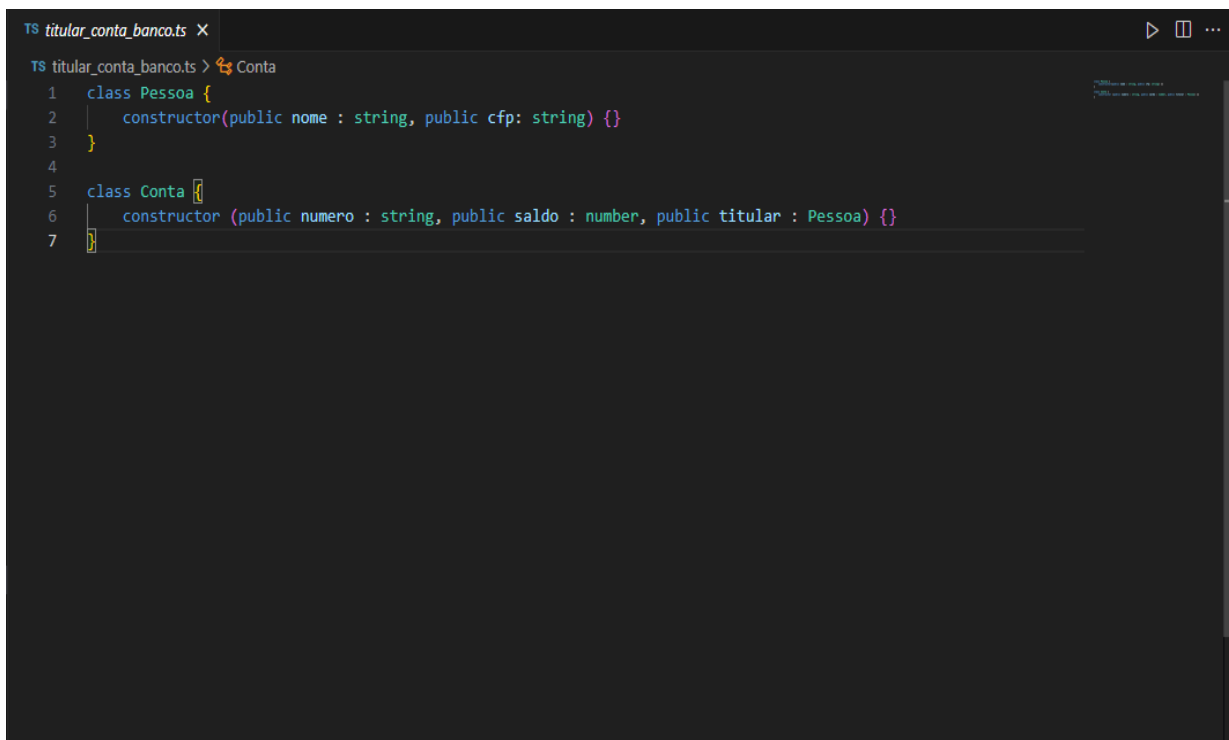
3. A abstração visa focar no que é importante para um sistema. Com isso, o atributo de uma pessoa pode ser tão importante ou não dependendo do contexto do sistema. Na tabela abaixo, eis uma prática sobre uma enumeração de contextos/sistemas distintos em que os atributos abaixo seriam importantes:

Atributo	Sistema em que não é importante	Sistema em que é mais ou menos importante	Sistema em que é importante
Peso	Biblioteca, Aplicação de provas, Bancário.	Estoque.	Academia, Nutrição, Aviação.
Tipo de CNH	Acadêmico, Escolinha de Futebol, Rede Social.	Exército, Hospitalar, Cadastro Civil, Concurso Público.	Detran, Locadora de Veículos, Empresa de ônibus.
Tipo Sanguíneo	Bancário, Estoque, SETUT.	Concurso Público, Academia.	HEMOPI, Hospitalar, Exército.
Habilidade Destra	Acadêmico.	Escola de Música.	Controle Desportivo(Tênis, Futebol).
Percentual de gordura	Bancário, Rede Social, Escolar.	Teste de Aptidão Física.	Nutrição, Academia, Hospitalar.

Saldo em Conta	Acadêmico, HEMOPI.	Rede Social.	Jogos, Bancário, Vendas.
Etnia	IFood, Entregas, Comércio Eletrônico.	Rede Social.	Censo Demográfico, Processos Seletivos, Hospitalar.

4. **a) Sim.** Em Programação Orientada a Objetos(POO), a decisão de incluir um objeto como atributo de outro depende do design do sistema e dos requisitos específicos. No contexto de um Sistema Bancário, a relação entre “Conta” e “Pessoa” é um exemplo clássico de associação entre classes. Vou discutir as vantagens e desvantagens dessa abordagem, utilizando TypeScript para ilustrar o conceito.

Abordagem com um Objeto “Pessoa” como atributo de “Conta”:



```

TS titular_conta_banco.ts X
TS titular_conta_banco.ts > Conta
1 class Pessoa {
2   constructor(public nome : string, public cpf: string) {}
3 }
4
5 class Conta {
6   constructor (public numero : string, public saldo : number, public titular : Pessoa) {}
7 }

```

Vantagens:

- **Associação Clara:** A inclusão da pessoa como atributo de conta reflete a relação natural entre elas no mundo real, tornando o código mais intuitivo.
- **Acesso Direto:** Permite acesso direto às informações do titular da conta a partir da instância da conta.

Desvantagens:

- **Acoplamento Forte:** As classes estão fortemente acopladas, o que pode tornar a manutenção mais desafiadora se houver mudanças nos requisitos.
- **Inflexibilidade:** Pode ser difícil lidar com casos em que uma conta precisa ser associada a mais de uma pessoa (conta conjunta).

Abordagem com referência a “Pessoa” (sem atributo interno):

```
TS titular_conta_banco_Pessoa.ts X
TS titular_conta_banco_Pessoa.ts > ...
1 class Pessoa {
2   constructor (public nome: string, public cpf : string) {}
3 }
4
5 class Conta {
6   constructor (public numero : string, public saldo: number, public titularCpf : string) {}
7
8   obterTitular(pessoas : Pessoa[]) : Pessoa | undefined {
9     return pessoas.find(pessoa => pessoa.cpf === this.titularCpf)
10  }
11 }
12
13
```

Vantagens:

- **Desacoplamento:** As classes são menos dependentes uma da outra, facilitando a manutenção e a evolução do sistema.
- **Flexibilidade:** Pode lidar com casos em que uma conta precisa ser associada a várias pessoas sem modificar a estrutura da classe “Conta”.

Desvantagens:

- **Acesso indireto:** Para obter informações sobre o titular, é necessário buscar a referência na lista de pessoas.
- **Maior complexidade:** Pode exigir lógica adicional para garantir consistência e integridade dos dados.

Conclusão

A escolha entre essas abordagens depende dos requisitos específicos do sistema. Se a relação entre “Conta” e “Pessoa” for fundamental e pouco sujeita a alterações, a primeira abordagem pode ser mais direta. Se a flexibilidade e o desacoplamento forem prioridades, a segunda abordagem pode ser mais adequada. Em muitos casos, uma abordagem intermediária, como o uso de interfaces ou classes abstratas, pode oferecer o melhor dos dois mundos.

b. Sim. Um array.

Teoria:

Na Programação Orientada a Objetos (POO), a relação entre “Pessoa” e “Conta” pode ser modelada de forma a permitir que uma pessoa possua mais de uma conta. Uma abordagem adequada para representar esse relacionamento é através de uma associação de “um-para-muitos” (one-to-many).

Um elemento fundamental para representar essa relação é uma lista (ou qualquer estrutura de dados adequada) que armazena as várias contas associadas a uma

peessoa. Essa lista seria um atributo da classe “Pessoa”, refletindo a capacidade de uma pessoa em possuir múltiplas contas.

Código em TypeScript:

```
TS inverso_titular_conta.ts X
TS inverso_titular_conta.ts > Pessoa > adicionarConta
1  class Conta {
2      constructor (public numero : string, public saldo : number) {}
3  }
4
5  class Pessoa {
6      contas : Conta[] = [];
7
8      constructor (public nome : string, public cpf : string) {}
9
10     adicionarConta(conta : Conta): void {
11         this.contas.push(conta);
12     }
13
14     // Outros métodos e lógicas relacionados à Pessoa...
15 }
```

Neste exemplo, a classe “Pessoa” possui um array de contas (`contas: Contas[]`), permitindo que uma pessoa tenha várias contas associadas a ela. O método `adicionarConta` é utilizado para adicionar uma nova conta à lista de contas da pessoa.

Essa abordagem oferece flexibilidade para lidar com situações em que uma pessoa pode ter mais de uma conta, mantendo a estrutura de classes clara e permitindo operações relacionadas a contas diretamente a partir de uma instância de pessoa.

Concluindo, o uso de uma lista (ou similar) é um elemento crucial para representar a relação “um-para-muitos” entre “Pessoa” e “Conta” na Programação Orientada a Objetos (POO).

5. Notas, Código da Matrícula, CPF, Nome, E-mail, Código da Disciplina, Professor, Turma.

6. 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.

7. Disponível e demonstrada no VS Code.

8. Disponível e demonstrada no VS Code.

9. Disponível e demonstrada no VS Code.

10. 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!