Estudo Dirigido 003 – Programação Orientada a Objetos

Introdução

O objetivo destes exercícios é **construir o raciocínio da Programação Orientada a Objetos (POO)**.
Você já sabe modularizar código com funções. Agora, precisa entender:

- **Funções** isoladas resolvem tarefas.
- **Classes e objetos** organizam os dados junto com seus comportamentos.

A POO tem como intenção **representar realidades materiais e lógicas** dentro do código.
Quando programamos com classes, traduzimos entidades do mundo real ou do negócio (como usuário, produto, pedido, pagamento) em estruturas que o sistema consegue manipular.
Cada classe define características (atributos) e comportamentos (métodos), aproximando o raciocínio técnico da forma como o negócio realmente funciona.

Por isso, trabalhar com POO está diretamente ligado ao **levantamento de requisitos**: ao ouvir uma descrição de processo ou regra de negócio, geralmente já pensamos em quais objetos existem, quais informações carregam e quais ações podem executar.
Assim, POO não é apenas uma técnica de programação, mas também um modo de alinhar código e realidade.

A lista a seguir ajuda a perceber, na prática, como esse modelo orientado a objetos se diferencia do uso de funções soltas e como ele serve para estruturar sistemas mais próximos do mundo real.

Parte 1 – Funções vs. POO

Exercício 1 – Funções soltas

Implemente um pequeno sistema para registrar produtos (nome, quantidade, preço) usando **apenas funções**:

- `adicionar_estoque(produto, qtd)`
- `remover_estoque(produto, qtd)`
- `valor_total(produto)`

Aqui o objetivo é lembrar como resolveríamos o problema sem POO. Você deve sentir a necessidade de passar os dados de produto de um lado para o outro o tempo todo.

Exercício 2 – Refatorando com POO

Agora, reescreva o mesmo sistema criando a classe `Produto`.

- Atributos: nome, quantidade, preço.
- Métodos: `adicionar`, `remover`, `valor_total`.

Aqui, os dados do produto e as operações ficam encapsulados na mesma estrutura. O objeto já sabe calcular seu valor total, não precisa mais de funções externas. Esse é o primeiro ganho da

POO.

Parte 2 – Traduzindo Realidades em POO

Exercício 3 – Cadastro de Usuários

Crie a classe `Usuario` com os atributos: `nome`, `email`, `senha`.

- Método: `autenticar(email, senha)` que retorna se o login é válido.
- Teste criando dois usuários e simulando logins corretos e incorretos.

Este exercício mostra que cada instância guarda seus próprios dados. Diferentes usuários podem existir no mesmo sistema, cada um com suas credenciais.

Exercício 4 – Sistema de Pedidos (E-commerce)

Crie as classes:

- `Produto` (nome, preço).
- `Pedido` (lista de produtos).

O pedido deve permitir:

- Adicionar produto.
- Listar produtos.
- Calcular valor total.

Aqui você começa a relacionar classes diferentes. `Pedido` usa `Produto`. Essa composição é um dos pontos fortes da POO: objetos trabalhando juntos.

Parte 3 – Situações Reais

Exercício 5 – Gestão de Usuários de Jogo

Crie a classe `Jogador`:

- Atributos: `nome`, `saldo`, `itens`.
- Métodos: `adicionar_saldo`, `comprar_item(item, preco)`.

Esse exercício mostra como as regras de negócio ficam dentro da classe. Não basta ter saldo: só o próprio jogador pode modificar o seu inventário.

Exercício 6 – Sistema de Pagamentos

Crie a classe `Pagamento`:

- Atributos: `usuario`, `valor`, `status`.
- Método: `processar()`, que adiciona saldo ao usuário.

Aqui aparece a interação entre classes: `Pagamento` atua diretamente sobre `Usuario`. O método conecta entidades distintas, simulando o fluxo de um sistema real.

Parte 4 – Desafios de Fixação

Exercício 7 – Sistema de Biblioteca

Crie as classes:

- `Livro` (título, autor, disponível).
- `Biblioteca` (lista de livros).

A biblioteca deve permitir:

- Adicionar livros.
- Emprestar livro (marcar como indisponível).
- Listar disponíveis.

Esse exercício mostra como pensar em regras de negócio específicas: um livro não pode ser emprestado duas vezes sem ser devolvido.

Exercício 8 – Gestão de Funcionários

Crie a classe `Funcionario`:

- Atributos: `nome`, `cargo`, `salario`.
- Método: `aumentar_salario(percentual)`.

Aqui você reforça a ideia de encapsulamento: só a classe sabe calcular corretamente o novo salário. O cálculo não fica espalhado pelo código.

Exercício 9 – POO e Levantamento de Requisitos (Reflexão)

Reúnam-se em grupos e revistem as ideias desenvolvidas na metodologia **5W2H** para analisar um problema simples do cotidiano ou um caso sugerido por vocês no inicio do nosso curso.

- A partir do levantamento inicial, discutam:
  - Quais seriam as principais **entidades** do sistema?
  - Quais **atributos** essas entidades precisariam ter?
  - Quais seriam os **comportamentos (métodos)** mais importantes?

Objetivo:
Perceber que a Programação Orientada a Objetos é, antes de tudo, um exercício de **modelagem de realidade**.
O código nasce depois de uma boa análise de requisitos e nada mais justo do que faz com algo mais pessoal de vocês.

Encerramento

Ao concluir esses exercícios, você terá visto:

- Diferença entre **funções soltas** e **métodos encapsulados**.
- Como classes representam **entidades reais** (usuário, produto, pedido, jogador, pagamento, livro, funcionário).
- Como objetos interagem e se relacionam, traduzindo **sistemas completos** em código.