Entender como organizar o código entre a classe **Produto** e a classe **AppGestao** envolve pensar nos **papéis e responsabilidades** de cada uma das classes. Esse conceito é fundamental na **Programação Orientada a Objetos (POO)**, onde a ideia principal é organizar o código em "objetos" que representam entidades reais ou abstratas, com cada um cuidando de sua própria responsabilidade.

Como pensar sobre a separação?

Aqui estão algumas perguntas que você pode se fazer ao organizar o código:

1. **O que a classe `Produto` representa?**
    - **Resposta**: Um **produto individual**, com seus atributos específicos (nome e preço). A classe `Produto` é responsável por armazenar e fornecer informações sobre um único produto.
2. **O que a classe `AppGestao` representa?**
    - **Resposta**: Uma **aplicação de gestão de produtos**. A responsabilidade da `AppGestao` é **gerenciar** os produtos, ou seja, adicionar produtos à lista, calcular o total de preços, e listar os produtos.

Com base nisso, a separação natural do código surge:

1. Classe `Produto` - Responsável por um Produto Individual

A classe `Produto` deve **somente** lidar com as informações de um **produto individual**. Isso significa que ela armazena os **atributos** do produto (nome e preço) e fornece métodos que permitem acessar essas informações.

Aqui está o que deve ir na classe `Produto`:

- **Atributos**: Quais são as propriedades de um produto? Nesse caso, nome e preço.
- **Construtor**: Um método que cria uma instância de `Produto` e inicializa seus atributos.
- **Getters**: Métodos para acessar os valores dos atributos (`getNome()` e `getPreco()`).

Exemplo da Classe `Produto`:

java Copiar código

```java
public class Produto {
    private String nome;
    private float preco;

    // Construtor para inicializar nome e preco
    public Produto(String nome, float preco) {
        this.nome = nome;
        this.preco = preco;
    }

    // Métodos para acessar os atributos
    public String getNome() {
        return nome;
    }

    public float getPreco() {
        return preco;
    }
}
```

Por que essas responsabilidades estão aqui?

- A **classe Produto** é **somente** responsável por armazenar os dados de um produto e fornecer métodos para acessar esses dados.
- Ela **não deve** saber como produtos são gerenciados em uma lista, nem somar preços. Essas tarefas pertencem à classe que faz a **gestão** dos produtos, ou seja, `AppGestao`.

2. Classe `AppGestao` - Responsável pela Gestão dos Produtos

A classe `AppGestao` é responsável por **gerenciar múltiplos produtos**. Isso inclui:

- **Adicionar** novos produtos à lista.
- **Listar** todos os produtos.
- **Somar** os preços de todos os produtos.

Aqui estão as responsabilidades da classe `AppGestao`:

- **Armazenar a lista de produtos**: Você precisa de uma estrutura de dados para armazenar vários produtos, como um `ArrayList`.
- **Método para inserir produto**: Adicionar um novo produto à lista.
- **Método para listar produtos**: Mostrar os detalhes (nome e preço) de todos os produtos na lista.
- **Método para somar preços**: Calcular e retornar o valor total de todos os preços.

Exemplo da Classe `AppGestao`:

java Copiar código

```java
import java.util.ArrayList;

public class AppGestao {
    // Lista de produtos
    private ArrayList<Produto> produtos;

    // Construtor que inicializa a lista
    public AppGestao() {
        produtos = new ArrayList<>();
        System.out.println("Bem-vindo/a à Aplicação de Gestão!");
    }

    // Método para inserir um novo produto
    public void inserirProduto(String nome, float preco) {
        Produto novoProduto = new Produto(nome, preco);
        produtos.add(novoProduto);
    }

    // Método para listar todos os produtos
```

```java
    }

    // Método para listar todos os produtos
    public void listarProdutos() {
        if (produtos.isEmpty()) {
            System.out.println("Não existem produtos!");
        } else {
            for (int i = 0; i < produtos.size(); i++) {
                Produto produto = produtos.get(i);
                System.out.println("Produto: " + produto.getNome() + ", Preço: " +
            }
        }
    }

    // Método para calcular o total dos preços dos produtos
    public void totalPrecos() {
        if (produtos.isEmpty()) {
            System.out.println("Não existem produtos!");
        } else {
            float total = 0;
            for (int i = 0; i < produtos.size(); i++) {
                total += produtos.get(i).getPreco();
            }
            System.out.println("Total dos preços: " + total);
```

Copiar código

Por que essas responsabilidades estão aqui?

- A **classe AppGestao não deve se preocupar** com os detalhes internos de cada produto, como armazenar os dados de um único produto. Em vez disso, ela se concentra em **como gerenciar vários produtos**.
- A lógica de **adicionar produtos**, **listar produtos** e **somar preços** pertence à `AppGestao`, porque ela lida com a **gestão de múltiplos objetos**.

Pensamento Orientado a Objetos

A chave para entender como separar o código em classes é o princípio da **responsabilidade única**. Cada classe deve ter **uma única responsabilidade bem definida**.

1. **Classe Produto**: Sua única responsabilidade é representar um produto individual.
    - Deve lidar apenas com os dados e comportamentos de um produto, como seu nome e preço.
2. **Classe AppGestao**: Sua única responsabilidade é gerenciar uma coleção de produtos.
    - Deve lidar com a inserção, listagem e manipulação de vários produtos.

Dicas para Novos Projetos:

- **Responsabilidade Única**: Cada classe deve fazer **uma coisa**. `Produto` armazena informações de um produto, `AppGestao` gerencia produtos.
- **Separação de Funções**: Se algo parece estar fazendo mais do que deveria (por exemplo, se `Produto` estivesse calculando o total de todos os produtos), provavelmente essa responsabilidade está na classe errada.
- **Manutenção e Evolução**: Separar bem as responsabilidades torna o código mais fácil de manter e expandir. Se você precisar adicionar novos tipos de produtos ou funcionalidades de gestão, a separação clara facilita essas alterações.

Essa abordagem o ajudará a organizar melhor seus projetos à medida que você avança no estudo de Java e da Programação Orientada a Objetos!