🧠 Revisão Completa: POO em Java – Polimorfismo, Abstração, Interface e Listas

🔍 Introdução à Programação Orientada a Objetos (POO)

A **POO** é um paradigma que organiza o código com base em **objetos**, que são instâncias de **classes**. Cada objeto possui **atributos** (dados) e **métodos** (ações). A POO aproxima a lógica do código à maneira como percebemos o mundo real, facilitando a modelagem de problemas complexos.

🔁 POLIMORFISMO

Polimorfismo significa "muitas formas". Na programação, ele permite que **o mesmo método ou nome tenha comportamentos diferentes** dependendo do contexto. Em Java, há dois tipos principais:

1. Polimorfismo Estático (Sobrecarga - *overload*)

- Ocorre **dentro da mesma classe**, quando se define **vários métodos com o mesmo nome**, mas com **diferentes parâmetros** (tipo ou quantidade).
- O compilador escolhe qual método usar com base na **assinatura**.

```
public void imprimir(String texto) { ... }
public void imprimir(int numero) { ... }
```

2. Polimorfismo Dinâmico (Sobrescrita - *override*)

- Acontece em **relações de herança**, quando uma **subclasse redefine um método herdado da superclasse** com a **mesma assinatura**.
- Permite usar uma **referência genérica** (como `Animal`) para apontar para vários tipos de objetos (`Cachorro`, `Gato`) e invocar comportamentos específicos.

```
class Animal {
    void emitirSom() {
        System.out.println("Som genérico");
    }
}

class Cachorro extends Animal {
```

```
    @Override
    void emitirSom() {
        System.out.println("Latido");
    }
}
```

Vantagem do Polimorfismo

- Proporciona **flexibilidade** e **reutilização de código**.
- Permite escrever **códigos genéricos**, como métodos que recebem `Animal` e funcionam com qualquer subclasse.

🔷 ABSTRAÇÃO E CLASSES ABSTRATAS

Abstração é o processo de **esconder os detalhes de implementação** e mostrar apenas a funcionalidade essencial. Em Java, isso é realizado com **classes abstratas**.

Características de Classes Abstratas:

- **Não podem ser instanciadas** diretamente.
- Podem ter métodos **com ou sem implementação**.
- Servem como **modelo base** para outras classes.

Uso comum:

Imagine uma classe `Animal`. Você **não cria um "Animal genérico"**, mas sim um `Cachorro`, `Gato`, etc. A classe `Animal` serve como um **padrão obrigatório**, exigindo que subclasses implementem comportamentos importantes.

```
abstract class Animal {
    abstract void emitirSom(); // método obrigatório
}
```

As subclasses devem implementar esse método:

```
class Gato extends Animal {
    void emitirSom() {
        System.out.println("Miau");
```

```
    }
}
```

Vantagens da Abstração:

- **Força a padronização** entre subclasses.
- Reduz código duplicado.
- Ajuda a separar **o que** um objeto faz de **como** ele faz.

🧩 INTERFACES

Uma **interface** define um **contrato**: um conjunto de métodos que uma classe **deve implementar**. Ao contrário de classes abstratas:

Características:

- **Todos os métodos são públicos e abstratos por padrão.**
- **Não pode conter atributos mutáveis nem construtores.**
- Permite que classes **sem relação direta** compartilhem comportamentos.

```
interface Voador {
    void voar();
}

class Passaro implements Voador {
    public void voar() {
        System.out.println("Passaro voando");
    }
}

class Aviao implements Voador {
    public void voar() {
        System.out.println("Avião voando");
    }
}
```

Diferença entre Interface e Classe Abstrata:

Classe Abstrata	Interface
Pode ter atributos e métodos	Apenas métodos (e constantes)
Pode ter implementação parcial	Não tem implementação
Suporta herança simples	Pode implementar várias interfaces

Quando usar interface?

- Quando você quer **padronizar comportamento entre classes diferentes** que **não têm herança em comum**.
- Exemplo: `SalvarEmArquivo`, `Voador`, `Calculavel`.

📚 LISTAS DE OBJETOS (ArrayList)

As **listas (ArrayList)** são estruturas que armazenam **coleções de objetos** de forma dinâmica — ou seja, **sem saber previamente a quantidade** de elementos.

```
ArrayList<Aluno> lista = new ArrayList<>();
```

Principais métodos:

- `add(obj)` – adiciona ao final
- `get(i)` – acessa pelo índice
- `remove(i ou obj)` – remove pelo índice ou objeto
- `size()` – retorna a quantidade atual

Exemplo de uso:

```
ArrayList<Animal> animais = new ArrayList<>();
animais.add(new Gato());
animais.add(new Cachorro());
```

```
for (Animal a : animais) {
    a.emitirSom(); // polimorfismo em ação
}
```

Vantagens:

- Armazena e organiza grandes quantidades de dados.
- Facilita a **reutilização de código** com métodos que recebem ou retornam listas.
- Funciona perfeitamente com **polimorfismo**: podemos ter uma lista de `Animal`, mas armazenar `Cachorro`, `Gato`, etc.

🎯 DICAS FINAIS PARA A PROVA

- Saiba **a diferença entre sobrecarga e sobrescrita**: a primeira acontece dentro da mesma classe; a segunda entre superclasse e subclasse.
- **Listas + polimorfismo** aparecem muito em questões práticas: domine os métodos do `ArrayList`.
- **Classe abstrata = modelo com métodos obrigatórios.**
- **Interface = contrato entre classes diferentes.**
- Use **getters e setters** para acessar atributos privados (encapsulamento).
- Esteja pronto para analisar exemplos que combinem herança, listas, interfaces e polimorfismo.