🧠 Resumo Geral – Programação Orientada a Objetos (Java)

🔹 Conceitos Fundamentais da POO

- **Classe**: molde/estrutura que define atributos e métodos de um objeto.
- **Objeto**: instância de uma classe.
- **Atributo**: características do objeto (ex: nome, idade).
- **Método**: comportamentos ou ações que um objeto pode realizar.
- **Construtor**: método especial com o mesmo nome da classe, usado para inicializar objetos. Pode ter sobrecarga (vários com parâmetros diferentes).

🔐 Encapsulamento

- Oculta os atributos da classe e permite acesso controlado através de:
    - **Getters**: métodos para obter valores (`getNome()`).
    - **Setters**: métodos para definir valores (`setNome("João")`).
- **Modificadores de acesso**:
    - `private`: visível apenas na própria classe.
    - `public`: visível de qualquer lugar.
    - `protected`: visível no mesmo pacote ou por subclasses.
    - (default): visível apenas no pacote.

🔁 Polimorfismo

1. Polimorfismo por sobrescrita (dinâmico)

- Subclasse redefine um método da superclasse com a mesma assinatura.

java
CopiarEditar
```
class Animal {
    void fazerSom() { System.out.println("Som genérico"); }
}
class Cachorro extends Animal {
    @Override
    void fazerSom() { System.out.println("Au au!"); }
}
```

2. Polimorfismo por sobrecarga (estático)

- Mesma classe, métodos com mesmo nome e parâmetros diferentes.

java
CopiarEditar
```
void emitirSom() {}
void emitirSom(String humor) {}
```

🧱 Herança

- Permite que uma classe (subclasse) herde atributos e métodos de outra (superclasse).
- Usada para reaproveitamento de código e especialização de comportamento.

java
CopiarEditar
```
class Pessoa {
    String nome;
}
class Aluno extends Pessoa {
    int matricula;
}
```

🔶 CLASSES ABSTRATAS (FOCO)

➢ O que é uma classe abstrata?

- Uma **classe abstrata não pode ser instanciada** diretamente.
- Serve como um modelo base para outras classes herdarem.
- Pode conter métodos **com ou sem implementação**.
- Declaração:

java
CopiarEditar
```
public abstract class Animal {
    public abstract void emitirSom(); // método obrigatório a ser
implementado
}
```

➢ Quando usar?

- Quando há uma estrutura genérica comum para várias classes, mas partes do comportamento devem ser definidas individualmente pelas subclasses.

➢ Exemplo:

java
CopiarEditar
```
abstract class Animal {
    protected String especie;
    public abstract void emitirSom();
}

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

🔷 INTERFACES (relacionadas às classes abstratas)

- Define **um contrato** que deve ser seguido por quem implementa.
- Só pode conter **métodos públicos e abstratos** e **constantes**.
- Não pode ter construtores nem atributos mutáveis.
- Declaração:

java
CopiarEditar
```
interface Cuidador {
    void cuidarPatio();
}
class Cachorro implements Cuidador {
    public void cuidarPatio() { ... }
}
```

📚 LISTAS EM JAVA (FOCO)

➢ O que é uma lista?

- Coleção que armazena **objetos dinamicamente**, podendo crescer ou diminuir de tamanho.
- As listas mais comuns são implementadas pela interface `List`, sendo `ArrayList` a mais usada.

➢ Declaração e uso:

java
CopiarEditar
```
import java.util.ArrayList;

ArrayList<Aluno> lista = new ArrayList<>();
lista.add(new Aluno("João", 8, 7)); // adiciona item

for (Aluno a : lista) {
    System.out.println(a.getNome());
}
```

➤ Métodos comuns da lista:

Método	Descrição
`add(obj)`	Adiciona elemento no final
`get(indice)`	Retorna o elemento no índice
`remove(indice ou obj)`	Remove elemento
`set(indice, novoObj)`	Substitui elemento
`size()`	Retorna quantidade de elementos
`contains(obj)`	Verifica se o item está na lista

➤ Passagem de listas como parâmetro:

java
CopiarEditar

```
public void imprimirAlunos(ArrayList<Aluno> lista) {
    for (Aluno a : lista) {
        System.out.println(a.getNome());
    }
}
```

🧩 EXEMPLO COMPLETO INTEGRANDO TUDO:

java
CopiarEditar

```
abstract class Animal {
    protected String especie;
    protected int idade;

    public Animal(String especie, int idade) {
        this.especie = especie;
        this.idade = idade;
    }

    public abstract void emitirSom();
```

```
    public void exibeDados() {
        System.out.println("Espécie: " + especie + ", Idade: " +
idade);
    }
}

interface Cuidador {
    void cuidarPatio();
}

class Cachorro extends Animal implements Cuidador {
    private String raca;

    public Cachorro(String especie, int idade, String raca) {
        super(especie, idade);
        this.raca = raca;
    }

    @Override
    public void emitirSom() {
        System.out.println("Au Au!");
    }

    @Override
    public void cuidarPatio() {
        System.out.println("Cuidando do pátio...");
    }

    @Override
    public void exibeDados() {
        super.exibeDados();
        System.out.println("Raça: " + raca);
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Animal> animais = new ArrayList<>();
        animais.add(new Cachorro("Canis", 5, "Pitbull"));

        for (Animal a : animais) {
```

```
            a.exibeDados();
            a.emitirSom();

            if (a instanceof Cuidador) {
                ((Cuidador) a).cuidarPatio();
            }
        }
    }
}
```

🎯 DICAS FINAIS PARA GABARITAR

- **Classe abstrata**: modelo base com métodos obrigatórios a serem implementados.
- **Interface**: contrato 100% abstrato para ser implementado por qualquer classe.
- **Lista (`ArrayList`)**: usada para armazenar e manipular objetos dinamicamente.
- **Polimorfismo**: permite usar a mesma referência para objetos de diferentes classes.
- **Encapsulamento**: use `private` + `get/set`.
- **Sobrecarga**: mesmo nome, parâmetros diferentes.
- **Sobrescrita**: mesma assinatura, comportamento diferente.