📘 RESUMO GERAL – POO EM JAVA (com foco em classes abstratas e listas)

🔹 Conceitos Básicos da Programação Orientada a Objetos (POO)

A POO é um paradigma de programação que simula o mundo real, trabalhando com **objetos**, que são instâncias de **classes**. Cada objeto possui **atributos** (características) e **métodos** (comportamentos). As classes funcionam como moldes que descrevem o que os objetos devem ter e fazer.

🧱 Encapsulamento

É o ato de **proteger os dados de uma classe** para que eles só possam ser acessados de forma controlada. Em Java, isso é feito usando **modificadores de acesso**, como:

- `private`: só pode ser acessado dentro da própria classe;
- `public`: pode ser acessado de qualquer lugar;
- `protected`: pode ser acessado por subclasses e pelo mesmo pacote;
- `default`: só dentro do mesmo pacote.

Para acessar atributos `private`, usamos métodos chamados **getters** (para acessar) e **setters** (para modificar). Isso garante que os dados sejam lidos ou alterados com segurança.

🧠 Construtores

Construtores são métodos especiais chamados automaticamente quando um objeto é criado. Eles têm o mesmo nome da classe e podem ter parâmetros para inicializar os atributos. É possível criar **mais de um construtor**, com parâmetros diferentes — isso é chamado de **sobrecarga de construtor**.

🔁 Polimorfismo

É a capacidade de usar **o mesmo método ou nome de forma diferente**, dependendo da situação.

Há dois tipos:

- **Polimorfismo dinâmico (ou sobrescrita)**: quando uma subclasse **redefine** um método da classe mãe, mudando seu comportamento. Isso permite usar uma mesma referência para objetos diferentes.

- **Polimorfismo estático (ou sobrecarga)**: quando criamos **métodos com o mesmo nome**, mas com diferentes parâmetros. O Java decide qual usar com base na assinatura (tipos e número de argumentos).

🔄 Herança

Permite que uma classe (chamada de **subclasse**) herde atributos e métodos de outra classe (a **superclasse**). É usada para **reaproveitar código** e criar uma hierarquia entre as classes. Por exemplo, uma classe `Funcionario` pode ser herdada por `Gerente` ou `Programador`, que terão comportamentos próprios além dos herdados.

🔷 CLASSES ABSTRATAS (FOCO PRINCIPAL)

Uma **classe abstrata** é uma classe incompleta, que **não pode ser instanciada diretamente**. Ela serve como base para outras classes. Geralmente define um modelo genérico, com métodos que podem ou não ter corpo.

Se um método é abstrato, ele **não tem implementação** e **deve ser obrigatoriamente implementado** pelas classes filhas. Isso garante que toda subclasse tenha certos comportamentos obrigatórios.

Exemplo prático: se temos uma classe abstrata `Animal`, cada animal específico (como `Gato` ou `Cachorro`) precisa implementar o método `emitirSom()` definido como abstrato.

As classes abstratas são úteis quando se quer **padronizar comportamentos** entre várias classes parecidas, mas que têm detalhes diferentes entre si.

🧩 INTERFACES (relacionadas às classes abstratas)

Uma **interface** é como um contrato. Ela **define métodos que uma classe deve implementar**, mas não fornece nenhuma implementação. Todas as classes que “assinam” a interface (usando a palavra `implements`) se comprometem a implementar todos os métodos declarados nela.

Diferente das classes abstratas, uma interface **não pode conter atributos mutáveis** e **não pode ter construtores**. Serve para organizar comportamentos comuns entre classes diferentes, mesmo que elas não tenham relação entre si na hierarquia de herança.

📚 LISTAS EM JAVA (FOCO PRINCIPAL)

As **listas** em Java (geralmente `ArrayList`) são estruturas de dados que armazenam vários objetos dinamicamente. Ou seja, não é preciso saber a quantidade de elementos com antecedência. É possível adicionar, remover, acessar e modificar elementos facilmente.

Para criar uma lista, usamos `ArrayList<T>`, onde `T` é o tipo do objeto. Por exemplo, `ArrayList<Aluno>` cria uma lista de objetos do tipo `Aluno`.

Principais métodos:

- `add(objeto)` — adiciona elemento ao final da lista.
- `get(índice)` — recupera um elemento pelo índice.
- `remove(objeto ou índice)` — remove o elemento.
- `size()` — retorna o tamanho atual da lista.

As listas são **muito usadas em programas que precisam trabalhar com coleções de dados**, como uma lista de alunos, contas bancárias, livros, pessoas etc.

Outro ponto importante: podemos passar listas como **parâmetros de métodos**, o que permite reutilizar código. Também é possível retornar listas de métodos.

🎯 Outros conceitos relevantes da prova

- **Sobrecarga**: vários métodos com o mesmo nome, mas com diferentes parâmetros.
- **Sobrescrita**: redefinir um método herdado, mantendo nome e assinatura.
- **Instanciação**: ato de criar um novo objeto com `new`.
- **Composição**: uma classe pode **conter objetos de outras classes como atributos**, criando uma estrutura mais rica (ex: uma classe `Boletim` que possui vários objetos do tipo `Aluno`).

📝 Dicas finais para gabaritar

- Classes abstratas são boas para “forçar” que certas classes filhas implementem métodos importantes.
- Interfaces são ideais para definir **regras comuns entre classes distintas**.
- Listas são extremamente úteis para **armazenar coleções de objetos** e são cobradas frequentemente com `ArrayList`.
- Sempre fique atento à diferença entre **sobrecarga** (overload) e **sobrescrita** (override).
- Não se esqueça de **usar `get` e `set`** para acessar atributos privados (encapsulamento).
- Estude exemplos que misturem herança, polimorfismo, listas e classes abstratas.