O ChatGPT disse:

Perfeito 👍
Então vamos **subir MUITO o nível**.
Abaixo estão **5 exercícios avançados de Reflection**, no **nível framework / infraestrutura**, bem acima do que estava no PDF e ideais para **avaliar aluno sênior**.

Todos exigem **arquitetura**, **organização** e **pensamento de mercado**.

🧩 Exercício 1 – Engine de Validação por Anotações Customizadas

🎯 Objetivo

Criar um **motor de validação genérico**, semelhante ao **Bean Validation**, usando Reflection.

📌 Requisitos

1. Criar as anotações:

```
java

@NotNull
@Min(valor = 5)
@Max(valor = 100)
```

2. O motor deve:

- Percorrer **todos os campos**
- Ler as anotações
- Validar os valores em runtime

- Retornar uma lista de erros

📌 Exemplo de classe

```
java

public class Produto {
    @NotNull
    private String nome;

    @Min(1)
    @Max(100)
    private int quantidade;
}
```

📌 Saída esperada

```
text

Campo 'nome' não pode ser nulo
Campo 'quantidade' deve ser entre 1 e 100
```

🔥 Nível

➡️ Mesmo conceito usado por **Hibernate Validator**

🧩 Exercício 2 – Mapper Avançado com Conversão de Tipos

🎯 Objetivo

Criar um **mapper genérico** capaz de:

- Copiar dados entre classes diferentes
- Converter tipos automaticamente

📌 Requisitos

- String → Integer
- String → LocalDate

- Enum → String

📌 Exemplo

```
java

class UsuarioDTO {
    String idade;
    String dataNascimento;
}

class Usuario {
    Integer idade;
    LocalDate dataNascimento;
}
```

📌 Regra

- Os nomes dos campos devem ser iguais
- Conversão deve ser automática via Reflection

🔥 Nível

➡️ Similar ao funcionamento interno do **MapStruct / ModelMapper**

🧩 Exercício 3 – Injeção de Dependência com Grafo de Objetos

🎯 Objetivo

Criar um **mini container IoC**, com **injeção recursiva**.

📌 Requisitos

1. Criar a anotação:

```
java

@Inject
```

2. O container deve:

- Instanciar a classe principal

- Detectar campos anotados com `@Inject`
- Criar dependências automaticamente
- Resolver dependências aninhadas

📌 Exemplo

```
java

class PedidoService {
    @Inject
    private PedidoRepository;
}

class PedidoRepository {
    @Inject
    private DataSource;
}
```

📌 Regra importante

- Nenhum `new` permitido fora do container

🔥 Nível

➡️ Mesmo princípio do **Spring Core**

🧩 Exercício 4 – Executor Dinâmico de Casos de Uso (Use Case Engine)

🎯 Objetivo

Criar um **motor de execução de casos de uso**, totalmente desacoplado.

📌 Requisitos

1. Criar a anotação:

```
java

@UseCase(nome = "CRIAR_USUARIO")
```

2. Criar uma interface:

```java
public interface UseCaseExecutor {
    void executar(Map<String, Object> params);
}
```

3. O engine deve:

- Escanear classes
- Registrar os casos de uso
- Executar pelo nome
- Passar parâmetros dinamicamente

📌 Exemplo

```java
engine.executar("CRIAR_USUARIO", params);
```

🔥 Nível

➡️ Arquitetura usada em **Clean Architecture / Hexagonal**

🧩 Exercício 5 – Proxy Dinâmico para Auditoria e Performance

🎯 Objetivo

Criar um **proxy dinâmico** que:

- Intercepte chamadas de métodos
- Meça tempo de execução
- Faça log antes e depois

📌 Requisitos

- Usar `InvocationHandler`
- Criar proxy em runtime

- Não alterar a classe original

📌 Exemplo de saída

```
text

[LOG] Método salvar iniciado
[LOG] Método salvar finalizado em 12ms
```

🔥 Nível

➡️ Base de **AOP (Aspect Oriented Programming)**

🎓 Critérios de Avaliação (estilo prova técnica)

Cada exercício deve ser avaliado em:

- Uso correto da Reflection API
- Organização do código
- Tratamento de exceções
- Clareza arquitetural
- Evitar uso indevido de Reflection