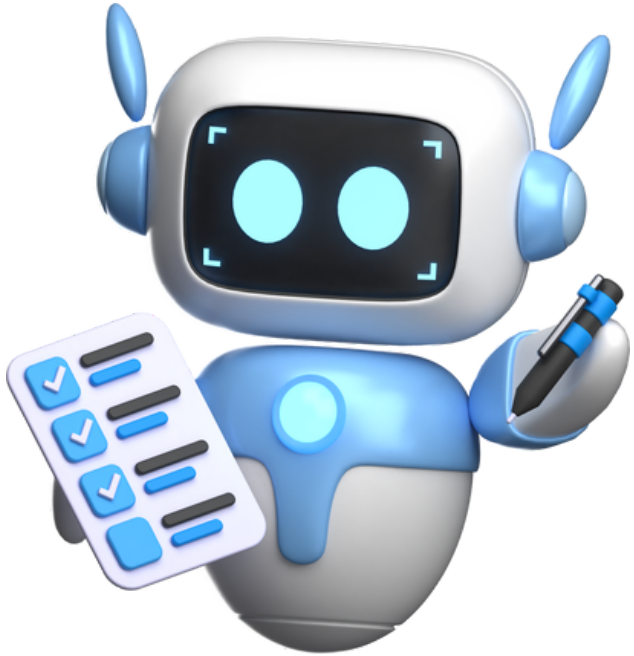
UNIVASSOURAS
CAMPUS UNIVERSITÁRIO DE MARICÁ

Curso Bacharelado em
Engenharia de Software

Lab. de Programação Orientada a Objetos

Prof. Tiago Ruiz

Prova_P2 - 28/11/2025



LSP

Passo a Passo para as resoluções

1. **Criar classe abstrata base**
    a. Use `**ABC**` e `**@abstractmethod**`
    b. Nome sugerido da classe**: `MaterialBiblioteca`**
    c. Métodos abstratos: `**emprestar()**` e `**devolver()**`
2. **Criar interface separada para renovação**
    a. Use `**ABC**` e `**@abstractmethod**`
    b. Nome sugerido da classe: **`Renovavel`**
    c. Método abstrato: `renovar()`
3. **Refatorar classe Livro**
    a. Herde de `**MaterialBiblioteca**` E `**Renovavel**`
    b. Implemente todos os métodos abstrato
4. **Refatorar classe LivroDigital**
    a. Herde **APENAS** de `**MaterialBiblioteca**` (não `Renovavel`)
    b. Implemente apenas `**emprestar()**` e `**devolver()**`
5. **Ajustar método processar_renovacao**
    a. Verifique se o material é `**Renovavel**` antes de renovar
    b. Use `**isinstance()**` para verificação

OCP

Passo a Passo para as resoluções

1. **Criar interface abstrata para estratégias**
   a. Use `**ABC**` e `**@abstractmethod**`
   b. Nome sugerido para classe: `**EstrategiaMulta**`
   c. Método abstrato: `**calcular_multa(self, dias_atraso)**`
2. **Criar classes concretas para cada tipo de multa**
   a. `**MultaAlunoGraduacao**` → R$ 2,00/dia
   b. `**MultaAlunoPos**` → R$ 1,50/dia
   c. `**MultaFuncionario**` → R$ 3,00/dia
   d. **Cada uma das classes acima implementa `EstrategiaMulta`**
3. **Modificar EmprestimoService**
   - Receba `**EstrategiaMulta**` no construtor (injeção de dependência)

```
class EmprestimoService:
    """Responsável APENAS pela lógica de empréstimos"""

    def __init__(self, repository, email_service, estrategia_multa):
```

No exemplo de uso fica assim

```
estrategia_multa = MultaAlunoGraduacao()
service = EmprestimoService(repository, email_service, estrategia_multa)
```

   - Remova o parâmetro `**tipo_usuario**` de `**realizar_emprestimo()**` e `**calcular_multa()**`
   - Use a **estratégia injetada** para calcular a multa
   - Armazene a estratégia como atributo da classe

SRP

Passo a Passo para as resoluções

1. **Criar classe para persistência**
   - Nome sugerido: `**LivroRepository**`
   - Métodos: `**salvar_emprestimo()**` e `**atualizar_emprestimo()**`
   - **Mova a lógica de banco de dados para cá**
2. **Criar classe para notificações**
   - Nome sugerido: `**EmailService**`
   - Métodos: `**enviar_confirmacao_emprestimo()**` e `**enviar_notificacao_renovacao()**`
   - **Mova a lógica de email para cá**
3. Refatorar **EmprestimoService**
   - Receba `**LivroRepository**` e `**EmailService**` no construtor

```python
class EmprestimoService:
    """Responsável APENAS pela lógica de empréstimos"""

    def __init__(self, repository, email_service, estrategia_multa):
```

No exemplo de uso fica assim

```python
repository = LivroRepository()
email_service = EmailService()
estrategia_multa = MultaAlunoGraduacao()
service = EmprestimoService(repository, email_service, estrategia_multa)
```

   - Remova métodos `**salvar_no_banco()**` e `**enviar_email_confirmacao()**`
   - Use injeção de dependência para delegar essas responsabilidades

Exemplo de USO

Passo a Passo para as resoluções

```python
if __name__ == "__main__":
    # Configuração (SRP + OCP)
    repository = LivroRepository()
    email_service = EmailService()
    estrategia_multa = MultaAlunoGraduacao()
    service = EmprestimoService(repository, email_service, estrategia_multa)

    # Criar materiais (LSP)
    livro_fisico = Livro("123", "Python Básico", "Autor X")
    livro_digital = LivroDigital("456", "Python Avançado", "Autor Y", "http://...")

    # Testes
    print("=== EMPRÉSTIMOS ===")
    service.realizar_emprestimo(livro_fisico, "João")
    service.realizar_emprestimo(livro_digital, "Maria")

    print("\n=== RENOVAÇÃO (LSP) ===")
    service.processar_renovacao(livro_fisico)  # OK
    service.processar_renovacao(livro_digital)  # OK (não quebra!)

    print("\n=== MULTA (OCP) ===")
    print(f"Multa (5 dias): R$ {service.calcular_multa(5):.2f}")
```

QUESTÃO 2: ISP

Passo a Passo para as resoluções

- **Analise quais métodos TODOS os veículos precisam**
  - Exemplo: **`ligar()`, `acelerar()`, `frear()`**

- **Identifique métodos específicos**
  - Exemplo: `**abastecer()**` (só motorizados)
  - Exemplo: `**trocar_pneu()**` (só terrestres)

PASSO 2: Criar interfaces segregadas

- **Interface básica (todos os veículos)**
  - Nome sugerido para classe: `**VeiculoBasico**`
  - Métodos: `**ligar()**`, `**acelerar()**`, `**frear()**`
- **Interface para motorizados**
  - Nome sugerido para classe: `**VeiculoMotorizado**`
  - Método: `**abastecer()**`
- **Interface para terrestres**
  - Nome sugerido para classe: `**VeiculoTerrestre**`
  - Método: `**trocar_pneu()**`

QUESTÃO 2: ISP

Passo a Passo para as resoluções

PASSO 3: Refatorar classes concretas

- **Classe Carro**
  - Implemente: `**VeiculoBasico**` + `**VeiculoMotorizado**` + `**VeiculoTerrestre**`
  - Use heranças múltiplas
- **Classe Bicicleta**
  - Implemente APENAS: `**VeiculoBasico**`
  - Remova métodos que não fazem sentido
- **Classe Barco**
  - Implemente: `**VeiculoBasico**` + `**VeiculoMotorizado**`
  - Não implemente `**VeiculoTerrestre**`

QUESTÃO 2: ISP

Passo a Passo para as resoluções

Exemplo de USO

```python
if __name__ == "__main__":
    print("=== TESTE COM CARRO ===")
    carro = Carro("Fusca")
    print(carro.ligar())  # OK
    print(carro.abastecer(50))  # OK
    print(carro.trocar_pneu("dianteiro"))  # OK
    print(carro.acelerar(60))  # OK
    print(carro.frear())  # OK

    print("\n=== TESTE COM BICICLETA ===")
    bicicleta = Bicicleta("Caloi")
    print(bicicleta.ligar())  # OK
    print(bicicleta.acelerar(10))  # OK
    print(bicicleta.frear())  # OK

    print("\n=== TESTE COM BARCO ===")
    barco = Barco("Lancha")
    print(barco.ligar())  # OK
    print(barco.abastecer(100))  # OK
    print(barco.acelerar(30))  # OK
    print(barco.frear())  # OK
```

Fim!

Boa Prova!