EXERCÍCIO 3

**Code smells:** “God Class” (classe com responsabilidades em excesso) e “LongClass”.  
**Refatoração:** Extração de classes (ex: GestaoAlunos, RelatorioService, PagamentoService), uso de pacotes por domínio. **Impacto:** Melhora na manutenibilidade, clareza arquitetural, maior possibilidade de evolução sem quebrar múltiplas partes do sistema

Desvantagens: Aumento de esforço de manutenção, risco de  
inconsistência (corrige em um lugar e esquece em outro), dificuldade de evolução.

Refatoração: Extract Method e Pull Up Method (centralizar lógica em uma classe utilitária ou serviço de domínio como FreteService).

Problemas: Lista de parâmetros longa, baixa clareza, risco de inversão na ordem de parâmetros, booleans pouco expressivos.  
Refatoração: Introduce Parameter Object.  
Benefícios: menor complexidade, código autodescritivo.

Problema: Condicionais aninhados, difícil manutenção.  
Refatoração: Replace Conditional with Polymorphism (criar subclasses de Funcionario para diferentes cargos/benefícios).  
Benefício: regras de negócio mais claras, facilidade para adicionar novos casos  
sem alterar código legado.

Problema: Nomenclatura ruim, aumenta curva de aprendizado e probabilidade de erro.

Refatoração: Rename Method/Variable com nomes significativos (calcularImposto() em vez de doIt()).

Justificativa: clareza semântica reduz a dependência de documentação externa,  
melhora comunicação entre desenvolvedores e facilita code review.

a) O código duplicado compromete a manutenção porque cada alteração precisa ser feita em vários pontos, aumentando o risco de inconsistências, dificultando evolução e tornando o sistema mais propenso a erros.  
b) Pull Up Method: mover o cálculo para uma superclasse comum. Extract Class: criar uma classe específica (CalculadoraPedidos)

1. a) Dependência excessiva entre RelatorioFinanceiro e Funcionario, quebrando o encapsulamento.Manutenção(dependências)  
   b) Move Method: mover os cálculos de bônus e descontos para Funcionario. Extract Class: criar uma classe dedicada (ex.: CalculadoraDeBeneficios) para concentrar essa responsabilidade.
2. SUBSTITUTE ALGORITHM

**Local:** No método extrato() da classe Cliente, movi o cálculo do valor para dentro da classe Aluguel().

A refatoração Substitute Algorithm foi aplicada para mover a lógica de cálculo do valor de cada aluguel do método extrato() da classe Cliente para um novo método chamado getValorAluguel() dentro da classe Aluguel. Isso torna o código mais coeso, facilita a manutenção e melhora a legibilidade, respeitando o princípio da responsabilidade única.

**ANTES DENTRO DE extrato()**

switch (aluguel.getFita().getCodigoDePreco()) {

case Fita.NORMAL:

valorCorrente += 2;

if (aluguel.getDiasAlugada() > 2)

valorCorrente += (aluguel.getDiasAlugada() - 2) \* 1.5;

break;

**DEPOIS DENTRO DE Aluguel()**

public double getValorAluguel() {

double valor = 0;

switch (fita.getCodigoDePreco()) {

case Fita.NORMAL:

valor += 2;

if (diasAlugada > 2)

valor += (diasAlugada - 2) \* 1.5;

break;

case Fita.LANCAMENTO:

valor += diasAlugada \* 3;

break;

case Fita.INFANTIL:

valor += 1.5;

if (diasAlugada > 3)

valor += (diasAlugada - 3) \* 1.5;

break;

}

return valor; }

- REMOVE PARAMETER  
**Local:** No método setCodigoDePreco(int codPreco) da classe Fita.  
**Justificativa:** Em vez de passar o valor como parâmetro diretamente, pode ter métodos específicos como setPrecoLancamento(), setPrecoInfantil(), etc.

**ANTES**

public void setcodigoDePreco(int codPreco) {

this.codigoDePreco = codPreco;

}

**DEPOIS**

public void setComoLancamento() {

this.codigoDePreco = LANCAMENTO;

}

public void setComoNormal() {

this.codigoDePreco = NORMAL;

}

public void setComoInfantil() {

this.codigoDePreco = INFANTIL;

}