ATIVIDADE:: REFATORAÇÃO

Atividades 1 & 2: Detecção e Melhoria de Code Smells (Exemplo da Locadora)

1. Identificação de Code Smells no Projeto (Versão Inicial):

Analisando o código inicial da locadora (especificamente o método conta(), que é o foco principal da refatoração), podemos identificar vários *code smells* definidos no próprio documento:

- **Método Longo (Inflador):** O método conta() é o exemplo clássico de um "Método Longo". Ele acumula muitas responsabilidades: calcula o valor de cada locação, soma os pontos de locador frequente, calcula o total e formata a string de resultado. A própria aula o descreve como "Um método longo como esse".
- **Comandos Switch (Abusador da OO):** O método conta() contém um switch complexo para determinar o preço com base no códigoPreço do filme. Isso é um *code smell* do tipo "Comandos Switch", que geralmente indica uma falha em usar polimorfismo.
- **Inveja dos Dados (Acoplador):** Quando o cálculo do preço é extraído para o método quantiaDe na classe Cliente, esse novo método exibe "Inveja dos Dados". Ele acessa muito mais dados de *outro* objeto (no caso, umaLocação e, por extensão, Filme) do que de sua própria classe (Cliente).
- **Código Duplicado (Dispensável):** O documento levanta a hipótese de criar um contaHtml(). Se o programador seguisse o caminho do "copiar, colar e adaptar o código", ele criaria "Código Duplicado", um *smell* da categoria "Dispensáveis". A lógica de cálculo de preço e pontos seria replicada em dois métodos.

2. Como Melhorar o Projeto e Remover os Code Smells:

O próprio documento demonstra as refatorações necessárias, que servem como tratamento para os *smells* identificados:

- **Para "Método Longo" (conta()):** A refatoração aplicada é **Extract Method**.
    - **Solução:** O bloco switch é extraído do conta() para um novo método, quantidade(cada), que depois é renomeado para quantiaDe(umaLocação).
- **Para "Inveja dos Dados" (quantiaDe()):** A refatoração é **Move Method**.
    - **Solução:** O método quantiaDe, que estava em Cliente, é movido para onde os dados estão: primeiro para a classe Locação (e renomeado lerPreço) e, mais tarde, para a classe Filme.
- **Para "Código Duplicado" (potencial):** A refatoração é **Replace Temp with Query** (Substituir Variáveis Temporárias por Consultas).
    - **Solução:** As variáveis temporárias que calculavam os totais (quantiaTotal e pontosLocadorFrequente) são substituídas por métodos de consulta (lerPreçoTotal() e lerTotalPontosLocadorFreqüente()). Isso permite que tanto o conta() quanto o novo contaHtml() reutilizem a mesma lógica de cálculo, evitando duplicação.

- **Para "Comandos Switch":** A refatoração é **Replace Conditional with Polymorphism** (ou, no caso, "Replace Type Code with State/Strategy").

- **Solução:** Esta é a mudança mais profunda. O códigoPreço é substituído por um objeto Preço que segue o padrão State. A lógica do switch é distribuída entre classes polimórficas (PreçoNormal, PreçoInfantil, PreçoLançamento).

Atividade 5: Ferramentas de Detecção de Code Smells (JAVA)

Abaixo está uma pesquisa e uma tabela comparativa de três ferramentas populares para detecção de *code smells* em código JAVA.

Ferramenta	Foco Principal	Exemplo de *Code Smells* Detectados	Integração
SonarQube (com SonarJava)	Qualidade e Segurança Contínua	É uma plataforma que agrega *Bugs*, *Vulnerabilidades* e *Code Smells*. Cobre uma vasta gama, incluindo "Método Longo", "Complexidade Ciclomática" alta, "Código Duplicado" e vulnerabilidades de segurança.	Servidor web independente, plugins para IDEs (SonarLint), e integração com CI/CD (Maven, Gradle).
PMD	Más Práticas de Programação	Foca em "más práticas", "código morto" (unused variables), *catch blocks* vazios, "complexidade excessiva" e uso ineficiente de objetos.	Plugins para a maioria das IDEs (Eclipse, IntelliJ) e ferramentas de *build* (Maven, Ant).
Checkstyle	Padrões de Estilo de Código	Foca primariamente em "convenções" e "estilo", como formatação (uso de chaves, espaçamento), convenções de nomenclatura e "magic numbers". Embora focado em estilo, o não cumprimento de padrões é considerado um *code smell* por muitos.	Plugins para IDEs e ferramentas de *build*. Muitas vezes usado em conjunto com PMD.

ATIVIDADE:: REFATORAÇÃO

Atividade 1: Tabelas Comparativas de Refatoração em IDEs

Para esta atividade, usarei meu conhecimento sobre três IDEs populares em Java:

IntelliJ IDEA, **Eclipse** e **Visual Studio Code** (com o "Extension Pack for Java").

Aviso: As IDEs estão em constante atualização. Esta tabela reflete as funcionalidades de refatoração mais comuns e conhecidas.

1. Compondo Métodos

Refatoração (Exemplo do PDF)	IntelliJ IDEA	Eclipse	VS Code (com Java Ext.)
Extract Method	✔ (Completo)	✔ (Completo)	✔ (Completo)
Replace Method with Method Object	✔ (Completo)	✔ (Completo)	✘ (Requer manual)
Split Temporary Variable	✔ (Completo)	✔ (Completo)	✘ (Requer manual)

2. Movendo Recursos entre Objetos

Refatoração (Exemplo do PDF)	IntelliJ IDEA	Eclipse	VS Code (com Java Ext.)
Extract Class	✔ (Completo)	✔ (Completo)	✘ (Requer manual)
Move (Method/Field)	✔ (Completo)	✔ (Completo)	✔ (Completo)
Remove Middle Man	⚠ (Parcial: "Inline")	⚠ (Parcial: "Inline")	✘ (Requer manual)
Introduce Foreign Method	✘ (Requer manual)	✘ (Requer manual)	✘ (Requer manual)

3. Organizando Dados

Refatoração (Exemplo do PDF)	IntelliJ IDEA	Eclipse	VS Code (com Java Ext.)
Replace Data Value with Object	✔ (Completo)	✔ (Completo)	✘ (Requer manual)
Replace Magic Number with Constant	✔ (Completo: "Introduce Constant")	✔ (Completo: "Extract Constant")	✔ (Completo: "Extract to constant")
Encapsulate Field	✔ (Completo)	✔ (Completo)	✔ (Completo)

4. Simplificando Expressões Condicionais

Refatoração (Exemplo do PDF)	IntelliJ IDEA	Eclipse	VS Code (com Java Ext.)
Decompose Conditional	⚠ (Parcial: via "Extract Method")	⚠ (Parcial: via "Extract Method")	⚠ (Parcial: via "Extract Method")
Consolidate Conditional	✔ ("Merge 'if's")	✔ ("Combine 'if's")	✘ (Requer manual)
Replace Conditional with Polymorphism	✔ (Completo)	✔ (Completo)	✘ (Requer manual)

ATIVIDADE:: REFATORAÇÃO

5. Simplificando Chamadas de Métodos

Refatoração (Exemplo do PDF)	IntelliJ IDEA	Eclipse	VS Code (com Java Ext.)
Replace Error Code with Exception	❌ (Requer manual)	❌ (Requer manual)	❌ (Requer manual)
Introduce Parameter Object	✔ (Completo)	✔ (Completo)	❌ (Requer manual)
Separate Query from Modifier	❌ (Requer manual)	❌ (Requer manual)	❌ (Requer manual)

6. Lidando com Generalização

Refatoração (Exemplo do PDF)	IntelliJ IDEA	Eclipse	VS Code (com Java Ext.)
Pull Up Field / Method	✔ (Completo)	✔ (Completo)	✔ (Completo)
Push Down Field / Method	✔ (Completo)	✔ (Completo)	✔ (Completo)
Extract Interface	✔ (Completo)	✔ (Completo)	✔ (Completo)
Replace Inheritance with Delegation	✔ (Completo)	✔ (Completo)	❌ (Requer manual)

Atividade 2: Exemplos de Code Smells e Refatorações

1. Compondo Métodos: Extract Method

- **Code Smell (Antes):** Um método longo com um bloco de código comentado explicando seu propósito.

```
void imprimirDívida(double quantia) {
  imprimirCabeçalho();
  // imprime os detalhes
  System.out.println("nome:" + _nome);
  System.out.println("quantia: " + quantia);
}
```

- **Refatoração (Depois):** O bloco de código é transformado em um novo método com um nome autoexplicativo.

```
void imprimirDívida(double quantia) {
  imprimirCabeçalho();
  imprimirDetalhes(quantia);
}

private void imprimirDetalhes(double quantia) {
  System.out.println("nome:" + _nome);
```

```
  System.out.println("quantia: " + quantia);
}
```

2. Movendo Recursos: Extract Class

- **Code Smell (Antes):** Uma classe (Pessoa) fazendo o trabalho de duas, acumulando campos que pertencem a outro conceito (Telefone).
- **Refatoração (Depois):** Uma nova classe (NúmeroDoTelefone) é criada, e os campos e métodos pertinentes são movidos para ela, dividindo as responsabilidades.

3. Organizando Dados: Replace Magic Number with Symbolic Constant

- **Code Smell (Antes):** Um número literal (9.81) com significado especial está "solto" no código.

```
double energiaPotencial(double massa, double altura) {
  return massa * 9.81 * altura;
}
```

- **Refatoração (Depois):** O número mágico é substituído por uma constante nomeada, que explica seu significado.

```
static final double CONSTANTE_GRAVITACIONAL = 9.81;

double energiaPotencial(double massa, double altura) {
  return massa * CONSTANTE_GRAVITACIONAL * altura;
}
```

4. Simplificando Condicionais: Decompose Conditional Expression

- **Code Smell (Antes):** Uma expressão condicional complexa e difícil de ler.

```
if (data.before(INÍCIO_VERÃO) || data.after(FIM_VERÃO))
  aCobrar = quantidade * _taxaDeInverno + _preçoDoServiçoNoInverno;
else
  aCobrar = quantidade * _taxaDeVerão;
```

- **Refatoração (Depois):** A condição e os blocos de instrução são extraídos para métodos com nomes claros, explicando a intenção.

```
if (nãoÉVerão(data))
  aCobrar = preçoDeInverno(quantidade);
else
  aCobrar = preçoDeVerão(quantidade);
```

5. Simplificando Chamadas: Replace Error Code with Exception

- **Code Smell (Antes):** O método retirado retorna um código de erro especial (-1) para indicar uma falha.

```
int retirada(int quantia) {
  if (quantia > _saldo)
    return -1;
  else {
    _saldo -= quantia;
    return 0;
  }
}
```

- **Refatoração (Depois):** O método passa a gerar uma exceção, separando o processamento normal do processamento de erro.

```
void retirada(int quantia) throws ExceçãoDeSaldo {
  if (quantia > _saldo)
    throw new ExceçãoDeSaldo();
  _saldo -= quantia;
}
```

6. Lidando com Generalização: Pull Up Field

- **Code Smell (Antes):** Duas subclasses (Vendedor e Engenheiro) possuem o mesmo campo (nome), indicando duplicação.
- **Refatoração (Depois):** O campo duplicado é movido para a superclasse (Empregado), generalizando o dado.

Atividade 3: Cenários de Teste

- **Teste (Antes):** O teste deve verificar o código de retorno.
  - testaRetiradaComSucesso():
    1. Cria uma conta com saldo 100.
    2. Chama retirada(70).
    3. Verifica se o método retornou 0 (sucesso).
    4. Verifica se o saldo da conta é 30.
  - testaRetiradaSemSaldo():
    1. Cria uma conta com saldo 100.
    2. Chama retirada(120).
    3. Verifica se o método retornou -1 (erro).
    4. Verifica se o saldo da conta permanece 100.

- **Teste (Depois):** O teste deve verificar se a exceção correta é lançada.
  - testaRetiradaComSucesso():
    1. Cria uma conta com saldo 100.

2. Chama retirada(70).
3. Verifica se *nenhuma* exceção foi lançada.
4. Verifica se o saldo da conta é 30.

- testaRetiradaSemSaldo():
    1. Cria uma conta com saldo 100.
    2. Verifica se a chamada retirada(120) lança a ExceçãoDeSaldo.
    3. Verifica se o saldo da conta permanece 100.

Atividade 4: Comentários sobre o Processo Manual

O passo-a-passo manual para a refatoração 3 (Replace Magic Number) seria:

1. Identificar o "número mágico" (ex: 9.81).
2. Declarar uma nova constante com um nome significativo (ex: static final double CONSTANTE_GRAVITACIONAL = 9.81;).
3. Usar a ferramenta de "Buscar e Substituir" da IDE ou editor.
4. Buscar *todas* as ocorrências de "9.81".
5. Revisar *manualmente* cada ocorrência para garantir que ela se refere de fato à constante gravitacional (e não a um preço de R$ 9,81, por exemplo).
6. Substituir as ocorrências corretas por CONSTANTE_GRAVITACIONAL.
7. Compilar e executar os testes.

Atividade 5: Comentários sobre o Processo com IDE

O processo de remover

Code Smells com as ferramentas de uma IDE (como IntelliJ ou Eclipse) é drasticamente mais seguro e eficiente:

1. O programador clica com o botão direito no número mágico (9.81).
2. Seleciona "Refactor" -> "Introduce Constant...".
3. Digita o nome (CONSTANTE_GRAVITACIONAL).
4. A IDE automaticamente altera o código-fonte e, crucialmente, oferece a opção de substituir *todas as ocorrências* detectadas daquele mesmo valor no escopo (classe, pacote ou projeto).
5. A refatoração é concluída em segundos e de forma segura.

Sobre os testes: Após aplicar a refatoração automática, os testes seriam executados novamente. Eles continuariam executando sem falhas. A ferramenta de refatoração garante que a substituição seja precisa e completa, sem alterar a lógica subjacente. A única mudança é a substituição de um valor literal por uma constante que armazena *exatamente o mesmo valor*. Isso valida o princípio fundamental da refatoração: alterar a estrutura interna sem alterar o comportamento observável.

ATIVIDADE:: REFATORAÇÃO

Atividade 6: Repetir o Processo

Não executada.