Entrega 3 - Projeto de código

1) Princípios de Bom Projeto de Código e Relação com os "Maus Cheiros" de Fowler

1. Simplicidade

Manter o código simples, claro e objetivo, evitando complexidade desnecessária.

Maus Cheiros:

- Método Longo: Dificulta a compreensão.
- Classe Grande: Acumula muitas responsabilidades.
- Generalização Especulativa: Cria estruturas sem necessidade atual. 2.

Elegância

Código deve ser bem estruturado, legível e organizado.

Maus Cheiros:

- Agrupamentos de Dados: Repetição desnecessária de dados.
- **Decisões Complexas:** Uso excessivo de *switch/if-else* em vez de polimorfismo.
- **Classe Ociosa**: Pouca utilidade, aumentando a complexidade sem necessidade.

3. Modularidade

O código deve ser organizado em módulos coesos e independentes.

Maus Cheiros:

- **Mudanças Divergentes:** Classe alterada por múltiplas razões, falta de coesão.
- Inveja de Funcionalidade: Método depende mais de outra classe do que da própria.
- Cirurgia de Espingarda: Uma mudança impacta várias classes, indicando fragilidade na modularização.

4. Boas Interfaces

Interfaces devem ser claras, mínimas e bem encapsuladas, definindo contratos específicos.

Maus Cheiros:

- Intimidade Inapropriada: Classes expõem detalhes internos entre si.

- Cadeias de Mensagens: Encadeamento excessivo de chamadas, revelando implementações internas.
- Violação de Segregação de Interface: Interfaces grandes e genéricas, sem especialização adequada.

5. Extensibilidade

Facilidade para adicionar funcionalidades sem afetar o sistema existente.

Maus Cheiros:

- **Hierarquias Paralelas:** Alterações em uma classe exigem mudanças em outra.
- Rigidez: Dificuldade de modificação devido a dependências rígidas.
- Classe Base Frágil: Mudanças na classe base impactam negativamente as subclasses.

6. Evitar Duplicação

Evitar redundância de código, seguindo o princípio DRY (**Don't Repeat Yourself**).

Maus Cheiros:

- Código Duplicado: Trechos iguais ou semelhantes em diferentes partes do código.
- Cirurgia de Espingarda: Mudança que exige ajustes repetidos em vários locais.

7. Portabilidade

Código deve ser adaptável a diferentes ambientes ou plataformas.

Maus Cheiros:

- Obsessão por Primitivas: Uso excessivo de tipos primitivos, dificultando abstração e portabilidade.
- Constantes Fixas: Valores hard-coded, como caminhos de arquivos, que limitam a flexibilidade do código.

8. Código Idiomático e Bem Documentado

Seguir convenções da linguagem e documentar claramente decisões de projeto.

Maus Cheiros:

 Comentários Excessivos: Explicações desnecessárias para código que deveria ser autoexplicativo.

- Classe de Dados: Classes com apenas campos e getters/setters, sem comportamento relevante.
- Construtos Não Idiomáticos: Implementação manual de recursos já oferecidos pela linguagem.

2) Identificação de Maus Cheiros e Princípios Violados

1. Obsessão por Primitivas Arquivos

Afetados:

- DependenteManager.java DeducaoManager.java Descrição:
 Uso excessivo de arrays (*String[]*, *float[]*) para armazenar listas de dependentes, deduções e valores, em vez de utilizar coleções (e.g., *List*) ou objetos dedicados.
 Princípios Violados:
- **Simplicidade:** Código complexo para manipulação manual de arrays.
- Modularidade: Falta de encapsulamento para operações de adição/consulta.
 Refatoração Sugerida:
- Criar classes como Dependente (com nome e parentesco) ou Deducao (com nome e valor).
- Substituir arrays por *List* para simplificar operações.

2. Código Duplicado

Arquivos Afetados:

- DependenteManager.java (método adicionarAoArray)
- DeducaoManager.java (métodos adicionarAoArray para String e float) Descrição:
 Métodos idênticos para expansão de arrays em classes diferentes.

Princípios Violados:

- Evitar Duplicação: Violação do princípio DRY (Don't Repeat Yourself)
 Refatoração Sugerida:
- Criar uma classe utilitária para operações genéricas com arrays.
- Implementar um método genérico para adicionar elementos a arrays.

3. Intimidade Inapropriada Arquivos

Afetados:

- CadastroDependente.java - DependenteManager.java **Descrição**:

A classe CadastroDependente chama diretamente o método adicionarDependente de DependenteManager, expondo detalhes internos.

Princípios Violados:

- Boas Interfaces: Acoplamento excessivo entre classes.

Refatoração Sugerida:

- Encapsular a lógica de adição dentro de DependenteManager, removendo a necessidade de CadastroDependente.
- Eliminar CadastroDependente e mover a lógica para DependenteManager.

4. Agrupamentos de Dados Arquivos

Afetados:

- CadastroDependente.java (parâmetros nome e parentesco).
- DependenteManager.java (arrays separados para nomes e parentescos).

Descrição:

Parâmetros e dados relacionados (nome e parentesco) são tratados separadamente, sem encapsulamento.

Princípios Violados:

- Elegância: Falta de coesão na representação de dados.

Refatoração Sugerida:

- Criar uma classe Dependente para agrupar nome e parentesco.
- 5. Inveja de Funcionalidade Arquivos

Afetados:

- IRPF.java (método getDeducao).

Descrição:

O método getDeducao em IRPF realiza cálculos que poderiam ser responsabilidade de DeducaoManager ou DependenteManager.

Princípios Violados:

- Modularidade: Lógica distribuída de forma não coesa.

Refatoração Sugerida:

 Transferir a lógica de cálculo de deduções para as classes responsáveis, por exemplo, DeducaoManager.

6. Constantes Fixas

Arquivos Afetados:

- IRPF.java (constantes FAIXAS e ALIQUOTAS).

Descrição:

Valores fixos para faixas de imposto e alíquotas, dificultando ajustes futuros.

Princípios Violados:

- Portabilidade: Dificulta adaptação a mudanças nas regras fiscais.

Refatoração Sugerida:

 Criar uma classe dedicada (e.g., `TaxasImposto`) para armazenar essas constantes.

7. Classe Ociosa

Arquivos Afetados: -

CadastroDependente.java

Descrição:

A classe CadastroDependente apenas delega uma operação para DependenteManager, sem adicionar valor significativo.

Princípios Violados:

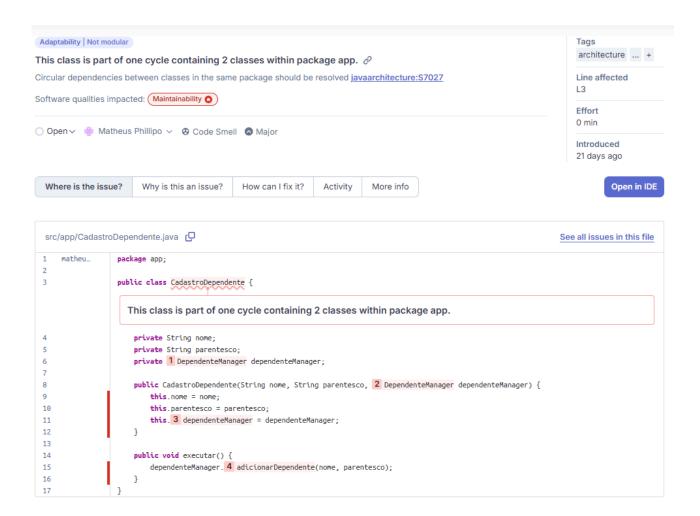
- Simplicidade: Complexidade desnecessária na estrutura.

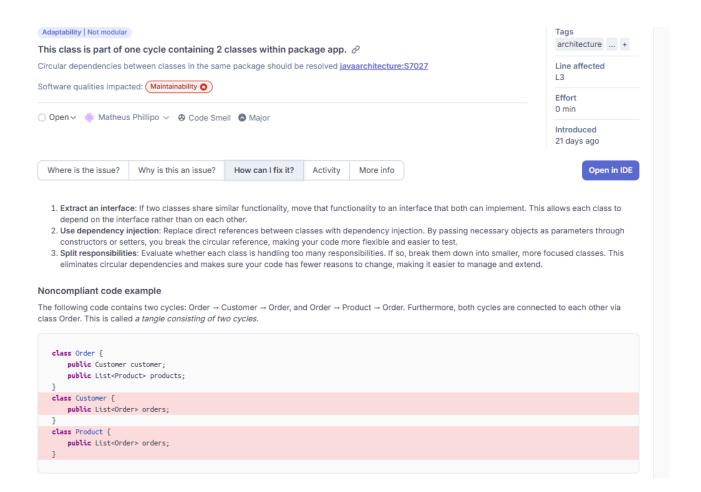
Refatoração Sugerida:

- Remover CadastroDependente e mover executar() para DependenteManager.

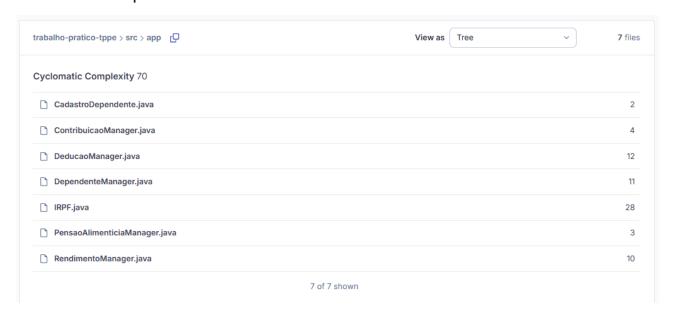
Extra - Uso do SonarCloud (SonarQube)

Issue encontrada:

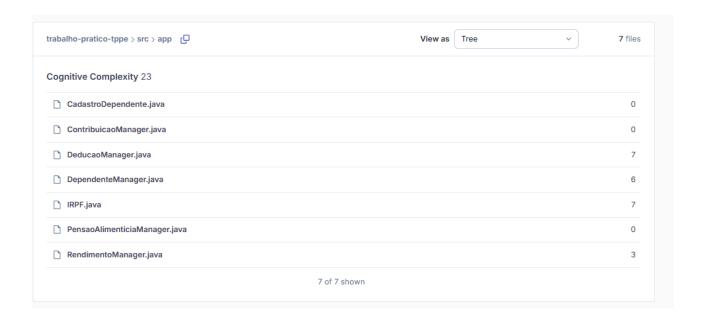




Casos de complexidade ciclomática:



Casos de Complexidade cognitiva:



Utilizamos esses casos para buscar na literatura as refatorações devidas, pois a complexidade cognitiva é uma medida de quão difícil é entender um código ou sistema de software e a complexidade ciclomática é uma métrica de software que quantifica o número de caminhos independentes em um programa. Quanto mais ocorrências desses casos, o software com o tempo será mais custoso para dar manutenção