Trabalho Prático 3 - Grupo 11

Técnicas de Programação em Plataformas Emergentes - Prof. André Lanna - 2024/2

Membros:

- Lorenzo de Lima Alves dos Santos - 190032921
- Lucas Rodrigues Monteiro - 180125974
- Eduardo Rodrigues de Farias - 190086521

Questão 1

- **Simplicidade**:
  - **Definição**: O código deve ser claro e suficientemente pequeno, evitando complexidade desnecessária;
  - **Mau-cheiro relacionado**: Métodos Longos (código extenso e difícil de entender).
- **Elegância**:
  - **Definição:** O código deve ser coerente, sem complexidade demasiada e casos especiais, com alterações pontuais que não afetem o resto;
  - **Mau-cheiro relacionado**: Código Duplicado (lógica repetida em vários lugares).
- **Modularidade**:
  - **Definição:** Os módulos do código deve ter funcionalidades específicas que serão agrupadas de acordo com um propósito, além de serem chamados por outros módulos apenas quando necessários;
  - **Mau-cheiro relacionado**: Classe Grande (muitas responsabilidades em uma única classe).

- **Boas interfaces**:
  - **Definição:** Os módulos devem estabelecer uma forma de comunicação consistente entre módulos, garantindo interações previsíveis e focadas, reduzindo dependências indesejadas.
  - **Mau-cheiro relacionado:** Falta de Encapsulamento (exposição direta de dados internos).
- **Extensibilidade**:
  - **Definição:** O código deve ser estruturado de forma que adicionar funcionalidades futuras não exija alterações drásticas, sem deixar muito genérico;
  - **Mau-cheiro relacionado:** Código Rígido (dificuldade em adicionar novas funcionalidades).

- **Evitar duplicação**:
  - **Definição:** Os módulos devem ser o mais diferentes possível uns dos outros, mantendo a elegância e a simplicidade;
  - **Mau-cheiro relacionado:** Código Duplicado (lógica repetida em vários lugares).
- **Portabilidade**:
  - **Definição:** O código deve ser adaptável a diferentes ambientes (hardware, SO, etc.) desde o projeto.
  - **Mau-cheiro relacionado:** Código Dependente de Plataforma (uso de funcionalidades específicas de uma plataforma).

Questão 2

Maus-cheiros e Princípios Violados

Duplicação de Código:

- Métodos como *adicionaNomeRendimento*, *adicionaRendimentoTributavel*, e *adicionaValorRendimento* em *IRPF.java* têm lógica duplicada para expandir arrays.
- Similar duplicação ocorre em *Dependentes.java* para adicionar dependentes.

Princípio Violado: Evitar duplicação.

Refatoração: Criar um método genérico para adicionar elementos a um array, que pode ser reutilizado em diferentes contextos.

Complexidade Desnecessária:

- O uso de múltiplas listas para armazenar dados relacionados como rendimentos e deduções pode ser simplificado usando objetos.

Princípio Violado: Simplicidade e Elegância.

Refatoração: Criar classes como *Rendimento* e *Deducao* para encapsular dados relacionados e substituir listas por uma lista de objetos.

Falta de Encapsulamento:

- A classe *Dependentes* expõe diretamente arrays através de métodos *get* e *set*, o que pode levar a modificações indesejadas.

Princípio Violado: Boas interfaces.

Refatoração: Retornar cópias dos arrays ou usar coleções imutáveis para evitar modificações externas.

Métodos Longos e Complexos:

- Métodos como *cadastrarDeducaoIntegral* e *cadastrarDependente* têm lógica que pode ser extraída para métodos auxiliares.

Princípio Violado: Simplicidade e Modularidade.

Refatoração: Extrair métodos para encapsular lógica repetitiva ou complexa.

Nomes de Métodos e Variáveis:

- Nomes como *getdependente* (deveria ser *getDependente*) não seguem convenções de nomenclatura.

Princípio Violado: Elegância.

Refatoração: Renomear métodos e variáveis para seguir convenções de nomenclatura Java.

Refatorações necessárias

Criar Classes *Rendimento* e *Deducao*:

- Crie uma classe *Rendimento* que encapsule *nome*, *tributavel*, e *valor*.
- Crie uma classe *Deducao* que encapsule *nome* e *valor*.

Substituir Listas por Lista de Objetos:

- Substitua as listas *nomeRendimento*, *rendimentoTributavel*, e *valorRendimento* por uma única lista de *Rendimento*.
- Substitua as listas *nomesDeducoes* e *valoresDeducoes* por uma única lista de *Deducao*.

Criar Métodos Auxiliares:

- Criar um método genérico para adicionar elementos a um array ou lista.
- Extrair lógica de adição de dependentes e deduções para métodos auxiliares.

Encapsular Acesso a Dados:

- Retornar cópias dos arrays ou listas em métodos *get* para evitar modificações externas.

Renomear Métodos e Variáveis:

- Corrigir nomes de métodos para seguir convenções de nomenclatura Java.

Revisar e Simplificar Lógica:

- Revisar a lógica de cálculo de impostos e deduções para simplificar e melhorar a legibilidade.