Tópicos Especiais em Plataformas Emergentes

TP3

Henrique Amorim Costa Melo - 17/0144488
Paulo Gonçalves Lima - 17/0122549
Pedro Vítor de Salles Cella - 17/0113060

Brasília, DF
2023

Dentre as 9 características mostradas foram escolhidas 5, para serem descritas e apresentadas melhor com exemplos e comparações, dentre as escolhidas estão:

- Simplicidade;
- Elegância;
- Modularidade (baixo acoplamento e alta coesão);
- Boas interfaces;
- Boa documentação.

Simplicidade

Simplicidade é uma característica desejável em código, pois facilita a manutenção e a compreensão do mesmo. A simplicidade é alcançada através de diversas técnicas, tais como:

- Estrutura: O código deve ser organizado de maneira clara e lógica, com funções e classes pequenas e bem definidas, que realizam tarefas específicas.

- Claridade: O código deve ser escrito de maneira clara e concisa, evitando a utilização de construções complexas ou desnecessárias.
- Coesão: As funções e classes devem ter alta coesão, ou seja, devem realizar apenas tarefas relacionadas entre si.

- Acoplamento: O código deve ter baixo acoplamento, ou seja, as partes do código devem ser independentes entre si, para que possam ser modificadas ou reutilizadas sem afetar outras partes do código.

- Outros efeitos: outras características desejáveis incluem a facilidade de teste e manutenção, e a facilidade de adição de novas funcionalidades.

Em resumo, a simplicidade em um código geralmente se traduz em um código mais fácil de entender, modificar e manter. Isso é especialmente importante em projetos de longa duração ou projetos com muitos desenvolvedores envolvidos.

A característica de simplicidade está intimamente relacionada aos maus-cheiros de código, pois a simplicidade é um dos principais objetivos na identificação e correção dos maus-cheiros. Martin Fowler, em seu livro "Refactoring: Improving the Design of Existing Code", define maus-cheiros como "indícios de que algo está errado no código e que precisa ser refatorado". Alguns dos maus-cheiros de código mais comuns relacionados à simplicidade são:

- Código duplicado: código repetido é um indicador de que o código não está organizado de maneira eficiente e pode ser simplificado.
- Funções longas: funções longas podem ser difíceis de entender e manter, e podem ser simplificadas dividindo-as em funções menores e mais específicas.

- Classes complexas: classes complexas podem ser difíceis de entender e manter, e podem ser simplificadas dividindo-as em classes menores e mais específicas.

- Variáveis com nomes ruins: variáveis com nomes ruins podem tornar o código difícil de entender, e podem ser simplificadas com nomes mais descritivos e claros.

Em resumo, a característica de simplicidade é um objetivo importante na identificação e correção de maus-cheiros de código, pois ajuda a tornar o código mais fácil de entender, modificar e manter. Isso é especialmente importante em projetos de longa duração ou projetos com muitos desenvolvedores envolvidos.

Uma refatoração possível é juntar os métodos checkValorDeducao e checkValorRendimento em um só, pois ambos possuem a mesma lógica. Além disso, o método checkNull pode ser removido, já que sua função é realizada pelos métodos de checagem de valor.

```
checkValor(value) {
  if (value === null || value < 0) throw new ValorInvalidoException();
}

cadastrarRendimentos = (desc, value) => {
  this.checkDescricao(desc);
  this.checkValor(value);
  this.rendimentos.push({ desc, value });
};

cadastrarContribuicaoPrevidenciaria = (desc, value) => {
  this.checkDescricao(desc);
  this.checkValor(value);
  this.contribuicaoPrevidenciaria.push({ desc, value });
};

cadastrarPensaoAlimenticia = (value) => {
```

```
  this.checkValor(value);
  this.pensaoAlimenticia += value;
};

cadastrarDeducoes = (desc, value) => {
  this.checkDescricao(desc);
  this.checkValor(value);
  this.deducoes.push({ desc, value });
};
```

Modularidade

Modularidade é a característica de dividir um sistema em módulos ou componentes independentes, que podem ser facilmente compreendidos, testados e modificados sem afetar o funcionamento do sistema como um todo. A modularidade é importante porque ajuda a manter o código organizado, fácil de entender e fácil de manter.

Alguns dos efeitos da modularidade no código incluem:

- Estrutura: O código é dividido em módulos ou componentes que são fáceis de entender e navegar.

- Claridade: A modularidade ajuda a manter o código claro, pois cada módulo ou componente tem uma única responsabilidade.

- Coesão: Cada módulo ou componente tem alta coesão, ou seja, realiza apenas tarefas relacionadas entre si.

- Acoplamento: A modularidade ajuda a manter o acoplamento baixo, ou seja, os módulos ou componentes são independentes uns dos outros, para que possam ser modificados ou reutilizados sem afetar outros módulos ou componentes.
- Outros efeitos: outras características desejáveis incluem a facilidade de teste, manutenção e adição de novas funcionalidades.

Em resumo, a modularidade é uma característica importante para o código, pois ajuda a manter o código organizado, fácil de entender e fácil de manter. Isso é

especialmente importante em projetos de longa duração ou projetos com muitos desenvolvedores envolvidos.

A característica de modularidade está intimamente relacionada aos maus-cheiros de código, pois a modularidade é um dos principais objetivos na identificação e correção dos maus-cheiros. Alguns dos maus-cheiros de código mais comuns relacionados à modularidade são:

- Acoplamento excessivo: acoplamento excessivo pode tornar o código difícil de modificar e manter, e pode ser simplificado através da melhoria da modularidade do código.

- Código duplicado: código repetido é um indicador de que o código não está organizado de maneira eficiente e pode ser simplificado através da modularidade.

- God Class: é uma classe muito grande, com muitas responsabilidades e muitas dependências, o que torna ela difícil de ser testada, modificada e compreendida.

Em resumo, a característica de modularidade é um objetivo importante na identificação e correção de maus-cheiros de código, pois ajuda a tornar o código mais fácil de entender, modificar e manter, além de tornar o código mais escalável e reutilizável.

Algumas operações de refatoração possíveis que foquem na característica de modulzarização são:

- Remover as constantes aliquota e deducoes: elas não são utilizadas.
- Criar uma classe separada para as regras de cálculo de imposto de renda: essa classe poderia ser responsável por calcular o imposto devido a partir da base de cálculo.
- Remover métodos com duplicidade de código: por exemplo, o método checkDescricao é idêntico ao método checkNome.

Para exemplificar a classe do cálculo do imposto de renda ficaria:

```
class ImpostoRenda:
    def __init__(self, base_calculo):
        self.base_calculo = base_calculo

    def calcular(self):
        if self.base_calculo <= 1903.98:
            return 0
```

```
        elif self.base_calculo <= 2826.65:
            return (self.base_calculo * 0.075) - 142.80
        elif self.base_calculo <= 3751.05:
            return (self.base_calculo * 0.15) - 354.80
        elif self.base_calculo <= 4664.68:
            return (self.base_calculo * 0.225) - 636.13
        else:
            return (self.base_calculo * 0.275) - 869.36
```

Boa Documentação

A boa documentação é a característica de ter um conjunto completo e preciso de documentação associado ao código, que inclui comentários, documentação de código, manuais e outros recursos. A boa documentação é importante porque ajuda a entender o código, facilitando a sua manutenção e evolução.

Alguns dos efeitos da boa documentação no código incluem:

- Estrutura: A documentação ajuda a entender a estrutura do código e como as diferentes partes do código estão relacionadas.

- Claridade: A boa documentação ajuda a explicar o propósito e a funcionalidade do código, tornando-o mais fácil de entender.

- Coesão: A documentação ajuda a entender a coesão do código, ou seja, como as diferentes partes do código estão relacionadas entre si.

- Acoplamento: A documentação ajuda a entender o acoplamento entre as diferentes partes do código, ou seja, como as diferentes partes do código estão relacionadas entre si.

- Outros efeitos: a boa documentação ajuda a guiar novos desenvolvedores, facilita o trabalho em equipe, além de ajudar na manutenção e evolução do código.

Em resumo, a boa documentação é uma característica importante para o código, pois ajuda a entender o código, facilitando a sua manutenção, assim como entender o que de fato um código faz e como ele deve ser instalado e utilizado.

A característica de boa documentação está diretamente relacionada aos maus-cheiros de código. Alguns dos maus-cheiros de código mais comuns relacionados à documentação são:

- Código sem comentários: Código sem comentários pode ser difícil de entender e manter, especialmente para novos desenvolvedores ou para aqueles que trabalham no código depois de um período de tempo.

- Comentários obsoletos ou desatualizados: Comentários obsoletos ou desatualizados podem ser confusos e enganosos, e podem ser removidos ou atualizados para garantir que a documentação seja precisa e útil.

- Código com nomes de variáveis e funções ambíguos: Código com nomes de variáveis e funções ambíguos pode ser difícil de entender e manter, e pode ser refatorado para garantir que os nomes sejam claros e precisos.
- Código com classes ou métodos sem documentação: Classe ou métodos sem documentação podem ser difíceis de entender e usar, e pode ser refatorado para incluir comentários e documentação.

Em resumo, a característica de boa documentação é uma parte importante na identificação e correção de maus-cheiros de código, pois ajuda a garantir que o código seja fácil de entender e usar, especialmente para novos desenvolvedores ou para aqueles que trabalham no código depois de um período de tempo.

Um exemplo de refatoração utilizando de boa documentação é adicionar uma descrição geral da classe IRPF, incluindo informações sobre o que ela faz, como usá-la e quais são suas propriedades e métodos:

```
  /**
   * Classe responsável por cálculo de Imposto de Renda Pessoa
Física (IRPF)
   *
   * A classe `IRPF` permite que você faça o cálculo do seu Imposto
de Renda, mantendo um registro de rendimentos, deduções,
contribuições previdenciárias e pensão alimentícia.
   *
   * Propriedades:
   *  - rendimentos: Array com os rendimentos registrados
   *  - deducoes: Array com as deduções registradas
   *  - contribuicaoPrevidenciaria: Array com as contribuições
previdenciárias registradas
   *  - pensaoAlimenticia: Valor da pensão alimentícia
   *  - dependentes: Array com os dependentes registrados
```

```
   *
   * Métodos:
   *  - cadastrarRendimentos(desc, value): Adiciona um novo
rendimento
   *  - totalRendimentos: Retorna o valor total dos rendimentos
registrados
   *  - cadastrarContribuicaoPrevidenciaria(desc, value): Adiciona
uma nova contribuição previdenciária
   *  - cadastrarPensaoAlimenticia(value): Adiciona o valor da
pensão alimentícia
   *  - cadastrarDependente(name, birth): Adiciona um novo
dependente
   *  - cadastrarDeducoes(desc, value): Adiciona uma nova dedução
```

Elegância

A característica de elegância se refere ao conjunto de qualidades que tornam o código atraente e fácil de entender. Ela é frequentemente associada a soluções simples e elegantes para problemas complexos. O código elegante é geralmente bem estruturado, fácil de ler e compreender, e tem um baixo acoplamento e alta coesão. Alguns dos efeitos da elegância no código incluem:

- Estrutura: O código elegante tem uma estrutura clara e consistente, com classes e métodos bem definidos e organizados.
- Claridade: O código elegante é fácil de ler e compreender, e usa nomes de variáveis e funções claros e precisos.
- Coesão: O código elegante tem alta coesão, ou seja, as classes e métodos têm uma única responsabilidade e trabalham juntos de forma coesa.
- Baixo acoplamento: O código elegante tmem baixo acoplamento, ou seja, as classes e métodos dependem o mínimo possível uns dos outros, o que facilita a manutenção e a extensão do código.
- Simplicidade: O código elegante é simples e direto, evitando soluções complexas ou desnecessárias.
- Reutilizabilidade: O código elegante é fácil de ser reutilizado, pois está bem estruturado e modularizado.

Em resumo, a característica de elegância é um conjunto de qualidades que tornam o código atraente e fácil de entender. O código elegante é geralmente bem estruturado, fácil de ler e compreender, e tem um baixo acoplamento e alta coesão.

A característica de elegância está diretamente relacionada aos maus-cheiros de código, pois a elegância é uma das principais qualidades que devem ser buscadas na refatoração de código para corrigir maus-cheiros. Alguns dos maus-cheiros de código mais comuns relacionados à elegância são:

- Código complexo: Código complexo é difícil de entender e manter, e pode ser refatorado para torná-lo mais simples e elegante.
- Código confuso: Código confuso é difícil de entender e manter, e pode ser refatorado para torná-lo mais claro e legível.
- Código com classes ou métodos muito longos: Classes ou métodos muito longos são difíceis de entender e manter, e podem ser refatorados para torná-los mais curtos e elegantes.
- Código com classe ou métodos com múltiplas responsabilidades: Classe ou métodos com múltiplas responsabilidades são difíceis de entender e manter, e podem ser refatorados para torná-los mais coesos e elegantes.

Para melhorar a elegância do código, algumas mudanças podem ser feitas:

1. Ajuste na formatação:
    - Utilizar a indentação padrão (2 espaços por nível) para facilitar a leitura e entendimento do código.
    - Adicionar espaços entre as operações matemáticas.
    - Adicionar espaços antes e depois de operadores lógicos.
2. Reorganização dos métodos:
    - Colocar métodos relacionados a validações (**checkDescricao, checkNome, checkAniversario, checkNull, checkValorDeducao e checkValorRendimento**) juntos, para facilitar a localização deles.
    - Colocar métodos relacionados a cadastros (**cadastrarRendimentos, cadastrarContribuicaoPrevidenciaria, cadastrarPensaoAlimenticia, cadastrarDependente, cadastrarDeducoes**) juntos, para facilitar a localização deles.
3. Utilização de métodos mais elegantes:
    - No método get **totalDeducoes**, usar o método **reduce** diretamente no array resultante da concatenação dos arrays **deducoes** e **contribuicaoPrevidenciaria**, em vez de primeiro mapear o array.
4. Nomenclatura:
    - Usar nomes de variáveis e métodos que sejam claros e descritivos.
    - Utilizar CamelCase para nomenclatura de métodos e variáveis.
    - Usar o padrão de nomenclatura de classes com letra maiúscula.

Boas Interfaces

A característica de boas interfaces se refere ao conjunto de qualidades que tornam as interfaces de um sistema fáceis de usar e entender. Boas interfaces são geralmente intuitivas, fáceis de aprender e usar, e fornece aos usuários a capacidade de realizar tarefas de forma eficiente. Alguns dos efeitos de boas interfaces no código incluem:

- Consistência: Boas interfaces são consistentes, ou seja, usam os mesmos elementos de interface e convenções para realizar tarefas similares, o que facilita a aprendizagem e o uso.
- Simplicidade: Boas interfaces são simples e diretas, evitando elementos desnecessários ou confusos.
- Feedback: Boas interfaces fornecem feedback claro e imediato para as ações dos usuários, o que ajuda a garantir que as tarefas estejam sendo realizadas corretamente.

Alguns dos maus-cheiros de código mais comuns relacionados às boas interfaces são:

- Classes God: Classes God são classes que possuem muitas responsabilidades e muitos métodos, tornando-as difíceis de entender e manter. Essas classes podem ser refatoradas para torná-las mais coesas e ter interfaces mais limadas
- Métodos longos : Métodos longos são difíceis de entender e manter, e podem ser refatorados para torná-los mais curtos e elegantes, e também para serem mais claros e precisos em suas interfaces.

Uma maneira de refatorar para melhorar a interface do código é criar métodos para validação de inputs, para evitar que ***exceptions*** sejam lançadas durante a execução. Além disso, pode-se criar métodos que retornem informações sobre o cálculo do IRPF em forma de objetos, tornando mais fácil para o usuário acessar as informações relevantes. Algumas formas são:

- Criar métodos **isDescricaoValida, isNomeValido, isAniversarioValido, isValorDeducaoValido** e **isValorRendimentoValido** para validar as entradas antes de adicioná-las às listas correspondentes.

- Criar métodos ***getImposto*** e ***getDadosImposto*** para retornar informações sobre o imposto calculado e os dados usados para o cálculo.