# UNIVERSIDADE DE BRASÍLIA

## Faculdade do Gama

Técnicas de Programação para Plataformas Emergentes

**Daniel Primo de Melo - 180063162**

**Danillo Gonçalves de Souza - 170139981**

**Ithalo Luiz de Azevedo Mendes - 170069800**

**Maicon Lucas Mares de Souza - 180023411**

**Victor Amaral Cerqueira - 170164411**

Brasília, DF

2022

# 1. Simplicidade

## 1.1 Descrição

Simplicidade é, sem dúvidas, a característica primordial de um bom *design* de código. Um código simples é fácil de entender, não possui artefatos desnecessários, mas não é fácil de ser alcançado. Esta última característica se dá devido a quantidade de informação que é necessária ser peneirada, a quantidade de reflexões sobre as decisões a serem tomadas e a quantidade de refatorações até se alcançar um *design* simples.

Os efeitos de um *design* de código simples afetam diretamente um projeto de *software*. Alguns exemplos são:

- A manutenibilidade do projeto torna-se mais fácil devido à clareza na implementação do mesmo
- A *performance* pode aumentar significativamente devido a não haver artefatos desnecessários, como interfaces genéricas com acessos de múltiplas partes
- Os testes são facilitados devido aos esquemas de contratos (entradas e saídas, para cada entrada a saída é conhecida) serem bem definidos
- O acoplamento torna-se baixo porque não há usos desnecessários entre as classes/módulos, apenas o necessário.
- A coesão torna-se alta, pois são explícitas as responsabilidades de cada classe, dessa maneira, funções indevidas não são assumidas.

## 1.2 Maus cheiros de código associados

- Código duplicado: pode indicar pontos do projeto (por exemplo classes) fazendo mais do que deveriam
- Método longo
- Classe grande
- Longa lista de parâmetros

## 1.3 Operações de refatoração necessárias

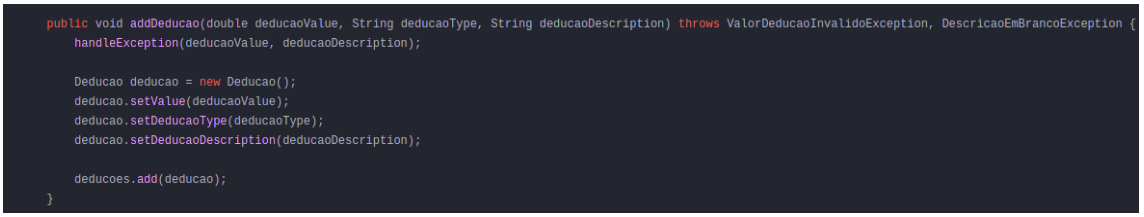
- Extrair método: aplicar quando o código estiver duplicado em uma mesma classe
- Extrair método/puxar para cima: aplicá-la quando o código duplicado

estiver presente em classes irmãs. Dessa maneira, elas podem herdar a partir da classe mãe.

- Dividir método longo em métodos menores: aumenta a coesão (relacionamento entre a classe) e, até possivelmente, o acoplamento (quando aplicada indireção).
- Introduzir objeto-parâmetro ou substituir método por método-objeto: possibilita a simplicidade na assinatura de métodos com longa lista de parâmetros.

Podemos ver nas imagens a seguir a diminuição da lista de parâmetros de um método por meio da operação de substituição de método por método-objeto:

**Imagem 1 - Antes da operação substituição método por método-objeto**




**Fonte:** próprio autor

**Imagem 2 - Após substituição método por método-objeto**

```java
import exception.DescricaoEmBrancoException;
import exception.ValorDeducaoInvalidoException;
import models.Deducao;

public class CalculateTax {
    // Referencia para objeto original
    Irpf _fonte;

    // Atributo para cada parametro
    double deducaoValue;
    String deducaoType;
    String deducaoDescription;

    // Atributo para cada variavel temporaria
    Deducao deducao;

    // Construtor do objeto-metodo
    public CalculateTax(Irpf _fonte, double deducaoValue, String deducaoType, String deducaoDescription){
        this._fonte = _fonte;
        this.deducaoValue = deducaoValue;
        this.deducaoType = deducaoType;
        this.deducaoDescription = deducaoDescription;
    }

    void computar () throws ValorDeducaoInvalidoException, DescricaoEmBrancoException {
        _fonte.handleException(deducaoValue, deducaoDescription);

        deducao = new Deducao();
        deducao.setValue(deducaoValue);
        deducao.setDeducaoType(deducaoType);
        deducao.setDeducaoDescription(deducaoDescription);

        _fonte.deducoes.add(deducao);
    }
}
```

**Fonte:** próprio autor

## 2. Ausência de duplicidades

### 2.1 Descrição

As duplicidades ocorrem quando existem trechos de código repetidos no projeto. A duplicidade pode vir a causar diversos empecilhos para a realização da manutenção de softwares, tanto grandes quanto pequenos, pois causa a necessidade de alterar diversas linhas de código repetidas, além disso, torna mais complexo o entendimento do mesmo possibilitando o surgimento de diversos bugs.

Por exemplo: Imagine um software de uma escola, onde há a necessidade de cadastrar alunos e professores pelo nome, data de nascimento e CPF. Teremos então 2 classes, *Aluno* e *Professor*, ambos possuindo atributos e métodos idênticos. Enquanto o projeto é pequeno é simples de realizar alterações. Agora imagine que, após várias manutenções, seja necessário aplicar as mesmas mudanças para 2 códigos semelhantes, mas com diferentes

finalidades, vai chegar um ponto que será insustentável manter a qualidade e complexidade de ambas as classes, podendo gerar possíveis atrasos para futuras entregas. Para evitar esses problemas, visa-se preservar o código com o mínimo de duplicidade possível.

## 2.2 Maus cheiros de código associados

- Código duplicado
- Classe inchada

## 2.3 Operações de refatoração necessárias

- Extrair método: Extrair comportamento comum de duas classes e unificar. Pode ser realizado a partir de herança de classes, criando o método que está repetindo na classe pai e enviando para as filhas.
- Criação de funções simples, eficientes e genéricas: Criando funções simples, e que realizam somente um tipo de tarefa, possibilita ao desenvolvedor utilizar esse mesmo trecho de código em diversos locais.

# 3. Boa documentação

## 3.1 Descrição

Uma boa documentação é extremamente essencial para toda e qualquer etapa de desenvolvimento desde o planejamento até a conclusão de um projeto. Uma boa documentação fornece informações suficientes para guiar o objetivo da equipe, para manter a rastreabilidade de tarefas, deixar requisitos bem claros para o time e também para auxiliar o programador durante o desenvolvimento.
Não há documentação mais ou menos importante. Todo e qualquer tipo de documentação é essencial e serve a um propósito, desde que seja clara e objetiva. Para a disciplina de Técnicas de Programação para Plataformas Emergentes o foco é a documentação de código e para o código a melhor documentação é sua estrutura limpa (Kernighan Plauger) .
Ao desenvolver um código complexo que o intuito é ser utilizado por terceiros ou que não está sendo feito somente por uma pessoa é estritamente necessário documentar a API ou biblioteca, e seguir os padrões do *Clean Code* para uma arquitetura simples, dessa forma o desenvolvedor estará ajudando

outros, programadores que futuramente irão contribuir para o projeto.

## 3.2 Maus cheiros de código associados

- Toda e qualquer característica que não seguem ao Clean Code (Boa estrutura de código)
    - Classe inchada
    - Lista de parâmetros longa demais
    - Obsessão primitiva
    - Comentários ruins
    - Aglomerados de dados

## 3.3 Operações de refatoração necessárias

- Reduzir as responsabilidades que uma classe tem, o inchaço da mesma é ruim e aumenta a complexidade do código
- Passar uma quantidade menor de parâmetros para dentro das funções e classes.
- Evitar repassar dados primitivos para funções, prefira utilizar objetos. Exemplo um par de coordenadas.
- Utilizar comentários somente quando estritamente necessário. Exemplo de descrever uma  função ou classe muito complexa, evitar comentários redundantes.
- Sempre que houver dados aglomerados avaliar a possibilidade de transformá-los em objetos.

# 4. Modularidade

## 4.1 Descrição

Na engenharia de software, modularidade se refere à extensão na qual um software/aplicativo da Web pode ser dividido em módulos menores. A modularidade do software indica que o número de módulos de aplicativos são capazes de atender a um domínio de negócios especificado.

A modularidade é bem-sucedida porque os desenvolvedores usam código pré-escrito, o que economiza recursos. Em geral, a modularidade

oferece maior capacidade de gerenciamento de desenvolvimento de software. Os problemas de negócios modernos crescem continuamente - em termos de tamanho, complexidade e demanda.

Os requisitos de capacidade de software aprimorados forçam os desenvolvedores a aprimorar os sistemas desenvolvidos com novas funcionalidades.

A modularidade da engenharia de software permite que aplicativos típicos sejam divididos em módulos, bem como integração com módulos semelhantes, o que ajuda os desenvolvedores a usar código pré-escrito. Os módulos são divididos com base na funcionalidade, e os programadores não estão envolvidos com as funcionalidades de outros módulos. Assim, novas funcionalidades podem ser facilmente programadas em módulos separados.

### 4.2 Maus cheiros de código associados

- Classe inchada (com muitas funções)
- Código duplicado

### 4.3 Operações de refatoração necessárias

- Isolamento de funções que se repetem
- Classes mais atômicas

## 5. Portabilidade

### 5.1 Descrição

Portabilidade pode ser definida como a facilidade na qual o software pode ser transferido de um sistema computacional ou ambiente para outro. Em outras palavras, o software é dito portável se ele pode ser executado em ambientes distintos.

De um modo geral, a portabilidade refere-se à habilidade de executar um sistema em diferentes plataformas. É importante observar que à medida que aumenta a razão de custos entre software e hardware, a portabilidade torna-se cada vez mais importante. Adicionalmente, pode-se ter a portabilidade de componentes e a portabilidade de sistemas. Esta última situação pode ser vista como caso especial de reusabilidade. O reuso de software ocorre quando

todo o sistema de software é reutilizado, implementando-o em diferentes sistemas computacionais.

A portabilidade de um componente ou sistema de software é proporcional à quantidade de esforço necessário para que funcione num novo ambiente. Se uma quantidade menor de esforço é exigida quando comparada ao trabalho de desenvolvimento, então o sistema é dito portável.

## 5.2 Maus cheiros de código associados

- Comentários ruins
- Aglomerados de dados
- Código duplicado

## 5.3 Operações de refatoração necessárias

- Comentar apenas quando necessário, evitando redundância.
- Sempre que houver dados aglomerados, estudar a possibilidade de transformá-los em objetos.
- Procurar fazer classes atômicas, para reduzir o inchaço.