UNIVERSIDADE DE BRASÍLIA
FACULDADE UnB GAMA
TÉCNICAS DE PROGRAMAÇÃO EM PLATAFORMAS EMERGENTES

DEFINIÇÕES DE MAL CHEIROS E EXEMPLIFICAÇÕES EM UM CÓDIGO FONTE JAVA

Equipe

Chaydson Ferreira da Aparecida, 211030700
Pedro Henrique Rodrigues de Carvalho, 211031459
Lucas Lopes Frazão (Gerente), 211031771

Brasília-DF,

Agosto 2024

Definições dos princípios de um bom código

Simplicidade

Esta é a característica mais importante de um código bem projetado. Um design simples é fácil de entender, não possui imperfeições ou falhas desnecessárias, e é fácil de implementar. É coerente e consistente. Um código simples é tão pequeno quanto possível, mas não menor. O software deve ter a quantidade mínima de código necessária, e apenas isso.

Elegância

A elegância incorpora os aspectos estéticos do design e muitas vezes anda de mãos dadas com a simplicidade. Significa que o código não é barroco, excessivamente engenhoso ou complexo demais. Um código bem projetado possui uma beleza em sua estrutura.

Modularidade

Decompor em subsistemas, bibliotecas, pacotes, classes, e assim por diante. Cada parte é menos complexa do que o problema original, mas juntas, formam uma solução completa.

Boas interfaces

A fachada pública atrás da qual esconde uma implementação interna. Esse conjunto de operações disponíveis é frequentemente chamado de interface de programação de aplicações (API). É a única rota para a funcionalidade de um módulo, e sua qualidade determina a qualidade desse módulo, pelo menos vista de fora.

Extensibilidade

Um código bem projetado permite que funcionalidades extras sejam inseridas em locais apropriados, quando necessário. A extensibilidade pode ser acomodada através de scaffolding de software: plug-ins carregados dinamicamente, hierarquias de classes cuidadosamente escolhidas com interfaces abstratas no topo, fornecimento de funções úteis de retorno de chamada e até mesmo uma estrutura de código fundamentalmente lógica e maleável.

Evitar duplicação

Um código bem projetado não contém duplicação; nunca precisa se repetir. A duplicação é inimiga do design elegante e simples. Código redundante desnecessário leva a um programa frágil.

Portabilidade

Portabilidade refere-se à capacidade do software de adaptar-se a diferentes ambientes de execução, porém, é necessário avaliar a necessidade de construir um código portátil, pois se não existe tal obrigação o design será afetado de maneira negativa.

Código deve ser idiomático

Um bom código deve seguir corretamente a implementação da linguagem utilizada, adequando-se as melhores práticas, expressões, convenções, operadores etc.

Bem documentado

Um bom design deve ser documentado. A documentação deve ser pequena porque o design é muito simples. Em uma extremidade do espectro, os projetos arquitetônicos são documentados em uma especificação. Na outra extremidade, as funções empregam código auto documentado.

O que o Flower define como mal cheiro

- Código duplicado
- Método longo
- Classes Grandes
- Lista de parâmetro longa
- Alteração divergente
- Cirurgia com rifle
- Inveja dos Dados
- Grupos de Dados
- Obsessão Primitiva
- Comandas Switch
- Hierarquias Paralelas de Herança
- Classe Ociosa
- Generalidade Especulativa
- Campo Temporário
- Cadeias de Mensagens
- Intermediário
- Intimidade Inadequada
- Classes Alternativas com Interfaces Diferentes
- Biblioteca de Classes Incompleta
- Classe de Dados
- Herança recusada
- Comentários

Relacionando mal-cheiros com os seus respectivos princípios

O princípio de **simplicidade** está relacionado com os mal-cheiros:

- Método longo
- Classes Grandes
- Lista de parâmetro longa
- Alteração divergente
- Cirurgia com rifle
- Hierarquias Paralelas de Herança
- Classe Ociosa
- Generalidade Especulativa
- Campo Temporário
- Classe de Dados

O princípio de **elegância** está relacionado com os mal-cheiros:

- Inveja dos Dados
- Hierarquias Paralelas de Herança
- Cadeias de Mensagens

O princípio de **modularidade** está relacionado com os mal-cheiros:

- Intimidade Inadequada
- Classes Alternativas com Interfaces Diferentes

O princípio de **boas interfaces** está relacionado com os mal-cheiros:

- Intermediário

O princípio de **extensibilidade** está relacionado com os mal-cheiros:

- Biblioteca de Classes Incompleta

O princípio de **evitar duplicação** está relacionado com os mal-cheiros:

- Grupos de Dados
- Comandas Switch

O princípio de **código deve ser idiomático** está relacionado com os mal-cheiros:

- Obsessão Primitiva
- Herança recusada

O princípio de **bem documentado** está relacionado com os mal-cheiros:

- Comentários

Avaliação do nosso código da segunda parte do trabalho

Método longo

- **Fere o princípio da simplicidade**
  - População dos controllers serem feitas em uma função a parte.
  - Função createCustomer, colocar a checagem do isPrime para uma função a parte.
  - Na função isSpecial, na SaleController, remover a conversão de Data para uma função a parte.
  - No construtor do Product, extrair as validações para uma função a parte.
  - Na função getCustomerFromUser, extrair o for que busca um customer, para uma função a parte

Classes Grandes

- Fere o princípio da simplicidade

- Na classe Customer, usar o extrair classe (132) e criar uma classe de Contact.

Lista de parâmetro longa

- Fere o princípio da elegância

- Na classe SaleController, calculateTotalValue, estamos passando muitos parametros, devemos usar o Introduzir Objeto Parâmetro (252).

- O mesmo na função calculateShipping

Inveja dos Dados

- Fere o princípio da elegância

- Na SaleController, temos diversas funções que poderiam estar em classes melhores, por exemplo: calculateCashback, getCustomerFromUser, getDiscountFromUser, calculateShipping, calculateTaxes, isSpecialForAllCustomers, isSpecial, verifyCustomer, isCompanyCard, devemos usar o mover Método (125)

Classe de Dados

- Fere o princípio da simplicidade

- Na classe Tax, ela contém apenas dados

- Na classe SaleDetails,

- Na classe Sale

- Na classe Product

- Na classe Customer

- Na classe Address

- Devemos utilizar o Extrair Método (100) e colocar as funções que estão nas controllers, nas models

Referências

FOWLER, Martin. **Refatoração**. Porto Alegre: Grupo A, 2004. E-book. ISBN 9788577804153. Acesso em: 12 ago. 2024.

GOODLIFFE, Pete. **Code craft: the practice of writing excellent code**. No Starch Press, 2007. Acesso em: 12 ago. 2024.