UFV

UNIVERSIDADE FEDERAL DE VIÇOSA - UFV - CAMPUS FLORESTAL

ENGENHARIA DE SOFTWARE II

Análise da Qualidade
Padrões e Diretrizes - Equipe 2
Versão: 1.1

Vitor Vasconcelos - 4255

Florestal - MG
2024

1. Introdução

Este documento estabelece os padrões e diretrizes para o desenvolvimento do sistema, utilizando Java para a lógica de aplicação e MySQL como banco de dados. A padronização garante que o código produzido pela equipe seja coeso, de fácil manutenção e compreensível por todos os membros, além de seguir as melhores práticas de desenvolvimento. Padrões de nomenclatura, formatação e versionamento são essenciais para manter a qualidade, facilitar a colaboração e reduzir a probabilidade de erros no código.

2. Padrões de Codificação em Java

2.1 Nomenclatura

Deve ser utilizado Camel Case como padrão de definição de nomes e não deve ser utilizado hífen (-) e nem subtraço (_)

- Utilizar nomes o mais descritivos possíveis e por isso evitar abreviações.
- Para os métodos, utilizar verbos seguidos de substantivos

2.1.1 Classes e Interfaces

O nome das classes deve começar com letra maiúscula, seguida de letras minúsculas, exceto no início de novas palavras.

- Bons exemplos: class Exemplo / class MeuExemplo
- Maus exemplos: class exemplo / class meu_exemplo

2.1.2 Métodos

O nome dos métodos deve ser em letras minúsculas, exceto no início de novas palavras internas, como sugere o Camel Case.

● Bons exemplos: void comprar() / class comprarProduto()
● Maus exemplos: void Compra() / class ComprarProduto()

2.1.4 Constantes

O nome de uma constante deve ser com letras maiúsculas e é o único caso em que deve ser utilizado subtraço (_) para separar palavras.

● Bons exemplos: static final int ALTURA_MIN = 1

2.2 Indentação e Espaçamento

- Utilizar quatro espaços (tab) para indentação
- Respeitar os escopos dos arquivos
- Utilizar espaço entre operadores e operandos
- Utilizar espaço após vírgulas
- Evitar a criação de linhas muito longas
- Em caso de linhas longas, deve-se quebrar após vírgulas e parentes e no caso de operadores lógicos-aritméticos, quebrar antes do operador

2.3 Comentários

Comentários devem conter apenas informações relevantes para o entendimento do código, fazendo com que sejam sucintos e objetivos mas sem perder poder de interpretação. Para a produção de comentários de múltiplas linhas, o operador de comentário deve ser utilizado (/* */). Já para comentários simples de linha única

deve ser utilizado o operador de comentário simples (///) devidamente espaçado do código para melhorar a legibilidade.

2.4 Composição de Classes

Cada classe deve ser definida em um arquivo, ou seja, um arquivo deve possuir somente uma classes, e para a definição de uma classe, as seguintes configurações devem ser aplicadas:

- Definição da Classe
- Atributos (Devem ser declaradas seguindo a ordem Public, Protected, Private)
- Instâncias
- Método Construtor
- Métodos
- Getters e Setter devem ser declarados no final

Além disso, seguem algumas regras de formatação:

- Não adicionar espaço entre o método e sua lista de parâmetros
- A abertura de chaves “{“ deve ser feita no fim da mesma linha em que foi declarado o código e o fechamento deve ser feito em um linha separada e deve ser alinhada no conjunto do método a qual foi aberta.
- Métodos devem ser sempre separados por uma linha em branco

3. Padrões de Nomenclatura de Banco de Dados

Uma nomenclatura consistente e padronizada é essencial para garantir que a estrutura do banco de dados seja compreensível e fácil de manter. Isso ajuda na comunicação entre os desenvolvedores e na manutenção do sistema a longo prazo, facilitando futuras expansões e otimizações. Abaixo estão as diretrizes que devem ser seguidas para nomear elementos do banco de dados, utilizando a convenção snake_case, onde todas as letras são minúsculas e as palavras são separadas por *underscores*.

3.1 Nomenclatura de Banco de Dados

Os nomes de bancos de dados devem seguir a convenção snake_case e refletir de forma clara o propósito do sistema que estão suportando. Um nome de banco de

dados bem definido ajuda a distinguir sistemas diferentes e a identificar facilmente a função do banco de dados em uma infraestrutura mais complexa.

- Boas práticas: O nome do banco de dados deve ser descritivo, sem abreviações desnecessárias, e deve indicar a funcionalidade do sistema. Exemplo de boa prática:
    - ecommerce_system — Para um sistema de comércio eletrônico.
    - hr_management — Para um sistema de gerenciamento de recursos humanos.
- Más práticas:
    - db1 — Nome genérico e sem indicação do propósito.
    - proj — Abreviação que não deixa claro o contexto ou função.

3.2 Nomenclatura de Tabelas

Os nomes das tabelas devem ser descritivos e utilizar a convenção snake_case. É importante que o nome da tabela descreva o conteúdo ou a entidade que ela armazena, como "cliente", "produto" ou "pedido". Quando possível, use substantivos no singular, para evitar confusões semânticas e manter um padrão uniforme.
Os nomes das tabelas devem ser simples e claros, representando entidades do domínio do sistema. Os nomes das tabelas devem estar no **singular**, e preferencialmente usando palavras da língua portuguesa.

Bom: usuario, post, cargo, quarto_categoria
Ruim: usuarios, posts, cargos, categories

3.3 Nomenclatura de Nomes de Atributos (Colunas)

Os atributos (colunas) de uma tabela devem ser nomeados em snake_case, preferindo sempre nomes descritivos que indiquem claramente o tipo de dado armazenado ou sua função dentro da entidade. Os nomes das colunas devem estar no **singular**.

Exemplo de boa prática:

id_cliente — Representa o identificador único do cliente.
data_pedido — Representa a data em que o pedido foi feito.

Más práticas:

id — Nome muito genérico que pode gerar ambiguidade quando a tabela é referenciada por outras tabelas.

od_dt — Abreviações que dificultam a compreensão do campo.

3.4 Nomenclatura de Chaves Primárias

As chaves primárias devem ser nomeadas seguindo o padrão: id_<nome_da_tabela>. Este padrão garante que o nome da chave primária indique claramente a qual entidade ela pertence, facilitando o entendimento e a criação de relações entre tabelas. Além disso, a consistência na nomeação das chaves primárias ajuda a evitar ambiguidades ao lidar com múltiplas tabelas que possuem colunas com o nome "id". O nome da chave primária deve refletir o nome da tabela à qual pertence, precedido por id_, tornando evidente o relacionamento entre a entidade e seu identificador único.

Exemplo de boa prática:
id_cliente — Chave primária da tabela cliente.
id_pedido — Chave primária da tabela pedido.

Más práticas:
id — Nome genérico e que não indica a que tabela pertence.
clientid — Abreviação desnecessária, dificultando a legibilidade.

3.5 Nomenclatura de Chaves Estrangeiras

As chaves estrangeiras devem seguir o padrão: <coluna_da_chave_primária>_<nome_da_tabela_referenciada>. Esse padrão torna evidente quais tabelas estão sendo referenciadas e quais colunas são usadas para definir a relação. Isso também facilita a leitura e interpretação de consultas SQL complexas que envolvem várias tabelas relacionadas. A chave estrangeira deve indicar claramente a tabela que ela referencia e a coluna da chave primária nessa tabela.

Exemplo de boa prática:

id_cliente — Referencia a chave primária id_cliente da tabela customer.

id_pedido — Referencia a chave primária id_pedido tabela order.

Más práticas:

cust_fk — Nome abreviado que não segue a convenção e dificulta a compreensão.
cliente_id — Inconsistência com o padrão de nomenclatura adotado.

4. Padrão de Pull Request (PR) - GitFlow

Título do Pull Request

O título do PR deve ser claro, objetivo e começar com um verbo no infinitivo. O nome da branch deve seguir a estrutura de branches do GitFlow, que são baseadas nas funcionalidades e correções.

Estrutura do GitFlow:

- feature/: Para novas funcionalidades.
- bugfix/: Para correções de bugs.
- hotfix/: Para correções rápidas em produção.
- release/: Para preparar versões de lançamento.

Exemplos de títulos e nomes de branches:

- Boa prática (feature): feature/addUserAuthentication
- Má prática: new_auth_feature
- Boa prática (bugfix): bugfix/fixLoginValidation
- Má prática: fix_bug

Descrição do Pull Request

A descrição deve ser detalhada, seguindo a estrutura:

1. Contexto: Breve explicação do problema ou da funcionalidade que motivou a criação deste PR.
2. O que foi feito: Detalhe das alterações feitas no código, incluindo o impacto no sistema.
3. Como testar: Passos necessários para testar as modificações, incluindo detalhes de como executar testes manuais ou automatizados.
4. Issue relacionada: Se houver uma issue ou tarefa correspondente, faça referência a ela.
5. Caso haja a necessidade, o uso de imagens, mostrando funcionalidades, testes, é muito importante.