UNIVERSIDADE FEDERAL DE VIÇOSA · UFV - CAMPUS FLORESTAL
CIÊNCIA DA COMPUTAÇÃO
PROJETO INTEGRADOR 2024

Análise da Qualidade

Padrões e Diretrizes - Equipe 4

v.1.1.0

Carlos Márcio Moreira Costa - 5079
Stefan Taiguara Couperus Leal - 5066

Florestal - MG
Outubro de 2024

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Este documento especifica os padrões e diretrizes a serem seguidos durante o desenvolvimento do produto para garantir sua qualidade. As especificações se aplicam ao código Java a ser produzido, à nomenclatura do banco de dados, à escrita dos testes e ao fluxo de uso do GitHub.

2 ATA DE REUNIÃO

Template da Ata de Reunião utilizada no Discord:

```
Unset
**Data X:**
**Tema:** Descrição do tema da reunião.
**Participantes:**: Participante 1, Participante 2,
Participante N...
**Modalidade:** Tipo da modalidade (Presencial/Online).
```

Ata de reunião exemplo:

```
Unset
**Data 21/10/2024:**
**Tema:** Discussão de assuntos importantes.
**Participantes:**: Fulano, Ciclano, Beltrano
**Modalidade:** Online.
```

3 PADRÕES DE CODIFICAÇÃO EM JAVA

A utilização de conversões de codificação tem como objetivo aumentar a legibilidade do código e a produtividade do programador. Os itens abaixo especificam um conjunto de regras e padrões a serem seguidos. É importante ressaltar que todo o código produzido pela equipe deve estar na língua portuguesa para melhorar o entendimento de toda equipe, isso inclui o banco de dados, e a

única exceção é se for especificado algum padrão de nomenclatura que deve ser utilizado em inglês.

3.1 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.

Além dos padrões gerais que devem ser seguidos sempre que for definida uma nomenclatura, segue algumas especificações:

3.1.1 CLASSES E INTERFACES

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

Sintaxe:

```
Java
class {[A-Z][a-z]}{}
class XxxxxXxxx{}
```

- **Bons exemplos:**

```
Java
class Produto{}
class ProdutoVendido{}
```

- **Maus exemplos:**

```
Java
class produto{}
```

```
class PRODUTO_vend{}
```

3.1.2 MÉTODOS

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

Sintaxe:

```
Java
tipo {[a-z][A-Z]}();
tipo xxxxxXxxx();
```

- **Bons exemplos:**

```
Java
void comprar();
void comprarProduto();
```

- **Maus exemplos:**

```
Java
void Compra();
void vender_PRODUTO();
```

3.1.3 VARIÁVEIS

O nome de uma variável deve ser curto e com significado claro e deve ser em letras maiúsculas, exceto no início de novas palavras internas, como sugere o Camel Case. Além disso, nome de variáveis com uma única letra só devem ser usados em variáveis com âmbito reduzido, como no caso dos índices utilizados no contexto de iteradores, como por exemplo, i, j e k.

Sintaxe:

```
Java
tipo {[a-z][A-Z]};
tipo xxxxxXxxx;
```

- **Bons exemplos:**

```
Java
int i;
char sexo;
```

- **Maus exemplos:**

```
Java
int iterador;
float saldo_Caixa;
```

3.1.4 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.

Sintaxe:

```
Java
tipo {[A-Z]};
tipo XXXXX_XXXX;
```

- **Bom exemplo:**

```
Java
static final int ALTURA MIN = 10;
```

3.2 Indentação e Espaçamento

- Utilizar quatro espaços para indentação;
- Utilizar espaço entre operadores e operandos;
- Utilizar espaço após vírgulas;
- Evitar a criação de linhas muito longas (mais de 80 caracteres):
    - Em caso de linhas longas, deve-se quebrar após vírgulas e parênteses e, no caso de operadores lógicos-aritméticos, quebrar antes do operador.

3.3 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.

3.3.1 EXEMPLO DE COMENTÁRIOS

```
Java
/*
 * Comentário
 * de Multiplas
 * Linhas
 */

public metodoX(parametroA, parametroB) {
    if (parametroA != parametroB) {       //comentario simples
        …
    }
}
```

3.4 Composição de Classes

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

3.4.1 CONFIGURAÇÕES

- Definição da Classe;
- Variáveis:
    - Devem ser declaradas seguindo a ordem **public**, **protected**, **private**.
- Instâncias;
- Método Construtor;
- Métodos:
    - Getters e Setters devem ser declarados primeiro;
    - Antes de cada método deve ser feita sua documentação como especificado no item 3.1.

3.4.2 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 uma linha separada, alinhada ao conjunto do método a qual foi aberta;
- Métodos devem ser sempre separados por uma linha em branco.

4 PADRÕES DE NOMENCLATURA DE BANCO DE DADOS

Alguns critérios gerais devem ser seguidos ao nomear objetos no banco de dados:

- Utilizar caracteres alfanuméricos
- Não utilizar subtraço (_) ou hífen (-) para separação de palavras; a separação deve ser feita por meio da utilização de letras maiúsculas na

primeira letra de novas palavras. Só se deve utilizar subtraço (_) na especificação de um objeto, por exemplo:
    - TB (para tabelas);
    - PK (para chaves primárias).
- Não utilizar caracteres especiais ou acentuação;
- Não utilizar proposições (de, o, a, etc.);
- Acrônimos de siglas devem ser utilizados apenas com letras maiúsculas, por exemplo: CPF, CEP, etc.

4.1 Nomenclatura do banco de dados

O nome do banco de dados deve identificar o negócio ou a sigla da aplicação.

Sintaxe:

```
Unset
[a-z] xxxx
```

4.1.1 REGRAS

- Utilizar apenas letras minúsculas;
- Utilizar no máximo 15 caracteres.

4.2 Nomenclatura de tabelas

O nome de cada tabela deve ter um significado sugestivo ao entendimento da mesma.

Sintaxe:

```
Unset
TB_{[A-Z][a-z]}
TB_XxxxXxxxx
```

4.2.1 REGRAS

- Utilizar o prefixo “TB ”;
- Utilizar letras maiúsculas na primeira letra de cada palavra que forme o nome da tabela;
- Evitar utilizar abreviações;
- O nome da coluna deve estar no singular.

Exemplo: **TB_Pessoa** (Tabela de pessoas).

4.3 Nomenclatura de nomes de atributos (colunas)

O nome de cada atributo (coluna) deve ter um significado sugestivo ao entendimento do mesmo.

Sintaxe:

```
Unset
[A-Z][a-z]
XxxxXxxxx
```

4.3.1 REGRAS

- Utilizar letras maiúsculas na primeira letra de cada palavra que forme o nome do atributo;
- O nome do atributo deve estar no singular;
- Individualizar o nome do atributo dentro do mesmo ambiente.

Exemplos: **nomeFuncionario**, **dataPartida**.

4.4 Nomenclatura de Chaves Primárias

Deve-se utilizar a mesma semˆantica utilizada para os atributos, como especificado no item 4.3, adicionando apenas o prefixo “PK_”.

Sintaxe:

```
Unset
PK_{[A-Z][a-z]}
PK_XxxxXxxxx
```

Exemplo: **PK_Pessoa** (Chave primária da tabela Pessoa).

4.5 Nomenclatura de Chaves Estrangeiras

Para as chaves estrangeiras, também deve-se utilizar a mesma semântica utilizada para os atributos, como especificado no item 4.3, mas, além de adicionar "FK_", é preciso adicionar também o nome da tabela de origem antes de identificar o nome da chave.

Sintaxe:

```
Unset
FK_{[A-Z][a-z]}_{[A-Z][a-z]}
FK_<TabelaOrigem>_<NomeChave>
```

Exemplo: **FK_Pessoa_CPF** (Chave estrangeira vinda da tabela Pessoa).

5 PADRÕES PARA CRIAÇÃO DE TESTES DE UNIDADE

O nome das classes de testes automatizados, assim como seus métodos, devem ter o prefixo "teste", seguindo os padrões de nomenclatura especificados, respectivamente, nos itens 3.1.1 e 3.1.2. Além disso, os testes unitários automatizados devem possuir comentários de acordo com os casos de teste elaborados especificando: DADO, QUANDO e ENTÃO, como no exemplo abaixo:

```java
@Test
public void testeInserirAluno(){
    // DADO que o jogo é iniciado e a tela para digitar o nome foi acionada
    controleAluno = new ControleAluno();

    // QUANDO a função para inserir for chamada
    Aluno aluno1 = new Aluno(nomeAluno: "Teste");
    Aluno alunoInserido = controleAluno.inserirAluno(aluno1);

    // ENTÃO o aluno deve ser inserido com sucesso
    assertNotNull(alunoInserido);

    // E o nome deve ser exatamente o mesmo
    assertEquals(aluno1.getNome(), alunoInserido.getNome());
}
```

6 PADRÕES PARA USO DO GITHUB

6.1 Git Flow

Git Flow é um fluxo de trabalho para o Git para facilitar o processo de desenvolvimento. A imagem a seguir mostra a estrutura do fluxo do Git Flow.

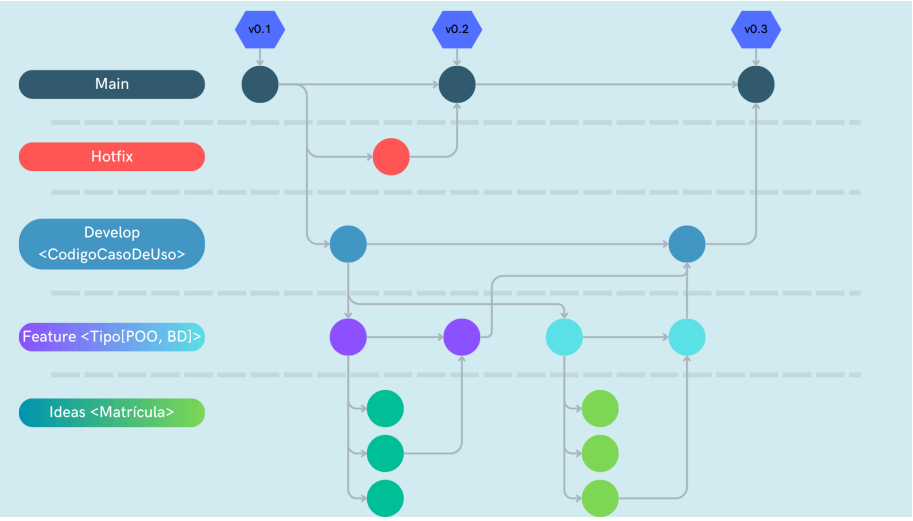

Figura 1: Exemplo de Git Flow que será usado no projeto.

No projeto vamos utilizar esse fluxo, sendo que para cada caso de uso deverá ser criada uma branch **Develop: <CodigoCasoDeUso>** em que os desenvolvedores terão que criar a partir dela um branch \texttt{Feature} para cada tarefa relacionada ao caso de uso com o nome **Feature: <Tipo[POO, BD]>**, as ideias devem seguir esse estilo de branch **Ideias: <Matricula>**.

6.2 Commit

Cada commit realizado nesta branch deverá seguir o padrão **tag: SintezeDoQueFoiFeito**. Em que as tags são:

- **Feat:** Função nova;
- **Refactor:** Refatorar código existente;
- **Test:** Fazer testes automatizados;
- **Fix:** Corrigir bug;
- **Docs:** Documentação.

Exemplos:

Feat: Classe Pessoa adicionada.

- Commit pelo terminal:

```
Unset
~$ git commit -m "Feat:Classe Pessoa adicionada"
```

6.3 Pull Request

Sempre que uma tarefa for finalizada, um Pull Request para a branch do caso de uso deve ser aberto e direcionado para o Desenvolvedor Sênior revisar, e sempre que um caso de uso for finalizado, um Pull Request para a Develop deve ser aberto e os Analistas de Qualidade devem ser adicionados como revisores.

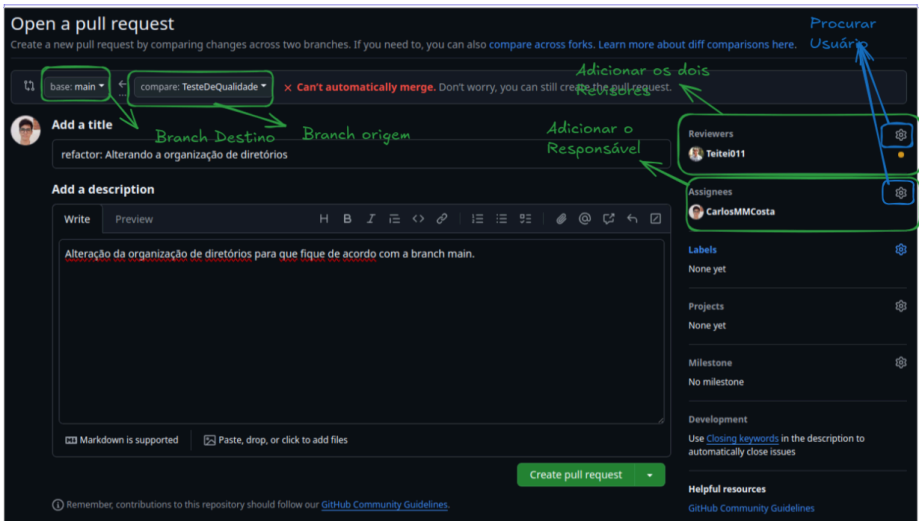

Figura 2: Criação do pull request.

- Branch destino: Branch para onde as mudanças serão direcionadas;
- Branch origem: Branch onde foram feitas as mudanças;
- Revisores: Pessoas que farão a revisão e aceitarão o pull request;
- Responsável: Pessoa responsável pela atribuição da tarefa (geralmente um Dev Sênior).
- Título: Resumo do que foi feito. Respeitando as indicações da seção 6.2.

6.3.1 PULL REQUEST DE TAREFA CONCLUÍDA

- Branch de Destino: *Develop: <CodigoCasoDeUso>;*
- Branch de Origem: *Feature: <CodigoDaTarefa>;*
- Título: *<Código da Tarefa>*: Breve descrição do código;
- Descrição: Um resumo do que foi feito, pontos importantes que precisam ser revisados, comentários importantes e dúvidas existentes;
- Revisores: Eduardo Antunes e Gabriel Zoéga.

6.3.2 PULL REQUEST DE CASO DE USO CONCLUÍDO

- Branch de Destino: main (**Somente Desenvolvedores Seniores**);
- Branch de Origem: Develop <CódigoDeCasoDeUso>;
- Título: <CódigoCasoDeUso> junto de uma breve descrição;
- Descrição: Escreva resumidamente o que foi feito, pontos importantes que precisam ser revisados, algum comentário importante e dúvidas existentes;

- Revisores: Carlos Marcio e Stefan Taiguara.

7 PADRÕES PARA ABERTURA DA ISSUE

7.1 Estrutura das Issues

7.1.1 NOMENCLATURA

As issues devem seguir o padrão de nomenclatura:

```
Unset
[TIPO]: Descrição concisa.
```

Onde TIPO pode ser:

- **Feat:** Nova funcionalidade;
- **Fix:** Correção de bug;
- **Docs:** Documentação;
- **Test:** Testes;
- **Refactor:** Refatoração.

7.1.2 TEMPLATE PADRÃO

```
Unset
##  Descrição
[Descrição detalhada do objetivo]

### Contexto
[Contexto e motivação]

### Comportamento Esperado
[Resultado esperado após implementação]
```

```
## ✅ Critérios de Aceitação
- [ ] Critério 1
- [ ] Critério 2

##  Tarefas
- [ ] Tarefa 1
- [ ] Tarefa 2

##  Links Relevantes
- Link 1
- Link 2
```

7.2 Critérios de Priorização

7.2.1 PRIORIDADE ALTA

- Bugs em funcionalidades críticas;
- Features essenciais para release;
- Questões de segurança.

7.2.2 PRIORIDADE MÉDIA

- Melhorias significativas;
- Bugs em funcionalidades não críticas;
- Novas features importantes.

7.2.3 PRIORIDADE BAIXA

- Melhorias cosméticas;
- Otimizações menores;
- Documentação não essencial.

7.2.4 CRIAÇÃO DE ISSUES

1. Selecionar template apropriado;
2. Preencher todas as seções obrigatórias;
3. Adicionar labels relevantes;
4. Definir prioridade;
5. Atribuir responsáveis quando aplicável.

7.2.5 DESENVOLVIMENTO

1. Criar branch seguindo padrão;
2. Referenciar número da issue nos commits;
3. Atualizar status regularmente.

7.2.6 REVISÃO E FECHAMENTO

1. Verificar critérios de aceitação;
2. Realizar code review quando aplicável;
3. Atualizar documentação;
4. Vincular PR (Pull request) relacionado.

7.3 Exemplos

7.3.1 ISSUE DE FEATURE

```
Unset
[FEAT] - Implementar Sistema de Notificações

##  Descrição
Implementar sistema de notificações push para
usuários mobile
```

```
### Contexto
Precisamos notificar usuários sobre novos eventos

### Comportamento Esperado
Usuários devem receber notificações em tempo real

## ✅ Critérios de Aceitação
- [ ] Configuração do Firebase
- [ ] Implementação do serviço
- [ ] Testes de integração
```

7.3.2 ISSUE DE BUG

```
Unset
[FIX] - Corrigir Erro no Login

##  Descrição
Usuários não conseguem fazer login após atualização

### Passos para Reproduzir
1. Acessar página de login
2. Preencher credenciais
3. Clicar em "Entrar"

### Comportamento Esperado
Login deve ser realizado com sucesso
```

7.4 Boas práticas

- Manter descrições claras e objetivas;
- Atualizar status regularmente;
- Documentar decisões importantes;
- Usar referências cruzadas;

- Manter issues atualizadas.

7.5 O que evitar

- Títulos vagos ou genéricos;
- Descrições incompletas;
- Falta de critérios de aceitação;
- Múltiplos problemas na mesma issue;
- Labels inconsistentes.