Universidade Federal da Bahia
Data: 29 de Outubro de 2018
Discentes: Enrique Wicks, Mateus Barbosa e Mateus Carvalho
Docente: Rita Suzana Pitangueira Maciel

# Padrões de Codificação Sistema de Matrículas

## 1. INTRODUÇÃO

### 1.1. Propósito

A padronização do código possibilita uma maior organização no processo de desenvolvimento, a partir do momento em que facilita a validação, a comunicação entre equipes de desenvolvimento, leitura, manutenção e retirada de bugs.

### 1.2. Escopo

O padrão de codificação é válido para o projeto Sistema de Matrículas.

## 2. CONVENÇÃO DE NOMENCLATURA

### 2.1. Regra Geral

a) Utilize descrições significativas para um nome.
b) Utilize terminologia correlata ao sistema.
c) Não use nomes diferentes de variáveis, classes, interfaces apenas alterando letras maiúsculas e minúsculas.

### 2.2. Pacotes

a) O nome do pacote deve seguir o padrão: “br.com.nome_da_empresa.nome_do_projeto.nome_do_pacote”.

### 2.3. Classes

a) O nome da classe segue o formato “CNomeDaClasseNomeDoPacote”
b) Toda classe começa com o caracter ‘C’. Obrigatoriamente em maiúsculo.
c) O “NomeDaClasse” deve ser um substantivo no singular.
d) O “NomeDoPacote” deve ser o nome do pacote que a classe pertence.
e) O nome de classes deve seguir a prática CamelCase com a primeira letra maiúscula.

### 2.4. Interfaces

a) O nome da classe segue o formato “INomeDaInterfaceNomeDoPacote”
b) Toda classe começa com o caracter ‘I’. Obrigatoriamente em maiúsculo.
c) O “NomeDaInterface” deve ser um substantivo no singular.
d) O “NomeDoPacote” deve ser o nome do pacote que a interface pertence.
e) O nome de interfaces deve seguir a prática CamelCase com a primeira letra maiúscula.

### 2.5. Métodos

a) Deve seguir o formato “verboComplementoNomeDaClasse”.
b) A primeira palavra deve ser um verbo no imperativo.

c) O “Complemento” é livre desde que sejam termos significativos e não sejam conflitantes com nenhuma regra neste documento.
d) O “NomeDaClasse” é o nome da classe que aquele método pertence.
e) Deve seguir a prática CamelCase com a primeira letra em minúsculo.

### 2.5. Constantes

a) O nome deve ser todo em em maiúsculo e com palavras separadas por underscore “_”.

### 2.6. Variáveis

a) Devem ter nome curto e intuitivo.
b) Evitar caracteres ‘_’ e ‘$’.
c) Deve seguir a prática CamelCase com a primeira letra em minúsculo.

## 3. COMENTÁRIOS

### 3.1. Comentários Obrigatórios

a) Comentários: as classes e interfaces devem ser precedidas de comentários para indicar a função que desempenham, nome e autor.
   - Classe:
     ```
     /*
      * Nome:
      * Descrição:
      * Autor:
      */
     ```
   - Interface:
     ```
     /*
      * Nome:
      * Descrição:
      * Autor:
      */
     ```

### 3.2. Quantidade de Comentários

a) Comentários não devem ser usados em excesso (sem violar o item **3.1**).
b) O comentário “//” não deve ser utilizado duas vezes seguidas, exceto para comentar linhas de código.

### 3.3. Formatação de Comentários

a) Comentários curtos podem ser feito em uma única linha.
b) Comentários não devem conter caracteres especiais.
c) Comentários curtos seguidos devem ser alinhados:
   - ```
     if(a == 2) {
             max = 4;          /* Comment  */
             return TRUE;    /* Comment  */
     }
     ```

## 4. INDENTAÇÃO
     ```

### 4.1 Regras Gerais

a) A indentação deve ser feita usando 4 espaços ou 1 tab.

## 5. APRESENTAÇÃO

### 5.1. Parênteses

a) Os parênteses de abertura da implementação de classes, métodos, blocos condicionais e blocos de loop devem estar na mesma linha da assinatura e sem espaços em branco entre eles.

### 5.2. Espaços

a) Não inserir um espaço entre o primeiro parâmetro e o parêntese de abertura de um método.
b) Não inserir um espaço entre o último parâmetro e o parêntese de fechamento de um método.
c) Deve haver 1 espaço em branco após cada parâmetro de um método.
d) Separar termos de operações aritméticas e de comparação com 1 espaço em branco:
    - Evitar: “var2= var1+4 ;”
    - Melhor: “var2 = var1 + 4;”

### 5.3 Linhas

a) Evitar linhas de código com muitos caracteres (não mais que 70) para facilitar a leitura.
b) Uma linha de código deve ser quebrada em pontos específicos, como após um ponto e vírgula, operador ou fechamento de parêntese.
c) Métodos devem ser separados por uma linha em branco.
d) Separar declarações de pacotes e importes com uma linha em branco.
e) Não devem existir linha em branco separando declarações de pacotes.
f) Não devem existir linha em branco separando imports.

## 6. COMPOSIÇÃO DE CLASSES E INTERFACES

### 6.1. Comentário Sobre a Classe ou Interface

a) Ver item **3.1.**

### 6.2. Pacotes e importes

a) Primeiro devem ser declarados os pacotes para posteriormente os importes.
b) Evitar importações com *, para não obter classes que não serão utilizadas.

### 6.3. Outros Elementos

Os outros elementos que serão declarados dentro deste item devem respeitar as regras a seguir e serem organizados na mesma ordem.

a) Variáveis de classe - devem ser declaradas seguindo a ordem de modificadores de acesso: *public*, *protected*, *private.*
b) Variáveis de instância - devem ser declaradas seguindo a ordem de modificadores de acesso: *public*, *protected*, *private.*

c) Método construtor.
d) Demais métodos: aparecem após o construtor e são organizados pelo nível da funcionalidade que desempenham.

## 7. COMPLEXIDADE

### 7.1. Classe ou Interface

a) Evitar classes ou interfaces com mais de 30 métodos.

### 7.1. Métodos

a) Evite funções com mais de 150 linhas.
b) Um método não deve possuir mais de 5 instruções encadeadas dos tipo *if-then-else*, *while*, *for*, *switch*.

## Referências

MAZZI, C. Convenções de Código Java. Disponível em: <https://www.devmedia.com.br/convencoes-de-codigo-java/23871>. Acesso em: 27 de out. de 2018.