

CCF 322 - Engenharia de Software II

Padrões de Codificação

“Reinados da Web”

<Versão 2.0>

Esses serão os padrões de codificação que deverão ser seguidos por todos os membros da equipe. Esse conjunto de regras têm como o objetivo padronizar e facilitar o entendimento e análise de códigos em Java feitos pela equipe. É necessário a compreensão e a utilização das regras abaixo.

Sumário

1. Regras de Nomeação Geral	3
2. Regras de Nomeação para Variáveis	3
3. Regras de Nomeação para Funções	4
4. Regras de Nomeação para Classes	5
5. Regras de Nomeação para Interfaces	6
6. Regras de Nomeação para Constantes	6
7. Regras de Nomeação para Pacotes	7
8. Regras de Nomeação para Enumerações	8
9. Regras de Padronização para Comentários	9
10. Regras de Indentação	12
11. Referências e leituras recomendadas	14

1. Regras de Nomeação Geral:

Servem para o nome de quaisquer identificadores, sendo eles classes, funções, métodos ou variáveis.

- Não usar acentos, pontuação, caracteres especiais e de outras linguagens.
 - Exemplo de nomeações aceitáveis: **Classe.numero**, **Maquina.circuito**, **Fruta.maca**
 - Exemplo de nomeações não aceitáveis: **Classe.número**, **Máquina.circuito**, **Fruta.maçã**
- Evitar abreviações na nomeação de identificadores. O nome deve ser claro para indicar o propósito do elemento, portanto use palavras completas sempre que possível.
 - Exemplo de nomeações aceitáveis: **Matematica.exponencial**, **Restaurante.numeroDeMesas**
 - Exemplo de nomeações não aceitáveis: **Mat.exp**, **Rest.nMesas**

2. Regras de Nomeação para Variáveis:

- É convencional utilizar o estilo de nomenclatura “camelCase” para nomear as variáveis, onde a primeira letra é minúscula, seguida de letras maiúsculas para toda letra que inicia uma outra palavra.
 - Exemplo de nomeações aceitáveis: **resultadoNota**, **precoProduto**

- Exemplo de nomeações não aceitáveis:
ResultadoNota, precoproduto
- Os nomes das variáveis devem ser sempre relevantes com o que armazenam, evitar o uso de palavras aleatórias ou letras singulares para as nomear.
 - Exemplo de nomeações aceitáveis: **nota, preco**
 - Exemplo de nomeações não aceitáveis: **x, y**
- A única exceção para a nomeação de variáveis com nomes com uma letra só são as variáveis temporárias utilizadas em laços de repetição. Para elas, é aconselhável seguir um padrão alfabético à medida da profundidade do laço, começando pela letra “x”.



```
1  for (int x = 0; x < 3; x++){
2      for (int y = 0; y < 3; y++){
3          for (int z = 0; z < 3; z++){
4              for(int w = 0; w < 3; w++){
5                  //código
6              }
7          }
8      }
9  }
10
```

Figura 1. Uso de variáveis temporárias com base em profundidade

3. Regras de Nomeação para Funções:

- É convencional utilizar o estilo de nomenclatura “camelCase” para nomear as funções, onde a primeira letra é minúscula, seguida de letras maiúsculas para toda letra que inicia uma outra palavra.

- Exemplo de nomeações aceitáveis: **fazerCalculo()**, **calcularPreco()**
 - Exemplo de nomeações não aceitáveis:
FazerCalculo(), **calcularpreco()**
-
- Os nomes das funções devem ser sempre relevantes com o que retornam/fazem, evitar o uso de palavras aleatórias ou letras singulares para as nomear.
 - Exemplo de nomeações aceitáveis: **calcularPreco()**, **depositar()**
 - Exemplo de nomeações não aceitáveis: **x()**, **y()**
 - Os nomes das funções devem ser preferencialmente iniciados por verbos, para indicar a ação que fazem, as outras palavras que compõem o nome das funções tendem a ser substantivos, compondo o objetivo da função.
 - Exemplo de nomeações aceitáveis: **receberAlgo()**, **calcularNumero()**
 - Exemplo de nomeações não aceitáveis: **recebeAlgo()**, **calculaNumero()**

4. Regras de Nomeação para Classes:

- É convencional utilizar o estilo de nomenclatura “PascalCase” para nomear as classes, onde todas as letras iniciais das palavras que compõem o nome da interface devem ser maiúsculas.
 - Exemplo de nomeações aceitáveis: **Pessoa**, **PessoaJuridica**
 - Exemplo de nomeações não aceitáveis: **pessoa**, **pessoaJuridica**
- Também é convencional que o nome das classes sejam substantivos, evitando palavras-ligadas ou hífens. Também nunca devem ser substantivos no plural, sendo sempre entidades ou conceitos singulares.
 - Exemplo de nomeações aceitáveis: **Celular**, **CaixaEletronico**
 - Exemplo de nomeações não aceitáveis: **Celulares**, **FazerTransacoes**
- Os nomes das classes devem ser sempre relevantes com o que fazem/armazenam, evitar o uso de palavras aleatórias ou letras singulares para as nomear.
 - Exemplo de nomeações aceitáveis: **Pessoa**, **Cachorro**
 - Exemplo de nomeações não aceitáveis: **P**, **C**

5. Regras de Nomeação para Interfaces:


- É convencional utilizar o estilo de nomenclatura “PascalCase” para nomear as interfaces, onde todas as letras iniciais das palavras que compõem o nome da interface devem ser maiúsculas.
 - Exemplo de nomeações aceitáveis: **Veiculo, DispositivoMobile**
 - Exemplo de nomeações não aceitáveis: **veiculo,dispositivoMobile**
- Também é convencional que o nome das interfaces sejam substantivos, evitando palavras-ligadas ou hífen. Também nunca devem ser substantivos no plural, sendo sempre entidades ou conceitos singulares.
 - Exemplo de nomeações aceitáveis: **Produto, Pedido, Compra**
 - Exemplo de nomeações não aceitáveis: **Produtos, Pedido-De-Compra, Compras**
- As interfaces por natureza são objetos mais gerais, que são implementadas por uma ou mais classes (preferencialmente por mais de uma). Portanto, a nomeação da interface deve remeter a essa característica.
 - Exemplo de nomeações aceitáveis: **Veiculo, DispositivoMobile**
 - Exemplo de nomeações não aceitáveis: **FordKa, CelularSamsungS9**

6. Regras de Nomeação para Constantes:

- É convencional utilizar o estilo de nomenclatura “CONSTANT_CASE” para nomear as constantes, onde todas as letras das palavras que compõem o nome da interface devem ser maiúsculas. Além disso, as palavras que compõem o nome devem ser separadas pelo caractere “_” (underline).
 - Exemplo de nomeações aceitáveis: **MAX_VALUE**, **RECT_LENGTH**
 - Exemplo de nomeações não aceitáveis: **MaxValue**, **RECTLENGTH**

7. Regras de Nomeação para Pacotes:

- É convencional que o prefixo do nome do pacote deve ser único para evitar colisão de nomes. Ex: um pacote “**com.empresa1.models.User**” é único, mas caso alguém criar um outro pacote “**com.empresa2.models.User**” ele passa a ser um possível problema, pois pode haver colisão de nomes, caso ocorrer a seguinte situação.



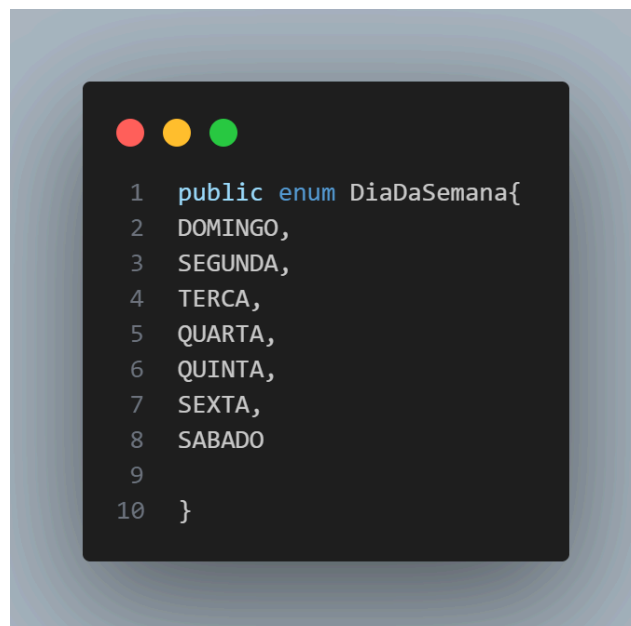
```
1 import com.empresa1.models.User; // Importando User da empresa 1
2 import com.empresa2.models.User; // Tentando também importar User da empresa 2
3
4 public class Exemplo {
5     public static void main(String[] args) {
6         User user1 = new User(); // Aqui o compilador não saberá qual User usar
7         User user2 = new User(); // O mesmo problema acontece aqui
8     }
9 }
10
```

Figura 2. Exemplo de um código com colisão de nomes

- O nome do pacote deve em geral ser escrito em letras minúsculas, lembrando que o “User” nesse caso é a classe que está sendo importada de dentro do pacote, portanto, seguirá as regras de nomenclatura da classe e não do pacote.

8. Regras de Nomeação para Enumerações:

- É convencional utilizar o estilo de nomenclatura “PascalCase” para nomear as enumerações, onde todas as letras iniciais das palavras que compõem o nome da interface devem ser maiúsculas.
- Os objetos dentro das enumerações seguem as mesmas regras de nomenclatura das constantes.

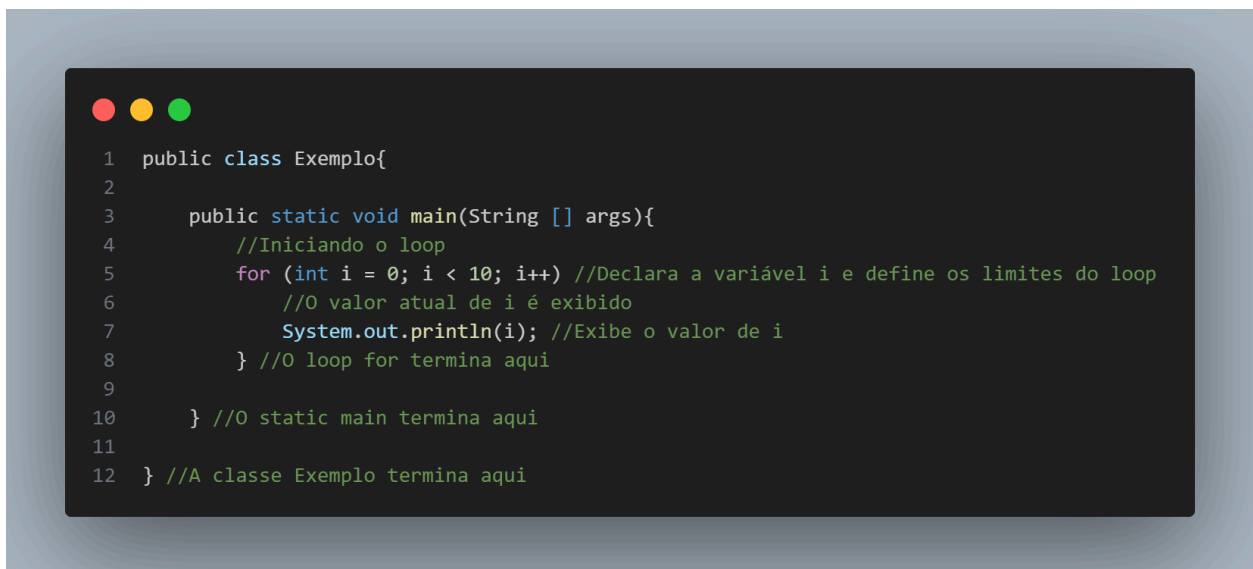


```
1 public enum DiaDaSemana{
2     DOMINGO,
3     SEGUNDA,
4     TERCA,
5     QUARTA,
6     QUINTA,
7     SEXTA,
8     SABADO
9
10 }
```

Figura 3. Maneira apropriada de nomenclatura de enumeração

9. Regras de Padronização para Comentários:

- Comentários devem ser utilizados somente para facilitar a compreensão de um trecho de código. Comentários que são utilizados para explicar termos simples que, por natureza, já são autoexplicativos, não devem ser utilizados pois podem dificultar o entendimento geral do código.



```
1 public class Exemplo{
2
3     public static void main(String [] args){
4         //Iniciando o loop
5         for (int i = 0; i < 10; i++) //Declara a variável i e define os limites do loop
6             //O valor atual de i é exibido
7             System.out.println(i); //Exibe o valor de i
8         } //O loop for termina aqui
9
10    } //O static main termina aqui
11
12 } //A classe Exemplo termina aqui
```

Figura 4. Exemplo de um trecho de código com comentários em excesso

- Comentários, quando necessários, devem ser breves e, ao mesmo tempo, explicar o trecho de código de forma eficiente.

```
1
2 public class Exemplo {
3
4     public static void main(String[] args) {
5         /*
6          * Esse loop é utilizado para iterar sobre um conjunto de números inteiros.
7          * Ele começa com a variável i inicializada em 0.
8          * O loop continuará a executar enquanto o valor de i for menor do que 10.
9          * A cada iteração, o valor dessa variável é incrementado em 1.
10         * Na primeira iteração, o valor será 0,
11         * na segunda iteração, o valor será 1, e por aí vai,
12         * até que o valor de i alcance 10.
13         * Em cada uma dessas iterações, o valor atual de i será exibido
14         * Assim, esse loop resultará na impressão dos números inteiros de 0 a 9.
15         */
16
17         for (int i = 0; i < 10; i++) {
18             System.out.println(i);
19         }
20     }
21 }
```

Figura 5. Exemplo de um trecho de código com um comentário explicativo porém ineficiente

```
1
2 public class Exemplo {
3
4     public static void main(String[] args) {
5         /* Um loop que imprime os números de 0 a 9 */
6         for (int i = 0; i < 10; i++) {
7             System.out.println(i);
8         }
9     }
10 }
```

Figura 6. Exemplo do mesmo trecho de código anterior com um comentário eficiente e explicativo

- Para comentários com mais de uma linha de tamanho, é conveniente utilizar o estilo de comentário de bloco (/***/) ao invés de comentários de linha única (//)

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) at the top left. The code is written in Java and includes multi-line comments using the // style. The code is as follows:

```
1
2 public class Exemplo {
3
4     public static void main(String[] args) {
5         // Esse método imprime os números de 0 a 9.
6         // O loop inicia com i igual a 0 e incrementa
7         // i até que seja menor do que 10
8         for (int i = 0; i < 10; i++) {
9             System.out.println(i);
10        }
11    }
12 }
```

Figura 7. Exemplo de código com um comentário de múltiplas linhas ineficiente

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) at the top left. The code is written in Java and includes multi-line comments using the /***/ style. The code is as follows:

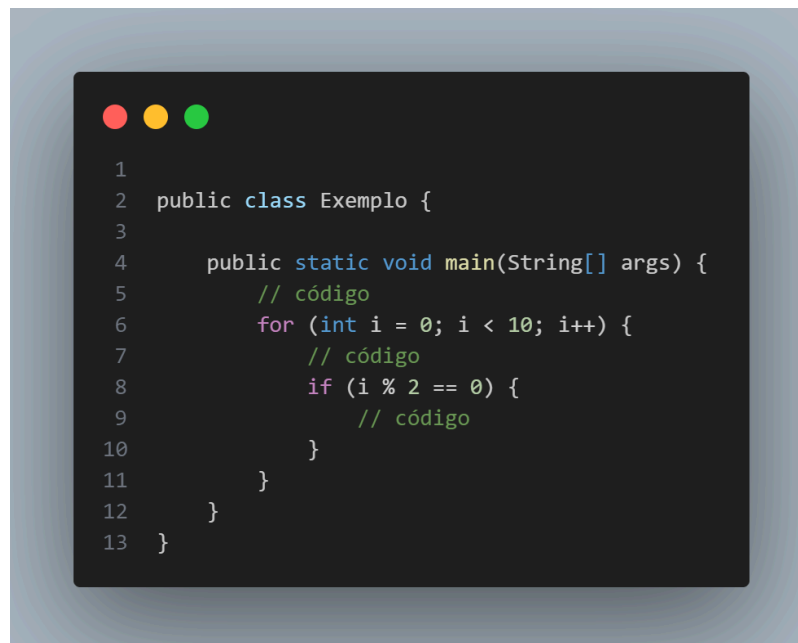
```
1
2 public class Exemplo {
3
4     public static void main(String[] args) {
5         /*
6          * Esse método imprime os números de 0 a 9.
7          * O loop inicia com i igual a 0 e incrementa
8          * i até que seja menor do que 10
9          */
10        for (int i = 0; i < 10; i++) {
11            System.out.println(i);
12        }
13    }
14 }
```

Figura 8. Exemplo do mesmo código do trecho anterior com um comentário de múltiplas linhas eficiente

10. Regras de Indentação:

A indentação, mesmo que não necessária para o funcionamento do código na linguagem Java (como o caso do Python) é uma boa prática para evitar a confusão durante a análise do código.

A forma aconselhável a se fazer isso é posicionar o caractere de abertura de um bloco de instruções ao final da linha que inicia o bloco, fazendo com que o caractere de fechamento se alinhe com a linha que deu início ao bloco. Um exemplo de boa prática de indentação é visto a seguir.

A imagem mostra um editor de código com um fundo escuro e uma barra de título com três botões (vermelho, amarelo, verde) no canto superior esquerdo. O código Java está escrito em uma fonte monoespaçada com coloração de sintaxe: palavras-chave em azul, comentários em verde e variáveis/valores em cinza. O código é bem indentado, com cada nível de bloco (classe, método, laço for, bloco if) deslocado para a direita em relação ao nível anterior. As linhas são numeradas de 1 a 13 no lado esquerdo.

```
1
2 public class Exemplo {
3
4     public static void main(String[] args) {
5         // código
6         for (int i = 0; i < 10; i++) {
7             // código
8             if (i % 2 == 0) {
9                 // código
10            }
11        }
12    }
13 }
```

Figura 9. Exemplo de código bem indentado



Figura 10. Exemplo de código mal indentado

11. Referências e leituras recomendadas:

- <https://www.devmedia.com.br/convencoes-de-codigo-java/23871>
- <https://www.frameworkdemoiselle.gov.br/wikie0c3.html>