Padronização de Código

De iftorrent

A Padronização de Código foi estabelecida pelo Prof. Rafael Vieira Coelho e seu aluno bolsista Gabriel Müller com o objetivo de criar um padrão de códigos em Java para os bolsistas do projeto IFTorrent. Este regulamento organiza assim os códigos e facilita o seu entendimento.

Índice

Regras

Regra 1

Para criação de métodos, sempre colocar chaves na linha inicial do método, e não na próxima.

Exemplo correto:

```
public static void main(String[] args) {
|
}
```

Exemplo incorreto:

```
public static void main(String[] args)
{
```

```
  |
}
```

Regra 2

Sempre utilizar nomes significativos para declaração de variáveis, métodos, classes e pacotes.

Exemplo correto:

```
public static void main(String[] args) {
|   int idade_do_usuario = 18;
|   String nome_do_usuario = "Rafael Vieira Lebre";
}
```

Exemplo incorreto:

```
public static void main(String[] args) {
|   int x = 18;
|   String y = "Exemplo Errado";
}
```

Regra 3

Na declaração, utilizar *underlines* para nomes compostos (não se aplica a classes).

Exemplo correto:

```
public static void main(String[] args) {
|   int idade_do_usuario = 18;
|   String nome_do_usuario = "Rafael Vieira Lebre";
}
```

Exemplo incorreto:

```
public static void main(String[] args) {
|   int idadeDoUsuario = 18;
|   String nomeDoUsuario = "Rafael Vieira Lebre";
}
```

Regra 4

Sempre modularizar o código, fazendo com que a função fique o menor que puder e que o número de atividades a serem realizadas por ela seja o mínimo possível.

Exemplo correto:

```
public static void main(String[] args) {
    int opcao_escolhida = 0;

    switch(){
        case 0:
            chamar_metodo_opcao0();
            break;
        case 1:
            chamar_metodo_opcao1();
            break;
        case 2:
            chamar_metodo_opcao2();
            break;
    }
}
```

Exemplo incorreto:

```
public static void main(String[] args) {
    int opcao_escolhida = 0;

    switch(opcao_escolhida) {
        case 0:
            opcao_escolhida = 1;
            opcao_escolhida *= 15;
            opcao_escolhida -= 8;
            opcao_escolhida /= 3;
            opcao_escolhida += 2;
            opcao_escolhida /= 10;
            opcao_escolhida += 1;
            break;
        case 1:
            opcao_escolhida = 4;
            opcao_escolhida *= 22;
            break;
        case 2:
            opcao_escolhida = 2;
            opcao_escolhida *= 33;
            opcao_escolhida += 8;
            opcao_escolhida /= 11;
            break;
    }
}
```

Regra 5

Utilizar constantes sempre que possível. Essas, por sua vez, devem estar em um arquivo separado apenas para as mesmas. As constantes devem sempre possuir seus respectivos nomes em letras maiúsculas.

Exemplo correto:

```
public class Constantes {
    public static final int     CONSTANTE01 = 5;
    public static final double  CONSTANTE02 = 2.5;
    public static final String  CONSTANTE03 = "Texto.";
    public static final int     CONSTANTE04 = 8;
    public static final boolean CONSTANTE05 = true;
    public static final char    CONSTANTE06 = 'c';
    public static final double  CONSTANTE07 = 4.000;
}
```

Exemplo incorreto:

```
public class Constantes {
|   public static final int     constante01 = 5;
|   public static final double  constante02 = 2.5;
|   public static final String  constante03 = "Texto.";
|   public static final int     constante04 = 8;
|   public static final boolean constante05 = true;
|   public static final char    constante06 = 'c';
|   public static final double  constante07 = 4.000;
}
```

Regra 6

Se após um *if* que está dentro de uma função com retorno não houver código até o retorno final, deve-se retornar sem usar a cláusula *else*.

Exemplo correto:

```
public boolean retorna_maioridade(int idade) {
|   if(idade >= 18) {
|   |   idade += 1;
|   |   return true;
|   }
|   return false;
}
```

Exemplo incorreto:

```
public boolean retorna_maioridade(int idade) {
|   if(idade >= 18) {
|   |   idade += 1;
|   |   return true;
|   } else {
|   |   return false;
|   }
}
```

Regra 7

Em um *if* que realiza um teste lógico, não utilizar "*== true*" ou "*== false*", mas sim apenas a condição (sendo precedida por *!* quando se deseja verificar se o valor é falso).

Exemplo correto:

```
public static void main(String[] args) {
|   boolean sou_maior_de_idade = true;
|   boolean permissao_para_entrar = false;
|
|   if(sou_maior_de_idade) {
|   |   permissao_para_entrar = true;
|   }
}
```

Exemplo incorreto:

```
public static void main(String[] args) {
|   boolean sou_maior_de_idade = true;
|   boolean permissao_para_entrar = false;
|
|   if(sou_maior_de_idade == true) {
|   |   permissao_para_entrar = true;
|   }
}
```

Regra 8

Se um teste lógico for ser realizado e o mesmo ser responsável pelo retorno de uma função, não utilizar *if* e nem *else*, e sim apenas *return* sucedido pela condição.

Exemplo correto:

```
public boolean retorna_maioridade(int idade) {
|   return idade >= 18;
}
```

Exemplo incorreto:

```
public boolean retorna_maioridade(int idade) {
|   if(idade >= 18) {
|   |   idade += 1;
|   |   return true;
|   }
|   return false;
}
```

Regra 9

Não utilizar SQL's em meio ao código, mas sim deixá-los todos em um arquivo separado apenas para os mesmos (Semelhante a regra número 5).

Regra 10

Sempre organizar os arquivos em pacotes, para facilitar o entendimento.

Exemplo correto:

```
iftorrent
|
+---arquivos
|
+---colecoes
|
+---conexao
|
```

```
+---criptografia
|
+---daos
|   |
|   +---daoArquivo
|   |
|   +---daoCategoria
|
+---excecoes
|
+---gui
|
+---ordenador
|
+---texto
```

Exemplo incorreto:

```
iftorrent
|
+---codigos
```

Regra 11

Criar um Javadoc para cada classe criada.

Exemplo correto:

```
package pasta;

/**
 * Esta classe é utilizada para exemplo de documentação
 * de classe.
 *
 * @author Gabriel Müller
 */
public class Classe {
|
}
```

Exemplo incorreto:

```
package pasta;

public class Classe {
|
}
```

Regra 12

Criar um Javadoc para cada função criada.

Exemplo correto:

```
package pasta;

/**
 * Esta classe é utilizada para exemplo de documentação
 * de classe.
 *
 * @author Gabriel Müller
 */
public class Classe {
|
|   /**
|    * Método que faz envelhecer.
|    *
|    * @param idade Inteiro que representa a idade.
|    * @return Inteiro que representa a nova idade.
|    * @throws Exception Se algo der muito errado no
|    * programa.
|    * @see pasta.Classe#envelhecer() ver documentação aqui
|    */
|   public int envelhecer(int idade) throws Exception {
|   |   return idade += 1;
|   }
|
}
```

Exemplo incorreto:

```
package pasta;

/**
 * Esta classe é utilizada para exemplo de documentação
 * de classe.
 *
 * @author Gabriel Müller
 */
public class Classe {
|
|   public int envelhecer(int idade) throws Exception {
|   |   return idade += 1;
|   }
|
}
```

Regra 13

Sempre que possível, utilizar *StringBuilder* para a manipulação de *Strings*.

Regra 14

Sempre que houver declarações de variáveis, deixar uma linha em branco após o final das mesmas (opcional entre métodos). Fora isso, jamais deixar linhas em branco.

Exemplo correto:

```
public static void main(String[] args) {
|   int idade = 18;
|   String nome = "String";
|
```

```
    System.out.println("Idade: " + idade);
    System.out.println("Nome: " + nome);
    int numero_favorito = 2;

    System.out.println("Número favorito: " + numero_favorito);
  }
```

Exemplo incorreto:

```
  public static void main(String[] args) {
    int idade = 18;
    String nome = "String";
    System.out.println("Idade: " + idade);

    System.out.println("Nome: " + nome);
    int numero_favorito = 2;
    System.out.println("Número favorito: " + numero_favorito);

  }
```

Regra 15

Sempre que possível utilizar *for each*, exceto quando o uso de índices do vetor for necessário.

Exemplo correto:

```
  public static void main(String[] args) {
    int[] vetor_de_numeros = {5, 4, 3, 2, 1};

    for(int numero : vetor_de_numeros) {
      System.out.println(numero);
    }
  }
```

Exemplo incorreto:

```
  public static void main(String[] args) {
    int[] vetor_de_numeros = {5, 4, 3, 2, 1};

    for(int i = 0; i < vetor_de_numeros.length; i++) {
      System.out.println(vetor_de_numeros[i]);
    }
  }
```

Regra 16

Em um teste, cada condição separada por um conectivo lógico deve possuir sua própria linha.

Exemplo correto:

```
  public static void main(String[] args) {
    int idade_mae = 30;
    int idade_pai = 32;

```

```
    if(idade_mae < idade_pai
    |    | && idade_mae < 40
    |    | && idade_pai < 40) {
    |    System.out.println("Mais velho: pai. Idade de ambos é < 40");
    |    }
  }
```

Exemplo incorreto:

```
  public static void main(String[] args) {
  |    int idade_mae = 30;
  |    int idade_pai = 32;
  |
  |    if(idade_mae < idade_pai && idade_mae < 40 && idade_pai < 40) {
  |    |    System.out.println("Mais velho: pai. Idade de ambos é < 40");
  |    }
  }
```

Regra 17

Sempre que se for utilizar exceções, usar *throws #nome_exceção* na criação do método e *try-catch* para a chamada do método na *main*.

Exemplo correto:

```
  public static void metodo() throws ArithmeticException {
  |    int x = 5/0;
  }

  public static void main(String[] args){
  |    try {
  |    |     metodo();
  |    } catch(ArithmeticException ex) {
  |    |     System.out.println("ERRO");
  |    }
  }
```

Exemplo incorreto:

```
  public static void metodo() {
  |    try {
  |    |    int x = 5/0;
  |    } catch(ArithmeticException ex) {
  |    |    System.out.println("ERRO");
  |    }
  |
  }

  public static void main(String[] args){
  |    metodo();
  }
```

Regra 18

Para Javadoc:

Tags a serem utilizadas:

- @author
- @throws
- @return
- @param
- @see

Tags que jamais serão usadas:

- @serial (e outras que utilizam serial)
- @deprecated
- @link
- @linkplain
- @exception
- @version Versão (para classes)
- @since Versão (para métodos)

Ver também

- Arquivos HTML e FXML
- Arquivos CSS
- Tags do javadoc (https://docs.google.com/spreadsheets/d/1KrZfd6p55UXxYKhYyTnzC4e-18MP5jOQaGbWjZc1JxE/edit?usp=sharing)

Categorias: Programação | Padronização
