Boas Práticas JAVA - orientado a Sonar

➠ Seguir o padrão na declaração das variáveis:

```
Integer numero_inteiro;
```

```
Integer numeroInteiro;
```

➠ Deve ser declarado / instanciado apenas uma variável por linha:

```
int primeiroValor, segundoValor;

public void setValores() {
  primeiroValor = segundoValor = 10;
}
```

```
int primeiroValor;
int segundoValor;

public void setValores() {
  primeiroValor = 10;
  segundoValor = 10;
}
```

➠ Seguir o padrão das constantes:

```
private static String patternPt = "dd/MM/yyyy";
```

```
private static final String PATTERN_PT = "dd/MM/yyyy";
```

➠ Transforme em constantes números acima de 2:

```
int num = 10;
```

```
private static final int DEZ_INT = 10;
int num = DEZ_INT;
```

➠ Não utilizar comentário na linha do código:

```
String str = "";// Comentário
```

```
// Comentário
String str = "";
```

➠ Utilizar a concatenação de strings de maneira mais performática possível:

```
String str = var1 + " - " + var2;
```

```
String str = String.format("%s - %s", var1, var2);

String str = var1.concat(" - ").concat(var2);

StringBuffer str = new StringBuffer(var1);
str.append(" - ");
str.append(var2);

StringBuilder str = new StringBuilder(var1);
str.append(" - ");
str.append(var2);
```

➠ Se possível usar a classe `StringBuilder` ao invés da classe `StringBuffer`:

```
StringBuffer
```

```
// Não usar em threads
StringBuilder
```

➠ Utilizar `equals` para comparar objetos:

```
if (cor == "verde")
```

```
if (cor.equals("verde"))
```

➠ Utilizar o `isEmpty()` para verificar se a lista está vazia:

```
if (lista.size == 0)
```

```
if (lista.isEmpty())
```

➠ Utilizar o `isEmpty()` para verificar se a string está vazia:

```
if (str.equals(""))
```

```
if (str.isEmpty())
```

➠ Se o valor de uma `String` for usada mais do que 2 vezes na classe, tranforme-a em constante:

```
String action = "voltar";
```

```
private static final String VOLTAR = "voltar";
String action = VOLTAR;
```

➠ Utilize o `equalsIgnoreCase()` se precisar comparar duas Strings ignorando o case sensitive:

```
boolean result = cor.toUpperCase().equals("vErDe");
```

```
boolean result = cor.equalsIgnoreCase("vErDe");
```

➠ Utilizar os métodos que as classes disponibilizam:

```
Integer num = new Integer("10");
```

```
Integer num = Integer.valueOf("10");
```

➠ Utilizar as constantes que as classes disponibilizam:

```
Boolean flag = true;
```

```
Boolean flag = Boolean.TRUE;
```

➠ Utilizar as interfaces ao invés das classes:

```
HashMap map = new HashMap();
ArrayList<String> list = new ArrayList<String>();
```

```
Map map = new HashMap();
List<String> list = new ArrayList<String>();
```

➠ Se instanciar qualquer classe de IO, feche-a:

```
InputStream is = new FileInputStream(path);
...
```

```
InputStream is = new FileInputStream(path);
...
is.close();
```

➠ Se não quiser que a uma data seja modificada, passe uma cópia da data:

```
public Date getDate() {
  return date;
}

dateValidation(date);
```

```
public Date getDate() {
  return date == null ? null : (Date)date.clone();
}

dateValidation(date == null ? null : (Date)date.clone());
```

➠ Usar chaves em blocos `if / if else`:

```
if ()
  ...
else
  ...
```

```
if () {
  ...
} else {
  ...
}
```

➠ Usar chaves em blocos `for`:

```
for ()
  ...
```

```
for () {
  ...
}
```

➠ Se possível usar o `for` melhorado:

```
for (int i = 0; i < size; i++)
```

```
for (String item : lista)
```

➠ Não deve existir mais do que 3 condições booleanas juntas:

```
if (condicao1 && condicao2 && condicao3 && condicao4)
```

```
if (validacao1 && validacao2)

private boolean validacao1() {
  return condicao1 && condicao2;
}

private boolean validacao2() {
  return condicao3 && condicao4;
}
```

➠ Simplifique as comparações com `Boolean`:

```
if (isValid == true)
```

```
if (isValid)
```

➠ Todo o `switch` deve possuir uma cláusula final `default`:

```
switch (status) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
}
```

```
switch (status) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
```

➠ Não dispare as exceptions genéricas (`Error, RuntimeException, Throwable e Exception`):

```
public void validar() throws Exception {
  ...
}
```

```
public void validar() throws MinhaException {
  ...
}
```

➠ Deletar classes, importes, métodos e variáveis não utilizadas.
➠ Não utilizar o mesmo nome para variável global e local.
➠ Se uma variável for usada em apenas em um método, não deixá-la como global.
➠ A complexidade ciclomática dos métodos não deve ser maior que 10.
➠ O método não deve ter mais do que 50 linhas.
➠ Não duplique métodos, coloque-os em classes utilitárias e depois utilize em suas classes.
➠ Em manipulação de datas, se possível, utilize `Calendar` e não `Date`.
➠ Sempre que sobrescrever o método `equals()`, sobrescreva o método `hashCode()` e vice-versa.
➠ Não commitar `printStackTrace()`.
➠ Não commitar `System.out.println()`.
➠ Não commitar códigos-fonte comentado.