By Omar Vargas
https://pmd.github.io/pmd-6.32.0/pmd_rules_java.html

MANUAL DE BUENAS PRACTICAS JAVA

Convención De Nomenclatura Estándar De Java (CamelCase)

By Omar Vargas
https://pmd.github.io/pmd-6.32.0/pmd_rules_java.html

Tabla De Contenido

1. Clases e interfaces en Java (ClassNamingConventions)

Los nombres de las clases deben ser sustantivos, con la primera letra de cada palabra interna en mayúscula.

El nombre de las interfaces también debe estar en mayúscula la primera letra al igual que los nombres de las clases.

Use palabras completas y debe evitar acrónimos y abreviaturas.

```
interface Bicicleta
class MountainBike implements Bicicleta

interface Deportes
class Futbol implements Deportes
```

2. Métodos en Java (MethodNamingConventions)

Los métodos deben ser verbos, en mayúsculas y minúsculas, la primera palabra debe ir en minúsculas y si tiene más palabras, cada una de estas con la primera letra en mayúscula, las variables utilizadas como parámetros de entrada deben anteponer la palabra parameter

```
void cambiarVelocidades (int parametervalue);
void Acelerar (int parameterincremento);
void usarFrenos (int parameterdecremento);
void setEdad (int parameteredad);
void getEdad ();
```

3. Constantes en Java (FieldNamingConventions)

Deben estar todo en mayúsculas con palabras separadas por guiones bajos (“_”).

```
static final int MIN_WIDTH = 4;
static final float POSITIVE_INFINITY = 1.0f;
static final float NEGATIVE_INFINITY = -1.0f;
```

4. Variables en Java (LocalVariableNamingConventions)

Los nombres de las variables deben ser cortos pero significativos. Deben estar en minúsculas. No debería comenzar con un guión bajo ('_') o caracteres especiales. Debe estar diseñado para indicar al observador casual la intención de su uso.

Al definir las variables se debe indicar si se está declarando una variable local, una variable estática o una variable de parámetro de un método.

Se deben evitar los nombres de variable de un carácter, excepto para variables temporales o variables de control. Los nombres comunes para las variables temporales son: i, j, k, m y n para enteros; c, d y e para los caracteres.

```
//variables globales de una clase
private int edad;

// variables locales para la clase
  int localspeed = 0;
  int localgear = 1;

//variables estáticas
static int staticforce = 15;
static double staticpeso = 8.3;

//variables de parámetro de método
void metodoPrueba (int parametervalue1, int parametervalue2);
```

5. Uso de paréntesis en Java (UselessParentheses)

Se debe evitar el uso de paréntesis innecesarios dentro de las instrucciones de cada código:

```
localvar1 = Integer.valueOf((n)); // esto no se debe usar
localvar2 = (n); // y esto tampoco
if ((a>b) && (c>b)) // En estas condiciones dobles tampoco es necesario el exceso
de paréntesis
```

6. Comentarios Requeridos (CommentRequired)

Todos los ficheros deben comenzar con un comentario en el que mínimo se debe indicar el nombre de la clase y el autor, como valores opcionales se puede indicar información de la versión y la fecha de despliegue.

```
void cambiarVelocidades (int parametervalue); //Método abstracto para definir
el cambio de velocidades
```

7. Comentarios de contenido (CommentContent)

Es necesario realizar comentarios de línea para explicar brevemente el uso de cada método

```
/**
 * Nombre de la clase
 * Información de la versión
 * Fecha
 * Autor
 */
```

8. Clases abstractas (AbstractNaming, AbstractClassWithoutAbstractMethod)

Las clases abstractas deben llamarse abstract; y al definir una clase como abstracta debe existir por lo menos un método abstracto dentro de la clase

```
class abstract MountainBike{

void Acelerar (int incremento) //Método definido con instrucciones
{
       Bloque de instruciones
}

void usarFrenos (int decremento); //Método definido de forma abstracta
```

9. Al menos un Constructor (AtLeastOneConstructor)

Cada clase no estática debe declarar al menos un constructor (puede ser vacío si no requiere parámetros).

```
class MountainBike{

Public class MountainBike(){ //Constructor de clase vacío
}

void procedimiento ();
```

10. Importaciones duplicadas (DuplicateImports)

Deben evitarse declaraciones de importación duplicadas o superpuestas.

```
import java.lang.String;
import java.lang.*;
```

11. Inicializador de matriz corta (UseShortArrayInitializer)

Al declarar e inicializar campos o variables de matriz, no es necesario crear explícitamente una nueva matriz utilizando new (A menos que se desee crear un arreglo y agregar los valores posteriormente). En su lugar, uno puede simplemente definir el contenido inicial de la matriz como una expresión entre llaves.

```
//Definición de arreglo y asignación posterior de variables
int notas[] = new int[3];
        notas[0] = 5;
        notas[1] = 8;
        notas[2] = 4;

//Definición de arreglo y asignación directa de variables
int[] x = new int[3] { 1, 2, 3}; //Esto no se debe usar
int[] x = { 1, 2, 3 }; //Esta es la forma correcta
```

12. Métodos con parámetros excesivos (ExcessiveParameterList)

Los métodos con numerosos parámetros son difíciles de mantener, especialmente si la mayoría de ellos comparten el mismo tipo de datos. Estas situaciones generalmente denotan la necesidad de nuevos objetos para envolver los numerosos parámetros.

```
//demasiados argumentos que pueden confundirse
public void addPerson(int birthYear, int birthMonth, int birthDate, int
height, int weight, int ssn)
{
    . . . .
}
//enfoque preferido
public void addPerson(Date birthdate, String measurements, int ssn)
{
    . . . .
}
```

13. Uso de llaves en estructuras de repetición (WhileLoopsMustUseBraces, ForLoopsMustUseBraces)

Evite usar declaraciones 'while' o 'for' sin usar llaves para rodear el bloque de código. Si se pierde el formato o la sangría del código, será difícil separar el código que se está controlando del resto.

```
while (true) //Forma no recomendada
      x++;

while (true) //Forma apropiada de utilizar
{
      x++;
}
```

14. Excepciones genéricas (AvoidCatchingGenericException)

Evite la captura de excepciones genéricas como NullPointerException, RuntimeException, Exception en el bloque try-catch

```
public class PrimitiveType {

    public void downCastPrimitiveType() {
        try {
            System.out.println(" i [" + i + "]");

         //No se recomienda invocar este tipo de excepciones
        } catch(Exception e) {
            e.printStackTrace();
        } catch(RuntimeException e) {
            e.printStackTrace();
        } catch(NullPointerException e) {
            e.printStackTrace();
        }
    }
}
```

15. Lógica Inversa (LogicInversion)

Utilice el operador opuesto en lugar de negar toda la expresión con un operador de complemento lógico.

```
public boolean bar(int a, int b) {

    if (!(a == b)) { // use !=
         return false;
     }

    if (!(a < b)) { // use >
         return false;
    }

    return true;
}
```