Propuesta de estándar de codificación

Montiel Acosta Edgar Didier
Rodríguez Franco Iván
Valdes Contreras Axel Omar
Universidad Veracruzana
Principios de construcción de software
Mtr. Pérez Arriaga Juan Carlos
18 de febrero de 2024

Tabla de contenido

Introducción

En la formación profesional en el campo de la informática y la ingeniería de software, las buenas prácticas de programación son esenciales para garantizar la calidad, la mantenibilidad y la eficiencia de un proyecto de software. En este contexto, establecer un estándar de codificación bien definido dentro de nuestro equipo de desarrollo se vuelve imprescindible.

Este trabajo tiene como objetivo principal la especificación de un estándar de codificación para el lenguaje de programación Java. Este estándar definirá reglas claras y consistentes de sintaxis, nomenclatura y estilo que deberán seguirse durante nuestro proyecto de software. Partiendo de la premisa de que una codificación consistente y legible es fundamental para facilitar la comprensión del código, el mantenimiento y la detección temprana de errores, este estándar abordará diversos aspectos de la programación en java como: convenciones de nomenclatura, formato de código, comentarios, etc.

Al finalizar este trabajo nuestro equipo contará con un estándar de codificación claro y bien definido que servirá como referencia para garantizar la coherencia, la calidad y la eficiencia dentro de nuestro proyecto de software.

Propósito

Propósito general

El propósito general del presente trabajo es crear un estándar de codificación para el lenguaje de programación Java que sirva como guía para el equipo durante la construcción del proyecto de software “COIL-VIC”.

Propósitos específicos

- Definir reglas claras y consistentes de sintaxis, nomenclatura y estilo para la codificación.
- Fomentar la legibilidad y la comprensión del código.
- Promover la coherencia y la calidad del código en todo el proyecto.

Reglas de nombrado

Se seguirá la convención de escritura “CamelCase” en todo el proyecto, con sus respectivas excepciones. El idioma predeterminado será inglés.

Variables

- Pluralizar nombres para colecciones como arreglos o vectores.
- Evitar comenzar nombres de variables con caracteres de guión bajo “_” o signo de dólar “$”.
- Mantener nombres cortos y significativos para indicar la intención de la variable.
- Evitar nombres de un solo carácter excepto para variables temporales.
- Evitar nombres que sean palabras reservadas.

Ejemplo correcto: int nombreSignificativo;

Ejemplo incorrecto: int _ns;

Constantes

- Representar con palabras completas en mayúsculas con guiones bajos entre palabras.
- Las constantes cuyos valores no cambian deben implementarse como “static final”.

Ejemplo correcto: public static final CONSTANT_EXAMPLE

Ejemplo incorrecto: public static final constantExample

Métodos

- Seguir la estructura verbo+sustantativo.
- Nombre que haga alusión al propósito del método.

Ejemplo correcto: int verbSustantive() { …

```
// code
}
```

Ejemplo incorrecto: int Sustantive() { …

```
// code
}
```

Clases

- Los nombres de las clases deben ser sustantivos simples, completos y singulares.
- Comenzar con una letra mayúscula, incluyendo palabras internas.
- Evitar acrónimos o abreviaturas a menos que sean ampliamente reconocidos. Ejemplo: URL o HTML

- Elegir nombres significativos que reflejen el propósito de la clase.

  Ejemplo correcto: public class Example() { …

  // code

  }

  Ejemplo incorrecto: public class eg() { …

  // code

  }

Estilo de código

Indentación

- Se deben de utilizar cuatro espacios como unidad de indentación.
- El patrón de indentación debe seguirse en todo el código.

Ejemplo correcto:

```
if (condición) {
   // code
} else {
   // code
}
```

Ejemplo incorrecto:

```
if (condición) {
        // code
} else {
 // code
}
```

Llaves Izquierda y Derecha

- La llave de apertura debe ir en la misma línea que la declaración o estructura de control.
- La llave de apertura debe estar al final de la condición y la llave de cierre debe estar en una línea separada y alineada con la condición.
- Toda declaración o estructura de control debe contener llaves, independientemente del número de líneas de código que contengan.

Ejemplo correcto:

```
if (condición) {
    // code
} else {
    // code
}
```

Ejemplo incorrecto:

```
if (condición)
{
    // code } else
{
    // code }
```

Longitud de líneas

- Se deben evitar líneas de más de 80 caracteres.
- Cuando una expresión no quepa en una sola línea, debe romperse de acuerdo con los siguientes principios generales:
    - Romper después de una coma.
    - Romper después de un operador.
    - Preferir romper a un nivel superior que a un nivel inferior.
    - Alinear la nueva línea con el inicio de la expresión en el mismo nivel de la línea anterior.
    - Si las reglas anteriores conducen a un código confuso o a un código que está pegado al margen derecho, simplemente indente 4 espacios en su lugar.

Ejemplo correcto:

```
// ejemplo sencillo para términos prácticos
result = (operand1 * operand2) +
        (operand3 / operand4);
```

Ejemplo incorrecto:

```
// ejemplo simulando mas de 80 caracteres
result = (operand1 * operand2) + (operand3 / operand4) – [ * code * ];
```

Líneas en Blanco

- Las líneas en blanco mejoran la legibilidad al separar bloques de código.
- Siempre se debe usar una línea en blanco en las siguientes circunstancias:
    - Entre métodos.
    - Entre las variables locales en un método y su primera instrucción.
    - Antes de un comentario de bloque o de una línea única.
    - Entre secciones lógicas dentro de un método para mejorar la legibilidad.
    - Antes y después de comentarios.

Ejemplo correcto:

```
public class ExampleCode {

    public static void main(String[] args) {
        int result;

        result = sum(5, 10);
        System.out.println("The result is: " + result);
    }

    // Comentario

    public static int sum(int a, int b) {
        return a + b;
    }

}
```

Ejemplo incorrecto:

```
public class ExampleCode {
    public static void main(String[] args) {
        int result = sum(5, 10);
        result = sum(5, 10)
        System.out.println("The result is: " + result);
```

```
    }
    // Comentario
    public static int sum(int a, int b) {
        return a + b;
    }
}
```

Espacios en Blanco

Los espacios en blanco deben usarse en las siguientes circunstancias:

- Una palabra clave seguida de un paréntesis debe separarse por un espacio.
- Debe aparecer un espacio en blanco después de las comas en listas de argumentos.
- Todos los operadores binarios excepto '.' deben separarse de sus operandos por espacios.
- Nunca separar los operadores unarios de sus operandos.
- Las expresiones en una instrucción for deben separarse por espacios en blanco.
- Los cast deben ir seguidos de un espacio en blanco.
- No debe usarse un espacio en blanco entre el nombre de un método y su paréntesis de apertura para distinguir las palabras clave de las llamadas a métodos.
- Después del paréntesis de cierre en la declaración de un método o estructura de control, debe haber un espacio antes de la llave de apertura.

Ejemplo correcto:

```
public class Example {

   public static void main(String[] args) {
      double pi = 3.14;
      int piEntero;

      System.out.println("Result: " + pi);

      for (int i = 0; i < 5; i++) {
         System.out.print(i + " ");
      }
```

```
        piEntero = (int) pi;
    }

}
```

Ejemplo incorrecto:

```
public class Example{
    public static void main(String[] args) {
        double pi=3.14;
        int piEntero;
        System.out.println("Result: "+pi);
        for(int i=0;i<5;i++){
            System.out.print(i+" ");
        }
        piEntero = (int)pi;
    }
}
```

Declaraciones

- Se debe realizar una declaración por línea.
- El orden y la posición de la declaración deben ser los siguientes:
    - Primero se deben colocar las variables estáticas/clase en la secuencia: primero variables protegidas, de nivel de paquete sin modificador de acceso y luego las privadas.
    - Las variables de instancia deben colocarse en la secuencia: primero variables de instancia protegidas, de nivel de paquete sin modificador de acceso y luego privadas.
    - Posteriormente se deben declarar los constructores de clase.
    - Esto debe ser seguido por las clases internas, si corresponde.

Ejemplo correcto:

```
public class Clase {
        protected static int variableProtegida;
        static int variablePaquete;
        private static int variablePrivada;
        protected int variableProtegida;
        int variablePaquete;
        private int variablePrivada;

        Clase () {
        }
}
```

Ejemplo incorrecto:

```
public class Clase {
        Clase() {
        }
        int variablePaquete;
        private int variablePrivada;
        protected int variableProtegida;
}
```

- Los métodos de clase deben agruparse por funcionalidad en lugar de por alcance o accesibilidad para facilitar la lectura y comprensión del código.
- Las declaraciones de variables locales deben estar solo al principio de los bloques, por ejemplo, al principio de un bloque try-catch.

Ejemplo correcto:

```
public int getEjemplo() {
        return ejemplo;
}
public void setEjemplo(int ejemplo) {
        this.ejemplo = ejemplo;
}
public String getVariable() {
        return variable;
}
public void setVariable(String variable) {
        this.variable = variable;
}
```

Ejemplo incorrecto:

```
public int getEjemplo() {
        return ejemplo;
}
public String getVariable() {
        return variable;
}
public void setEjemplo(int ejemplo) {
        this.ejemplo = ejemplo;
}
public void setVariable(String variable) {
        this.variable = variable;
}
```

Comentarios

Usar comentarios al comienzo de cada archivo o en los lugares donde se debe explicar alguna característica/funcionalidad del código.

Los comentarios dentro de un método deben tener el mismo nivel de indentación que la parte de código que describen.

Comentarios en bloque

- Solo usar este tipo de comentario en caso de necesitar más de una línea de código
- El comentario debe ir después de un salto de línea
- La estructura de los comentarios de bloque es: /* … */.

Ejemplo correcto:

```
/*
 * comentario
 *  de ejemplo
 */
```

Ejemplo incorrecto:

```
/* comentario de ejemplo */
```

Comentarios de solo una línea

- Solo usar este tipo de comentario en caso de necesitar solo una línea de código
- En caso de dos comentarios de solo una línea consecutivos considerar el comentario en bloque.
- El comentario debe ir después de un espacio en blanco
- La estructura de los comentarios de solo una línea es: // […].

Ejemplo correcto:

```
// comentario de ejemplo
```

Ejemplo incorrecto:

```
// comentario
// de ejemplo
```

Comentario de fin de línea

- Usar cuando se desee agregar un comentario al final de una línea
- Considerar el tipo de comentario adecuado con base en los dos anteriores

- Si se tiene varios comentarios seguidos se debe asegurar su alineamiento.

Ejemplo correcto:

```
hacerAlgo();        // puede retornar una excepción
hacerOtraCosa(); // crea una instancia
```

Ejemplo incorrecto:

```
// puede retornar una excepción
hacerAlgo();
hacerOtraCosa(); // crea una instancia
```

Comentarios de TODO y FIXME

- Se puede hacer uso de comentarios 'TODO' para marcar tareas pendientes que deben abordarse a futuro.

Ejemplo correcto:

```
// TODO: Implementar método para resolver tal problema
```

Ejemplo incorrecto:

```
// Implementar método para resolver tal problema
```

- Uso de comentarios 'FIXME' para resaltar errores que necesitan ser corregidos.

Ejemplo correcto:

```
// FIXME: Corregir tal error de lógica
```

Ejemplo incorrecto:

```
// Corregir tal error de lógica
```

Estructuras de control

Declaraciones de if y if-else

- La palabra “if” y la expresión condicional deben estar en la misma línea.
- Las declaraciones if siempre deben usar llaves {}.

Ejemplo correcto:

```
if (condición) {
   // código
} else {
   // código
}
```

Ejemplo incorrecto:

```
if (condición)
   // código
```

Declaración de bucle for

- La palabra “for” y los paréntesis deben estar separados por un espacio y las expresiones en un bucle for deberán estar separadas por espacios en blanco.
- El bloque de código se coloca en la siguiente línea.
- La llave de cierre comienza en una nueva línea, indentada para que coincida con su declaración de apertura correspondiente.

Ejemplo correcto:

```
for (int i = 0; i < 10; i++) {
        System.out.println(i);
}
```

Ejemplo incorrecto:

```
for (int i=0;i<10;i++)
{
        System.out.println(i);}
```

Declaración de bucle while

- La estructura de un bucle while sigue el mismo formato que la estructura if.

- La palabra “while” debe aparecer en su propia línea, seguida inmediatamente por la expresión condicional.
- La palabra “while” y los paréntesis deben estar separados por un espacio.
- El bloque de código se coloca en la siguiente línea.
- La llave de cierre comienza en una nueva línea, indentada para que coincida con su declaración de apertura correspondiente.

Ejemplo correcto:

```
While (true) {
   // code
}
```

Ejemplo incorrecto:

```
While(true){
   // code}
```

Declaración de bucle do-while

- El bloque de declaración se coloca en la siguiente línea.
- La llave de cierre comienza en una nueva línea, indentada para que coincida con su declaración de apertura correspondiente

Ejemplo correcto:

```
do {
   // code
while (true);
```

Ejemplo incorrecto:

```
do { // code
while (true);
```

Declaración de switch

- La estructura de un switch sigue el mismo formato que la estructura if.
- La palabra “switch” debe aparecer en su propia línea, seguida inmediatamente por su expresión de prueba.
- La palabra “switch” y los paréntesis deben estar separados por un espacio.

- El bloque de código debe estar en la siguiente línea.
- La llave de cierre comienza en una nueva línea, indentada para que coincida con su declaración de apertura correspondiente.
- Cada vez que un caso no incluya una instrucción “break”, se debe agregar un comentario donde normalmente estaría el “break”.
- Todo switch debe incluir un caso default.
- Un “break” en el caso predeterminado es redundante, pero evita errores si se agrega otro caso más tarde.
- Todas las declaraciones “break” faltantes deben estar correctas y marcadas con un comentario.

Ejemplo correcto:

```
switch (expression) {
  case 1:
    // code
    break;
  default:
    // code
    // No se necesita break aquí
    break; // Comentario para la declaración break redundante
}
```

Ejemplo incorrecto:

```
switch (expression) {
  case 1:
    // code
  default:
    // code
    break; }
```

Declaración de try-catch

- En la estructura try-catch, la palabra “try” debe ir seguida por la llave de apertura en su propia línea.

- Esto es seguido por el cuerpo de la declaración y la llave de cierre en su propia línea.
- Puede seguir cualquier número de “catch”, que consisten en la palabra clave “catch” y la expresión de excepción en su propia línea con el cuerpo del “catch”, seguido de la llave de cierre en su propia línea.
- El bloque try-catch debe incluir un bloque “finally” para liberar todos los objetos no requeridos.
- No debe haber bloques try-catch vacíos.

Ejemplo correcto:

```
try {
   int resultado = 10 / 0;
} catch (ArithmeticException e) {
   // code
} finally {
   // code
}
```

Ejemplo incorrecto:

```
try {
   int resultado = 10 / 0;
} catch (ArithmeticException e) {
   // code
} catch () {
} finally {
   // code
}
```

Referencias bibliográficas

Naveen, S. (2010). *Java Standards*.

https://www.nea.gov.bh/Attachments/eServices%20Standards/Java_standards_V1.0.pdf