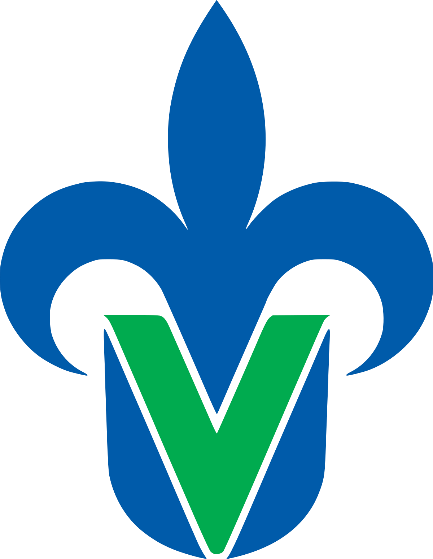
2024



ESTÁNDAR DE CODIFICACIÓN

ADMINISTRACIÓN DE PROYECTOS DE SOFTWARE

EQUIPO 2

Juan David Carmona Ávila

David Carrión Romero

Zaid Alexis Vázquez Ramírez

UNIVERSIDAD VERACRUZANA

Contenido

[Introducción 2](#_bookmark0)

[Propósito 2](#_bookmark1)

[Reglas de nombrado 2](#_bookmark2)

[Variables 2](#_bookmark3)

[Constantes 3](#_bookmark4)

[Métodos 3](#_bookmark5)

[Clases 4](#_bookmark6)

[Estilo de código 4](#_bookmark7)

[Comentarios 7](#_bookmark8)

[Estructuras de control 7](#_bookmark9)

# Introducción

En este documento se establecen los estándares y prácticas de codificación que se seguirán a lo largo del proyecto, para producir principalmente un producto de software de calidad procurando que la gestión de este sea eficiente manteniendo una política de uniformidad en toda la extensión del código. Utilizando buenas prácticas de codificación, se obtendrá como resultado un código simple y elegante, que cuente con las siguientes características:

* Código simple: A través de la utilización de estructuras bien definidas, así como convenciones estandarizadas, se logra un código más comprensible y legible, que ayude a su gestión o modificación a futuro.
* Código de calidad: Las buenas prácticas de programación ayudan a prevenir errores y bugs, lo que se traduce en un software más robusto y estable.
* Código seguro: Está construido para prevenir cualquier posible vulnerabilidad.
* Código eficiente: Se construye para optimizar los recursos que utiliza. Esto se logra mediante la implementación de buenas prácticas de optimización de código.

Seguir buenas prácticas de programación es vital para crear software de alta calidad, sencillo de mantener y escalable. Esto se traduce en un software más eficiente y seguro.

# Propósito

El propósito del presente documento es establecer el estándar que va a regir la manera en que se escribirá el código en lenguaje de programación Java por parte de todos los miembros del equipo a lo largo del desarrollo el proyecto final de la experiencia educativa de Administración de Proyectos de Software. Esto con el objetivo de reflejar calidad, elegancia y buenas prácticas en el código y la forma de programar de cada miembro del equipo.

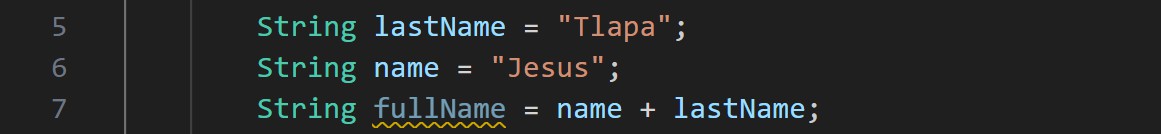
# Reglas de nombrado

Idioma

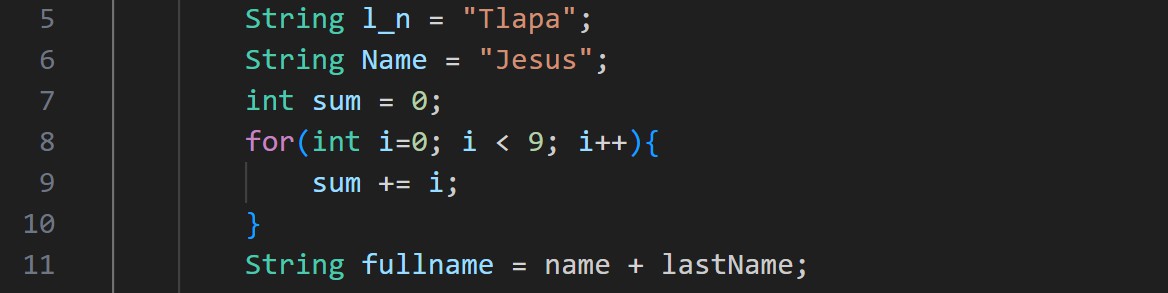
* El idioma en el que se codificará será inglés.

## Variables

* Los nombres de variables serán escritos utilizando el estilo de escritura lowerCamelCase.
* Se minimizará lo máximo posible el uso de caracteres especiales en los nombres de variables, así como la asignación de nombres que carezcan de significado.
  + Con excepción a ciclos o bucles en donde las variables utilizadas se llamarán i, j, k, l en ese orden de anidamiento.
* La declaración de variables se realizará lo más cercano posible a su primer uso.
* El nombre de las variables además deberá ser descriptivo con el fin de evitar ambigüedades. Se evitará además el uso de abreviaturas que dificulten la legibilidad en revisiones posteriores.

Uso correcto:

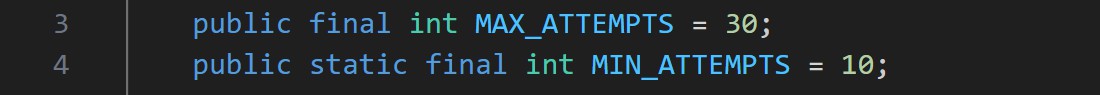
Uso incorrecto:



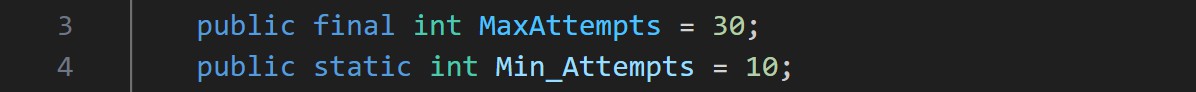
## Constantes

* Para etiquetar una constante, la variable debe ser declarada como *static* y *final.*
* Los nombres de las constantes se encuentran en mayúsculas y en caso de separar palabras se utilizará *guion bajo* ( \_ ) remplazando al espacio.

Uso correcto:



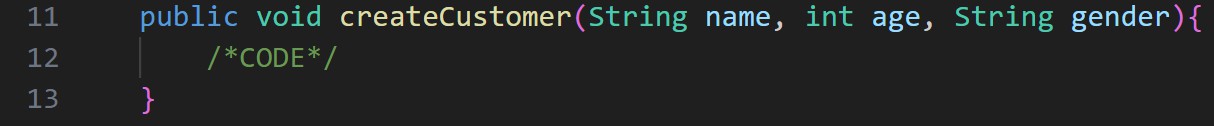
Uso incorrecto:



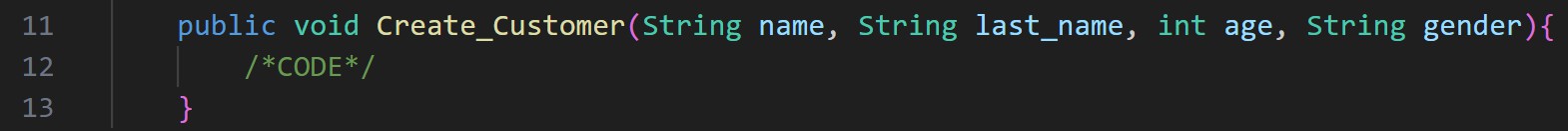
## Métodos

* Los nombres de los métodos deberán ser verbos en infinitivo. No importa la longitud del nombre, siempre y cuando sea lo suficientemente claro y descriptivo.
* La cantidad de parámetros recibidos por cada método debe no ser superior a 3, respetando un orden semántico previamente acordado por el equipo de desarrollo.
* La primera letra del nombre deberá estar escrita en minúscula. En caso de ser un nombre compuesto, la primera letra de cada palabra subsecuente comenzará por mayúscula (lowerCamelCase).

Uso correcto:



Uso incorrecto:



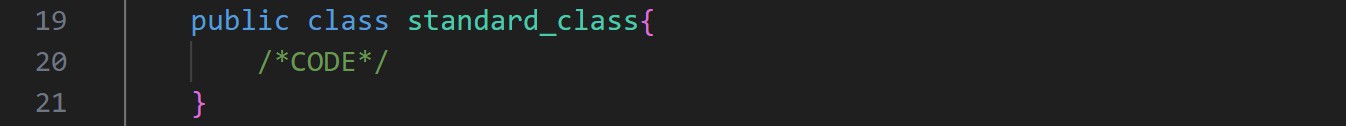
## Clases

* Los nombres de las clases deben ser sustantivos.
* La primera letra del nombre de la clase debe ser mayúscula.
* En caso de ser un nombre compuesto, cada palabra subsecuente comenzará también por mayúscula (CamelCase).
* Para acceder a los atributos de una clase se deberá utilizar la palabra reservada *this* para evitar ambigüedad.

Uso correcto:



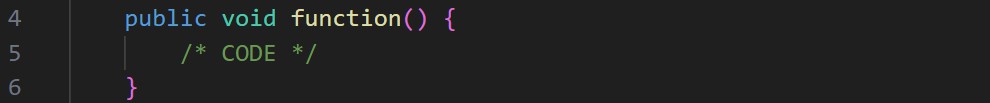
Uso incorrecto:



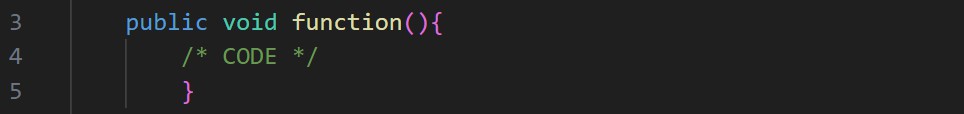
# Estilo de código

Llaves

* En el caso de clases, interfaces, constructores y métodos, la llave de apertura se coloca al final de la misma línea de la declaración dejando un espacio entre el paréntesis que cierra y la llave.
* Para las llaves de cierre, deben estar colocadas al inicio de un salto de línea, y alineadas con la indentación del inicio de la declaración.
* Un bloque vacío puede cerrarse inmediatamente después de abrirse (sin salto de línea). Uso correcto:



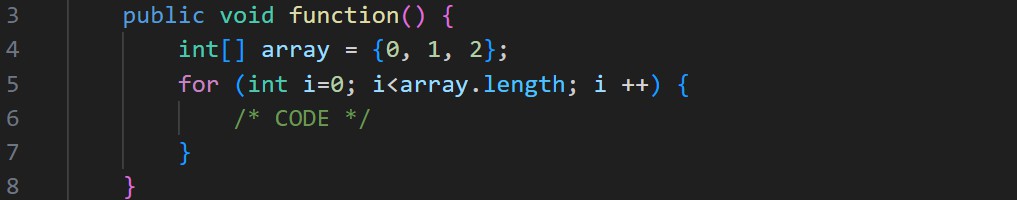
Uso incorrecto:



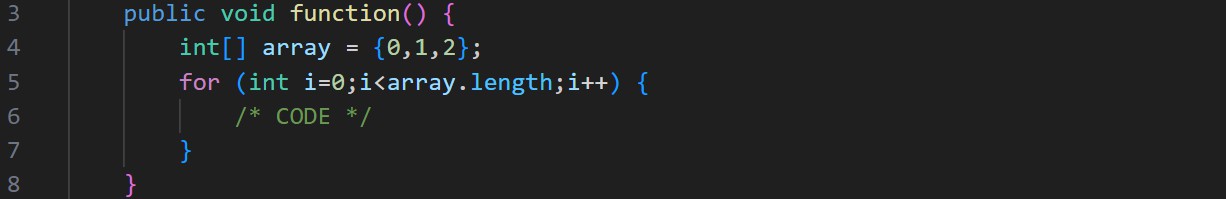
Espacios en blanco

* Se agregarán espacios en blanco entre operadores lógicos-aritméticos y sus operandos.
* También se agregarán espacios en blanco en los bucles *for*, después del punto y coma, y en los arreglos después de la coma.

Uso correcto:



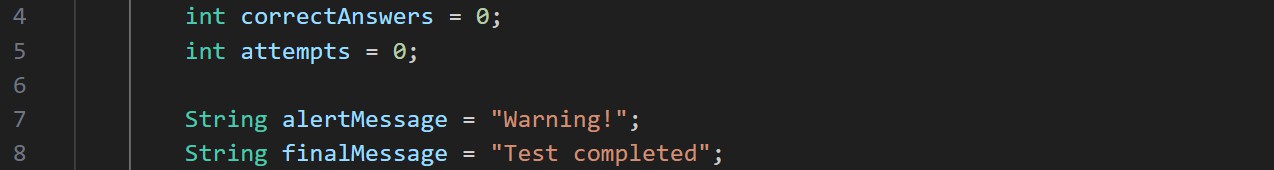
Uso incorrecto:



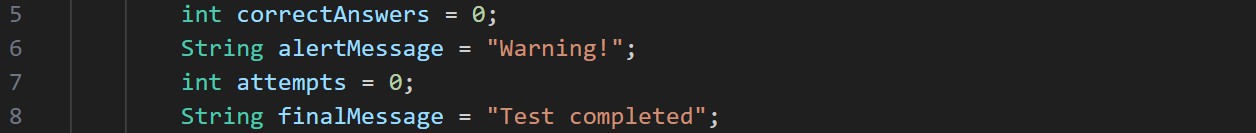
Declaraciones

* Al declarar una variable hay asignarle un valor, es decir, inicializarla. El único motivo para no asignarle un valor en su declaración sería en casos donde su primer valor depende de algún cálculo posterior.
* Sólo se puede realizar una declaración por línea. Con excepción de los bucles *for*, que se pueden declarar en la misma sentencia.
* Al declarar variables, deberán ser agrupadas en caso de ser necesario por su tipo de dato, escritas en líneas continuas. En caso de existir una variable de otro tipo de dato, deberá dejarse un salto de línea entre declaraciones.

Uso correcto:



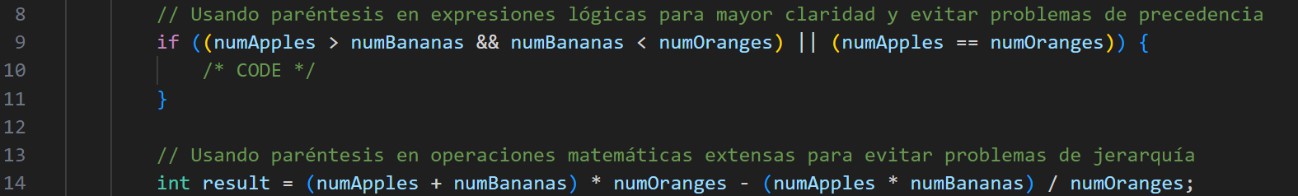
Uso incorrecto:



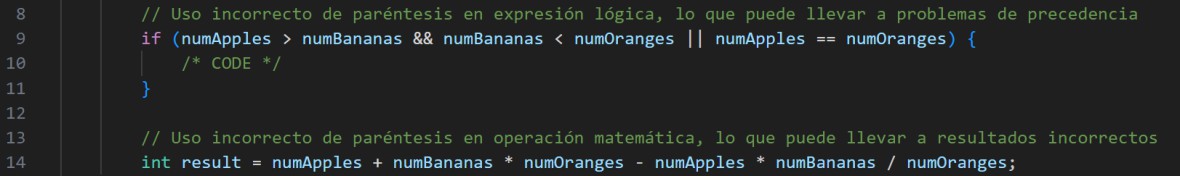
Paréntesis

* Se utilizarán paréntesis en expresiones o condiciones lógicas que incluyan diferentes y más de 1 operador para una mayor comprensión y evitar problemas de precedencia de operadores.
* Se utilizarán paréntesis en operaciones matemáticas extensas que utilicen diferentes operadores aritméticos, con el fin de evitar problemas de jerarquía de operaciones.

Uso correcto:



Uso incorrecto:



Prefijos

* Los prefijos serán válidos en la capa de interfaz gráfica de usuario, conservando un nombre lo suficientemente descriptivo para la variable.

Uso correcto:



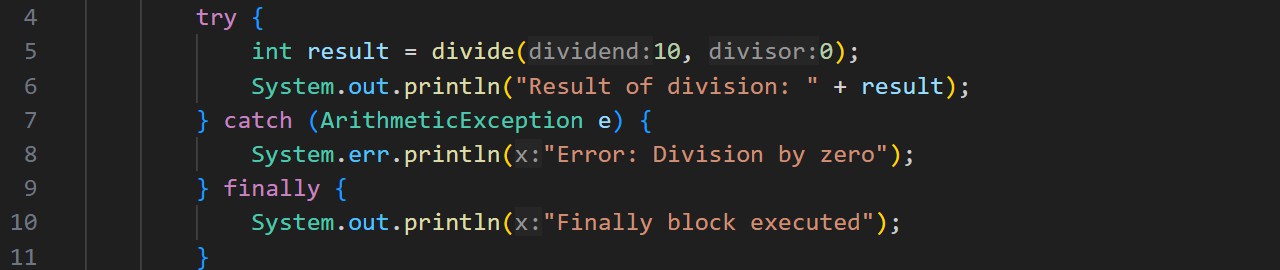
Uso incorrecto:



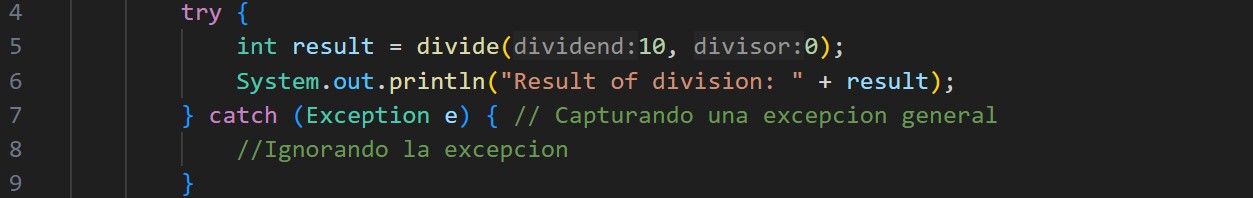
Excepciones

* La sentencia *try* debe ir seguida de una llave de apertura después del cierre del paréntesis sobre su propia línea. A continuación, después del cuerpo de la sentencia se dará un salto de línea antes de la llave de cierre. Después se hará la captura de la excepción o las excepciones, la sentencia debe hacerse con un salto de línea después del *catch*.
* Cada sentencia debe incluir un bloque *finally* después del *catch*.
* Cada bloque *catch* debe manejar solo un tipo de excepción y utilizar la más específica posible, es decir, no se permiten clases generales como *Exception.*
* No deben existir bloques *try-catch* vacíos.
* No ignorar ninguna posible excepción y en caso de hacerlo dejar una justificación como comentario para dejar en claro por qué se está evadiendo.

Uso correcto:



Uso incorrecto:



# Comentarios

* Se evitará el uso de caracteres especiales dentro de comentarios
* Un comentario se definirá comenzando por: /\* y cerrando con: \*/ Uso correcto:

Uso incorrecto:



Estructuras de control Estructuras condicionales if simple

* Todos los bloques *if* deben tener llave de apertura y de cierre, sin importar la cantidad de líneas

que contengan.

* Después de la palabra *if*, debe existir un espacio antes de los paréntesis de la condición.
* La llave de apertura debe ir un espacio después de la condición.
* Debe existir un salto de línea antes y después de cada llave de cierre.

Uso correcto:



Uso incorrecto:



### if – else

* Esta estructura aplicará las mismas reglas que el *if* simple. Uso correcto:

Uso incorrecto:



### if anidado

* Aplicará las mismas reglas que el *if* simple y el *if-else.* Uso correcto:

Uso incorrecto:



Estructuras iterativas o bucles

### While

* Tiene las mismas reglas que un *if* simple.

Uso correcto:



Uso incorrecto:



### do-while

* Tiene las mismas reglas que un *if* simple, con la diferencia de que la palabra *while* y su condición, se escribirán en la misma línea que la llave de cierre, con un espacio antes y después de la palabra *while*.

Uso correcto:



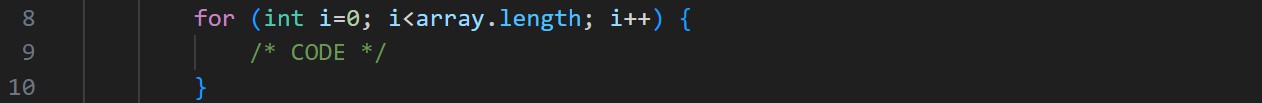
Uso incorrecto:



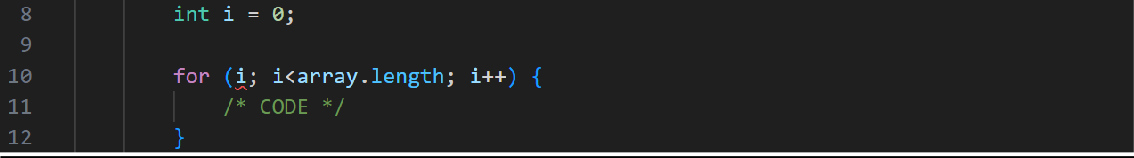
### for

* Tiene las mismas reglas que un *if* simple. Agregando que la variable de control se debe inicializar dentro de los paréntesis.

Uso correcto:



Uso incorrecto:



Bibliografía

Naveen, S. (2010, Julio 04). *Java Standards.* Retrieved from

https://[www.nea.gov.bh/Attachments/eServices%20Standards/Java\_standards\_V1.0.pdf](http://www.nea.gov.bh/Attachments/eServices%20Standards/Java_standards_V1.0.pdf)