UNIVERSIDAD VERACRUZANA

ESTÁNDAR DE CODIFICACIÓN

EQUIPO 8

Lizeth Guadalupe Bello Peralta
Aldo Antonio Campos Gomez
Miguel Angel Mendez Ronzón

Tabla de contenido

Introducción

El presente documento establece el estándar de codificación que deberá seguirse durante el desarrollo del sistema de gestión de prácticas profesionales. Su propósito es garantizar que el código fuente sea coherente, legible y fácil de mantener a lo largo del tiempo. Contar con un conjunto común de reglas permite mejorar la colaboración entre los desarrolladores, reducir errores durante la implementación y facilitar la depuración y evolución del sistema.

Este estándar ha sido elaborado tomando en cuenta buenas prácticas de programación ampliamente aceptadas en la industria, así como las necesidades específicas del proyecto y el contexto académico en el que se desarrolla. Al aplicarlo, se promueve una cultura de orden, calidad y responsabilidad técnica, fundamentales para el trabajo en equipo y el desarrollo de soluciones sostenibles.

Este documento está dirigido especialmente a los estudiantes encargados del desarrollo y mantenimiento del sistema, quienes, a través de su cumplimiento, podrán desarrollar habilidades profesionales en la escritura de código limpio, estructurado y mantenible. También servirá como referencia para futuros equipos que se integren al proyecto, asegurando continuidad, claridad y coherencia en el desarrollo.

Propósito

Definir un conjunto de lineamientos de codificación que sirvan como guía para el desarrollo del sistema de gestión de prácticas profesionales. Su objetivo es establecer una base uniforme para escribir código claro, consistente y de calidad, lo que facilita el trabajo en equipo, la comprensión del sistema y su mantenimiento.

Se busca fomentar buenas prácticas de programacuón, reducir la ambigüedad, garantizar la mantenibilidad y legibilidad además de apoyar en la formación de los estudiantes encargados del desarrollo

Idioma

Para asegurar la compatibilidad, portabilidad y correcta interpretación del código fuente en diferentes sistemas y plataformas, todo el código, incluidos los nombres de variables, funciones, clases, archivos creados y comentarios técnicos deberán de escribirse en inglés. Esto incluye eviar el uso de caracteres especiales como la letra “ñ” y tildes.

Por otro lado, la interfaz gráfica de Usuario, al estar orientada a usuarios hispanohablantes, utlizará el idioma español.

Reglas de nombrado

Variables

Se utilizarán nombres descriptivos que permitan comprender y manejar el código. Se utilizará camelCase en caso de que la variable contenga más de una palabra.

```
String telefonoResponsable = “000-0000000”;
```

El único momento donde se puede utilizar una sola letra o una variable poco descriptiva es cuando su alcance dentro del código s corto como por ejemplo, existe únicamente dentro de un ciclo o el catch de una excepción.

```
 for (int i = 0; i < 10; i++) {
        // Code
}

try {
        PreparedStatement statement = Statement(query);
        statement.execute()
} catch (SQLException e) {
        System.error.println(e.getMessage());
}
```

Constantes

Al igual se usan nombres descriptivos en el formato SCREAMING_SNAKE_CASE. Si contiebe más de una palabra, estas se separan por guiones bajos

```
private static final String DB_USERNAME “admimPracticasProfesionales”
static final String DB_NAME = “sistemaPracticasProfesionales”
```

Métodos

Para su nombramiento, se debe de usar camelCase al igual que con el caso de las variables. La primera palabra en minúscula y las siguientes inician con mayúscula. Deben utilizarse verbos e indicar lo que hacen.

```
public void calculateGrade(){}
```

Clases e interfaces

Para su nombramiento debe de utilizarse PascalCase. La primera letra debe estar en mayúsculas.

```
Public class Student{
        {
```

No puede ser verbo, debe ser sustantivo además de que deben de descubrir su propósito.

Se pueden utilizar abreviaciones dependiendo del tipo de clase de la que se trate. A continuación se muestran las abreviaciones consideradas

Palabra	Abreviación
Data Base	DB
Data Access Object	DAO
Plain Old Java Object	POJO

Paquetes

Se escribirán en minúscula y sin espacio, por ejemplo

app.modelo
app. Vistas

Componentes de la GUI

Cuando se necesite nombrar un componente, se usará un prefijo de dos letras que indique su tipo, seguido de un nombre descriptivo en notación PascalCase.

Por ejemplo, para un campo de texto que almacena el nombre de un alumno, se nombrará como **tfNombreAlumno**, dónde:

- tf indica que se trata de un TextField
- NombreAlumno describe el propósito del componente

Estilo de código

Espacios en blanco

Necesario para mantener la legibilidad. Se utiliza Solamente un solo espacio al terminar de escribir cada palabras

```
int edad = 28;
```

Indentación

Para los paquetes e importaciones de las distintas bibliotecas se ocupa un indentado de 0. Para los atributos y declaración de métodos se ocupa una indentación de 4 espacios respecto al margen izquierdo mientras que para el contenido de los métodos (incluyendo constructores, getters y setters) se necesita una indentación de 4 espacios adicionales a la indentación del método

```
import java.util.*;

public class Muestra {
    private nombre

    public Muestra{
nombre = “”;
    }
}
```

Uso de llaves

Para un código limpio, coherente y fácil de mantener, se aplican las siguientes reglas respecto al uso de las llaves:

- Clases, interfaces, métodos, constructores, getters y setters

La llave de apertura ({) debe ir al final de la misma línea en la que se realiza la declaración. La llave de cierre (}) debe colocarse en una línea nueva, sola y alineada con el inicio de la declaración correspondiente

- Bloques de control de flujo
  Siempre deben incluir llaves de apertura y cierre, incluso si el bloque se trata de una sola instrucción con el fin de prevenir errores y sigue el mismo método de colocación que los métodos y clases: lleva de apertura al final de la línea, llave de cierre sola en la línea nueva y correctamente indentada

Límite de caracteres

Cada línea debe tener un máximo de 100 caracteres.
Si se excede —o está cerca de hacerlo— se debe realizar un salto en lugares adecuados para dividir la línea para mantener la legibilidad del código.
Los puntos recomendados para hacer el salto incluyen:

- Operadores (como +, -, *, /)
- Signos de puntuación, especialmente comas en listas de argumentos o parámetros
- Delimitadores de bloques o estructuras como [ ], { }, ( )

Comentarios

Comentarios de bloque

Se utilizan para explicar secciones de código que abarcan más de una línea. Deben de utilizarse con moderación y sólo cuando aporten un valor real. Deben estar escritos en lenguaje claro y conciso. La primera línea sólo abre el bloque pero se comienza a escribir en la línea de abajo. Para el caso de la línea de cierre, se cierra el bloque una línea después del comentario.

```
/*
Ejemplo de comentario en bloque:
Utilizado con el fin de dar una explicación detallada
*/
```

Comentarios de línea única

Utilizarlos únicamente cuando se necesite de una descripción corta. Se escribe en la misma línea de declaración del comentario.

```
//comentario de una sola línea
```

```
public class Muestra(){
}
```

Estructuras de control

- Usa un espacio entre la palabra clave de control y el paréntesis de la condición

```
if (condicion) {
}
```

- Los bloques if – else deben escribirse en la misma línea que la llave de cierre del bloque anterior

```
if (age < 18) {
    showKidsContent();
} else {
    showNormalContent();
}
```

- Evita estructuras de control anidadas excesivas

Si se necesita más de dos niveles de anidación, considerar extraer lógica para ponerla en métodos auxiliares.

- Bucles for, while y do-while

Sigue las mismas reglas de espaciado entre la palabra clave y la condición

- Switch-case

Cada case debe terminar en un break para evitar uso de varios casos en una misma oportunidad.
Los case van indentados dentro del switch
Siempre incluir un default para manejar casos no contemplados

```
    switch (opcion) {
       case 1:
             ejecutarOpcionUno();
             break;
        case 2:
            ejecutarOpcionDos();
            break;
```

```
        default:
            mostrarMensajeError();
            break;
    }
```

Excepciones y errores

Toda excepción comprobada debe manejarse adecuadamente ya sea con bloques try-catch en el mismo método donde surge la excepción o utilizando un throws en el método donde surge la excepción y tratandola en el método que necesita utilizar el otro método.

Se debe evitar capturar clases genéricas como Exception o Throwable a menos de que sea necesario. Además, se deben evitar dejar bloques de catch vacíos o utilizar excepciones dentro de la lógica normal del programa.

En caso de que ocurra un error que obligue al cierre total del programa se debe informar claramente al usuario que ha ocurrido un error y que la aplicación se cerrará. Se debe de intentar conservar la mayor cantidad de información posible y de tener una salida controlada y segura.