Universidad Veracruzana

Propuesta de Estándar de Codificación

Chacón Fernández Alejandro
Domínguez Carmona José Javier
Obando Muñoz Armando Omar

15/Junio/2022

INDÍCE

Estándar de Código

1. Introducción

El presente documento presenta la funcionalidad de definir de manera completa los estándares de codificación establecidos por el equipo de desarrollo para la mejora de la calidad del producto de trabajo, dichos estándares asignan reglas estrictas a seguir evitando establecer prácticas en el estilo de código.

De manera similar a guías de desarrollo ya existentes los aspectos definidos no solo son de carácter estético enfocado al formato, sino que abarca otras clases de convenciones o estándares de escritura de código de calidad.

2. Propósito

La intención de desarrollar un documento de estandarización de código consiste en dar el formato o pautas a seguir como herramienta para conseguir el estilo homogéneo y correcto entre los miembros del equipo, además con el uso de este nos permite garantizar a nivel de calidad un código limpio, consistente, legible y comprensible durante todo el desarrollo, buscando el evitando retrasos o reinvertir tiempo en errores de codificación o comprensión de código. Asu vez permite la apropiación de código por parte de o de los programadores dando la aportación de la intención propia del programador ante su código en la implementación de sus propias "buenas prácticas" en un formato consistente y documentado. Por último, el documento se anexa a los contenidos de documentación de código en donde se justifican las practicas implementadas permitiendo un entendimiento mayor al momento de leer el código.

3. Reglas de Nombrado

- Las clases de Data Access Object serán tendrán el sufijo "DAO".
- Esta permitido el uso de sufijos ya establecidos como "Test", etc. por el IDE o el estándar de desarrollo general de Java.
- Las variables de las Clases de Data Access Object tendrán el sufijo como "DAO" o "Dao".
- Los archivos con extensión ". FXML" llevarán el nombre de la extensión en el sufijo.
- Los archivos de controlador de "GUI" tendrán el nombre de su respectivo archivo FXML seguido de la palabra "Controller". Ejemplo: "RegistroSolucionProblematicaAcademicaFXMLController.java".
- Las clases de uso intermedio en las tablas tendrán la siguiente estructura "DIG" + nombre de la case + lugar donde se utilice. Si se requiere ser más específico, la estructura será "DIG" + nombre de la case + razón +lugar donde se utilice. Ejemplo "DIGAcademicoAsignacionTutorTabla." Donde DIG = Dominio Interfaz Gráfica.
- Los elementos de interfaz gráfica serán nombrados con la siguiente nomenclatura: Tipo de elemento + nombre. Ejemplo: "buttonCancelar".
- El nombrado de las excepciones será las siglas del tipo de excepción más la palabra "Exception". Ejemplo "SQLIntegrityConstraintViolationException = sqlicvException", SQLException = "sqlException",
- La referencia a la base de datos puede ser abreviada como "db".

- Todo nombrado debe ser autodescriptivo a consideración del programador, según el contexto del dominio.
- Nunca se deben utilizar palabras reservadas.
- Clases o métodos que hagan referencia directa al empleo de un tipo de dato se le permitirá el uso de la palabra reservada en el nombrado de dicho método o clase.

3.1. Variables

- Todo nombre de variable debe ser escrito en lowerCamelCase.
- La declaración de variables se debe de hacer en sus respectivos renglones, una a la vez.
- La mayoría de las variables deben ser declaradas de manera "local" o buscando que su alcance sea el menor posible, para minimizar su alcance dentro del programa, dejando esto a las necesidades coherentes y justificadas del programador.
- No se hará uso de abreviaturas para los elementos de la GUI como Button, ComboBox, TextField, etc.
- Para declarar elementos de la GUI se debe anteceder el nombre del elemento más la variable a la que están asociados (Ver Apéndice A).

Correcto	Incorrecto

`public class Estudiante(){`
`  private String nombre;`
`  private String apellidoPaterno;`
`  private String apellidoMaterno;`
`  private int edad;`
`  private int semestresCursados;`
`  private float peso;`
`  private float altura;`
	`public class Estudiante(){`
	`  private String nombre, apellidoPaterno;`
	`  private int EDAD;`
	`  private float peso, altura;`
	`  private String Apellido_Materno;`
	`  private int SEMESTRES_CURSADOS;`

3.2. Constantes

- Todo nombre de constante debe ser escrito en UPPER_SNAKE_CASE

Correcto	Incorrecto

`public class Estudiante(){`
`  private String nombre;`
`  private String apellidoPaterno;`
`  private String apellidoMaterno;`
`  private static final int UNA_CONSTANTE = 123;`
	`public class Estudiante(){`
	`  private static final int UnaConstante = 123;`
	`  private String nombre;`
	`  private String apellidoPaterno;`
	`  private String apellidoMaterno;`

3.3. Métodos

- Todo nombre de método debe ser escrito en lowerCamelCase
- Todo nombre de método debe ser un verbo o una frase verbal. Ejemplos: correr o mandarMensaje
- Los métodos de las clases Data Access Object deben dar un tipo de retorno distinto de void.
- Ningún método debe tener más de 5 parámetros.

Correcto	Incorrecto
`public boolean registrarEstudiante(Estudiante e){ //codigo }`	`public void estudianteRegistrado( String matricula, String nombre1, String nombre2, String apellidoPaterno, String apellidoMaterno){ //codigo }`

3.4. Clases

- Los nombres de clase se escriben en UpperCamelCase.
- Los nombres de clase serán un sustantivo o una frase sustantiva en singular. Ejemplos: Lista o ListaInmutable.
- La clase de implementación de conexión a base de datos está definida como "DataBaseConecction".
- La clase que contiene el encriptado de contraseñas será nombrada SHA_512 haciendo referencia al algoritmo de codificación de contraseñas "Security Hash Algorithm".
- Clases de apoyo para clases principales se denominarán con el sufijo "Helper".

Correcto

```
public class Estudiante() extends Object implements Comparable<Estudiante>{
    private String nombre;
    private String apellido;
    private static final int UNA_CONSTANTE = 123;
    public Estudiante(String n, String a){

    }
    public void setNombre(String n){ }
    public String getNombre(){ }
    public void setApellido(String a){ }
    public String getApellido(){ }
    public int compareTo(Estudiante e){ }
}
```

4. Estilo de Código

- Por cada línea de código no deben de existir más de 100 caracteres, en su defecto utilizar el uso de Enter para continuar en la siguiente línea sin aplicar sangría. (Con una tolerancia de 20 caracteres más de la longitud máxima establecida).
- Ningún salto de línea antes de la llave de apertura.
- Las llaves siguen el estilo de Kernighan y Ritchie (" corchetes egipcios ").
- Salto de línea después de la llave de cierre, solo si dicha llave termina una declaración, cuerpo de un método (con excepción de los bloques try{}catch{} y estructuras lógicas que van seguidas como en el caso de if(){}else{}), constructor o clase con nombre.

Correcto

```
intente {
  vacíoStack . estallar ();
  fallar (); } catch ( se esperaba NoSuchElementException ) { }
```

Incorrecto

```
interruptor ( entrada ) { caso 1 : caso 2 :
    prepareOneOrTwo (); // pasar por el caso 3 :
    handleOneTwoOrThree (); romper ; predeterminado :
    handleLargeNumber ( entrada ); }
```

5. Comentarios

- Todo comentario debe explicar el porqué del código, no su funcionamiento.
    - Si el funcionamiento necesita explicación, entonces se debe reescribir el código para incrementar su legibilidad, utilizando métodos y variables bien nombrados.
- Ninguna variable debe ser comentada. Su identificador debe ser suficiente para explicar su propósito.
- En cada comentario se debe incorporar el nombre de quien lo escribió, anteponiendo dos guiones y el nombre de la persona.

Correcto	Incorrecto

`//Utilicé un ArrayList porque la clase Vector no tiene los métodos necesarios para hacer esta operación -- Alejandro`	`/*`
	`########################################`
	`# la bariable “x” significa “edad” #####`
	`########################################`
	`*/`

6. Estructuras de Control

- Uso de llaves en los bloques condicionales, incluyendo aquellos en los que el bloque de código sea de una sola línea.
- Dentro de la indentación se permite en un mismo renglón el cierre y apertura de estructuras lógicas si estas van seguidas.

```
if(condicion){
    realizar...
}else{
    realizar...
}
Continua Codigo
```

Apéndice A: Nombre de elementos de la GUI

Nombre de clase GUI	Abreviatura/Prefijo	Ejemplo
ComboBox	No aplica.	comboBoxProfesores
Label	No aplica.	labelNombreEstudiante
Button	No aplica.	buttonRegistrar
TextField	No aplica.	textFieldDescripcion
TextArea	No aplica.	textAreaEjemplo
PasswordField	No aplica.	passwordFieldUsuario
ScrollPane	No aplica.	scrollPaneDocumentos
ScrollBar	No aplica.	scrollBarNombres

Correcto

```
@FXML
private TextField textFieldUsuario;
@FXML
private PasswordField passwordFieldContrasenia;
@FXML
private Label labelErrorUsuario;
@FXML
private Label labelErrorContrasenia;
@FXML
private Label labelSistemaTutoriasFEI;
@FXML
private Label labelInicioSesion;
@FXML
private Label labelUsario;
@FXML
private Label labelContrasenia;
@FXML
private Button buttonIniciarSesion;
```

Incorrecto

```
@FXML
private TextField txtFUsuario;
@FXML
private PasswordField pssContrasenia;
@FXML
private Label lblErrorUsuario;
@FXML
private Label  lblErrorContrasenia;
@FXML
private Label lblSistemaTutoriasFEI;
@FXML
private Label  lblInicioSesion;
@FXML
private Label  lblUsario;
@FXML
private Label  lblContrasenia;
@FXML
private Button btnIniciarSesion;
```