UNIVERSIDAD VERACRUZANA

Licenciatura en Ingeniería de Software

Estándar de codificación para el Proyecto

Principios de Construcción de Software

Académico: Ramón Gómez Romero

Realizado por:

Uriel Cendón Diaz

Miguel Eduardo Escobar Ladrón de Guevara

Alan Raziel Filobello Aguilar

Coding Standard Template

Propósito	Crear un estándar de codificación (en conjunto del equipo) para el proyecto de la experiencia educativa de Principios de Construcción de Software y garantizar su claridad, mantenibilidad y la calidad del código que se realice.
Autor	Uriel Cendón Díaz | Miguel Eduardo Escobar Ladrón de Guevara | Alan Raziel Filobello Aguilar
Encabezados de programa	Comenzar todos los programas con un encabezado descriptivo.
Formato de encabezados	/** * Autor: [El nombre del autor] * Fecha de creación: [Fecha en formato DD/MM/AAAA] * Descripción: [Breve explicación de la funcionalidad del archivo/clase] */
Identificadores	• Utilizar nombres descriptivos para todas las variables, nombres de funciones, constantes y otros identificadores (como clases y constantes). • Evite las abreviaturas o las variables de una sola letra, salvo en variables temporales o contadores. • No usar abreviaturas que puedan ser confusas.

Ejemplos de identificadores	//correcto.
	class Moto {
	private final int velocidadMaximaKmh;
	private final String codigoDeSerie;
	}
	
	//incorrecto.
	class Moto {
	private final int v;
	private final String c;
	}

Comentarios	• Documente el código para que el lector pueda entender su funcionamiento. • Los comentarios deben explicar tanto la finalidad como el comportamiento del código. • Los comentarios deben explicar el "por qué" del código, no "cómo" lo hace. • Utilizar JavaDoc para las clases, métodos y atributos que sean públicos. • Si se usan comentarios en la misma línea, deben ser claros y concisos, además de escribirse, separado de un espacio, justo al finalizar la instrucción del código.

Buen comentario	/**
	* Calcula el interés simple.
	* @param capital El capital inicial.
	* @param tasa La tasa de interés anual en porcentaje.
	* @param tiempo El tiempo en años.
	* @return El interés simple calculado.
	*/
	double calcularInteresSimple(double capital, double tasa, double tiempo) {
	return (capital * tasa * tiempo) / 100;
	}

Mal comentario	//calcular interés

	double calcularInteresSimple(double capital, double tasa, double tiempo) {
	  return (capital * tasa * tiempo) / 100;
	}

Secciones principales	Anteponga a las secciones principales del programa un comentario de bloque que describa el procesamiento que se realiza en la sección siguiente.
Ejemplo	/* /* Sección: Cálculo de estadísticas /* Esta sección del código obtiene las calificaciones de los /* estudiantes y calcula el promedio, la máxima y mínima nota. */
Espacios vacíos	• Escriba los programas con suficiente espaciado para que no parezcan abarrotados. • Separe cada construcción del programa con al menos un espacio.
Indentación	• Utilizar sangría cada nivel de corchete con respecto al anterior. • Los corchetes de apertura deben estar en la misma línea que la instrucción en la que se declaran. • Los corchetes de cierre deben estar en líneas solas y alineadas entre sí con la instrucción y el corchete de apertura. • Usar espacios en lugar de tabulaciones (4 espacios por nivel de indentación). • Dejar un espacio entre operadores y palabras clave. • No juntar llaves en una misma línea. • No se deben dejar líneas en blanco dentro de los métodos.

Ejemplo de indentación	// correcto
	if (valor > 0) {
	  realizarAccion();
	} else {
	  manejarError();
	}
	
	// incorrecto
	if(valor>0){realizarAccion();}else{manejarError();}

Capitalización	**1. camelCase**
	• Uso: Variables, métodos y parámetros.
	• Formato: La primera palabra en minúscula; las siguientes inician con mayúscula.
	• Tipo de palabra:
	  ○ Variables y parámetros: sustantivos o frases nominales.
	  ○ Métodos: verbos o verbos + complemento.
	• Ejemplos:
	  ○ Variable: userName, totalAmount
	  ○ Método: getUserInfo(), sendEmail()
	  ○ Parámetro: clientId, emailAddress
	
	2. PascalCase
	• Uso: Clases e interfaces.
	• Formato: Todas las palabras comienzan con mayúscula.

	• Tipo de palabra:
	  ○ Clases: sustantivo singular o compuesto.
	  ○ Interfaces: sustantivo o adjetivo descriptivo con prefijo I.
	• Ejemplos:
	  ○ Clase: User, InvoiceManager, ProductList
	  ○ Interfaz: IUser, IShape, ISerializable
	3. UPPER_SNAKE_CASE
	• Uso: Constantes (valores fijos que no cambian).
	• Formato: Todas las letras en mayúscula, palabras separadas por guiones bajos.
	• Tipo de palabra: sustantivo o nombre claro del valor.
	• Ejemplos:
	MAX_USERS, DEFAULT_TIMEOUT, PI_VALUE
	4. Nombres de Paquetes
	• Uso: Nombres de paquetes (en lenguajes como Java).
	• Formato: Todo en minúsculas, palabras separadas por puntos.
	• Tipo de palabra: sustantivos.
	• Ejemplos:
	com.empresa.proyecto, org.example.utilidades
	5. Interfaces
	• Uso: Contratos que definen comportamientos.
	• Formato: PascalCase con prefijo I.
	• Tipo de palabra: sustantivo o adjetivo que describe una capacidad o entidad.
	• Ejemplos:
	IRepository, IUserService, IDrivable

Ejemplos de capitalización	// Clases en PascalCase
	GestionEstudiantes;
	
	// Interfaces con prefijo I
	IProcesable;
	
	// Paquetes en minúsculas separadas por puntos
	com.miempresa.utilidades;
	
	// Métodos y variables en camelCase
	calcularPromedio(int total, int cantidad);
	radioCirculo;
	
	// Constantes en SCREAMING_SNAKE_CASE
	#define MAX_TAMANIO 100
	PI_VALOR = 3.1416;

Saltos de línea	• Se debe limitar a un máximo de 100 columnas por línea.
Recursos de JavaFX	• **Vistas (.fxml):** ○ Todos los archivos con extensión .fxml deben ubicarse dentro del paquete view.

	○ Deben comenzar con el prefijo FXML, seguido del nombre de la vista en formato PascalCase. ○ Ejemplo: FXMLMainView.fxm ○ • **Controladores:** ○ Los controladores deben ubicarse dentro del paquete controller. ○ Su nombre debe coincidir con el del archivo .fxml, iniciando con FXML y finalizando con el sufijo Controller. ○ **Ejemplo**: FXMLMainViewController.java • **Cargar vistar** ○ Para cargar vistas, se debe usar FXMLLoader. • **Componentes de la Interfaz (fx:id):** ○ Los elementos de la interfaz se deben declarar como variables, utilizando notación camelCase. ○ Se recomienda utilizar una contracción como prefijo que identifique el tipo de componente(para el proyecto pondremos iniciales de los componentes): ▪ btn para botones (Button) ▪ lb para etiquetas (Label) ▪ txt para campos de texto (TextField, TextArea) ▪ tv para tablas (TableView) ▪ Etc. ○ **Ejemplos**: btnIniciarSesion, lblMensajeError, txtNombreUsuario, tblUsuarios • **Eventos y Métodos de Acción:** ○ Los eventos deben implementarse como métodos, utilizando nombres en camelCase que comiencen con un verbo que indique la acción realizada. ○ Se recomienda incluir una referencia al componente en el nombre si aplica, para mayor claridad. ○ **Ejemplos**: clicBtnIniciarSesion(), guardarDatosUsuario(), cerrarVentana()
Ejemplos	FXMLLoader loader = new FXMLLoader(getClass().getResource("/view/MainView.fxml")); Parent root = loader.load();
Manejo de Excepciones	• Las excepciones se deben manejar de manera específica, según del tipo que sean. • No se debe usar catch genérico. (Exception e). • Pueden existir excepciones personalizadas. • No se deben usar para control de flujo, deben ser únicamente para situaciones inesperadas.

Ejemplos	// correcto
	try {
	FileReader archivo = new FileReader("datos.txt");
	} catch (FileNotFoundException e) {

	System.err.println("Error: El archive no se encontró en la computadora - " + e.getMessage());
	}
	
	// incorrecto
	try {
	FileReader archivo = new FileReader("datos.txt");
	} catch (Exception e) {
	System.out.println("Ocurrió un error");
	}

Importaciones	• Las importaciones siguen un orden lógico separadas por líneas en blanco, primero las de Java estándar, luego las de librerías externas, las del proyecto propio y finalmente las importaciones estáticas específicas. • No usar comodín (*) para las importaciones.

Ejemplos de importaciones	// correcto
	import java.util.List;
	import java.util.Map;
	
	import org.apache.commons.lang3.StringUtils;
	
	import com.miempresa.utilidades.MiClase;
	import com.miempresa.servicio.Servicio;
	
	import static java.lang.Math.PI;
	import static java.lang.Math.pow;
	
	// incorrecto
	import java.util.*;
	import com.miempresa.*;
	import static java.lang.Math.*;

Seguridad	• Usar consultas parametrizadas en vez de concatenación de strings para las consultas en SQL. • Usar hashing para las contraseñas (BCrypt) • Evitar exposición de datos sensibles en logs o excepciones.

Ejemplos de seguridad	// correcto
	PreparedStatement stmt = conexion.prepareStatement("SELECT * FROM estudiante WHERE matricula = ?");
	stmt.setString(1, matricula);
	ResultSet rs = stmt.executeQuery();
	
	String contraseña = "miContraseñaSegura";
	String hash = BCrypt.hashpw(contraseña, BCrypt.gensalt());
	
	try {
	// Código que puede fallar
	} catch (Exception e) {
	logger.error("Error al procesar la solicitud: " + e.getMessage());
	}

Pruebas	• Usar JUnit para la automatización de las pruebas. • Deben seguir la convención de la clase Test.

	• Cada una debe ser independiente de las demás pruebas.

Ejemplo de Pruebas	import org.junit.jupiter.api.Test;
	import static org.junit.jupiter.api.Assertions.*;
	
	class FiguraGeometricaTest {
	
	  @Test
	  void testCalcularAreaCuadrado() {
	  FiguraGeometrica figura = new FiguraGeometrica();
	  double lado = 4.0;
	  double areaEsperada = 16.0; // lado * lado = 4 * 4 = 16
	
	  double areaCalculada = figura.calcularAreaCuadrado(lado);
	
	  assertEquals(areaEsperada, areaCalculada, "El área calculada no es correcta");
	  }
	}

