UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

Documento de Estándar de Programación

Plataforma Avanzada para la Generación Automática de Diagramas UML empresa Tech Solutions

Curso: Patrones de Software

Docente: Mg. Patrick Jose Cuadros Quiroga

Integrantes:

- Alexis Jeanpierre Martínez Vargas (2019063638)
- Juan José David Pérez Vizcarra (2019063636)
- Jhon Thomas Ticona Chambi (2018062232)

Tacna – Perú
2025

Plataforma Avanzada para la Generación Automática de Diagramas UML empresa Tech Solutions

Documento de Estándares de Programación

Versión *1.0*

Presentado Por:

Martínez Vargas, Alexis Jeanpierre
Documentador
2025

CONTROL DE VERSIONES					
Versión	**Hecha por**	**Revisada por**	**Aprobada por**	**Fecha**	**Motivo**

1.0	AMV	AMV, JTC,JJPV	AMV, JTC,JJPV	20/05/2025	Primera Versión

Índice General

1. Objetivo

El propósito de este estándar es reglamentar la implementación del código fuente del proyecto, estableciendo normas claras para la definición de variables, controles, clases, métodos, archivos y demás elementos del código.

A través de las reglas propuestas, se busca mejorar y uniformizar el estilo de

programación, garantizando que todos los desarrolladores sigan una estructura coherente y comprensible.
Para ello, se establecen las siguientes directrices:

- **Nombres mnemotécnicos**: Las variables deben seguir una nomenclatura que permita identificar su tipo de dato solo con ver su nombre.
- **Nombres sugestivos**: Tanto variables como funciones deben tener nombres descriptivos, de modo que su propósito y uso sean claros a simple vista.
- **Nomenclatura estandarizada**: La asignación de nombres a variables, métodos y clases debe seguir un esquema definido en este estándar, permitiendo un proceso automático y estructurado.
- **Compatibilidad con herramientas automáticas**: Se fomentará el uso de herramientas de análisis estático y verificación de nomenclatura para validar el cumplimiento de estas reglas.

La implementación de este estándar permitirá un código más legible, mantenible y comprensible, facilitando el trabajo en equipo y la evolución del sistema a lo largo del tiempo.

2. Declaración de Variables

La declaración de variables debe ajustarse a su propósito y cumplir con un conjunto de reglas que permitan una mejor organización y comprensión del código.

Para ello, se establecen los siguientes criterios:

➢ Longitud del Nombre de la Variable

El nombre de las variables debe ser lo suficientemente descriptivo para expresar su propósito, pero sin ser excesivamente largo.
Se establece una longitud máxima de 16 caracteres para asegurar un equilibrio entre claridad y facilidad de manejo.

➢ Alcance de la Variable

A medida que el proyecto crece, es fundamental reconocer rápidamente el alcance de las variables. Para ello, se utilizarán prefijos que indiquen su nivel de visibilidad dentro del código.

Alcance	Prefijo	Ejemplo
Global	G	gstrNombreUsuario
Nivel de la Clases	Cls	clsUsuarios
Local del procedimiento / método	Mtd	mtdIngresarUsuario
Público	Pbl	pblCantidadUsuario
Privado	Pr	prCantidadVenta

➢ Tipo de Dato

Las variables deben indicar su tipo de dato en su nomenclatura, utilizando un prefijo adecuado.

➢ Formato del Nombre

- Se debe seguir un estilo **camelCase**, donde la primera palabra del nombre comienza en minúscula y las siguientes con mayúscula.

- Ejemplo de estructura de nombres:

Estructura	Descripción
numCuenta	Variable de tipo numérica

2.1. Descripción de la Variable

El nombre asignado a una variable debe ser descriptivo y representativo de su propósito dentro del código. Su denominación debe reflejar claramente su uso y estar alineada con el contexto en el que se declara.

Ejemplos:

- idCuenta → Identificador único de una cuenta.
- tipoEstado → Indica el tipo de estado asociado a un registro.
- instalacion → Representa una instalación específica dentro del sistema.

2.2. Variables de Tipo Arreglo

Cuando una variable almacene un conjunto de elementos (arreglo o lista), se debe utilizar el prefijo lista para indicar que se trata de una estructura que contiene múltiples valores. Esto facilita la identificación rápida de su función y contenido.

Ejemplo:

- listaTiposDoc → Lista de diferentes tipos de documentos.

3. Definición de Controles

Para nombrar los controles dentro de cualquier aplicación de tipo visual, es necesario identificar el tipo de control al que pertenece y su función dentro de la aplicación. Esto facilita la legibilidad del código y su mantenimiento.

3.1. Tipo de datos

A continuación, se definen los prefijos mnemotécnicos para cada tipo de dato, con el objetivo de estandarizar la nomenclatura en el desarrollo del proyecto:

Tipo de variable	Mnemónico	Descripción
Byte	by	Entero de 8 bits sin signo.
Integer	int	Entero de 32 bits con signo.
Char	ch	Un carácter UNICODE de 16 bits
String	st	Cadena de caracteres
Date	dt	Formato de fecha/hora
Boolean	bln	Valor lógico: verdadero y falso
Float	fl	Comas flotantes, 11-12 dígitos significativos.
Double	dbl	Coma flotante, 64 bits (15-16 dígitos significativos)
Object	obj	Objeto genérico

3.2. Prefijo para el Control

El prefijo de cada control estará compuesto por tres caracteres representativos de

su tipo. Esto permitirá identificar fácilmente el tipo de control dentro del código. Por ejemplo, el control Button estará asociado al prefijo btn.

3.3. Nombre descriptivo del Control

El nombre del control debe estar compuesto por el prefijo correspondiente seguido de una descripción clara y específica de su función. Esto facilita la comprensión y el mantenimiento del código.

Tipo de control	Prefijo	Ejemplo
Label	lbl	lblNombre
TextBox	txt	txtApellido
Button	btn	btnLogin
RadioButton	rdo	rdoSeleccion
CheckBox	chk	chkRuta1
DropDownList	cmb	cmbDocumentos

3.4. Declaración de variables, atributos y objetos

Título	Descripción
Sintaxis	[TipoVariable] [Nombre de la Variable]
Descripción	Todas las variables o atributo tendrán una longitud máxima de 30 caracteres. El nombre de la variable puede incluir más de un sustantivo los cuales se escribirán juntos. Si se tuvieran variables que puedan tomar nombres iguales, se le agregará un número asociado (si está dentro de un mismo método será correlativo).
Observaciones	En la declaración de variables o atributos no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	Public String nombre Indica una variable o atributo que guardará un nombre.

3.5. Declaración de clases

Título	Descripción
Sintaxis	[Tipo] Class [Nombre de Clase]
Descripción	El nombre de las clases tendrá una longitud máxima de 30 caracteres y las primeras letras de todas las palabras estarán en mayúsculas. Tipo se refiere a si la clase será: Private, Public o Protected.
Observaciones	En la declaración de clases no se deberá utilizar caracteres como: • Letra Ñ o ñ.

	• Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,]. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	Private Class Empleado Indica una clase Empleado

3.6. Declaración de métodos

Título	Descripción
Sintaxis	nombreProcedim[(ListaParámetros)]
Descripción	El nombre del método constará hasta de 25 caracteres. La primera letra de la primera palabra del nombre será escrita en minúscula y las siguientes palabras empezarán con letra mayúscula.
Observaciones	En la declaración de métodos no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	Protected calcularSueldo(String empleado) Indica un método calcularSueldo que recibe una variable por valor de tipo string al ámbito de la clase

3.7. Declaración de funciones

Título	Descripción
Sintaxis	[TipoDato] nombreFuncion[(ListaParámetros)]
Descripción	El nombre del objeto constará hasta de 25 caracteres, no es necesario colocar un nombre que indique la clase a la cual pertenece. La primera letra de la primera palabra del nombre será escrita en mayúsculas El tipo de dato de retorno se coloca al final y será obligatorio colocarlo.
Observaciones	En la declaración de objetos no se deberá utilizar caracteres como: • Letra Ñ o ñ. • Caracteres especiales ¡, ^, #, $, %, &, /, (,), ¿, ‘, +, -, *, {, }, [,], _. • Caracteres tildados: á, é, í, ó, ú.
Ejemplo	Public int sumar(int A, int B) Indica una función que suma dos variables enteras

3.8. Control de versiones de código fuente

Para garantizar un adecuado control de versiones y facilitar la trazabilidad de los

cambios en el código fuente, cada modificación será almacenada siguiendo un formato estandarizado.

- **Formato de Nombres de Archivos de Versiones:**

[NOMBRE_DOCUMENTO]_[FECHA]_[HORA]

  - **FECHA:** Se registrará en formato YYYYMMDD (AñoMesDía).
  - **HORA:** Se utilizará el formato de 24 horas HHMM (HoraMinutos).
  - **Separador:** Se empleará el guion bajo _ para mejorar la legibilidad del nombre del archivo.

- **Extensiones Permitidas:**

  - **ZIP (.zip)**
  - **RAR (.rar)**

- **Ejemplo de Nombres de Archivo de Versiones:**

WSTENNIS_20070421_2056.zip

  - WSTENNIS → Nombre del documento o proyecto.
  - 20070421 → Fecha de la versión (21 de abril de 2007).
  - 2056 → Hora de la versión (20:56).
  - .zip → Formato de compresión utilizado.

Este método de almacenamiento permitirá identificar rápidamente la versión más reciente del código y recuperar versiones anteriores en caso de ser necesario.

3.9. Controles ADO.NET

En el desarrollo de aplicaciones que interactúan con bases de datos, es común el uso de ADO.NET. Para garantizar un código uniforme y legible, se establecen los siguientes estándares de nomenclatura para los objetos más utilizados en ADO.NET.

Prefijos para Objetos de ADO.NET

Componente	Prefijo
DataSet	Ds
DataTable	Dt
DataView	Dv
DataRow	Drw
Connection*	Cnn
Command*	Cmd
DataAdapter*	Da
CommandBuilder*	Bld
DataReader*	Dr

Ejemplos de Declaración de Objetos ADO.NET

- SqlDataReader drEmps = new SqlDataReader();
- SqlDataReader drCust = new SqlDataReader();
- DataSet dsEmps = new DataSet();
- DataSet dsCust = new DataSet();

4. Clases

El nombre de las clases debe ser autodescriptivo, permitiendo comprender su propósito sin necesidad de revisar su código interno.

Convenciones para nombres de clases:

- Se debe utilizar el prefijo cls en **minúscula**.
- El nombre de la clase debe ser en **PascalCase**, iniciando con mayúscula.
- No se deben usar espacios ni caracteres de subrayado _.

Ejemplos:

- clsCuenta
- clsEmpleado
- clsFactura

5. Métodos, Procedimientos y Funciones definidos por el Usuario

El nombre de los métodos debe ser autodescriptivo, siguiendo la estructura Verbo + Sustantivo, para que sea claro qué acción realiza y sobre qué entidad actúa.

Convenciones para nombres de métodos:

- El nombre debe indicar **qué hace** el procedimiento o función.
- Se usa un **verbo en infinitivo** seguido de un **sustantivo representativo**.
- Se debe **mantener coherencia** en los nombres y en el orden de palabras.
- **No se deben utilizar espacios ni caracteres de subrayado _.**

Ejemplos:

- modificarCuenta
- registrarEmpleado
- eliminarFactura

6. Beneficios

- Una buena documentación mejora la legibilidad y comprensión del programa.
- Documentar correctamente un programa desde el inicio evita que, en futuras modificaciones, se tenga que analizar en profundidad su funcionamiento, ahorrando tiempo y esfuerzo.
- Facilita la reutilización de módulos y rutinas, ya sea dentro del mismo programa o en otros proyectos.

- Ayuda a identificar cuándo un código debe ser reescrito. Si un fragmento de código es difícil de explicar mediante comentarios, probablemente necesite ser mejorado.

7. Conclusiones

- ✓ El uso de un estándar de programación bien definido ha sido clave para garantizar la calidad, coherencia y mantenibilidad del sistema de generación automática de diagramas UML, asegurando que el código sea comprensible y escalable para futuros desarrollos.

- ✓ La adopción disciplinada de convenciones de codificación, principios SOLID y buenas prácticas de arquitectura ha permitido una implementación más eficiente, reduciendo errores y facilitando la colaboración entre miembros del equipo de desarrollo.

- ✓ La documentación técnica clara y actualizada del sistema, tanto en el código como en los artefactos asociados, ha demostrado ser un componente esencial para la comprensión y evolución del proyecto, especialmente en entornos donde la automatización y la inteligencia artificial forman parte del núcleo funcional.