
Costa Rica Institute of Technology

Computer Engineering

# Laboratorio 1

Lee Sang Cheol

23 de febrero de 2025

# 1. Parte I: Estándares de Codificación

## 1.1. ¿Qué es el estándar de codificación?

Imaginamos que el equipo de personas está construyendo la casa. Si cada persona coloca los ladrillos de manera diferente, sin seguir un plan, la casa sería un desastre. En la programación pasa lo mismo: si cada programador escribe código a su manera, el resultado sería muy difícil de entender y mantener. Para evitar esto, existen los **estándares de codificación**, que son reglas que ayudan a escribir código de manera clara y organizada.

el estándar de codificación no solo define cómo estructurar el código, sino también cómo nombrar las variables, dónde y cómo escribir comentarios y qué reglas seguir para que el código sea más eficiente. Seguir estas reglas hace que el programa sea comprensible para cualquiera que lo lea, incluso si no fue la persona que lo escribió.

## 1.2. ¿Por qué son importantes estos estándares?

Siguiendo el estándar, logramos:

- **Código más fácil de leer**: Otros programadores pueden entender rápidamente lo que hiciste, sin necesidad de adivinar.
- **Menos errores**: Un código ordenado es más fácil de depurar y corregir, ya que se evitan errores comunes causados por el formato inconsistente.
- **Trabajo en equipo eficiente**: Todos siguen las mismas reglas, así que es más fácil colaborar y modificar código sin que se vuelva caótico.
- **Código reutilizable**: el código bien escrito y estructurado puede ser reutilizado en otros proyectos sin necesidad de grandes modificaciones.
- **Mejor mantenimiento**: Si en el futuro se necesita hacer cambios, será mucho más sencillo si el código sigue una estructura clara y bien documentada.

## 1.3. Tipos de reglas en los estándares de codificación

Los estándares incluyen distintas reglas, pero las más comunes son:

- **Estructura del código**: Define dónde colocar las llaves, cómo usar los espacios, las sangrías y el orden de los elementos dentro del código.
- **Nombres de identificadores**: Indica cómo nombrar variables, funciones y clases para que su significado sea claro.
- **Comentarios**: Explica cómo y cuándo escribir notas dentro del código para que otros programadores puedan entender mejor su propósito.
- **Buenas prácticas**: Sugerencias para escribir código eficiente, modular y fácil de mantener.

## 1.4. Convenciones para nombrar identificadores

Al igual que en la escuela cada materia tiene un nombre, en programación cada variable o función también debe tener un nombre claro. Existen distintas formas de nombrarlas:

- **Hungarian Notation**: Usa el prefijo según el tipo de variable. Ejemplo: `int iEdad`.
- **Camel Case**: La primera palabra es minúscula y las siguientes inician con mayúscula. Ejemplo: `calcularPromedio()`.
- **Pascal Case**: Similar a Camel Case, pero la primera palabra también empieza con mayúscula. Ejemplo: `MostrarDatos()`.
- **Underscore Naming**: Se usan guiones bajos para separar palabras. Ejemplo: `calcular_promedio()`.

El uso de una convención de nombres adecuada facilita la lectura y comprensión del código. Por ejemplo, si ves una variable llamada `totalVentasAnuales`, de inmediato sabes que guarda el total de las ventas en un año, sin necesidad de más explicaciones.

## 1.5. Beneficios de seguir un estándar

Seguir el estándar hace que el código sea más fácil de entender y mantener. También ayuda a trabajar mejor en equipo, ya que todos siguen las mismas reglas. Además, facilita la detección de errores y reduce el tiempo necesario para modificar el código en el futuro.

Algunos de los beneficios más importantes son:

- **Claridad**: Facilita la lectura del código para programadores nuevos en el proyecto.
- **Consistencia**: Permite que todo el código en un proyecto siga el mismo formato, sin importar quién lo escribió.
- **Eficiencia**: va a Ahorrar tiempo al evitar errores causados por la mala estructuración del código.
- **Facilidad de colaboración**: Equipos grandes pueden trabajar mejor si todos siguen las mismas reglas.

## 1.6. Preguntas y respuestas

**1. *¿Qué es el estándar de codificación?***

Es el conjunto de reglas que hacen que el código sea ordenado, claro y fácil de entender.

**2. *Tipos de estándares de nombres y ejemplos.***

**3. *¿Cuáles son los beneficios de usar un estándar?***

- Hace que el código sea más fácil de entender.
- Facilita el trabajo en equipo.
- Evita errores y mejora la calidad del software.
- Permite la reutilización del código en otros proyectos.

**4. *¿En qué partes se divide el estándar de codificación?***

- Estructura del código
- Nombres de identificadores
- Comentarios
- Buenas prácticas de programación

## 2. Parte II: Estándar de Codificación de Google para Java

Para garantizar que el código escrito en Java sea claro, estructurado y fácil de mantener, Google ha definido una serie de reglas que deben seguirse en el desarrollo de software. Estas normas ayudan a evitar errores, mejorar la legibilidad del código y facilitar la colaboración en equipo. A continuación, se presenta el resumen detallado de estas reglas en formato de tabla.

| **Categoría** | **Reglas según el estándar de Google** |
|---|---|
| **Terminología** | ▪ **Clase**: Se refiere a clases normales, enumeraciones, interfaces o tipos de anotaciones (`@interface`).<br>▪ **Miembro**: Incluye clases anidadas, campos, métodos y constructores.<br>▪ **Comentario**: Hace referencia a comentarios de implementación, no a documentación (Javadoc). |
| **Extensión del archivo** | Todos los archivos deben tener la extensión `.java` |
| **Codificación del archivo** | UTF-8 |
| **Nombre del archivo** | Debe coincidir con el nombre de la clase pública y respetar las mayúsculas y minúsculas. Solo debe haber una clase por archivo. |
| **Caracteres de espaciado** | Se usa espacio en lugar de tabulaciones para la indentación del código. La tabulación está prohibida. |
| **Estructura del Archivo Fuente** | |
| **Orden de los elementos** | ▪ Información de derechos de autor<br>▪ Declaración de paquete<br>▪ Importaciones ordenadas (sin `import.*`)<br>▪ Código de la clase (con estructura organizada) |

| | |
|---|---|
| **Declaración de Clases** | ▪ Cada clase de nivel superior debe estar en su propio archivo.<br>▪ El orden dentro de la clase debe seguir la lógica clara (atributos primero, luego constructores, luego métodos).<br>▪ Métodos sobrecargados deben agruparse juntos sin otros métodos intercalados. |
| **Formato y Estilo** | |
| **Uso de llaves** | Se deben usar en todas las estructuras de control `if, else, for, do, while`, incluso si el cuerpo tiene una sola línea. |
| **Indentación** | Se deben usar 2 espacios por nivel de indentación. No se deben usar tabulaciones. |
| **Límite de columna** | No debe exceder los 100 caracteres para facilitar la lectura. |
| **Líneas en blanco** | Se recomienda insertar líneas en blanco entre bloques lógicos de código para mejorar la legibilidad. |
| **Uso de paréntesis** | Se recomienda agrupar expresiones para mejorar la claridad del código. |
| **Convenciones de Nombres** | |
| **Nombres de clases** | Deben seguir `PascalCase`, por ejemplo: `MiClasePrincipal`. |
| **Nombres de métodos** | Deben seguir `camelCase`, por ejemplo: `calcularPromedio()`. |
| **Nombres de variables** | Deben ser descriptivos y utilizar `camelCase`. |
| **Constantes** | Se deben escribir en mayúsculas con guiones bajos. Ejemplo: `VALOR_PI`. |
| **Buenas Prácticas de Programación** | |

| | |
|---|---|
| **Manejo de Excepciones** | ▪ Las excepciones capturadas no deben ignorarse.<br>▪ Se debe evitar capturar excepciones genéricas como `Exception`.<br>▪ Si la excepción se ignora intencionalmente, debe estar documentada con un comentario claro. |
| **Uso de miembros estáticos** | Los miembros estáticos deben ser calificados con el nombre de la clase que los contiene. |
| **Finalizadores** | No se deben usar finalizadores en la mayoría de los casos. |
| **Javadoc** | |
| **Uso de Javadoc** | ▪ Se debe documentar cada clase pública y método público.<br>▪ Se deben usar etiquetas como `@param`, `@return`, `@throws` cuando corresponda.<br>▪ Documentar métodos de prueba con nombres descriptivos. |
| **Comentarios** | ▪ Se deben usar comentarios `//` para comentarios en una línea.<br>▪ Se deben usar comentarios de bloque `/* */` solo cuando sea necesario.<br>▪ Evitar comentarios redundantes. |

# 3. Parte III: Instalación de aplicaciones de software

Para el desarrollo de los laboratorios y proyectos del curso, es necesario instalar el siguiente software:

- **Java Development Kit (JDK) v23**
- **IDE BlueJ**
- **IDE NetBeans**

A continuación, se presentan las capturas de pantalla como evidencia de la instalación:
También se verifica la correcta instalación de Java ejecutando el siguiente comando en la terminal:

```
javac --version
```

| Tipo | Características | Ejemplo |
|---|---|---|
| Hungarian | Usa prefijo según el tipo de variable | `int iEdad` |
| Camel Case | Primera palabra minúscula, siguientes con mayúscula | `calcularPromedio()` |
| Pascal Case | Todas las palabras inician con mayúscula | `MostrarDatos()` |
| Underscore | Palabras separadas con _ | `calcular_promedio()` |

Cuadro 1: Tipos de nombres de identificadores

```
C:\Users\diego>java -version
java version "23.0.2" 2025-01-21
Java(TM) SE Runtime Environment (build 23.0.2+7-58)
Java HotSpot(TM) 64-Bit Server VM (build 23.0.2+7-58, mixed mode, sharing)
```

Figura 1: Evidencia de instalación de Java Development Kit v23

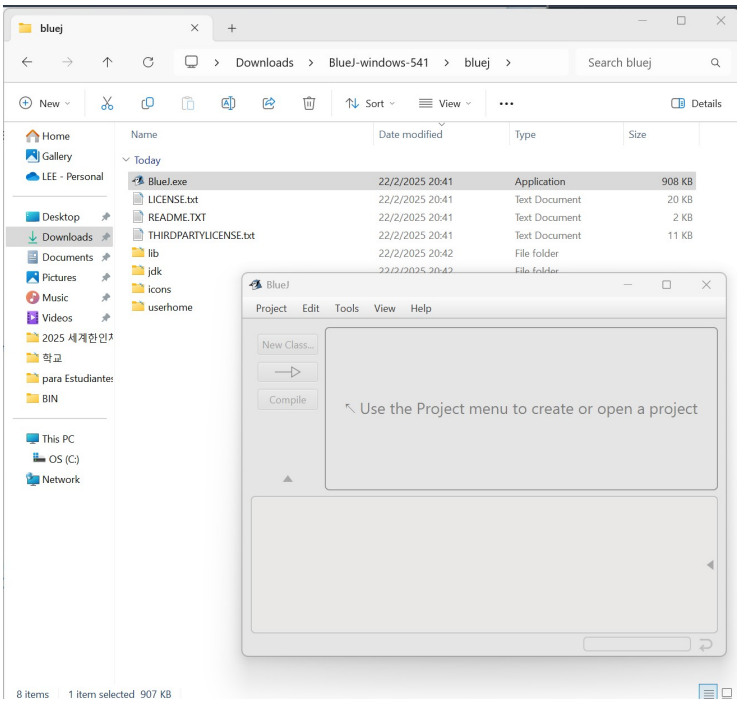


Figura 2: Evidencia de instalación de BlueJ

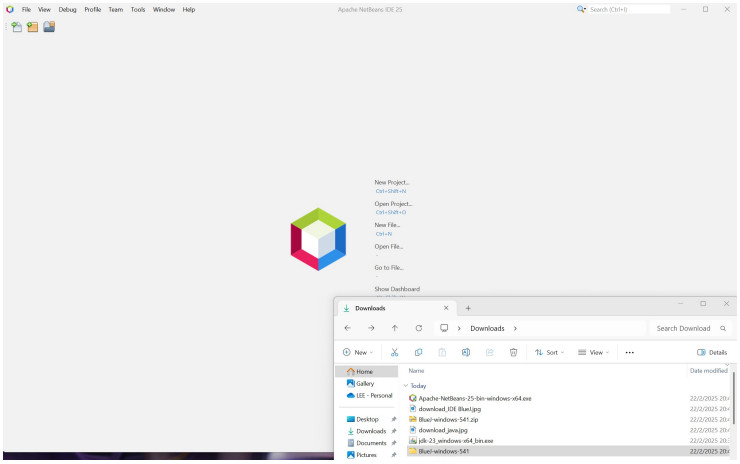


Figura 3: Evidencia de instalación de NetBeans