ESCUELA POLITÉCNICA NACIONAL

PROGRAMACIÓN

GR-1

GRUPO 5

INTEGRANTES:

Paulette Cushicagua
Adrián Jara
Danny Ponce
Shamir Rivera
Larsson Umatambo
Diego Villagómez

Lunes, 26 de mayo del 2025

TAREA NO.2

OBJETIVO:

Asimilar los principios del desarrollo de código limpio propuestos por Robert C. Martin en “Código Limpio - Manual de estilo para el desarrollo ágil de software” (2018), con énfasis en la autodescripción del código, la eliminación de redundancias y el uso adecuado de comentarios, para lograr un código más claro, mantenible y expresivo.

DESAROLLO:

1. **El código debe ser autodescriptivo**

Un código debe ser autodescriptivo se refiere a que el código debe explicarse por sí mismo, sin ayuda de comentarios o especificaciones extras, es decir, que la mayoría de las personas pueden entenderlo sin necesidad de lo anterior mencionado. Para que esto sea posible debe tener nombres claros, que las funciones estén bien definidas, con una buena estructura evitando abreviaciones que puedan causar confusión y comentarios solo en caso de ser necesarios

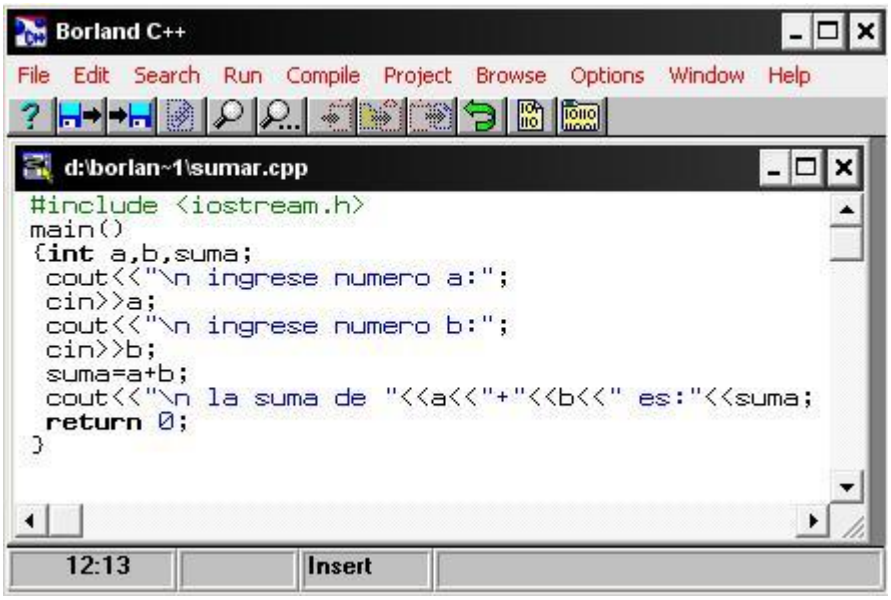

Imagen 1: Código para sumar dos números fácil de entender [1].

2. **No sea redundante**

Un código es redundante si describe algo que ya se define correctamente por sí mismo, quiere decir que no repitas innecesariamente variables, condiciones o instrucciones que ya están definidas. Esto hace que el programa sea más largo, difícil de entender y mantener incluso se puede ahorrar varias líneas de código. En lugar de copiar y pegar líneas parecidas, usar funciones es una buena idea. Evitar redundancia ahorra tiempo,

reduce errores y mejora la claridad del código, la idea es escribir un código limpio y claro es mucho mejor que escribir mucho.

```
/**
* @param sellRequest
* @return
* @throws ManagedComponentException
*/
public SellResponse beginSellItem(SellRequest sellRequest)
   throws ManagedComponentException
```

Imagen 2: comentario redundante [2]

3. **No añada ruido obvio**

Cuando añadimos un comentario este debe añadir información que resulte útil para entender el código, no con información que resulte innecesaria e incluso repita lo mismo que ya se encuentra escrito en el código; de tal manera que si un comentario genera un ruido o no aporta en algo a la comprensión es mejor simplemente eliminarlo

```
// if the student is at least 18 years of age
if (student.age> = 18){
// send meeting invitation to the student notificationService.sendMessageTo(student, meetingI
nvitation);
}
else // if the student is younger than 18 years
{
// sends a meeting invitation to the student's legal guardian notificationService.sendMessage
To(student.parent,meetingInvitation);
}
```

Imagen 3: Comentario con Ruido Obvio [3]

4. **No use corchetes de cierre en los comentarios**

A la hora de comentar es preferible no colocar corchetes de cierre por varias razones, las cuales son: Puedes confundir a las personas quien está leyendo el código, puede parecer parte de algo diferente y no un comentario, llega a ser innecesario si solo lo usamos para encerrar texto, los corchetes no aportan nada necesario a nuestros comentarios y estos deben ser los más claros y simples posibles para evitar confusión

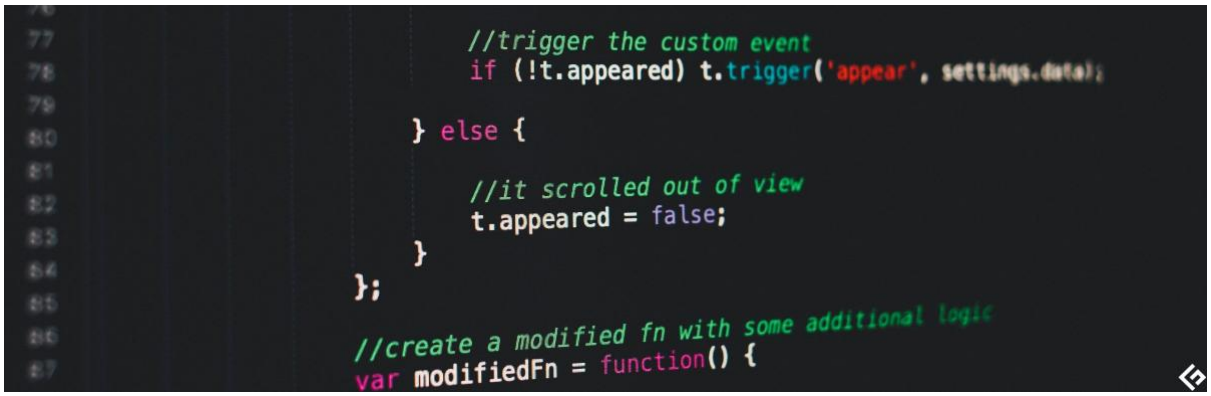

Imagen 4: Comentarios simples y claros en código [4].

5. **No comente código. Simplemente elimínelo.**

Los comentarios son fragmentos añadidos antes o después de un código, con la finalidad de explicar los procesos que estén sucediendo. Al hacer uso de comentarios se está compensando la incapacidad que existe para expresarse solo en el código, pero al realizar un uso excesivo o incensario provoca una dificultad en la comprensión clara del código.

Los lenguajes de programación no son lo suficientemente expresivos o no expresan las intenciones de manera directa, por ello se necesitan de los comentarios, pero si se escribiese un código mucho más expresivo y claro desde el principio, los comentarios serían mínimos o nulos.

Un código puede ir cambiando o evolucionando, y así ciertos fragmentos irán variando de posición para optimizar o agrandar su funcionamiento. Mientras, más antiguo sea el código, el comentario ira quedando, desactualizando y terminara obsoleto.

Por ello el eliminar los comentarios (no solo los incensarios) forzamos a que el código sea más claro y expresivo para que el lector puedo entender el código sin la necesidad

de comentarios, además de que el código será adaptable a futuras versiones.

```
Algoritmo codigo_sincomentarios
    Escribir "Ingrese el valor al que pertenecera k";
    Leer k;
    a ← 0;
    b ← 1;
    Para i←1 Hasta k Con Paso 1 Hacer
        Escribir a, " " ;
        c ← a + b;
        a ← b;
        b ← c;
    FinPara
FinAlgoritmo
```

Imagen 5: Código sin comentarios [5].

6. **Utilícelos como explicación de intenciones.**

En ocasiones, un comentario pasa de ser solo información útil de la implementación, a proveer la intención que hay tras una decisión. Los comentarios no son una disculpa por un mal código, sino una herramienta para explicar lo inexplicable (Martin, 2008), el comentario deberá ser usado para explicar a los demás lo que el código no pueda explicar por sí mismo, como en este punto, explicar las intenciones.

```
#include <iostream>
#include <vector>
#include <unordered_set>

// Eliminamos duplicados manteniendo el orden de inserción
// Usamos un unordered_set para verificar duplicados rápidamente (O(1)),
// pero mantenemos un vector para conservar el orden original.
std::vector<int> removeDuplicates(const std::vector<int>& input) {
    std::unordered_set<int> seen;
    std::vector<int> result;

    for (int num : input) {
        if (seen.find(num) == seen.end()) {
            seen.insert(num);
            result.push_back(num);
        }
    }

    return result;
}

int main() {
    std::vector<int> numbers = {1, 2, 2, 3, 1, 4, 5, 3};
    std::vector<int> unique = removeDuplicates(numbers);

    for (int n : unique) {
        std::cout << n << " ";
    }

    return 0;
}
```

Imagen 6: Explicación de intenciones en el código [6].

7. **Utilícelos como aclaración del código.**

Los comentarios deben ser usados para aclarar el propósito del código o para añadir información extra a este, la cual resulte útil para quienes trabajaran en el posteriormente, así como dar explicaciones que no son evidentes a simple vista.

```
' validar si el ISIN existe en RNVI ( 1ra validacion)
Dim valorOficial As Model.EmisionAutorizada = laListaDeEmisiones.FirstOrDefault(Function(item) item
If valorOficial Is Nothing Then
    mBitacora.Add(String.Format(MENSAJE_NO_EXISTE_EN_REGISTRO_OFICIAL, laFecha, activoLocal.CodIsir
    Exit For
End If

' valida si el Emisor existe ( 2da validación)
Dim elEmisor = laListaDeEmisores.FirstOrDefault(Function(item) item.DesEmisor.Equals(valorOficial.D
If elEmisor Is Nothing Then
    mBitacora.Add(String.Format(MENSAJE_EMISOR_NO_REGISTRADO, laFecha, valorOficial.DesEmisor, acti
    Exit For
End If

' validar si la moneda de activo es válida (3ra validacion)
If Not MonedasPermitidas.Contains(valorOficial.CodMoneda) Then
    mBitacora.Add(String.Format(MENSAJE_MONEDA_DE_ACTIVO_NO_ES_VALIDA, laFecha, activoLocal.CodIsir
    Exit For
End If
```

Imagen 7: Ejemplo de un comentario que brinda una aclaración al código [7].

8. **Utilícelos como advertencia de las consecuencias**

Usar comentarios como advertencia de las consecuencias significa avisar al usuario o programador sobre riesgos al usar cierto código. Sirve para prevenir errores graves, como borrar datos o afectar el rendimiento, estos comentarios son una forma de proteger el sistema y al programador, se colocan antes de métodos peligrosos o delicados. Deben ser claros y directos, indicando lo qué puede llegar a salir mal. No es solo explicar qué hace el código, sino qué pasaría si se usa mal. Ayudan a otros desarrolladores a tomar decisiones con cuidado. También evitan que alguien modifique funciones sensibles sin saber los efectos.

```
/**
 * ¡Advertencia!: Borra todos los datos del usuario. Acción irreversible.
 */
void deleteUserData(const std::string& userId) {
    // Lógica para borrar datos
    std::cout << "Usuario " << userId << " eliminado.\n";
}
```

Imagen 8; advertencia a una consecuencia [8].

Conclusión:

El desarrollo de software de calidad no solo depende de que el código funcione correctamente, sino de que sea comprensible, mantenible y claro para cualquier desarrollador que lo lea. A través del análisis y aplicación de los principios planteados en *"Código Limpio"* de Robert C. Martin, se destaca la importancia de escribir un código autodescriptivo, evitar redundancias innecesarias y utilizar los comentarios de manera estratégica, no como sustituto de una mala implementación, sino como complemento para explicar intenciones o advertir sobre posibles consecuencias.

Adoptar estas buenas prácticas permite reducir errores, facilitar la colaboración entre equipos y garantizar la evolución continua del software a lo largo del tiempo. Un código limpio no solo refleja profesionalismo, sino que también construye una base sólida para el desarrollo ágil y sostenible.

Referencias:

Cecil, R. (2018). Código Limpio Manual de estilo para el desarrollo ágil de software [Archivo PDF]. https://elhacker.info/manuales/Lenguajes%20de%20Programacion/Codigo%20limpio%20-%20Robert%20Cecil%20Martin.pdf.com

[1] Suma de dos números enteros en c++. (s.f.). *Programa código:* https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEg2wWH-kgYXp3jUlDE-Qydq1dcS-5zYkCCDM19jLpdaLMJ46_Y_ru2St6vJuQS9dCv5Cj8AlA_MtiP9qcRLSPqNOe_T4bc4GEVmXoYZuULikUoD5TJYvXO2-uqWDY8ccjX5dSdKaBhRQuNm/s1600/1im.JPG.com

[2] *Elhacker.INFO - Descargas cursos, manuales, tutoriales y libros*. (s. f.). https://elhacker.info/manuales/Lenguajes%20de%20Programacion/Codigo%20limpio%20-%20Robert%20Cecil%20Martin.pdf.com

[3] *9 consejos que promueven el código limpio: escribir comentarios de una buena manera*. (2020b, noviembre 28). ICHI.PRO. https://ichi.pro/es/9-consejos-que-promueven-el-codigo-limpio-escribir-comentarios-de-una-buena-manera-142952749337701.com

[4] Geekflare team. (2025). *Ejemplo de adición de comentarios en 15 lenguajes de programación*. Geekflare. https://geekflare.com/es/wp-content/uploads/2021/02/code-comments.jpg.com

[5] Rivera, S. (2025). *codigo_sin_comentarios* [Imagen]. Elaboración propia.

[6] Jara, Adrián. (2025). *Explicación_de_intenciones* [Imagen]. Elaboración propia.

[7] Oscarcenteno. (2016, 11 octubre). *El buen uso de los comentarios en el código fuente*. Software Mantenible.com. https://softwaremantenible.wordpress.com/2016/10/10/el-buen-uso-de-comentarios-en-el-codigo-fuente/.com

[8] *Elhacker.INFO - Descargas cursos, manuales, tutoriales y libros*. (s. f.-b). https://elhacker.info/manuales/Lenguajes%20de%20Programacion/Codigo%20limpio%20-%20Robert%20Cecil%20Martin.pdf.com