ZOMBIE´S WAR: Ataque Final

Documento de Estándares de Programación

Versión 1.0

Historia de Revisión

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Versión | Hecha por | Revisada por | Aprobada por | Fecha | Motivo |
| 1.0 | Rene Poma, Jhon Salluca y Juan Lostaunau |  |  | 05/06/2024 | Versión 1.0 |
| 2.0 | Rene Poma, Jhon Salluca y Juan Lostaunau |  |  | 19/06/2024 | Versión 2.0 |

Tabla de Contenidos

[**1. OBJETIVO 4**](#_Toc170740317)

[**2. NAMING: NOMBRADO DE VARIABLES Y FUNCIONES 5**](#_Toc170740318)

[**2.1 Nombres Significativos 5**](#_Toc170740319)

[**2.2 Longitud y precisión: 5**](#_Toc170740320)

[**2.3 Uso de prefijos y sufijos: 5**](#_Toc170740321)

[**3. Convenciones de nombrado 5**](#_Toc170740322)

[**3.1 CamelCase y PascalCase 5**](#_Toc170740323)

[**4. Estructura del código 5**](#_Toc170740324)

[**4.1 Declaraciones de Variables 5**](#_Toc170740325)

[**4.2 Métodos y Funciones 5**](#_Toc170740326)

[**5. Comentarios 5**](#_Toc170740327)

[**6. Formateo 5**](#_Toc170740328)

[**7. Manejo de Errores 5**](#_Toc170740329)

[**8. Beneficios 5**](#_Toc170740330)

[**9. Conclusiones 6**](#_Toc170740331)

Estándares de Programación

# OBJETIVO

Establecer estándares de programación siguiendo los principios de Clean Code para garantizar que el código fuente sea legible, mantenible y eficiente. Esto implica:

* Uso de nombres significativos y descriptivos para variables y funciones.
* Implementación clara y concisa de lógica.
* Mantenimiento de un estilo de codificación coherente.
* Facilitación del uso de herramientas de revisión de código.

# NAMING: NOMBRADO DE VARIABLES Y FUNCIONES

## Nombres Significativos

* Variables:

Usar nombres que revelen intención. Ejemplo: `contadorDeUsuarios` en lugar de `cnt`.

* Globales: Minimizar el uso de variables globales en caso de usarse deben estar claramente justificadas en un archivo independiente, `ESTADO\_USUARIO` en lugar de `estusr`
* Privadas: Deben ser claras con nombres descriptivos.
* Públicas: Evitarse en medida de lo posible con getters y setters.
* Funciones/Métodos  
  Nombrar con verbos que reflejen la acción realizada, por ejemplo, `eliminarUsuario` en lugar de `elim`.

## Longitud y precisión:

* Preferir nombres descriptivos de longitud razonable, como `actualizarDireccionDeEnvio` en vez de `actDir`

## Uso de prefijos y sufijos:

* Solo usarlos cuando aporten significado claro, por ejemplo `esVisible`, `tieneAcceso`. Generalmente en valores booleanos.

# Convenciones de nombrado

## CamelCase y PascalCase

* CamelCase: Usar para nombres de variables y funciones, iniciando con minúscula, por ejemplo, `calcularTotal`.
* PascalCase: Aplicar en nombres de clases empezando con mayúscula, como `GestorDeUsuarios`

# Estructura del código

## Declaraciones de Variables

* Declarar cerca de su uso y agruparlas por propósito
* Inicializar al momento de la declaración siempre que sea posible.

## Métodos y Funciones

* Mantenerlos cortos, realizando una sola acción.
* Considerar dividirlos si exceden 20 lineas.

# Comentarios

* Explicar el “por qué” más que el “qué”, evitando comentarios obsoletos o redundantes. Solo en caso de ser necesarios.

# Formateo

* Mantener un estilo de codificación coherente con reglas de indentación, espaciado y uso de llaves.

# Manejo de Errores

* Preferir excepciones en lugar de códigos de retorno, con mensajes claros y descriptivos.

# Beneficios

* Seguir estos estándares incrementa la legibilidad, facilita la colaboración y minimiza el mantenimiento.
* Facilita la reutilización de módulos y rutinas desde cualquier otro programa o el mismo, promoviendo un desarrollo de software más eficiente y modular.

# Conclusiones

* Una buena programación e implementación legible se logra mediante la adhesión a un estándar o patrón de programación robusto.
* Es crucial que el programador conozca bien el estándar utilizado, o en su defecto, que consulte el documento de estándares para minimizar discrepancias y errores en el código.