Universidad Veracruzana

El Ahorcado – Estándar de codificación

Proyecto Final de la Experiencia Educativa de Tecnologías para la Construcción de Software

Equipo 5
Luis Ángel Elizalde Arroyo
Daniel Mongeote Tlachy

20/04/2025

Propósito	Promover prácticas de programación que resulten en un software seguro, fiable, verificable y de fácil mantenimiento.
Idioma	Se ha decidido usar el idioma Español para la nomenclatura utilizada en este estándar; la decisión fue tomada por todos los miembros del equipo con fines académico y de mejor nuestro aprendizaje en esta tecnología.
Estructura del código fuente	Se ha decidido organizar los archivos de código fuente en carpetas lógicas según la funcionalidad. Se ha decidido dividir el código fuente del lado del **Cliente** en las siguientes capas: **Vistas** (interfaces que el Cliente consume), **Alertas** (interfaces tipo pop-up que el Cliente recibe dependiendo una acción realizada), **Utilidades** (clases que contienen código especifico de ayuda para ciertas funcionalidades), **SocketCliente** (método de comunicación que gestiona funcionalidades como conectar, escuchar y enviar mensaje al SocketServidor) y **Seguridad** (clase que ayuda al proceso de validación del inicio de sesión). Por otro lado, del lado del **Servidor** se decidió dividir el código fuente en las siguientes capas: **Modelo** (definición de la estructura de las clases cargadas desde la base de datos), **HangmanServicio** (definición de las funcionalidades especificadas en Servicios) y **SocketServidor** (método de comunicación que gestiona la lógica de la partida). Adicionalmente, se decidió agregar una división más que trabaja en común con ambas capas para ser consumido, la cual es **Biblioteca** que esta dividido de la siguiente manera: **DTO** (definición de los Data Transfer Objects que son usados en la lógica del proyecto) y **Servicios** (definición de las funcionalidades necesarias para ser consumidas).

Nombrado de clases	Las Clases deben tener como nombre un sustantivo o una frase que haga alusión al mismo (persona, cosa, etc.) y empezar con letra mayúscula. Se debe evitar nombre con verbos.
	a) Se ocupará como formato el **UpperCamelCase** para el nombrado de clases.
	Como ejemplo de nombrado de clases se tiene:
	`private class PalabraDTO{...}`
	`private class JugadorDTO{...}`

Nombrado de Componentes de Interfaz	Los componentes de una interfaz gráfica tales como botones, campos de texto, etiquetas, etc. deben de empezar con una abreviación del componente.
	• Para abreviar un componente de una sola palabra, se deben quitar sus vocales y ocupar las tres primeras consonantes.
	• Para abreviar un componente formado de dos palabras: de la primera palabra, se deben quitar sus vocales y ocupar las tres primeras consonantes, y juntar esto con la segunda palabra, empezando por mayúscula.
	• Se ocupará el formato **lowerCamelCase**.
	Como ejemplos de nombrado se tiene:
	`Label x:Name="txtUsuario"`
	`Label x:Name="txtFechaNacimiento"`
	`Label x:Name="psBContrasenia"`
	`Label x:Name="btnRegresar"`

Nombrado de Variables	Las variables del programa deben tener como nombre un sustantivo que haga alusión a el mismo y empezar con una letra minúscula, se ocupara como formato el **lowerCamelCase**.
	Como ejemplo de nombrado de variables se tiene:
	// Mal nombrado
	`var token = "";`
	// Buen nombrado
	`var token = new string[] {};`

Nombrado de Constantes	Las constantes del programa deben tener como nombre un sustantivo que haga alusión a el mismo y se escribirán con mayúsculas, además serán nombradas con el formato **SCREAMING_SNAKE_CASE.**
	Como ejemplo de nombrado de constantes se tiene:
	// Mal nombrado
	`const int direccion = 192.168.0.1;`
	// Buen nombrado
	`const int direccionIP = 192.168.0.1;`

<table>
<tr><td>Nombrado de namespace</td><td>Los namespace permiten declarar que todos los tipos de un archivo están en un único espacio de nombres. El nombre elegido para un namespace debe indicar la funcionalidad disponible por tipos en el espacio de nombres; serán nombrados con el formato UpperCamelCase.
Como ejemplo de nombrado de namespace se tiene:
// Mal nombrado
namespace Vistas
// Buen nombrado
namespace HangmanGame_Cliente.Cliente.Vistas</td></tr>
<tr><td>Nombrado de Métodos</td><td>Los métodos de deben ser verbos o frases que hagan alusiones a verbos y se ocuparan prefijos en caso de ser necesario, además se utilizara el formato UpperCamelCase.
Como ejemplo de nombrado de métodos se tiene:
public static void ActualizarJugador(){…}
private async Task NotificarClientes(string codigoPartida, string mensaje){…}
private void AjustarTextos(string nickname){…}</td></tr>
<tr><td>Comentarios</td><td>Los comentarios en programación se utilizan para poner aclaraciones del código, y así es más fácil de entender lo que hace.
a) Explicar el por qué, no el cómo.
b) Documentar si existe algo inusual o inesperado dentro del código.
c) Evitar palabras altisonantes, chistes, bromas o cualquier comentario innecesario.</td></tr>
<tr><td>Buenos Comentarios</td><td>Un buen comentario describe el que hace cierta parte del código y no el cómo. Todos los comentarios deben ser claros y consistentes, utilizando mismo estilo. Como ejemplo se tiene el siguiente:
// Limpiar clientes desconectados
lock (clientesPartida)
{
if (!clientesPartida.Any())
{
clientes.TryRemove(codigoPartida, out _);
}
}</td></tr>
<tr><td>Malos Comentarios</td><td>En un mal comentario se invierte mucho tiempo en generar cajas de comentarios “bonitas”, esto hace que se pierda mucho tiempo intentando acomodar cada caja cada vez que el comentario sea editado. Como ejemplo se tiene el siguiente:
// <summary>
// User Class
// </summary>
public class User
{
// <summary>
// User Type
// </summary>
public string Type { get; set; }

// <summary>
// Verifies if user is active
// </summary>
public bool IsActive { get; set; }
}</td></tr>
</table>

<table>
<tr><td>Indentación</td><td>La indentación (o mejor conocido como sangría) es utilizado para una mejor lectura y comprensión del código. Esta debería seguir las siguientes características:
No hay un salto de línea antes de abrir una llave.
Hay un salto de línea después de la llave de cierre, solo si esa llave termina una declaración o el cuerpo de un método, constructor o clase con nombre. Por ejemplo, no hay salto de línea después de la llave si va seguido de un else o “;”.
Poner una sola instrucción por línea.
Dejar un espacio entre cada uno de los elementos de una expresión.
Agregar una tabulación en las instrucciones que están dentro de estructuras de control, métodos o bloques específicos de código.
Como ejemplo se tiene el siguiente bloque de código:

<pre>
public class example {
  public void method() {
    if(condition()) {
      try{
        something();
      }catch (ProblemException e) {
        recover();
      }
    }else if (otherCondition()) {
        somethingElse();
    }else {
        lastThing();
    }
  }
};
</pre></td></tr>
<tr><td>Capitalización</td><td>lowerCamelCase: La primera letra no está en mayúsculas, pero las siguientes palabras ocupan una letra mayúscula para diferenciar unas palabras de otras.

UpperCamelCase/Pascal Case: La primera letra está en mayúsculas y las primeras letras de las subsecuentes palabras se encuentran también en mayúsculas.

SCREAMING_SNAKE_CASE: Las palabras se escriben en mayúsculas separadas por un “_”.
Algunos ejemplos son:
lowerCamelCase:
private double workedHours = 0.00;
UpperCamelCase/Pascal Case:
private class PlayerGuest{...}
SCREAMING_SNAKE_CASE:
private double MEXICAN_PESO = 18.50;</td></tr>
</table>