Integrantes

*Albhieri Cristoff Villa Contreras  
Raúl Hernández Olivares  
Victoria Priscilla Moyano Arguelles  
Miguel Ángel Morales Cruz*

Estándar de codificación

Proyecto final

# Estándar de codificación

|  |  |
| --- | --- |
| Propósito | El propósito de este documento consiste en ser una guía para desarrollar aplicaciones de escritorio para Windows en base al lenguaje de programación C# con el marco de trabajo WPF (Windows Presentation Foundation). |
| Declaraciones | En los archivos .cs se deben seguir los siguientes criterios:   * Realizar solo una declaración por línea. * Escribir las declaraciones al principio del bloque de código donde serán utilizadas. * Inicializar las variables locales donde son declaradas, a menos que la solución del código lo requiera distinto. |
| Formato de Declaraciones | public class InicioSesion()  {  private string \_rol; //Asignar estilo a corde  private int \_numeroSerie = 53;  private int \_numeroAbsoluto = 28;  ...    If (\_numeroSerie > \_numeroAbsoluto)  {  Int numeroSinAbsoluto = \_numeroSerie -  \_numeroAbsoluto;  Label\_NumeroSerie.Content =  \_numeroSinAbsoluto;  }  else  {  Label\_NumeroSerie.Content =  \_numeroSerie;  }  } |
| Espaciado | Se deben dejar dos saltos de línea en las siguientes circunstancias:   * Entre métodos. * Entre las variables locales de un método y su primera sentencia. * Antes de un comentario (excepto el encabezado). * Antes de estructura de control * Al iniciar y al finalizar un bloque de código (excepto si es solo una línea dentro del bloque).   Se debe dejar un espacio en blanco en las siguientes circunstancias:   * Una palabra reservada del lenguaje seguida por un paréntesis. * Después de cada coma en las listas de argumentos. * Antes y después de operadores binarios. * Las expresiones dentro de la cláusula. de control de una sentencia for. * Antes de una llave de apertura. |
| Formato de Espaciado | private void Button\_Close\_MouseLeave(object sender, MouseEventArgs e)  {    Button\_Close.Background =  (SolidColorBrush) Application.Current.Resources[  "SolidColorBrush\_Gunmetal"];  Label\_Close.Foreground = (SolidColorBrush) Application.Current.Resources["SolidColorBrush\_Saftron"];    }    private void Button\_Exchange\_MouseLeftButtonDown(  object sender, MouseButtonEventArgs e)  {    if(Grid.GetRow(Border\_MXN) == 0)  {    TextBlock\_Leyenda.Text = "Desde Dólar de EE.UU. hacia Peso mexicano";  TextBox\_CurrencyFrom.IsEnabled = false;  TextBox\_CurrencyTo.IsEnabled = true;    Grid.SetRow(Border\_MXN, 1);  Grid.SetRow(Border\_USD, 0);    if (TextBox\_CurrencyFrom.Text.Length ==  0)  {  TextBox\_CurrencyFrom.Text = "0";  }    }  else  {    TextBlock\_Leyenda.Text = "Desde Peso mexicano hacia Dólar de EE.UU.";  TextBox\_CurrencyTo.IsEnabled = false;  TextBox\_CurrencyFrom.IsEnabled = true;    Grid.SetRow(Border\_MXN, 0);  Grid.SetRow(Border\_USD, 1);    if (TextBox\_CurrencyTo.Text.Length == 0)  {  TextBox\_CurrencyTo.Text = "0";  }    }    } |
| Indentación | Para la indentación se debe seguir lo siguiente:   * Se deben usar 4 espacios como unidad de indentación * Se debe indentar el contenido dentro de cada nuevo bloque de código * Cuando existan sentencias anidadas se debe aumentar el nivel de indentación * Las expresiones separadas por su longitud en varias líneas, la nueva línea quedará al nivel de dos tabulaciones a la derecha respecto a donde comienza |
| Formato de Indentación | ...  if (TextBox\_CurrencyFrom.Text.Length ==  0)  {  If (TextBox\_CurrencyFrom.Text != “0”)  {  TextBox\_CurrencyTo.Text = "0";  }  }  ... |
| Longitud de la Línea | Evitar la línea de más de 80 caracteres.    Cuando una expresión no entre en una sola línea, separarla de acuerdo con los siguientes criterios:   * Después de una coma * Antes de un operador * Después de un signo +.     Una expresión separada en varias líneas será considerada como solo una línea de código para la aplicación de las reglas de líneas en blanco. |
| Formato de Longitud de la Línea | ...  if(Grid.GetRow(Border\_MXN) == 0)  {    TextBlock\_Leyenda.Text = "Desde Dólar “ +  “de EE.UU. hacia Peso mexicano";  TextBox\_CurrencyFrom.IsEnabled = false;  TextBox\_CurrencyTo.IsEnabled = true;    Grid.SetRow(Border\_PesosMexicanos,  1);  Grid.SetRow(Border\_USD, 0);    if (TextBox\_CurrencyFrom.Text.Length ==  0)  {  TextBox\_CurrencyFrom.Text = "0";  }    }  ... |
| Jerarquía | Los modificadores tienen el siguiente orden de declaración para variables, constantes y métodos:   1. Static 2. Public 3. Private 4. Protected 5. Internal   Variables y constantes  Se declaran en el siguiente orden:   1. Constantes 2. Resto de variables     Métodos  Los métodos se declaran después de constantes y variables en el siguiente orden:   1. Métodos escuchadores (En caso del code behind) 2. Resto de métodos |
| Sentencias | * Siempre deben usar llaves de apertura y cierre. * La llave de apertura se coloca en la siguiente línea que en la que se encuentra la estructura de control. * La llave de cierre se coloca en una línea separada y al mismo nivel de indentación que donde inicia la estructura de control. * Cada vez que un case no incluya la sentencia break en un switch, incluir un comentario donde se encontraría normalmente. |
| Formato de Sentencias | ...  public string MensajeError  {  get  {  \_mensajeError;  }    set  {  \_mensajeError = value.  OnPropertyChanged(  nameof(MensajeError));  }  }  public bool EsVistaVisible  {  get  {  \_esVistaVisible;  }    set  {  \_esVistaVisible = value;  OnPropertyChanged(  nameof(  EsVistaVisible));  }  }  ...  switch (mes)  {  case 12:  // Cae en Invierno  case 1:  // Cae en Invierno  case 2:  Console.WriteLine(“Invierno”);  break;  case 3:  // Cae en Primavera  case 4:  // Cae en Primavera  case 5:  Console.WriteLine(“Primavera”);  break;  case 6:  // Cae en Verano  case 7:  // Cae en Verano  case 8:  Console.WriteLine(“Verano”);  break;  case 9:  // Cae en Otoño  case 10:  // Cae en Otoño  case 11:  Console.WriteLine(“Otoño”);  break;  default:  Console.WriteLine(“Operación” +  “ incorrecta”);  }  ... |
| Identificadores | * Evitar acrónimos y abreviaturas (a menos que sean reconocidas). * Usar nombres descriptivos y con la menor ambigüedad posible.     Clases   * Deben ser sustantivos en singular (salvo que se requiera algo diferente). * Evitar el uso de verbos. * Seguir el estilo PascalCase.     Métodos   * Deben ser verbos. * Seguir el estilo PascalSnakeCase. * Los métodos escuchadores deben llevar un sufijo con el nombre del evento generado.     Variables   * El nombre debe indicar la función que cumple la variable. * Seguir el estilo camelCase. * Evitar el nombrado de un solo carácter, a menos que sean variables para índices temporales. * Si es un atributo con el modificador de acceso private inicia con un guion bajo o underscore.   Constantes   * Seguir el estilo UPPERCASE.   Propiedades   * El nombre debe llamarse igual al atributo que encapsulan. * Seguir el estilo PascalCase. |
| Formato de Identificadores | public partial class CalculadoraView : Window  {  public Calculadora View ()  {  InitializeComponent();  }    private void Button\_CerrarVentana\_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)  {  this.Close();  }  ...  private string? \_mensajeError = "";  private bool \_esVistaVisible = true;    public string MensajeError  {  get  {  \_mensajeError;  }    set  {  \_mensajeError = value.  OnPropertyChanged(  nameof(MensajeError));  }  }  public bool EsVistaVisible  {  get  {  \_esVistaVisible;  }      set  {  \_esVistaVisible = value;  OnPropertyChanged(  nameof(  EsVistaVisible));  }  }  ... |
| Lengua | Todo nombrado será en inglés, a menos que el propio estándar indique otra cosa  Excepto:   * Es un nombre propio * Si es parte de un patrón de diseño o arquitectura de software |
| Formato de Lengua | * ... private string \_password; ... * LogInView.xaml.cs * public void calculateRoot()... |
| Ficheros | XAML   * Seguir el estilo PascalCase. * Utilizar el sufijo “View” * Utilizan la extensión .xaml * Se coloca en el Namespace “Views”   Code Behind del XAML (C#)   * Seguir el estilo PascalCase. * Tiene de sufijo “View” * Tiene la extensión .cs * Se coloca en el Namespace “Views”   Controlador (C#)   * Seguir el estilo PascalCase. * Tiene la extensión .cs   POCO (C#)   * Seguir el estilo PascalCase * Debe llamarse a corde al modelo de datos del objeto. * Tiene la extensión .cs * Se coloca en el Namespace “POCO”   DTO (C#)   * Seguir el estilo PascalCase * Utilizar el sufijo “DTO” * Debe llamarse a corde al modelo de datos del objeto. * Tiene la extensión .cs * Se coloca en el Namespace “DTO”   Adapter (C#)   * Seguir el estilo PascalCase * Utilizar el sufijo “Adapter” * Debe llamarse a corde al modelo de datos del objeto. * Tiene la extensión .cs * Se coloca en el Namespace “Adapters”   Repository (C#)   * Seguir el estilo PascalCase * Debe llamarse a corde al modelo de datos del objeto. * Utilizar el sufijo “Repository” * Tiene la extensión .cs * Se coloca en el Namespace “Repositories”   Visual Component (C#)   * Seguir el estilo PascalCase * Utilizar el sufijo “Control” * Utilizan la extensión .xaml * Se coloca en el Namespace “VisualComponents” |
| Formato de Ficheros | XAML  LogInView.xaml  Code Behind del XAML (C#)  LogInView.cs  Controlador (C#)  LogInController.cs  POCO (C#)  Estudent.cs  DTO (C#)  EstudentDTO.cs  Adapter (C#)  SearchGameAdapter.cs  Repository (C#)  EstudentRepository.cs  Visual Component (C#)  ButtonControl.xaml |