**PROYECTO FINAL**

Sistema del Servicio Social de la FEI

**Equipo 5: Dulce Regina Saavedra Mata, Arturo Villa López,**

**Principios de Construcción**

## Estándar de Código

## Introducción

## Objetivo

En el siguiente documento se establecen algunas normas o reglas que ayudan a la construcción de software para brindar una mejor interpretación.

* Se asegura la calidad del producto de software.
* Facilita el mantenimiento de este.
* Simplifica la detección de errores y bugs.
* Facilitar la comprensión para el trabajo en conjunto.

Todos estos conceptos van alineados con el uso de la Metodología Orientada a Objetos en la plataforma Visual Studio.NET

## Alcance

Para este proyecto en específico desarrollador en el lenguaje C# en Visual Studio. Incluye los estándares de nomenclatura, diseño del código, uso de comentarios, así como unas buenas prácticas.

## Nomenclaturas

Existen varios estilos de estándares para definir los nombres e identificar los elementos del proyecto en C#, entre los cuales destacan 2:

**Notación de Pascal (Pascal Case):** El primer carácter de todas las palabras se escriben en Mayúscula y todos los demás caracteres en minúsculas. Por ejemplo: CodigoPrueba.

**Notación de camello (Camel Case):** El primer carácter de todas las palabras excepto de la primera se escriben con Mayúscula y todos los demás en minúsculas. Por ejemplo: codigoPrueba.

## Variables

* Los nombres de variables y de atributos de clase deberán de ser de más de un carácter.

Una excepción pueden ser los nombres de los iteradores en los ciclos.

**Ejemplo:**

For (int i = 0 ; i < Variable ; i ++ ) {

}

* Usar notación Camel Case.
* Distinguir los nombres de las variables booleanas.
* Las instancias usan la notación Camel Case
* Diferenciar entre variables globales y locales.
* No usar palabras reservadas en nombres de variables.
* Las variables booleanas de retorno se escriben con Camel case.

**Ejemplo:**

**Correcto: Incorrecto:**

string Nombre; string nom;

int NumeroDeAlumnos; int nA;

## Constantes

La definición de los constantes se efectúa siempre y cuando el valor no cambie durante el tiempo de ejecución del software, todo en mayúsculas.

**Ejemplo:**

public const int MESES = 12;

## Métodos

La definición del nombre de los métodos este guiado por las siguientes normas:

* Utilizar Pascal Case.
* Los metodos de prueba unitaria se inicializan con la palabra “Prueba” seguido del tipo de prueba.

Ejemplo : PruebaRegistrarAlumnoValido().

* Utilizar verbos o frases descriptivas sobre la acción que tendrá en el programa.
* Se usan prefijos para nombrar métodos y clases.
* No se deben usar nombres extraños o engañosos, el nombre debe ser obvio por lo cual no se necesita documentación para explicarlo.

Por Ejemplo: VaciarCubeta(), TirarPelota(), CerrarVentana().

## Clases

La definición del nombre de las clases este guiado por las siguientes reglas:

* Se debe usar un sustantivo en singular, con excepción que la clase represente múltiples cosas.
* Se debe utilizar Pascal Case.
* No se deben utilizar prefijos como C, T o clr.
* Las clases solo representas “cosas” y no “acciones”.

Ejemplo: Empleado, Proveedor, CuentaBancaria.

## Interfaces

Usa el prefijo **“I”** con notación **Pascal** para las interfaces.

//Ejemplo:

interface IClaseDAO{

void Metodo1(int variable);

int Metodo2();

}

## Interfaces Gráficas

La definición del nombre de las interfaces este guiado por las siguientes reglas:

* Se debe utilizar pascal case.
* La nomenclatura es “Actor que usa la pantalla” + “Elemento o Actor con el que interactúa”.

## Objetos y Listas de Objetos

La definición del nombre de las interfaces este guiado por las siguientes reglas:

* Se escriben con letra mayúscula al inicio.
* La nomenclatura es “Actor que usa la pantalla” + “Elemento o Actor con el que interactúa”.

## Diseño de código

## Estructura

Estas convenciones tienen la finalidad siguiente:

* Facilitar la lectura, modificación y mantenimiento del código.
* Dar un aspecto coherente al código, de tal manera que el lector centre su atención en el contenido y no en el diseño.
* Permitir la lectura y comprensión con mayor rapidez.
* Utilizar una sola instrucción por línea.
* Utilizar una sola declaración por línea.
* Si a las líneas no se les aplica una sangría automática, aplicarla con una tabulación (cuatro espacios).
* Utilizar un espacio en blanco para separar grupos lógicos de códigos.
* Utilizar paréntesis para cláusulas de una expresión, así se hace más evidente.

Ejemplo:

if ((numero1 < numero2) && (numero1 < numero3))

{

*//Ingresar código*

}

## Uso de Comentarios

Los comentarios deberán seguir las siguientes convenciones:

* No usar comentarios para cada línea y/o variable.
* Usar de preferencia // en lugar de /\* …...\*/
* Usar los menos comentarios posibles.
* Usarlos para explicar lógica compleja.
* Usarlos para explicar asignaciones a las variables.
* Tener buena ortografía, puntuación y gramática.

## Documentación de la API

C# utiliza XML para generar la API del proyecto.

* Los comentarios utilizados para la documentación de clases y métodos usan tres barras diagonales: /// y un cuerpo de comentario con formato XML. Por ejemplo:

/// <summary>

/// Esta clase tiene una funcion.

/// </summary>

public class claseRandom

{

}

La base para la documentación de una clase cualquiera en este proyecto es:

/// <summary>

/// Explicación de la clase.

/// </summary>

public class claseRandom

{

}

La base para la documentación de un método en este proyecto es:

/// <summary>

/// Explicación del metodo.

/// </summary>

/// <param name="nombre de parametro">Tipo de parametro.</param>

/// <returns>

/// Aclaración de valor que regresa.

/// </returns>

/// <exception cref="excepción">Condición para aparición de la excepción.</exception>

private void saludo(String mensaje)

{

}

## Etiquetas

Las etiquetas que se pueden realizar están redactadas en la web .NET de Microsoft.

*<summary>*

**Requerido**

Se utiliza usualmente para describir un objeto. Es la primera línea de las etiquetas de la documentación.

*<seealso>*

**Opcional**

Agrega enlaces asociados al método.

*<param>*

**Requerido**

Describe un parámetro en un método.

*<typeparam>*

**Requerido**

Describe un parámetro o método de tipo genérico.

*<returns>*

**Requerido**

Se utiliza para describir el valor a retornar por un método.

*<exception>*

**Opcional**

Describe una excepción que podría ser lanzada por un método.

*<permission>*

**Opcional**

Se utiliza para describir el nivel de acceso a un método.

*<include>*

**Opcional**

Utilizado para incluir en la documentación un archivo o parte de otro en el código.

*<para>*

**Opcional**

Utilizado para definir un bloque, solo se utiliza a nivel del elemento de documentación.

*<code>*

**Opcional**

Se utiliza para especificar el formato que tendrá el código.

*<see>*

**Opcional**

Se utiliza para definir un enlace a una página interna (cref) o externa (href).

*<paramref>*

**Opcional**

Hace referencia a un parámetro dentro de otra etiqueta

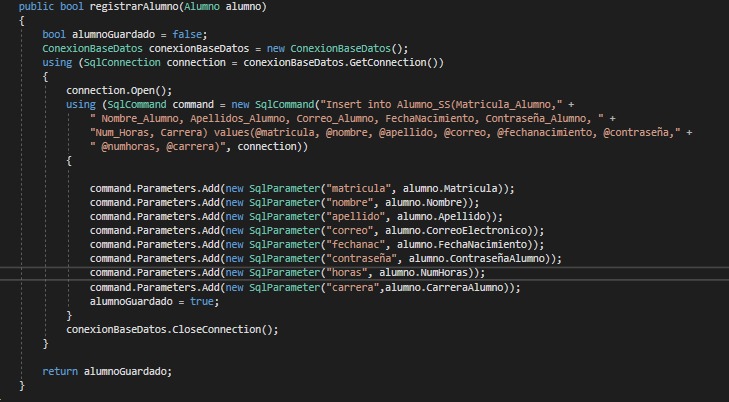
*<typeparamref>*

**Opcional**

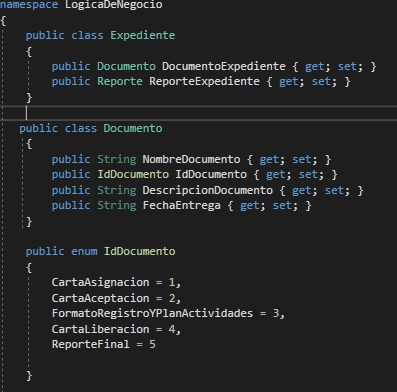
Hace referencia a un tipo de parámetro genérico

## Buenas prácticas

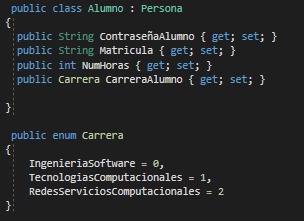
1. Evitar escribir métodos muy largos. Se recomienda que un método deba tener entre 1 a 25 líneas de código, si se llega a tener más de 25 líneas se deberá refactorizar , las excepciones no hacen que el método sea mas largo.



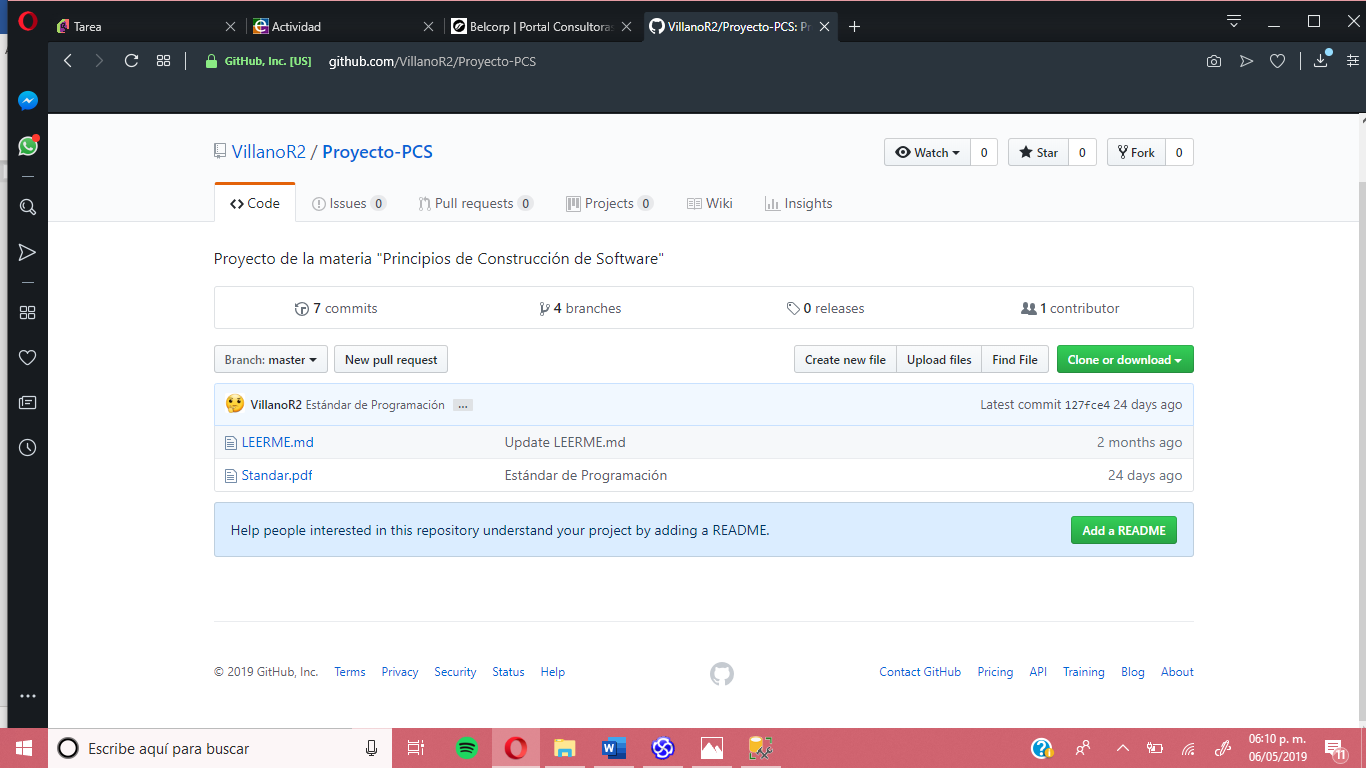
1. Aplicar las sangrías apropiadas a los segmentos de código que lo ameriten, por ejemplo, los ciclos, los selectivos y los condicionales.



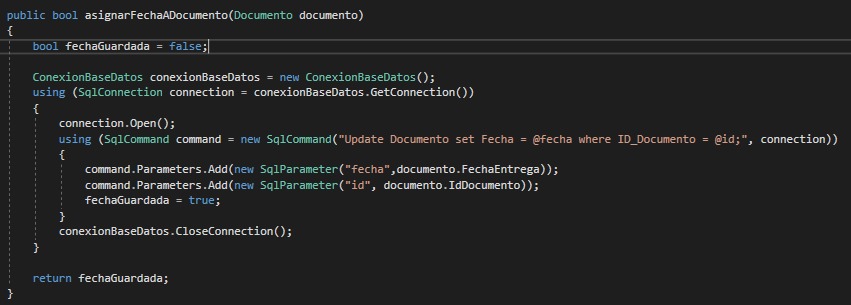
1. Respetar las nomenclaturas definidas para las variables, clases y métodos.



1. Compartir el estándar con todo el equipo.



1. Usar solo los comentarios estrictamente necesarios.  
   **No se han manejado comentarios hasta el momento.**
2. Dejar espacios entre líneas de código y secciones diferentes.



## Modelo Dominio (Visto Bueno Diseño)

## 

Modelo Modelo De Dominio

## Modelo Entidad Relación

## 

Modelo Modelo Entidad Relación

## Prototipos

## 

Pantalla Pantalla de Inicio de sesión

## 

Pantalla Pantalla principal del director

## 

Pantalla Pantalla para registro de coordinador

## 

Pantalla Pantalla principal director con ventana de “cerrar sesión”.

## 

Pantalla Pantalla registro de alumno

## 

Pantalla Pantalla alumno pendiente de validación

## 

Pantalla Pantalla principal alumno

## 

Pantalla Pantalla alumno documentos subidos

## 

Pantalla Pantalla alumno solicitar servicio social

## 

Pantalla Pantalla alumno principal

## 

Pantalla Pantalla principal coordinador

## 

Pantalla Pantalla Coordinador Alumno

## 

Pantalla Pantalla Registro de Técnico

## 

Pantalla Pantalla coordinador con ventana de cerrar sesión

## 

Pantalla Realizar calendarización

## 

Pantalla Principal Coordinador

## 

Pantalla Registrar institución

## 

Pantalla Registrar proyecto

## 

Pantalla Registrar encargado

## 

Pantalla Asignar Proyecto

## 

Pantalla Coordinador principal con cerrar sesión

## 

Pantalla Principal Técnico

## 

Pantalla Subir documento

## 

Pantalla Explorador de Archivos

## 

Pantalla Explorador de archivos

## Propuesta de Capas

## Nuestra propuesta para el sistema del Servicio Social se encuentra dividido en 3 capas:

## Acceso a Datos:

## En esta capa se establece la conexión con la base de datos Servicio Social.

## Lógica del Negocio

## En esta capa se encuentran codificadas todas las clases planteadas en el modelo de dominio así como sus relaciones.

## Se realizará la lectura y escritura en la base de datos.

## Se encuentran la carpeta acceso a datos: interfaces con la definición de los DAO, la definición de los mismos métodos; en la carpeta excepciones: las clases de manejo de excepciones personalizadas: la carpeta modelo con la definición de todas las clases y las clases enum por último en la carpeta Útil se encuentra una clase para validar usuarios y un clase convertidora de enums.

## Servicio Social App

## En esta capa está todo aquello que será visible para el usuario.

## Pantallas en las que podrá ingresar datos y visualizar.

## Documentacion de pruebas.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre de Prueba | Fallida | Existosa | Observaciones |
| PruebaRegistrarAlumnoValido |  | √ | Se registra correctamente un alumno en la base de datos, se espera no obtener error alguno, regresa una confirmacion en “true”. |
| PruebaRegistrarAlumnoValido2 |  | √ | Se registra correctamente un alumno con datos diferentes a la prubea anterior en la base de datos, se espera no obtener error alguno, regresa una confirmacion en “true”. |
| PruebaRegistrarAlumnoLlavePrimaria | X |  | Se intenta registrar un alumno con una matricula ya existente en la base de datos, se esepra una excepcion a la llave duplicada, regresa una confirmacion en “false”. |
| PruebaRegistrarAlumnoMatriculaNoValida | X |  | Se intenta registrar un alumno con una matricula que excede el limite de carateres en la base de datos, se espera una excepcion a matricula no valida, regresa una confirmacion en “false”. |
| PruebaRegistrarAlumnoLlavePrimariaVacia | X |  | Se intenta registrar un alumno sin ingresar ninguna matricula, se espera una excepcion a matricula vacia, regresa una confirmacion en “false”. |
|  |  |  |  |
| PruebaRegistrarCoordinadorValido |  | √ | Se registra correctamente un coordinador en la base de datos, se espera no obtener error alguno, regresa una confirmacion en “true” |
| PruebaRegistrarCoordinadorMatriculaErronea | X |  | Se intenta registrar un coordiandor con una matricula que excede el limite de carateres en la base de datos, se espera una excepcion a matricula no valida, regresa una confirmacion en “false”. |
| PruebaRegistrarCoordinadorMatriculaRepetida | X |  | Se intenta registrar un coordinador con una matricula ya existente en la base de datos, se espera una excepcion a la llave duplicada, regresa una confirmacion en “false”. |
| PruebaRegistrarCoordinadorVacio | X |  | Se intenta registrar un coordinador sin ingresar ningun dato, se espera una excepcion a campos vacios, regresa una confirmacion en “false”. |
|  |  |  |  |
| RegistrarEncargadoValido |  | √ | Se registra correctamente un encargado en la base de datos, se espera no obtener error alguno, regresa una confirmacion en “true”. |
| RegistrarEncargadoRepetido | X |  | Se intenta registrar un encargado con una matricula ya existente en la base de datos, se espera una excepcion a la llave duplicada, regresa una confirmacion en “false”. |
| RegistrarEncargadoCamposVacios | X |  | Se intenta registrar un encargado sin ingresar ningun dato, se espera una excepcion a campos vacios, regresa una confirmacion en “false”. |
|  |  |  |  |
| PruebaRegistrarInstitucionValida |  | √ | Se registra correctamente una institucion en la base de datos, se espera no obtener error alguno, regresa una confirmacion en “true”. |
| PruebaRegistrarInstitucionCamposVacios | X |  | Se intenta registrar una institución sin ingresar ningun dato, se espera una excepcion a campos vacios, resa una confirmacion en “false”. |
| PruebaRegistrarInstitucionRepetida | X |  | Se intenta registrar una institucion ya existente en la base de datos, se espera una excepcion a la llave duplicada (El nombre de la institución funciona como llave unica), regresa una confirmacion en “false”. |
|  |  |  |  |
| PruebaRegistrarTecnicoValido |  | √ | Se registra correctamente un tecnico académico en la base de datos, se espera no obtener error alguno, regresa una confirmacion en “true”. |
| PruebaRegistrarTecnicoNoValido | X |  | Se intenta registrar una tecnico académico sin ingresar ningun dato, se espera una excepcion a campos vacios, regresa una confirmacion en “false”. |
| PruebaRegistrarTecnicoLlaveRepetida | X |  | Se intenta registrar un tecnico académico con una matricula ya existente en la base de datos, se espera una excepcion a la llave duplicada, regresa una confirmacion en “false”. |
| PruebaRegistrarTecnicoCoordinadorNoExiste | X |  | Se intenta registrar un tecnico académico con una matricula de un coordinador no existente en la base de datos, se espera una excepcion de coordinador no existente, regresa una confirmacion en “false”. |
|  |  |  |  |

## Conclusiones

## Arturo:

## Gracias a el uso de pruebas unitarias se han detectado errores que al momento de compilar y ejecutar el software se pasaban por alto, añadiéndole un manejo de excepciones en diversos casos, se han solucionado algunos problemas que generaba al probar el código, también llegue a la conclusión que algunos fragmentos de código factibles al cambio y puede generar algunos problemas si no se maneja de manera unitaria los fragmentos de código que no dependen entre sí.

## Regina:

## Con el proyecto al día de hoy, se puede observar lo importante que es tener los modelos hechos en diseño, totalmente definidos ya que son la base para tomar muchas de las decisiones en el código y permite saber que métodos y atributos se encuentran en cada clase y como se encuentran relacionadas entre sí, lo que hace que la codificación sea una actividad casi de traducción de estos métodos. También noté que los prototipos son un gran apoyo ya que permite ver las ideas un poco más aterrizadas y poder tener un acercamiento al proyecto final, y llegar a esa meta realizando el código.

## Referencias

El Estándar de Código está basado en el siguiente estándar de C#, escrito por José Luis Clemente Montes. <https://es.calameo.com/read/00443793525640a944420>

Uso de excepciones

https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/>

<https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/exceptions/using-exceptions>

<https://albertcapdevila.net/control-excepciones-csharp/>

Uso de SQL en Visual Studio

<https://social.msdn.microsoft.com/Forums/es-ES/ad3cfe1e-7ec3-476f-9e0e-91a89f17cd02/hacer-una-consulta-sql-con-c-utilizando-un-data-reader-y-mostrarlo-por-consola?forum=vsexes>

<https://www.lawebdelprogramador.com/foros/C-sharp/1579952-Consultas-sql-en-C.html>

<https://solocodigo.com/30600/consulta-sql-desde-c/>