# UNIVERSIDAD VERACRUZANA

## FACULTAD DE ESTADISTICA E INFORMATICA

# ESTÁNDAR DE CODIFICACIÓN


**RICARDO MOGUEL SÁNCHEZ**

**S18012183**

**Cesar Sergio Martínez Palacios**

**S18012143**


**DOCENTE**

**MTRO. SAMUEL BÁEZ HERRERA**

**MTRO. JUAN LUIS LOPEZ HERRERA**

**EXPERIENCIA EDUCATIVA**

**DESARROLLO DE SISTEMAS EN RED**

**DESARROLLO DE APLICACIONES**

**XALAPA ENRÍQUEZ, VERACRUZ 19 DE ABRIL DE 2021**

# Tabla de contenido


TABLA DE CONTENIDO 1

1. INTRODUCCIÓN 2

2. CONVENCIÓN DE NOMBRES 2

3. FORMATOS O ESTILO 2

**3.1. Indentación 2**

**3.1.1. Espacios en blanco 2**
**2.1.2 Ajuste de línea** 2

**2.2 Comentarios 4**
**2.2.1 Comentarios de bloque** 4
**2.2.2 Comentarios de línea** 5

4. PATRONES DE ESCRITURA DE CÓDIGO 5

**3.1 Declaraciones 5**
**3.1.1 Ubicación** 5
**3.1.2 Inicialización** 6
**3.1.3 Declaraciones de variables locales** 6
**3.1.4 Declaración de clases e interfaces** 6
**3.1.5 Declaración de *return*** 8
**3.1.6 Declaraciones de if, if-else, if-else-if-else** 8
**3.1.7 Declaraciones de un ciclo *for*** 9
**3.1.8 Declaración de un ciclo *while*** 10
**3.1.9 Declaración de un ciclo *do-while*** 11
**3.1.10 Declaración de un switch** 11

**3.2 Espacio en blanco 12**
**3.2.1 Líneas en blanco** 12
**3.2.2 Espacios en blanco** 14

**3.3 Acceso a las variables de clases e instancias 15**

**3.4 Referencia a los métodos y variables de clase 15**

**3.5 Asignaciones de variables 16**

**3.6 Uso de paréntesis 17**

4. MANEJO DE EXCEPCIONES 17

# 1. Introducción

Este estándar de codificación fue usado anteriormente para el proyecto de la EE Principios de Construcción. Está dirigido a el lenguaje de C# y se basa en la guía de codificación oficial de este mismo creada por Microsoft. Se realizaron ajustes para acomodar a nuestras necesidades con el proyecto actual de un servidor con un cliente web y un cliente de escritorio para poder seguir manejando buenas prácticas de codificación.

# 2. Convención de nombres

Los nombres para las clases, paquetes, métodos, variables y constantes deben tener un nombre descriptivo con respecto al propósito de estas. Además, dicho nombre no debe de superar más de cinco palabras. Evitar uso de adverbios y artículos. Los nombres deben seguir la siguientes notaciones:

Clases Interfaces y Métodos en notación Pascal Case

Variables en notación Camel Case

Constantes en notación de Mayúsculas

Por ejemplo:

Si se desea subir una canción, los nombres **aceptables** son:

- Clase o Interfaces: *Song*
- Método: *UploadSong*
- Variable: songTitle
- Constante: FILE_*SIZE*

Los nombres **NO** aceptados son:

- Clase o Interfaces: *MP3File*
- Método: *Upload, UploadItem*
- Variable: *xName, x, xN*
- Constante: constant, Const

# 3. Formatos o estilo

## 3.1. Indentación

### 3.1.1. Espacios en blanco

Utilizar la tecla “Tab” para identar el código en lugar de hacerlo utilizando espacios.

### 3.1.2 Ajuste de línea

Cuando una expresión sobrepase el margen del editor se debe hacer un salto de línea cumpliendo una de las siguientes condiciones:

1. Salto de línea después del uso de una coma.
2. Salto de línea después del uso de operadores aritméticos.

Ejemplos:

```
/* CASO 1 */

//Correcto

calcularAreaTrapecio (altura, baseMenor,
baseMayor);


//Incorrecto


/* CASO 2 */

//Correcto

float perimetroTrapecio = baseMenor + baseMayor +
ladoIzquierdo + ladoDerecho;


//Incorrecto

float perimetroTrapecio = baseMenor + baseMayor

+ ladoIzquierdo + ladoDerecho;
```

## 3.2. Comentarios

### 2.2.1 Comentarios de bloque

Los comentarios de bloque son utilizados únicamente al inicio de una clase para describir el contexto de la clase.

Estos comentarios deben de tener como mínimo: El o los autores de la clase, fecha de creación de la clase y una descripción. El comentario de bloque debe iniciar con "/*" y terminar con "*/", dentro de él cada línea comienza con un "Tab".

Comentario de bloque **correcto**:

```
/*
     Authors: Pedro Juarez y Abizair Martinez
     Date: 20 de enero de 2018
     Description: Clase utilizada para manejar cálculos de
*/
```

Comentario de bloque **NO** correcto:

```
/*Autores: Pedro Juarez y Abizair Martinez
Fecha: 20 de enero de 2018
Descripción: Clase utilizada para manejar cálculos de figuras
geométricas*/
```

### 2.2.2 Comentarios de línea

No serán utilizados los comentarios de línea para describir una línea de código ya que las variables o métodos deben ser suficientemente descriptivas.

# 4. Patrones de escritura de código

## 4.1 Declaraciones

### 4.1.1 Ubicación

Ponga declaraciones solo al comienzo de cada bloque. (Un bloque es cualquier código rodeado de llaves "{" y "}"). No espere para declarar variables hasta su primer uso.

Ejemplo:

```
void myFunction(){
     //inicio del bloque del método
     int variable1;


     if(condition){
          //inicio del bloque de if
          int variable2;
```

La única excepción es para el bucle *for*, que contiene una declaración dentro del mismo bucle declarado como:

```
for(int i = 0; i < 3; i++)
```

### 4.1.2 Inicialización de variables

Intente inicializar las variables locales donde se declaran. La única razón para no inicializar una variable donde se declara es si el valor inicial depende de algún cálculo que ocurra primero.

### 4.1.3 Declaraciones de variables locales

Se debe de hacer una declaración por línea sin importar que sean del mismo tipo.

Como mencionado anteriormente, se debe seguir el estilo camelCase: Esta notación define que la primera letra de cada palabra a excepción del primero debe ser mayúscula.

Ejemplo correcto para declarar variables:

```
int globalRank;
string userNickname;
```

Ejemplo **NO** correcto para declarar variables:

```
int Rank,string  USERNickName;
```

#### 4.1.4 Declaración de clases e interfaces

Cuando se codifican clases o interfaces en C#, se debe seguir las siguientes reglas:

1. No incluir espacios entre el nombre de un método y los paréntesis donde se encuentran los parámetros del método.
2. El nombramiento debe seguir la notación Pascal Case. Esta nomenclatura o estilo define que la primera letra de cada palabra debe ser mayúscula. Ejemplo: *GameBoard*.
3. En las clases, la llave de apertura debe aparecer en la siguiente línea de la declaración. Ejemplo: public void MoveChecker()
   {
4. En las clases, la llave de cierre debe comenzar en una línea nueva, alineada verticalmente con la llave de inicio, a excepción cuando es una declaración nula, la llave de inicio es declarada junto al método y la llave de cierro inmediatamente después de la llave de inicio.
   Ejemplo: public void ClicKChecker(){}
5. Los métodos son separados por una línea en blanco.

Ejen

```
/* DECLARACIÓN CORRECTA DE UNA CLASE */

public class MyClass : Object

{

    int variable1;
     int variable2;


     public MyClass(int i, int j)
     {
          variable1 = i;
          variable2 =j;
     }
     public int doSomething()
     {
           //code
     }
     ...
}


/* DECLARACIÓN INCORRECTA DE UNA CLASE */
public class myClass : Object{
     int variable1;
     int variable2;


     public myClass(int i, int j){
           variable1 = i;
           variable2 = j;
     }
     int doSomething()
     {}
}
```

### *4.1.5* Declaración de *return*

Una declaración de retorno con un valor no debe usar paréntesis a menos que hagan que el valor de retorno sea más obvio de alguna manera. Ejemplo:

```
return;

return vector1.size();

return (size ? size : defaultSize);
```

### 4.1.6 Declaraciones de if, if-else, if-else-if-else

La declaración de un if debe tener la siguiente forma:

```
if(condition)
{
     statements;
}


//Incorrecto
if condition){
     Statements;
}
```

La declaración de un if-else debe tener la siguiente forma:

```
//Correcto
if(condition)
{
      statements;
}
else
{
      statements;
}
//Incorrecto
if(condition){
      statements;
}else {
      statements;
}
```

La declaración de un if-else-if-else de tener la siguiente forma:

```
if(condition)
{
     statements;
}
else if(condition)
{
    statements;
}
else if(condition)
{
    statements;
}


//Incorrecto
if (condition){
     statements;
}
else if (condition){
     statements;
}
else if (condition){
     statements;
}
```

Las declaraciones de un if siempre deben de llevar llaves sin importar que solo cuente con una línea de código.

### *4.1.7* Declaraciones de un ciclo *for*

Un ciclo *for* debe tener la siguiente forma:

```
for(initialization; condition; update)
{
    statements;
}

//Incorrecto
for(initialization; condition; update){
     statements;
}
```

Cuando use el operador de coma en la cláusula de inicialización o actualización de una instrucción *for*, evite la complejidad de usar más de cuatro variables. Si es necesario, use sentencias separadas antes del ciclo *for* (para la cláusula de inicialización) o al final del ciclo (para la cláusula de actualización).

```
//Incorrecto
for(int i = 2 – index1 + index2 + index3 - 1; i <= index3; i++){
     //code
}

//Correcto
int startPoint = 2 – index1 + index2 + index3 – 1;
```

### *4.1.8* Declaración de un ciclo *while*

Un ciclo *while* debe tener la siguiente forma:

```
 while(condition)
{
   statements;
}
//Incorrecto
while condition){
     statements;
}
```

### *4.1.9* Declaración de un ciclo *do-while*

Un ciclo *do-while* debe tener la siguiente forma:

```
do
{
    statements;
}
while(condition);


//Incorrecto
do{
     statements;
} while (condition);
```

#### 4.1.10 Declaración de un switch

Un *switch* debe tener la siguiente forma:

```
switch(condition)
{
     case ABC:
          statements;
          case DEF:
          statements;
          break;
     case XYZ:
          statements;
          break;
     default:
          statements;
          break;
}


//Incorrecto
switch condition){
     case ABC:
     statements;
     break;
     case DEF:
}
```

Cada *switch* debe incluir un *default*. El *break* en el *default* es redundante, pero evita un error de caída si luego se agrega otro caso.

## 4.2 Bloques con espacios en blanco

### 4.2.1 Líneas en blanco

Las líneas en blanco mejoran la legibilidad al activar secciones de código que están relacionadas lógicamente.

Siempre deben usarse dos líneas en blanco en las siguientes circunstancias:

a. Entre secciones de un archivo fuente.
b. Entre las definiciones de clase e interfaces.

```
//Correcto
public class Circle
{
      //Code
}

public class Triangle
{
      //code
}
```

```
//Incorrecto
public class Circle{
      //Code
}
public class Triangle{
      //code
}
```

Siempre se debe usar una línea en blanco en las siguientes circunstancias:

c. Entre métodos.
d. Antes de un comentario de una línea.
e. Entre secciones lógicas dentro de un método para mejorar la legibilidad.

```
//Correcto
public class Circle

{

private float radius;

      public Circle(float radius)
      {
            //code
      }


      public float calculateCircleArea()
      {
      if(radius < 1)
      {
                  //code
      }
}
```

```
//Incorrecto
public class Circle{

      private float radius;

      public Circle(float radius){
            //code
      }
      public float calculateCircleArea(){
            //this is a line comment
            if(radius < 1){
                  //code
            }
      }
}
```

### 4.2.2 Espacios en blanco

Tenga en cuenta que **NO** se debe utilizar un espacio en blanco entre el nombre de un método y su paréntesis de apertura. Al igual que las declaraciones de ciclos y de los *if*.

Los espacios en blanco deben utilizarse en las siguientes situaciones:

a. Debe aparecer un espacio en blanco después de las comas en las listas de argumentos.
b. Todos los operadores binarios excepto "." deben estar separados de sus operandos por espacios. Los espacios en blanco nunca deben separar los operadores unarios, como el unario menos, incremento ("++") y decremento ("--") de sus operandos. Ejemplo:

```
//Correcto
public int doMathStuff(int number1, int
     number2)
{
     int response;
     response = (number1 + number2) / 2;
}

myObject.getName(
); while(isOpen)

{
     n++;
}
//Incorrecto
public int doMathStuff(int number1,int
     number2){ int response;
     response = (number1+number2) / 2;
}myObject. getName();
while(isOpen) {
     n + +;
```

c. Las expresiones en una declaración *for* deben estar separadas por espacios en blanco. Ejemplo:

```
for(int i = 0; i < 5; i ++)
{
     //code
}
//Incorrecto
for(int i = 0;i < 5;i ++){
     //code
}
```

d. Los Casts **NO** deben de ir seguidos de un espacio en blanco. Ejemplos:

```
public void myMethod((byte)aNum, (Object)x)
{
     //code
}
//Incorrecto
public void myMethod((byte) aNum, (Object) x){
     //code
}
```

### 4.3 Acceso a las variables de clases e instancias

No hacer uso de *setters* y *getters* cuando las variables pueden ser calculadas por un método.

### 4.4 Referencia a los métodos y variables de clase

Evitar el uso de un objeto para acceder a una clase estática, variable o método. En su lugar, usa el nombre de la clase. Ejemplo:

```
classMethod(); //Correcto
Circle.calculateArea(); //correcto


Circle circle1 = new Circle();
circle1.calculateArea (); //Incorrecto
```

### 4.5 Asignaciones de variables

Evita asignar a múltiples variables el mismo valor en una sola línea. Ejemplo:

```
//Correcto
int variable1 = 5;
int variable2 = variable1;
//Incorrecto
int variable1 = variable2 = 5;
```

No usar el operador de asignación (=) en un lugar donde puede ser fácilmente confundido con un operador de comparación. Ejemplo:

```
//Incorrecto
if(variable1 = variable2){
     …
}
```

Debe ser escrito así:

```
if(variable1 == variable2)
{
      …
}
```

No usar asignaciones anidadas. Ejemplo:

```
total = (result1 = variable1 + variable2) + result2


//Correcto
result1 = variable1 + variable2;
total = result1 + resul2;
```

### 4.6 Uso de paréntesis

Usar paréntesis en expresiones que contengan operadores mezclados para evitar los problemas de precedencia.

Ejemplo incorrecto:

```
if(a == b && c == d){
      …
}
```

Ejemplo correcto:

```
if((a == b) && (c == d))
{
      …
}
```

## 5. Manejo de excepciones

Se usa un bloque *try* para particionar el código que podría verse afectado por una excepción. Los bloques *catch* asociados se usan para manejar cualquier excepción resultante. Un bloque *finally* contiene código que se ejecuta independientemente de si se lanza o no una excepción en el bloque *try*, como la liberación de recursos que se asignan en el bloque try. Un bloque try requiere uno o más bloques catch asociados, o finalmente un bloque, o ambos.

Un bloque *catch* puede especificar el tipo de excepción para capturar. El tipo de excepción debe derivarse de la clase *Exception*. Es importante que se especifique cuál es la excepción que se va a manejar y no dejarla en general (*Exception*).

Ejemplo de un bloque de *try-catch-finally*.

```
Try
{
     …
}
catch(Exception e)
{
     …
}
finally
{
     …
}
```

Por ejemplo, si se desea abrir una conexión a la base de datos con la clase SqlCliente, debemos especificar el tipo de excepción que puede ocurrir al hacer esa operación. Por ejemplo:

```
public void dataBaseConnection(string info)
{
     using(SqlConnection connection = new
          SqlConnection((info))
     {
          try
          {
               connection.Open();
          }
          catch(SqlException exc)
          {
               //code
          }
          Finally
          {
//Incorrecto
public void dataBaseConnection(string info){
     using(SqlConnection connection = new
          SqlConnection((info)){ try{
               connection.Open();
          }catch(Exception exc){
               //code
          }finally{
               connection.Close();
          }
     }
```

# 6. 6.0. Documentación técnica

Los comentarios de documentación XML son un tipo especial de comentarios que se agregan encima de la definición de un tipo o un miembro definido por el usuario. Son especiales porque los puede procesar el compilador para generar un archivo de documentación XML en tiempo de compilación (Microsoft, 2020).
En este proyecto vamos a hacer uso principalmente de las siguientes etiquetas

- <summary>
- <remarks>
- <returns>
- <c>
- <exception>
- <see>
- <param>

## 6.1 <Summary>

La etiqueta `<summary>` agrega información breve sobre un tipo o miembro. Sera usada únicamente en el proyecto GUI WPF para describir la funcionalidad de las ventanas. A continuación, se muestra un ejemplo de cómo hacerlo:

```
 /// <summary>
 ///  Descripción de la funcionalidad de la clase
 /// </summary>
```

## 6.2 <remarks>

La etiqueta `<remarks>` complementa la información sobre los tipos o los miembros que proporciona la etiqueta `<summary>`. Esta etiqueta debe ir siempre como continuación a una etiqueta `<summary>` para ofrecer un contexto más técnico sobre la clase en donde se encuentra. A continuación, se muestra un ejemplo de cómo hacerlo:

```
/// <summary>
///  Descripción de la funcionalidad de la clase
/// </summary>

/// <remarks>
/// Descripción sobre tipos de datos, patrones y
algoritmos                                    usados
/// </remarks>
```

## 6.3 <return>

La etiqueta <returns> describe el valor -*devuelto de una declaración de método. Esta etiqueta debe ser usada antes de la firma de un método, en esta se especificará el tipo de dato devuelto por el método y en caso de ser necesario una observación por parte del programador. A continuación, se muestra un ejemplo de cómo hacerlo:

```
/// <returns>
/// Se espera un dato de tipo TipoDato observación opcional
/// </returns>
```

### 6.4 <c>

Se puede usar <c> para marcar una parte del texto como código, es útil si quiere mostrar un ejemplo de código rápido como parte del contenido de la etiqueta. A continuación, se muestra un ejemplo de cómo hacerlo:

```
///<summary>
///La siguiente clase hace uso de <c>Clase</c> para realizar operaciones
///</summary>
```

### 6.5 <exception>

Esta etiqueta proporciona una manera de documentar las excepciones que un método puede iniciar. Usado para indicar el surgimiento de una excepción que puede generarse durante el uso de la clase. Cuando se desea documentar una excepción se coloca la etiqueta seguida por el nombre de la excepción. A continuación, se muestra un ejemplo de cómo hacerlo:

```
/// <exception>
///  Este método puede retornar una excepción tipo  MySqlException
/// </exception>
```

### 6.6 <see>

Esta etiqueta permite especificar un vínculo dentro del texto.
Cuando es necesario hacer referencia a otro elemento dentro de la misma clase. Se coloca la etiqueta seguida por el vínculo a la sección referenciada. A continuación, se muestra un ejemplo de cómo hacerlo:

```
/// Consulta <see cref=”referencia"/> para más información.
```

### 6.7 <param>

Esta etiqueta se usa para describir un parámetro dentro de un método, antes de colocar el nombre del parámetro se pone la etiqueta. A continuación, se muestra un ejemplo de cómo hacerlo:

```
/// <param>
///este metodo usa el parametro project tipo Project
///</param>
```

## 7. Referencias


Diez, L. (2003). *Reglas de codificación en C#.* Madrid: Danysoft Internacional. Obtenido de https://www.danysoft.com/estaticos/free/csharp2.pdf

Microsoft. (21 de enero de 2020). *Microsoft*. Obtenido de https://docs.microsoft.com/es-es/dotnet/csharp/codedoc