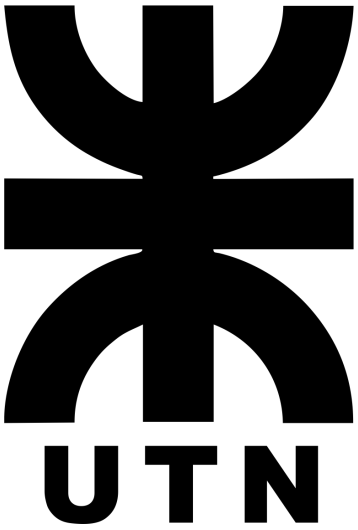
UNIVERSIDAD TECNOLÓGICA NACIONAL



REGIONAL CÓRDOBA

Ingeniería y Calidad de Software

4K3

Trabajo Práctico Nº 6

Buenas Prácticas - Reglas de Estilo del Código

PROFESORES:

- Covaro, Laura
- Massano, Cecilia
- Gonzalez, Georgina

ALUMNOS:

- Bergesio, Genaro
- Cuello, Marcelo
- Galiano, Tomas
- Nayi, Carlitos
- Floreano, Micaela
- Liendo, Agustin

Índice

Propósito

Las buenas prácticas en el código o reglas de estilo de código nos permite tener un software consistente, legible, entendible y mantenible en el tiempo. Es necesario para esto establecer una serie de reglas que los desarrolladores deben seguir para que garanticemos la calidad del software. En este documento pasaremos a enumerar diferentes aspectos que tenemos que considerar para que nuestro producto sea homogéneo cuando múltiples desarrolladores participan en el mismo.

Estas reglas se crean en base a el lenguaje C# y su utilización en el framework Windows Presentation Foundation (WPF) para el Trabajo Práctico Nº 6 de la materia Ingeniería y Calidad de Software.

Reglas

Convención de Nombrado C#:

Clases, Interfaces, Espacio de nombres, Métodos, Atributos o constantes:

Para estos casos utilizaremos PascalCase que corresponde a utilizar mayúscula al principio de cada palabra utilizada sin espacios y guiones. Los nombres deben ser significativos para el programador y entenderse a que está haciendo referencia.

Ejemplo:

- MiClaseRecienCreada (Bien)
- mCRC_ (Mal)

En caso que la palabra supere las 14 letras la palabra será abreviada y de ser necesario aclarado con un comentario:
Ejemplo:

- Organizaciones→ Orgs

En código:

```
using System.IO

namespace AplicacionEjemplo
{
   public static int NumeroConstante = 0;
   public class LaClaseCreada
   {
      // Lo ponemos público cuando no necesitemos restringir su acceso, ya que C# brinda esta
      // forma para facilitar el código.
      public int Id {get; set;}

      //Por otro lado si queremos restringir el Set o Get lo ponemos como "private"
      private string NombreClase;

      public string GetNombreClase() => this.NombreClase;

      public void MetodoConvencion()
      {
         // Las dos barritas representan un comentario en C#
      }
   }
}
```

Por otro lado las interfaces no solo es con PascalCase sino que además debemos agregar la letra "I" primero para diferenciarla de las clases.

```
public interface IClaseGenerica
{
      // Las dos barritas representan un comentario en C#
}
```

Variables y Parámetros:

En el caso que estemos hablando de variables locales de un método utilizaremos camelCase siendo la primera letra minúscula y usar mayúscula al inicio de una nueva palabra.

Ejemplo:

- miAtributo

En caso que la palabra supere las 10 letras será abreviada con las primeras letras y de ser necesario aclarado con un comentario:
Ejemplo:

```
public class ClaseGenerica
{
        public int MetodoGenerico(int miParametro)
        {
                // Las dos barritas representan un comentario en C#
                // Onomatopeya -> ono
                int ono = 5;
                return (ono + miParametro);
        }
}
```

Estructura y Comentarios.

Ubicación de atributos y métodos:

La estructura general de una clase es la siguiente:

1. Imports
2. Nombre del Espacio
3. Constantes
4. Clases-Interfaces
    a. Atributos
    b. Constructor
    c. Métodos

Comentarios y etiquetas:

En lo posible si un código parece complejo o tomó tiempo desarrollarlo se debe comentar que está haciendo dentro del mismo con "//". Por otro lado todos los clases,atributos y métodos deben llevar el comentario tipo XML de:

```
/// <summary>
/// Esta propiedad retorna el valor de 1.
/// </summary>
```

Convenciones en el código fuente

- **Corchetes:** Siempre usar corchetes, incluso en bloques de una sola línea, para mejorar la legibilidad y evitar errores.
- **Indentación:** Utilizar una indentación consistente (generalmente 4 espacios) para resaltar la estructura del código y facilitar su comprensión.

- **Ifs:** Evitar anidar demasiados `if`s, ya que dificulta la lectura. Considerar usar `else if` o variables booleanas para simplificar la lógica.
- **Operadores ternarios:** Usarlos para expresiones condicionales simples, pero evitar anidarlos en expresiones complejas, ya que pueden disminuir la legibilidad.

Convención de estructura de archivos

- Cada elemento que constituye un componente (.cs, .Designer.cs, .resx) tendrá el mismo nombre (Pantalla.cs, Pantalla.resx), y se mantendrán bajo la misma carpeta, lo que vuelve más legible y entendible la estructura de archivos. <Inserte fotitos comparando el antes y después.>

Definición de Responsabilidad

- Regla del 1: Definir un componente o servicio por archivo y limitar las líneas de código a 400.
    - Un componente por archivo lo vuelve más legible, mantenible y evita colisiones en el equipo al controlar el código fuente.
    - Un componente por archivo evita bugs escondidos que emergen cuando combinamos componentes que comparten variables en un archivo, creando cierres, acoplamiento y dependencias no deseadas.
    - Es una mejor práctica redistribuir el componente y sus clases de soporte en sus propios archivos dedicados.

Definir funciones pequeñas

- Las funciones pequeñas son más fáciles de testear, especialmente cuando hacen una sola cosa (cumpliendo SRP) y sirven un solo propósito.
- Las funciones pequeñas promueven la reutilización, son más fáciles de leer y de mantener.
- Las funciones pequeñas ayudan a evitar bugs escondidos que traen las funciones grandes que comparten variables con el entorno, creando cierres, acoplamiento y dependencias no deseadas.