Anexo 6

Estándares de Programación

1. Introducción

1.1. Objetivo

El objetivo de este documento es especificar los estándares que se han utilizado en la elaboración del sistema con la finalidad de garantizar una fácil lectura y comprensión de los archivos fuente por parte de personas diferentes a los programadores originales, del mismo modo facilita en el futuro mantenimiento que se le pueda dar al proyecto y la integración con librerías de terceros.

1.2. Alcance

Este documento provee orientación sobre el formato, comentarios, nombres de variables, y el estilo de programación de código fuente en C# y es aplicable a las librerías de componentes y aplicaciones de clientes.

2. Reglas básicas de programación

Las siguientes pautas son aplicables a todos los aspectos del desarrollo:

- Se sugiere hacer el código lo más legible posible. Tener en consideración que alguien más leerá el código.
- Optar por métodos cortos en vez de largos y complejos. En otras palabras, trate de dividir una tarea larga en pequeñas tareas.
- Escriba comentarios primero. Cuando escriba un método nuevo, escriba los comentarios para cada paso que el método realizará antes de codificar las instrucciones respectivas. Estos comentarios se convertirán en las cabeceras para cada bloque de código que sea implementado.
- Use comentarios que tengan un significado dentro de cada clase, método y bloque de código para documentar el propósito del código.
- Declara las variables una por línea, utilizar nombres significativos y darle un valor inicial.

3. Archivos

3.1. Nombres

Para los nombres se establecen las siguientes convenciones:

- Los archivos creados en C# tienen extensión .cs.
- Los nombres de los archivos deben ser claros, simples y descriptivos. Ej. UsuarioBean.cs

3.2. Organización

En la organización de los archivos hay que tener en cuenta los siguientes aspectos, estos son:

- Comentarios de Inicio: Suministran información sobre el nombre de la clase, la versión, la fecha y el nombre del autor. Por ejemplo:

  ```
  /*
  * Clase Central
  */
  ```

- Sentencias namespace y using: Son las primeras líneas no-comentarios en una clase. La primera de ellas es using y después de esta se define el nombre de namespace. Por ejemplo:

  ```
  using Cafeteria.Models.Almacen.Ingrediente;
  using Cafeteria.Models.Almacen;
  using Cafeteria.Models.Almacen.Notaentrada;

  namespace Cafeteria.Controllers.Compras{}
  ```

- Declaraciones de clase e interfaces: Se establece las partes y el orden en el que estas deben ser descritas, tal y como se muestra a continuación:

  - Comentario de Documentación.
  - Sentencia class o interface.
  - Comentario de implementación.
  - Variables de clase.
  - Variables de instancia.
  - Constructores.
  - Métodos

4. Indentación

Supone mover un bloque de texto a la derecha utilizando tabulaciones. Se usa con el fin de mejorar la legibilidad del código fuente por parte de los programadores.

Una unidad de indentacion es un carácter de tabulación. Ejemplo:

```
Public String limpiarCadena(String s)
{
        int i;
        for(i=s.length();i>0;i--)
        {
                if(s.charAt(i-1)!='*')
                {
                break;
                }
        }
```

```
        return s.substring(0,i);
}
```

4.1. Longitud de Líneas

Se sugiere que una línea de código no exceda 70 carácteres. Ejemplo:

```
Private void escribirLinea(RandomAccessFile archivo, String str){
```
(66 caracteres)

4.2. Saltos de Líneas

Los saltos de línea ocurren cuando, para una expresión, no es suficiente una sola línea, debido a su longitud, con lo cual se continúa en la línea siguiente, teniendo en cuenta los criterios que se muestran a continuación:

- A Continuación de una coma:

```
enviarAlerta(getCodigo(),getNombre(), getApellido(),
                    getDireccion(), getTelefono() );
```

```
alerta = datosAlerta(getCodigoVuelo,
                        datosIncidencia(getCodIncidencia,
                                getNombre)
```

- Antes de un operador:

```
Total= cantidad * (valor + recargo          (BIEN)
        – descuento) + 100;
```

```
Total= cantidad * (valor + recargo          (MAL)
                    – descuento) + 100;
```

- Alinear la línea siguiente con el inicio de expresiones al mismo nivel de lo anterior:

```
enviarAlerta(getCodigo(), getNombre, getTipo(),
                    getHora, getVuelo() );
```

- Si todo lo anterior confunde el código, indentar con 2 tabulaciones:

```
If ((condicion1 and condición2)
                    || (condicion3 && condicion4)
                    || ¡(condicion5 && condicion6))
{
enviarAlerta();
}
```

- Error común en la indentacion:

```
If ((condicion1 and condición2)
    || (condicion3 && condicion4)
    || ¡(condicion5 && condicion6))
{
    enviarAlerta();
}
```

5. Comentarios

Existen dos tipos de comentarios: **de implementación y de documentación**.

Los primeros se usan para comentar el código o una implementación específica a través de los caracteres /*…*/ y //. Sin embargo, los de documentación se usan para describir especificaciones del código usando los caracteres /**…*/.

Recomendación:

- Claros.
- Apropiados.
- Evitar la redundancia.
- No incluir caracteres especiales.
- No hacer uso excesivo de asteriscos u otros caracteres en su diseño.

5.1. Implementación

Los comentarios de implementación se clasifican en:

- ✓ **Comentarios de Bloque**: Son usados al comienzo de cada archivo, antes de cada método o en el interior de estos, con el fin de dar una descripción de los mismos. Están precedidos por una línea en blanco. Ejemplo:

```
/*
Método que retorna el valor total de
la ganancia diaria
*/
```

- ✓ **Comentarios de una línea:** Son comentarios cortos indentados al mismo nivel de la línea de código que sigue. Están precedidos por una línea en blanco. Ejemplo:

```
If (CodAlerta = 1{

/*La alerta es de Relámpagos */
…
}
```

- ✓ **Comentarios de aclaración:** Aparecen en la misma línea del código comentado. Se escriben a una distancia prudente de tal manera que no confundan el código mismo. Cuando se escribe más de un comentario aclaratorio estos deben ser indentados al mismo nivel. Ejemplo:

```
Public int buscarCodigo(double id)
{
      Int pos= -1;                                        /* No encontrado */
      for(int i = 0; i<10; i++)
     {
         if(obtenerCodigo(i) == id)
         {
           pos = i;                                  /*Encontrado*/
           break;
         }
          }
     return pos;
      }
```

- ✓ Comentarios de fin de línea: Convierten una línea o parte de ella en comentario usando los caracteres //. También se pueden comentar secciones de código completas. Ejemplo:

```
Public int buscarCodigo(double id)
{
      Int pos= -1;
//    for(int i = 0; i<10; i++)
       {
         if(obtenerCodigo(i) == id)
         {
           pos = i;
           break;     // No seria necesaria
         }
       //  }
        return pos;
       }
```

5.2. Documentación

Los comentarios de documentación describen las clases, interfaces, constructores, métodos y atributos. Van siempre ubicados antes de la declaración y con los caracteres /**…*/.

Este tipo de comentarios no se escriben dentro del bloque de un método o constructor, ya que Java asocia el comentario con la primer sentencia que se encuentra después de este. Ejemplo:

```
/**
El método buscarCodigo realiza una búsqueda
secuencial del código del cliente dentro del
arreglo…
*/
Public int buscarCodigo (double id){
…
```

6. Declaraciones

6.1. Cantidad

En la cantidad de declaraciones por línea, se recomienda hacer una sola declaración, ya que favorece el uso de comentarios. Ejemplos:

```
Int posición;    // un lugar del arreglo
Int tamanio;    //  tamanio del arreglo
```

Por otro lado, se sugiere no hacer declaraciones de diferentes tipos en la misma línea. Ejemplos:

```
Int posición, array cliente[ ];          (BIEN)


Int posición;                            (MAL)
array cliente[ ];
```

6.2. Inicialización

La inicialización es la asignación de valores iniciales a las variables. Al declararlas, se hace necesario inicializarlas inmediatamente en el lugar en el que han sido declaradas. Si esta acción no se efectúa, debe ser sólo porque el valor inicial de dicha variable depende de un cálculo u operación a realizar. Ejemplos:

```
Int tamanio = 100;
```

```
Public int buscarCodigo (double id){
    Int pos = -1;
   …
```

6.3. Ubicación

La ubicación de las declaraciones es siempre al inicio de cada bloque (código delimitado por {…}), excepto cuando se hace uso de índices en bucles como el for. Ejemplo:

```
Public int buscarCodigo (double id){
  Int por = -1;
  For( int i=0; i<10; i++){
 …
```

Nota: Evitar declaraciones locales que oculten declaraciones de un nivel más alto. Ejemplo:

```
Int pos = 100;
…
Public int buscarCodigo (doublé id){          (MAL)
Int pos = -1;
…
```

6.4. Clases e Interfaces

Para la declaración de clases, se deben tener en cuenta los siguientes aspectos:

1) Debe existir al menos una línea en blanco que separe los métodos.

2) No dejar espacios en blanco entre el nombre del método y el paréntesis de la lista de parámetros.

3) La llave de apertura "{" debe estar ubicada en la siguiente línea de la declaración de sentencia.

4) La de cierre "}" se indenta al comienzo de la línea, de tal manera que se alinee con la sentencia de apertura, excepto cuando se trata de un método vacio.

Ejemplo:

```
public class ArchivoEjemplar
{
  prívate RandomAccessFile archivo;
```

```
 Ejemplar ejemplar;

public void setPosicion(long pos)
{
    try
  {
           archivo.seek(pos);
        } catch (Exception err){ }
     }

 public long getPosicion()
{
     long p=0;
          try
     {
               p = archivo.getFilePointer();
          } catch (Exception error){ }
          return p;
      }
      …
```

7. Espacios en blanco

Son fundamentales en la lectura del código empleado en un programa. Puede darse de dos formas:

7.1. Líneas en blanco

Las líneas separas secciones de código de la siguiente manera:

- ✓ 1 Línea
  - Entre los métodos.
  - Entre variables locales y la primera sentencia de un método.
  - Antes de un comentario de bloque o de línea.

- ✓ 2 Líneas
  - Entre las secciones de un archivo fuente.
  - Entre las definiciones de clases e interfaces.

7.2. Espacios en blanco

Los espacios separan elementos dentro de las líneas de código, así:

- ✓ Palabra clave seguida de paréntesis, no aplica para los métodos.
- ✓ Después de cada coma (,) en una lista de argumentos.
- ✓ Antes y después de cada operador excepto el punto (.) y los de incremento o decremento.
- ✓ En las expresiones dentro de un for.

8. Convenciones de nombre

Ayudan a comprender un programa, ya que el nombre suministra información acerca de la función de un identificador.

8.1. Clases

A la hora de nombrar las diferentes **clases** que son el DAO, Service y Facade se deben tener en cuenta los siguientes aspectos:

- ✓ Nombre debe ser un sustantivo.
- ✓ Si es compuesto, cada palabra debe ser con mayúscula inicial.
- ✓ Nombre simple y descriptivo.
- ✓ Evitar abreviaturas.

Ejemplos:

```
class UsuarioBean;
class UsuarioDao;
class UsuarioService;
class UsuarioFacade;
```

8.2. Métodos

A la hora de nombrar un **método** se deben tener en cuenta los siguientes aspectos:

- ✓ Nombre debe ser un verbo.
- ✓ Si el nombre es simple se escribe en minúscula.
- ✓ Si es compuesto, la primera palabra inicia con minúscula y la segunda con mayúscula.

Ejemplo:

```
String leerLinea();
void guardar();
```

8.3. Variables

A la hora de nombrar una **variable** se debe tener en cuenta los siguientes aspectos:

- ✓ Se escriben en minúscula.
- ✓ Si es compuesta, la primera palabra inicia con minúscula y la segunda con mayúscula.
- ✓ No deben iniciar con "_", ni "$".
- ✓ Nombre corto y con significado.
- ✓ Nombre mnemotécnico.
- ✓ Evitar nombre de un solo carácter, excepto para variables índices (i,j).

Ejemplos:

```
String nombre;
int tipoPelicula;
```