

Connect! / Where is my Friend?

Estándar de Implementación

Versión 1.0

Historia de revisiones

Fecha	Versión	Descripción	Autor
28/08/2013	1.0	Primera versión	Nicolás Díaz

Contenido

CONNECT! – WHERE IS MY FRIEND?	1
ESTÁNDAR DE IMPLEMENTACIÓN.....	1
VERSIÓN 1.0	1
HISTORIA DE REVISIONES	1
CONTENIDO.....	2
1. CONVENCIONES GENERALES	3
1.1. BLOQUES DE CÓDIGO	3
1.2. SENTENCIAS.....	3
1.3. COMENTARIOS	3
1.4. CLASES	4
1.5. VARIABLES	4
1.6. CONSTANTES	5
1.7. INDENTACIÓN GENERAL.....	5
2. CONVENCIONES ESPECÍFICAS C# Y C++ DE VISUAL STUDIO .NET	6
2.1. NOMBRES DE OPERACIONES.....	6
2.2. COMENTARIOS DE OPERACIONES Y CLASES	7
2.3. TIPOS DE DATOS BÁSICOS (C#)	8
3. CONVENCIONES ESPECÍFICAS JAVA.....	9
3.1. NOMBRES DE OPERACIONES.....	9
3.2. COMENTARIOS DE OPERACIONES Y CLASES	10

1. Convenciones Generales

Dado que la implementación será en varios lenguajes, se definirá un estándar para cada lenguaje a utilizar. Algunas de estas pautas son compartidas por todos los lenguajes por lo que se incluirán en esta sección.

1.1. Bloques de código

Los bloques de código entre llaves serán indentados en forma de bloque, con las llaves de apertura y cerrado en el mismo nivel de tabulación.

Incorrecto:

```
if (a > 0) {  
    Console.WriteLine("hola");  
}
```

Correcto:

```
if (a > 0)  
{  
    Console.WriteLine("hola");  
}
```

1.2. Sentencias

Todas las sentencias iterativas como condicionales deberán encontrarse dentro de un bloque de código definido entre llaves.

Incorrecto:

```
if (a > 0) Console.WriteLine("hola");
```

Correcto:

```
if (a > 0)  
{  
    Console.WriteLine("hola");  
}
```

1.3. Comentarios

Los comentarios entre código (es decir, los que comentan una o varias sentencias de código) se agregarán antes de las sentencias comentadas y no a continuación de alguna de las líneas propiamente dichas.

Si el comentario requiere más de una línea, se utilizarán // al comienzo de cada línea y nunca /*

Incorrecto (comentario a continuación de una sentencia):

```
if (a > 0) // Si a es mayor que 0  
{  
    Console.WriteLine("hola");  
}
```

Incorrecto (usa /* */):

```
/* Si a es mayor que 0
entonces imprimo la línea */
if (a > 0)
{
    Console.WriteLine("hola");
}
```

Correcto:

```
// Si a es mayor que 0
// entonces imprimo la línea
if (a > 0)
{
    Console.WriteLine("hola");
}
```

1.4. Clases

Para el nombrado de clases se utilizará en todos los lenguajes el mismo estándar, comenzando siempre con letra mayúscula y separando (si aplica) las palabras que definen la clase por cambio de minúscula a mayúscula.

Incorrecto (comienza con minúscula):

```
classmiClase
{
}
```

Incorrecto (no cambia de min a mayúscula al comenzar otra palabra):

```
classMiclase
{
}
```

Correcto:

```
classMiClase
{
}
```

1.5. Variables

Las variables tanto de clase (atributos) como en el cuerpo de los métodos comenzarán con minúscula, podrán tener letras y números (nunca símbolos), y la separación entre palabras que forman el concepto comenzará con mayúscula como en las clases.

Incorrecto (comienza con mayúscula):

```
int Variable = 7;
```

Incorrecto (separación entre palabras en minúscula):

```
intmivariabile = 7;
```

Correcto:

```
intmiVariable = 7;
```

1.6. Constantes

Las constantes se nombrarán completamente en mayúscula, y se utilizarán underscores (_) para separar las palabras que componen el concepto.

Incorrecto (no está toda en minúscula):

```
constint miConstante = 0;
```

Incorrecto (sin separación entre palabras):

```
constint MICONSTANTE = 0;
```

Correcto:

```
constint MI_CONSTANTE = 0;
```

1.7. Indentación general

Los bloques de código deberán ser indentados de acuerdo a la jerarquía a la que pertenecen con TAB.

Incorrecto:

```
class MiClase
{
void MetodoEjemplo()
{
if (a > 0)
{
Console.WriteLine("hola");
}
}
}
```

Correcto:

```
class MiClase
{
void MetodoEjemplo()
{
{
if (a > 0)
{
Console.WriteLine("hola");
}
}
}
}
```

2. Convenciones Específicas C# y C++ de Visual Studio .NET

2.1. Nombres de operaciones

Los nombres de las operaciones deberán utilizar exclusivamente letras o números, nunca símbolos, y deberá comenzar en mayúscula, y cambiará a mayúscula en cada nombre que la componga.

Incorrecto (comienza con minúscula):

```
class MiClase
{
    void metodoEjemplo()
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Incorrecto (no cambia a mayúscula al cambiar palabra):

```
class MiClase
{
    void Metodoejemplo()
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Incorrecto (tiene caracteres no alfanuméricos):

```
class MiClase
{
    void Metodo_Ejemplo()
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Correcto:

```
class MiClase
{
    void MetodoEjemplo()
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

2.2. Comentarios de operaciones y clases

Las operaciones y clases deberán estar comentadas antes de la definición de la misma, con un comentario cuyas líneas comienzan con `///` y tienen una serie de definiciones que la propia IDE generará y se deberá rellenar para todos los parámetros.

Incorrecto (no usa comentarios de VS) :

```
class MiClase
{
    // Método de ejemplo
    void MetodoEjemplo()
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Incorrecto (El comentario del método no define la variable a) :

```
class MiClase
{
    ///<summary>
    /// Método de ejemplo
    ///</summary>
    ///<param name="a"></param>
    void MetodoEjemplo(int a)
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Correcto:

```
///<summary>
///Clase de ejemplo
///</summary>
class MiClase
{
    ///<summary>
    /// Método de ejemplo
    ///</summary>
    ///<param name="a">Variable de decisión</param>
    void MetodoEjemplo(int a)
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

2.3. Tipos de datos básicos (C#)

Los tipos de datos básicos de C# deberán ser utilizados con su nomenclatura unboxed, es decir el data type y no la clase que lo representa.

Incorrecto (se utiliza la clase Int32 en vez del tipo básico int):

```
class MiClase
{
    void MetodoEjemplo(Int32 a)
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Correcto:

```
class MiClase
{
    void MetodoEjemplo(int a)
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```


3. Convenciones Específicas Java

3.1. Nombres de operaciones

Los nombres de las operaciones deberán utilizar exclusivamente letras o números, nunca símbolos, y deberá comenzar en minúscula, y cambiará a mayúscula en cada nombre que la componga.

Incorrecto (el nombre de la operación comienza con mayúscula):

```
public class MiClase
{
    void MetodoEjemplo(booleana)
    {
        if (a)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Incorrecto (tiene caracteres no alfanuméricos):

```
class MiClase
{
    void metodo_ejemplo(booleana)
    {
        if (a > 0)
        {
            Console.WriteLine("hola");
        }
    }
}
```

Correcto:

```
public class MiClase
{
    void metodoEjemplo(booleana)
    {
        if (a)
        {
            Console.WriteLine("hola");
        }
    }
}
```

3.2. Comentarios de operaciones y clases

Las operaciones y clases deberán estar comentadas utilizando el estándar javadoc, antes de la definición de la misma, con un comentario cuyas líneas están entre `/* */`, y tienen una serie de definiciones para cada parámetro que se deberá rellenar para todos los parámetros.

Incorrecto (no brinda información sobre el parámetro a):

```
/**
 * Método de ejemplo
 *
 * @return retorna -1 si a es TRUE
 */
int MetodoEjemplo(booleana)
{
    if (a)
    {
        return -1;
    }
}
```

Correcto:

```
/**
 * Método de ejemplo
 *
 * @param a indica si la condición es válida
 * @return retorna -1 si a es TRUE
 */
int MetodoEjemplo(booleana)
{
    if (a)
    {
        return -1;
    }
}
```