**Proyecto “Aplicación móvil de ayuda para la toma**

de decisiones de cocina”

Documento de Estándares de Programación

Versión 1.0

Historia de Revisión

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Historial de revisiones | | | | |
| Ítem | Fecha | Versión | Descripción | Equipo |
| 1 | 20/05/23 | 1.0 | Elaboración inicial del documento de estándares de programación | Villanueva Yucra, Josue  Neira Machaca, Javier  Mamani Condori, Gilmer |

Contenido

[1. OBJETIVO 5](#_Toc99472662)

[2. DECLARACION DE VARIABLES 6](#_Toc99472663)

[2.1 Descripción de la Variable. 6](#_Toc99472664)

[2.2 Variables de Tipo Arreglo 8](#_Toc99472665)

[3. Definición de Controles 9](#_Toc99472666)

[3.1 Tipo de datos 9](#_Toc99472667)

[3.2 Prefijo para el Control 9](#_Toc99472668)

[3.3 Nombre descriptivo del Control 9](#_Toc99472669)

[3.4 Sangría, espaciamiento y comentarios 10](#_Toc99472670)

[3.5 Declaración de variables, atributos y objetos 12](#_Toc99472671)

[3.6 Declaración de clases 13](#_Toc99472672)

[3.7 Declaración de métodos 14](#_Toc99472673)

[3.8 Declaración de funciones 15](#_Toc99472674)

[3.9 Control de versiones de código fuente 15](#_Toc99472675)

[4. Clases. 18](#_Toc99472677)

[4.1 Clases por archivo 18](#_Toc99472678)

[4.2 Visibilidad de atributos de instancia y de clase 19](#_Toc99472679)

[4.3 Referencias a miembros de una clase 19](#_Toc99472680)

[5. Métodos, Procedimientos y Funciones definidos por el Usuario. 19](#_Toc99472681)

[5.1 Código atómico 20](#_Toc99472682)

[5.2 Métodos largos 21](#_Toc99472683)

[5.3 Uso de return 21](#_Toc99472684)

[5.4 Métodos y propiedades Públicas 22](#_Toc99472685)

[5.5 Número de parámetros en un método 22](#_Toc99472686)

[5.6 Número de parámetros en un método 22](#_Toc99472687)

[5.7 Retorno de datos complejos 22](#_Toc99472688)

[5.8 Versionamiento de librerías o código 22](#_Toc99472689)

[5.9 Manejo de errores 23](#_Toc99472690)

[5.10 Bitácoras de acciones 23](#_Toc99472691)

[5.11 Bloque finally para conexiones a base de datos, archivos, sockets, etc 23](#_Toc99472692)

[6. Beneficios 23](#_Toc99472693)

[7. Conclusiones 24](#_Toc99472694)

Estándares de Programación

# OBJETIVO

Reglamentar la forma en que se implementará el código fuente del proyecto, pasando, por las variables, controles, clases, métodos, ficheros, archivos y todo aquello que esté implicado en el código,

Mejorar y uniformizar a través de las reglas que se proponen, el estilo de programación que tiene cada programador.

* Los nombres de variables serán mnemotécnicos con lo que se podrá saber el tipo de dato de cada variable con sólo ver el nombre de la variable.
* Los nombres de variables serán sugestivos, de tal forma que se podrá saber el uso y finalidad de dicha variable o función fácilmente con solo ver el nombre de la variable.
* La decisión de poner un nombre a una variable o función será mecánica y automática, puesto que seguirá las reglas definidas por nuestro estándar.
* Permite el uso de herramientas automáticas de verificación de nomenclaturas.

Por tanto, se seguirán dichos patrones para un entendimiento legible del código y para facilitar el mantenimiento de este.

# DECLARACION DE VARIABLES

Se propone que la declaración de las variables, se ajusten al motivo para la que se requieran. El nemotécnico definido se establece tomando en consideración principalmente lo siguiente:

* La longitud debe ser lo más recomendable posible. No debe ser tan grande de tal forma que el programador tenga la facilidad de manejo sobre la variable y ni tan corta que no pueda describirse claramente. Para el caso establecemos una longitud máxima de variable de 16 caracteres.
* El tipo de dato al que pertenece la variable.

Por lo tanto la estructura de la variable es como sigue:

| **Estructura** | **Descripción de la Variable** |
| --- | --- |
| LONGITUD. MAX. | ß 1 àß 16 à |
| FORMATO | Minúscula la primera parte y luego la segunda con Mayúsculas |
| EJEMPLO | numCuenta |

Siendo el nombre que identifica a la variable: **previewWidth**

## Descripción de la Variable.

Nombre que se le asignará a la variable para que se le identifique y deberá de estar asociada al motivo para la cual se le declara.

El nombre de las variables no puede ser igual a alguna palabra reservada del lenguaje de programación.

* + as
  + abstract
  + base
  + bool
  + break
  + byte
  + case
  + catch
  + char
  + checked
  + class
  + const
  + continue
  + decimal
  + default
  + delegate
  + do
  + double
  + else
  + enum
  + event
  + explicit
  + extern
  + false
  + finally
  + fixed
  + float
  + for
  + foreach
  + goto
  + if
  + implicit
  + in
  + int
  + interface
  + internal
  + is
  + lock
  + long
  + namespace
  + new
  + null
  + object
  + operator
  + out
  + override
  + params
  + private
  + protected
  + public
  + readonly
  + ref
  + return
  + sbyte
  + sealed
  + short
  + sizeof
  + stackalloc
  + static
  + string
  + struct
  + switch
  + this
  + throw
  + true
  + try
  + typeof
  + uint
  + ulong
  + unchecked
  + unsafe
  + ushort
  + using
  + virtual
  + void
  + volatile
  + while

## 

# Definición de Controles

Para poder determinar el nombre de un control dentro de cualquier aplicación de tipo visual, se procede a identificar el tipo al cual pertenece y la función que cumple dentro de la aplicación.

## Tipo de datos

|  |  |  |
| --- | --- | --- |
| **Tipo de variable** | **Mnemónico** | **Descripción** |
| Byte | by | Entero de 8 bits sin signo. |
| Integer | in | Entero de 32 bits con signo. |
| Char | ch | Un carácter UNICODE de 16 bits |
| String | st | Cadena de caracteres |
| Date | dt | Formato de fecha/hora |
| Boolean | bl | Valor lógico: verdadero y falso |
| Float | fl | Coma flotantes, 11-12 dígitos significativos. |
| Double | db | Coma flotante, 64 bits (15-16 dígitos significativos) |

## Prefijo para el Control

El prefijo del control será determinado mediante tres caracteres que estarán conformados por las consonantes más representativas del control, es así, por ejemplo; el control Button, estará asociado al prefijo btn.

## Nombre descriptivo del Control

Formado por la descripción de la función que lleva a cabo el control, esta debe ser descrita en forma específica y clara.

|  |  |  |
| --- | --- | --- |
| **Tipo de control** | **Prefijo** | **Ejemplo** |
| Edit text | et | et\_Nombre |
| TextBox | txt | txt\_Apellido |
| Button | bt | btn\_Login |
| RadioButton | rdo | rdo\_Seleccion |
| CheckBox | chk | chk\_Ruta |
| DropDownList | cmb | cmb\_Documentos |
| DataGrid | dtg | dtg\_Estudiantes |
| Image | img | img\_Foto |
| TextView | tw | tw\_Lista |
| Table | tbl | tblAlumnos |
| Layout | ly | ly\_nombre |
| viewHeaderBackground | vh | vh\_Notas |
| Card | cr | cr\_Lista |

## Sangría, espaciamiento y comentarios

Para la sangría se usará TAB, jamás de usarán espacios, la sangria deberá estar definida por 2 TABs. Los comentarios deben estar al mismo nivel que el código.

Las llaves { } deben estar en el mismo nivel que el código de las llaves. También, deben de estar siempre en líneas separadas y no en la misma línea de las sentencias if, for, while, switch, etc, esto facilita la lectura y compresión del código.

Usar una línea en blanco como separador para dos grupos lógicos de código y deberá de haber solo una línea en blanco para separar dos métodos dentro de una clase.

## Declaración de variables, atributos y objetos

1. Se debe declarar una variable por línea.
2. Cuando se declara una variable es imperativo usar los tipos específicos de las mismas en lugar de usar los que se encuentran en el espacio de nombres System.
3. Evitar usar variables globales. Declarar variables locales siempre que sea necesario y pasarlas a otro método en vez de compartir una variable global entre métodos.

|  |  |
| --- | --- |
| **Título** | **Descripción** |
| **Sintaxis** | [TipoVariable] [Nombre de la Variable] |
| **Descripción** | Todas las variables o atributo tendrán una longitud máxima de 30 caracteres.  El nombre de la variable puede incluir más de un sustantivo los cuales se escribirán juntos.  Si se tuvieran variables que puedan tomar nombres iguales, se le agregará un número asociado (si está dentro de un mismo método será correlativo). |
| **Tipo** | Las variables de clase estática, se tendrá como orden en primer lugar las variables de clase públicas (public), después las protegidas (protected), posteriormente las de nivel de paquete (sin modificador), y por último las privadas (private).  En las variables de instancia se tendrá primero las públicas (public), después las protegidas (protected), luego las de nivel de paquete (sin modificador), y finalmente las privadas (private). |
| **Observaciones** | En la declaración de variables o atributos no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ]. * Caracteres tildados: á, é, í, ó, ú.   No se usará abreviaciones a menos que sea necesario porque supera la cantidad de caracteres.  Evitar el uso de nombres como i, a, s, t, es mejor usar nombres descriptivos a excepción de iteración en los ciclos. |

## Declaración de clases

|  |  |
| --- | --- |
| **Título** | **Descripción** |
| **Sintaxis** | [Tipo] Class [Nombre de Clase] |
| **Descripción** | El nombre de las clases tendrá una longitud máxima de 30 caracteres y las primeras letras de todas las palabras estarán en mayúsculas. Tipo se refiere a si la clase será: Private, Public o Protected.  Los nombres serán simples y descriptivos. Las interfaces se nombrarán siguiendo los mismos criterios que los indicados para las clases. Como norma general toda interfaz se nombrará con el prefijo "I" para diferenciarla de la clase que la implementa (que tendrá el mismo nombre sin el prefijo "I"). |
| **Observaciones** | En la declaración de clases no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ]. * Caracteres tildados: á, é, í, ó, ú.   Debe evitarse el uso de acrónimos o abreviaturas, salvo en aquellos casos en los que dicha abreviatura sea más utilizada que la palabra que representa (URL, HTTP, etc.). |
| **Ejemplo** | Private Class Empleado  Indica una clase Empleado |

## Declaración de métodos

|  |  |
| --- | --- |
| **Título** | **Descripción** |
| **Sintaxis** | nombreProcedim[(ListaParámetros)] |
| **Descripción** | El nombre del método constará hasta de 25 caracteres.  La primera letra de la primera palabra del nombre será escrita en minúscula y las siguientes palabras empezarán con letra mayúscula. |
| **Observaciones** | En la declaración de métodos no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ], \_. * Caracteres tildados: á, é, í, ó, ú. |
| **Ejemplo** | Protected calcularSueldo(String empleado)  Indica un método calcularSueldo que recibe una variable por valor de tipo string al ámbito de la clase |

## Declaración de funciones

|  |  |
| --- | --- |
| **Título** | **Descripción** |
| **Sintaxis** | [TipoDato] nombreFuncion[(ListaParámetros)] |
| **Descripción** | El nombre del objeto constará hasta de 25 caracteres, no es necesario colocar un nombre que indique la clase a la cual pertenece.  La primera letra de la primera palabra del nombre será escrita en mayúsculas  El tipo de dato de retorno se coloca al final y será obligatorio colocarlo. |
| **Observaciones** | En la declaración de objetos no se deberá utilizar caracteres como:   * Letra Ñ o ñ. * Caracteres especiales ¡, ^, #, $, %, &, /, (, ), ¿, ‘, +, -, \*, {, }, [, ], \_. * Caracteres tildados: á, é, í, ó, ú. |
| **Ejemplo** | Public int sumar(int A, int B)  Indica una función que suma dos variables enteras |

## Control de versiones de código fuente

Cada modificación realizada será guardada de la forma:

|  |  |
| --- | --- |
| **Título** | **Descripción** |
| **Formato** | [NOMBRE DOCUMENTO][ \_ ][FECHA][ \_ ][HORA] donde y la fecha estará en formato yyyymmdd y la hora en formato HHMM. |

## Métodos largos

Se debe de evitar escribir métodos muy largos en la medida de lo posible, típicamente un método deberá de tener entre 1 y 40 líneas de código, si el método contiene más de 40 líneas de código se deberá de considerar el refactorizarlo en métodos separados.

## Uso de return

Un método solo deberá de tener una entrada y una salida de tal forma que el uso de la directiva return debe de estar al final del método, no es recomendable tener múltiples return en la rutina de código.

## Métodos y propiedades Públicas

Se debe de evitar generar métodos y propiedades públicos, a menos que sea muy necesario el uso de ellos de forma externa, se debe de usar el tipo “internal” si es que ellas son accedidas desde el mismo ensamblado.

internal class PruebaClaseInterna

{

public int MiPruebaClaseInterna;

}

## Número de parámetros en un método

Si existe la necesidad de pasar más de 8~10 parámetros a un método especifico, este es un buen candidato para definir una clase con parámetros y una estructura de datos tipada, de lo contrario el consumo de memoria es afectado de forma considerable, permite la corrupción de datos, y se castiga en ciclos de procesamiento al servidor lo cual pega en performance.

## Retorno de datos complejos

Si tienes un método que regresa una colección, es recomendable que si el resultado no trajo datos es mejor regresar una colección vacía que nula, esto hará que siempre se espere un tipo de datos y el manejo de validación o excepciones sea el más óptimo.

## Versionamiento de librerías o código

Se deberá de usar el archivo AssemblyInfo para llenar la información correspondiente al número de versión, la descripción general del ensamblado, derechos de autor, nombre de la compañía etc, en caso de clases que no generen dll se deberá de colocar al inicio de la clase dicha información.

## Manejo de errores

Se deberá de registrar en un log de base de datos el error descriptivo, el cual deberá de permitir la rastreabilidad y trazabilidad del evento, que lo ocasiono, quien lo genero, que proceso se ejecutó y en qué momento sucedió, este log deberá de ser eliminado en un periodo de tiempo optimo y deberá de ser eficiente para no afectar a las soluciones que lo están invocando.

## Bitácoras de acciones

Se deberá de registrar en una bitácora de base de datos el mensaje que se requiere almacenar, el cual deberá de permitir la rastreabilidad y trazabilidad del evento, quien lo genero, que proceso se ejecutó y en qué momento sucedió, este log deberá de ser eliminado en un periodo de tiempo optimo y deberá de ser eficiente para no afectar a las soluciones que lo están invocando.

## Bloque finally para conexiones a base de datos, archivos, sockets, etc

Cuando se hagan conexiones a base de datos, archivos, sockets, etc, se deberá de asegurar que en el bloque finally se cierren dichas conexiones a los recursos utilizados en la rutina programada, esto asegurara que, aunque una excepción no controlada exista, este código se ejecute cerrando los objetos usados no dejándolos abiertos utilizando memoria o recursos del servidor donde se está ejecutando.

# Beneficios

* La documentación hace más legible un programa.
* Al documentar bien un programa desde un principio se evita que para cada modificación deba estudiarse profundamente el funcionamiento del programa, redescubriendo todo lo no documentado, con la ventaja adicional de que generalmente quién modifica el programa no es siempre quién lo escribió.
* Facilita la reutilización de módulos y rutinas desde cualquier otro programa o el mismo.
* Ayuda a determinar cuándo debe ser reescrito un código. Si existen problemas para explicar el código con un comentario, probablemente el código esté mal escrito.

# Conclusiones

* Una buena programación e implementación legible, solo se logra usando y llevando de la mano un buen estándar o patrón de programación.
* Es muy importante que el programador tenga un conocimiento previo del estándar o en su defecto que lea el documento para prever diferencias.
* Al documentar se obtienen dos cosas fundamentales, un documento legible y segundo una buena base para los futuros desarrollos de mantenimiento del código.