**Proyecto *Sistema Web para la publicación de eventos y emisión de la Radio Conexion Latam***

**Documento de Estándares de Programación**

**Versión 1.0**

**Historia de Revisión**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Historial de revisiones** | | | | |
| **Ítem** | **Fecha** | **Versión** | **Descripción** | **Equipo** |
| 1 | 01/07/2024 | 1.0 | Versión 1 | * Chambilla Martinez, Renato Eduardo (2020066918) * Lupaca Mamani, Ronal Daniel (2020067146) * Valle Bustamante, Gustavo Alonso (2020066916) * Viveros Blanco, Farley Eduardo (2020066896) |

**Tabla de Contenidos**

1. [OBJETIVO 4](#_TOC_250015)
2. [DECLARACION DE VARIABLES 5](#_TOC_250014)
   1. [Descripción de la Variable. 5](#_TOC_250013)
   2. Variables de Tipo Arreglo 6
3. [Definición de Controles 6](#_TOC_250012)
   1. [Tipo de datos 6](#_TOC_250011)
   2. [Prefijo para el Control 6](#_TOC_250010)
   3. [Nombre descriptivo del Control 6](#_TOC_250009)
   4. [Declaración de variables, atributos y objetos 6](#_TOC_250008)
   5. [Declaración de clases 7](#_TOC_250007)
   6. [Declaración de métodos 7](#_TOC_250006)
   7. [Declaración de funciones 8](#_TOC_250005)
   8. [Control de versiones de código fuente 8](#_TOC_250004)
4. [Clases. 9](#_TOC_250003)
5. [Métodos, Procedimientos y Funciones definidos por el Usuario. 9](#_TOC_250002)
6. [Beneficios 10](#_TOC_250001)
7. [Conclusiones 10](#_TOC_250000)

**Estándares de Programación**

# OBJETIVO

El objetivo de este documento es establecer los estándares de programación y las mejores prácticas para el desarrollo de aplicaciones dentro de la organización RadioConexionLatam. Este documento tiene como propósito principal:

- Asegurar la consistencia y calidad del código fuente.

- Facilitar el mantenimiento y evolución de las aplicaciones.

- Promover el uso de buenas prácticas en el desarrollo de software.

- Reducir la posibilidad de errores y mejorar la eficiencia del desarrollo.

- Proporcionar una guía clara y detallada para los desarrolladores en cuanto a la estructura, estilo y convenciones de codificación.

Este documento está dirigido a todos los desarrolladores y equipos técnicos que participen en el desarrollo, mantenimiento y revisión de las aplicaciones de RadioConexionLatam.

# DECLARACION DE VARIABLES

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

* **Longitud**: La longitud debe ser lo más adecuada posible. No debe ser tan grande que dificulte su manejo, ni tan corta que no pueda describirse claramente. Se establece una longitud máxima de variable de 16 caracteres.
* **Alcance de la variable**: A medida que aumenta el tamaño del proyecto, también aumenta la necesidad de reconocer rápidamente el alcance de las variables. Esto se consigue al escribir un prefijo de alcance de una letra delante del tipo de prefijo propio, sin aumentar demasiado la longitud del nombre de las variables.

|  |  |  |
| --- | --- | --- |
| **Alcance** | **Prefijo** | **Ejemplo** |
| Global | Ninguno | anuncioRepository, categoriaRepository, eventoService, imagenEmitter, videoEmitter |
| Nivel de la clase | Ninguno | anuncio, categoria, evento, imagen, video |
| Local del método | Ninguno | var anuncios = GetAnuncios(), List<Categoria> categorias |
| Público | id | idAnuncio, idCategoria, idEvento, idImagen, idVideo, fechaPublicacion, titulo |
| Privado | Ninguno | \_anuncioService, \_categoriaService, \_eventoService, \_imagenService, \_videoService |

**Tipo de Dato**

* **Int**: Variables de tipo entero, usualmente para identificadores y contadores.
  + Ejemplo: idAnuncio, idCategoria, visitas
* **String**: Variables de tipo cadena, para textos y descripciones.
  + Ejemplo: titulo, subtitulo, estado, nombreUsuario
* **DateTime**: Variables de tipo fecha y hora.
  + Ejemplo: fechaPublicacion, fechaEvento
* **Byte Array**: Variables para almacenar imágenes o archivos binarios.
  + Ejemplo: Imagen

**Ejemplos de Declaración de Variables**

* **Globales**:

csharp

Copiar código

public class AnuncioRepository { ... }

public class CategoriaRepository { ... }

public class EventoService { ... }

public Emitter<String> imagenEmitter;

public Emitter<String> videoEmitter;

* **A Nivel de Clase**:

csharp

Copiar código

public class Anuncios

{

public int idAnuncio { get; set; }

public string titulo { get; set; }

public string subtitulo { get; set; }

public string contenido { get; set; }

public DateTime fechaPublicacion { get; set; }

public int? idUsuario { get; set; }

public int? idCategoria { get; set; }

public int? idImagenPrincipal { get; set; }

public int? idVideoPrincipal { get; set; }

public string estado { get; set; }

public virtual Categorias Categorias { get; set; }

public virtual Imagenes Imagenes { get; set; }

public virtual Usuarios Usuarios { get; set; }

public virtual Videos Videos { get; set; }

}

* **Locales del Método**:

csharp

Copiar código

public List<Anuncios> ListarAnuncios()

{

var anuncios = new List<Anuncios>();

using (var db = new Model1())

{

anuncios = db.Anuncios.ToList();

}

return anuncios;

}

* **Privadas**:

csharp

Copiar código

private AnuncioService \_anuncioService;

private CategoriaService \_categoriaService;

private EventoService \_eventoService;

private ImagenService \_imagenService;

private VideoService \_videoService;

**Tipo de Dato al que Pertenece la Variable**

Las variables deben ser declaradas de acuerdo al tipo de dato que representan, asegurando que el nombre de la variable sea descriptivo y refleje claramente su propósito y contenido.

**Estructura de la Variable**

* **Longitud Máxima**: Hasta 16 caracteres.
* **Formato**: Minúscula la primera parte y luego la segunda con Mayúsculas.
* **Ejemplo**: idAnuncio

**Tabla de Estructura y Descripción de Variables**

|  |  |
| --- | --- |
| Estructura | Descripción de la Variable |
| LONGITUD MAX | 1 - 16 caracteres |
| FORMATO | Minúscula la primera parte y luego la segunda con Mayúsculas |
| EJEMPLO | idAnuncio, fechaPublicacion, nombreEvento |

#### **Tipos de Datos y Ejemplos**

|  |  |  |
| --- | --- | --- |
| **Tipo de Dato** | **Descripción** | **Ejemplo** |
| **Int** | Identificadores y contadores | idAnuncio, idCategoria, visitas |
| **String** | Textos y descripciones | titulo, subtitulo, estado, nombreUsuario |
| **DateTime** | Fechas y horas | fechaPublicacion, fechaEvento |
| **Byte Array** | Imágenes o archivos binarios | Imagen |

**Ejemplos de Variables Declaradas**

* **Variables de Tipo Int**:

csharp

Copiar código

public int idAnuncio { get; set; }

public int idCategoria { get; set; }

public int visitas { get; set; }

* **Variables de Tipo String**:

csharp

Copiar código

public string titulo { get; set; }

public string subtitulo { get; set; }

public string estado { get; set; }

public string nombreUsuario { get; set; }

* **Variables de Tipo DateTime**:

csharp

Copiar código

public DateTime fechaPublicacion { get; set; }

public DateTime fechaEvento { get; set; }

* **Variables de Tipo Byte Array**:

csharp

Copiar código

public byte[] Imagen { get; set; }

## Descripción de la Variable.

El nombre que se le asignará a la variable debe ser descriptivo y estar asociado al propósito para el cual se declara. A continuación se presentan las variables utilizadas en el proyecto **RadioConexionLatam**, con sus respectivas descripciones y ejemplos.

**Variables del Modelo Anuncios**

* idAnuncio: Identificador único del anuncio.
* titulo: Título del anuncio.
* subtitulo: Subtítulo del anuncio.
* contenido: Contenido del anuncio.
* fechaPublicacion: Fecha y hora de la publicación del anuncio.
* idUsuario: Identificador del usuario que creó el anuncio.
* idCategoria: Identificador de la categoría a la que pertenece el anuncio.
* idImagenPrincipal: Identificador de la imagen principal asociada al anuncio.
* idVideoPrincipal: Identificador del video principal asociado al anuncio.
* estado: Estado del anuncio (activo o inactivo).
* visitas: Número de visitas que ha recibido el anuncio.
* ImagenRuta: Ruta de la imagen principal (propiedad no mapeada).
* VideoUrl: URL del video principal (propiedad no mapeada).

**Variables del Modelo Categorias**

* idCategoria: Identificador único de la categoría.
* nombre: Nombre de la categoría.

**Variables del Modelo EnlacesRelacionados**

* idEnlace: Identificador único del enlace relacionado.
* idAnuncio: Identificador del anuncio al que está relacionado el enlace.
* url: URL del enlace relacionado.
* descripcion: Descripción del enlace relacionado.

**Variables del Modelo Eventos**

* idEvento: Identificador único del evento.
* nombreEvento: Nombre del evento.
* descripcion: Descripción del evento.
* fechaEvento: Fecha y hora del evento.
* lugar: Lugar donde se llevará a cabo el evento.
* organizador: Organizador del evento.
* estado: Estado del evento (activo o inactivo).
* capacidad: Capacidad del evento.
* idCategoria: Identificador de la categoría a la que pertenece el evento.
* Imagen: Imagen asociada al evento.

**Variables del Modelo Imagenes**

* idImagen: Identificador único de la imagen.
* url: URL de la imagen.
* descripcion: Descripción de la imagen.

**Variables del Modelo Usuarios**

* idUsuario: Identificador único del usuario.
* nombre: Nombre del usuario.
* apellido: Apellido del usuario.
* correo: Correo electrónico del usuario.
* idRol: Identificador del rol del usuario.
* contrasena: Contraseña del usuario.
* estado: Estado del usuario (activo o inactivo).

**Variables del Modelo Roles**

* idRol: Identificador único del rol.
* descripcion: Descripción del rol.

**Variables del Modelo Videos**

* idVideo: Identificador único del video.
* url: URL del video.
* descripcion: Descripción del video.

**Variables del Modelo ProgramacionSemanal**

* Id: Identificador único de la programación.
* Dia: Día de la semana en el que se transmite el programa.
* Hora: Hora de transmisión del programa.
* NombrePrograma: Nombre del programa.
* idUsuario: Identificador del usuario responsable del programa.
* idCategoria: Identificador de la categoría del programa.
  1. **Variables de Tipo Lista**

En el caso de las definiciones de listas de elementos, se declarará la variable con el prefijo de las 3-4 primeras letras abreviadas, tomando como ejemplo a “list”. Esto nos permitirá entender que se trata de una variable del tipo colección que contendrá de cero a más datos,

A continuación, se presentan ejemplos de variables de tipo lista en el proyecto **RadioConexionLatam**:

#### Modelo Anuncios

csharp

Copiar código

public class Anuncios

{

// Otras propiedades...

// Lista de enlaces relacionados al anuncio

public virtual ICollection<EnlacesRelacionados> listEnlacesRelacionados { get; set; }

// Constructor

public Anuncios()

{

listEnlacesRelacionados = new HashSet<EnlacesRelacionados>();

}

// Método para listar anuncios

public List<Anuncios> Listar()

{

var listAnuncios = new List<Anuncios>();

try

{

using (var db = new Model1())

{

listAnuncios = db.Anuncios.ToList();

}

}

catch (Exception)

{

throw;

}

return listAnuncios;

}

// Método para buscar anuncios

public List<Anuncios> Buscar(string Buscar)

{

var listAnuncios = new List<Anuncios>();

try

{

using (var db = new Model1())

{

if (int.TryParse(Buscar, out int idAnuncio))

{

listAnuncios = db.Anuncios.Where(x => x.idAnuncio == idAnuncio).ToList();

}

else

{

listAnuncios = db.Anuncios.Where(x => x.titulo.Contains(Buscar)).ToList();

}

}

}

catch (Exception ex)

{

throw new ApplicationException("Error al buscar anuncios: " + ex.Message, ex);

}

return listAnuncios;

}

}

#### Modelo Categorias

csharp

Copiar código

public class Categorias

{

// Otras propiedades...

// Lista de anuncios asociados a la categoría

public virtual ICollection<Anuncios> listAnuncios { get; set; }

// Constructor

public Categorias()

{

listAnuncios = new HashSet<Anuncios>();

}

}

#### Modelo Usuarios

csharp

Copiar código

public class Usuarios

{

// Otras propiedades...

// Lista de anuncios creados por el usuario

public virtual ICollection<Anuncios> listAnuncios { get; set; }

// Constructor

public Usuarios()

{

listAnuncios = new HashSet<Anuncios>();

}

}

### Declaraciones en los Controladores

#### Controlador AnuncioController

csharp

Copiar código

public class AnuncioController : Controller

{

// Lista de categorías para la vista

public ActionResult CrearAnuncio()

{

ViewBag.listCategorias = new SelectList(GetCategorias(), "idCategoria", "nombre");

return View();

}

// Método para obtener las categorías

private List<Categorias> GetCategorias()

{

using (var db = new Model1())

{

var listCategorias = db.Categorias.ToList();

return listCategorias;

}

}

// Método para visualizar anuncios

public ActionResult VisualizarAnuncio(string Buscar)

{

List<Anuncios> listAnuncios;

using (var db = new Model1())

{

if (string.IsNullOrEmpty(Buscar))

{

listAnuncios = db.Anuncios.ToList();

}

else

{

listAnuncios = db.Anuncios

.Where(x => x.idAnuncio.ToString() == Buscar || x.titulo.Contains(Buscar))

.ToList();

}

}

return View(listAnuncios);

}

}

#### Controlador EventosController

csharp

Copiar código

public class EventosController : Controller

{

// Lista de eventos para la vista

public ActionResult Index()

{

var listEventos = db.Eventos.Include(e => e.Categoria).ToList();

return View(listEventos);

}

// Método para obtener la lista de categorías

public ActionResult CrearEvento()

{

ViewBag.listCategorias = new SelectList(db.Categorias, "idCategoria", "nombre");

return View();

}

// Método para visualizar la publicidad de eventos

public ActionResult VistaPublicitaria()

{

var listEventosActivos = db.Eventos.Include(e => e.Categoria)

.Where(e => e.estado == "A")

.ToList();

return View(listEventosActivos);

}

}

# 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. Esta práctica permite mantener un código más organizado y entendible, facilitando su mantenimiento y ampliación.

**Reglas de Nomenclatura**

1. **Prefijo por Tipo de Control**: Cada tipo de control tendrá un prefijo específico que lo identifique claramente.
2. **Función del Control**: El nombre del control debe describir su función o propósito dentro de la aplicación.
3. **Formato**: El prefijo estará en minúscula seguido por el nombre descriptivo del control con la primera letra en mayúscula.

**Prefijos Comunes para Controles**

|  |  |  |
| --- | --- | --- |
| **Tipo de Control** | **Prefijo** | **Ejemplo** |
| Button | btn | btnGuardar |
| TextBox | txt | txtNombre |
| Label | lbl | lblNombre |
| DropDownList | ddl | ddlCategorias |
| GridView | gv | gvAnuncios |
| CheckBox | chk | chkActivo |
| RadioButton | rb | rbOpcion1 |
| HyperLink | hl | hlPaginaPrincipal |
| Image | img | imgLogo |
| Panel | pnl | pnlFormulario |
| ListBox | lst | lstUsuarios |
| DataList | dl | dlEventos |
| Repeater | rpt | rptComentarios |
| FileUpload | fu | fuImagen |

**Ejemplos en el Proyecto RadioConexionLatam**

A continuación, se presentan ejemplos de nombres de controles en el proyecto **RadioConexionLatam**:

**Controlador AnuncioController**

csharp

Copiar código

public class AnuncioController : Controller

{

private Anuncios objAnuncios = new Anuncios();

private Model1 db = new Model1();

// GET: Anuncio

public ActionResult Index()

{

return View();

}

public ActionResult CrearAnuncio()

{

ViewBag.ddlCategorias = new SelectList(GetCategorias(), "idCategoria", "nombre");

return View();

}

[HttpPost]

[ValidateInput(false)]

public ActionResult CrearAnuncio(Anuncios anuncio, HttpPostedFileBase fuImagenFile, string txtVideoUrl)

{

try

{

if (ModelState.IsValid)

{

// Procesar imagen

if (fuImagenFile != null && fuImagenFile.ContentLength > 0)

{

var fileName = Path.GetFileName(fuImagenFile.FileName);

var path = Path.Combine(Server.MapPath("~/Content/images/Anuncios"), fileName);

fuImagenFile.SaveAs(path);

anuncio.idImagenPrincipal = SaveImageToDatabase(fileName, path);

}

// Procesar video

if (!string.IsNullOrEmpty(txtVideoUrl))

{

anuncio.idVideoPrincipal = SaveVideoToDatabase(txtVideoUrl);

}

anuncio.fechaPublicacion = DateTime.Now;

anuncio.Guardar();

return RedirectToAction("VisualizarAnuncio");

}

}

catch (Exception ex)

{

ModelState.AddModelError("", "Ocurrió un error al crear el anuncio: " + ex.Message);

}

ViewBag.ddlCategorias = new SelectList(GetCategorias(), "idCategoria", "nombre");

return View(anuncio);

}

public ActionResult VisualizarAnuncio(string txtBuscar)

{

List<Anuncios> anuncios;

using (var db = new Model1())

{

if (string.IsNullOrEmpty(txtBuscar))

{

anuncios = db.Anuncios.ToList();

}

else

{

anuncios = db.Anuncios

.Where(x => x.idAnuncio.ToString() == txtBuscar || x.titulo.Contains(txtBuscar))

.ToList();

}

foreach (var anuncio in anuncios)

{

if (anuncio.idImagenPrincipal.HasValue)

{

anuncio.Imagenes = db.Imagenes.Find(anuncio.idImagenPrincipal.Value);

if (anuncio.Imagenes != null)

{

anuncio.ImagenRuta = anuncio.Imagenes.url;

}

}

if (anuncio.idVideoPrincipal.HasValue)

{

anuncio.Videos = db.Videos.Find(anuncio.idVideoPrincipal.Value);

if (anuncio.Videos != null)

{

anuncio.VideoUrl = anuncio.Videos.url;

}

}

}

}

return View(anuncios);

}

}

**Controlador EventosController**

csharp

Copiar código

public class EventosController : Controller

{

private Model1 db = new Model1();

// GET: Eventos

public ActionResult Index()

{

var eventos = db.Eventos.Include(e => e.Categoria).ToList();

return View(eventos);

}

// GET: Eventos/CrearEvento

public ActionResult CrearEvento()

{

ViewBag.ddlCategorias = new SelectList(db.Categorias, "idCategoria", "nombre");

return View();

}

// POST: Eventos/CrearEvento

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult CrearEvento([Bind(Include = "idEvento,nombreEvento,descripcion,fechaEvento,lugar,organizador,estado,capacidad,idCategoria")] Eventos evento, HttpPostedFileBase fuImagenFile)

{

if (ModelState.IsValid)

{

if (fuImagenFile != null && fuImagenFile.ContentLength > 0)

{

using (var reader = new System.IO.BinaryReader(fuImagenFile.InputStream))

{

evento.Imagen = reader.ReadBytes(fuImagenFile.ContentLength);

}

}

db.Eventos.Add(evento);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.ddlCategorias = new SelectList(db.Categorias, "idCategoria", "nombre", evento.idCategoria);

return View(evento);

}

// GET: Eventos/DetallesEvento/5

public ActionResult DetallesEvento(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Eventos evento = db.Eventos.Find(id);

if (evento == null)

{

return HttpNotFound();

}

return View(evento);

}

}

## Tipo de datos

|  |  |
| --- | --- |
| **Tipo de variable** | **Descripción** |
| Byte | Entero de 8 bits sin signo. |
| Integer | Entero de 32 bits con signo. |
| Char | Un carácter UNICODE de 16 bits |
| String | Cadena de caracteres |
| Date | Formato de fecha/hora |
| Boolean | Valor lógico: verdadero y falso |
| Float | Coma flotante, 11-12 dígitos significativos. |
| Double | Coma flotante, 64 bits (15-16 dígitos significativos) |
| Object | Objeto genérico |

## Prefijo para el Control

No se harán uso de controles

## Nombre descriptivo del Control

|  |  |
| --- | --- |
| **Clase** | **Descripción** |
| AnuncioController | Controlador para gestionar los anuncios. |
| EventoController | Controlador para gestionar los eventos. |
| UsuarioController | Controlador para gestionar los usuarios. |
| ProgramacionController | Controlador para gestionar la programación. |
| Anuncios | Modelo que representa los anuncios. |
| Eventos | Modelo que representa los eventos. |
| Usuarios | Modelo que representa los usuarios. |
| ProgramacionSemanal | Modelo que representa la programación semanal. |
| Model1 | Contexto de la base de datos. |
| Categorias | Modelo que representa las categorías. |
| Imagenes | Modelo que representa las imágenes. |
| Videos | Modelo que representa los videos. |
| EnlacesRelacionados | Modelo que representa los enlaces relacionados. |

## Declaración de variables, atributos y objetos

* + 1. Se debe declarar una variable por línea.

|  |  |
| --- | --- |
| **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). |
| 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. |
| Ejemplo | public int idAnuncio: ID del anuncio. |
|  | public String titulo: Título del anuncio. |
|  | public Date fechaEvento: Fecha del evento. |
|  | private Model1 db: Contexto de la base de datos. |
|  | private SelectList ddlCategorias: Lista desplegable de categorías. |

|  |  |
| --- | --- |
|  | método será correlativo). |
| **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. |
| **Ejemplo** | public int idBeca: ID de la beca.  public int idCurso: ID del curso.  public int idDetalle: ID del detalle.  public int idMatricula: ID de la matrícula.  public int idProntoPago: ID del descuento por pronto pago.  public Date timestamp: Marca de tiempo del mensaje.  public String level: Nivel del mensaje.  public String message: Mensaje en sí.  private final DetalleService detalleService: Servicio para gestionar los detalles de matrícula.  private final MatriculaService matriculaService: Servicio para gestionar las matrículas.  private final DetalleRepository detalleRepository: Repositorio para acceder a los datos de los detalles de matrícula.  private final MatriculaRepository matriculaRepository: Repositorio para acceder a los datos de las matrículas.  private int fkIdMatricula: ID de la matrícula asociada al detalle.  private Curso curso: Curso del detalle.  private BigDecimal subTotal: Subtotal del detalle.  private Usuario usuario: Usuario asociado a la matrícula.  private Semestre semestre: Semestre de la matrícula.  private ProntoPago prontoPago: Descuento por pronto pago de la matrícula.  private Beca beca: Beca de la matrícula.  private LocalDate fecha: Fecha de la matrícula.  private Estado estado: Estado de la matrícula.  private LocalDate vencimiento: Fecha de vencimiento de la matrícula.  private BigDecimal total: Total de la matrícula. |

## 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. |
| 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. |
| Ejemplo | public class AnuncioController: Controlador para gestionar los anuncios. |
|  | public class Eventos: Representa un evento. |
|  | public class Model1: Contexto de la base de datos. |
|  | public class ProgramacionSemanal: Representa la programación semanal. |

## Declaración de métodos

|  |  |
| --- | --- |
| **Título** | **Descripción** |
| Sintaxis | [Tipo] [nombredelmetodo] [(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 | public List<Anuncios> listarAnuncios(): Obtiene todos los anuncios. |
|  | public Anuncios getAnuncio(int idAnuncio): Obtiene un anuncio por su ID. |
|  | public void crearAnuncio(Anuncios anuncio): Crea un nuevo anuncio. |
|  | public void actualizarAnuncio(Anuncios anuncio): Actualiza un anuncio existente. |
|  | public void eliminarAnuncio(int idAnuncio): Elimina un anuncio. |

## Declaración de funciones

No se hicieron uso de funciones

## 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. |
| Descripción | Se generarán archivos con las siguientes extensiones: .zip o .rar. Por ejemplo: RadioConexionLatam\_20240701\_1656.zip |

# Clases.

Las clases en el proyecto **RadioConexionLatam** deben seguir las siguientes reglas para asegurar claridad y coherencia:

* **Nombres Auto Descriptivos**: Utiliza nombres que indiquen claramente el propósito y la responsabilidad de la clase.
* **Capas de la Aplicación**: Si tu aplicación tiene diferentes capas, agrega un prefijo o sufijo para indicar a qué capa pertenece la clase.
* **No Usar Caracteres Especiales**: No se debe utilizar espacios en blanco " " ni caracteres de subrayado "\_".

# Métodos, Procedimientos y Funciones definidos por el Usuario.

Para los métodos, procedimientos y funciones, utiliza nombres que representen una acción y un objeto. El nombre debe indicar qué hace la acción sobre el objeto o qué hace la acción con el objeto.

* **Verbo en Infinitivo**: El nombre debe comenzar con un verbo en infinitivo.
* **VerboSustantivo**: Usa una combinación consistente de verbo y sustantivo.
* **No Usar Caracteres Especiales**: No se debe utilizar espacios en blanco " " ni caracteres de subrayado "\_".

|  |  |
| --- | --- |
| **Método** | **Descripción** |
| crearUsuario | Crea un nuevo usuario en el sistema. |
| obtenerDetalleById | Obtiene un detalle específico según su ID. |
| guardarMatricula | Guarda una matrícula en el sistema. |
| actualizarCurso | Actualiza la información de un curso existente. |
| calcularSubTotal | Calcula el subtotal de un detalle de transacción. |
| eliminarUsuario | Elimina un usuario del sistema. |
| generarReporte | Genera un reporte basado en ciertos datos. |
| validarFecha | Valida si una fecha es válida o no. |
| asignarBeca | Asigna una beca a un estudiante. |
| buscarDetallePorCurso | Busca detalles de transacción asociados a un curso específico. |

# Beneficios

 **Legibilidad**: La documentación hace más legible un programa.

 **Facilita Modificaciones**: Al documentar bien un programa desde el principio, se evita que para cada modificación deba estudiarse profundamente el funcionamiento del programa, redescubriendo todo lo no documentado.

 **Reutilización**: Facilita la reutilización de módulos y rutinas desde cualquier otro programa o el mismo.

 **Mejora la Calidad del Código**: 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

 **Estándares y Patrones**: Una buena programación e implementación legible solo se logra usando y llevando de la mano un buen estándar o patrón de programación.

 **Conocimiento del Estándar**: Es muy importante que el programador tenga un conocimiento previo del estándar o en su defecto que lea el documento para prever diferencias.

 **Documentación**: Al documentar se obtienen dos cosas fundamentales, un documento legible y una buena base para los futuros desarrollos de mantenimiento del código.