

**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERÍA**

**Escuela Profesional de Ingeniería de Sistemas**

***Estándares de Programación***

**Proyecto: Sistema Web PetGuard - Gestión y Cuidado Inteligente de Mascotas**

Curso: Programación III

Docente: Ing. Juan Manuel Choque Flores

Docente: Ing. Elard Rodriguez Marca

Integrantes:

**Rosas Chambilla, Jefferson (2021072618)**

**Challo Coaquera Alexander wilson (2017057578)**

**Rodriguez Cardenas, Patrick Elvis (2022075751)**

**Tacna – Perú**

***2025***

**Sistema Web PetGuard - Gestión y Cuidado Inteligente de Mascotas**

**Documento de Estándares de Programación**

**Versión 1.0**

**Historia de Revisión**

| **Historial de revisiones** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Ítem** | **Fecha** | **Versión** | **Descripción** | **Equipo** | |
| 1 | 20/05/2025 | 1.0 | Versión Inicical. | 3 | |
| 2 | 20/06/2025 | 2.0 | Revision con estandares Formal | Challo/Rosas/ | |

ÍNDICE GENERAL

[Introducción 3](#_heading=h.5kqeh1b3ip7o)

[Estándares de Nomenclatura 4](#_heading=h.s7efyxmkxskd)

[2.1. Nomenclatura de Variables y Métodos (Java) 4](#_heading=h.gs45pgwvbu19)

[2.2. Nomenclatura de Clases 4](#_heading=h.w61c3meoc1z6)

[2.3. Nomenclatura de Paquetes 5](#_heading=h.k20u2ois1k33)

[Estándares de Base de Datos 5](#_heading=h.5eq57l8139y0)

[3.1 Nomenclatura de Tablas 6](#_heading=h.bwp70fk06kua)

[3.2 Nomenclatura de Campos 6](#_heading=h.sosczstsbg0o)

[3.3 Tipos de Datos Estándar 6](#_heading=h.u8527z8ae9zs)

[3.4 Restricciones y Validaciones 7](#_heading=h.tu558gcwk5o6)

[Estándares de Codificación Java 7](#_heading=h.4hjz24urjufz)

[4.1 Estructura de Clases 7](#_heading=h.1wbsw5wy44j4)

[4.2 Manejo de Excepciones 8](#_heading=h.vii90kxz765u)

[4.3 Validaciones Estándar 8](#_heading=h.m9issigryihx)

[4.4 Métodos de Conversión 9](#_heading=h.lxh5mbfn6x0f)

[Estándares de Documentación 10](#_heading=h.6rzfvvfpxsa)

[5.1 Comentarios de Clase 10](#_heading=h.t50npwex8zhv)

[5.2 Comentarios de Métodos 10](#_heading=h.rhi74aqmtr8b)

[5.3 Comentarios en Línea 12](#_heading=h.nzaof4qqfl5m)

[Estándares de Estructura de Archivos 12](#_heading=h.7avdm5bnju09)

[6.1 Organización de Paquetes 12](#_heading=h.q0gpcnunylv)

[6.2 Nomenclatura de Archivos 13](#_heading=h.2zgehaifhzjn)

[6.3 Archivos de Configuración 13](#_heading=h.l534j3v0522h)

[Buenas Prácticas 13](#_heading=h.tqxmh56qgbfw)

[7.1 Gestión de Estados 13](#_heading=h.5zmxq0h4x9pn)

[7.2 Validación de Datos 13](#_heading=h.mom6kym890l8)

[7.3 Manejo de Fechas 15](#_heading=h.c3rvkxhfm1dt)

[7.4 Conexión a Base de Datos 15](#_heading=h.523gfvh5lvl5)

[7.5 Logging y Mensajes 16](#_heading=h.rrz80y50jjq4)

[7.6 Integración MongoDB 16](#_heading=h.5gyy4eep289t)

## Introducción

Este documento establece los estándares de programación para el proyecto **PetGuard**, un sistema de gestión veterinaria que permite administrar mascotas, propietarios, historiales médicos, observaciones y recomendaciones. El cumplimiento de estos estándares garantiza la **consistencia, mantenibilidad y calidad del código fuente**, tanto en aplicaciones de escritorio como en entorno web.

## Estándares de Nomenclatura

### 2.1. Nomenclatura de Variables y Métodos (Java)

**Convención:** camelCase

* Variables: nombreMascota, dniPropietario, fechaRegistro
* Métodos: registrarMascota(), obtenerHistorialMedico(), actualizarEstado()

**Ejemplos correctos:**

****String nombreCompleto;

int idMascota;

Date fechaNacimiento;

boolean estadoActivo;

public void registrarPropietario() { }

public String obtenerNombreMascota() { }

### 

### 2.2. Nomenclatura de Clases

**Convención:** PascalCase

Las clases deben ser descriptivas, sin abreviaturas y representando entidades o funcionalidades específicas.

* ControladorPropietario
* ControladorMascota
* HistorialMedicoDAO
* ChatDAO
* VeterinarioDAO
* ConexionBD

### 2.3. Nomenclatura de Paquetes

**Convención:** Todo en **minúsculas** y separado por puntos (.), siguiendo la convención estándar de Java.

* petguard.config
* petguard.controlador
* petguard.modelo
* petguard.modelo.dao
* petguard.interfaces
* petguard.utilidades

Esto garantiza compatibilidad con herramientas, IDEs y frameworks modernos.

## Estándares de Base de Datos

### 3.1 Nomenclatura de Tablas

**Convención recomendada (MySQL):** Utilizar **snake\_case** en minúsculas para todas las tablas. Se puede usar el prefijo tb\_ para indicar tabla si se desea mantener una estructura clara.

* 🔹 **Formato recomendado:** tb\_nombre\_tabla
* 🔹 **Formato prohibido:** tbNombreTabla (uso de PascalCase en SQL es inconsistente)

**Ejemplos de tablas MySQL del proyecto:**

* ✅ tb\_mascota
* ✅ tb\_historial\_medico
* ✅ tb\_propietario
* ✅ tb\_veterinario
* ✅ tb\_observacion
* ✅ tb\_recomendacion

#### 3.1.1 Nomenclatura de Colecciones MongoDB (si aplica)

**Convención recomendada (MongoDB):**

* En minúsculas, **sin prefijo tb\_**, nombre en plural si representa múltiples documentos.
* No se recomienda usar guiones bajos si no son necesarios.

**Ejemplos en PetGuard:**

* ✅ chats
* ✅ messages
* ✅ observaciones
* ✅ recomendaciones

### 3.2 Nomenclatura de Campos

**Convención recomendada:**

Se debe utilizar exclusivamente **snake\_case** para todos los campos en la base de datos relacional (MySQL), en minúsculas y sin tildes ni caracteres especiales.

**Campos utilizados en el proyecto:**

| **Tipo de campo** | **Ejemplos** | **Observaciones** |
| --- | --- | --- |
| Claves primarias | id, dni | Usar nombres cortos y únicos. |
| Claves foráneas | id\_mascota, dni\_propietario, id\_historial | Prefijadas según la tabla de referencia. |
| Campos descriptivos | nombre, apellido, telefono, direccion, correo, tipo\_evento, tratamiento\_aplicado | Se recomienda longitud estándar (VARCHAR).    Si el nombre del campo es más largo se usa “\_” para separar las palabras. |
| Campos de estado | estado | Usar valores controlados: 'activo', 'inactivo' |
| Campos de fecha | fecha, proxima\_cita, fecha\_registro | Siempre con tipo DATE o DATETIME según el caso. |

### 3.3 Tipos de Datos Estándar

A continuación se presentan los tipos de datos estándar utilizados en el sistema PetGuard para la tabla tb\_propietario. Se sigue una estructura clara con uso de snake\_case, tipos de datos consistentes y tamaños recomendados.

tbPropietario

| **Campo** | **Tipo de Dato** | **Longitud** | **Ejemplo** | **Observaciones** |
| --- | --- | --- | --- | --- |
| dni | VARCHAR | 8 | '45671238' | Clave primaria |
| nombre | VARCHAR | 50 | 'Maria' | Texto sin tildes |
| apellido | VARCHAR | 50 | 'Gonzales' | Texto sin tildes |
| telefono | VARCHAR | 9 | '998877667' | Se valida con regex |
| direccion | VARCHAR | 150 | 'Jr. Las Flores 202' | Sin tildes, solo texto |
| correo | VARCHAR | 100 | 'maria.gonzales@yahoo.com' | Requiere validación de formato |
| contra | VARCHAR | 100 | 'maria123' | Contraseña en texto plano (recomendar hash) |
| estado | VARCHAR | 10 | 'activo' | 'activo' por defecto, 'inactivo' manual |
| foto | VARCHAR | 200 | 'https://i.ibb.co/.../img.png' | URL completa de imagen |

### 3.4 Restricciones y Validaciones

Las siguientes restricciones se aplican a nivel de base de datos para asegurar integridad referencial, unicidad y consistencia en los datos registrados en el sistema PetGuard:

| **Tipo de Restricción** | **Descripción** | **Ejemplo aplicado** |
| --- | --- | --- |
| **Clave primaria** | Todo registro debe tener una clave primaria única que lo identifique. | PRIMARY KEY (id) |
| **Clave foránea** | Establece la relación entre dos tablas y asegura la integridad referencial. | FOREIGN KEY (dni\_propietario) REFERENCES tb\_propietario(dni) |
| **Campo único** | Campos sensibles como teléfono y correo deben ser únicos para evitar duplicidad. | UNIQUE (telefono), UNIQUE (correo) |
| **Valor por defecto** | Algunos campos como estado se inicializan automáticamente si no se especifica valor. | estado VARCHAR(10) DEFAULT 'activo' |
| **Autoincremento** | Identificadores numéricos son generados automáticamente. | id INT AUTO\_INCREMENT PRIMARY KEY |
| **Restricciones lógicas** | Validación a nivel de aplicación para garantizar formatos válidos (DNI, correo, teléfono, etc.) | Validación con REGEX en backend y frontend |

## Estándares de Codificación Java

### 4.1 Estructura de Clases

Se recomienda el siguiente orden lógico para la declaración de clases:

****public class MascotaController {

// 1. Constantes

private static final String ESTADO\_ACTIVO = "activo";

// 2. Variables de instancia

private MascotaService mascotaService;

private String nombreClase;

// 3. Constructor

public MascotaController() {

this.mascotaService = new MascotaService();

}

// 4. Métodos públicos

public void registrarMascota(Mascota mascota) {

// Implementación

}

// 5. Métodos privados

private boolean validarDatos(Mascota mascota) {

// Implementación

return true;

}

}

### 4.2 Manejo de Excepciones

**Estándar obligatorio:** El uso de try-catch es obligatorio en operaciones críticas, como validaciones y conexiones con base de datos.

public boolean registrarPropietario(Propietario propietario) {

try {

// Validaciones

if (propietario.getDni() == null || propietario.getDni().isEmpty()) {

throw new IllegalArgumentException("DNI es requerido");

}

// Operación de base de datos

propietarioDAO.insertar(propietario);

return true;

} catch (SQLException e) {

System.err.println("Error en base de datos: " + e.getMessage());

return false;

} catch (IllegalArgumentException e) {

System.err.println("Error de validación: " + e.getMessage());

return false;

}

}

### 

### 4.3 Validaciones Estándar

****// Validación de DNI

public boolean validarDNI(String dni) {

return dni != null && dni.matches("\\d{8}");

}

// Validación de teléfono

public boolean validarTelefono(String telefono) {

return telefono != null && telefono.matches("\\d{9}");

}

// Validación de correo

public boolean validarCorreo(String correo) {

return correo != null && correo.contains("@") && correo.contains(".");

}

### 4.4 Métodos de Conversión

****// Conversión de String a Integer

public int convertirAEntero(String valor) {

try {

return Integer.parseInt(valor);

} catch (NumberFormatException e) {

return 0; // Valor por defecto

}

}

// Conversión de tipos para mostrar

public String convertirATexto(Object valor) {

return String.valueOf(valor);

}

## Estándares de Documentación

### 5.1 Comentarios de Clase

****/\*\*

@WebServlet(name = "ControladorMascota", urlPatterns = {"/ControladorMascota"})

public class ControladorMascota extends HttpServlet {

String listar = "Vista/VistaMascota/listar.jsp";

String add = "Vista/VistaMascota/agregar.jsp";

String edit = "Vista/VistaMascota/editar.jsp";

Mascota pet = new Mascota();

MascotaDAO dao = new MascotaDAO();

\*/

public class MascotaController {

// Implementación

}

### 5.2 Comentarios de Métodos

****/\*\*En ChatDAO.java

// Insertar un chat

// Buscar un chat por ID

// Listar todos los chats de un propietario

// Listar todos los chats de un veterinario

// Puedes agregar más métodos según necesidad (actualizar, eliminar, etc.)

/\*\*En HistorialMedicoDAO.java

// Se elimina el campo "veterinario" del INSERT y del VALUES

// Se elimina el campo "veterinario" del UPDATE

// Se obtiene el nombre del propietario asociando por dni\_propietario

// No cierres la conexión ni el PreparedStatement aquí porque el ResultSet se usará fuera de este método

// Se elimina "veterinario" del SELECT

/\*\*En MascotaDAO.java

// Construcción de la sentencia SQL

// ID es obligatorio para el WHERE

// Valor predeterminado si no se encuentra ninguna fila

// Consulta para obtener el id máximo

// Establece la conexión

// Prepara la sentencia SQL

// Ejecuta la consulta

// Obtiene el valor de la última id

// Devuelve el valor de la última id (o -1 si no se encuentra ninguna fila)

/\*\*En MessageDAO.java

// Insertar un mensaje

// Listar mensajes de un chat

// Puedes agregar más métodos según necesidad (buscar por sender, eliminar mensaje, etc.)

/\*\*En ObservacionDAO.java

// Insertar una observación

// Listar observaciones por id\_historial

// Actualizar una observación por \_id

// Eliminación lógica (cambia estado a 'inactivo')

/\*\*En PropietarioDAO.java

// Login correcto

// Obtener la conexión

// Consulta SQL para actualizar los datos del propietario, ahora incluye la columna foto

// Preparar la sentencia SQL

// Ejecutar la actualización

// Verificar si se afectaron filas (lo que significa que se actualizó correctamente)

// Cerrar recursos

/\*\*En RecomendacionDAO.java

// Insertar una recomendación

// Listar recomendaciones por id\_historial y estado "activo"

// Ahora el filtro exige que el estado sea "activo"

// Actualizar una recomendación (por \_id)

// Eliminar (lógico: cambia estado) una recomendación

// Puedes agregar métodos según necesites (buscar por \_id, etc.)

/\*\*En VeterinarioDAO.java

### 5.3 Comentarios en Línea

****// Validar que el DNI tenga exactamente 8 dígitos

if (!dni.matches("\\d{8}")) {

return false;

}

// Establecer estado activo por defecto

/\*mascota.setEstado("activo");\*/

## Estándares de Estructura de Archivos

### 6.1 Organización de Paquetes

****Web Pages/

├──

│ ├── META-INF/

│ ├── Vista/

│ ├── WEB-INF/

│ ├── css/

│ ├── img/

│ └── index.jsp/

│ ├── principal.jsp/

│ ├── validarCaptcha.jsp/

│ └── validarLogin.jsp/

Remote Files/

├── html2pdf.bundle.min.js/

Source Packages/

├──

│ ├── Config/

│ ├── Controlador/

│ ├── Interfaces/

│ ├── Modelo/

│ ├── ModeloDAO/

### 6.2 Nomenclatura de Archivos

* **Controladores:** ControladorPropietario.java
* **Servicios:** MascotaService.java
* **DAOs:** HistorialMedicoDAO.java
* **Entidades:** Veterinario.java
* **Vistas:** RegistroMascotaFrame.java

### 6.3 Archivos de Configuración

* **Base de datos:** database.properties
* **Configuración:** config.properties
* **Scripts SQL:** schema.sql, data.sql

## Buenas Prácticas

### 7.1 Gestión de Estados

* Siempre usar constantes para estados: "activo"(predeterminado),
* Validar estados antes de operaciones críticas

### 7.2 Validación de Datos

****// Validación completa de Veterinario

public boolean validarLogin(String dni, String clave) {

boolean acceso = false;

String sql = "SELECT \* FROM tbVeterinario WHERE dni = ? AND contra = ? AND estado = 'activo'";

try {

con = cn.getConnection();

ps = con.prepareStatement(sql);

ps.setString(1, dni);

ps.setString(2, clave);

rs = ps.executeQuery();

if (rs.next()) {

acceso = true;

}

} catch (SQLException e) {

System.out.println("Error en login: " + e.getMessage());

} finally {

try { if (rs != null) rs.close(); } catch (SQLException e) {}

try { if (ps != null) ps.close(); } catch (SQLException e) {}

try { if (con != null) con.close(); } catch (SQLException e) {}

}

return acceso;

}

// Validación completa de Propietario

public boolean validarLogin(String dni, String clave) {

boolean acceso = false;

String sql = "SELECT \* FROM tbPropietario WHERE dni = ? AND contra = ? AND estado = 'activo'";

try {

con = cn.getConnection();

ps = con.prepareStatement(sql);

ps.setString(1, dni);

ps.setString(2, clave);

rs = ps.executeQuery();

if (rs.next()) {

acceso = true; // Login correcto

}

} catch (SQLException e) {

System.out.println("Error en login: " + e.getMessage());

} finally {

try {

if (rs != null) rs.close();

if (ps != null) ps.close();

if (con != null) con.close();

} catch (SQLException e) {

System.out.println("Error al cerrar recursos: " + e.getMessage());

}

}

return acceso;

}

### 7.3 Manejo de Fechas

****// Usar Date para fechas con Timestamp

Timestamp proximaCitaTS = rs.getTimestamp("proxima\_cita");

msg.setTimestamp(doc.getString("timestamp"));

// Usar DateTime para fechas con hora

DateTime proximaCita = new DateTime();

// Formateo estándar

SimpleDateFormat formato = new SimpleDateFormat("yyyy-MM-dd");

String fechaFormateada = formato.format(fecha);

### 

### 7.4 Conexión a Base de Datos

****public class clsConexion {

Connection con;

public Connection getConnection() {

try {

// Establece la conexión a la base de datos MySQL en la nube

Class.forName("com.mysql.cj.jdbc.Driver");

con = DriverManager.getConnection(

"jdbc:mysql://sql.freedb.tech:3306/freedb\_petguardbd",

"freedb\_Rofferson21",

"F67##yVuPm4CT&&"

);

} catch (SQLException | ClassNotFoundException e) {

// En caso de error, muestra el mensaje

System.out.println("Error en la conexión: " + e.getMessage());

}

return con;

}

}

### 7.5 Logging y Mensajes

****// Mensajes informativos

System.out.println("Mascota registrada exitosamente: " + mascota.getNombre());

// Mensajes de error

System.err.println("Error al registrar mascota: " + e.getMessage());

// Debug (opcional)

if (DEBUG\_MODE) {

System.out.println("DEBUG: Validando datos de mascota...");

}

### 7.6 Integración MongoDB

Para las colecciones de MongoDB (observaciones y recomendaciones):

// Nomenclatura de colecciones

// PetGuard.observaciones

// PetGuard.recomendaciones

// Estructura de documentos JSON consistente

{

"\_id": '684c442aafc4b4b833261560',

"id\_historial": 1,

"dni\_propietario": "12345678",

"dni\_veterinario": "87654321",

"observacion": "La mascota presentó mejoría notable...",

"fecha": 2025-06-13T15:00:00.000+00:00 ISODate,

"estado": "activo"

}



**8. 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.