ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA EN SISTEMAS
PROGRAMACION I
PROYECTO FINAL
GRUPO 5

Manual de Estándares: Sistema de Prueba Psicosensométrica GR5

1. Introducción y Principios Generales

Este manual define los estándares de codificación para todo el proyecto "GR5_ExamenPsicosensometrico", abarcando tanto el software Java (aplicación de escritorio) como el firmware de Arduino (C++) al seguir estas pautas asegura la consistencia, legibilidad, mantenibilidad y eficiencia del código, facilitando el trabajo colaborativo y la depuración en ambos entornos.

Principio General (Prefijo GR5_):
Todos los nombres de clases, métodos, variables, constantes, estructuras y componentes visuales principales deben comenzar con el prefijo GR5_. Esto identifica claramente los elementos propios del proyecto y evita conflictos de nombres.

Excepciones: Variables locales de bucles, variables temporales muy cortas, y componentes generados automáticamente por el IDE (NetBeans para Java) pueden omitir el prefijo si su alcance es muy limitado y su propósito es obvio.

2. Nomenclatura de Elementos'

La consistencia en los nombres es crucial para la legibilidad en ambos lenguajes.

Elemento	Estándar General	Aplicación Java	Firmware Arduino (C++)
Clases/Estructuras	GR5_NombreClase (Camel Case, con prefijo GR5_)	public class GR5_DBConnexion public class GR5_Inicio	struct GR5_Configuracion
Constantes	GR5_NOMBRE_CONSTANTE (MAYÚSCULAS con guiones bajos, final en Java, const en C++, con prefijo GR5_)	private static final String GR5_DB_URL	const int GR5_PIN_LED = 8;
Variables de Instancia	GR5_nombreVariable (camelCase, con prefijo GR5_)	private DefaultTableModel GR5_modeloTabla;	GR5_Configuracion GR5_configActual;
Variables Estáticas/Globales	GR5_nombreVariable (camelCase, con prefijo GR5_)	public static int GR5_usuarioID; (en GR5_Sesion.java)	bool GR5_pruebaActiva = false;
Variables Locales	nombreVariable (camelCase). Opcionalmente	String nombre = txtGR5Nombre.getText();	bool botonPresionado;

	GR5_nombreVariable si son muy importantes o su nombre es ambiguo.		
Funciones/Métodos	GR5_NombreMetodo() (Camel Case, con prefijo GR5_. Verbos al inicio para indicar acción.)	void GR5_cargarUsuarios() boolean GR5_camposValidos()	void GR5_LeerConfiguracion() void GR5_ActivarEstimulos()
Parámetros	nombreParametro (camelCase, sin prefijo GR5_ a menos que sea necesario para evitar ambigüedad).	boolean GR5_validar(String cedula)	void GR5_ActivarEstimulos(bool activarLed, int intensidadLed)
Manejadores de Eventos	Se mantiene la nomenclatura generada automáticamente por el IDE.	btnGR5AgregarActionPerformed tblGR5UsuariosMouseClicked	*No aplica directamente en Arduino, se gestionan en loop() o funciones.*

Inicialización de Variables: Inicializar las variables al momento de su declaración siempre que sea posible, para evitar errores o valores inesperados.

3. Estructura y Organización del Código

El código debe estar bien estructurado y modularizado.

Java (Paquetes y Clases):

- **Paquetes:** El código de la aplicación Java se organiza bajo un paquete principal (ej. app). Si el proyecto creciera, se considerarían subpaquetes (ej. app.model, app.view, app.controller).
- **Clases:** Cada archivo .java debe contener una única clase principal con el mismo nombre que el archivo.

Archivos. form (NetBeans): Estos archivos son generados automáticamente por el IDE y describen el diseño de las interfaces gráficas. No deben ser modificados manualmente si se utiliza el diseñador visual de NetBeans.

Arduino (Estructura de Archivo .ino / .cpp):

Orden Lógico:

Encabezado: Comentarios iniciales del archivo.
Inclusión de Librerías: \#include <Libreria.h>
Definición de Constantes: const int PIN_X = Y;
Definición de Estructuras: struct MiEstructura { ... };

Variables Globales: Declaración de variables de estado compartidas (limitar su uso).
Instancias de Objetos Globales: SoftwareSerial miSerial(...)
setup(): Configuración inicial (pines, Serial.begin()).
loop(): Bucle principal de ejecución. Debe ser lo más "limpio" posible, delegando tareas a funciones.
Implementación de Funciones: Agrupar funciones por su lógica (ej. comunicación serial, lógica de prueba, auxiliares).

Ejemplo de Bloque de Funciones en Arduino:C++

```
// --- Funciones de Comunicación Serial ---
void GR5_LeerComandoSerial() {
  // ...
}

/**
 * Lee la configuración de la prueba enviada desde Java.
 * @return true si la configuración fue leída exitosamente.
 */
void GR5_LeerConfiguracion() {
  String configStr = Serial.readStringUntil('\n'); // Lectura con delimitador
  // Parsear configStr y actualizar GR5_configActual
  Serial.println("OK"); // Confirmar recepción a Java
}

// --- Funciones de Lógica de Prueba ---
void GR5_iniciarNuevaPrueba() {
  GR5_pruebaActiva = true;
  // ...
}
```

4. Comentarios y Documentación

Los comentarios son vitales para la comprensión del código a largo plazo.
Encabezado de Archivo:
Todo archivo (Java y Arduino) debe comenzar con un encabezado de comentario (/* ... */) que incluya: Nombre del proyecto, Versión, Autor(es), Descripción breve del archivo/clase.

Ejemplo (Arduino):
C++

```
/* ========================================================
 * PROYECTO: GR5_ExamenPsicosensometrico
 * VERSIÓN: 2.1
 * AUTOR: [Grupo 5: Peña Erick, Pinos Abrahan, Wilman Perugachi]
 * DESCRIPCIÓN: Firmware para la gestión de estímulos y respuestas.
 * ========================================================
 */
```

Comentarios Javadoc (Java) / Bloque (Arduino):

- **Java:** Todos los métodos public y protected, así como los private complejos, deben tener comentarios Javadoc (/** ... */) describiendo propósito, @params, @return y @throws.
- **Arduino:** Usar comentarios de bloque (/** ... */) para funciones/métodos complejos, explicando su propósito, parámetros (@param) y lo que hace.

Ejemplo (Arduino):

```
C++
/**
 * Activa los estímulos físicos (LED y Buzzer) según los parámetros.
 * @param activarLed TRUE si el LED debe encenderse.
 * @param activarBuzzer TRUE si el Buzzer debe sonar.
 * @param intensidadLed Nivel de intensidad para el LED (1-3).
 * @param intensidadBuzzer Nivel de intensidad para el Buzzer (1-3).
 */
void GR5_ActivarEstimulos(bool activarLed, bool activarBuzzer, int intensidadLed, int
intensidadBuzzer) {
  // ...
}
```

Comentarios de Línea (//):

Utilizar para explicar líneas específicas o ideas complejas.
Regla: Deben explicar por qué se hace algo, no solo qué se hace (asumiendo que el código es legible).
Mantener los comentarios actualizados con los cambios en el código.

5. Formato y Estilo del Código

Indentación:

Usar un estilo de indentación consistente

Espacios en Blanco:

Dejar un espacio después de las comas en las listas de argumentos.
Dejar un espacio alrededor de los operadores (=, +, ==, &&, ||, etc.).
No espacios entre el nombre de la función y el paréntesis (miFuncion()).
Dejar una línea en blanco entre métodos y entre bloques lógicos importantes dentro de los métodos.

Llaves ({}):

Usar un estilo de llaves consistente. Se recomienda la llave de apertura en la misma línea que la declaración de la clase/método/bloque, y la llave de cierre en su propia línea.

Ejemplo (Java/C++):

```
Java
public void miMetodo() {
    // ...
}
if (condicion) {
```

```
    // ...
}
```

Longitud de Línea:
Intentar mantener las líneas de código por debajo de un límite razonable (ej. 120 caracteres) para mejorar la legibilidad, dividiendo líneas largas si es necesario.