UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Julio Eduardo González Constante 2067567
Daniel Alejandro Barbosa Ramírez 2173469
Sebastián Flores González 2104564

Laboratorio de Programación Orientada a Objetos

PIA: Manual Tecnico "GYM POWERHOUSE JDS"

Profesor José Anastacio Hernández Saldaña

6 de diciembre del 2025

MANUAL TÉCNICO DEL SISTEMA

“GYM POWER HOUSE JDS”

1. INTRODUCCIÓN

El presente manual técnico describe la arquitectura, componentes, requisitos, instalación, estructura interna y funcionamiento del sistema, una aplicación desarrollada en Java 25, con uso de archivos.

El sistema permite la administración de clientes dentro de un gimnasio, proporcionando operaciones de registro, consulta, actualización y eliminación, ya sean de clientes, empleados, actividades dentro del gimnasio y productos que se venden dentro del gimnasio.

El objetivo de este documento es proporcionar la información técnica necesaria para desarrolladores, administradores del sistema y personal de soporte, facilitando modificaciones, despliegues y mantenimiento del software.

2. REQUISITOS TÉCNICOS

Software necesario

Para compilar, ejecutar o dar mantenimiento al proyecto se requiere:

Java Development Kit (JDK) 25

Cómo descargar e instalar JDK 25

Entrar al sitio oficial de Oracle

Ve a: https://www.oracle.com/java/technologies/downloads/

(Es la fuente oficial y segura para descargar Java 25)

Elegir tu sistema operativo

En la página verás opciones como:

- Windows

- Linux

- macOS

Selecciona la que corresponda a tu computadora.

Descargar el instalador

En Windows: selecciona el archivo .exe

En macOS: selecciona el archivo .dmg

En Linux: puedes elegir paquetes .rpm, .deb o archivos tar.gz

Haz clic en Download.

Aceptar la licencia

Oracle te pide marcar un cuadro que dice que aceptas el acuerdo de licencia.

Instalar

En Windows: doble clic en el instalador → "Next" → "Next" → "Install".

En macOS: abrir el .dmg → instalar.

En Linux: instalar con tu gestor de paquetes o extraer la carpeta.

Configurar variables de entorno (solo Windows)

Java normalmente lo hace automáticamente, pero si no:

Ir a Configuración del sistema → Variables de entorno

Verificar la instalación

Abre la terminal o CMD y escribe:

java --version

Debe mostrar algo como:

java version "25.x.x"

Esto requerido debido a las características modernas del lenguaje

IDE recomendado

IntelliJ IDEA

Cómo descargar IntelliJ IDEA

1. Entrar a la página oficial

Ve a: https://www.jetbrains.com/idea/download/

(Es el sitio oficial y seguro de JetBrains.)

2. Elegir edición

Verás dos versiones:

IntelliJ IDEA Community (GRATIS)

Recomendada para:

- Java
- Spring Boot
- Maven
- Proyectos de escuela

IntelliJ IDEA Ultimate

Incluye más herramientas empresariales

Es de pago

Para estas necesidades: elige COMMUNITY.

3. Elegir tu sistema operativo

Puedes elegir:

- Windows
- macOS
- Linux

Haz clic en el botón Download de tu sistema.

4. Instalar IntelliJ

En Windows:

Abrir el archivo .exe

Next → Next → Install

(Opcional) Marcar:

"Add to PATH"

"Create Desktop Shortcut"

En macOS:

Abrir .dmg

Arrastrar IntelliJ a Applications

En Linux:

Extraer el .tar.gz

Ejecutar /bin/idea.sh

5. Primer inicio

Al abrir IntelliJ:

Acepta los término

Crea un nuevo proyecto o abre el tuyo

Verificación

Para verificar que todo funciona:

Crea un nuevo proyecto Java

Selecciona Java 25

Compila un programa sencillo

Hardware mínimo recomendado:

- Procesador Dual Core 2.0 GHz
- 4 GB de memoria RAM (mínimo)
- 2 GB de espacio en disco para proyecto + dependencias
- Conexión a Internet (para descargar librerías Maven)

Cómo descargar JavaFX

JavaFX no viene incluido en las versiones actuales de Java (a partir de JDK 11), por lo que debe descargarse por separado desde la distribución oficial JavaFX SDK.

1. Descargar JavaFX SDK

Acceder al sitio oficial de Gluon, proveedor de las distribuciones actuales de JavaFX:
https://gluonhq.com/products/javafx/

En la sección Latest Release, seleccionar el paquete correspondiente al sistema operativo donde se desarrollará el proyecto:

Windows (ZIP)

macOS (ZIP)

Linux (TAR.GZ)

Descargar el archivo y descomprimirlo en un directorio
Estructura del JavaFX SDK descargado

Después de la descompresión, se obtiene una estructura similar:

```
javafx-sdk-XX/
 ├─ bin
 ├─ legal/
 ├─ lib/   ← Contiene los módulos .jar de JavaFX
 └─ javafx-src.zip (opcional)
```

El directorio lib/ es el que se utilizará para configurar el proyecto.

Como agregar JavaFX en nuestro proyecto:

En IntelliJ IDEA
Abrir Project Structure → Libraries.
Seleccionar + → Java.
Elegir la carpeta lib del JavaFX SDK descargado.
Aceptar los cambios.

En Run Configuration, agregar parámetros VM:

```
--module-path "ruta/javafx-sdk/lib"
--add-modules javafx.controls,javafx.fxml
```

3. ARQUITECTURA DEL SISTEMA

GymPOSSF564\src\GymPOSSF564

controlador

text

«Controlador»

└── Login

    ├── +initialize()

    ├── +btnIngresar()

    ├── +btnRegistrar()

    └── +cambiarDeEscena()

«Controlador»

└── Registro

    ├── +initialize()

    ├── +registrarUsuario()

    ├── +validarID()

    ├── +validarSalario()

    └── +validarContrasena()

«Controlador»

└── Controller

    ├── +initialize()

    ├── +cambiarContenido()

    ├── +mostrarMenuClientes()

    └── +mostrarMenuEmpleados()

«Controlador»

└── ClientesMenu

    ├── +initialize()

    ├── +mostrarRegistrar()

    ├── +mostrarActualizar()

    ├── +mostrarBuscar()

    ├── +mostrarEliminar()

    └── +mostrarVentas()

«Controlador»

└── EmpleadosMenu

    ├── +initialize()

    ├── +mostrarRegistrar()

    ├── +mostrarActualizar()

    ├── +mostrarBuscar()

    ├── +mostrarEliminar()

    └── +mostrarEntradaSalida()

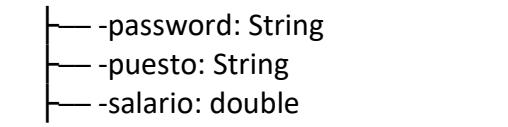
«Controlador»
└── InventarioMenu
    ├── +initialize()
    ├── +mostrarRegistrar()
    ├── +mostrarActualizar()
    ├── +mostrarBuscar()
    ├── +mostrarEliminar()
    └── +mostrarVentas()

«Controlador»
└── Calendario
    ├── +initialize()
    ├── +registrarClase()
    ├── +actualizarClaseSeleccionada()
    └── +eliminarClaseSeleccionada()

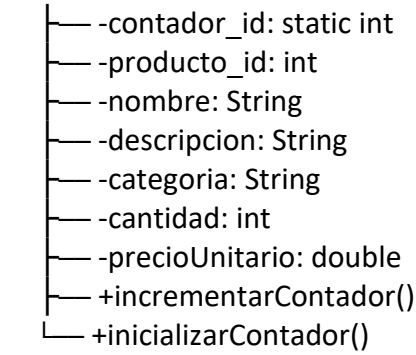
«Controlador»
└── ReportesMenu
    ├── +initialize()
    └── +generarReporte()

modelo
text
«Modelo - Entidad»
└── Cliente
    ├── -contador_id: static int
    ├── -cliente_id: int
    ├── -nombres: String
    ├── -apellidos: String
    ├── -num_telefono: String
    ├── -puntos: int
    ├── +incrementarContador()
    └── +inicializarContador()

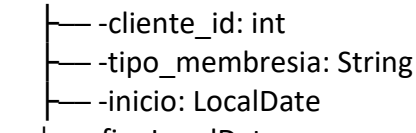
«Modelo - Entidad»
└── Empleado
    ├── -nombre: String
    ├── -apellido: String
    ├── -edad: int
    ├── -direccion: String
    ├── -id: int



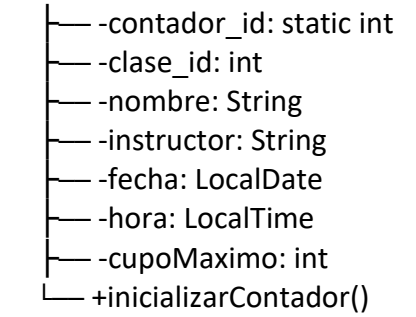
«Modelo - Entidad»
└── Inventario



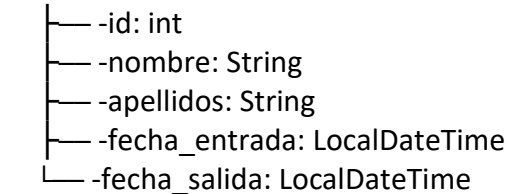
«Modelo - Entidad»
└── Membresia



«Modelo - Entidad»
└── ClaseGrupal



«Modelo - Entidad»
└── Acceso



«Modelo - Gestión»
└── GestionClientesFlores

```
    ├── +cargarClientes()
    ├── +serializarCliente()
    ├── +agregarClientes()
    ├── +actualizarCliente()
    └── +buscarCliente()



«Modelo - Gestión»
└── GestionEmpleadosFlores
    ├── +cargarEmpleado()
    ├── +serializarEmpleado()
    ├── +agregarEmpleado()
    └── +actualizarEmpleado()

«Modelo - Gestión»
└── GestionInventarioFlores
    ├── +cargarInventario()
    ├── +serializarInventario()
    ├── +agregarInventario()
    ├── +actualizarInventario()
    └── +actualizarInventarioArchivo()

«Modelo - Gestión»
└── GestionClases
    ├── +cargarClases()
    ├── +serializarClases()
    ├── +buscarClasePorId()
    └── +eliminarClase()

«Modelo - Sistema»
└── SistemaMembresias0112
    ├── -precio_bronce: double
    ├── -precio_plata: double
    ├── -precio_oro: double
    ├── -puntos_bronce: int
    ├── -puntos_plata: int
    ├── -puntos_oro: int
    ├── +cargarMembresias()
    ├── +serializarMembresia()
    └── +agregarMembresia()

«Modelo - Control»
└── ControlAccesoClientesFlores
    ├── +registrarEntrada()
    └── +registrarSalida()
```

«Modelo - Control»
└── ControlAccesoEmpleadosFlores
    ├── +registrarEntrada()
    └── +registrarSalida()

«Modelo - Control»
└── ControlVentasProductosFlores
    └── +registrarVenta()

«Modelo - Utilidad»
└── ProcesadorPagos564
    ├── +leerDinero()
    ├── +procesarPago()
    └── +guardarFactura()

«Modelo - Utilidad»
└── GeneradorReportesS
    ├── +run()
    └── +generarReporte()

«Modelo - Utilidad»
└── NotificadorMembresias
    ├── +run()
    └── +verificarMembresias()

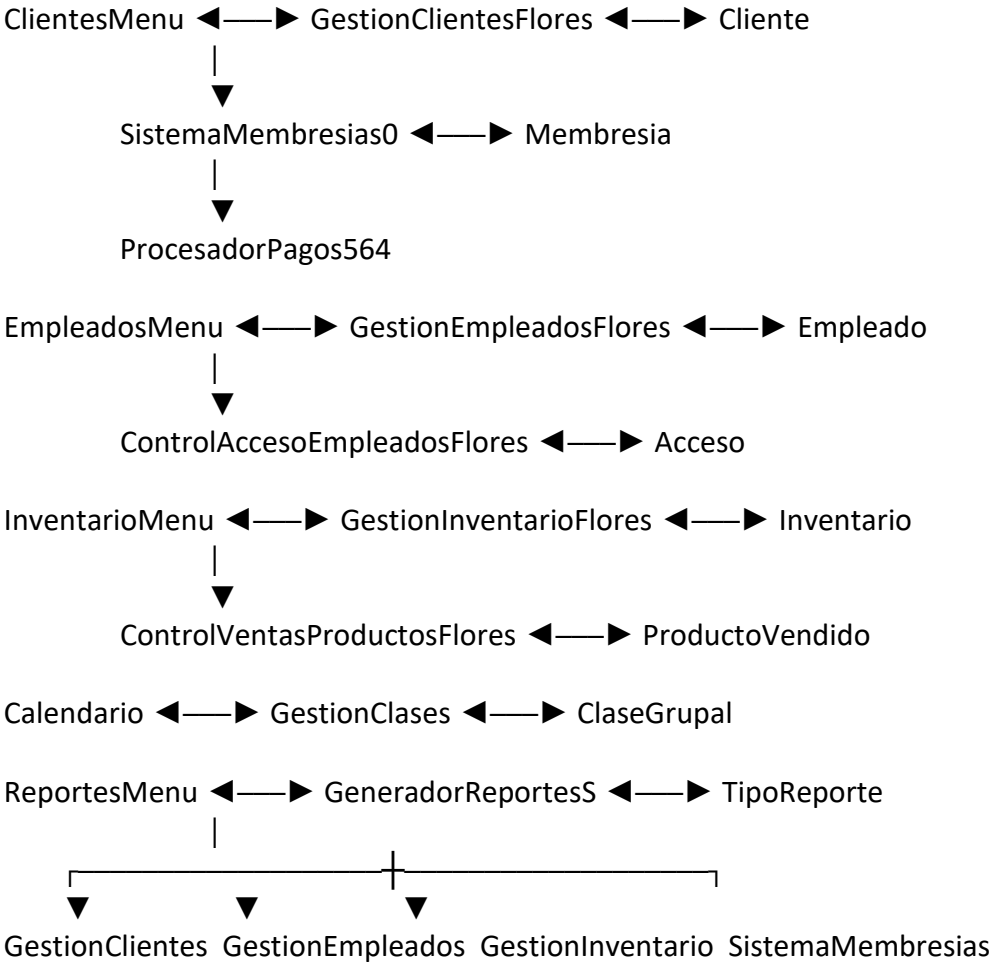
«Enum»
└── TipoReporte
    ├── CLIENTES
    ├── EMPLEADOS
    ├── INVENTARIO
    └── MEMBRESIAS

«Modelo - Transacción»
└── ProductoVendido
    ├── -producto: Inventario
    ├── -cantidadVendida: int
    ├── +getProducto()
    └── +getCantidadVendida()

vista

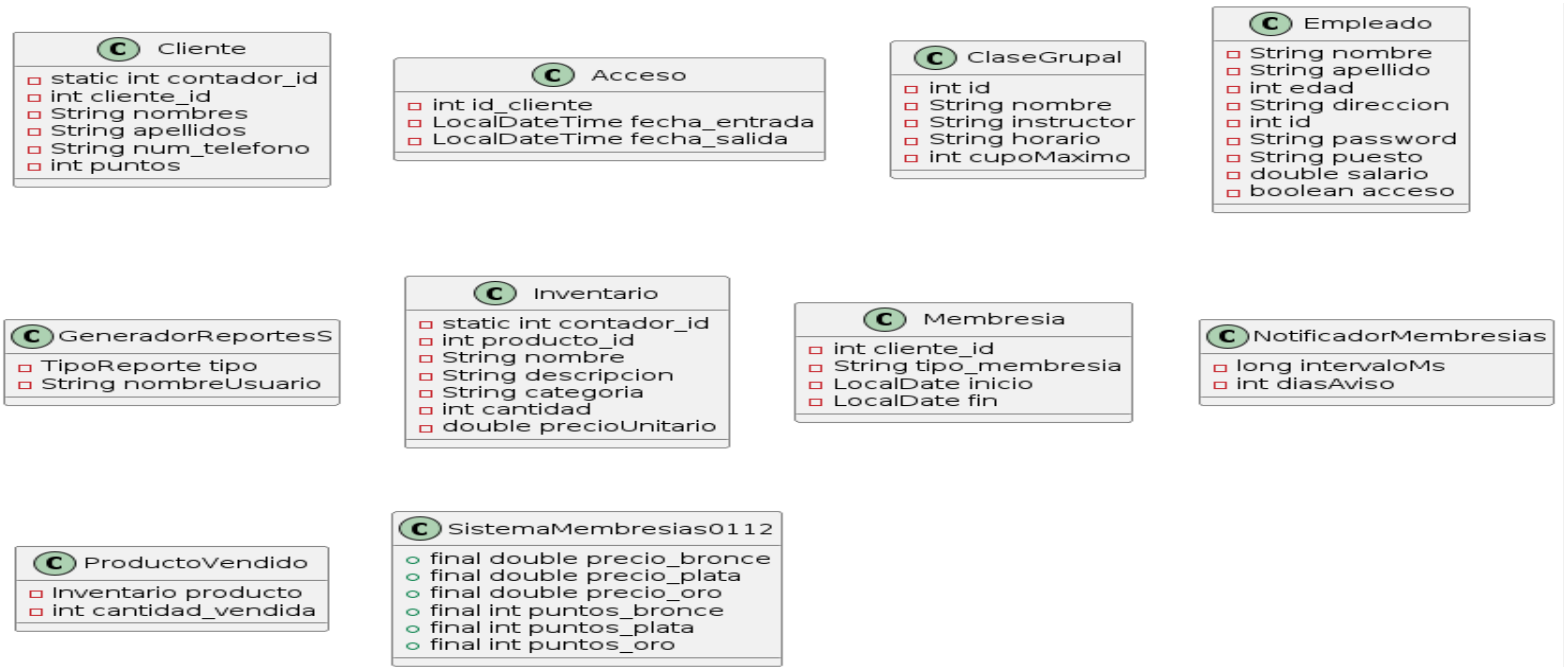
text
«Vista FXML»
├── guifxml.fxml (Controller)
├── registroView.fxml (Registro)
├── menu_clientes.fxml (ClientesMenu)
├── menu_empleados.fxml (EmpleadosMenu)
├── menu_inventario.fxml (InventarioMenu)
├── menu_reportes.fxml (ReportesMenu)
└── calendarioView.fxml (Calendario)

4. ESTRUCTURA DEL PROYECTO (Diagrama UML)



Login ◄──► Empleado
Registro ◄──► GestionEmpleadosFlores

Controller ──► Todos los menús (navegación)



5. MANEJO DE ERRORES

ResponseEntity para devolver códigos HTTP
Manejo centralizado con @ExceptionHandler
Validaciones en la capa servicio
Mensajes describiendo el problem
Validación con @Valid y Bean Validation en DTOs

6. ESCALABILIDAD

1. Sistema de Roles y Permisos Avanzados

Actualmente hay acceso a empleados, pero podrías implementar:
Roles: Administrador, Recepción, Entrenador, Ventas.
Permisos por módulo o acción.
Acceso restringido a inventario, pagos, reportes, etc.

Eleva la seguridad y control de la operación del gimnasio.

2. Membresías Inteligentes

Agregar funciones como:
Renovación automática.
Recordatorios antes del vencimiento.
Congelación de membresía por enfermedad o vacaciones.
Historial de períodos activos/inactivos.

3. Control de Asistencias

Un módulo nuevo donde:
El cliente registra entrada/salida.
Se almacenan estadísticas de visitas.
Se muestra asistencia por semana, mes, año

4. Agenda de Entrenadores / Calendario

Aprovechando tu carpeta Calendario:
Gestión de citas con entrenadores.
Reservación de clases grupales.
Notificaciones automáticas al cliente.

5. Módulo de Evaluación Física del Cliente

Para registrar parámetros como:
peso
estatura
porcentaje de grasa
circunferencias
progreso mensual
Con gráficos visuales en Reportes.

7. RECOMENDACIONES PARA FUTURAS MEJORAS

* Implementación de JWT para seguridad

* Documentación automática con Swagger

* Implementación de pruebas unitarias con JUnit 5

8. CONCLUSIÓN

El sistema GYM POWER HOUSE JDS es una aplicación moderna basada en Java 25 y JavaFX, diseñada para ser modular, mantenible y escalable.

Este manual proporciona toda la información necesaria para instalar, ejecutar y extender el proyecto de forma segura y eficiente.