Universidad Mariano Gálvez
Ingeniería En Sistemas De Información Y Ciencias de la Computación

Curso: Estadística I

Ciclo: 5to ciclo

Sección A

Catedrática: ing. Luis Fernando Alvarado cruz

Nombre: Angel Roberto López López

No. De carné: 7690-24-17905

Sede: Liceo Centroamericano, Boca del Monte

Manual del Sistema de Atletas de Guatemala

Índice

Contenido

Introducción

El Sistema de Atletas de Guatemala es una aplicación de consola desarrollada en Java para la gestión del Comité Olímpico Guatemalteco. Permite registrar atletas, gestionar entrenamientos, generar estadísticas y manejar reportes.

Características Principales

- Gestión completa de atletas y entrenamientos
- Soporte para MariaDB y archivos JSON
- Generación de reportes CSV
- Sistema de estadísticas avanzadas
- Procesamiento de planilla de pagos
- Sincronización de datos entre formatos

Requisitos del Sistema

Software Necesario

- **Java JDK 17 (tu proyecto usa Java 17)**
- **IntelliJ IDEA (recomendado) o cualquier IDE Java**
- **MariaDB Server (opcional, para base de datos)**
- **Maven para gestión de dependencias**

Archivo pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>gt.olimpico</groupId>
    <artifactId>sistema-atletas-guatemala</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <!-- JSON -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
```

```
        <version>2.10.1</version>
    </dependency>

    <!-- MariaDB JDBC Driver -->
    <dependency>
        <groupId>org.mariadb.jdbc</groupId>
        <artifactId>mariadb-java-client</artifactId>
        <version>3.2.0</version>
    </dependency>

    <!-- HikariCP -->
    <dependency>
        <groupId>com.zaxxer</groupId>
        <artifactId>HikariCP</artifactId>
        <version>5.0.1</version>
    </dependency>

    <!-- SLF4J API -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>2.0.9</version>
    </dependency>

    <!-- Logback como implementación de SLF4J -->
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.4.11</version>
    </dependency>
</dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Instalación y Configuración

Paso 1: Configurar el Proyecto

1. Crear un nuevo proyecto Maven en IntelliJ IDEA
2. Copiar todas las clases Java en el paquete org.example
3. Agregar las dependencias al archivo pom.xml
4. Ejecutar mvn clean install para descargar dependencias

Paso 2: Configurar Base de Datos (Opcional)

1. Instalar MariaDB Server
2. Crear la base de datos:

```
-- Crear base de datos si no existe
CREATE DATABASE IF NOT EXISTS atletas_guatemala;
USE atletas_guatemala;

-- ========================
-- TABLA DE ATLETAS
-- ========================
DROP TABLE IF EXISTS entrenamientos;
DROP TABLE IF EXISTS atletas;

CREATE TABLE atletas (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    nombre VARCHAR(100) NOT NULL,
    edad INT NOT NULL,
    disciplina VARCHAR(100) NOT NULL,
    departamento VARCHAR(50) NOT NULL,
    nacionalidad VARCHAR(50) NOT NULL,
    fecha_ingreso DATE,
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    activo BOOLEAN DEFAULT TRUE
) ENGINE=InnoDB;

-- ==========================
-- TABLA DE ENTRENAMIENTOS
-- ==========================
CREATE TABLE entrenamientos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    atleta_id INT NOT NULL,
    fecha DATE NOT NULL,
    tipo VARCHAR(100) NOT NULL,
    marca DECIMAL(10,2) NOT NULL,
    ubicacion VARCHAR(20) NOT NULL,
    pais VARCHAR(60),
```

```
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (atleta_id) REFERENCES atletas(id) ON DELETE CASCADE
) ENGINE=InnoDB;

-- ==========================
-- VISTA: Información completa de atletas
-- ==========================
CREATE OR REPLACE VIEW vista_atletas_completa AS
SELECT
    a.id,
    a.nombre,
    a.edad,
    a.disciplina,
    a.departamento,
    a.nacionalidad,
    a.fecha_ingreso,
    COUNT(e.id) AS total_entrenamientos,
    COALESCE(MAX(e.marca), 0) AS mejor_marca,
    COALESCE(AVG(e.marca), 0) AS marca_promedio
FROM atletas a
LEFT JOIN entrenamientos e ON a.id = e.atleta_id
```

```
WHERE a.activo = TRUE
GROUP BY a.id, a.nombre, a.edad, a.disciplina, a.departamento, a.nacionalidad,
a.fecha_ingreso;


-- ==========================
-- VISTA: Estadísticas por departamento
-- ==========================
CREATE OR REPLACE VIEW estadisticas_departamento AS
SELECT
    departamento,
    COUNT(*) AS total_atletas,
    COUNT(DISTINCT disciplina) AS disciplinas_diferentes
FROM atletas
WHERE activo = TRUE
GROUP BY departamento
ORDER BY total_atletas DESC;


-- ==========================
-- Reiniciar AUTO_INCREMENT si las tablas están vacías
-- ==========================
ALTER TABLE atletas AUTO_INCREMENT = 1;
```

ALTER TABLE entrenamientos AUTO_INCREMENT = 1;

Paso 3: Configurar Conexión

Modificar los parámetros de conexión en ConexionBD.java:

```
private static final String DB_HOST = "localhost";
private static final String DB_PORT = "3306";
private static final String DB_NAME = "atletas_guatemala";
private static final String DB_USER = "tu_usuario";
private static final String DB_PASSWORD = "tu_contraseña";
```

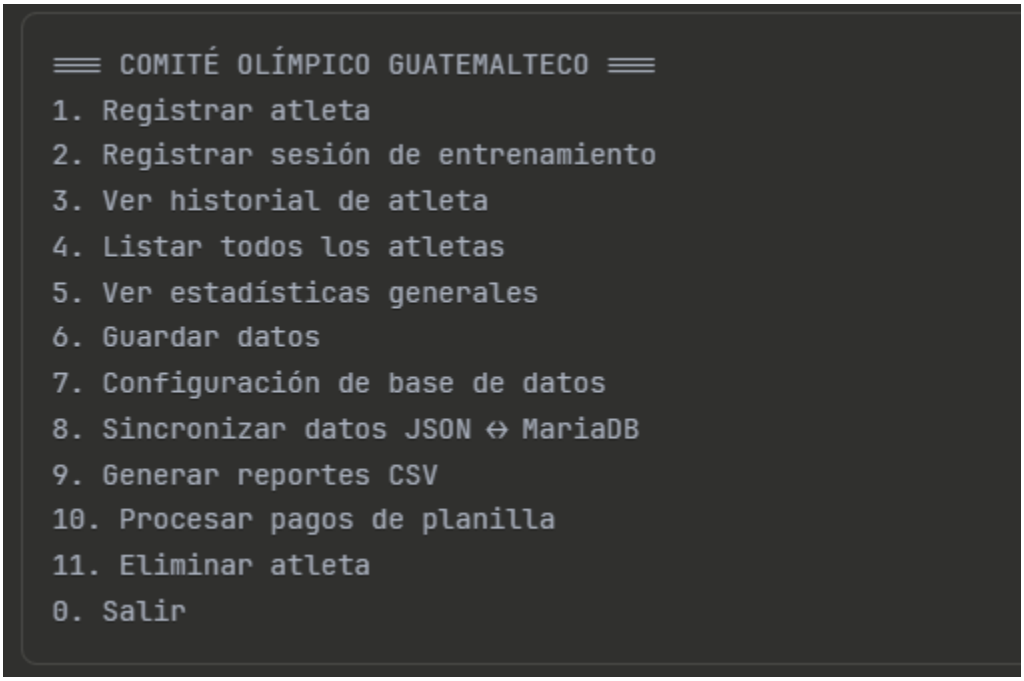
Estructura del Proyecto

```
org.example/
├── Main.java                    # Punto de entrada
├── model/
│   ├── Atleta.java              # Entidad atleta
│   └── Entrenamiento.java       # Entidad entrenamiento
├── dao/
│   ├── AtletaDAO.java           # Acceso a datos de atletas
│   └── EntrenamientoDAO.java    # Acceso a datos de
entrenamientos
├── service/
│   ├── ServicioAtletas.java     # Lógica de negocio
│   ├── PlanillaService.java     # Procesamiento de pagos
│   └── EstadisticasAtleta.java # Cálculos estadísticos
├── repository/
│   └── RepositorioAtletas.java # Gestión de persistencia
├── presentation/
│   └── PresentadorAtletas.java # Interfaz de usuario
├── utils/
│   ├── ConexionBD.java          # Conexión a base de datos
│   ├── CsvUtil.java             # Utilidades CSV
│   └── LocalDateAdapter.java    # Adaptador para fechas
```

Guía de Usuario

Iniciar el Sistema

1. Ejecutar la clase Main.java
2. Aparecerá el menú principal con las opciones disponibles



Registro de Atletas

1. Seleccionar opción **1**
2. Ingresar los datos solicitados:
    - Nombre completo
    - Edad
    - Disciplina deportiva
    - Departamento de origen
    - Nacionalidad
    - Fecha de ingreso (formato YYYY-MM-DD)

Registro de Entrenamientos

1. Seleccionar opción **2**
2. Ingresar:
    - Nombre del atleta existente
    - Fecha del entrenamiento (YYYY-MM-DD)
    - Tipo de entrenamiento
    - Valor de rendimiento (marca)
    - Ubicación (NACIONAL/INTERNACIONAL)
    - País (solo si es internacional)

Consulta de Historial

1. Seleccionar opción **3**
2. Ingresar el nombre del atleta
3. Ver la evolución completa de entrenamientos

Configuración de Base de Datos

Cambiar Modo de Almacenamiento

El sistema soporta dos modos:

Modo MariaDB

- Datos persistentes en base de datos
- Mayor rendimiento para grandes volúmenes
- Consultas SQL avanzadas

Modo JSON

- Archivos locales en formato JSON
- Ideal para desarrollo y pruebas
- No requiere servidor de base de datos

Sincronización

La opción **8** permite sincronizar datos entre JSON y MariaDB:

- Importa atletas desde archivo JSON a la base de datos
- Útil para migración de datos
- Solo disponible en modo MariaDB

Funcionalidades del Sistema

1. Gestión de Atletas

- **Registro**: Crear nuevos perfiles de atletas
- **Consulta**: Buscar atletas por nombre
- **Listado**: Ver todos los atletas registrados
- **Eliminación**: Remover atletas del sistema

2. Gestión de Entrenamientos

- **Registro**: Agregar sesiones de entrenamiento
- **Clasificación**: Nacional vs Internacional
- **Seguimiento**: Historial completo por atleta

3. Estadísticas

- **Promedio general**: Rendimiento promedio del atleta
- **Mejor marca**: Máximo rendimiento registrado
- **Análisis por ubicación**: Comparación nacional/internacional
- **Evolución temporal**: Progreso en el tiempo

4. Reportes y Exportación

- **CSV de atletas**: Lista completa en formato Excel
- **CSV de entrenamientos**: Detalle por atleta
- **Ubicación de archivos**: Carpeta del proyecto

5. Planilla de Pagos

- **Cálculo automático**: Basado en entrenamientos
- **Procesamiento masivo**: Todos los atletas
- **Fórmula configurable**: Q250 por entrenamiento

Solución de Problemas

Error de Conexión a Base de Datos

Problema: No se puede conectar a MariaDB **Solución**:

1. Verificar que MariaDB esté ejecutándose
2. Comprobar credenciales en ConexionBD.java
3. Verificar que la base de datos existe
4. El sistema automáticamente usará JSON como respaldo

Archivo JSON Corrupto

Problema: Error al cargar datos desde JSON **Solución**:

1. Verificar formato del archivo atletas.json
2. Restaurar desde backup si existe
3. Crear archivo vacío: []

Error de Dependencias

Problema: Clases no encontradas **Solución**:

1. Ejecutar mvn clean install
2. Verificar conexión a internet
3. Actualizar repositorios Maven

Problema de Codificación

Problema: Caracteres especiales mal mostrados **Solución**:

1. Configurar IDE con UTF-8
2. Verificar configuración de consola
3. Agregar -Dfile.encoding=UTF-8 a JVM

Rendimiento Lento

Problema: Sistema lento con muchos datos **Solución**:

1. Usar modo MariaDB para grandes volúmenes
2. Aumentar pool de conexiones en ConexionBD.java
3. Optimizar consultas SQL

Información Técnica

Patrones de Diseño Implementados

- **DAO (Data Access Object)**: Separación de lógica de datos
- **Repository**: Abstracción de persistencia
- **Service Layer**: Lógica de negocio centralizada
- **Factory Method**: Creación de objetos especializados

Manejo de Errores

- Captura de excepciones SQL
- Validación de entrada de datos
- Logging de errores en consola
- Recuperación automática (fallback JSON)

Características Avanzadas

- Pool de conexiones con HikariCP
- Serialización JSON con Gson
- Adaptadores para tipos de datos especiales
- Transacciones automáticas

Contacto y Soporte

Para reportar problemas o sugerir mejoras:

- Revisar la documentación técnica
- Verificar logs de error en consola
- Comprobar configuración de base de datos
- Validar integridad de archivos JSON

NOTA: El sistema está diseñado para ser robusto y recuperarse automáticamente de errores comunes, alternando entre MariaDB y JSON según la disponibilidad.