Manual técnico

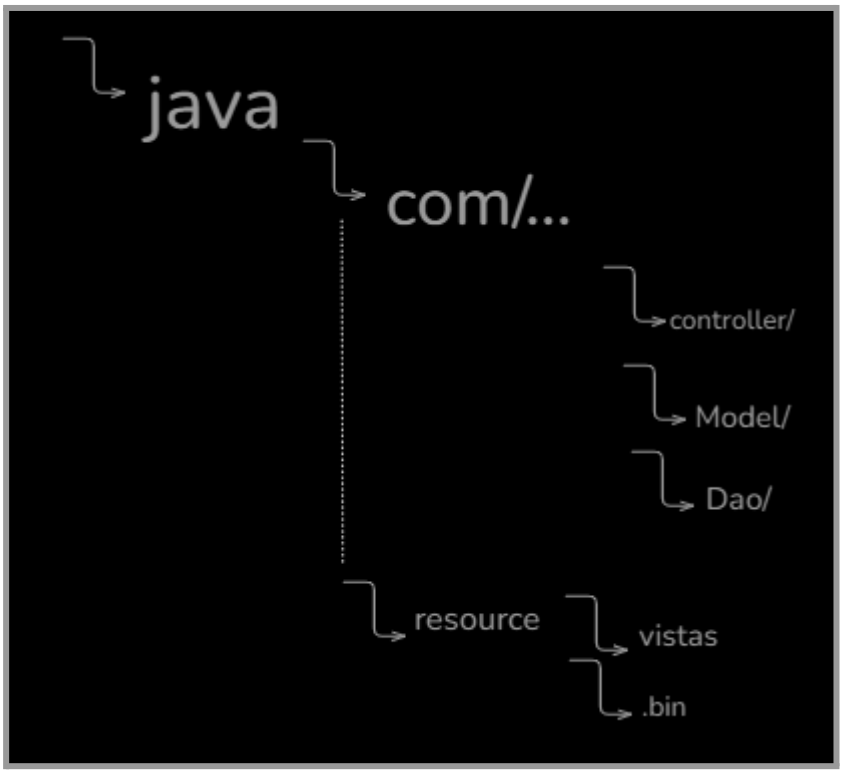
-Por Daniel Álvarez Morales

Índice

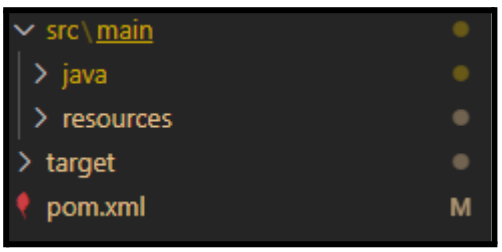
1.- Estructura y patrón mvc

1.1.- Estructura proyecto

-El proyecto tiene una estructura de carpetas como la de esta imagen.



-Y el pom estaría al nivel de carpeta del src\main que es donde se guardarán y gestionarán las dependencias y extensiones de maven.

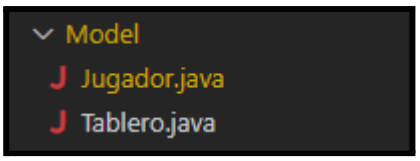


-La mayoría de carpetas están separadas con la razón de respetar el patrón mvc que tiene su sentido y se explicará más adelante.

1.2.- Patrón mvc

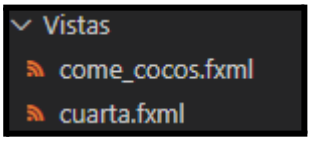
-El patrón MVC separa una aplicación en tres componentes:

-El Modelo son los objetos java que representan los datos y la lógica de la aplicación.

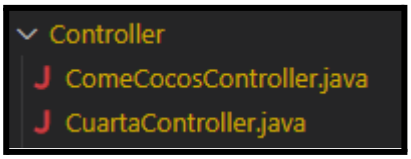


- El “DAO” se encarga del acceso a datos de los objetos.

-La Vista se encarga de la interfaz de usuario.



-El Controlador actúa como intermediario entre ambos, respondiendo a las acciones del usuario y actualizando Modelo y Vista según sea necesario.



-En resumen, el patrón MVC permite organizar el código separando la lógica de negocio, la interfaz gráfica y la interacción del usuario, facilitando el mantenimiento y la escalabilidad.

1.3.- La clase app.

-La clase principal hereda de Application, que es una clase abstracta de JavaFX.

```
public class App extends Application {
```

-Guardamos un conjunto de raíces y controladores.

```
// Guardamos las raíces
private static Parent secondaryRoot;
```

```
// Guardamos los controladores
private static SecondaryController secondaryController;
```

-Para luego en el método “Start” cargarlos.

```
// Cargar secondary.fxml
FXMLLoader loaderSecondary = new FXMLLoader(getClass().getResource("/Vistas/secondary.fxml"));
secondaryRoot = loaderSecondary.load();
secondaryController = loaderSecondary.getController();
```

-”Start” donde se define la ventana principal (Stage) y lo que se va a mostrar (la escena o interfaz).

-Este método se ejecuta después de que JavaFX ya está completamente cargado.

```
@Override
public void start(Stage stage) throws IOException {
```

-En JavaFX, la ejecución comienza con main(), que llama a “launch()”,
este método prepara todo el entorno gráfico y después invoca start().
Donde el programa define la interfaz y mostramos la ventana principal (primaryStage).

- ”launch()” inicializa el entorno de JavaFX: carga internamente el toolkit gráfico y prepara todo para mostrar ventanas.

```
public static void main(String[] args) {
    launch();
}
```

-Después de ejecutar “launch()”, automáticamente llama al método start().

1.4.- La conexión a la base de datos

-El constructor que alberga una URL que es la dirección donde se creará la base de datos.

```
public class DatabaseConnector {
    //private static final String URL = "jdbc:sqlite:base.db";
    private static final String URL = "jdbc:sqlite:C:/Users/daniy/OneDrive/Escritorio/visualStudioClases/MavenBase/base.db";
```

-El procedimiento para conectarse a la base de datos que devuelve la URL.

```
public static Connection conectar() {
    try {
        return DriverManager.getConnection(URL);
    } catch (SQLException e) {
        System.err.println("Error al conectar a SQLite: " + e.getMessage());
        return null;
    }
```

-Esto sirve por ejemplo para inicializar la tabla o lista de base de datos.

1.5.- El dao

-El dao contiene el acceso a datos por lo tanto tiene albergado procedimientos como guardar, recuperar, actualizar y demás procedimiento que en este caso es para jugador.

a) Guardar en archivo binario de jugadores.

```
public void grabar_datos(String ruta, String fichero) throws Exception {
```

b) Recuperar el archivo binario de jugadores.

```
public List<Jugador> recuperar_datos(String ruta, String fichero) {
```

c) Eliminar jugador de la lista de jugadores.

```
public boolean eliminarJugador(Integer busquedaId) {
```

d) Actualizar jugador de la lista de jugadores.

```
public boolean actualizarJugador(Integer busquedaId, String nombre) {
```

1.6.- Conexión controladores y vista

-Vista fxml:

```
<VBox alignment="CENTER" spacing="20.0" xmlns="http://javafx.com/javafx/8.0.171"
  xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.base.Controller.PrimaryController"
  stylesheets="stylesheets= @/ css/ Menu.css">
```

- fx:controller=”com.base.Controller.PrimaryController”.

-Así es como el proyecto conecta la vista (xml) con el controller.

-En stylesheets es donde se conecta el css que da diseño a la interfaz de la app.

1.7.- Manejo del id del jugador

-En el proyecto desarrollado con JavaFX, se le asigna a cada jugador un identificador (id) incremental cuando se trabaja en modo local. Este identificador se le genera automáticamente al jugador mediante un contador global estático dentro de la clase, y se le incrementa cada vez que se le crea un nuevo objeto. De esta manera, se le garantiza a cada jugador un id único mientras los datos se almacenan en archivos binarios.

```
// Crear tabla si no existe
String createTableQuery = "CREATE TABLE IF NOT EXISTS jugadores
(id INTEGER PRIMARY KEY AUTOINCREMENT, nombre TEXT, puntos REAL)";
```

-Por otro lado, en el modo online, los jugadores también pero al insertarlos a la base de datos los gestiona los identificadores como PRIMARY KEY AUTOINCREMENT, por lo que a cada nuevo jugador se le asigna automáticamente un nuevo id desde la base de datos, sin importar el valor que se le haya proporcionado desde Java mediante el contador global.

-Sin embargo el próximo jugador que se cree offline tendrá un id más alto pero no es importante mientras el id no se repita. Además, hay un límite de jugadores tanto online como offline.

```
if (listaJugadores.size() >= 4) {
    System.out.println("No se puede agregar mas de 4 jugadores");
```

```java
if (cantidadJugadores >= 12) {
    System.out.println("No se puede insertar: ya hay 12 jugadores o más.");
```

(Condicionales de la clase JugadorDao.java)

-Si se llega a eliminar un jugador y su id también siempre se genera por encima.

```java
public static void actualizarContadorId(List<Jugador> jugadores) {
    // Estático: para sincronizar el contador con la lista de jugadores recuperados

    // Si la lista está vacía, volvemos al valor inicial del contador
    if (jugadores == null || jugadores.isEmpty()) {
        contadorId = 0;
    } else {
        int maxId = jugadores.get(0).id; // Asumimos que hay al menos un jugador
        // Recorremos la lista para encontrar el ID más alto
        for (int i = 1; i < jugadores.size(); i++) {
            Jugador jugador = jugadores.get(i);
            if (jugador.id > maxId) {
                maxId = jugador.id;
            }
        }
        // Establecemos el contador al siguiente valor disponible
        contadorId = maxId + 1;
    }
}
```

(Método de la clase modelo jugador.java)

2.-Dependencias utilizadas en pom.xml

```xml
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>17</version>
</dependency>
```

-Incluye los componentes visuales básicos de JavaFX como botones, etiquetas y demás.

```xml
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>17</version>
</dependency>
```

-Permite trabajar con archivos .fxml (XML) y que se conecten a los controladores java.

```xml
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.43.2.0</version>
</dependency>
```

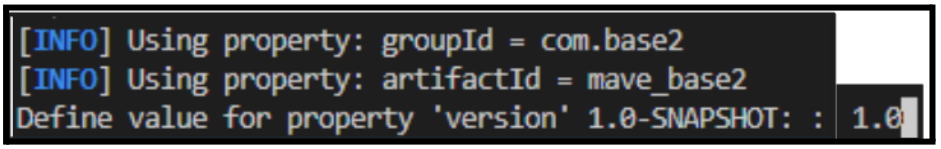
-Es el driver JDBC que permite conectarse y trabajar con bases de datos SQLite desde Java. Esto nos permitirá la conexión a la base de datos.

3.-Instrucciones para compilar y ejecutar el proyecto.

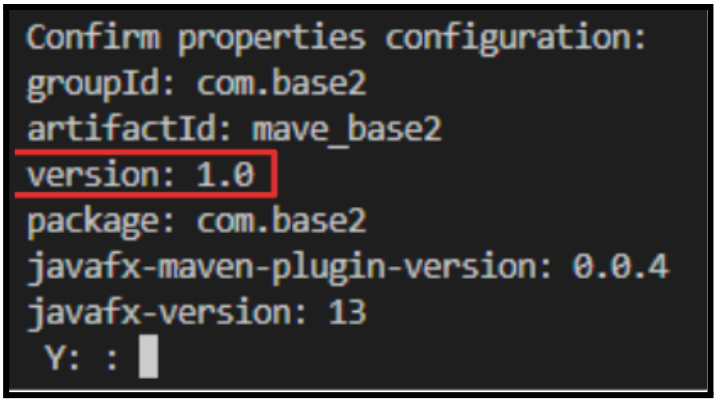
3.1.- Inicialización del proyecto con maven

- El programa debe usar maven e iniciar el proyecto maven de esta forma

-El usuario le asigna las versiones cuando el ide se lo exija, también por consola.



-El usuario cuando haya asignado todo tendrá que confirma la creación escribiendo ‘y’;



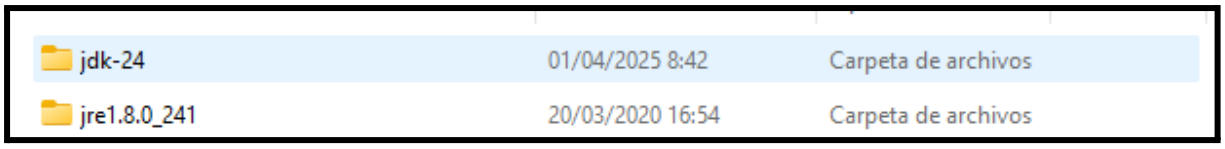
-La consola mostrará si el proceso ha sido exitoso.

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------
[INFO] Total time:  05:58 min
[INFO] Finished at: 2025-04-24T10:53:46+02:00
[INFO] ------------------------------------------
```

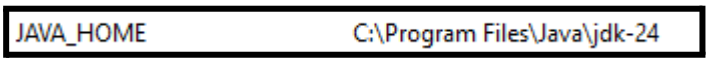
3.3.- Compilación y ejecución del proyecto

-Después de inicializar con maven el proyecto hay que trabajar sin generar errores de compilación, ejecución y además gestionar adecuadamente el pom a las necesidades del proyecto.

-También hay que tener en cuenta la versión del jdk y como lo tenemos asignado en java_home.

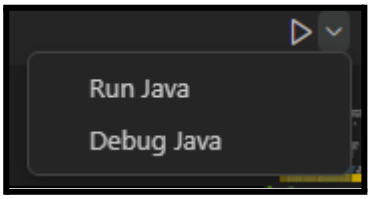


(El jdk en la carpeta Java)



(El jdk asignado a java_home en la variable de entornos)

-Y finalmente, ejecutar el botón de ejecución del entorno de desarrollo correspondiente donde está el programa java fx en este caso visual studio.



(El botón de ejecutar de visual studio)