

INFORME TÉCNICO DEL PROYECTO

Football Manager 2025

Asignatura: Programación III

ALUMNOS: Jaime Roldán Ortiz de Uriarte , Hugo Armentia Vázquez , Josu Sáez de Gordoa, Kimetz Hernández

1. Introducción

El proyecto **Football Manager 2025** consiste en el desarrollo de una aplicación de escritorio en Java que simula la gestión de un club de fútbol. La aplicación permite al usuario asumir el rol de mánager, tomando decisiones relacionadas con jugadores, equipos y organización general del club, siguiendo un enfoque similar al de los videojuegos de tipo *football manager*.

El objetivo principal del proyecto es aplicar de forma práctica los conocimientos adquiridos en la asignatura **Programación III**, integrando programación orientada a objetos, diseño de interfaces gráficas con **Swing**, gestión de eventos, concurrencia mediante **hilos**, y una arquitectura bien estructurada y mantenible.

Este proyecto no solo busca cumplir los requisitos técnicos de la asignatura, sino también desarrollar una aplicación coherente, usable y escalable.

2. Arquitectura general del proyecto

El proyecto sigue una **arquitectura modular basada en paquetes**, separando claramente las responsabilidades de cada parte del sistema. Esta organización facilita la comprensión del código, su mantenimiento y futuras ampliaciones.

La estructura general del proyecto se divide en los siguientes paquetes principales:

- main
- gui
- domain
- db (cuando aplica persistencia)

Esta separación respeta el principio de **separación de responsabilidades**, fundamental en el desarrollo de software orientado a objetos.

3. Paquete main

El paquete `main` contiene el **punto de entrada de la aplicación**, incluyendo la clase que define el método `main`.

Desde este método se lanza la interfaz gráfica inicial utilizando:

```
Java  
SwingUtilities.invokeLater(...)
```

Esto garantiza que toda la interfaz gráfica se cree y gestione dentro del **Event Dispatch Thread (EDT)**, evitando problemas de concurrencia, ya que Swing no es thread-safe.

Responsabilidades principales del paquete `main`:

- Arranque de la aplicación
- Inicialización de la ventana principal
- Coordinación inicial entre componentes

4. Paquete `gui` – Interfaz gráfica de usuario

El paquete `gui` contiene todas las clases relacionadas con la **interfaz gráfica**, desarrollada íntegramente con **Java Swing**, sin el uso de editores visuales automáticos.

4.1 Ventanas y contenedores

Se utilizan principalmente:

- `JFrame` como ventanas principales
- `JDialog` para ventanas emergentes y diálogos modales

Cada ventana representa una funcionalidad concreta del sistema, facilitando la navegación y evitando interfaces sobrecargadas.

4.2 Componentes Swing utilizados

La interfaz hace uso de múltiples componentes Swing, entre ellos:

- `JButton` para acciones del usuario
- `JLabel` para mostrar información

- JTextField y JTextArea para entrada de datos
- JList y JComboBox para selección de elementos
- JTable para mostrar información estructurada (jugadores, equipos, estadísticas)

El uso de estos componentes permite una interacción rica y clara con el usuario.

5. Layout Managers y organización visual

Para la organización de los componentes se emplean distintos **Layout Managers**, evitando el posicionamiento absoluto siempre que es posible.

Entre los layouts utilizados destacan:

- BorderLayout para estructurar ventanas principales
- FlowLayout para agrupaciones simples
- GridLayout para formularios y botones
- BoxLayout para disposiciones verticales u horizontales

Gracias a esto, la interfaz se adapta correctamente a diferentes tamaños de ventana y mantiene una estructura coherente.

6. Gestión de eventos

La interacción del usuario se gestiona mediante el **modelo de eventos de Swing**. Cada componente registra los listeners necesarios para responder a las acciones del usuario.

Listeners utilizados:

- ActionListener para botones y menús
- ListSelectionListener para listas y tablas
- KeyListener para eventos de teclado
- MouseListener / MouseMotionListener para eventos de ratón

Los listeners se implementan mediante:

- Clases internas
- Clases anónimas
- Expresiones lambda (Java 8+)

Todo el código de los listeners se ejecuta en el **hilo de Swing**, por lo que se evita introducir tareas costosas directamente en ellos.

7. Modelos de datos y componentes avanzados

7.1 Uso de JTable

El componente `JTable` se utiliza para mostrar información compleja y estructurada, como listas de jugadores o datos de equipos.

La tabla se basa en un **modelo de datos**, generalmente `DefaultTableModel` o un modelo personalizado que hereda de `AbstractTableModel`.

Esto permite:

- Separar datos y visualización
- Actualizar la tabla dinámicamente
- Controlar qué celdas son editables

7.2 Renderers personalizados

Para mejorar la visualización, se emplean **renderers personalizados**, adaptando la forma en la que se muestran ciertos datos en listas o tablas.

Esto mejora la experiencia de usuario y cumple con los requisitos avanzados de la asignatura.

8. Paquete domain – Lógica de negocio

El paquete `domain` contiene las **clases del dominio**, que representan los elementos fundamentales del sistema.

Ejemplos de clases típicas:

- Jugador
- Equipo
- Club
- Partido
- Temporada
- Entrenador

Estas clases:

- Encapsulan los datos mediante atributos privados
- Ofrecen acceso controlado mediante getters y setters
- Implementan la lógica del negocio
- Son independientes de la interfaz gráfica

Este diseño permite reutilizar la lógica del sistema sin depender de la UI.

9. Concurrencia y uso de hilos

El proyecto utiliza **hilos de ejecución independientes** para realizar tareas costosas sin bloquear la interfaz gráfica.

Se crean hilos mediante:

- Implementación de la interfaz Runnable
- Creación de instancias de Thread

El uso de hilos permite:

- Mantener la interfaz responsive
- Evitar bloqueos del Event Dispatch Thread
- Ejecutar procesos en segundo plano

Se respetan las normas de concurrencia de Swing, evitando modificar componentes gráficos desde hilos externos.



10. Persistencia y base de datos (si aplica)

Cuando se utiliza persistencia, el proyecto incorpora una base de datos **SQLite**, gestionada desde el paquete db.

Se implementan operaciones CRUD:

- Creación de tablas
- Inserción de datos
- Consulta
- Actualización
- Eliminación

Se utilizan PreparedStatement para:

- Evitar inyecciones SQL
- Mejorar el rendimiento
- Mantener el código más limpio y seguro

11. Buenas prácticas y estilo de código

Durante el desarrollo del proyecto se han aplicado buenas prácticas de programación:

- Código organizado en paquetes
- Nombres de clases y métodos significativos
- Uso de encapsulación y orientación a objetos
- Separación entre lógica y presentación
- Gestión adecuada de excepciones
- Comentarios claros cuando es necesario

12. Uso de recursividad

El proyecto incorpora **recursividad** para cumplir los requisitos de la asignatura y aplicar este concepto a un problema real del dominio. En concreto, se ha implementado un método recursivo para la **búsqueda del próximo partido** que debe disputar un equipo dentro del calendario de la liga.

La recursividad se ha ubicado en la clase LeagueCalendar, perteneciente al paquete domain, mediante un método que recorre el calendario jornada a jornada hasta encontrar el primer partido en el que participa el equipo o hasta que no quedan más jornadas por analizar.

Esta solución reemplaza una implementación previa basada en bucles anidados, mejorando la **claridad del código**, reforzando la **separación entre la lógica de negocio y la interfaz gráfica**, y demostrando un uso correcto y justificado de la recursividad dentro del proyecto.

13. Ejecución apropiada para el ejecutable

Requiere Java 21 LTS. Ejecutar con java -jar fm25.jar o doble click.



GANNT

GANTT_G14