



## **Proyecto Tenistas**

### **Integrantes:**

Luis Diego Hidalgo Aguero

Camilla Quirós Torres

**Universidad CENFOTEC**

**Curso:** Estructuras de datos 2

**Profesor:** David Campos Castro

**Fecha:** III Cuatrimestre, 2025

## **Resumen ejecutivo**

Este informe documenta el análisis, diseño e implementación de una aplicación de escritorio en Java/JavaFX para gestionar datos de los mejores tenistas de la ATP conforme a la consigna del curso. La solución permite el ingreso validado de registros (nombre, país, edad, puntaje), su visualización tabular y la aplicación de tres ordenamientos: puntaje (descendente), edad (ascendente) y país + nombre (ascendente), implementados sin librerías de ordenamiento de alto nivel. Se describe la arquitectura por capas, la relación entre interfaz y lógica, la política de validaciones, la trazabilidad con los requisitos y se proponen ajustes para cumplir literalmente el almacenamiento inicial en matriz exigido por la consigna. Se incluyen pautas de prueba, manejo de errores y recomendaciones de mejora incremental.

Palabras clave: JavaFX; listas; ordenamientos; validaciones; interfaz gráfica; trazabilidad de requisitos.

Repositorio del proyecto:

<https://github.com/camillaquiro2/ProyectoTenistas-EstructurasdeDatos.git>

## **Índice general**

<b>Capítulo 1 - Introducción</b>	<b>6</b>
<b>Capítulo 2 - Requisitos del sistema</b>	<b>7</b>
<b>Capítulo 3 - Visión general de la solución desarrollada</b>	<b>11</b>
<b>Capítulo 4 - Arquitectura y diseño</b>	<b>13</b>
4.1. Modelo de datos (model.Tenista)	13
4.2. Lógica de negocio y validaciones (logic.*)	13
4.3. Interfaz de usuario (gui.*) y flujo de eventos	13
<b>Capítulo 5 - Algoritmos implementados y justificación</b>	<b>14</b>
<b>Capítulo 6 - Lista de Referencias</b>	<b>16</b>

## **Índice de Tablas**

**Tabla de Requerimientos**

**7**

## **Índice de figuras**

**(Figura 1. Diagrama de paquetes y flujo MVC)**

**8**

## **Capítulo 1**

### **Introducción**

El proyecto se enmarca en el curso Estructura de Datos 2 (BISOFT-20) y busca que el estudiantado integre estructuras y ordenamientos en una aplicación gráfica. La consigna solicita: ingreso de 15 tenistas de la ATP, almacenamiento inicial en matriz, visualización sin orden previo y tres ordenamientos (puntaje descendente; edad ascendente; país ascendente con nombre como segundo criterio), sin uso de librerías prediseñadas para ordenar y con validaciones suficientes para evitar fallos en ejecución. La presente memoria técnica describe la solución, su arquitectura y el grado de cumplimiento con la consigna, además de proponer mejoras puntuales.

## **Capítulo 2**

### **Requisitos del sistema**

El sistema es una aplicación de escritorio (JavaFX) para la gestión de tenistas ATP.

Basado en la lógica implementada, el sistema cumple con lo siguiente:

- Modelo de Datos (Tenista.java): Gestiona nombre, país, edad y puntaje.
- Validaciones (Validaciones.java): Impone reglas de negocio (edad entre 15-45, puntaje no negativo, campos obligatorios).
- Gestión de Datos (ControladorTenistas.java): Usa un ArrayList para almacenar datos en memoria y evita duplicados por nombre.
- Algoritmos (Ordenamientos.java): Implementa algoritmos específicos de ordenamiento:
  - QuickSort para Puntaje (Descendente).
  - Inserción para Edad (Ascendente).
  - ShellSort para País + Nombre.
- Interfaz Gráfica: Permite el ingreso de datos y la visualización en una tabla (TableView).

### **Tabla de Requerimientos**

Tipo de requerimiento	ID / Descripción	Requerimiento previo(s)	Estado
Funcional	<b>RF01:</b> Permitir al usuario registrar un tenista ingresando Nombre, País, Edad y Puntaje.	N/A	Implementado
Funcional	<b>RF02:</b> Validar que la edad esté entre 15 y 45 años y el puntaje sea positivo antes de guardar.	RF01	Implementado
Funcional	<b>RF03:</b> Validar que no se registren dos tenistas con el mismo nombre (control de duplicados).	RF01	Implementado
Funcional	<b>RF04:</b> Visualizar el listado de tenistas registrados en una tabla con sus atributos.	RF01	Implementado
Funcional	<b>RF05:</b> Ordenar la lista de tenistas por <b>Puntaje</b> de forma descendente (algoritmo QuickSort).	RF04	Implementado
Funcional	<b>RF06:</b> Ordenar la lista de tenistas por <b>Edad</b> de forma ascendente (algoritmo Inserción).	RF04	Implementado
Funcional	<b>RF07:</b> Ordenar la lista de tenistas por <b>País y Nombre</b> (algoritmo ShellSort).	RF04	Implementado

Requerimientos no funcionales	ID / Descripción	Requerimiento previo(s)	Estado
No funcional	<b>RNF01:</b> El sistema debe contar con una Interfaz Gráfica de Usuario (GUI) desarrollada en JavaFX.	N/A	Implementado
No funcional	<b>RNF02:</b> El sistema debe manejar excepciones en la entrada de datos numéricos (edad y puntaje).	RF01	Implementado
No funcional	<b>RNF03:</b> La persistencia de datos debe ser volátil durante la ejecución (uso de ArrayList en memoria).	N/A	Implementado

## **Capítulo 3**

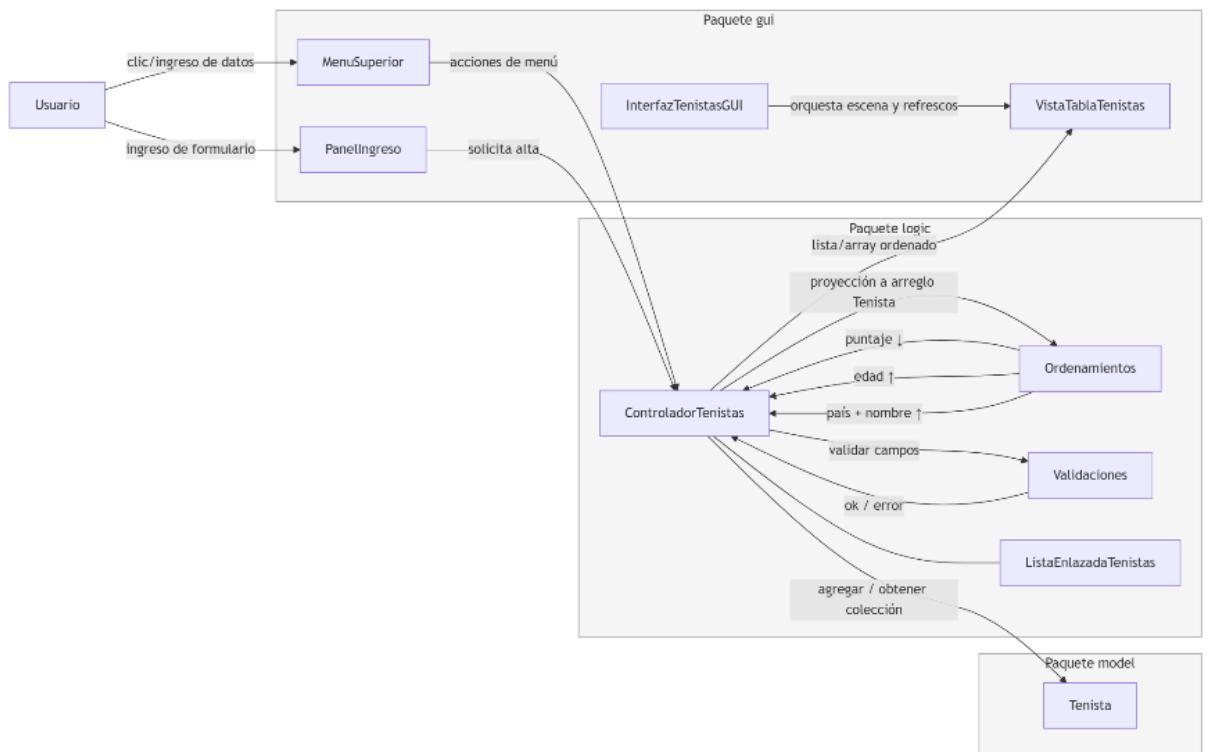
### **Visión general de la solución desarrollada**

El código fuente se organiza en tres capas:

- **Modelo:** model.Tenista (POJO con nombre, país, edad, puntaje).
- **Lógica:** logic.ControladorTenistas (gestión de colección, agregado, ordenamientos, utilitarios); logic.Ordenamientos (implementaciones propias); logic.Validaciones (reglas de entrada); logic.ListaEnlazadaTenistas (estructura alternativa didáctica).
- **Interfaz:** gui.InterfazTenistasGUI (escena principal), gui.MenuSuperior (botones de acciones), gui.PanelIngreso (formulario y alta), gui.VistaTablaTenistas (TableView con columnas enlazadas).

La interfaz invoca acciones del controlador; este transforma los datos (p. ej., a arreglo) y llama a los algoritmos de Ordenamientos; luego actualiza la tabla. Este patrón concuerda con el enfoque recomendado de JavaFX para separar datos observables y vista.

(Figura 1. Diagrama de paquetes y flujo MVC)



## Capítulo 4

### Arquitectura y diseño

#### 4.1. Modelo de datos (model.Tenista)

Entidad con cuatro atributos y métodos de acceso; `toString()` formatea columnas para la salida tabular. Es la unidad base para mostrar y ordenar.

#### 4.2. Lógica de negocio y validaciones (logic.\*)

- **Contenedor principal:** ControladorTenistas utiliza `ArrayList<Tenista>` como fuente de verdad y expone `obtenerTenistas() → Tenista[]` para ordenar.
- **Validaciones:** Validaciones.validarYObtenerError comprueba campos obligatorios, edad en [15, 45] y puntaje no negativo; PanellIngreso muestra mensajes de error coherentes antes de insertar.
- **Ordenamientos sin librerías:** Ordenamientos implementa comparadores y tres algoritmos in-place:
  - Puntaje ↓: QuickSort propio.
  - Edad ↑: Inserción propia.
  - País + nombre ↑: ShellSort propio.

#### 4.3. Interfaz de usuario (gui.\*) y flujo de eventos

InterfazTenistasGUI compone un BorderPane con:

- **Top:** MenuSuperior (Mostrar, Ordenar por Puntaje, Ordenar por Edad, Ordenar por País + Nombre).
- **Center:** VistaTablaTenistas (`TableView<Tenista>` con columnas y `ObservableList`).
- **Bottom:** PanellIngreso (formulario y botón “Aregar”).

## **Capítulo 5**

### **Algoritmos implementados y justificación**

En este capítulo se describen los algoritmos de ordenamiento manual implementados en la clase `logic.Ordenamientos`, su funcionamiento, complejidad y motivo de selección según los criterios de la consigna.

#### **6.1. QuickSort (puntaje descendente)**

El QuickSort fue elegido para ordenar por puntaje por su alta eficiencia promedio ( $O(n \log n)$ ). Divide recursivamente el arreglo en subarreglos menores y mayores que un pivote, logrando ordenamientos rápidos con bajo uso de memoria ( $O(\log n)$ ). Se implementó una versión *in-place* adaptando el comparador para obtener un orden descendente, lo cual prioriza los puntajes más altos de los jugadores. QuickSort es ideal para datos numéricos, demuestra recursividad y eficiencia en tiempo.

#### **6.2. Inserción (edad ascendente)**

El Insertion Sort se seleccionó para ordenar por edad debido a su simplicidad y estabilidad. El algoritmo recorre la lista desde el segundo elemento y “inserta” cada registro en su posición correcta dentro de la parte ya ordenada. Aunque su peor caso es  $O(n^2)$ , con 15 registros su desempeño es excelente y permite un código claro y didáctico. Es un método clásico, fácil de visualizar y óptimo para volúmenes pequeños de datos.

#### **6.3. ShellSort (país + nombre ascendente)**

El ShellSort generaliza la inserción aplicando un *gap* o salto entre elementos, reduciéndolo progresivamente hasta llegar a 1. Este enfoque mejora el rendimiento frente a la inserción pura, con una complejidad promedio de  $O(n \log^2 n)$ . Se usó para ordenar alfabéticamente por país y, en caso de coincidencia, por nombre. Permite ordenar cadenas de texto con mayor eficiencia y demuestra dominio de un algoritmo más avanzado.

Los tres algoritmos trabajan directamente sobre la estructura Tenista[], cumpliendo el requisito de usar una matriz inicial y evitando librerías predeterminadas.

Cada uno demuestra un enfoque diferente (recursivo, iterativo y por saltos), evidenciando dominio de distintas estrategias de ordenamiento.

## **Capítulo 6**

### **Lista de Referencias**

Quirós, C. & Hidalgo, L. D. (2025). *Proyecto Tenistas – Repositorio de código fuente*.

GitHub. <https://github.com/camillaquiros2/ProyectoTenistas-EstructurasdeDatos.git>