DOCUMENTACIÓN
Proyecto: Inventario de Franelas de Fútbol

Integrantes: Ricardo Elbazi, Steven Alcalá, Jorge Fattal

Fecha: 2025-09-04

1. Ciclo de vida de la aplicación

La aplicación inicia en Main, instancia ProcessMain y ejecuta run(). El usuario puede crear o cargar un inventario. En el menú de gestión se permiten: consultas recursivas en memoria, agregados, reportes, guardado, buscador recursivo (Cola→Pila→Archivo), edición y eliminación. El flujo finaliza cuando el usuario decide salir.

1.1 Precondiciones generales

• Entradas válidas: nombres no vacíos y enteros ≥ 0.

• Estructuras inicializadas antes de operar (DataInitializer).

• Evitar valores nulos en estructuras dinámicas (Queue/Stack/List).

1.2 Postcondiciones generales

• Cambios en memoria pueden persistirse en archivos .txt con nombres estandarizados.

• El buscador deja traza persistida de resultados (archivo serialesArchivos_...).

• La carga de inventarios reconstruye estructuras consistentes.

1.3 Dependencias

• Lenguaje: Java SE 24 (sin preview).

• Librerías estándar: java.io, java.nio.file, java.util.

• Entorno: consola con permisos de lectura/escritura.

1.4 Diagrama de flujo (gráfico + ASCII

```
DIAGRAMA DE FLUJO (ALTO NIVEL)

Main → ProcessMain.run()
 ├─ [1] Crear inventario → DataInitializer + DataCollector →
DataManager.saveInventoryState()
 ├─ [2] Cargar inventario → DataManager.listAvailableInventories() + loadInventoryState()
 └─ Menú de Gestión
    ├─ Consultas (recursivo en memoria) → DataConsultant.runConsultMenu()
    ├─ Agregar liga → DataCollector.addLeague()
    ├─ Reporte completo → CompleteReportGenerator.generate()
    ├─ Guardar cambios → DataManager.saveInventoryState()
    ├─ Buscador (Cola→Pila→Archivo) → DataConsultant.runFileSearchQueueStack()
    ├─ Editar datos → DataEditor.runEditMenu()
    └─ Eliminar datos → DataRemover.runDeleteMenu()
```

1.5 Esquema de interacción de usuario

Usuario: selecciona opción → ingresa datos → confirma operación.

Sistema: valida → ejecuta → muestra resultados → persiste si aplica.

2. Tecnologías y prácticas implementadas

• Java SE 24, aplicación de consola.

• Patrones/prácticas: Encapsulación; Modularización por paquetes; DAO-like (DataManager); Factory-like (DataInitializer); Validación (validate).

• Estructuras dinámicas: SinglyLinkedList<T> como base de Queue<T> (FIFO) y Stack<T> (LIFO).

• Recursividad: no final (explorar directorios) y final (escaneo de líneas).

• Persistencia con nombres `inventory_state_YYYY-MM-DD_HHmmss_serial.txt` y `serialesArchivos_YYYY-MM-DD_serial.txt`.

```
ESTRUCTURAS DINÁMICAS

[Node<T>]
 - data:T
 - next:Node<T>

[SinglyLinkedList<T>]
 - head:Node<T>
 - tail:Node<T>
 - size:int
 + addFirst(T)
 + addLast(T)
 + removeFirst():T
 + isEmpty():boolean
 + size():int

[Queue<T>]
 - list:SinglyLinkedList<T>
 + enqueue(T)
 + dequeue():T
 + isEmpty():boolean
 + size():int

[Stack<T>]
 - list:SinglyLinkedList<T>
 + push(T)
 + pop():T
 + isEmpty():boolean
 + size():int
```

3. Detalles por módulo (firma / contrato de cada subrutina)

3.1 Paquete arr.ds – Estructuras de datos

arr.ds :: Node<T>

Nodo de lista enlazada simple (data, next).

- **Node(T data)**

  Precondición: data != null

  Postcondición: instancia válida

  Excepciones: IllegalArgumentException si data == null

arr.ds :: SinglyLinkedList<T>

Lista enlazada simple que soporta Queue y Stack.

- **addFirst(T v): void**

  Precondición: v != null

  Postcondición: v es nuevo head; size == size_prev + 1

  Excepciones: IllegalArgumentException si v == null

- **addLast(T v): void**

  Precondición: v != null

  Postcondición: v es nuevo tail; size == size_prev + 1

  Excepciones: IllegalArgumentException si v == null

- **removeFirst(): T**

  Precondición: !isEmpty()

  Postcondición: retorna head previo; size == size_prev - 1

  Excepciones: IllegalStateException si vacía

- **isEmpty(): boolean**

  Precondición: —

  Postcondición: retorna true si size == 0

  Excepciones: —

- **size(): int**

Precondición: —

Postcondición: retorna tamaño actual

Excepciones: —

arr.ds :: Queue<T>

Cola genérica FIFO sobre SinglyLinkedList.

- **enqueue(T v): void**

  Precondición: v != null

  Postcondición: v al final; size == size_prev + 1

  Excepciones: IllegalArgumentException si v == null

- **dequeue(): T**

  Precondición: !isEmpty()

  Postcondición: retorna/elimina frente; size == size_prev - 1

  Excepciones: IllegalStateException si vacía

- **isEmpty(): boolean**

  Precondición: —

  Postcondición: retorna true si size == 0

  Excepciones: —

- **size(): int**

  Precondición: —

  Postcondición: retorna tamaño actual

  Excepciones: —

arr.ds :: Stack<T>

Pila genérica LIFO sobre SinglyLinkedList.

- **push(T v): void**

  Precondición: v != null

  Postcondición: v en tope; size == size_prev + 1

  Excepciones: IllegalArgumentException si v == null

- **pop(): T**

  Precondición: !isEmpty()

  Postcondición: retorna/elimina tope; size == size_prev - 1

  Excepciones: IllegalStateException si vacía

- **isEmpty(): boolean**

  Precondición: —

  Postcondición: retorna true si size == 0

  Excepciones: —

- **size(): int**

  Precondición: —

  Postcondición: retorna tamaño actual

  Excepciones: —

3.2 Paquete arr.domain – TDAs del dominio

arr.domain :: League

TDA Liga con nombre.

- **League(String name)**

  Precondición: name no vacío

  Postcondición: instancia válida

  Excepciones: IllegalArgumentException si vacío

- **getName(): String**

  Precondición: —

  Postcondición: retorna nombre

  Excepciones: —

- **setName(String name): void**

  Precondición: name no vacío

Postcondición: nombre actualizado

Excepciones: IllegalArgumentException si vacío

arr.domain :: Team

TDA Equipo con nombre y liga.

- **Team(String name, String league)**

  Precondición: name/league no vacíos

  Postcondición: instancia válida

  Excepciones: IllegalArgumentException si vacío

- **getName(): String**

  Precondición: —

  Postcondición: retorna nombre

  Excepciones: —

- **setName(String name): void**

  Precondición: name no vacío

  Postcondición: nombre actualizado

  Excepciones: IllegalArgumentException si vacío

- **getLeague(): String**

  Precondición: —

  Postcondición: retorna liga

  Excepciones: —

arr.domain :: Player

TDA Jugador con nombre, equipo y stock.

- **Player(String name, String team, int stock)**

  Precondición: name/team no vacíos; stock ≥ 0

  Postcondición: instancia válida

  Excepciones: IllegalArgumentException si inválidos

- **getName(): String**

Precondición: —

Postcondición: retorna nombre

Excepciones: —

- **setName(String name): void**

  Precondición: name no vacío

  Postcondición: nombre actualizado

  Excepciones: IllegalArgumentException si vacío

- **getStock(): int**

  Precondición: —

  Postcondición: retorna stock

  Excepciones: —

- **setStock(int s): void**

  Precondición: s ≥ 0

  Postcondición: stock actualizado

  Excepciones: IllegalArgumentException si s < 0

arr.domain :: InventoryItem

Elemento del pipeline de búsqueda (FILE_HIT, LINE_HIT).

- **InventoryItem(Kind k, String summary)**

  Precondición: k != null; summary no vacío

  Postcondición: instancia válida

  Excepciones: IllegalArgumentException si inválidos

- **getKind(): Kind**

  Precondición: —

  Postcondición: retorna tipo

  Excepciones: —

- **getSummary(): String**

Precondición: —

Postcondición: retorna resumen

Excepciones: —

- **toString(): String**

  Precondición: —

  Postcondición: representación legible

  Excepciones: —

3.3 Paquete arr.io – Archivos

arr.io :: ArchiveUtil

TDA de archivos: directorios, E/S y generación de nombres.

- **ensureDirectory(String dir): void**

  Precondición: dir no vacío

  Postcondición: directorio existe (creado si no)

  Excepciones: IOException si falla FS

- **openWriter(String filename, boolean append): BufferedWriter**

  Precondición: ruta válida; dir existente

  Postcondición: escritor listo

  Excepciones: IOException

- **openReader(String filename): BufferedReader**

  Precondición: archivo existe y legible

  Postcondición: lector listo

  Excepciones: IOException

- **safeName(String base, String serial): String**

  Precondición: base/serial no vacíos

Postcondición: nombre `base_YYYY-MM-DD_HHmmss_serial.txt` (o similar por SO)

Excepciones: —

• **serialesName(String serial): String**

Precondición: serial no vacío

Postcondición: nombre `serialesArchivos_YYYY-MM-DD_serial.txt`

Excepciones: —

3.4 Paquete arr.helpers – Soporte/Persistencia

arr.helpers :: DataManager

Persistencia de inventarios: listar, guardar, cargar.

• **listAvailableInventories(): String[]**

Precondición: STORAGE_PATH válido

Postcondición: retorna lista de archivos disponibles

Excepciones: IOException si falla FS

• **saveInventoryState(String inv, String[] leagues, String[][] teams, String[][][] players, int[][][] availability): void**

Precondición: coherencia dimensional; nombres válidos

Postcondición: archivo guardado con nombre seguro

Excepciones: IllegalArgumentException, IOException

• **loadInventoryState(String file): InventoryData**

Precondición: archivo válido y legible

Postcondición: estructuras reconstruidas

Excepciones: IOException, IllegalStateException

arr.helpers :: InventoryData

DTO con estructuras cargadas/guardadas.

• **getLeagues(): String[]**

Precondición: —

Postcondición: retorna arreglo de ligas

Excepciones: —

- **getTeams(): String[][]**

  Precondición: —

  Postcondición: retorna matriz de equipos

  Excepciones: —

- **getPlayers(): String[][][]**

  Precondición: —

  Postcondición: retorna cubo de jugadores

  Excepciones: —

- **getAvailability(): int[][][]**

  Precondición: —

  Postcondición: retorna disponibilidad

  Excepciones: —

arr.helpers :: validate

Validaciones y utilidades de entrada/salida para menús.

- **valInt(String prompt, Scanner in): int**

  Precondición: Scanner activo

  Postcondición: retorna entero válido

  Excepciones: IllegalArgumentException si inválido

- **valName(String prompt, Scanner in): String**

  Precondición: Scanner activo

  Postcondición: retorna string no vacío

  Excepciones: IllegalArgumentException si vacío

- **utilDirectory(String dir): void**

  Precondición: dir no vacío

  Postcondición: asegura directorio

Excepciones: IOException si falla FS

- **useArchive(String name, String content, boolean append): void**

  Precondición: name no vacío

  Postcondición: archivo escrito

  Excepciones: IOException

- **nameArchiveGenerate(String base): String**

  Precondición: base no vacío

  Postcondición: retorna nombre estándar

  Excepciones: —

3.5 Paquete arr.consult – Consultas

arr.consult :: DataConsultant

Consultas recursivas en memoria y buscador por archivos con pipeline.

- **runConsultMenu(String[] leagues, String[][] teams, String[][][] players, int[][][] availability, java.util.Scanner in): void**

  Precondición: estructuras no nulas/coherentes

  Postcondición: consultas mostradas en consola

  Excepciones: IllegalArgumentException

- **runFileSearchQueueStack(java.util.Scanner in): void**

  Precondición: ruta base válida; término y extensión provistos

  Postcondición: resultados en consola y archivo seriales

  Excepciones: IOException, IllegalArgumentException

- **findFilesNonTail(java.io.File base, String ext, arr.ds.Queue[str], String serial): void**

  Precondición: base dir legible; ext válida

  Postcondición: enqueue de FILE_HIT por archivo candidato

  Excepciones: IOException

• **findLinesTailCI(java.util.List[str] lines, String term, int idx, arr.ds.Queue[int]): void**

Precondición: lines no nulo; term no vacío

Postcondición: enqueue de índices de coincidencia (LINE_HIT)

Excepciones: —

3.6 Paquete arr.process – Orquestación

arr.process :: ProcessMain

Ciclo de vida y menús.

• **run(): void**

Precondición: dependencias internas listas

Postcondición: flujo principal ejecutado

Excepciones: —

• **createNewInventory(): void**

Precondición: dimensiones válidas

Postcondición: estructuras creadas y (opcional) guardadas

Excepciones: IllegalArgumentException, IOException

• **loadAndManageInventory(): void**

Precondición: archivo válido disponible

Postcondición: inventario cargado y gestionado

Excepciones: IOException, IllegalStateException

• **runPostLoadMenu(): void**

Precondición: estructuras en memoria listas

Postcondición: opciones de gestión ejecutadas

Excepciones: —

arr.process :: DataCollector

Captura de datos por consola.

• **addLeague(...): void**

Precondición: entradas válidas

Postcondición: liga agregada

Excepciones: IllegalArgumentException

- **captureString(String prompt, java.util.Scanner in): String**

  Precondición: Scanner activo

  Postcondición: retorna string no vacío

  Excepciones: IllegalArgumentException

- **captureInt(String prompt, java.util.Scanner in): int**

  Precondición: Scanner activo

  Postcondición: retorna entero válido

  Excepciones: IllegalArgumentException

arr.process :: DataInitializer

Inicialización de estructuras base.

- **initDimensions(...): void**

  Precondición: dimensiones > 0

  Postcondición: dimensiones establecidas

  Excepciones: IllegalArgumentException

- **initArrays(...): void**

  Precondición: dimensiones válidas

  Postcondición: arreglos inicializados

  Excepciones: IllegalArgumentException

arr.process :: StatsCalculator

Cálculo de estadísticas derivadas.

- **calculateTotalsByTeam(int[][][] availability): int[]**

  Precondición: availability no nulo

  Postcondición: retorna totales por equipo

  Excepciones: IllegalArgumentException

- **totalsByLeague(int[][][] availability): int[]**

Precondición: availability no nulo

Postcondición: retorna totales por liga

Excepciones: IllegalArgumentException

arr.process :: ReportGenerator

Generación de reportes puntuales.

- **generate(...): void**

  Precondición: datos válidos; path accesible

  Postcondición: reporte generado

  Excepciones: IOException, IllegalArgumentException

arr.process :: CompleteReportGenerator

Generación de reporte completo.

- **generate(...): void**

  Precondición: datos válidos; path accesible

  Postcondición: reporte completo generado

  Excepciones: IOException, IllegalArgumentException

arr.process :: DataEditor

Edición en runtime (rename/ajustes).

- **runEditMenu(...): void**

  Precondición: estructuras listas

  Postcondición: cambios en memoria aplicados

  Excepciones: IllegalArgumentException

- **renameLeague(...): void**

  Precondición: índice válido; nombre no vacío

  Postcondición: liga renombrada

  Excepciones: IllegalArgumentException

- **renameTeam(...): void**

  Precondición: índices válidos; nombre no vacío

Postcondición: equipo renombrado

Excepciones: IllegalArgumentException

- **renamePlayer(...): void**

  Precondición: índices válidos; nombre no vacío

  Postcondición: jugador renombrado

  Excepciones: IllegalArgumentException

- **adjustStock(...): void**

  Precondición: índices válidos; stock ≥ 0

  Postcondición: stock actualizado

  Excepciones: IllegalArgumentException

arr.process :: DataRemover

Eliminación en cascada (jugador/equipo/liga).

- **runDeleteMenu(...): void**

  Precondición: estructuras listas

  Postcondición: eliminaciones aplicadas

  Excepciones: IllegalArgumentException

- **deletePlayer(...): void**

  Precondición: índices válidos

  Postcondición: jugador eliminado

  Excepciones: IllegalArgumentException

- **deleteTeam(...): void**

  Precondición: índices válidos

  Postcondición: equipo eliminado (cascada jugadores)

  Excepciones: IllegalArgumentException

- **deleteLeague(...): void**

  Precondición: índice válido

Postcondición: liga eliminada (cascada equipos/jugadores)

Excepciones: IllegalArgumentException

arr :: Main

Punto de entrada de la aplicación.

- **main(String[] args): void**

  Precondición: JRE disponible; consola

  Postcondición: instancia ProcessMain y llama run()

  Excepciones: —

4. Descripción del proyecto

Nombre: Inventario de Franelas de Fútbol.

Planteamiento del problema: la gestión manual de inventarios de camisetas de fútbol ocasiona errores, desactualización y pérdida de datos. La aplicación automatiza estas operaciones y asegura persistencia.

Análisis y pruebas: se eligieron estructuras dinámicas propias por requerimiento; se probó el pipeline Cola→Pila→Archivo con recursión no final/final; se validó persistencia y carga.

Objetivo general: gestionar inventarios dinámicamente con estructuras propias, recursividad y persistencia en archivos.

Objetivos específicos: implementar lista/cola/pila; buscador recursivo; edición/eliminación en runtime; reportes; nombres de archivo estandarizados.

Visión: sistema extensible a tiendas deportivas reales.

Misión: ofrecer exactitud y control del stock.

Alcance: consola; sin interfaz gráfica; persistencia en disco; compatible con Java 24.