**1) Explorador de ficheros (texto)**

**Enunciado.** Crea una app de consola que, dado un **directorio**, muestre:

* nº de archivos y carpetas
* tamaño total (solo archivos)
* los 5 archivos más grandes (nombre y tamaño)

**Pistas suaves.**

* File / Files, listFiles(), isFile(), length()
* Para ordenar: Comparator.comparingLong(File::length).reversed()

**Comprobación.** Prueba con una carpeta con 8–10 ficheros variados.

**2) Bloc de notas con historial (texto)**

**Enunciado.** Escribe un programa que:

* Añada líneas a un archivo notas.txt
* Liste las últimas **N** líneas
* Busque líneas que contengan un texto

**Pistas suaves.**

* Escritura: BufferedWriter + FileWriter(true)
* Lectura: BufferedReader.lines()
* Últimas N: guarda en List y toma subList

**Comprobación.** Añade 6–8 líneas y pide las últimas 3.

**3) Estadísticas de números (texto → parseo)**

**Enunciado.** Dado un archivo datos.csv con una columna de enteros por línea, calcula **mínimo, máximo, media y mediana**.

**Pistas suaves.**

* Files.lines(Path) → mapToInt(Integer::parseInt)
* Mediana: ordena; si par, media de los dos centrales.

**Comprobación.** Verifica con un set pequeño (p. ej., 1,2,3,4,100).

**4) Registro binario de sensores (binario)**

**Enunciado.** Diseña un fichero binario que almacene registros de sensor:

* long timestamp, int idSensor, double valor
* Escribe 100 registros simulados; luego léelos y calcula media por idSensor.

**Pistas suaves.**

* DataOutputStream / DataInputStream con el **mismo orden**.
* Agrupa con Map<Integer, List<Double>> o Map<Integer, DoubleSummaryStatistics>.

**Comprobación.** Compara nº registros escritos vs leídos.

**5) Agenda serializada (objetos)**

**Enunciado.** Implementa Contacto implements Serializable (nombre, email, móvil).

* Guarda List<Contacto> en binario.
* Carga, añade/borra un contacto, guarda de nuevo.

**Pistas suaves.**

* ObjectOutputStream / ObjectInputStream
* Marca serialVersionUID manualmente.

**Comprobación.** Verifica que tras cargar, el tamaño es el esperado.

**6) Fichero de acceso aleatorio (RandomAccessFile)**

**Enunciado.** Crea un fichero con **registros de tamaño fijo**:

* id (int), nombre (String fijo 20 chars), nota (double)
* Implementa: **insertar**, **leer por índice**, **actualizar nota por índice**.

**Pistas suaves.**

* Usa RandomAccessFile("rw")
* Para String fijo: rellena con espacios a 20 chars; seek(index \* tamRegistro)
* Calcula tamRegistro = 4 (int) + 20\*2 (char) + 8 (double) = 52 bytes

**Comprobación.** Inserta 3 registros, actualiza el 2º, vuelve a leer.

**7) Mini CRUD con JDBC (MySQL/MariaDB)**

**Enunciado.** Crea BD agenda con tabla contacto(id PK, nombre, email).

* Implementa consola con opciones: **insertar, listar, actualizar email, borrar por id**.

**Pistas suaves.**

* Connection, PreparedStatement, ResultSet
* try-with-resources
* URL típica: jdbc:mysql://localhost:3306/agenda

**Comprobación.** Inserta 3, lista, actualiza 1, borra 1, lista de nuevo.

**8) DAO básico (capa de acceso)**

**Enunciado.** Implementa:

* Contacto (DTO/POJO)
* ContactoDAO (interfaz): insertar, buscarPorId, buscarPorEmail, listarTodos, actualizar, borrar
* ContactoDAOImpl con JDBC (usa PreparedStatement y **no** SQL en Statement)
* ConnectionManager / Conexion estática

**Pistas suaves.**

* Validar unicidad de email con SELECT COUNT(\*) previo.
* Maneja SQLException y devuelve Optional<Contacto> en búsquedas.

**Comprobación.** Tests manuales desde main.

**9) Exportador/importador (CSV ↔ JDBC)**

**Enunciado.** Añade a la app del ejercicio 7:

* Exportar todos los contactos a contactos.csv
* Importar desde contactos.csv evitando duplicados por **email**

**Pistas suaves.**

* Lectura CSV: Files.lines + split(",") o Scanner con delimitador
* Evitar duplicados: SELECT por email antes de INSERT

**Comprobación.** Exporta, borra la tabla, importa, lista.

**10) Transacciones y atomicidad (JDBC)**

**Enunciado.** Simula una operación **transferencia** entre dos cuentas:

* Tabla cuenta(id PK, titular, saldo DECIMAL)
* Método transferir(origen, destino, cantidad) que:
  + resta al origen
  + suma al destino
  + **todo en una transacción** (commit/rollback)

**Pistas suaves.**

* conn.setAutoCommit(false)
* try con commit() en el final, catch con rollback()
* Comprueba saldo suficiente

**Comprobación.** Fuerza un error entre las dos operaciones y verifica que no hay cambios (rollback).

**¿Cómo trabajamos?**

* Elige **uno** y te guío **paso a paso**: primero diseño, luego pseudo-código, y por último **trozos de código** (no la solución entera de golpe).
* Si te atascas, te doy **una pista** más concreta o un **test** para verificar.