1. 🧰 La Preparación: Configurar las Herramientas

Antes de hacer nada, siempre preparan las "herramientas" de trabajo.

1. **Crear el "Cerebro" (`ObjectMapper`):** En todos los archivos, lo primero que se crea (o se define) es un objeto `ObjectMapper`. Piensa en esto como **la herramienta principal** que sabe cómo leer y escribir JSON.
   a. `private static ObjectMapper mapper = new ObjectMapper();`
2. **Enseñar a leer Fechas (`JavaTimeModule`):** Por defecto, el `ObjectMapper` no entiende las fechas modernas de Java (como `LocalDate`). Los programadores le "enseñan" a hacerlo registrando un módulo especial.
   a. `.registerModule(new JavaTimeModule());`
3. **Preparar la Red de Seguridad (`try-catch`):** Como están trabajando con archivos (que pueden no existir o estar corruptos), **siempre** envuelven el código de lectura o escritura en un bloque `try-catch` para capturar `IOException`. Esto evita que el programa se rompa si el archivo falla.

2. 📥 Leer JSON (Convertir de JSON a Objetos Java)

Este es el proceso de "Deserialización". El objetivo es leer un archivo de texto JSON y convertirlo en objetos que Java entienda (como `Equipo`, `Pedidos`, etc.).

1. **Revisar que el Archivo Existe:** Antes de intentar leer, comprueban si el archivo se puede leer.
   a. `if (!ficheroJson.canRead()) { ... }`
2. **Cargar el Archivo en Memoria:** Usan el `ObjectMapper` para leer el archivo. Hay dos formas comunes que has mostrado:
   a. **Opción A (Archivos 1 y 2):** Leerlo como un "árbol" de nodos (`JsonNode`). Esto te da un objeto genérico que te permite "preguntar" por cada campo.
      i. `JsonNode nodoRaiz = mapper.readTree(ficheroJson);`
   b. **Opción B (Archivo 3):** Leerlo como un `Map` genérico de Java.
      i. `var jsonData = mapper.readValue(ficheroJson, Map.class);`
3. **Navegar a los Datos Importantes:** El JSON casi siempre tiene una estructura (como `{ "resultados": [...] }`). El siguiente paso es "entrar" a esa lista principal.
   a. `JsonNode nodoResultados = nodoRaiz.get("resultados");`
   b. `var pedidosArray = (List<Map<String, Object>>) jsonData.get("pedidos");`

4. **Recorrer (Iterar) el Array:** Como los datos suelen venir en una lista (`[...]`), usan un bucle `for` para procesar cada elemento (cada partido, cada pedido) uno por uno.
    a. `for (JsonNode resultado : nodoResultados) { ... }`
    b. `for (Map<String, Object> pedidoData : pedidosArray) { ... }`
5. **Extraer los Datos y Crear Objetos Java:** Dentro del bucle, hacen dos cosas:
    a. Sacan los valores de cada campo (ej. "HomeTeam", "FTHG", "id_producto").
        i. `String equipoLocal = resultado.get("HomeTeam").asText();`
        ii. `int golesLocal = resultado.get("FTHG").asInt();`
    b. Usan esos valores para crear una instancia de su clase Java (un POJO).
        i. `Resultado resLocal = new Resultado(...);`
        ii. `Pedidos pedido = mapper.convertValue(pedidoData, Pedidos.class);` (Esta es una forma "automática" muy útil que se ve en el archivo 3).

3. 📤 Escribir JSON (Convertir de Objetos Java a JSON)

Este es el proceso de "Serialización". Es el paso inverso: cogen un objeto Java (como `Equipo` o `InformeSalida`) y lo convierten en un archivo de texto JSON.

Este es **el paso más sencillo y repetido** en todos los archivos:

1. **Preparar el Formato Bonito:** Usan `.writerWithDefaultPrettyPrinter()` para asegurarse de que el JSON que escriban esté bien formateado (indentado, legible por humanos) y no todo en una sola línea.
2. **Escribir el Archivo:** Llaman al método `writeValue()`, pasándole el archivo de salida y el objeto Java que quieren convertir.
    a. `mapper.writerWithDefaultPrettyPrinter().writeValue(archivoSalida, equipo);`
    b. `mapper.writerWithDefaultPrettyPrinter().writeValue(archivoSalida, informe);`

Resumen (La "Chuleta")

Si te quedas con lo esencial, los pasos que se repiten siempre son:

1. **Configurar:** Crear el `ObjectMapper` (y añadir `JavaTimeModule` si hay fechas).
2. **Leer (Opción A - Manual):** `mapper.readTree()` -> `.get()` -> `.asText()` / `.asInt()` -> `new Objeto()`.
3. **Leer (Opción B - Automática):** `mapper.readValue()` o `mapper.convertValue()`.
4. **Escribir:** `mapper.writerWithDefaultPrettyPrinter().writeValue()`.
5. **Seguridad:** Envolver todo lo que lee o escribe archivos en un `try-catch`.