Code & Learn (Manipulación de ficeros con Jackson (xml -Json))

Vamos a detallar como construir una API con **Spring Boot** que gestione **ficheros y directorios** usando `java.nio.file`, persista datos en **JSON** o **XML** con **Jackson**, implemente **CRUD** sobre ficheros, realice **conversión entre formatos**, y maneje **excepciones**, además de pautas para la **documentación de aplicaciones**.

Objetivo didáctico de esta unidad será

- Utilización de **clases para la gestión de ficheros y directorios**.
- **Ventajas e inconvenientes** de distintos **formatos de acceso** a ficheros.
- **Recuperación y manipulación** de información almacenada en ficheros (**CRUD**).
- **Conversión** entre formatos de ficheros (**JSON** ⇄ **XML**).
- **Gestión de excepciones**.
- **Documentación** de aplicaciones (Javadoc).

🧰 Stack y requisitos

- **Java 17+**
- **Spring Boot 3.x** (`spring-boot-starter-web`, `spring-boot-starter-validation`)
- **Jackson** (`jackson-databind`, `jackson-dataformat-xml`)
- **Maven**

Maven (pom.xml — dependencias clave):

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
```

```
  </dependency>
</dependencies>
```

🗂️ Estructura propuesta del proyecto

```
src/
 ├─ main/java/com/docencia/files/
 │    ├─ FilesConfig.java
 │    ├─ model/Note.java
 │    ├─ repo/FileNoteRepository.java
 │    ├─ service/NoteService.java
 │    ├─ web/NoteController.java
 │    ├─ web/GlobalExceptionHandler.java
 │    └─ util/FormatConverter.java
 ├─ main/resources/
 │
 └─ test/java/... (tests)
```

📁 Clases para gestión de ficheros y directorios (java.nio.file)

Clases clave y ejemplos:

- `Path`, `Paths`: representación inmutable de rutas.
- `Files`: operaciones de alto nivel (crear, leer, escribir, copiar, mover, borrar, stream).
- `DirectoryStream`: recorrer directorios eficientemente.
- `FileVisitor`/`SimpleFileVisitor`: recorrer árbol de directorios.
- `FileTime`: metadatos (última modificación).
- `StandardOpenOption`/`StandardCopyOption`: opciones de apertura/copia.
- `FileChannel`/`MappedByteBuffer`: acceso avanzado (memoria mapeada).

Ejemplos rápidos:

```
Path root = Paths.get(storageRoot);
Files.createDirectories(root);

Path notePath = root.resolve(id + "." + ext);
boolean exists = Files.exists(notePath);

byte[] content = Files.readAllBytes(notePath);
Files.writeString(notePath, json, StandardOpenOption.CREATE,
StandardOpenOption.TRUNCATE_EXISTING);

try (DirectoryStream<Path> stream = Files.newDirectoryStream(root, "*.
{json,xml}")) {
  for (Path p : stream) { /* ... */ }
}
```

⚙️ Formatos de acceso a ficheros: ventajas e inconvenientes

Formato de acceso	Ventajas	Inconvenientes / Cuándo evitar
Secuencial (streams)	Simple, bajo consumo de memoria, ideal para ficheros grandes.	Acceso aleatorio costoso; relecturas completas para pequeñas ediciones.
Buffered (BufferedInput/OutputStream, Reader/Writer)	Menos I/O físico, mejora rendimiento.	Buffer tuning necesario en cargas inusuales.
Aleatorio (RandomAccessFile)	Lectura/escritura en posiciones específicas.	API más compleja; sincronización en concurrencia.
Canales/NIO (`FileChannel`)	Transferencias rápidas (zero-copy), soporte `lock`, memory-mapping.	Curva de aprendizaje; cuidado con `MappedByteBuffer` y GC.
Memoria mapeada	Máximo rendimiento en lectura aleatoria; útil para grandes ficheros.	Gestión de recursos delicada; no portable a FS remotos.

📦 Modelo de datos y serialización (Jackson JSON & XML)

`Note.java` **(Bean (clase) con validación):**

```java
package com.docencia.files.model;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;

public class Note {

  @NotBlank
  private String id;

  @NotBlank @Size(max = 200)
  private String title;

  @NotBlank
  private String content;

  // getters/setters/constructores
}
```

🧱 Conversión entre formatos (JSON ⇄ XML)

`FormatConverter.java`:

```
package com.docencia.files.util;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.dataformat.xml.XmlMapper;

public class FormatConverter {
  private final ObjectMapper json;
  private final XmlMapper xml;

  public FormatConverter(ObjectMapper json, XmlMapper xml) {
    this.json = json; this.xml = xml;
  }

  public <T> String jsonToXml(String jsonString, Class<T> type) throws
Exception {
    T obj = json.readValue(jsonString, type);
    return xml.writeValueAsString(obj);
  }

  public <T> String xmlToJson(String xmlString, Class<T> type) throws
Exception {
    T obj = xml.readValue(xmlString, type);
    return json.writeValueAsString(obj);
  }
}
```

Repositorios (JSON/XML)

NoteRepository.java

```
package com.formacion.files;

import java.util.List;

public interface NoteRepository {
    Note save(Note note);            // crea/actualiza por id
    List<Note> findById(String id);  // devuelve 0..1 elementos
    List<Note> findAll();
    boolean deleteById(String id);
}
```

JsonNoteRepository.java

```
package com.formacion.files;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import org.springframework.beans.factory.annotation.Qualifier;
```

```
import org.springframework.stereotype.Repository;

import java.io.IOException;
import java.nio.file.*;
import java.util.*;
import java.util.concurrent.locks.ReentrantReadWriteLock;

@Repository
@Qualifier("jsonNoteRepo")
public class JsonNoteRepository implements NoteRepository {

    private final Path path;
    private final ObjectMapper mapper;
    private final ReentrantReadWriteLock lock = new
ReentrantReadWriteLock();

    public JsonNoteRepository(StorageProperties props) {
        this.path = Paths.get(props.getJsonPath());
        this.mapper = new ObjectMapper().registerModule(new
JavaTimeModule());
        ensureFile();
    }

    @Override
    public Note save(Note note) {
        lock.writeLock().lock();
        try {
            List<Note> all = readAllInternal();
            if (note.getId() == null || note.getId().isBlank()) {
                note.setId(UUID.randomUUID().toString());
            }
            all.removeIf(n -> Objects.equals(n.getId(), note.getId()));
            all.add(note);
            writeAllInternal(all);
            return note;
        } finally {
            lock.writeLock().unlock();
        }
    }

    @Override
    public List<Note> findById(String id) {
        lock.readLock().lock();
        try {
            return readAllInternal().stream()
                    .filter(n -> Objects.equals(n.getId(), id))
                    .findFirst().map(List::of)
                    .orElseGet(List::of);
        } finally {
            lock.readLock().unlock();
        }
    }

    @Override
```

```
    public List<Note> findAll() {
        lock.readLock().lock();
        try {
            return Collections.unmodifiableList(readAllInternal());
        } finally {
            lock.readLock().unlock();
        }
    }

    @Override
    public boolean deleteById(String id) {
        lock.writeLock().lock();
        try {
            List<Note> all = readAllInternal();
            boolean removed = all.removeIf(n -> Objects.equals(n.getId(),
id));
            if (removed) writeAllInternal(all);
            return removed;
        } finally {
            lock.writeLock().unlock();
        }
    }

    // ------- helpers -------
    private void ensureFile() {
        try {
            Files.createDirectories(path.getParent());
            if (!Files.exists(path)) {
                writeAllInternal(new ArrayList<>());
            }
        } catch (IOException e) {
            throw new RuntimeException("No se pudo inicializar el fichero
JSON", e);
        }
    }

    private List<Note> readAllInternal() {
        try {
            if (!Files.exists(path) || Files.size(path) == 0) return new
ArrayList<>();
            Note[] arr = mapper.readValue(Files.readAllBytes(path),
Note[].class);
            return new ArrayList<>(Arrays.asList(arr));
        } catch (IOException e) {
            throw new RuntimeException("Error leyendo JSON", e);
        }
    }

    private void writeAllInternal(List<Note> items) {
        try {
            byte[] bytes =
mapper.writerWithDefaultPrettyPrinter().writeValueAsBytes(items);
            Files.write(path, bytes,
                    StandardOpenOption.CREATE,
```

```
                    StandardOpenOption.TRUNCATE_EXISTING,
                    StandardOpenOption.WRITE);
        } catch (IOException e) {
            throw new RuntimeException("Error escribiendo JSON", e);
        }
    }
}
```

XmlNoteRepository.java

```
package com.formacion.files;

import com.fasterxml.jackson.dataformat.xml.XmlMapper;
import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Repository;

import java.io.IOException;
import java.nio.file.*;
import java.util.*;
import java.util.concurrent.locks.ReentrantReadWriteLock;

@Repository
@Qualifier("xmlNoteRepo")
public class XmlNoteRepository implements NoteRepository {

    private final Path path;
    private final XmlMapper mapper;
    private final ReentrantReadWriteLock lock = new
ReentrantReadWriteLock();

    public XmlNoteRepository(StorageProperties props) {
        this.path = Paths.get(props.getXmlPath());
        this.mapper = (XmlMapper) new XmlMapper().registerModule(new
JavaTimeModule());
        ensureFile();
    }

    @Override
    public Note save(Note note) {
        lock.writeLock().lock();
        try {
            List<Note> all = readAllInternal();
            if (note.getId() == null || note.getId().isBlank()) {
                note.setId(UUID.randomUUID().toString());
            }
            all.removeIf(n -> Objects.equals(n.getId(), note.getId()));
            all.add(note);
            writeAllInternal(all);
            return note;
        } finally {
```

```
            lock.writeLock().unlock();
        }
    }

    @Override
    public List<Note> findById(String id) {
        lock.readLock().lock();
        try {
            return readAllInternal().stream()
                    .filter(n -> Objects.equals(n.getId(), id))
                    .findFirst().map(List::of)
                    .orElseGet(List::of);
        } finally {
            lock.readLock().unlock();
        }
    }

    @Override
    public List<Note> findAll() {
        lock.readLock().lock();
        try {
            return Collections.unmodifiableList(readAllInternal());
        } finally {
            lock.readLock().unlock();
        }
    }

    @Override
    public boolean deleteById(String id) {
        lock.writeLock().lock();
        try {
            List<Note> all = readAllInternal();
            boolean removed = all.removeIf(n -> Objects.equals(n.getId(),
id));
            if (removed) writeAllInternal(all);
            return removed;
        } finally {
            lock.writeLock().unlock();
        }
    }

    // ------- helpers -------
    private void ensureFile() {
        try {
            Files.createDirectories(path.getParent());
            if (!Files.exists(path)) {
                writeAllInternal(new ArrayList<>());
            }
        } catch (IOException e) {
            throw new RuntimeException("No se pudo inicializar el fichero
XML", e);
        }
    }
```

```
    private List<Note> readAllInternal() {
        try {
            if (!Files.exists(path) || Files.size(path) == 0) return new
ArrayList<>();
            NoteStore store = mapper.readValue(Files.readAllBytes(path),
NoteStore.class);
            return new ArrayList<>(store.getItems());
        } catch (IOException e) {
            throw new RuntimeException("Error leyendo XML", e);
        }
    }

    private void writeAllInternal(List<Note> items) {
        try {
            NoteStore store = new NoteStore(items);
            byte[] bytes =
mapper.writerWithDefaultPrettyPrinter().writeValueAsBytes(store);
            Files.write(path, bytes,
                    StandardOpenOption.CREATE,
                    StandardOpenOption.TRUNCATE_EXISTING,
                    StandardOpenOption.WRITE);
        } catch (IOException e) {
            throw new RuntimeException("Error escribiendo XML", e);
        }
    }
}
```

Router de repositorios

RoutedNoteRepository.java

```
package com.formacion.files;

import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Repository;

import java.util.List;

@Primary
@Repository
public class RoutedNoteRepository {

    private final NoteRepository jsonRepo;
    private final NoteRepository xmlRepo;

    public RoutedNoteRepository(@Qualifier("jsonNoteRepo") NoteRepository
jsonRepo,
                                @Qualifier("xmlNoteRepo") NoteRepository
xmlRepo) {
        this.jsonRepo = jsonRepo;
```

```
        this.xmlRepo = xmlRepo;
    }

    private NoteRepository select(FileFormat format) {
        return (format == FileFormat.XML) ? xmlRepo : jsonRepo;
    }

    public Note save(Note note, FileFormat format) {
        return select(format).save(note);
    }

    public List<Note> findById(String id, FileFormat format) {
        return select(format).findById(id); // 0..1 elementos
    }

    public List<Note> findAll(FileFormat format) {
        return select(format).findAll();
    }

    public boolean deleteById(String id, FileFormat format) {
        return select(format).deleteById(id);
    }
}
```

Servicio

NoteService.java

```
package com.formacion.files;

import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class NoteService {

    private final RoutedNoteRepository repo;

    public NoteService(RoutedNoteRepository repo) {
        this.repo = repo;
    }

    public Note crear(Note note, FileFormat formato) {
        return repo.save(note, formato);
    }

    public List<Note> obtenerPorId(String id, FileFormat formato) {
        return repo.findById(id, formato); // lista vacía o de 1 elemento
    }
```

```
    public List<Note> listar(FileFormat formato) {
        return repo.findAll(formato);
    }

    public boolean eliminar(String id, FileFormat formato) {
        return repo.deleteById(id, formato);
    }
}
```

NoteController.java:

```
package com.docencia.files.web;

import com.docencia.files.model.Note;
import com.docencia.files.service.NoteService;
import jakarta.validation.Valid;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.util.List;

@Controller
public class NoteController {
  //implementar
}
```

Gestión de excepciones

GlobalExceptionHandler.java:

```
package com.docencia.files.exceptios;

import org.springframework.http.HttpStatus;
import org.springframework.http.ProblemDetail;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import java.nio.file.NoSuchFileException;

@ControllerAdvice
public class GlobalExceptionHandler {

  @ExceptionHandler(NoSuchFileException.class)
  public ProblemDetail notFound(NoSuchFileException ex) {
    ProblemDetail pd = ProblemDetail.forStatus(HttpStatus.NOT_FOUND);
    pd.setTitle("Recurso no encontrado");
    pd.setDetail(ex.getMessage());
    return pd;
```

```
    }

    @ExceptionHandler(IllegalStateException.class)
    public ProblemDetail conflict(IllegalStateException ex) {
      ProblemDetail pd = ProblemDetail.forStatus(HttpStatus.CONFLICT);
      pd.setTitle("Conflicto de estado");
      pd.setDetail(ex.getMessage());
      return pd;
    }

    @ExceptionHandler(Exception.class)
    public ProblemDetail generic(Exception ex) {
      ProblemDetail pd =
  ProblemDetail.forStatus(HttpStatus.INTERNAL_SERVER_ERROR);
      pd.setTitle("Error interno");
      pd.setDetail(ex.getMessage());
      return pd;
    }
  }
```

Buenas prácticas:

- Validar entrada (`@Valid` + anotaciones en el modelo).
- Usar `ProblemDetail` (Spring 6) para respuestas de error consistentes.
- Registrar logs con `Slf4j` (no mostrado por brevedad).
- En concurrencia, considerar bloqueos a nivel de fichero (`FileChannel#lock`) o estrategias de nombrado temporal + `move` atómico.