Resumen Adicional Ficheros

Java proporciona varias formas de trabajar con archivos y directorios. Las dos principales APIs son:

- `java.io`: la API tradicional, disponible desde las primeras versiones de Java.
- `java.nio`: introducida en Java 7 con mejoras importantes, especialmente en rendimiento y facilidad de uso.

Manejo de rutas (crear, borrar, renombrar, mover, obtener atributos)

java.io

En la API tradicional, `java.io`, todas las operaciones sobre rutas se realizan con la clase `File`.

Para usarla, se puede crear un objeto de esta clase, indicando la ruta a manejar en el constructor

```
File ruta = new File("/ruta/a/manejar");
```

Luego, se pueden usar los métodos del objeto `File` para realizar las operaciones deseadas:

```
ruta.mkdir();               // crea un directorio
ruta.mkdirs();              // crea directorios intermedios si no existen
ruta.delete();              // elimina el archivo o directorio (si está vacío)
ruta.renameTo(new File("/nueva/ruta")); // renombrar o mover
ruta.list();                // listar el contenido del directorio
ruta.exists();              // comprobar existencia
ruta.length();              // tamaño en bytes
ruta.lastModified();        // fecha de última modificación
ruta.setLastModified(...); // cambiar fecha de modificación
```

⚠ `java.io.File` no lanza excepciones para la mayoría de operaciones: simplemente devuelve `true` o `false` según el éxito, lo cual puede dificultar el manejo de errores.

java.nio

En la **nueva** API, `java.nio`, las operaciones se realizan usando las clases `Path` y `Files`, que proporcionan una forma más moderna de trabajar con el sistema de archivos.

La clase `Path` sirve para definir rutas, y la clase `Files` para realizar operaciones sobre ellas.

Los objetos de clase Path se crean con el método *factory* `of`:

```
Path ruta = Path.of("/ruta/a/manejar");
```

Luego, se pueden realizar operaciones sobre dicha **ruta** con los métodos estáticos de la clase `Files`:

```
Files.createDirectory(ruta);         // crea un directorio
Files.createDirectories(ruta);       // crea todos los directorios necesarios
Files.createFile(ruta.resolve("archivo.txt")); // crea un archivo
Files.delete(ruta);                  // elimina archivo o directorio (si está vacío)
Files.deleteIfExists(ruta);          // igual que el anterior, pero no lanza excepción si no existe
Files.move(origen, Path.of("/ruta/nueva.txt"), StandardCopyOption.REPLACE_EXISTING);

BasicFileAttributes attrs = Files.readAttributes(ruta, BasicFileAttributes.class);
System.out.println("Tamaño: " + attrs.size());
System.out.println("Creado: " + attrs.creationTime());
System.out.println("Última modificación: " + attrs.lastModifiedTime());
Files.setLastModifiedTime(ruta, ...);
```

✔️ La API `java.nio` lanza excepciones, lo que permite manejar mejor los errores. Además, permite acceder a muchos más metadatos que `java.io`.

Lectura/escritura en ficheros

Para leer/escribir Strings en un fichero podemos hacer uso de estos métodos estáticos de la clase `Files`:

🪚 readString

Lee **todo el contenido del fichero** como un único `String`.

```
String contenido = Files.readString(ruta);
System.out.println(contenido);
```

🪧 readAllLines

Lee el contenido del fichero **línea por línea**, devolviendo una `List<String>`

```
List<String> lineas = Files.readAllLines(ruta);
for (String linea : lineas) {
   System.out.println(linea);
}
```

🪧 writeString

Escribe texto en un fichero, **sobrescribiendo su contenido por defecto**:

```
Files.writeString(ruta, "Hola, mundo!");
```

Si quieres modificar el comportamiento de la escritura (por ejemplo, añadir en lugar de sobrescribir), puedes usar **opciones adicionales** mediante `StandardOpenOption`.

```
Files.writeString(ruta, "Nueva línea\n",
       StandardOpenOption.CREATE,           // Crea el archivo si no existe
       StandardOpenOption.APPEND            // Añade contenido al final
);
```

Las opciones disponibles son:

`CREATE`	Crea el archivo si no existe.
`CREATE_NEW`	Crea el archivo, pero lanza excepción si ya existe.
`APPEND`	Añade el contenido al final del archivo existente.
`WRITE`	Abre el archivo para escritura (implícito en muchos casos).
`TRUNCATE_EXISTING`	Borra el contenido existente antes de escribir (por defecto si no usas `APPEND`).
📝 Si no se especifica ninguna opción, por defecto se usa `WRITE` y `TRUNCATE_EXISTING`.	

✅ Todos estos métodos pueden lanzar excepciones como `IOException`, por lo que deben usarse dentro de bloques `try-catch` o declarar `throws IOException`.

 Scanner

Para **leer** contenido de un fichero también se puede hacer uso de la clase Scanner, aunque en lugar de `System.in`, proporcionaremos la ruta al fichero del cual queremjos leer:

```
Scanner scanner = new Scanner(Path.of("/ruta/al/fichero"));
```

Luego, podemos hacer uso de los métodos conocidos `next()`, `nextLine()`, `nextInt()`, etcétera...