Gestión de archivos
Manipulación de Archivos en Java

En Java, trabajar con archivos implica leer, escribir, y modificar datos en ellos. La **manipulación de archivos** es fundamental para desarrollar aplicaciones que interactúan con sistemas de almacenamiento. Java proporciona varias clases para trabajar con archivos, tanto en formato de texto como binario.

1. Clases para manipular archivos

Java tiene varias clases dentro del paquete `java.io` que nos permiten manejar archivos. Algunas de las más utilizadas son:

1.1. Clase `File`

La clase `File` se usa para representar archivos y directorios. Con esta clase, puedes crear, eliminar, renombrar y obtener información sobre archivos.

- **Creación de un archivo:**

```
File file = new File("archivo.txt");
if (!file.exists()) {
    file.createNewFile();
}
```

- **Comprobación de existencia de un archivo:**

```
if (file.exists()) {
    System.out.println("El archivo existe");
}
```

- **Eliminación de un archivo:**

```
    boolean deleted = file.delete();
```

- **Obtener la ruta absoluta:**
  `String path = file.getAbsolutePath();`

1.2. Clases para Lectura y Escritura de Archivos `FileReader` y `BufferedReader` (lectura de texto)

Estas clases permiten leer archivos de texto.

- `FileReader`: Lee caracteres de un archivo. Se usa generalmente para archivos de texto sin codificación especial.

```
FileReader reader = new FileReader("archivo.txt");
int data;
while ((data = reader.read()) != -1) {
    System.out.print((char) data);
}
reader.close();
```

- **`BufferedReader`**: Lee texto de manera más eficiente, almacenando los datos en un búfer. Es muy útil cuando necesitas leer líneas completas de un archivo.

```
BufferedReader br = new BufferedReader(new FileReader("archivo.txt")); String
line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
br.close();
```

`FileWriter` y **`BufferedWriter`** (escritura de texto)

Permiten escribir texto en un archivo.

- **`FileWriter`**: Escribe caracteres en un archivo. Si el archivo no existe, lo crea.

```
FileWriter writer = new FileWriter("archivo.txt");
writer.write("Hola Mundo!");
writer.close();
```

- **`BufferedWriter`**: Escribe texto de manera eficiente, similar a `BufferedReader`.

```
BufferedWriter bw = new BufferedWriter(new FileWriter("archivo.txt"));
bw.write("Hola Mundo!");
bw.newLine();  // Escribe una nueva línea
bw.close();
```

`PrintWriter` (escritura formateada)

Es una clase más conveniente para escribir texto formateado, especialmente cuando quieres imprimir directamente en un archivo.

```
PrintWriter pw = new PrintWriter(new FileWriter("archivo.txt"));
pw.println("Hola Mundo");
```

```
pw.println("Java es genial!");
pw.close();
```

1.3. Lectura y escritura de archivos binarios

- **`FileInputStream`** y **`FileOutputStream`**: Usados para leer y escribir archivos binarios (por ejemplo, imágenes o archivos de audio).

  Lectura de archivo binario:

```
FileInputStream fis = new FileInputStream("archivo.bin");
int data;
while ((data = fis.read()) != -1) {
    System.out.print(data + " ");
}
fis.close();
```

Escritura de archivo binario:

```
FileOutputStream fos = new FileOutputStream("archivo.bin");
fos.write(65);  // Escribe el byte correspondiente a 'A'
fos.close();
```

2. Manejo de Excepciones

El manejo de excepciones es crucial cuando trabajas con archivos. La mayoría de las operaciones con archivos pueden lanzar excepciones como `IOException` o `FileNotFoundException`. Asegúrate de capturarlas usando un bloque `try-catch`.

```
try {
    FileReader fr = new FileReader("archivo.txt");
    BufferedReader br = new BufferedReader(fr);
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
    br.close();
} catch (IOException e) {
    System.out.println("Hubo un error al leer el archivo: " +
e.getMessage());
}
```

3. Acceso aleatorio a archivos: `RandomAccessFile`

La clase `RandomAccessFile` es una opción poderosa para leer y escribir datos en cualquier parte de un archivo. Permite un acceso no secuencial.

- **Lectura de un archivo con `RandomAccessFile`**:

```
RandomAccessFile raf = new RandomAccessFile("archivo.bin", "r");
raf.seek(10);  // Mueve el puntero de lectura a la posición 10
int data = raf.read();
System.out.println((char) data);
raf.close();
```

- **Escritura en un archivo con `RandomAccessFile`**:

```
RandomAccessFile raf = new RandomAccessFile("archivo.bin", "rw");
raf.seek(10);  // Mueve el puntero de escritura a la posición 10 r
af.write(65); // Escribe el byte correspondiente a 'A'
raf.close();
```

- **Lectura y escritura en una posición específica**:

```
RandomAccessFile raf = new RandomAccessFile("archivo.bin", "rw");
raf.writeInt(100);  // Escribe un entero en la posición actual
raf.seek(0);// Vuelve al principio
int value = raf.readInt();  // Lee el entero que se escribió
System.out.println(value);
raf.close();
```

4. Clases auxiliares para manipulación de archivos

- **`Files` (Java 7 y superior)**: Una clase dentro del paquete `java.nio.file` que proporciona métodos útiles para trabajar con archivos y directorios.
  - **Copiar un archivo**:

```
Path source = Paths.get("source.txt");
Path destination = Paths.get("destination.txt");
Files.copy(source, destination,
StandardCopyOption.REPLACE_EXISTING);
```

  - **Mover un archivo**:

```
Path source = Paths.get("source.txt");
Path destination = Paths.get("destination.txt");
Files.move(source, destination,
StandardCopyOption.REPLACE_EXISTING);
```

- **Leer todo el contenido de un archivo**:

```
Path path = Paths.get("archivo.txt");
try {
  List<String> lines = Files.readAllLines(path);
  lines.forEach(System.out::println);
} catch (IOException e) {
  e.printStackTrace();
}
```

---

5. Consejos y Buenas Prácticas

- **Cerrar los streams**: Siempre cierra los flujos de lectura/escritura al terminar, ya que de lo contrario, puede haber pérdidas de datos o errores en el sistema de archivos.
- **Uso de `try-with-resources`**: Puedes usar esta estructura para manejar automáticamente el cierre de recursos.

```
try (
  BufferedReader br = new BufferedReader(
        new FileReader("archivo.txt")
  )
) {
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

- **Usa rutas absolutas o relativas según sea necesario**: Si no se encuentra el archivo, asegúrate de que la ruta proporcionada es correcta, ya sea absoluta o relativa.