

Programación I

# Archivos

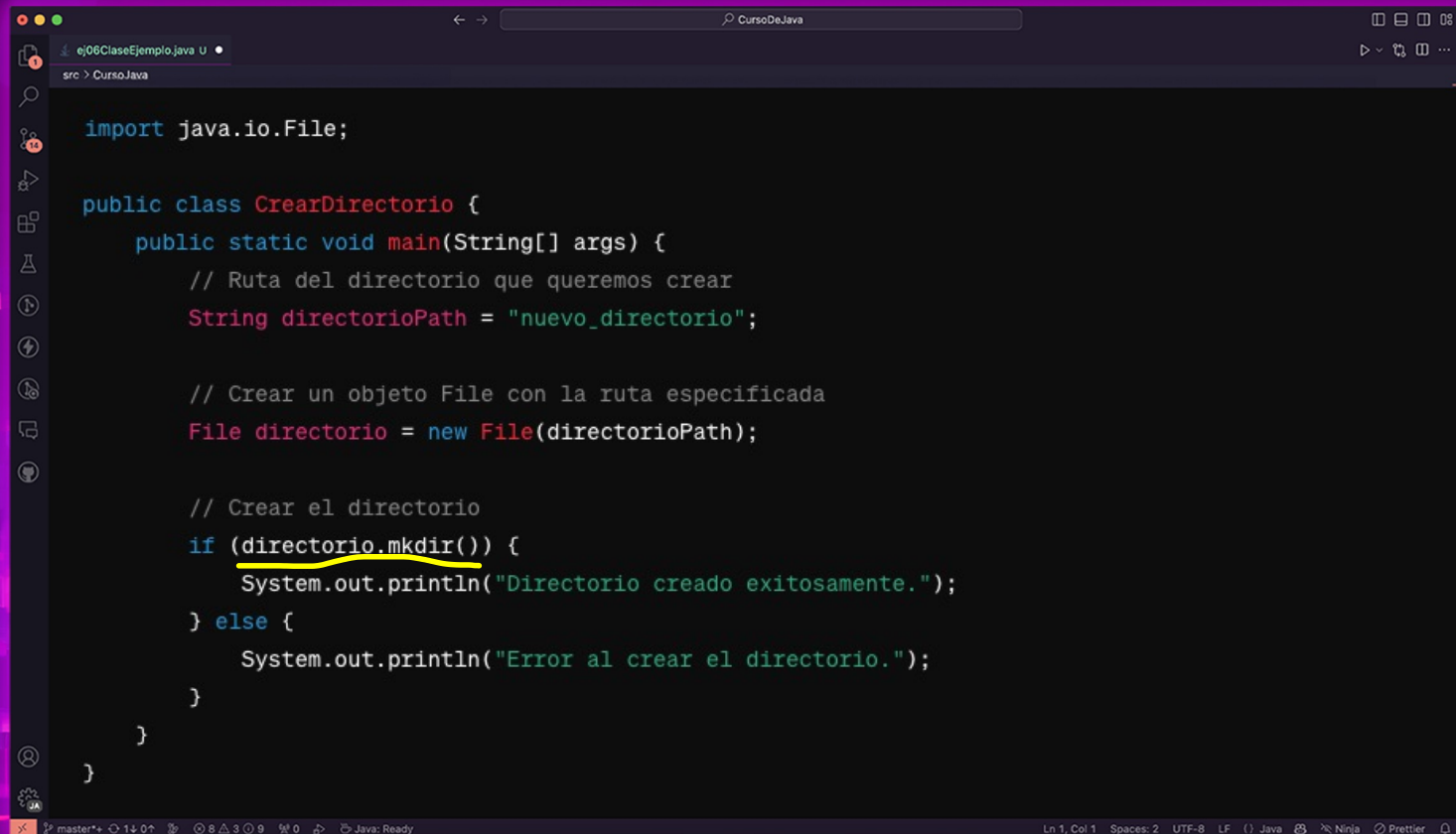
# Archivos y Directorios

Trabajar con directorios y archivos es una tarea común en la programación, y Java proporciona una amplia gama de clases y métodos en el paquete `java.io` y `java.nio.file` para manejar archivos y directorios.

## Trabajar con la clase File

La clase File en Java se utiliza para **representar archivos y directorios** en el sistema de archivos.

# Creación de un Directorio



The image shows a screenshot of a code editor window with a dark theme. The title bar at the top indicates the file is 'ej06ClaseEjemplo.java' and the project is 'CursoDeJava'. The code is written in Java and is enclosed in a public class named 'CrearDirectorio'. The main method takes an array of strings as an argument. It defines a string 'directorioPath' with the value 'nuevo\_directorio'. It then creates a 'File' object using this path. The code attempts to create the directory using 'mkdir()'. If successful, it prints 'Directorio creado exitosamente.'; otherwise, it prints 'Error al crear el directorio.'.

```
import java.io.File;

public class CrearDirectorio {
    public static void main(String[] args) {
        // Ruta del directorio que queremos crear
        String directorioPath = "nuevo_directorio";

        // Crear un objeto File con la ruta especificada
        File directorio = new File(directorioPath);

        // Crear el directorio
        if (directorio.mkdir()) {
            System.out.println("Directorio creado exitosamente.");
        } else {
            System.out.println("Error al crear el directorio.");
        }
    }
}
```

The status bar at the bottom shows 'master' branch, 140 commits, 8 files, 3 lines, 0 characters, and the editor is ready for Java. The cursor is at line 1, column 1. The encoding is UTF-8, line feed (LF), and the editor uses the Prettier formatter.



# Creación de un Archivo

```
ej06ClaseEjemplo.java U
src > CursoDeJava

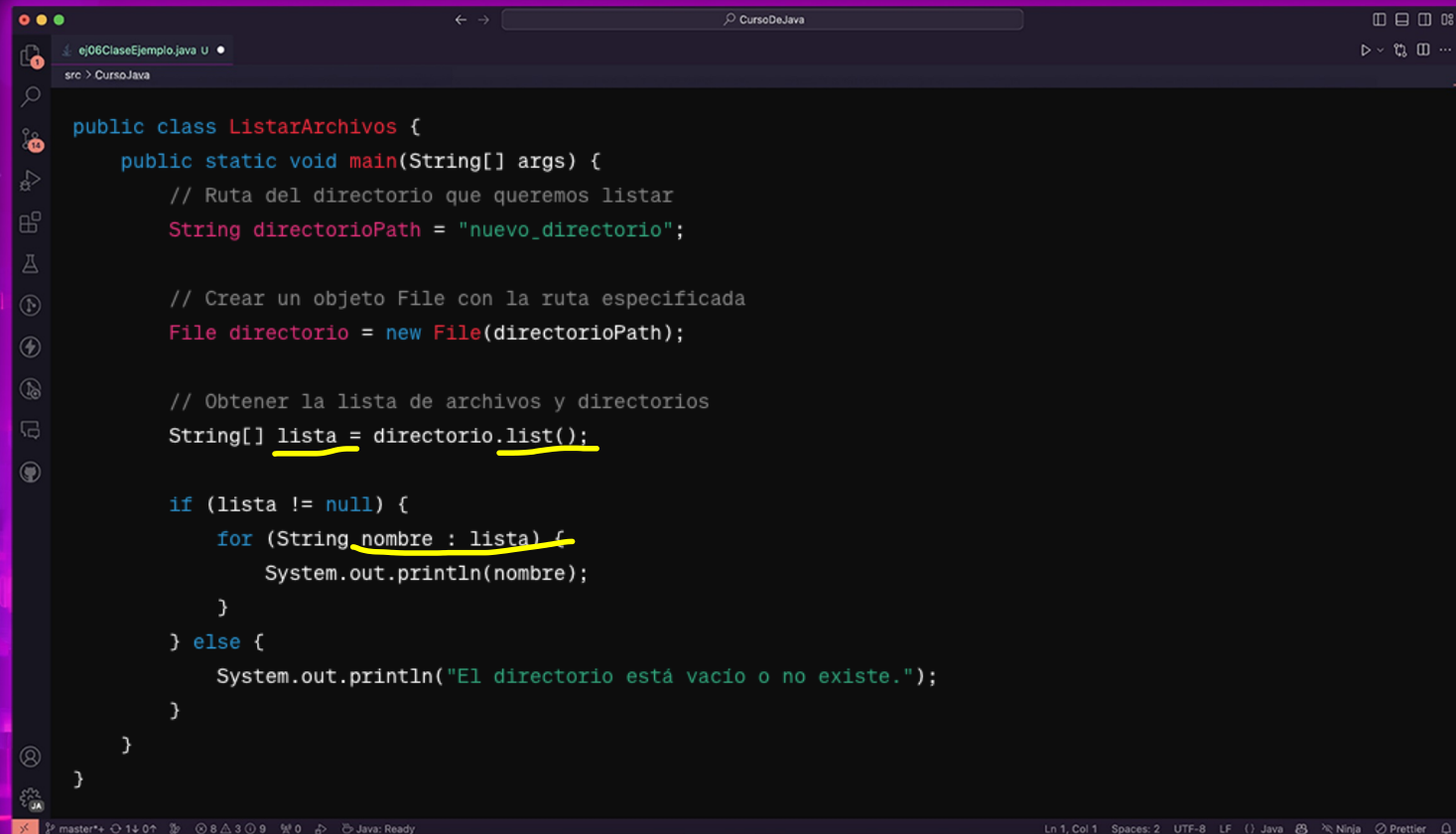
// Ruta del archivo que queremos crear
String archivoPath = "nuevo_directorio/nuevo_archivo.txt";

// Crear un objeto File con la ruta especificada
File archivo = new File(archivoPath);

try {
    // Crear el archivo
    if (archivo.createNewFile()) {
        System.out.println("Archivo creado exitosamente.");
    } else {
        System.out.println("Error al crear el archivo.");
    }
} catch (IOException e) {
    System.out.println("Ocurrió un error al crear el archivo.");
    e.printStackTrace();
}
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF () Java Ninja Prettier

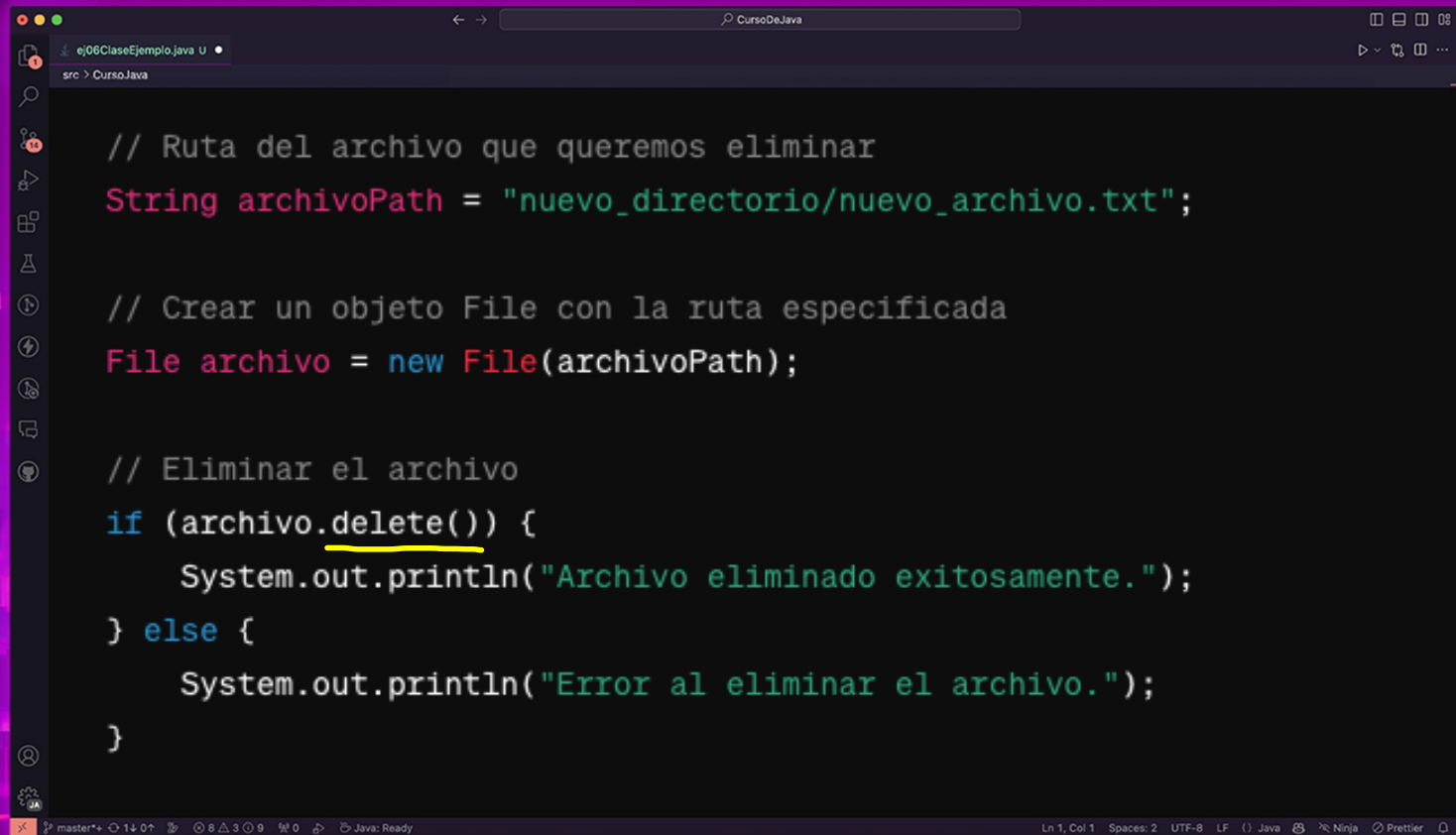
# Listar Archivos y Directorios



The image shows a screenshot of an IDE window with a dark theme. The title bar at the top says "CursoDeJava". The editor displays a Java class named "ListarArchivos". The code uses the "File" class to create a file object from a path and then calls the "list()" method to get an array of strings representing the files and directories in that path. The array is iterated over to print each item. The IDE interface includes a sidebar on the left with icons for Explorer, Search, Run and Debug, Source Control, and Extensions. The status bar at the bottom shows "master+ 140", "8 3 9", "0", "Java: Ready", "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "Java", "Ninja", and "Prettier".

```
public class ListarArchivos {  
    public static void main(String[] args) {  
        // Ruta del directorio que queremos listar  
        String directorioPath = "nuevo_directorio";  
  
        // Crear un objeto File con la ruta especificada  
        File directorio = new File(directorioPath);  
  
        // Obtener la lista de archivos y directorios  
        String[] lista = directorio.list();  
  
        if (lista != null) {  
            for (String nombre : lista) {  
                System.out.println(nombre);  
            }  
        } else {  
            System.out.println("El directorio está vacío o no existe.");  
        }  
    }  
}
```

# Eliminar Archivos

A screenshot of a code editor window titled 'CursoDeJava'. The editor shows a Java file named 'ej06ClaseEjemplo.java'. The code is as follows:

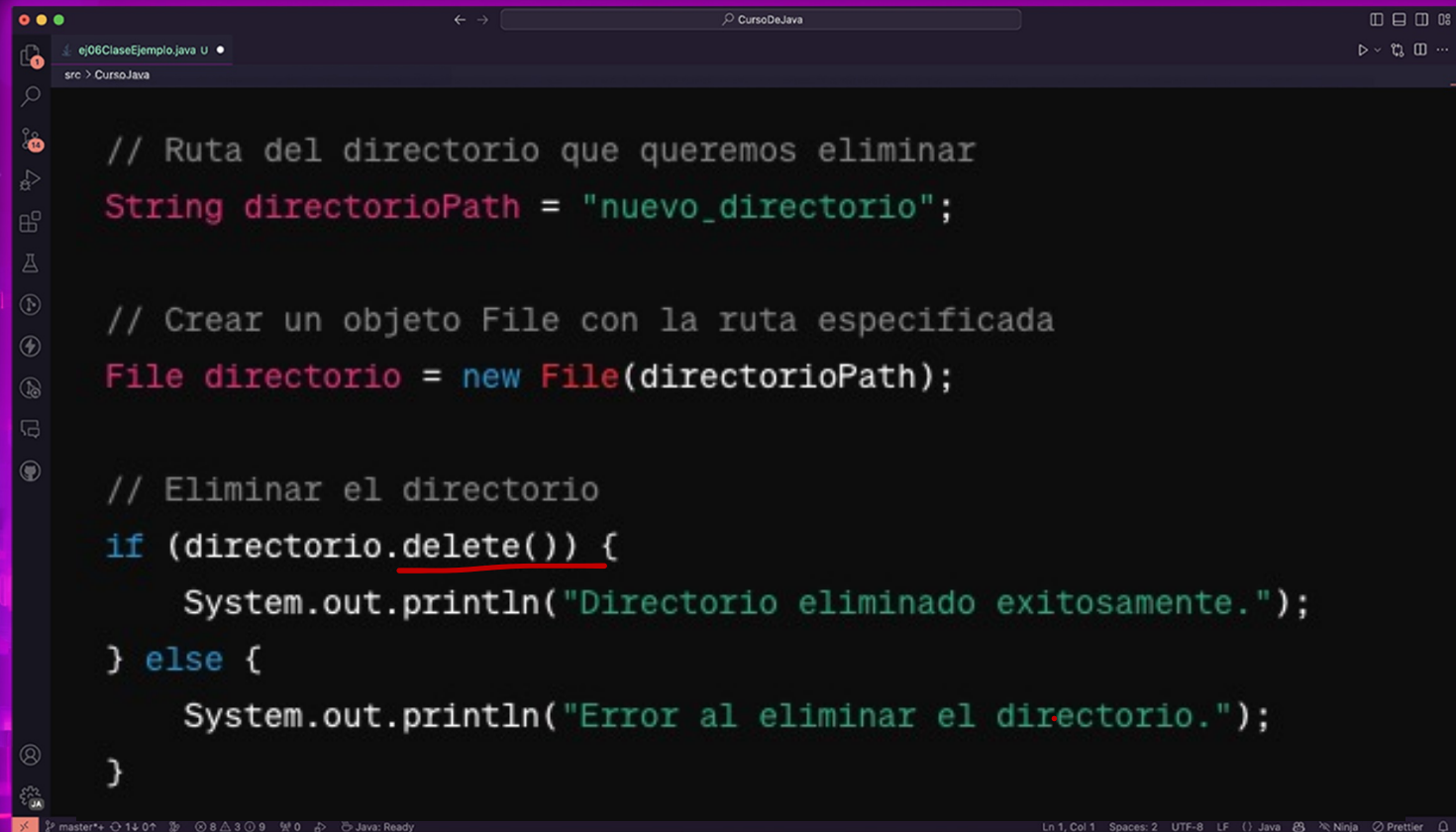
```
// Ruta del archivo que queremos eliminar
String archivoPath = "nuevo_directorio/nuevo_archivo.txt";

// Crear un objeto File con la ruta especificada
File archivo = new File(archivoPath);

// Eliminar el archivo
if (archivo.delete()) {
    System.out.println("Archivo eliminado exitosamente.");
} else {
    System.out.println("Error al eliminar el archivo.");
}
```

The word 'delete' in the `archivo.delete()` call is underlined in yellow. The IDE interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, Extensions, Testing, Remote Explorer, Docker, and Settings. The status bar at the bottom shows 'master\* 140', '8 3 0', 'Java: Ready', and 'Ln 1, Col 1 Spaces: 2 UTF-8 LF () Java Ninja Prettier'.

# Eliminar Directorios

A screenshot of a code editor window with a dark theme. The title bar shows 'ej06ClaseEjemplo.java U' and 'CursoDeJava'. The code is in Java and demonstrates how to delete a directory using the File class. The code includes comments in Spanish and uses System.out.println for output. The 'delete()' method is underlined in the code. The status bar at the bottom shows 'master+ 14 0', '8 3 0 9', '0', 'Java: Ready', 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'Java', 'Ninja', and 'Prettier'.

```
// Ruta del directorio que queremos eliminar
String directorioPath = "nuevo_directorio";

// Crear un objeto File con la ruta especificada
File directorio = new File(directorioPath);

// Eliminar el directorio
if (directorio.delete()) {
    System.out.println("Directorio eliminado exitosamente.");
} else {
    System.out.println("Error al eliminar el directorio.");
}
```



Programación I

# Lectura de archivos de texto

# Leer un archivo de texto

Para leer un archivo de texto en Java, puedes usar **FileReader** junto con **BufferedReader**.

- 1) Asegúrate de que el archivo exista en el directorio de trabajo.
- 2) Ejecuta el programa LeerArchivo.
- 3) El programa leerá e imprimirá el contenido del archivo línea por línea.

# Leer el archivo línea por línea

```
ej06ClaseEjemplo.java U
src > CursoJava

String rutaArchivo = "archivo.txt"; // Ruta del archivo a leer
BufferedReader lector = null;

try {
    // Crear un FileReader para leer el archivo
    FileReader fileReader = new FileReader(rutaArchivo);

    // Envolver FileReader con BufferedReader para una lectura más eficiente
    lector = new BufferedReader(fileReader);

    // Leer el archivo línea por línea
    String linea;
    while ((linea = lector.readLine()) != null) {
        System.out.println(linea); // Imprimir cada línea leída
    }
} catch (IOException e) {
    e.printStackTrace(); // Manejar posibles excepciones de E/S
} finally {
    try {
        if (lector != null) {
            lector.close(); // Cerrar el BufferedReader
        }
    } catch (IOException e) {
        e.printStackTrace(); // Manejar posibles excepciones al cerrar
    }
}
```

FileReader y BufferedReader se utilizan para leer archivos de texto de manera eficiente.

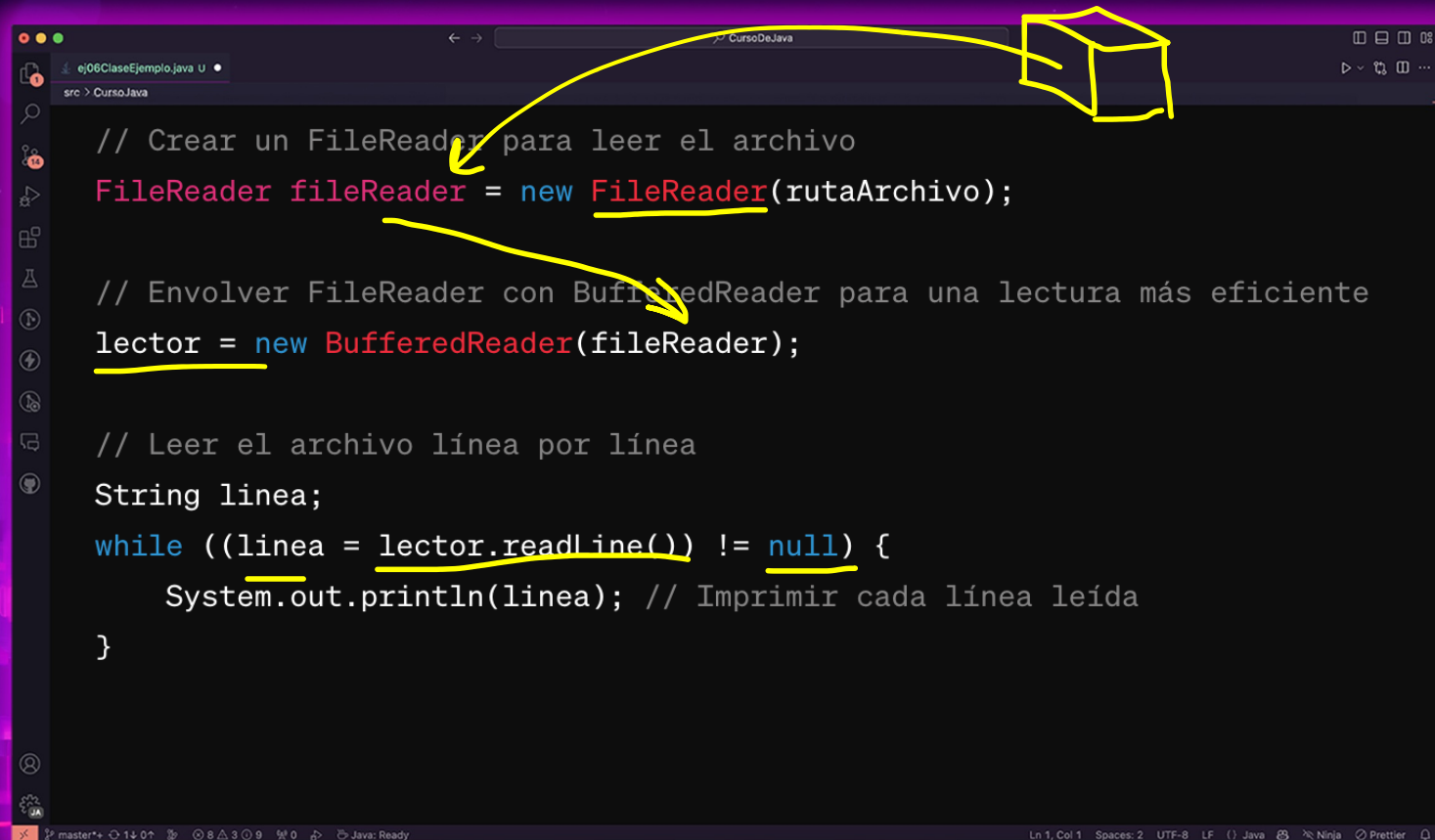
FileReader se encarga de la lectura básica del archivo.

BufferedReader se envuelve alrededor de FileReader para proporcionar una lectura más eficiente y métodos adicionales como readLine().

Se usa un bucle while para leer el archivo línea por línea hasta que readLine() devuelva null, lo que indica el final del archivo.

El bloque finally asegura que el BufferedReader se cierre, incluso si ocurre una excepción.

# Leer el archivo línea por línea



```
// Crear un FileReader para leer el archivo
FileReader fileReader = new FileReader(rutaArchivo);

// Envolver FileReader con BufferedReader para una lectura más eficiente
lector = new BufferedReader(fileReader);

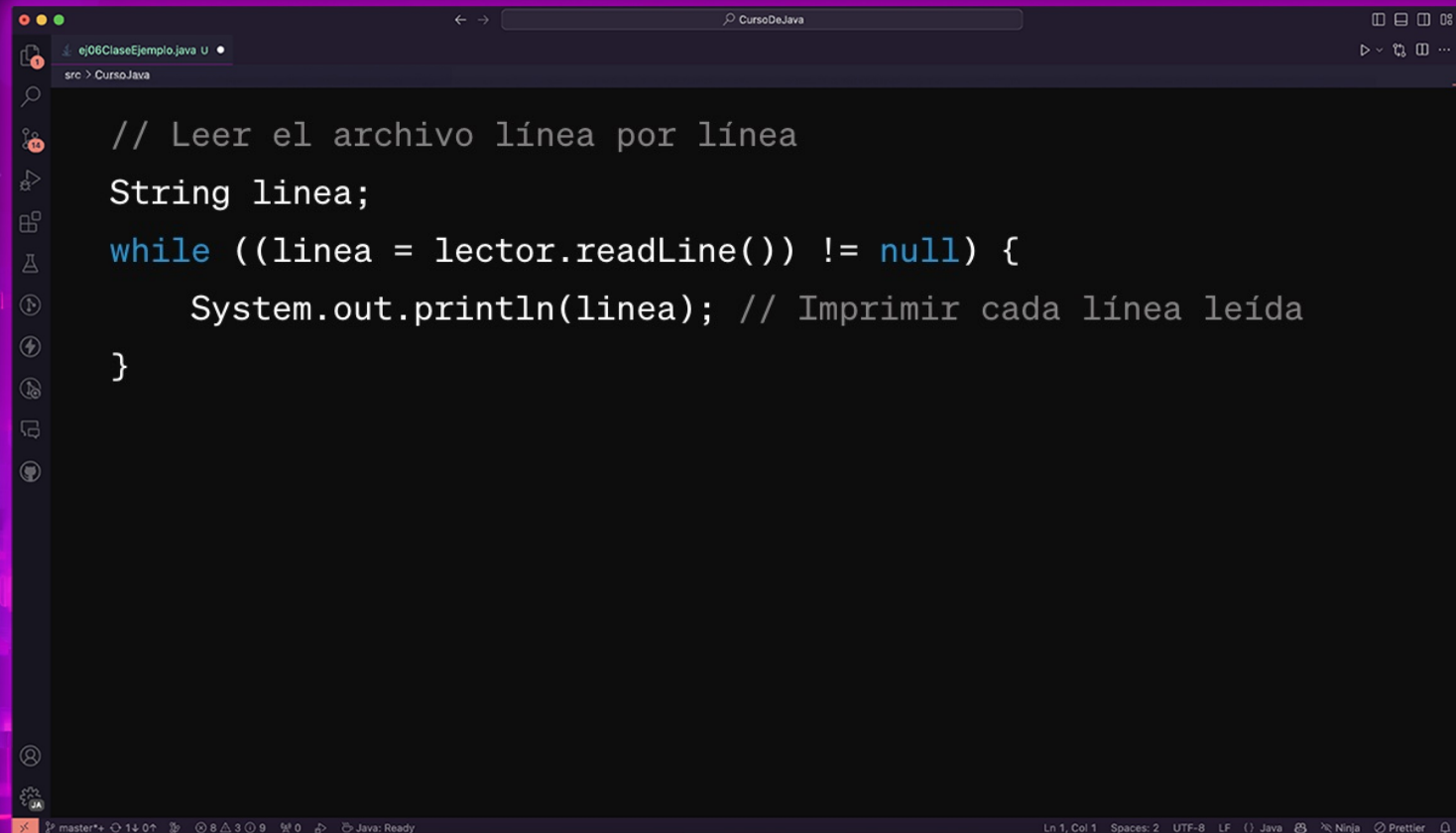
// Leer el archivo línea por línea
String linea;
while ((linea = lector.readLine()) != null) {
    System.out.println(linea); // Imprimir cada línea leída
}
```

The image shows a code editor window with the following Java code. Hand-drawn yellow annotations highlight specific parts of the code:

- A yellow box highlights the `FileReader` class in the first line of code.
- A yellow arrow points from the box to the `FileReader` class in the second line of code.
- A yellow arrow points from the box to the `BufferedReader` class in the third line of code.
- A yellow arrow points from the box to the `readLine()` method in the fourth line of code.



# Leer el archivo línea por línea



The image shows a screenshot of an IDE window with a dark theme. The title bar at the top says 'CursoDeJava'. The editor contains the following Java code:

```
// Leer el archivo línea por línea
String linea;
while ((linea = lector.readLine()) != null) {
    System.out.println(linea); // Imprimir cada línea leída
}
```

The IDE interface includes a sidebar on the left with various icons, a top toolbar with window management icons, and a bottom status bar showing 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'Java', and 'Prettier'.

# Escribir en un archivo de texto

Para escribir en un archivo de texto en Java, puedes usar **FileWriter** junto con **BufferedWriter**.

- 1) El programa escribirá las líneas de texto en archivo.
- 2) Verifica el contenido de archivo para asegurarte de que las líneas se hayan escrito correctamente.

# Leer el archivo línea por línea

```
ej06ClaseEjemplo.java U
src > CursoJava

String rutaArchivo = "archivo.txt"; // Ruta del archivo a escribir
BufferedWriter escritor = null;

try {
    // Crear un FileWriter para escribir en el archivo
    FileWriter fileWriter = new FileWriter(rutaArchivo);

    // Envolver FileWriter con BufferedWriter para una escritura más eficiente
    escritor = new BufferedWriter(fileWriter);

    // Escribir en el archivo
    escritor.write("Esta es la primera línea.");
    escritor.newLine(); // Escribir una nueva línea
    escritor.write("Esta es la segunda línea.");
} catch (IOException e) {
    e.printStackTrace(); // Manejar posibles excepciones de E/S
} finally {
    try {
        if (escritor != null) {
            escritor.close(); // Cerrar el BufferedWriter
        }
    } catch (IOException e) {
        e.printStackTrace(); // Manejar posibles excepciones al cerrar
    }
}
```

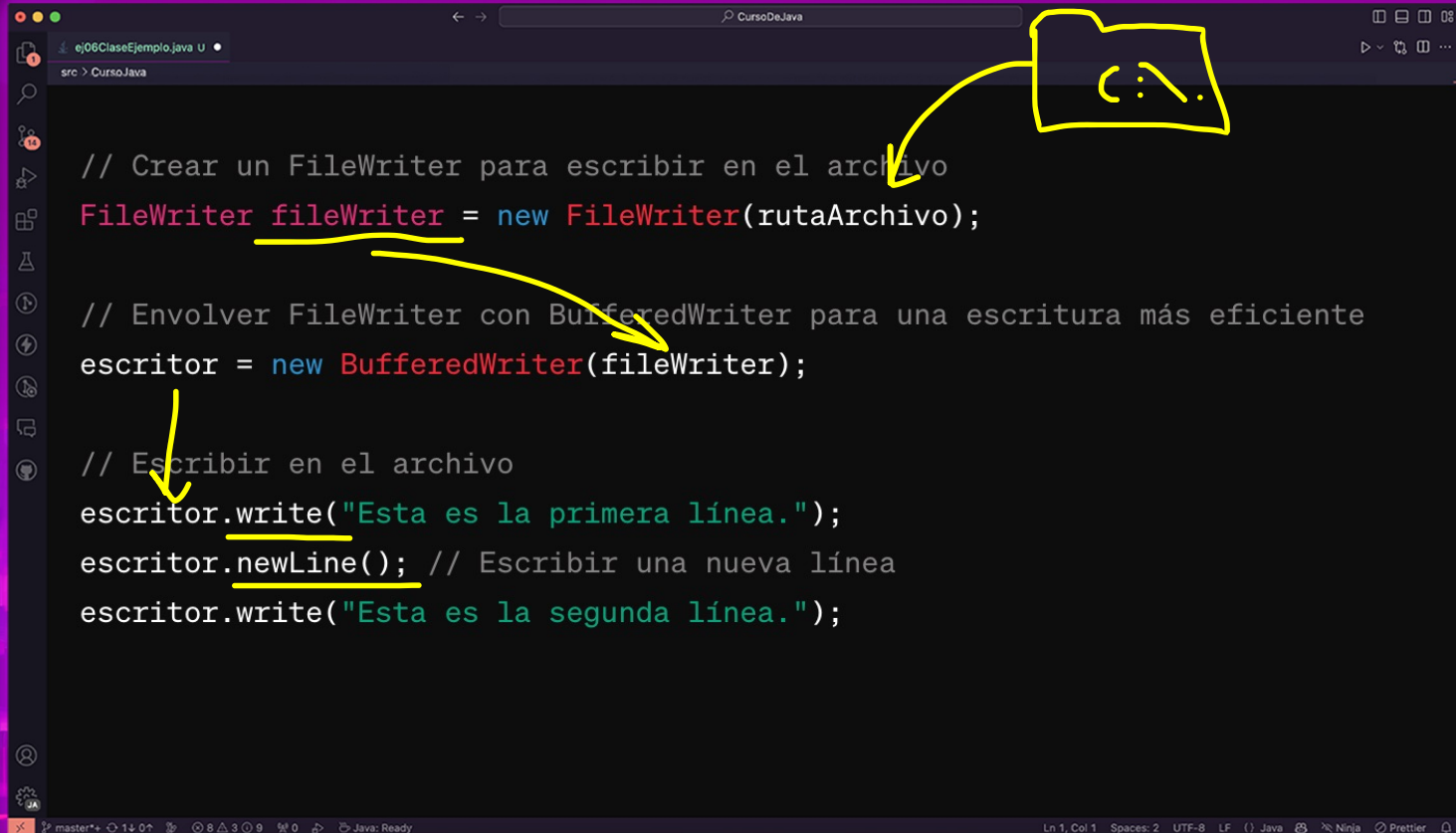
FileWriter y BufferedWriter se utilizan para escribir en archivos de texto de manera eficiente.

FileWriter se encarga de la escritura básica del archivo. BufferedWriter se envuelve alrededor de FileWriter para proporcionar una escritura más eficiente y métodos adicionales como newLine().

Se escriben líneas de texto en el archivo utilizando el método write() y newLine() para insertar saltos de línea.

El bloque finally asegura que el BufferedWriter se cierre, incluso si ocurre una excepción.

# Leer el archivo línea por línea



```
// Crear un FileWriter para escribir en el archivo
FileWriter fileWriter = new FileWriter(rutaArchivo);

// Envolver FileWriter con BufferedWriter para una escritura más eficiente
escriptor = new BufferedWriter(fileWriter);

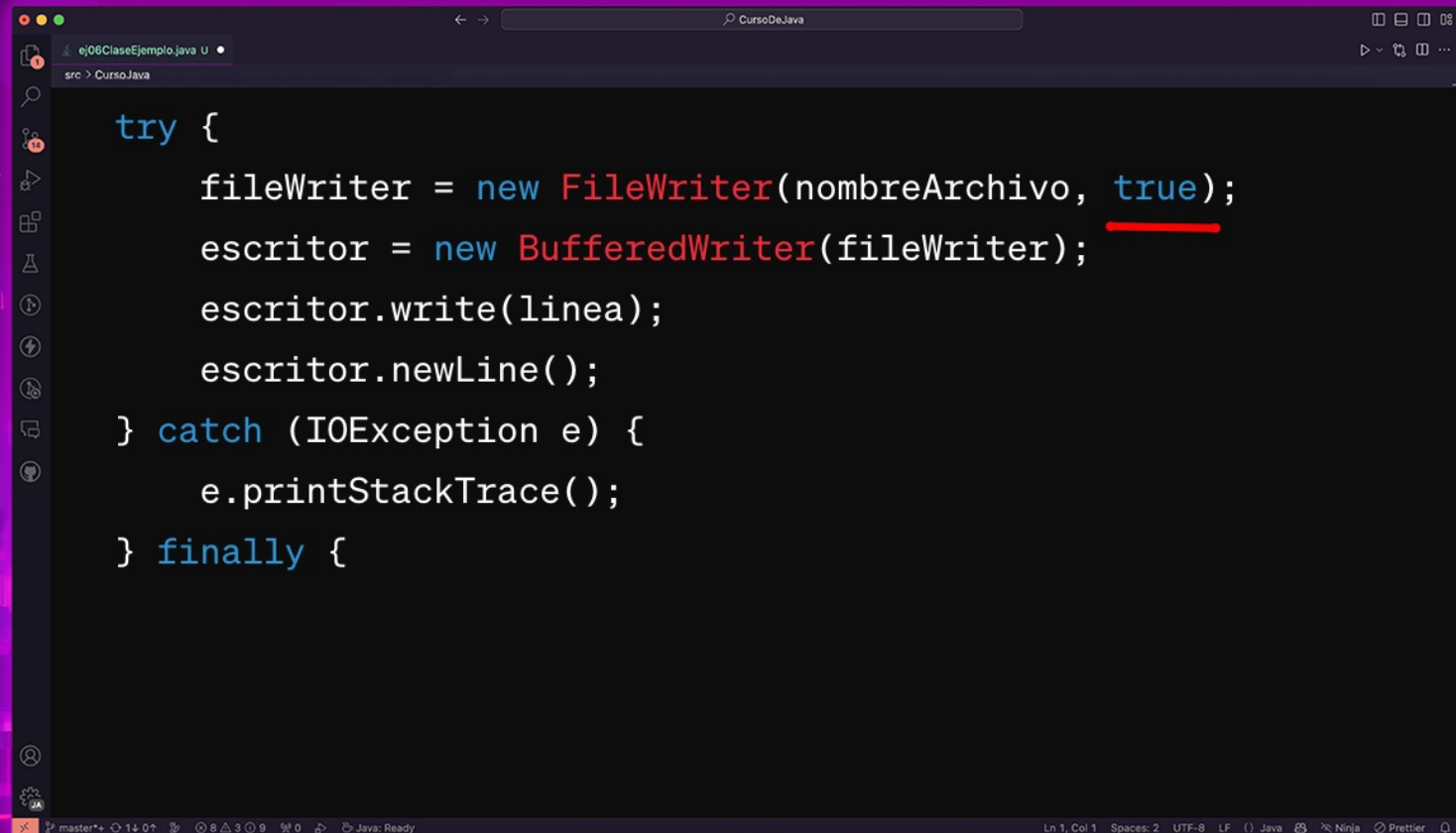
// Escribir en el archivo
escriptor.write("Esta es la primera línea.");
escriptor.newLine(); // Escribir una nueva línea
escriptor.write("Esta es la segunda línea.");
```

The image shows a code editor window with the following Java code. Hand-drawn yellow annotations highlight specific parts of the code:

- A yellow box with a yellow arrow points to the `FileWriter` and `BufferedWriter` classes in the first two lines of code.
- A yellow arrow points to the `write` and `newLine` methods in the last three lines of code.



# Escribir al final del archivo



The image shows a screenshot of an IDE window with a dark theme. The title bar at the top says "CursoDeJava". The editor shows a file named "ej06ClaseEjemplo.java". The code is as follows:

```
try {  
    fileWriter = new FileWriter(nombreArchivo, true);  
    escritor = new BufferedWriter(fileWriter);  
    escritor.write(linea);  
    escritor.newLine();  
} catch (IOException e) {  
    e.printStackTrace();  
} finally {
```

The status bar at the bottom indicates "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "Java", and "Prettier".

# Leer archivo carácter por caracter

Para recorrer y leer un archivo de texto carácter por carácter en Java, puedes utilizar la clase **FileReader** que es adecuada para leer flujos de datos de caracteres.

# Leer el archivo caracter por caracter

```
ej06ClaseEjemplo.java U
src > CursoJava
String nombreArchivo = "archivo.txt";

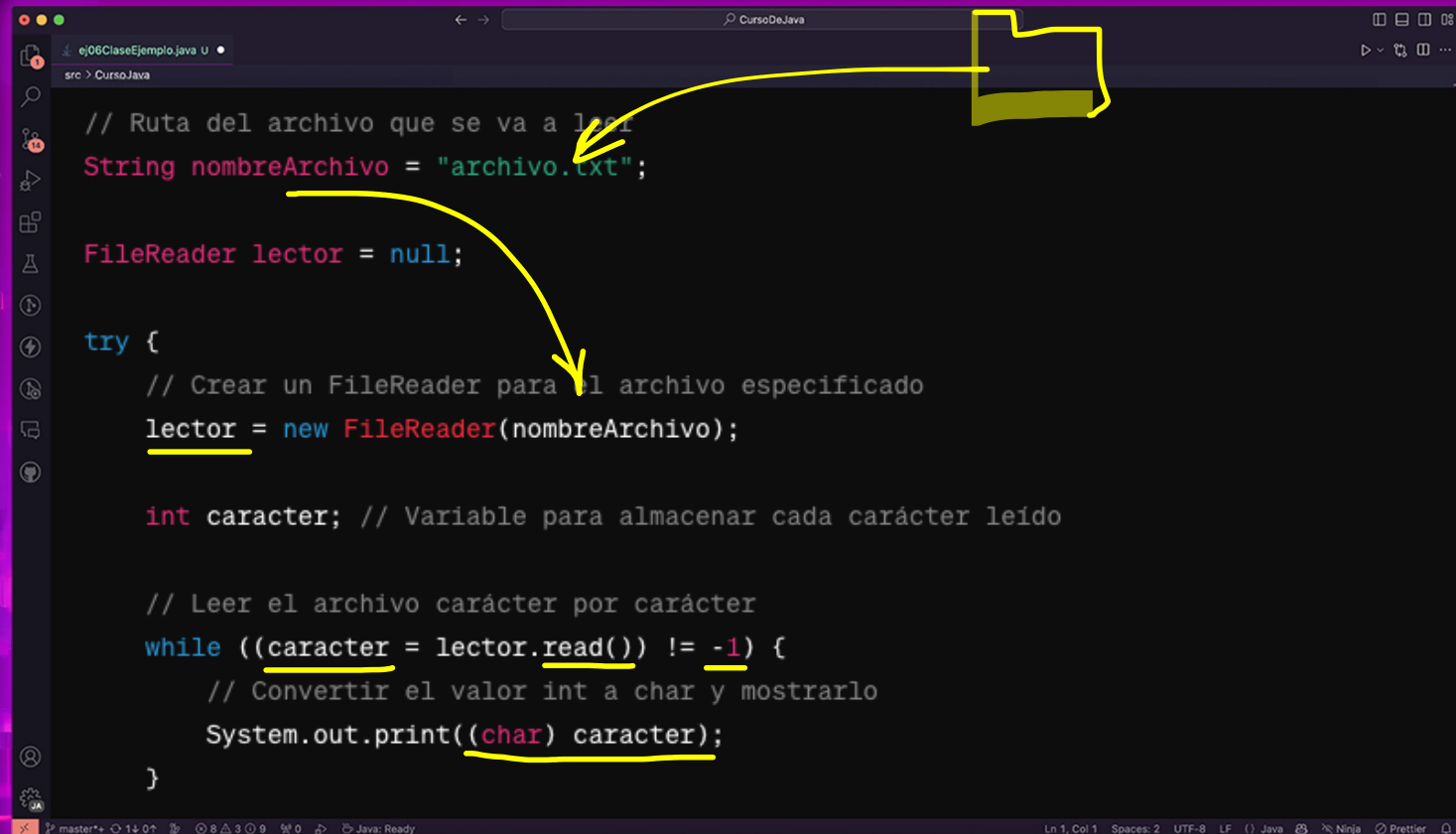
FileReader lector = null;

try {
    // Crear un FileReader para el archivo especificado
    lector = new FileReader(nombreArchivo);

    int caracter; // Variable para almacenar cada carácter leído

    // Leer el archivo carácter por carácter
    while ((caracter = lector.read()) != -1) {
        // Convertir el valor int a char y mostrarlo
        System.out.print((char) caracter);
    }
} catch (IOException e) {
    // Manejar cualquier excepción de entrada/salida
    e.printStackTrace();
} finally {
    // Cerrar el lector en el bloque finally para asegurar que se cierra
    if (lector != null) {
        try {
            lector.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

# Leer el archivo caracter por caracter



```
// Ruta del archivo que se va a leer
String nombreArchivo = "archivo.txt";

FileReader lector = null;

try {
    // Crear un FileReader para el archivo especificado
    lector = new FileReader(nombreArchivo);

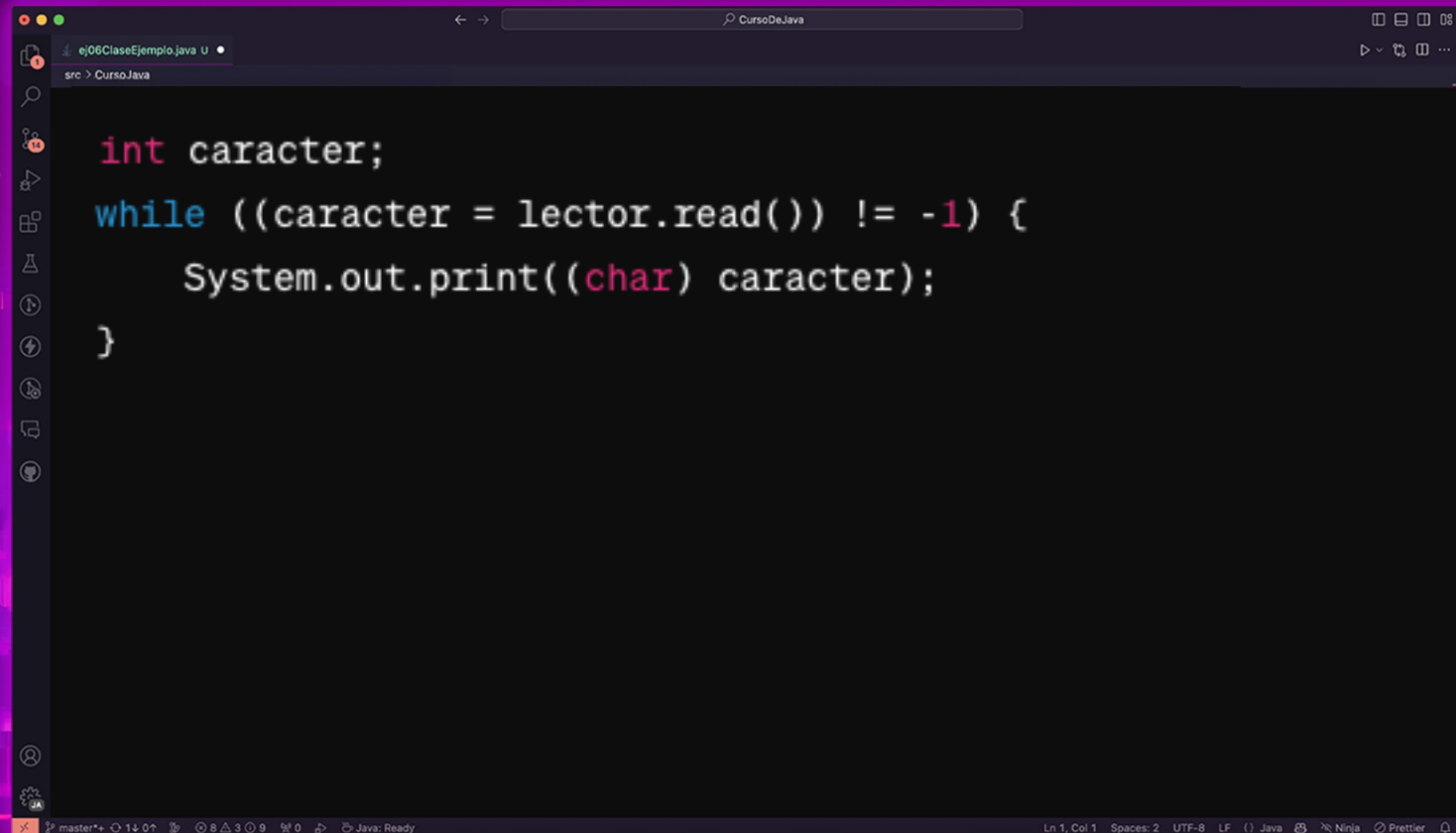
    int caracter; // Variable para almacenar cada carácter leído

    // Leer el archivo carácter por carácter
    while ((caracter = lector.read()) != -1) {
        // Convertir el valor int a char y mostrarlo
        System.out.print((char) caracter);
    }
}
```

The image shows a code editor window with a dark theme. The title bar indicates the file is 'ej06ClaseEjemplo.java'. The code is in Java and demonstrates how to read a file character by character using the `FileReader` class. A yellow box highlights the file path `"archivo.txt"` in the `nombreArchivo` variable. A yellow arrow points from this box to the `lector.read()` method call in the `while` loop. Another yellow arrow points from the `lector` variable to the `lector.read()` method call. The `lector` variable is also underlined. The `while` loop condition `((caracter = lector.read()) != -1)` is underlined. The `(char) caracter` cast in the `System.out.print` statement is also underlined. The status bar at the bottom shows 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'Java', and 'Prettier'.



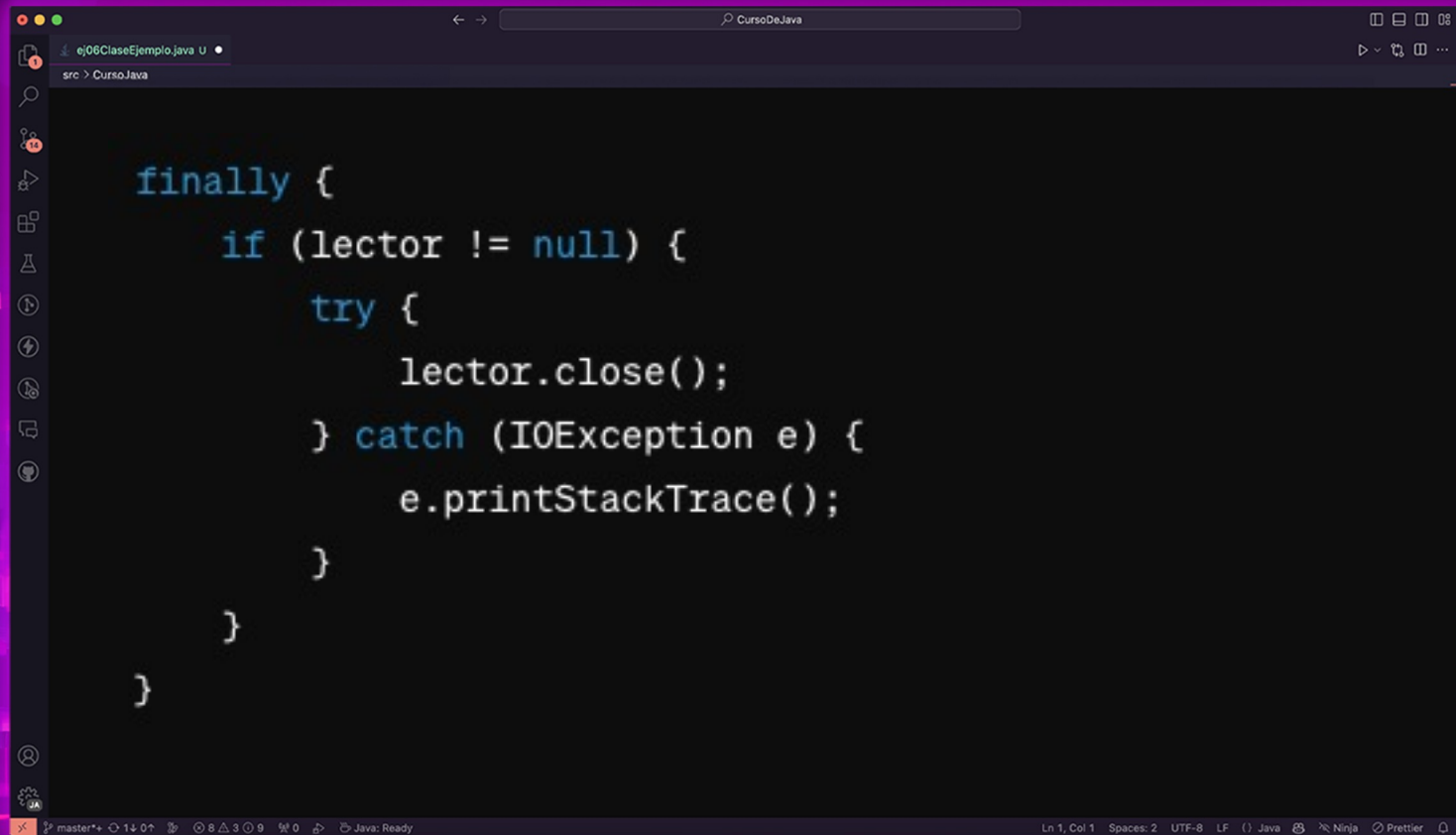
# Leer el archivo carácter por carácter



The image shows a screenshot of a code editor window. The title bar at the top indicates the file is 'ej06ClaseEjemplo.java' and the editor is 'CursoDeJava'. The code is written in Java and is designed to read a file character by character. It uses a 'while' loop that continues as long as the 'lector.read()' method returns a value not equal to -1. Inside the loop, the character is cast to a 'char' and printed to the console using 'System.out.print()'. The editor interface includes a sidebar on the left with various icons for file management and a status bar at the bottom showing 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', and 'Java: Ready'.

```
int character;  
while ((character = lector.read()) != -1) {  
    System.out.print((char) character);  
}
```

# Cerrar el FileReader

A screenshot of an IDE window titled 'CursoDeJava'. The editor shows a Java file named 'ej06ClaseEjemplo.java'. The code is a snippet for closing a 'lector' object. It uses a 'finally' block to ensure the 'lector' is closed, even if an exception occurs. The code is as follows:

```
finally {  
    if (lector != null) {  
        try {  
            lector.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

The IDE interface includes a sidebar with icons for Explorer, Search, Run and Debug, Source Control, Extensions, Testing, Remote Explorer, Docker, and Settings. The status bar at the bottom shows 'master\*+', 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'LF', 'Java', and 'Prettier'.