

Programación I

try-catch

try-catch

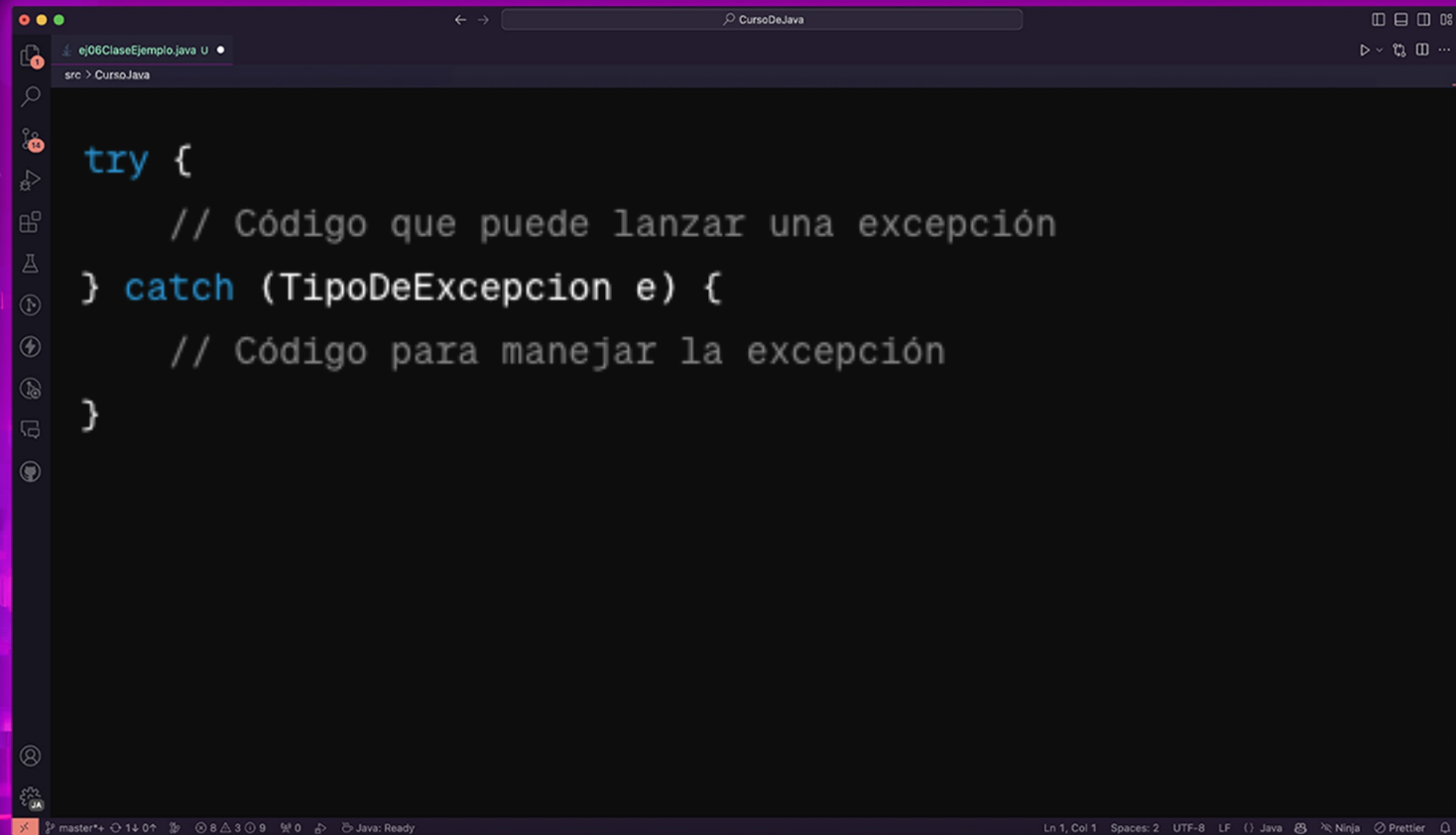
En Java, el bloque try-catch se utiliza para manejar excepciones. Una excepción es un evento que interrumpe el flujo normal de la ejecución de un programa. Las excepciones pueden ocurrir por diversas razones, como errores de entrada/salida, operaciones aritméticas inválidas, acceso a índices fuera de los límites de un array, entre otros.

El manejo de excepciones es crucial para escribir programas robustos y que puedan gestionar errores de manera adecuada.

¿Por qué se usa try-catch?

1. Robustez: Mejora la robustez del programa al permitir que el programa maneje errores de manera controlada en lugar de terminar abruptamente.
2. Mantenimiento: Facilita el mantenimiento del código al centralizar la lógica de manejo de errores.
3. Seguridad: Permite limpiar recursos (como archivos o conexiones de base de datos) de manera segura incluso cuando ocurre un error.
4. Experiencia del Usuario: Proporciona una mejor experiencia de usuario al manejar los errores de manera adecuada, mostrando mensajes de error amigables o intentando recuperarse del error.

Estructura de try-catch



The image shows a screenshot of a code editor window with a dark theme. The editor displays a Java code snippet for a try-catch block. The code is as follows:

```
try {  
    // Código que puede lanzar una excepción  
} catch (TipoDeExcepcion e) {  
    // Código para manejar la excepción  
}
```

The editor's interface includes a top bar with a search icon and the text "CursoDeJava". On the left, there is a sidebar with various icons for file management and development tools. The bottom status bar shows "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "Java", and "Prettier".

Manejo de Excepciones en Entrada/Salida

```
ej06ClaseEjemplo.java U
src > CursoJava

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class ManejoExcepciones {
    public static void main(String[] args) {
        String archivoPath = "nuevo_directorio/nuevo_archivo.txt";

        // Intentar escribir en un archivo
        try (FileWriter writer = new FileWriter(archivoPath)) {
            writer.write("Hola, este es un archivo de prueba.");
            System.out.println("Datos escritos exitosamente en el archivo.");
        } catch (IOException e) {
            // Manejar la excepción
            System.out.println("Ocurrió un error al escribir en el archivo.");
            e.printStackTrace();
        }
    }
}
```

try: Intenta escribir datos en un archivo. `FileWriter` puede lanzar una excepción `IOException` si ocurre un error durante la escritura.

catch (IOException e): Captura la excepción `IOException` y permite al programa manejarla, en este caso, imprimiendo un mensaje de error y la traza de la excepción.

Excepciones en Operaciones Aritméticas

```
ej06ClaseEjemplo.java U
src > CursoDeJava

public class DivisionPorCero {
    public static void main(String[] args) {
        int numerador = 10;
        int denominador = 0;

        try {
            int resultado = numerador / denominador;
            System.out.println("El resultado es: " + resultado);
        } catch (ArithmeticException e) {
            // Manejar la excepción de división por cero
            System.out.println("Error: No se puede dividir por cero.");
        }
    }
}
```

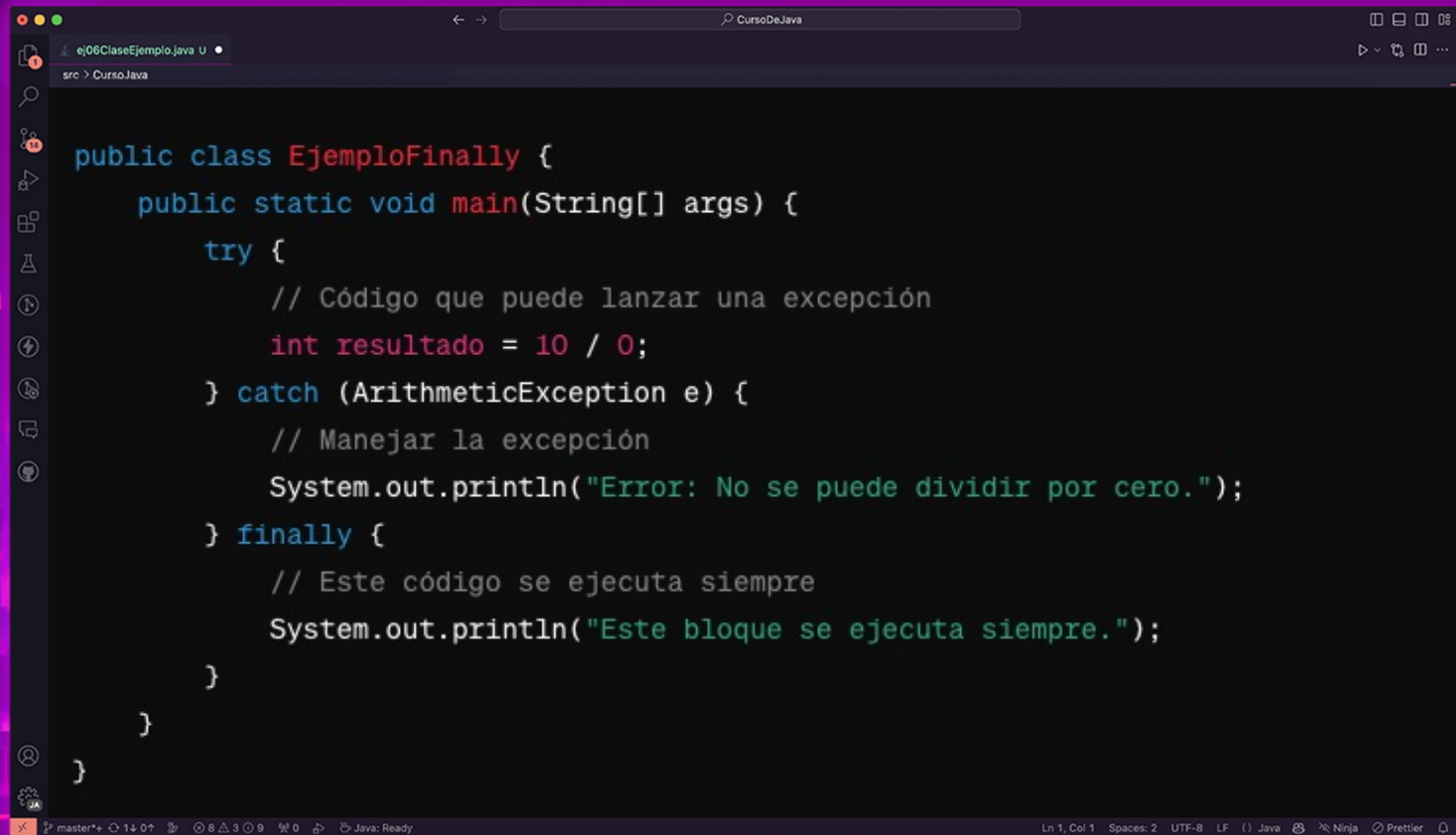
try: Intenta realizar una división. Dividir por cero lanzará una excepción `ArithmeticException`.

catch (`ArithmeticException e`): Captura la excepción `ArithmeticException` y maneja el error, en este caso, imprimiendo un mensaje de error específico.

Bloques finally

A veces, además de try y catch, se usa un bloque finally que se ejecuta siempre, independientemente de si se lanzó una excepción o no. Es útil para liberar recursos.

finally



```
public class EjemploFinally {  
    public static void main(String[] args) {  
        try {  
            // Código que puede lanzar una excepción  
            int resultado = 10 / 0;  
        } catch (ArithmeticException e) {  
            // Manejar la excepción  
            System.out.println("Error: No se puede dividir por cero.");  
        } finally {  
            // Este código se ejecuta siempre  
            System.out.println("Este bloque se ejecuta siempre.");  
        }  
    }  
}
```