



“Informe de ejecución y explicación de excepciones en códigos JAVA”

Programación Avanzada



Ing. David Santiago Velásquez Cifuentes.¹
*Maestría en Ingeniería. Facultad de Ingeniería y Ciencias Básicas.
Fundación Universitaria Los Libertadores.*

1. Introducción

En el presente informe, se abordará la ejecución y gestión de excepciones en códigos Java. El objetivo principal es proporcionar una comprensión detallada de cómo se manejan y resuelven las excepciones en el desarrollo de aplicaciones Java. A lo largo de este informe, se explicarán cuatro excepciones comunes en Java: “FileNotFoundException” (Archivo no encontrado), “UnsupportedOperationException” (Operación no Soportada), “IllegalArgumentException” (Argumento ilegal) y “IOException” (Excepción de entrada/salida).

Excepción 1 – Archivo no encontrado: Esta excepción se arroja cuando un programa intenta abrir o leer un archivo que no se encuentra en la ubicación especificada. Es fundamental asegurarse de que el archivo exista antes de intentar acceder a él.

Excepción 2 – Operación no soportada: Esta excepción se produce cuando se intenta realizar una operación no soportada por un objeto en particular. Por ejemplo, tratar de agregar un elemento a una lista de solo lectura arrojará esta excepción.

Excepción 3 – Argumento ilegal: Se arroja cuando un método recibe un argumento no válido. Es importante garantizar que los argumentos pasados a los métodos cumplan con las restricciones especificadas.

Excepción 4 – Entrada/Salida: Esta excepción abarca una variedad de problemas relacionados con la entrada y salida de datos, como problemas de lectura y escritura en archivos. Puede manifestarse en situaciones como la falta de permisos de escritura o la desconexión de un flujo de datos.

Para ilustrar la correcta gestión de excepciones y la ejecución exitosa de códigos Java, se presentarán capturas de pantalla que muestran cómo se han enfrentado y resuelto estas excepciones en ejemplos prácticos. Además, se destacará la importancia de documentar y manejar las excepciones de manera adecuada para garantizar la robustez y la estabilidad de las aplicaciones.

2. Ejemplo de excepción 1 – Archivo no encontrado

Explicación detallada del código:

- Importamos las clases y excepciones necesarias para trabajar con archivos (File, FileNotFoundException) y para la lectura de archivos (Scanner).
- Definimos la clase ExcepcionArchivoNoEncontrado.
- En el método main, empezamos a manejar la lógica principal del programa.
- Dentro del bloque try, creamos un objeto File llamado archivo que representa el archivo "archivoInexistente.txt" (esta línea solo crea una referencia al archivo y no comprueba si el archivo realmente existe en el sistema de archivos)
- Creamos un objeto Scanner llamado scanner y lo inicializamos con el objeto archivo. Esto nos permite leer el contenido del archivo.
- Verificamos si el archivo tiene al menos una línea de texto utilizando el método hasNextLine() del Scanner.

¹ Email: dsvelasquezc@ulibertadores.edu.co

Actividad 4 – “Informe de ejecución y explicación de excepciones en códigos JAVA”

- Si el archivo tiene al menos una línea de texto, imprimimos la primera línea en la consola utilizando el método `nextLine()` del `Scanner`.
- Cerramos el `Scanner` utilizando el método `close()` para liberar los recursos asociados al archivo.
- Comenzamos el bloque `catch`, que maneja la excepción `FileNotFoundException`. Si se produce esta excepción, el programa imprimirá un mensaje de error que incluye la descripción de la excepción (mensaje de error) utilizando `e.getMessage()`. Esto informa al usuario que el archivo no se encontró o no pudo abrirse.

```
// Excepción 1 con explicación - Archivo no existente. Presentado por DavidVelásquez.

import java.io.File;           // Importa la clase File del paquete java.io para trabajar con archivos.
import java.io.FileNotFoundException; // Importa la excepción FileNotFoundException del paquete java.io.
import java.util.Scanner;      // Importa la clase Scanner del paquete java.util para leer el archivo.

public class ExcepcionArchivoNoEncontrado {
    public static void main(String[] args) {
        try {
            File archivo = new File("ArchivoNoExistente.txt"); // Crea un objeto File que representa el archivo
            "ArchivoNoExistente.txt".
            Scanner scanner = new Scanner(archivo);             // Crea un objeto Scanner para leer el contenido del
            archivo.

            if (scanner.hasNextLine()) { // Verifica si el archivo tiene al menos una línea de texto.
                System.out.println(scanner.nextLine()); // Imprime la primera línea del archivo en la consola.
            }

            scanner.close(); // Cierra el Scanner para liberar los recursos asociados al archivo.
        } catch (FileNotFoundException e) {
            System.out.println("¡Alerta! Error o excepción 'File not Found': " + e.getMessage()); // Maneja la excepción
            FileNotFoundException. Imprime un mensaje de error y la descripción de la excepción.
        }
    }
}
```

Imagen 1 – Ejemplo de excepción – Archivo no encontrado

3. Ejemplo de excepción 2 – Operación no soportada

```
// Excepción 2 con explicación - Operación no soportada. Presentado por DavidVelásquez.

import java.io.File; // Importa la clase File del paquete java.io para trabajar con archivos.
import java.util.Scanner; // Importa la clase Scanner del paquete java.util para leer el archivo.
import java.io.FileNotFoundException; // Importa la excepción FileNotFoundException.

public class ExcepcionOperacionNoSoportada {
    public static void main(String[] args) {
        try {
            File archivo = new File("archivoExistente.txt"); // Crea un objeto File para el archivo
            "archivoExistente.txt".
            Scanner scanner = new Scanner(archivo); // Crea un objeto Scanner para leer el contenido del archivo.

            if (scanner.hasNextLine()) { // Comprueba si el archivo tiene al menos una línea de texto.
                System.out.println(scanner.nextLine()); // Imprime la primera línea del archivo en la consola.
            }

            scanner.close(); // Cierra el Scanner para liberar los recursos asociados al archivo.
        } catch (FileNotFoundException e) { // Captura la excepción FileNotFoundException si se produce.
            System.out.println("¡Alerta! Error: El archivo no se pudo encontrar o las operaciones no son soportadas."); //
            Imprime un mensaje de error.
        }
    }
}
```

Imagen 2 – Ejemplo de excepción – Operación no soportada

Explicación detallada del código:

- Importamos las clases necesarias para trabajar con archivos (`File`, `Scanner`) y la excepción `FileNotFoundException`. Esto nos permite utilizar estas clases y manejar la excepción en el código.
- Definimos una clase llamada `ExcepcionOperacionNoSoportada`.
- El método `main` es el punto de entrada del programa. Aquí comienza la ejecución.
- En el bloque `try`, se intenta realizar una operación que podría lanzar una excepción.
- Creamos un objeto `File` llamado `archivo`, que representa el archivo "archivoExistente.txt" (esto es solo una referencia al archivo en el sistema de archivos, y no verifica si el archivo realmente existe)

Actividad 4 – “Informe de ejecución y explicación de excepciones en códigos JAVA”

- Luego, creamos un objeto Scanner llamado scanner y lo inicializamos con el objeto archivo. Esto nos permite leer el contenido del archivo.
- Verificamos si el archivo tiene al menos una línea de texto utilizando el método hasNextLine() del Scanner.
- Si el archivo tiene al menos una línea de texto, imprimimos la primera línea en la consola utilizando el método nextLine() del Scanner.
- Cerramos el Scanner utilizando el método close(). Esto libera los recursos asociados al archivo y lo cierra.
- En el bloque catch, capturamos la excepción FileNotFoundException si se produce. Esto significa que si el archivo "archivoExistente.txt" no se encuentra o no se puede abrir, ninguna otra operación puede realizarse o continuar el flujo de control.
- Dentro del bloque catch, imprimimos un mensaje de error que indica que el archivo no se pudo encontrar o que la operación requerida no fue soportada.

4. Ejemplo de excepción 3 – Argumento ilegal

```
// Exepción 3 con explicación - Excepción de argumento ilegal. Presentado por DavidVelásquez.  
  
public class ExcepcionArgumentoIlegal {  
    public static void main(String[] args) {  
        try {  
            int edad = -5; // Supongamos que esta es la edad ingresada por el usuario.  
  
            if (edad < 0) {  
                throw new IllegalArgumentException("La edad no puede ser negativa");  
            }  
  
            System.out.println("Edad: " + edad);  
        } catch (IllegalArgumentException e) {  
            System.out.println("¡Alerta! Error de argumento ilegal: " + e.getMessage());  
        }  
    }  
}
```

Imagen 3 – Ejemplo de excepción – Argumento ilegal

Explicación detallada del código:

- Definimos una clase llamada ExcepcionArgumentoIlegal.
- El método main es el punto de entrada del programa.
- Suponemos que edad es una variable que representa la edad ingresada por el usuario. En este caso, hemos asignado un valor negativo a edad para ilustrar un argumento ilegal.
- Verificamos si edad es menor que 0. Si es así, lanzamos una excepción IllegalArgumentException con un mensaje que indica que la edad no puede ser negativa.
- Si la edad no es negativa, imprimimos la edad en la consola.
- En el bloque catch, capturamos la excepción IllegalArgumentException si se produce. Esto ocurre cuando la edad es negativa, y el mensaje de error especificado se imprime en la consola.

5. Ejemplo de excepción 4 – Entrada/Salida

Explicación detallada del código:

- Importamos las clases necesarias para trabajar con archivos (BufferedReader, FileReader) y la excepción IOException.
- Definimos una clase llamada ExcepcionEntradaSalida.
- El método main es el punto de entrada del programa.
- Definimos nombreArchivo como el nombre del archivo que intentamos abrir. En este caso, suponemos que estamos intentando abrir un archivo inexistente.
- En el bloque try, intentamos abrir el archivo y leer su contenido utilizando un BufferedReader y un FileReader.

Actividad 4 – “Informe de ejecución y explicación de excepciones en códigos JAVA”

- Dentro del bucle while, leemos el contenido del archivo línea por línea y lo imprimimos en la consola.
- Cerramos el BufferedReader utilizando el método close() para liberar los recursos asociados al archivo.
- En el bloque catch, capturamos la excepción IOException si se produce algún error de entrada/salida al abrir o leer el archivo. El mensaje de error se imprime en la consola.

```
// Excepción 4 con explicación - Excepción de entrada/salida. Presentado por DavidVelásquez.
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ExcepcionEntradaSalida {
    public static void main(String[] args) {
        String nombreArchivo = "archivoInexistente.txt"; // Supongamos que estamos intentando abrir un archivo
        inexistente.

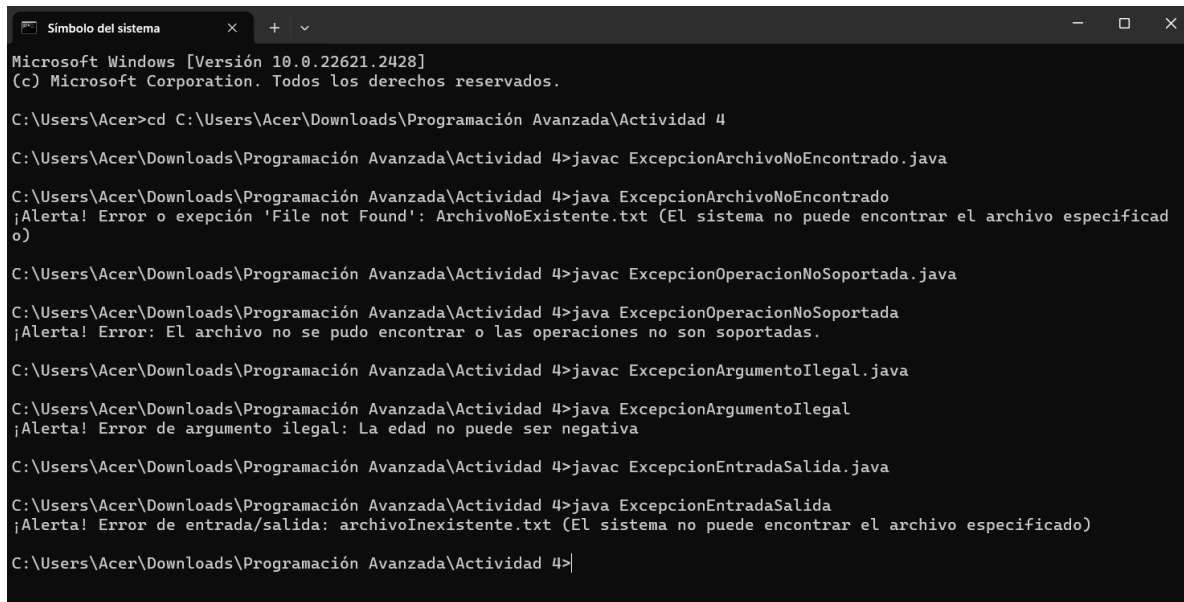
        try {
            BufferedReader lector = new BufferedReader(new FileReader(nombreArchivo));

            String linea;
            while ((linea = lector.readLine()) != null) {
                System.out.println(linea);
            }

            lector.close();
        } catch (IOException e) {
            System.out.println("¡Alerta! Error de entrada/salida: " + e.getMessage());
        }
    }
}
```

Imagen 4 – Ejemplo de excepción – Entrada / Salida

6. Ejecución de los ejemplos de excepciones en CMD



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.22621.2428]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Acer>cd C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>javac ExcepcionArchivoNoEncontrado.java

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>java ExcepcionArchivoNoEncontrado
¡Alerta! Error o excepción 'File not Found': ArchivoNoExistente.txt (El sistema no puede encontrar el archivo especificado)

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>javac ExcepcionOperacionNoSoportada.java

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>java ExcepcionOperacionNoSoportada
¡Alerta! Error: El archivo no se pudo encontrar o las operaciones no son soportadas.

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>javac ExcepcionArgumentoIlegal.java

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>java ExcepcionArgumentoIlegal
¡Alerta! Error de argumento ilegal: La edad no puede ser negativa

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>javac ExcepcionEntradaSalida.java

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>java ExcepcionEntradaSalida
¡Alerta! Error de entrada/salida: archivoInexistente.txt (El sistema no puede encontrar el archivo especificado)

C:\Users\Acer\Downloads\Programación Avanzada\Actividad 4>
```

Imagen 5 – Compilación y ejecución de las excepciones de JAVA anteriormente mencionadas en CMD

7. Carga de la actividad al repositorio en GITHUB

Los archivos de esta actividad han sido añadidos al repositorio personal (dsvelasquezc) y están disponibles en: https://github.com/dsvelasquezc/Actividad_4_Excepciones.git

Actividad 4 – “Informe de ejecución y explicación de excepciones en códigos JAVA”

```
MINGW64:/c:/Users/Acer/Downloads/Programación Avanzada/Actividad 4

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4
$ git init
Initialized empty Git repository in C:/Users/Acer/Downloads/Programación Avanzada/Actividad 4/.git/

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ExcepcionArchivoNoEncontrado.class
    ExcepcionArchivoNoEncontrado.java
    ExcepcionArgumentoIllegal.class
    ExcepcionArgumentoIllegal.java
    ExcepcionEntradaSalida.class
    ExcepcionEntradaSalida.java
    ExcepcionOperacionNoSoportada.class
    ExcepcionOperacionNoSoportada.java

nothing added to commit but untracked files present (use "git add" to track)

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (master)
$ git add .

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (master)
$ git commit -m "Agregar actividad 4"
[master (root-commit) 286c1fd] Agregar actividad 4
 8 files changed, 86 insertions(+)
 create mode 100644 ExcepcionArchivoNoEncontrado.class
 create mode 100644 ExcepcionArchivoNoEncontrado.java
 create mode 100644 ExcepcionArgumentoIllegal.class
 create mode 100644 ExcepcionArgumentoIllegal.java
 create mode 100644 ExcepcionEntradaSalida.class
 create mode 100644 ExcepcionEntradaSalida.java
 create mode 100644 ExcepcionOperacionNoSoportada.class
 create mode 100644 ExcepcionOperacionNoSoportada.java

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (master)
$ git branch -M main

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (main)
$ git remote add origin https://github.com/dsvelasquezc/Actividad_4_Excepciones.git

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (main)
$ git push -u origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 4.52 KiB | 1.51 MiB/s, done.
Total 10 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/dsvelasquezc/Actividad_4_Excepciones.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.

Acer@DavidSantiagoVC MINGW64 ~/Downloads/Programación Avanzada/Actividad 4 (main)
$
```

Imagen 6 – Proceso de carga al repositorio mediante líneas de código

Actividad 4 – “Informe de ejecución y explicación de excepciones en códigos JAVA”

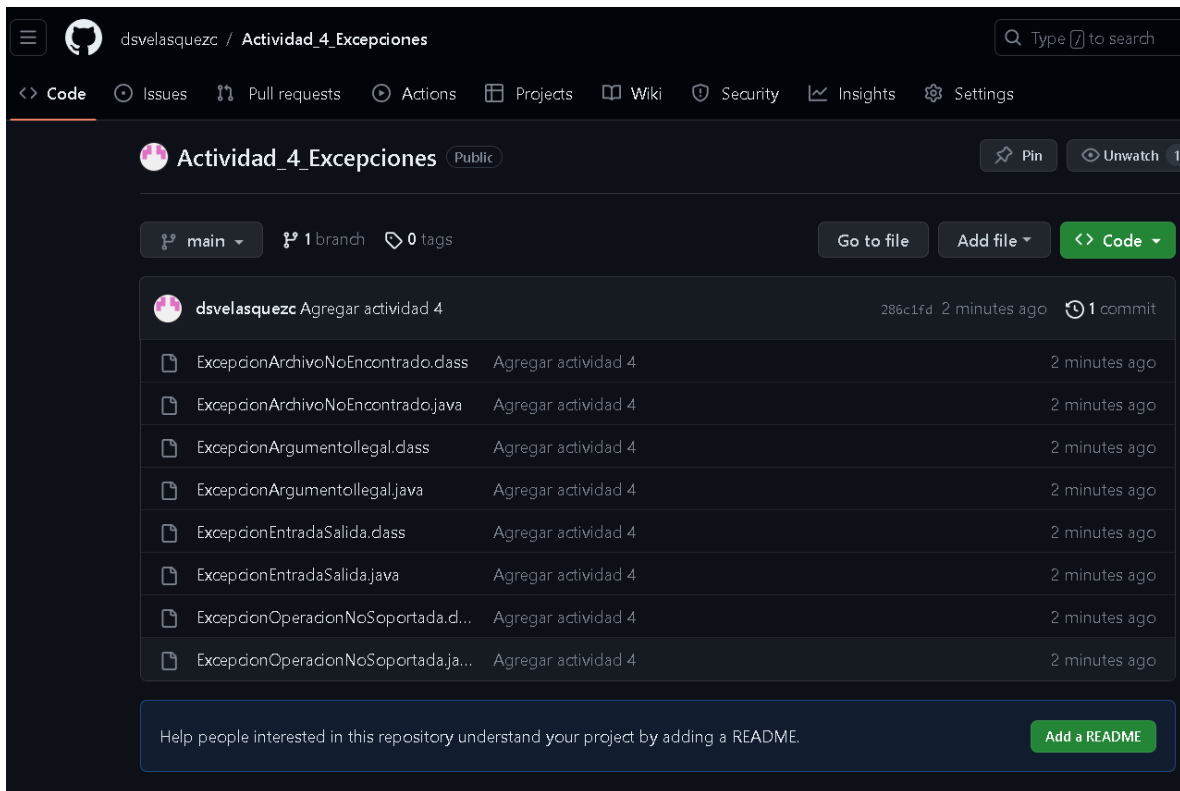


Imagen 7 – Confirmación de la creación del repositorio y la carga exitosa de los documentos