Excepciones

Importancia de la gestión de errores

Gestionar adecuadamente errores es una **tarea crucial en el desarrollo de *software*.**

La gestión efectiva de errores no solo mejora la confiabilidad y seguridad del *software*, sino que también **contribuye significativamente a una experiencia de usuario positiva y a la eficiencia en el desarrollo y mantenimiento de aplicaciones.**

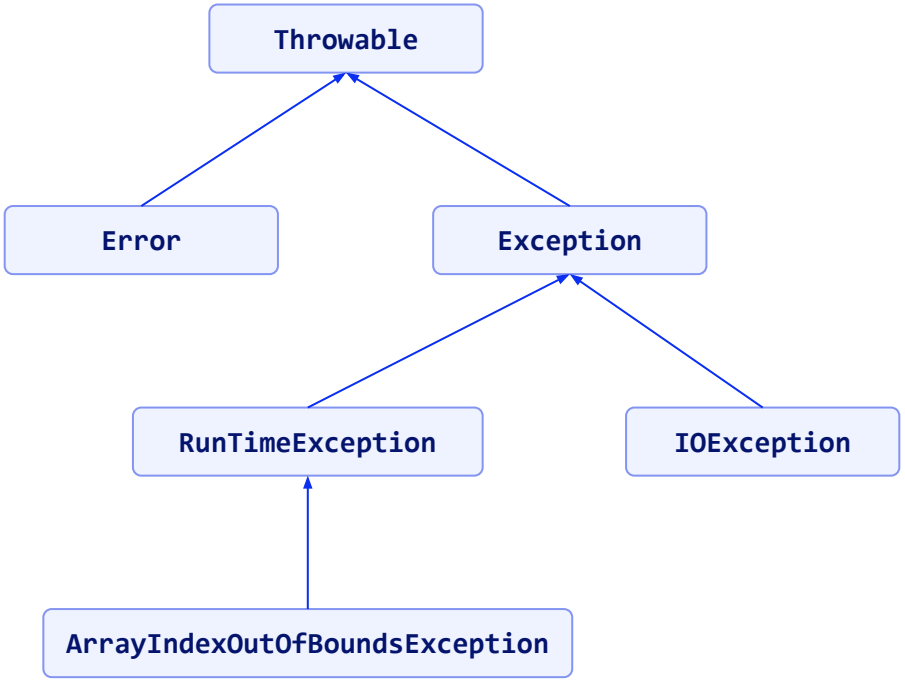
- **Robustez del *software:*** Asegura que pueda recuperarse de situaciones inesperadas sin interrumpir abruptamente su ejecución.
- **Experiencia del usuario:** Proporciona mensajes claros y útiles al usuario final, mejorando la experiencia de uso del *software*.
- **Depuración y mantenimiento:** Facilita la identificación y corrección de problemas durante el desarrollo y el mantenimiento del *software*.
- **Seguridad:** Ayuda a prevenir vulnerabilidades y ataques al manejar situaciones inesperadas.

Excepciones

Las excepciones son **eventos o condiciones anómalas que interrumpen el flujo normal de ejecución de un programa.** Representan **situaciones inesperadas** que pueden surgir durante la ejecución de un programa y que requieren manejo especial.

> Un ejemplo típico de **operación que lanza una excepción** es una división por cero, o convertir en un número una cadena de caracteres con letras.

Las excepciones **permiten al código responder adecuadamente a errores** y situaciones imprevistas. Esto mejora la robustez y confiabilidad del *software.*



Manejo de errores (bloque *try-catch*)

En todos los lenguajes las excepciones se manejan principalmente utilizando el **bloque *try-catch* o *try-except***.

Ese bloque *try-catch* es una estructura utilizada para envolver **segmentos de código donde pueden ocurrir errores o excepciones.**

Permite al programador **ejecutar un bloque de código alternativo y capturar (manejar) cualquier error** que ocurra durante su ejecución.

El uso del bloque *try-catch* permite a los programadores **gestionar de manera estructurada las excepciones** que pueden surgir durante la ejecución del programa. De esta manera, mejora la capacidad de manejo de errores y la robustez del *software.*

Excepciones en Python

```
try:
    # Código que puede lanzar una excepción
    resultado = dividendo / divisor
except ZeroDivisionError as e:
    # Manejo específico de la excepción ZeroDivisionError
    print("Error: División por cero")
```

Excepciones en JavaScript

```
try {
   // Código que puede lanzar una excepción
   let resultado = dividendo / divisor;
} catch (error) {
    // Manejo genérico de la excepción
    console.error("Error:", error.message);
}
```

Excepciones en Java

```
try {
    // Código que puede lanzar una excepción
    int resultado = dividendo / divisor;
} catch (ArithmeticException e) {
    // Manejo específico de la excepción ArithmeticException
    System.out.println("Error: División por cero");
}
```

Excepciones en C#

```
try
{
      // Código que puede lanzar una excepción
      int resultado = dividendo / divisor;
}
catch (DivideByZeroException e)
{
      // Manejo específico de la excepción DivideByZeroException
      Console.WriteLine("Error: División por cero");
}
```

Manejo de errores (bloque *try-finally*)

En todos los lenguajes mencionados, además del manejo de excepciones con *try-catch*, se utiliza el bloque ***try-finally*** **para asegurar que ciertos recursos se liberen correctamente o que se realicen ciertas acciones,** independientemente de si ocurre una excepción.

Propósito del bloque *try-finally*

- **Liberación de recursos**: Archivos, conexiones de red o bases de datos de manera segura.
- **Ejecución de código crítico**: Asegura que ciertas operaciones críticas se ejecuten, como la limpieza de datos temporales o la restauración de estados.
- **Mejora de la confiabilidad:** Incrementa la robustez del *software* al asegurar que el código de limpieza se ejecute siempre.

Excepciones del lenguaje

Son las **excepciones predefinidas** que vienen integradas en el lenguaje de programación.

Ejemplos:

Python	`ZeroDivisionError`, `IndexError`.
JavaScript	`TypeError`, `ReferenceError`.
Java	`NullPointerException`, `IOException`.
C#	`DivideByZeroException`, `ArgumentNullException`.

Excepciones definidas por el usuario

Son excepciones creadas por los programadores para representar **situaciones específicas de la aplicación que no están cubiertas por las excepciones del lenguaje.**

Uso típico: Permiten un manejo más preciso de errores relacionados con la lógica del negocio o condiciones particulares de la aplicación.

Excepciones comprobadas y no comprobadas

En el caso particular de **Java,** se distingue entre estos dos tipos de excepciones:

Comprobadas *(Checked)*	No comprobadas
• Declaradas en la firma del método con **`throws`**. • Verificadas en tiempo de **compilación.** • Obligan a manejarla con el **bloque *try-catch***. • Algunas son: **`IOException`**, **`SQLException`**. • Facilitan identificación de errores en desarrollo. • Aseguran que el manejo de errores críticos.	• No requieren ser declaradas ni manejadas explícitamente en el código. • Verificadas en tiempo de **ejecución.** • Algunas son (Java): **`NullPointerException`**, **`ArithmeticException`**. • Simplifican la escritura de código al no forzar el manejo de todas las excepciones. • Usadas para errores que generalmente son causados por lógica del programa y deben ser corregidos.