**2025**

Entregable tarea: UT6 – Excepciones

**Programación**

**PLATA CORTÉS, ALEJANDRO**

Contenido

[Normas de formato 2](#_Toc182904687)

[Normas de estructura 2](#_Toc182904688)

[Normas de entrega y evaluación de las mismas 2](#_Toc182904689)

# Introducción

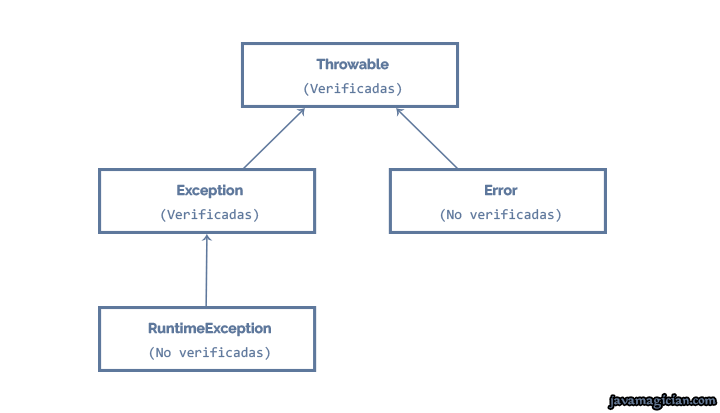
Las excepciones en Java son eventos que ocurren durante la ejecución de un programa y que interrumpen su flujo normal. Estos eventos pueden ser causados por errores en el código, condiciones inesperadas o situaciones externas al programa, como la falta de recursos.​

**¿Qué es una excepción?**

Una excepción es un mecanismo que permite a un método informar sobre un error o situación anómala al código que lo invocó. Cuando se produce una excepción, el programa crea un objeto que encapsula información sobre el error, como su tipo y el estado del programa en el momento en que ocurrió. Este objeto se denomina "objeto de excepción" y es una instancia de una clase que hereda de la clase base *Throwable*.

La clase *Throwable* tiene dos subclases principales: Error y Exception.​

* **Error**: Representa problemas graves que generalmente no pueden ser manejados por la aplicación, como errores de sistema o de recursos. Ejemplos incluyen OutOfMemoryError y StackOverflowError.​
* **Exception**: Representa condiciones que una aplicación podría querer capturar y manejar. Dentro de esta categoría, existen dos tipos principales:​
  + **Excepciones comprobadas (checked exceptions)**: Son excepciones que el compilador obliga a manejar explícitamente, ya sea capturándolas con un bloque try-catch o declarando que el método las puede lanzar con la cláusula throws. Ejemplos incluyen IOException y SQLException.​
  + **Excepciones no comprobadas (unchecked exceptions)**: Son subclases de RuntimeException y no requieren ser declaradas ni manejadas explícitamente. Ejemplos incluyen NullPointerException y ArrayIndexOutOfBoundsException.
* **Excepciones comprobadas (checked):** Estas excepciones se detectan en tiempo de compilación y el compilador exige que sean manejadas explícitamente mediante bloques try-catch o declaradas en la firma del método con la cláusula throws. Son típicas en situaciones fuera del control del programa, como errores de entrada/salida o problemas de red. Ejemplos comunes incluyen IOException, SQLException y ClassNotFoundException.​
* **Excepciones no comprobadas (unchecked):** Estas excepciones ocurren en tiempo de ejecución y no requieren ser declaradas ni manejadas explícitamente. Generalmente resultan de errores en la lógica del programa, como acceder a una posición inválida en un array o intentar operar con un objeto nulo. Ejemplos comunes son NullPointerException, ArrayIndexOutOfBoundsException y ArithmeticException.​



**Excepciones Verificadas**

* IOException: Esta excepción suele indicar que algo en la red, el sistema de archivos o la base de datos falló.

[**7.2.**](https://javamagician.com/java-manejo-excepciones/?utm_source#7-2-excepciones-de-tiempo-de-ejecucion)**Excepciones de Tiempo de Ejecución**

* ArrayIndexOutOfBoundsException: Esta excepción significa que intentamos acceder a un índice de array que no existe, como al intentar obtener el índice 5 de un array de longitud 3.
* ClassCastException: Esta excepción significa que intentamos realizar una conversión ilegal, como intentar convertir una String en una List. Usualmente podemos evitarlo realizando comprobaciones defensivas con *instanceof* antes de hacer la conversión.
* IllegalArgumentException: Esta excepción es una forma genérica de decir que uno de los parámetros proporcionados al método o constructor es inválido.
* IllegalStateException: Esta excepción es una forma genérica de decir que nuestro estado interno, como el estado de nuestro objeto, es inválido.
* NullPointerException: Esta excepción significa que intentamos referenciar un objeto *null*. Usualmente podemos evitarlo realizando comprobaciones defensivas contra *null* o utilizando Optional.
* NumberFormatException: Esta excepción significa que intentamos convertir una String en un número, pero la String contenía caracteres no válidos, como intentar convertir 13f37 en un número.

[**7.3.**](https://javamagician.com/java-manejo-excepciones/?utm_source#7-3-errores)**Errores**

* StackOverflowError: Este error significa que la traza de la pila es demasiado grande. Esto a veces puede ocurrir en aplicaciones masivas; sin embargo, generalmente significa que hay una recursión infinita en nuestro código.
* NoClassDefFoundError: Este error significa que una clase no se pudo cargar ya sea porque no está en el classpath o debido a un fallo en la inicialización estática.
* OutOfMemoryError: Este error significa que la JVM no tiene más memoria disponible para asignar más objetos. A veces, esto se debe a una fuga de memoria.

**Estructura básica para el manejo de excepciones en Java**

Java utiliza los bloques try, catch y finally para manejar excepciones:​

* **try**: Encierra el código que podría generar una excepción.​
* **catch**: Sigue al bloque try y captura excepciones específicas que puedan surgir, permitiendo definir cómo manejarlas.​
* **finally**: Es un bloque opcional que se ejecuta siempre, haya o no una excepción, y es útil para liberar recursos como archivos o conexiones de base de datos.

Existe un tercer tipo de estructura, denominada *try with resources*, que está específicamente diseñada para cerrar todos aquellos elementos que implementen la interfaz AutoClosable (es decir, que creen un flujo abierto), como son las clases InputStream, OutputStream y sus subclases.

# Utilidad del manejo de excepciones

​El manejo de excepciones en Java es fundamental para desarrollar aplicaciones robustas y confiables. Al integrar técnicas adecuadas de gestión de errores, se logra mantener la continuidad del programa y mejorar la experiencia del usuario.​

**Mejora de la experiencia del usuario**

Un manejo efectivo de excepciones permite que, ante situaciones inesperadas, la aplicación informe al usuario de manera clara y precisa sobre el problema ocurrido, en lugar de finalizar abruptamente o comportarse de manera errática. Esto contribuye a una interacción más amigable y profesional. ​[Java Magician](https://javamagician.com/java-manejo-excepciones/?utm_source=chatgpt.com)

**Mantenimiento y legibilidad del código**

Al separar la lógica principal del programa de la gestión de errores mediante bloques try-catch, se mejora la claridad y organización del código. Esto facilita su mantenimiento y reduce la probabilidad de introducir nuevos errores al modificar o ampliar funcionalidades. ​

**Prevención de vulnerabilidades y seguridad**

Un manejo adecuado de excepciones ayuda a identificar y controlar situaciones que podrían ser explotadas por atacantes, como desbordamientos de búfer o inyecciones de código. Al anticipar y gestionar estos escenarios, se refuerza la seguridad de la aplicación.​

En resumen, implementar un manejo de excepciones efectivo en Java no solo garantiza la estabilidad y continuidad del programa, sino que también mejora la experiencia del usuario, facilita el mantenimiento del código y contribuye a la seguridad general de la aplicación.

# Ejemplo

# Bibliografía

1. <https://www.it.uc3m.es/java/prog/resources/excepciones/index_es.html>
2. <https://openwebinars.net/blog/introduccion-a-poo-en-java-excepciones/>
3. <https://picodotdev.github.io/blog-bitix/2017/10/las-excepciones-para-gestionar-errores-en-java/>
4. <https://jarroba.com/excepciones-exception-en-java-con-ejemplos/>
5. <https://javamagician.com/java-manejo-excepciones/?utm_source>