## 

En Java, todas las clases de excepciones heredan de una clase base común: **Throwable**.

**Throwable (Lanzable):** Es la clase raíz de toda la jerarquía de errores y excepciones que pueden ser lanzados en Java. Cualquier objeto que pueda ser lanzado por la sentencia **throw** debe ser una subclase de **Throwable**.

De **Throwable** descienden dos ramas principales: **Error** y **Exception**.

#### Error:Representan problemas graves e irrecuperables que están fuera del control del programador y que generalmente indican fallos críticos en el entorno de ejecución de Java (la Máquina Virtual Java - JVM) o en el sistema.

#### Exception:Representan condiciones que un programa podría querer capturar y manejar. Indican situaciones anómalas que un programa bien diseñado debería ser capaz de gestionar o recuperarse de ellas.

## Excepciones: Manejo de Errores en Programación

Las **excepciones** son un mecanismo fundamental en la programación moderna para **gestionar errores y situaciones anómalas** que ocurren durante la ejecución de un programa. Son **objetos** que se "lanzan" (o "lanzan una excepción") cuando se detecta un problema que impide que el programa continúe su flujo normal. En lugar de que el programa simplemente falle, una excepción permite que el control se transfiera a una parte del código diseñada específicamente para manejar ese tipo de problema.

Cuando una excepción es lanzada, el flujo de ejecución normal se interrumpe, y el sistema de tiempo de ejecución busca un bloque de código (catch en muchos lenguajes) que pueda "capturar" y manejar esa excepción. Si el programa no encuentra un manejador adecuado para la excepción y esta "persiste" (es decir, no es capturada) hasta el punto de entrada principal del programa (como la función main), la aplicación **fallará y se detendrá abruptamente**. Esto a menudo resulta en un mensaje de error o un "stack trace" que puede ser útil para la depuración.

### Tipos de Excepciones

La clasificación de las excepciones varía ligeramente entre lenguajes de programación, pero la distinción principal suele ser entre las que el compilador te obliga a manejar y las que no.

#### 1. Excepciones No Verificadas (Unchecked Exceptions / Runtime Exceptions)

Estas excepciones, a menudo denominadas **RuntimeExceptions** en lenguajes como Java o la mayoría de las excepciones en C#, suelen indicar **problemas que surgen por errores de programación o lógica** que no deberían ocurrir si el código está bien escrito. El compilador **no te obliga a resolverlas** explícitamente (es decir, no tienes que usar bloques try-catch o declararlas en la firma del método).

**Características y Ejemplos:**

* **Errores Lógicos:** Indican situaciones que podrían haberse evitado con una validación adecuada o una lógica correcta.
* **No Obligatorias:** El compilador no te fuerza a manejarlas, lo que permite que el código sea más limpio cuando se espera que estas situaciones sean extremadamente raras o indicativas de un bug grave.
* **Frecuentes en Desarrollo:** A menudo se detectan durante la etapa de desarrollo y pruebas. Suelen señalar defectos en el código.
* **Ejemplos Comunes:**
  + NullPointerException (intentar acceder a un objeto que es null).
  + ArrayIndexOutOfBoundsException (intentar acceder a un índice fuera de los límites de un array).
  + ArithmeticException (ej. división por cero).
  + IllegalArgumentException (un método recibe un argumento inválido).

#### 2. Excepciones Verificadas (Checked Exceptions)

Este tipo de excepciones, presentes en lenguajes como Java (y rara vez en otros como Python o C# donde son menos comunes), representan **situaciones problemáticas que son esperables y sobre las cuales el código cliente puede y debería tomar una decisión**. El compilador **obliga a tratarlas**; es decir, debes manejar explícitamente estas excepciones. Si no lo haces, el código no compilará.

**Características y Ejemplos:**

* **Esperables y Recuperables:** Indican condiciones que están fuera del control directo del programador o que son parte del flujo normal de errores de una operación (ej. problemas de entrada/salida).
* **Obligatorias:** El compilador te fuerza a hacer una de dos cosas:
  1. **Capturarlas y manejarlas (try-catch):** Escribir código para recuperarse del error o al menos gestionarlo de alguna manera.
  2. **Declararlas (throws en la firma del método):** Indicar que tu método puede lanzar esta excepción y que, por lo tanto, el método que te llama deberá manejarla.
* **Fomentan un Código Robusto:** Al obligar al programador a pensar en los posibles fallos, se promueve un manejo de errores más robusto.
* **Ejemplos Comunes:**
  + IOException (problemas al leer o escribir archivos, problemas de red).
  + SQLException (errores al interactuar con bases de datos).
  + FileNotFoundException (intentar abrir un archivo que no existe).

### ¿Por qué esta distinción?

La razón principal de esta distinción es guiar al programador. Las **excepciones verificadas** te obligan a considerar fallos que son parte esperada del entorno (red, archivos, bases de datos) y a diseñar tu programa para que sea resistente a ellos. Las **excepciones no verificadas**, por otro lado, están destinadas a alertar sobre **errores de programación** que deberían ser corregidos durante el desarrollo, en lugar de ser "manejados" en producción, donde podrían ocultar un bug subyacente.

El uso adecuado de las excepciones es crucial para escribir código robusto, mantenible y que pueda recuperarse elegantemente de situaciones imprevistas o, al menos, proporcionar información útil cuando algo sale mal.