

# Universidad Nacional de Ingeniería

Dirección del Área de Conocimiento de  
Tecnología de la Información y  
Comunicación

## Introducción a la programación Unidad II Manejo de Excepciones

Ing. Cristopher Larios



# Excepciones

En Java, una excepción es un evento anormal que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de ejecución. Las excepciones pueden ser causadas por una variedad de situaciones, como errores de programación, condiciones inesperadas en el entorno de ejecución (como la falta de memoria o un archivo que no se puede encontrar), o acciones del usuario.

01

Tipos de excepciones

02

Cláusulas Try-catch-finally

03

Sentencia throw

04

Sentencia throws

# EXCEPCIONES

una excepción es la indicación de un problema que ocurre durante la ejecución de un programa. El manejo de excepciones le permite crear aplicaciones que puedan resolver (o manejar) las excepciones. En muchos casos, el manejo de una excepción permite que el programa continúe su ejecución como si no se hubiera encontrado el problema. Las características que presenta  mos en este cap  tulo permiten a los programadores escribir programas tolerantes a fallas y robustos, que traten con los problemas que puedan surgir sin dejar de ejecutarse o que terminen sin causar estragos.

# TIPOS EXCEPCIONES

Excepciones Comprobadas (Checked Exceptions):

- Las excepciones comprobadas son aquellas que el compilador de Java obliga a manejar explícitamente en el código o a declarar en la firma del método usando la cláusula throws.
- Ejemplos comunes de excepciones comprobadas incluyen IOException, FileNotFoundException, SQLException, entre otras.
- Estas excepciones generalmente indican condiciones que podrían ocurrir durante la ejecución de un programa pero que son predecibles y, por lo tanto, pueden manejarse de manera apropiada.

# TIPOS EXCEPCIONES

Excepciones No Comprobadas (Unchecked Exceptions):

- Las excepciones no comprobadas son subclases de RuntimeException y sus subclases.
- A diferencia de las excepciones comprobadas, el compilador de Java no requiere que se manejen explícitamente mediante bloques try-catch o la cláusula throws.
- Ejemplos comunes de excepciones no comprobadas incluyen NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, ClassCastException, IllegalArgumentException, entre otras.
- Estas excepciones generalmente indican errores de programación o condiciones excepcionales que el programador debería corregir durante el desarrollo del software.

# TIPOS EXCEPCIONES

## Errores (Errors):

- Los errores en Java son situaciones graves que generalmente están fuera del control del programador y no deben manejarse directamente en el código.
- Ejemplos comunes de errores incluyen OutOfMemoryError, StackOverflowError, NoClassDefFoundError, entre otros.
- Estos errores generalmente indican problemas en tiempo de ejecución que pueden estar relacionados con limitaciones del sistema, problemas de configuración, o fallos graves en la infraestructura de la máquina virtual Java (JVM).

# EXCEPCIONES

Captura de excepciones: Para manejar las excepciones lanzadas por el código, se utilizan bloques try-catch. El código que puede generar una excepción se coloca dentro de un bloque try, y las excepciones se capturan y manejan dentro de uno o más bloques catch. Por ejemplo:

```
java

try {
    // Código que puede lanzar una excepción
} catch (Exception e) {
    // Manejo de la excepción
    System.out.println("Ocurrió una excepción: " + e.getMessage());
}
```

# EJEMPLO

Demostraremos primero qué ocurre cuando surgen errores en una aplicación que no utiliza el manejo de errores. En el ejemplo se pide al usuario que introduzca dos enteros y éstos se pasan al un método , que calcula el cociente y devuelve un resultado int. En este ejemplo veremos que las excepciones se lanzan (es decir, la excepción ocurre) cuando un método detecta un problema y no puede manejarlo.

# Rastreo de la pila



## Pila

Rastreo de la pila La primera de las tres ejecuciones de ejemplo muestra una división exitosa. En la segunda ejecución de ejemplo, el usuario introduce el valor 0 como denominador. Se muestran varias líneas de información en respuesta a esta entrada inválida. Esta información se conoce como el rastreo de la pila, la cual lleva el nombre de la excepción (`java.lang.ArithmetricException`) en un mensaje descriptivo, que indica el problema que ocurrió y la pila de llamadas a métodos (es decir, la cadena de llamadas) al momento en que ocurrió la excepción. El rastreo de la pila incluye la ruta de ejecución que condujo a la excepción, método por método. Esta información nos ayuda a depurar un programa.

**/ by zero**

La primera línea especifica que ocurrió una excepción `ArithmetricException`. El texto después del nombre de la excepción (" / by zero") indica que esta excepción ocurrió como resultado de un intento de dividir entre cero. Java no permite la división entre cero en la aritmética de enteros. Cuando ocurre esto, Java lanza una excepción `ArithmetricException`. Este tipo de excepciones pueden surgir debido a varios problemas distintos en aritmética, por lo que los datos adicionales (" / by zero") nos proporcionan



## InputMismatchException

En la tercera ejecución, el usuario introduce la cadena "hola" como denominador. Observe de nuevo que se muestra un rastreo de la pila. Esto nos informa que ha ocurrido una excepción `InputMismatchException` (paquete `java.util`). En nuestros ejemplos en donde se leían valores numéricos del usuario, se suponía que éste debía introducir un valor entero apropiado. Sin embargo, algunas veces los usuarios cometen errores e introducen valores no enteros. Una excepción `InputMismatchException` ocurre cuando el método `nextInt` de `Scanner` recibe una cadena (`string`) que no representa un entero válido.

# EXCEPCIONES

Bloque finally: El bloque finally se utiliza para ejecutar código que debe ejecutarse independientemente de si se produce una excepción o no. Por ejemplo, para asegurarse de que ciertos recursos se liberan correctamente, como cerrar un archivo o una conexión de base de datos.

```
java

try {
    // Código que puede lanzar una excepción
} catch (Exception e) {
    // Manejo de la excepción
    System.out.println("Ocurrió una excepción: " + e.getMessage());
} finally {
    // Código que se ejecutará siempre, independientemente de si se lanza
}
```

# TROWS

Si un método es capaz de provocar una excepción que no maneja él mismo, debería especificar este comportamiento, para que todos los métodos que lo llamen puedan colocar protecciones frente a esa excepción. La palabra clave throws se utiliza para identificar la lista posible de excepciones que un método puede lanzar.

# THROW

La sentencia throw se utiliza para lanzar explícitamente una excepción. En primer lugar se debe obtener un descriptor de un objeto Throwable, bien mediante un parámetro en una cláusula catch o, se puede crear utilizando el operador new.

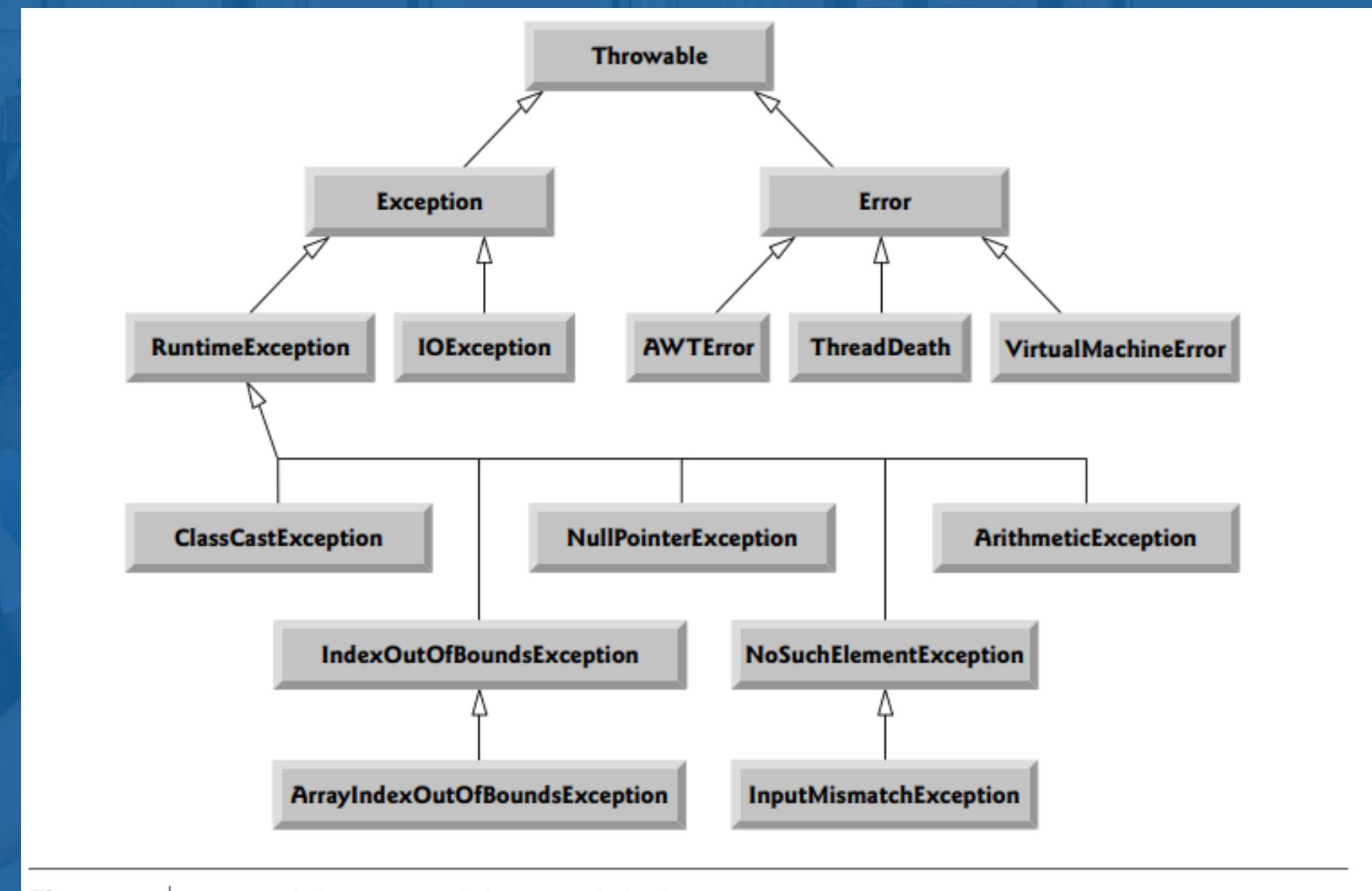


Fig. 11.4 | Posición de las excepciones heredadas de la clase Throwable.



**HASTA LA  
PROXIMA**