



Curso de Java Standard



Ing. Octavio Robleto



octavio.robleto@gmail.com



<https://octaviorobleto.com>



Introducción

En los sistemas pueden ocurrir eventos excepcionales que corten el flujo correcto y provoquen comportamientos inesperados.

Una excepción no es mas que un error que se presenta en nuestro software, estos errores pueden ser mas o menos graves.

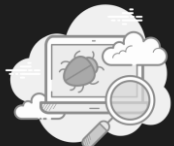
En Java no se pueden evitar pero se pueden gestionar y evitar la interrupción abrupta de nuestro sistema tratando correctamente el problema y así tener un software mas robusto y estable.

```
System.out.println("Inicio del Programa");

Integer a = 10;
Integer b = 0;

System.out.println("El resultado de la division es: " + (a / b));

System.out.println("Fin del Programa");
```



```
Inicio del Programa
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

try, catch y finally

Este bloque esta conformado para capturar los errores y ejecutar las instrucciones deseadas y así poder evitar la interrupción del sistema.

try: contendrá las instrucciones que pueden provocar el error. De este tipo de bloque solo se puede crear uno por grupo.

catch: contendrá el código necesario para gestionar el error.

Si se presenta un error en el bloque try el catch lo capturara creando un objeto según el tipo de excepción esperada, es por ello que este bloque contiene un parámetro y se pueden crear tantos bloques catch crea conveniente.

No es obligatorio tener un catch pero es recomendable hacerlo.

finally: contendrá el código que se ejecutara suceda o no un error, este bloque no es obligatorio y de requerirlo solo puede existir uno al final del try si no posee bloque catch y de poseerlo al final de todo el grupo.

```
System.out.println("Inicio del Programa");

Integer a = 15;
Integer b = 0;
try {
    System.out.println("El resultado de la division es: " + (a / b));
} catch (ArithmeticException e) {
    System.out.println("Error: No se puede dividir por cero un numero entero");
} catch (NullPointerException e) {
    System.out.println("Error: No se puede dividir con un valor nulo");
} finally {
    System.out.println("Fin del Programa");
}
```

```
Inicio del Programa
No se puede dividir por cero numeros de tipo entero
Fin del Programa
```

Existen muchos tipo de excepciones y es imposible numerarlas todas, aprenderemos poco a poco de cuales existen y cuando utilizarlas según van apareciendo y obtenemos experiencia

Métodos

Cuando se captura un error se crea un objeto del tipo indicado en la clausula catch, lo cual nos permite usar métodos de dicha clase para tener un detalle mas amplio del error.

Tipo	Método	Descripción
String	toString()	Devuelve una breve descripción del objeto que capturo el error.
String	getClass()	Devuelve la clase que capturo el error.
String	getMessage()	Devuelve un mensaje con un detalle del error capturado
void	printStackTrace()	Imprime la pila de errores que se produjo .
String	getCause()	Devuelve la causa del error o null si la causa es inexistente o desconocida.

Métodos

```
System.out.println("Inicio del Programa");

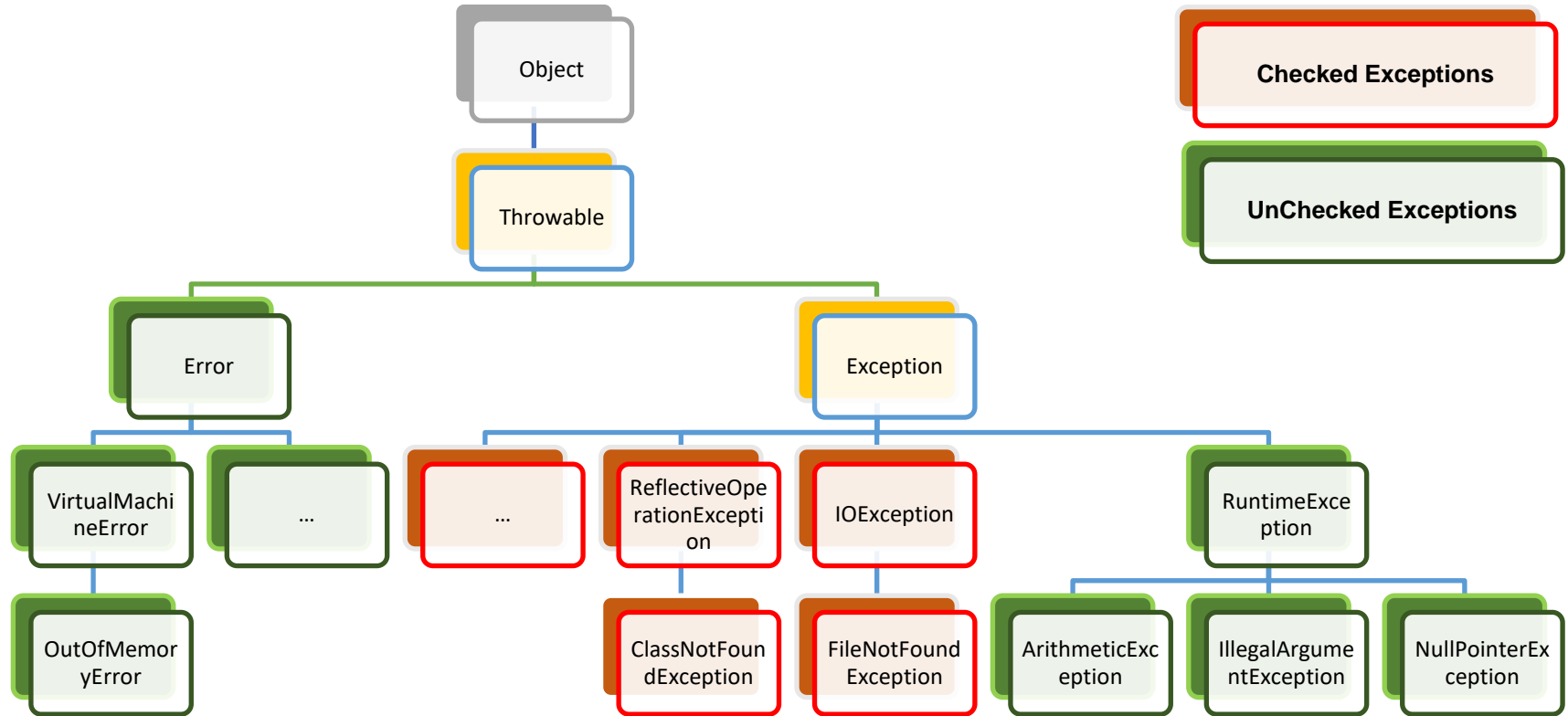
Integer a = 15;
Integer b = 0;
try {
    System.out.println("El resultado de la division es: " + (a / b));
} catch (ArithmeticException e) {
    System.out.println("Error: " + e.getMessage() + ", Causa: " + e.getCause() + ", Clase: " + e.getClass());
} catch (NullPointerException e) {
    e.printStackTrace();
} finally {
    System.out.println("Fin del Programa");
}
```

Inicio del Programa

Error: / by zero, Causa: null, Clase: class java.lang.ArithmeticException

Fin del Programa

Jerarquía de las Excepciones



Checked Exceptions

Son las excepciones que deben ser capturadas o delegadas (punto que hablaremos mas adelante) obligatoriamente por nosotros ya que de lo contrario nos dará un error en tiempo de compilación.

Todas estas excepciones tienen como padre la superclase **Exception**.

```
System.out.println("Inicio del Programa");  
  
FileReader leerArchivo = new FileReader("c:\\\\..");  
  
System.out.println("Fin del Programa");
```

```
java.io.FileReader.FileReader(String fileName) throws  
FileNotFoundException
```

Acá podemos ver que el IDE nos muestra un error al tratar de instanciar un objeto del tipo **FileReader** con el constructor que recibe como argumento la dirección de un archivo, en este caso el JDK nos indica que tipo de problema puede ocurrir (**FileNotFoundException**).

Esta excepción si analizamos tiene sentido, la clase **FileReader** es una clase que trata de leer un archivo y el JDK nos sugiere capturar un error en el caso que la dirección indicada este errada o no exista el archivo a leer.

Unchecked Exceptions

Son las excepciones que tienen como superclase a la clase **RuntimeException**. No es necesario capturarlas, pero al saltar una excepción de este tipo, como todas las excepciones corta el flujo de ejecución, este tipo de excepciones son se suelen llamar excepciones del programador ya que nosotros siempre debemos estar al tanto que ciertos errores pueden suceder, por ejemplo: Error al capturar un dato de tipo numérico y el usuario ingresa una cadena o en un ciclo que recorre un arreglo tratamos de acceder aun índice fuera del tamaño que posee.

```
String[] nombres = {"Octavio", "Macarena"};

for (int i = 0; i < 5; i++) {
    System.out.println(nombres[i]);
}
```

```
Octavio
Macarena
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2
```

Las excepciones de la super clase **Error** son en las que el sistema no puede hacer nada con ellas, son clasificadas como errores irreversibles y que en su mayoría provienen desde la **JVM**,