

Tabla de Contenidos



- [Java RuntimeException](#)
- [Runtime vs Checked](#)
- [Otros artículos relacionados](#)

El concepto de Java RuntimeException o Excepciones sin Checkear (UncheckedException) . Es uno de los conceptos claves a la hora de mejorar el manejo de excepciones en Java. Java en este aspecto es una lenguaje peculiar y que soporta dos tipos de Excepciones a nivel fundamental Checked y UnChecked . Esto siempre complicado de entender . Para abordarlo lo mejor es construir un ejemplo sencillo. Imaginemos que en Java queremos leer un fichero para ello podemos usar el api de Java IO y leer un fichero

```
package com.arquitecturajava;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
public class Principal {
```

```
    public static void main(String[] args) {
```

```
        // Ruta del archivo a leer
        String rutaArchivo = "prueba.txt";
```

```
        // Intenta leer el archivo
        try {
```

```
            // Abre el archivo utilizando FileInputStream
```

```

        FileInputStream fis = new FileInputStream(new
File(rutaArchivo));

        // Crea un lector de buffer para leer el
archivo
        BufferedReader br = new BufferedReader(new
InputStreamReader(fis));

        // Lee cada línea del archivo e imprímela en
la consola

        String linea;
        while ((linea = br.readLine()) != null) {
            System.out.println(linea);
        }

        // Cierra el lector
        br.close();

        } catch (IOException e) {
            // Manejo de excepciones en caso de que ocurra
un error al leer el archivo
            e.printStackTrace();
        }
    }
}

```

Esta es la forma de leer un fichero sencillo linea a linea en Java con un InputStream. El resultado se ve por la consola:

```

hola1
hola2

```

hola3

El código es un poco engorroso al tratarse del API más clásica del lenguaje , nos permite leer línea a línea el fichero e imprimir los resultados en la consola. Eso sí también nos obliga a capturar las excepciones porque una `IOException` es lo que en Java se denomina una `Checked Exception` . Estamos obligados a capturar la excepción sino el código directamente no compila.

Try

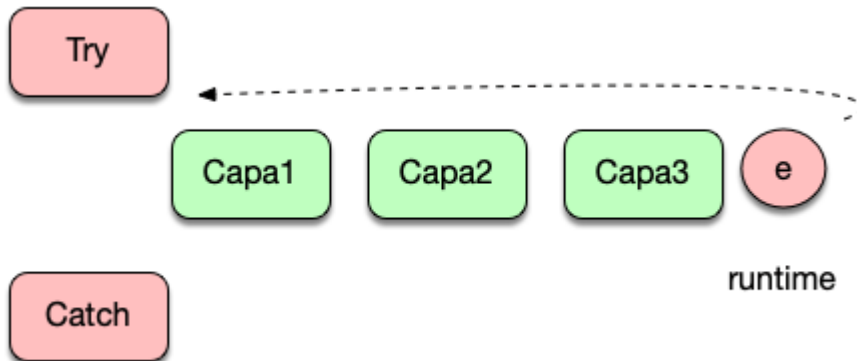
Codigo

Catch

Esto se debe a que el código puede producir un error del que nos podemos recuperar perfectamente es decir podríamos solicitar por la consola otra ruta del fichero para ver realmente cual es el fichero válido. Eso sucede también en muchas ocasiones como una conexión a base de datos `SQLException` . Todas estas excepciones debemos capturarlas ya que en principio indican que hay una situación que es recuperable si cambiamos algo de fuera de nuestro código.

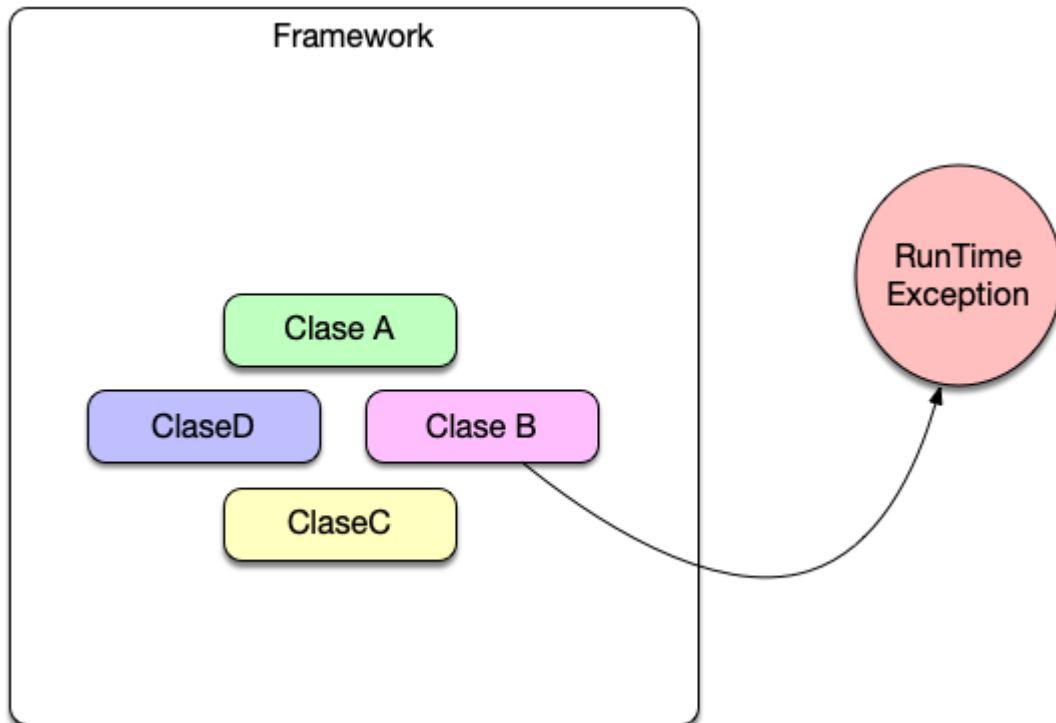
Java RuntimeException

Las excepciones de Runtime son excepciones diferentes estas excepciones cuando se producen no hace falta capturarlas y pueden fluir entre las diferentes capas de la aplicación . Algo que las excepciones chequeadas no pueden hacer y necesitan obligatoriamente de bloques try catch o cláusulas throws que las apoyen. Eso si en un momento determinado el programa deberá capturarlas o sino el programa fallara por completo.



Runtime vs Checked

Muchos frameworks usan `RuntimeExceptions` , incluso muchos lenguajes como C Sharp . Eso hace que en Java nos surja la duda de si debemos usar `CheckedExceptions` como `IOException` o en sistemas más modernos solo `RuntimeExceptions`. Es una buena pregunta y la clave es entender si nos podemos recuperar del error de una forma sencilla. En el caso de la ausencia de un fichero o una consulta a una base de datos es probable que podamos hacerlo de ahí que usemos `CheckedExceptions` . En cambio cuando hablamos de un framework esto se complica porque cuando un framework de un error , es muy difícil recuperarse ya que es la combinación de la acción de muchas clases .



Por lo tanto todos los diseñadores de frameworks optan por que las excepciones sean de tipo Runtime mientras que por otro lado el API fundamental de Java o librerías sencillas como Jackson hacen uso de CheckedExceptions tipo IOException etc .Estas son las diferencias fundamentales

Otros artículos relacionados

- [Eclipse recuperar archivos borrados](#)
- [Java 7](#)
- [Java Try Catch y su manejo](#)
- [Java Stream File y manejo de ficheros](#)
- [Java MultiCatch Block y compactación](#)