

JORGE CARMONA

EXCEPCIONES EN JAVA

INTRODUCCION

“Muchachines y muchachinas... esta semana entra en escena uno de los guardianes del código limpio: ¡el bloque try/catch!”

- Breve repaso de errores comunes en programación.

Pregunta activadora:

- “¿Qué creen que debería pasar si en vez de un número... el usuario escribe ‘perro’?”

QUE ES UNA EXCEPTION?

En simple, es un evento inesperado que interrumpe el flujo normal de un programa.



NO OLVIDAR

Para entender las exceptions, debemos saber dos cosas:

- Jerarquia de clases de Exceptions
- Exceptions chequeadas y no chequeadas
- Bloques try-catch-finaly

JERARQUIA

En Java, todas las excepciones derivan de la clase `Throwable`, que tiene dos grandes ramas:

Class Throwable

java.lang.Object

java.lang.Throwable

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Error, Exception

ERROR

Son situaciones graves que indican problemas del sistema o de la JVM.

Class Error

java.lang.Object
java.lang.Throwable
java.lang.Error

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AnnotationFormatError, AssertionError, AWTError, CoderMalfunctionError, FactoryConfigurationError, ServiceConfigurationError, ThreadDeath, TransformerFactoryConfigurationError, VirtualMachineError

ERROR

Regla: No las atrapes con try/catch. No son para ti. Son para los dioses de la máquina.

Hagamos una prueba con el temible
StackOverflowError

```
java.lang.StackOverflowError Create breakpoint  
at Principal.generarError(Principal.java:13)  
<1,016 folded frames>
```

Process finished with exit code 0

EXCEPTIONS

Estas son las que nos interesan.

Class Exception

java.lang.Object
java.lang.Throwable
java.lang.Exception

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AclNotFoundException, ActivationException, AlreadyBoundException,
BadLocationException, BadStringOperationException, BrokenBarrierException,
DestroyFailedException, ExecutionException, ExpandVetoException, F
InvalidApplicationException, InvalidMidiDataException, InvalidPref
LambdaConversionException, LastOwnerException, LineUnavailableException,
NoninvertibleTransformException, NotBoundException, NotOwnerException,
ReflectiveOperationException, RefreshFailedException, RemarshalException,
TooManyListenersException, TransformerException, TransformException,
UnsupportedLookAndFeelException, URISyntaxException, URISyntaxException

EXCEPTIONS

Esta es la rama que te interesa como desarrolladores. Dentro de Exception hay dos grandes subcategorías:

CH&OQUEDADAS

NO
CH&OQUEDADAS

CHEQUEADAS

Estas deben ser capturadas con try/catch o declaradas con throws, por ejemplo:

`IOException ==> Lectura/escritura de archivos`

`SQLException==>Operaciones con base de datos`

`FileNotFoundException==>Archivo inexistente`

Juguemos con el `IOException`

NO CHEQUEADAS

Estas no es obligatorio capturarlas, pero deberías hacerlo si quieres que tu app no se caiga al primer error tonto.

NULLPOINTEREXCEPTION

ARITHMETICEXCEPTION

**ARRAYINDEXOUTOFBOUND
SEXCEPTION**

NO CHEQUEADAS

Regla de oro

Son descendientes de RuntimeException, y no necesitas try/catch obligado, pero es buena práctica si quieres evitar crasheos.

¿TRI-CATCH?

Concepto general:

- **try** → Aquí va el código que podría fallar.
- **catch** → Aquí va el código que maneja el error si ocurre.
- **finally** → Aquí va el código que SIEMPRE se ejecuta, haya o no haya error.



ESTRUCTURA



```
try {  
    // Bloque de código que puede lanzar una excepción  
} catch (TipoDeExpcion e) {  
    // Bloque de manejo de errores  
} finally {  
    // Bloque que siempre se ejecuta (opcional, pero útil)  
}
```

CAOS MATEMÁTICO

El Mario y el Luigi estan calculando bonus por enemigo derrotado.

Formula:

`vidasExtra = monedas / enemigosDerrotados;`

¡QUÉ SIGNIFICA THROW Y THROWS?

Aunque se parecen, **throw** y **throws** son cosas distintas y se usan en contextos diferentes:

THROW

→

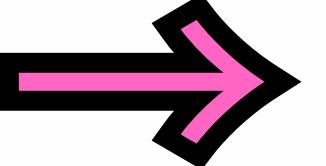
**LANZAR UNA
EXCEPCIÓN
ESPECÍFICA**

THROW

```
throw new NombreDeLaExcepcion("mensaje");
```



THROWS



**DECLARAR QUE
UN MÉTODO
PUEDE LANZAR
UNA EXCEPCIÓN**

THROWS

```
public void miMetodo() throws IOException {}
```



EXCEPCIONES PERSONALIZADAS

Son clases que tú mismo defines, y que extienden una clase de excepción existente, para representar errores específicos de tu sistema.



¿POR QUÉ CREARLAS?

Por que puedes crear tus propias excepciones como por ejemplo `vidasPerdidasException` o `noSeQuePasaException`.



COMO SE CREAN

Extiende una clase base...

- Si tu excepción será chequeada: extiende `Exception`.

Vamos a hacer una excepcion personalizada para un robot de combate.

SEACABO

