Exception handling

Categorías

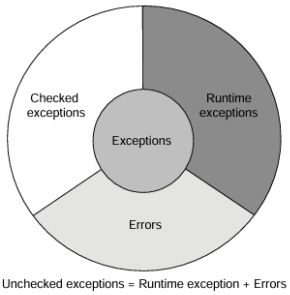

Figure 7.8 Categories of exceptions: checked exceptions, runtime exceptions, and errors

Jerarquía de las clases de excepción

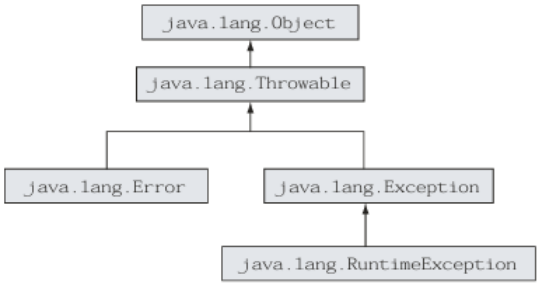

Figure 7.9 Class hierarchies of exception categories

Checked exceptions

Es una condición inaceptable que debe ser prevista por el autor desde que se escribe el código.

Se nombran así porque son revisadas durante el tiempo de compilación.

El compilador se asegura que se esté manejando lo excepción.

Es una subclase de `Exception` , pero no es subclase de `RuntimeException` .

Runtime exceptions

Es una condición que se da por un error de programación o por uso inapropiado de piezas de código.

La excepción se genera en tiempo de ejecución

Es una subclase de `RuntimeException`

Es opcional el manejos de estas, incluso puede considerarse mala práctica.

Los `Errors` suelen incluirse en esta categoría

Errors

Son lanzados por la JMV y son considerados condiciones serías excepcionales y que no pueden ser directamente controladas a través del código.

No es parte de la firma del método.

Reglas de declaración en métodos

Manejar la excepción → Utilizar `try catch`

Declarar el lanzamiento → Añade `throws` a la firma del método.

Maneja y declara → combinación de ambos.

Un método puede declarar el lanzamiento de una excepción, checked o unchecked incluso si no se produce. Pero un bloque try no puede definir un catch con una excepción checked si el bloque try no lanza una excepción.

```
void method10() {
    try {}
    catch (FileNotFoundException e) {}
}
```

Won't compile

Bloques Try catch

Un método puede declarar el lanzamiento de una excepción, checked o unchecked incluso si no se produce. Pero un bloque try no puede definir un catch con una excepción checked si el bloque try no lanza una excepción.


```
void method10() {
    try {}
    catch (FileNotFoundException e) {}
}
```

Won't compile

El bloque `finally` siempre se ejecuta, excepto cuando hay un System.exit.

Se pueden tener multiples catch pero un único `try` y `finally` .

`finally` también se ejecuta incluso si hay un `return` en try o catch

Si catch y finally tienen `return` únicamente se considera el de finally

Si en finally se modifica el valor primitivo que se va a retornar, la modificación se ignora y se retorna el pactado.

Se puede definir únicamente un try con su finally, pero se considera que la excepción no está manejada.

El orden de los catch no importa para clases no relacionadas

Si se coloca la clase superior en el bloque catch antes que la subclase, no compilará por código inalcanzable

Figure 7.15 The order of placement of exception handlers is important.

Los bloques try catch finally no pueden existir independientemente

Finally no puede aparecer antes que catch

Cuando se relanza una excepción en un catch hay que tratarla de nuevo, y si es checked hay que designar responsabilidades.

Excepciones comunes y categorias

Table 7.2 Common errors and exceptions

Runtime exceptions	Errors
ArrayIndexOutOfBoundsException	ExceptionInInitializerError
IndexOutOfBoundsException	StackOverflowError
ClassCastException	NoClassDefFoundError
IllegalArgumentException	OutOfMemoryError
ArithmeticException	
NullPointerException	
NumberFormatException	

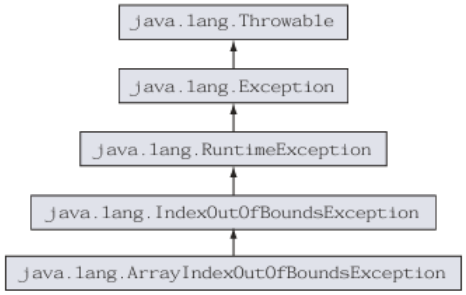

Figure 7.17 Class hierarchy of ArrayIndexOutOfBoundsException

IllegalArgumentException

Se lanza cuando a un método se le pasa un argumento inapropiado.

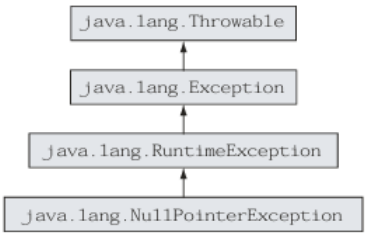

Figure 7.20 Class hierarchy of NullPointerException

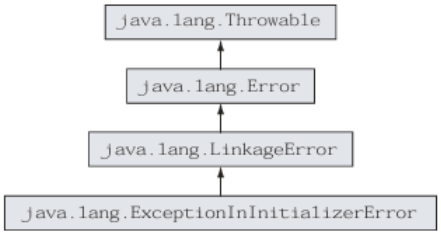

Figure 7.23 Class hierarchy of `ExceptionInInitializerError`

ExceptionInitializerError → Error

Se lanza cuando hay alguna excepcion unchecked en un bloque de inicialización estático

StackOverFlowError - Error

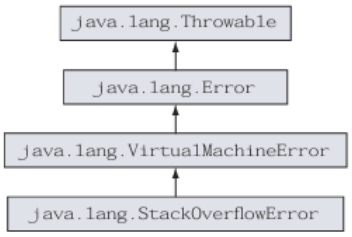

Figure 7.24 Class hierarchy of `StackOverflowError`

Se lanza cuando la memoria se desborda a causa de múltiples llamadas recursivas

OutOfMemoryError → Error

Se produce cuando hay muchos objetos y la memoria se agota.

NoClassDefFoundError - Error

Cuando no se encuentra la clase para usar.