Excepciones en Java

¿Qué es una Exception en Java?

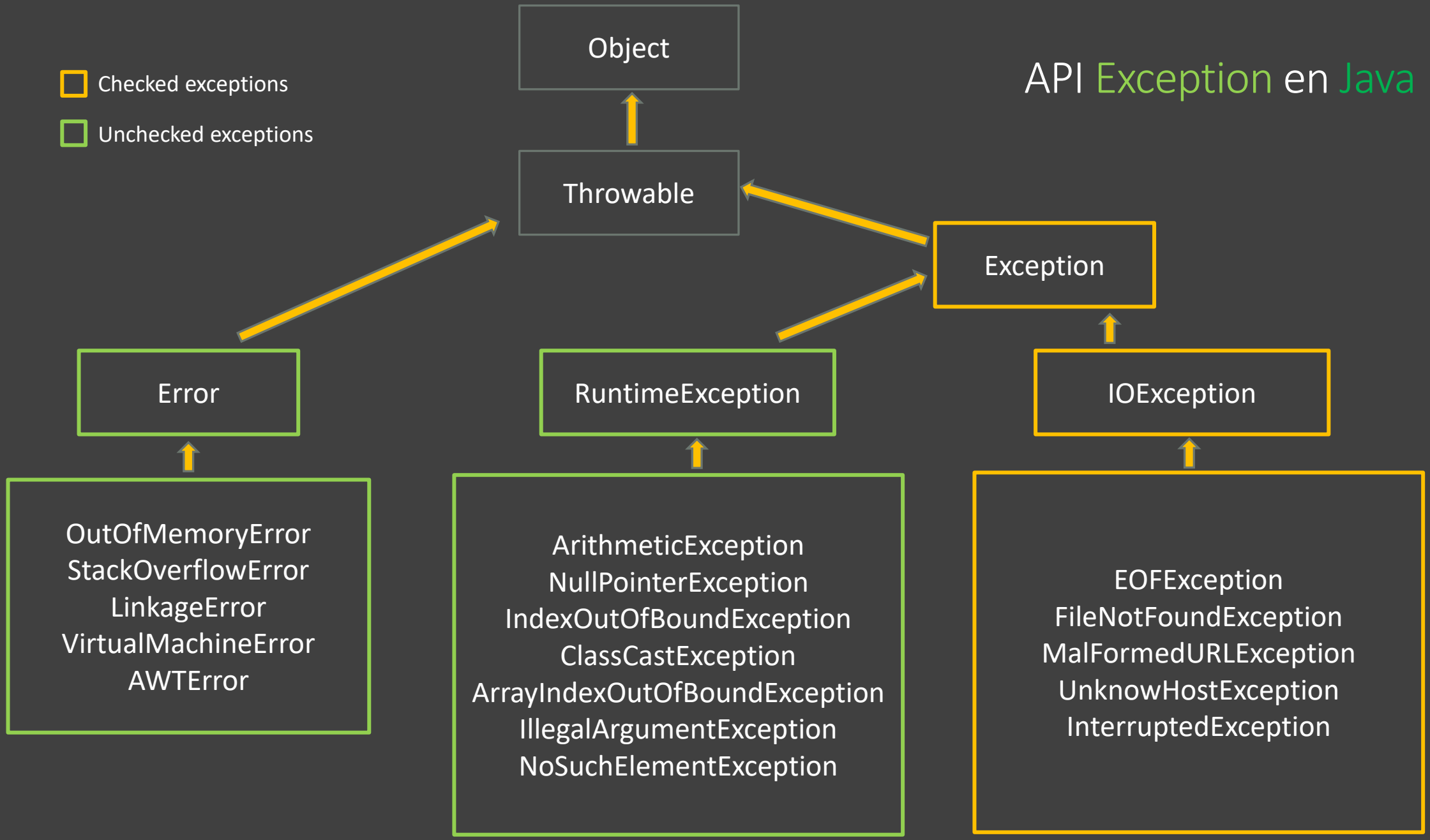
Una Exception en Java es un evento, que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de las instrucciones del mismo.

Cuando se produce un error dentro de un método, éste crea un objeto (Exception) y lo entrega al runtime del sistema.

Este objeto contiene información sobre el error, incluyendo su tipo y el estado del programa cuando se produjo el error. A este proceso crear un objeto Exception y lanzarlo a través del runtime de sistema se le conoce como lanzar una excepción.

Después de que un método lance una excepción, el runtime de sistema intenta encontrar una forma para manejarlo y resolverlo. Para hacerlo recurre a una lista ordenada de los métodos que fueron llamados para llegar al método donde ocurrió el error y buscar si alguno de ellos captura el error y lo resuelve. La lista de métodos se conoce como “Call Stack” o pila de llamadas.

API Exception en Java



Joselu Cantero

Error

Cuando el programa lanza un Error (subclase de Throwable), indica que un problema grave ha sucedido y, según la documentación de Java, la aplicación NO debería intentar solucionar.

Las excepciones de tipo Error son excepciones en las que el sistema no puede hacer nada, son clasificadas como errores irreversibles y que en su mayoría provienen desde la JVM, como por ejemplo: IOError, NoClassDefFoundError, NoSuchMethodError, OutOfMemoryError y VirtualMachineError por mencionar algunos de los errores.

Exception

Cuando el programa lanza una Exception (subclase de Throwable), indica que existe un problema que la aplicación debería tratar o solucionar.

Ejemplo: un error aritmético, un archivo no encontrado…

Tenemos dos tipos de Exception:

1 - RuntimeException: Se producen cuando existen errores propios del programador (división entre 0, tratar de tener acceso a los límites exteriores de un array…).

2 - IOException: son errores que se producen generalmente con eventos relacionados con la entrada/salida (E/S) del sistema y que el programador no puede evitar.

Unchecked exceptions (RuntimeException)

Generalmente este tipo de excepciones son lanzadas por la aplicación y se generan a partir de errores en tiempo de Runtime.

Este tipo de excepciones representan errores en el código y que la aplicación no es capaz de controlar. Algunos de errores causados y que lanzan este tipo de excepciones, por ejemplo, argumentos inválidos pasados a un método (argumentos null pueden causar NullPointerException), otro error común son la excepciones del tipo IndexOutOfBoundsException y que son lanzadas cuando se quieren obtener elementos de una lista y el índice que pasamos está fuera del tamaño de la misma.

Son errores de programación que solamente podemos detectar cuando la aplicación está ejecutándose (runtime) y que no podemos detectar en el proceso de la compilación.

Las excepciones de tipo Unchecked son subclases que heredan desde RuntimeException. Además este tipo de excepciones no tienen la obligación de ser declaradas con la cláusula throws en la cabecera del método. Otra característica es que tampoco se tiene la obligación de atraparlas con un catch.

Checked exceptions (IOException)

Si nuestro programa genera una excepción Checked el compilador solicitará y nos obligará a “hacer algo” con él.

Este tipo de excepción ocurre a pesar de que la lógica de nuestro programa sea correcta. Por ejemplo un error de conexión en una operación E/S (entrada/salida) al utilizar una petición a una base de datos.

Debemos tratar este tipo de errores dentro de un bloque try/catch y/o propagándolos (throws) en la declaración del método hacia el método invocante en la Call Stack.

Las excepciones de tipo Checked deben ser declaradas en la firma del método.
Además deben ser capturadas y tratadas (catch) dentro de un bloque catch.

El Stack Trace

En términos simples, el Stack Trace es una lista de las llamadas que se han producido de unos métodos a otros cuando se produce una Exception. Puede tener un aspecto parecido a esto:

```
Exception in thread "main" java.lang.IllegalStateException: A book has a null property
        at com.example.myproject.Author.getBookIds(Author.java:38)
        at com.example.myproject.Bootstrap.main(Bootstrap.java:14)
Caused by: java.lang.NullPointerException
        at com.example.myproject.Book.getId(Book.java:22)
        at com.example.myproject.Author.getBookIds(Author.java:36)
        ...
```

De esta forma podemos averiguar el origen del error y solucionarlo si fuera necesario. Es una herramienta excelente para el debug de errores: no sólo nos muestra dónde ocurrió el error, sino también el camino que siguió el programa hasta finalizar en un determinado lugar del código.

try/catch

La sentencia catch se utiliza para capturar y tratar las excepciones que se producen en un bloque de código try.

En el momento de producirse la excepción el código del bloque try termina y se ejecuta el bloque catch que recibe como argumento un objeto Throwable.

Ejemplo con try/catch I

```
BLOQUE 1
try{
    BLOQUE 2
}catch(Exception e){
    BLOQUE 3
}
BLOQUE 4
```

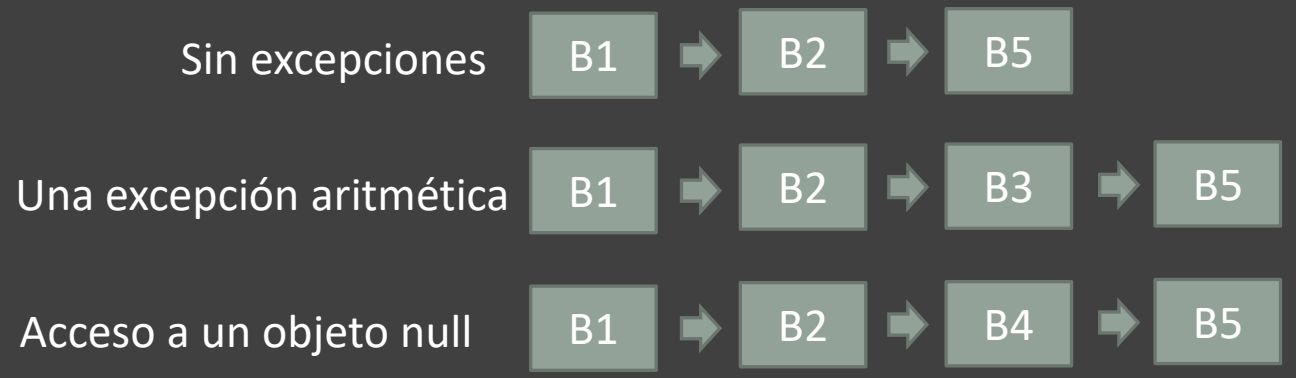
Sin excepciones

Una excepción en el B2

Cualquier excepción que ocurra en B1 no será tratada

Ejemplo con try/catch II

```
BLOQUE 1
try{
    BLOQUE 2
}catch(ArithmeticException ae){
    BLOQUE 3
}catch(NullPointerException ne){
    BLOQUE 4
}
BLOQUE 5
```



Cualquier excepción que ocurra en B1 no será tratada

Cualquier excepción que ocurra en B2 que no sea ArithmeticException o NullPointerException no será tratada

Ejemplo con try/catch III

```
BLOQUE 1
try{
    BLOQUE 2
}catch(ArithmeticException ae){
    BLOQUE 3
}catch(Exception e){
    BLOQUE 4
}
BLOQUE 5
```

Sin excepciones: B1 → B2 → B5

Una excepción aritmética: B1 → B2 → B3 → B5

Exception de otro tipo: B1 → B2 → B4 → B5

Cualquier excepción que ocurra en B1 no será tratada

Ejemplo con try/catch IV

BLOQUE 1

```
try{
```

BLOQUE 2

```
}catch(Exception e){
```

BLOQUE 3

```
}catch(ArithmeticException ae){
```

BLOQUE 4

```
}
```

BLOQUE 5

1º

2º

Las clausulas Check se comprueban en orden
El B4 nunca se llegará a ejecutar!!! (un ArithmeticException es un Exception…)

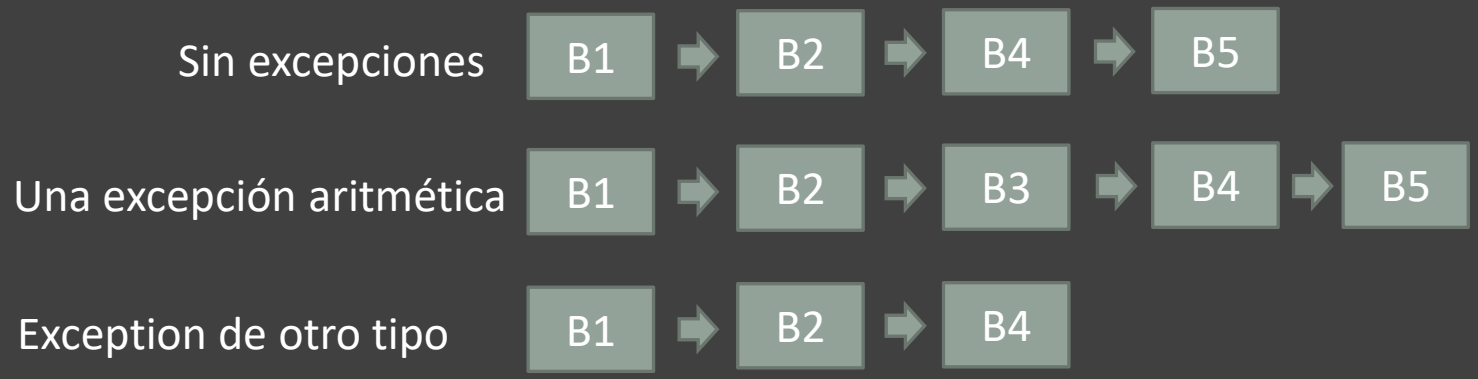
Sin excepciones	B1	B2	B5	
Una excepción aritmética	B1	B2	B3	B5
Exception de otro tipo	B1	B2	B3	B5

Cualquier excepción que ocurra en B1 no será tratada

Ejemplo con try/catch y finally

```
BLOQUE 1
try{
    BLOQUE 2
}catch(ArithmeticException ae){
    BLOQUE 3
}finally{
    BLOQUE 4
}
BLOQUE 5
```

Hay veces que desearemos ejecutar un fragmento de código independientemente de si se produce un Exception. Por ejemplo cerrar una conexión a una BD. En ese caso emplearemos un bloque finally que se ejecutará siempre.



Cualquier excepción que ocurra en B1 no será tratada
Si el B2 comienza a ejecutarse el B4 finally se ejecutará siempre.

La sentencia try con recursos

```
try(

Connection con = ConexionBD.getInstance();
Statement stmt = con.createStatement();
ResultSet result = stmt.executeQuery("SELECT * FROM productos")

){
        CÓDIGO
}catch(SQLException e){
        e.printStackTrace();
}
```

La sentencia try con recursos es una sentencia try que declara uno o más recursos entre los paréntesis que siguen a la palabra try.
Un recurso es un objeto que debe cerrarse después de que el programa haya terminado con él.
La sentencia try con recursos asegura que cada recurso se cierra al final de la sentencia (autoclose). Cualquier objeto que implemente java.lang.AutoCloseable, que incluye todos los objetos que implementan java.io.Closeable, puede ser utilizado como recurso.

La sentencia throw

En Java utilizamos la sentencia throw para lanzar objetos de tipo Throwable en este caso una Exception:

```
throw new Exception ("Este es el mensaje de error");
```

Cuando lanzamos una Exception:

1 - Salimos inmediatamente del bloque de código en el que nos encontramos.
2 - Si el bloque en el que se produce el error tiene asociada una claúsula catch adecuada se ejecutará el bloque de código de dicho bloque catch.
3 - Si no tiene asociada una clausula catch adecuada salimos inmediatamente del bloque (o del método dentro del cual se encuentra el bloque que produjo la excepción) y se busca una clausula catch adecuada recorriendo el orden de llamada de los métodos (Call Stack).
4 - El proceso continúa hasta llegar al método main de la aplicación. Si ahí tampoco existe el catch adecuado la JVM finaliza la ejecución del programa con un mensaje de error.

Propagación de excepciones: throws

Si lanzamos una excepción en el cuerpo de un método (de un tipo derivado de la clase Exception) debemos de añadir en la cabecera del mismo una clausula throws que incluye una lista de los tipos de excepciones que se pueden producir al invocar al método:

public String miMetodo (String a) throws IOException{…dentro del método existe un código que puede lanzar una excepción de tipo IOException…}

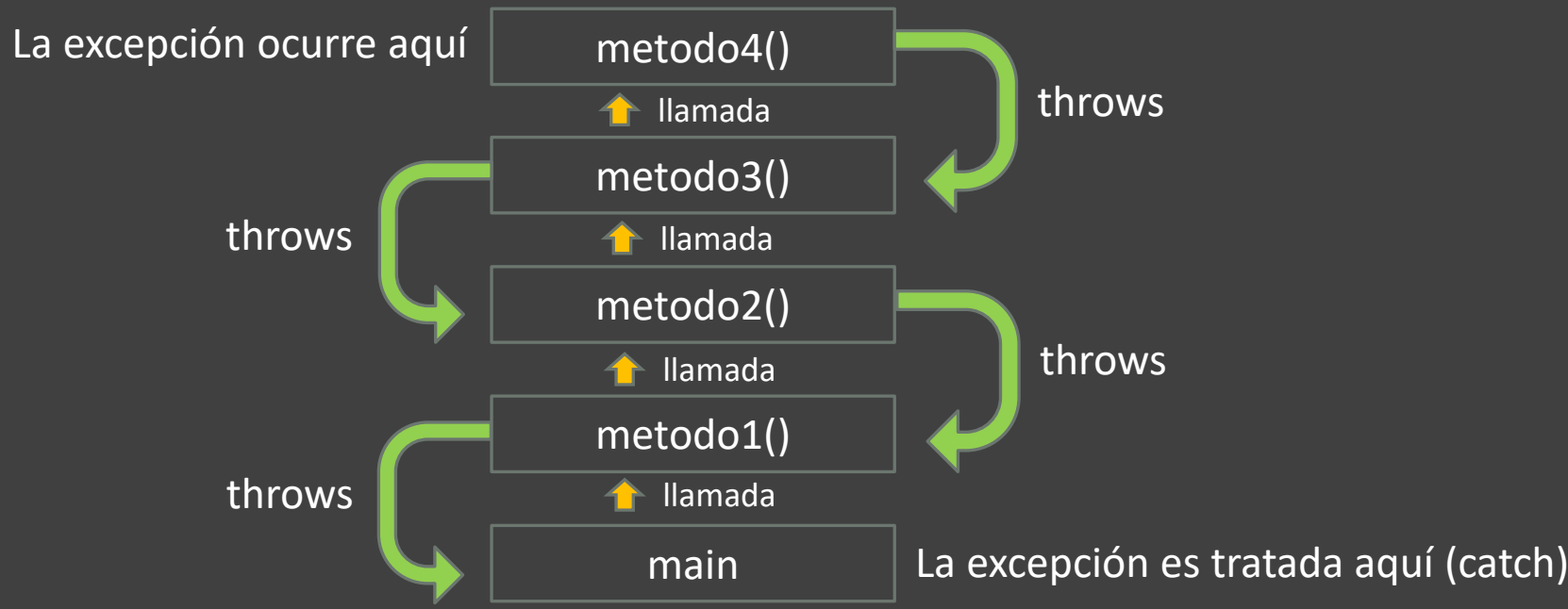
Un método puede lanzar (throws) una excepción por crear explícitamente un objeto Throwable:
throw new Exception("mensaje de error")
… o bien por llamar a un método que genere un error y éste método no lo capture (catch)

Solo es obligatorio incluir esta clausula en las excepciones checked (IOException).
En el caso de las excepciones unchecked (RuntimeException) no es obligatorio.

Cuando implementamos un método debemos decidir si tratamos la excepción dentro del método (con try/catch) o lanzamos la excepción (throws) para que sea otro método el que se haga cargo de ella.

Propagación de una Exception

Los métodos 1, 2, 3 y 4 no tienen un “Exception Handler” y lanzan el error hacia el método que los invocó hasta encontrar un método (en este ejemplo el main) que captura y trata el error.



throw vs throws

throw	throws
Lanza un objeto Exception	Declara un tipo de Exception que es susceptible de ser generado por el método
“throw” es seguido por una instancia (throw new Exception)	“throws” es seguido por el nombre de una clase de Exception (throws IOException, SQLException)
Se usa dentro del método	Se utiliza en la firma del método
No podemos lanzar múltiples Exception	Podemos declarar múltiples excepciones en la firma del método separándolos mediante una coma.
Una excepción “checked” no pude ser propagada utilizando únicamente throw	Una excepción “checked” puede ser propagada mediante throws.

Crear excepciones personalizadas

Podemos definir fácilmente excepciones personalizadas. Simplemente debemos crear subclases de un tipo de excepción ya existente (también podemos extender excepciones personalizadas):

```
public class DividirPorCeroExcepcion extends ArithmeticException{

    public DividirPorCeroExcepcion(String msg){
        super(msg);
    }
}
public class UnaClase {

    public  void unMetodo(Integer numerador, Integer denominador){
        if(denominador == 0){throw new DividirPorCeroExcepcion (“Error, división por cero”);
    }
}
```

Constructor

Malas prácticas en el tratamiento de excepciones I

```
try{
    CÓDIGO QUE LANZA UNA EXCEPCIÓN CHECKED
}catch(Exception e){

}
```

No hacer nada en el catch:

Dejarlo en blanco sin implementación. Siempre se recomienda hacer algo en el bloque catch ya que ocurrió un error y debe ser controlado.

Malas prácticas en el tratamiento de excepciones II

```
try{
    CÓDIGO QUE LANZA UNA EXCEPCIÓN CHECKED
}catch(ArithmeticException ae){
    throw ae;
}
```

Relanzar la excepción capturada al método invocante:

Se presta a confusión al leer el código. Además estaríamos ejecutando dos bloques de código catch para la misma excepción: el del método y el del método invocante.

Malas prácticas en el tratamiento de excepciones III

```
try{
    CÓDIGO QUE LANZA UNA EXCEPCIÓN CHECKED O UNCHECKED
}catch(Exception e){
    CÓDIGO QUE CONTROLA LA EXCEPCIÓN
}
```

No capturar todos los tipos de excepción utilizando Exception:
De esta forma sabemos que se ha producido un error pero no sabemos de qué naturaleza es. Además también capturaríamos excepciones unchecked (runtime) que deberíamos de depurar. Lo óptimo es utilizar todo tipo de excepciones para darle distintos tratamientos.

Malas prácticas en el tratamiento de excepciones IV

```
try{
    CÓDIGO QUE LANZA UNA EXCEPCIÓN CHECKED O UNCHECKED
}catch(Exception e){
    If (e instanceof TalException){
    }else if  (e instanceof CualException){
    }else if (e instanceof FooException){
    }
}
```

No utilizar un control de excepciones “manual”:

Debemos utilizar las herramientas que nos proporciona el lenguaje, por ejemplo con diferentes catch.

Malas prácticas en el tratamiento de excepciones V

```
try{
    CÓDIGO QUE LANZA UNA EXCEPCIÓN CHECKED
}catch(Exception ae){
    throw new RuntimeException("Error");
}
```

Relanzar al método invocante un RuntimeException:

De esta forma el error llegará a la capa más alta sin encontrar un catch y provocará un error general del código. Tenemos que recordar que los RuntimeException son errores de lógica, que no tienen recuperación y que estamos tratando de hacer exactamente lo contrario que es gestionar los errores de la lógica de negocio del programa.