POO Tema 7. Excepciones

Martín González Dios

1 de diciembre de 2024

Una de las tareas importantes que es necesario realizar en el desarrollo de un programa es la **gestión de los errores** que ocurren durante su ejecución, ya que su correcto tratamiento facilita la usabilidad y mantenimiento del programa. Se busca **separar la lógica de la gestión de errores** con la lógica restante.

Parte 1 (arriba). La detección y el tratamiento del error **están mezclados** con el código del método rojo, haciéndolo mucho más complejo entender y mantener. El método verde únicamente se encarga de invocar al método rojo.

Parte 2 (abajo). La detección del error forma parte del código del método rojo, pero es una **parte muy pequeña del código**, lo que simplifica el método si lo comparamos con la versión (1). El método verde mezcla el código de tratamiento del error con su código.

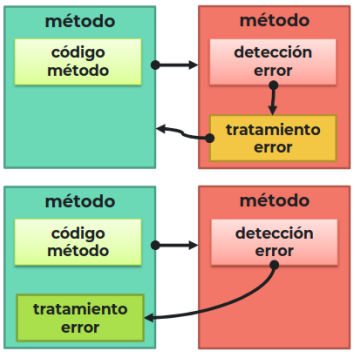

Figura 1: Comparación de metodologías para la gestión de errores

En el código método rojo se mantiene la detección del error para sacar ventaja de la simplificación en su código en relación a la versión (1). El tratamiento del error continua en el código del método verde, pero **existe una clara separación** entre el código de tratamiento del error y el código que proporciona la funcionalidad del método verde. De este modo, un cambio en uno de los dos tipos de código **no afectaría al otro**.

Una **excepción** es la ocurrencia de errores y/o situaciones de interés durante la ejecución de los métodos que proporcionan la funcionalidad del programa. También se relacionan con situaciones a las que se debe dar respuesta y no solo errores.

El mecanismo de Java para el tratamiento de las excepciones aplica la filosofía de **separar el código** para la detección, la comunicación y el tratamiento de las excepciones del código de los métodos que proporciona la funcionalidad del programa.

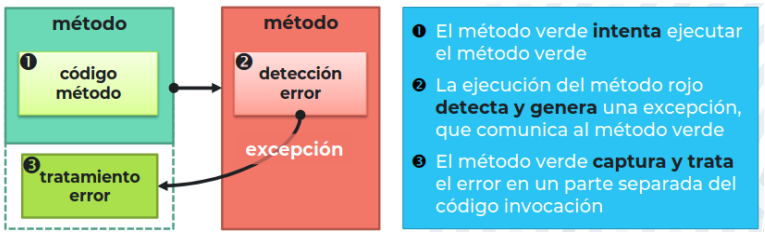

Figura 2: Tratamiento de excepciones en Java

1. Definición de excepciones

Antes de especificar las etapas correspondientes a la gestión de las excepciones en tiempo de ejecución es necesario (1) definir qué excepciones pueden tener lugar; y (2) definir en qué métodos ocurrirán dichas excepciones.

Una excepción se especifica como una clase derivada de la **clase Throwable**, cuyos métodos más usados habitualmente son los siguientes:

- **String getMessage()** devuelve el mensaje que se ha indicado en la ocurrencia de la excepción.
- **void printStackTrace()** imprime en consola la secuencia de invocaciones de métodos que ha llevado a la ocurrencia de la excepción.

En **Java existen 2 tipos de excepciones**:

- **Excepciones chequeadas ("checked")**, se comprueba en tiempo de compilación en qué métodos puede ocurrir una excepción, en cuyo caso **esos métodos deben indicar de forma explícita** esa posible ocurrencia. Son **clases derivadas de la clase Exception**.
- **Excepciones no chequeadas ("uncheked")**, no se comprueba en tiempo de compilación cuáles son los métodos en los que tiene lugar la ocurrencia de la excepción, descargando en el **programador la responsabilidad de revisar** dónde puede ocurrir para intentar capturarla y tratarla. Son **clases derivadas de las clases RuntimeException o Error**.

Excepciones de Java chequeadas	Excepciones de Java no chequeadas
IOException SQLException URISyntaxException ...	ArithmeticException ClassCastException IndexOutOfBoundsException NullPointerException UnsupportedOperationException ...

Figura 3: Ejemplos de excepciones checkeadas y no checkeadas

Si el error o la condición que ha provocado la ocurrencia de la excepción **no impide** la ejecución del programa, la excepción **debería ser del tipo chequeada**. (Ejemplo: NullPointerException es no chequeada porque en pocas ocasiones se puede "arreglar" un nulo de un objeto).

La definición de excepciones como **clases derivadas de la clase Exception** confiere a los programadores un mayor control sobre la gestión de dichas excepciones, puesto que obliga a que los métodos declaren qué excepciones podrían ocurrir durante su invocación.

(**public Exception(String mensaje)**) message es el mensaje con el que típicamente el programador explica por qué ha tenido lugar la ocurrencia de la excepción, que es el momento en el que se crea el objeto de tipo clase derivada de Exception. Este mensaje es lo que habitualmente se le muestra al usuario. Este constructor **debería ser invocado** desde el constructor de la clase derivada de Exception.

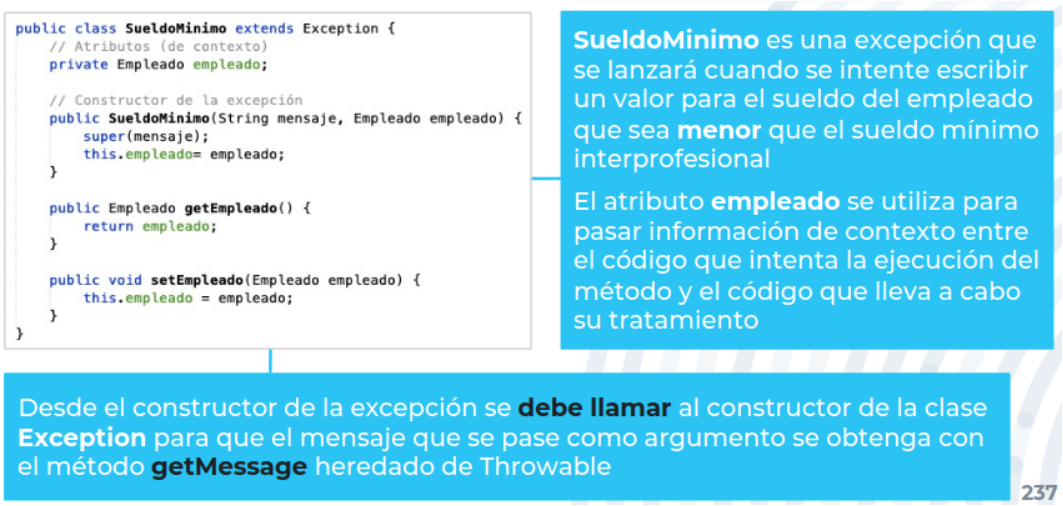

Figura 4: Ejemplo de manejo de excepciones

Para las **excepciones chequeadas**, el compilador exige que se indiquen **explícitamente** los métodos en los que tienen lugar dichas excepciones.

- Se deben **etiquetar** los métodos en los que tienen lugar las excepciones.
- Un mismo método puede estar etiquetado **con más de un tipo de excepción**, es decir, con más de una clase derivada de Exception.
- Un **constructor también puede generar excepciones**.

$< tipo\_acceso > < tipo\_dato >$ nombre_método($< args > *$) throws $< excepcion >$
$< tipo\_acceso >$ nombre_clase($< args > *$) throws $< excepcion >$ (En la signatura del método se deberán de indicar todas las excepciones)

2. Gestión de excepciones: intentar

La gestión de las excepciones comienza con la invocación de un métodoA desde otro métodoB, de modo que se puede decir que el **métodoA intenta ejecutar con éxito el métodoB**. Es necesario indicar en **qué parte del código del métodoA se va intentar la ejecución con éxito del métodoB**, es decir, se intenta la invocación del métodoB esperando obtener el resultado por el cual tiene lugar dicha invocación.

try: en el bloque try se deberá de invocar obligatoriamente al métodoB, aunque lo más deseable es que todo el código funcional del métodoA se sitúe dentro de un único bloque try para que sea más legible.

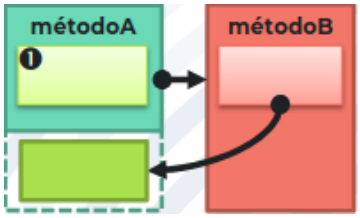

Figura 5: El número 1 indica cuál es la parte "*Intentar*" en la gestión de excepciones

En el código de un puede haber **tantos bloques try como invocaciones a métodos que generan una excepción**, incluso aunque sean varias invocaciones a un mismo método.

```
/* Código método A */
try {
    /* Invocación a método B1 */
} catch(ExcepcionB1 exc) { /* Tratamiento 1 */ }
/* Continúa código método A */
try {
    /* Invocación a método B2 */
} catch(ExcepcionB2 exc) { /* Tratamiento 2 */ }
/* Continúa código método A */
try {
    /* Invocación a método B1 */
} catch(ExcepcionB1 exc) { /* Tratamiento 1 */ }
/* Continúa código método A */
try {
    /* Invocación a método B3 */
} catch(ExcepcionB3 exc) { /* Tratamiento 3 */ }
```

vs.

```
try {
    /* Código método A */
    /* Invocación a método B1 */
    /* Continúa código método A */
    /* Invocación a método B2 */
    /* Continúa código método A */
    /* Invocación a método B1 */
    /* Continúa código método A */
    /* Invocación a método B3 */
} catch(ExcepcionB1 exc) { /* Tratamiento 1 */ }
} catch(ExcepcionB2 exc) { /* Tratamiento 2 */ }
} catch(ExcepcionB3 exc) { /* Tratamiento 3 */ }
```

El código está **unificado** en un único bloque de código

Figura 6: Ejemplo de 2 implementaciones distintas con try

3. Gestión de excepciones: lanzar

Una vez se invoca la ejecución del métodoB, el control del programa pasa al código de dicho método, que tiene dos partes: (1) **la detección** y (2) **la generación** de la excepción para que sea tratada en el método que realizó la invocación.

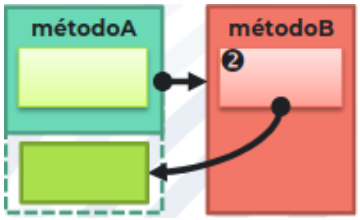

Figura 7: El número 2 indica cuál es la parte "*Lanzar*" en la gestión de excepciones

- La detección de la excepción consiste en especificar dentro del código del métodoB las **condiciones** en las cuales se genera la excepción. (Si el numerador de la división es 0)
- Al generar la excepción se completa lo que se entiende por **lanzar la excepción**, que consiste en crear un objeto del tipo de excepción que se transferirá al trozo de código en el cual tiene lugar el tratamiento de la excepción. **Cuando se crea el objeto del tipo excepción**, es habitual **incluir en ella información** sobre el contexto en el que ha ocurrido, incluyendo (a) una descripción textual sobre la causa que la ha provocado; y/o (b) referencias a objetos que se podrán usar en el tratamiento de la excepción.
  Al lanzar una excepción se interrumpe la ejecución del método en el mismo punto en el cual se ha lanzado, es decir, el flujo de programa abandona el método y apunta directamente al código en el que se lleva a cabo el tratamiento de la excepción.

```
if(<condición>) {
    <tipo_excepción> <nombre_objeto> = new <constructor>;
    throw <nombre_objeto>;
}
```

Figura 8: Uso de la palabra throw

La palabra **throw** se utiliza para indicar que se interrumpe la ejecución del método y se lanza la excepción para que pueda ser capturada por el código en el que se trata la excepción, que es el **punto en el que continúa** la ejecución del programa.
La información del contexto de ocurrencia de la excepción **se pasa en los argumentos del constructor** de la clase asociada a la excepción.

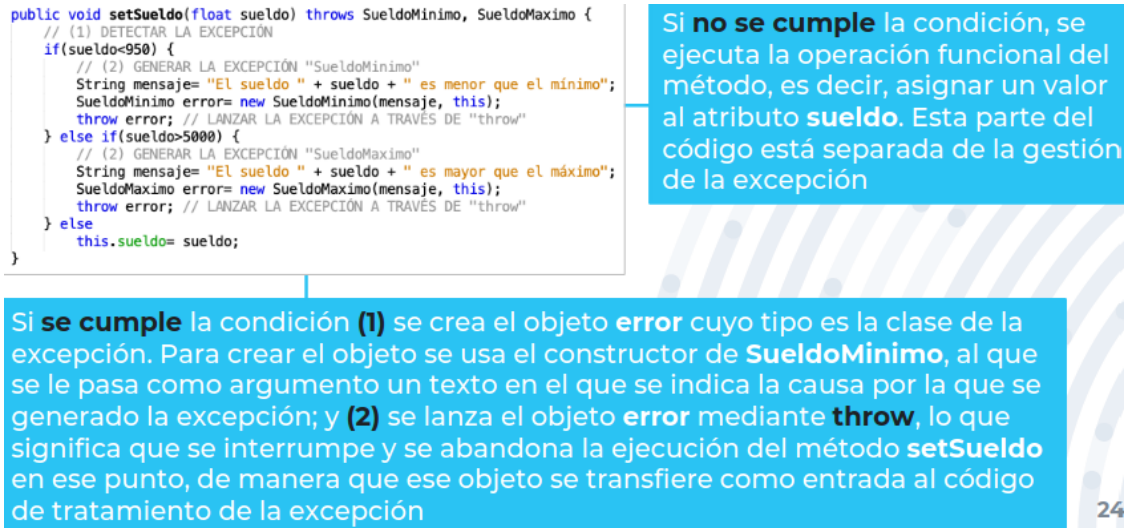

Figura 9: Ejemplo de throw

4. Gestión de excepciones: tratar

Una vez se lanza una excepción, el objeto de tipo excepción se transfiere **como "argumento"** al código en el que se realiza su tratamiento, de modo que dicho código tiene información contextual sobre la ocurrencia de la excepción.
El momento en el que el flujo de programa entra en el código para el tratamiento de la excepción se denomina **captura de la excepción**. El tratamiento de la excepción tiene lugar en los denominados **bloques catch**, los cuales **no pueden existir** sin haber especificado previamente un bloque try.

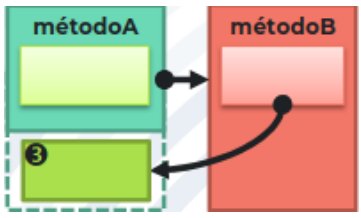

Figura 10: El número 3 indica cuál es la parte "*Tratar*" en la gestión de excepciones

Restricciones entre los bloques try-catch:

- Si existe un bloque catch deberá existir necesariamente **un único bloque try**.
- Un bloque try deberá tener asociado, **al menos**, un bloque catch.
- Un bloque try **podría tener tantos** bloques catch como el número de tipos de excepciones se puedan lanzar en el bloque try.
- Un bloque try **podría tener menos** bloques catch que el número de excepciones que se hayan lanzado en el bloque try. (Si las excepciones que se lanzan tienen la misma clase base, entonces puede definirse **un único bloque catch**) (Si el tratamiento es el mismo puede usarse **multi-catch**)

La **opción multi-catch** no añaden ningún tipo de semántica a la gestión de la excepción, sino que está orientada a simplificar el código del bloque catch.

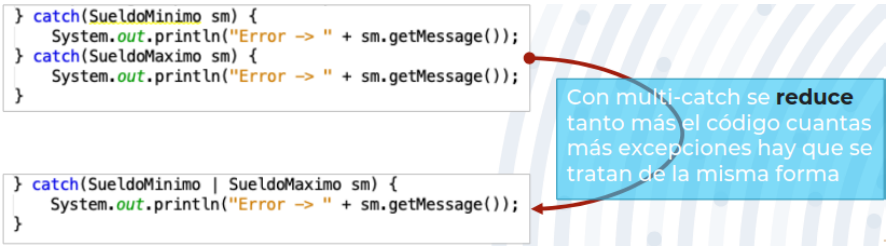

Figura 11: Ejemplo de multi-catch

Es importante el papel de la **herencia y el polimorfismo** a la hora de decidir qué bloque catch se ejecutará cuando se lanza una excepción. Si el argumento del bloque catch es una de las clases superiores a una claseA, después de ese catch **no podrá existir** un catch con la claseA, puesto que la excepción se habrá capturado en el catch de la clase superior.

En el tratamiento de las excepciones, además de los bloques catch también existe el **bloque finally**, que se ejecuta siempre; independientemente de la excepción que se haya capturado o incluso aunque no se haya capturado ninguna excepción.

- Un bloque try puede tener asociado un **único bloque finally**.
- El bloque finally siempre **deberá ir después** de todos los bloques catch asociados al tratamiento de la excepción.
- No es posible definir un bloque finally si no existe, **al menos**, un bloque catch el que se capturen las excepciones generadas en el bloque try.
- Uno de los usos habituales del bloque finally **es liberar recursos**.

5. Como gestionar la generación de excepciones

Un métodoA que invoca a un métodoB que lanza una excepción **no debe tener necesariamente un try–catch** como parte de su código, es decir, el métodoA no tiene por qué realizar la captura de la excepción. **El métodoA puede delegar el tratamiento de la excepción a otro métodoC que lo invoque**, de modo que existe un encadenamiento lanzamientos de excepciones que tiene como fin la unificación del código en el cual se realizar el tratamiento final de la excepción.

Figura 12: Ejemplo de generación de excepciones estructurado