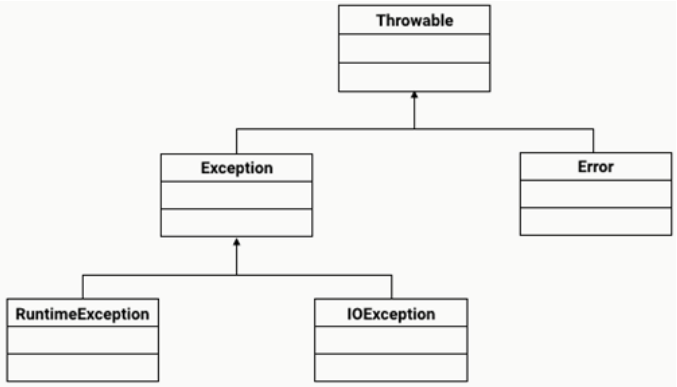
Manejo de excepciones

Manejo de excepciones

Jerarquía de clases

Todo en Java es jerárquico. Tiene mucho sentido cuando lo pensamos. De lo contrario, es posible que el código no se pueda mantener a largo plazo. En este caso, la clase *Throwable* representa todo lo que se puede "lanzar" en Java. Contiene una instantánea del estado de la pila de memoria en el momento en que se creó el objeto, la cual se conoce también como seguimiento de pila o cadena de llamada. En el proceso, guarda un mensaje de tipo *String* que podemos usar para detallar qué salió mal (Galán, 2019).



¿Qué es un error y qué es una excepción?

Las otras dos subclases se derivan de la clase principal. Hay bastantes personas que tratan a ambas de la misma manera, pero, en realidad, se debe distinguir entre las dos. Por ejemplo, cuando hablamos de un Error, nos referimos a subclases que indican problemas graves en nuestra aplicación. Este problema no se puede solucionar de ninguna manera, por lo que el programa suele detenerse. La excepción y sus subclases indican que la aplicación debe intentar resolver las condiciones "incorrectas" de forma transparente para el usuario, emitiendo una advertencia (Galán, 2019).

Hay dos tipos de excepciones: ***Runtime Exception*** (errores de programa) como el acceder fuera de los límites de una matriz o hacer algo tan simple como intentar dividir entre cero. Y ***IOExceptions***, errores de entrada y salida que normalmente no preocupan al programador (Galán, 2019).

Atrapar excepciones: Bloques *try... catch*

Según Galán (2019), atrapar excepciones es casi como pescar. En tal situación, debemos lanzar nuestras redes para estar preparados. En Java, usamos los términos *try* y *catch* para detectar las excepciones que pueden haberse producido con el fin de aislar el código que queremos probar. Cuando ocurre una excepción, "intentar" finaliza y "atrapar" recibe un objeto *Throwable* como parámetro. Por ejemplo:

```
1   // Bloque 1
2   try {
3   // Bloque 2
4   } catch (Exception error) {
5   // Bloque 3
6   }
7   // Bloque 4
```

Ahora, la pregunta es, por supuesto, cuántos tipos de excepciones podemos capturar o filtrar. La respuesta: tantos como necesitemos. Tal como lo ejemplifica Galán (2019):

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

Quizás necesitemos ejecutar un fragmento de código independientemente de las excepciones, como cerrar el archivo en el que estamos trabajando o una conexión a una base de datos. Para esto, podemos agregar una cláusula *finally* como la sugerida por Galán (2019):

```
// Bloque1
try {
// Fragmento de código que puede
// lanzar una excepción de tipo IOExcepti
// (p.ej. Acceso a un fichero)
} catch (IOException error) {
// Tratamiento de la excepción
} finally {
// Liberar recursos (siempre se hace)
}
```

Referencias bibliográficas

- Galán, D. (2019). *Uso de excepciones en Java*. Recuperado dehttps://bit.ly/43mlMlj