# Manejando errores

Aclaración: Este breve tutorial intenta explicar el ABC del uso de excepciones en java. Es muy inicial y no abarca ni la creación de excepciones propias, ni como manejar más de un tipo de excepción, tampoco cómo hacer wrapping de excepciones (capturar y relanzar una nueva a partir de la capturada); ni el uso de unchecked exceptions.
Para ampliar el conocimiento del uso de excepciones consultar libros o tutoriales más detallados.

Es algo normal cuando se desarrolla software tener que enfrentar situaciones ante las cuales pueden ocurrir errores, por ejemplo en el siguiente código

```
public Integer dividir(Integer dividendo, Integer divisor){
        return dividendo/divisor;
}
```

Qué ocurre si el valor de *divisor* es 0? por supuesto que al intentar dividir por cero tendríamos un error en tiempo de ejecución. Entonces lo que se hace es validar antes de hacer la división que *divisor* no tenga como valor 0:

```
public Integer dividir(Integer dividendo, Integer divisor){
      if(divisor!=0){
          return dividendo/divisor;
      }
      return 0;
  }
```

Ahora estoy seguro que nunca voy a dividir por cero, pero tengo un problema nuevo: en caso que *divisor* sea cero, que valor devuelvo?
El problema es que al devolver un valor dentro del universo de los resultados posibles podría caer en el problema de no poder distinguir un caso de error de uno válido:

```
calculadora.dividir(2, 0);
calculadora.dividir(0, 2);
```

En ambos casos el resultado es 0, y en un caso es una situación de error y en el otro no.

Por este motivo es que no debe utilizarse el valor de retorno de un método para manejar errores, para eso se utilizan las Excepciones.
Las Excepciones son un conjuntos de clases que nos permiten manejar situaciones que podrían generar un error. Decimos “podrían” porque al momento de escribir el método *dividir*, no sé con qué valores será llamado dicho método, por lo tanto algunas invocaciones al mismo generarán error y otras no.

Lo que se hace es en caso de detectar una situación no deseada se lanza una excepcion. primero se instancia un objeto del tipo Exception y luego se lanza (linea 5):

```
public Integer dividir(Integer dividendo, Integer divisor) throws Exception{
      if(divisor!=0){
          return dividendo/divisor;
      }
      throw new Exception("El divisor no puede ser 0");
  }
```

Cuando un método puede lanzar excepción debe indicarse dicha situación en el encabezado del mismo utilizando la palabra reservada *throws*, lo cual me dice “el método dividir puede lanzar una exception”

Ahora bien, ya sabemos cómo lanzar una excepción en caso de error, pero que debe hacer quién llame al método *dividir*? Tiene dos opciones: relanzarla o tratarla

Para relanzarla sólo hace falta agregar el throws en el método que usa (en nuestro ejemplo) al método dividir. En este caso, si el método dividir lanzara una excepción, la misma será relanzada por *metodoCliente* a su llamador.

```
 public void metodoCliente() throws Exception {
      Claculadora calculadora = new Claculadora();
      calculadora.dividir(4, 2);
  }
```

Nótese que el método *dividir* está siendo llamado con valores válidos. El throw (o try-catch como veremos a continuación) debe colocarse siempre, independientemente de los valores con los que se llame al método

En cambio si quiero tratarla, debo utilizar el bloque try-catch, en este caso el método dividir está vigilado, en caso que no haya error se ejecuta el código dentro del try, y en caso que se lance una excepción se ejecuta el código dentro del catch

```
public void metodoCliente() {
      Calculadora calculadora = new Calculadora();
       try {
          calculadora.dividir(4, 2);
      } catch (Exception e) {
          // hacer algo con la excepcion
          System.out.println("el error es: " + e.getMessage());;
      }
  }
```