Excepciones y Clases Utilitarias

Excepciones

Las *excepciones* son eventos que alteran el flujo del programa.

Tipos de excepciones:

Excepciones chequeadas (Checked)

Son aquellas que derivan de la clase **Exception**. Deben ser manejadas o declaradas, es decir, **requieren utilizar el bloque try/catch**. Entre las clases más comunes encontramos:

- **FileNotFoundException → lanzada programáticamente cuando el código trata de referenciar a un archivo que no existe**
- **IOException**→ lanzada cuando hay un problema leyendo o escribiendo un archivo

Excepciones no chequeadas (Unchecked)

Derivan de la clase **RuntimeException**. No deben ser manejadas o declaradas, es decir, no requieren el uso obligatorio del bloque try/catch. Las más comunes son:

- **ArrayIndexOutOfBoundsException → lanzada por la JVM cuando se utiliza un índice no permitido al acceder un array**
- **IllegalArgumentException → lanzada por el programador para indicar que se ha pasado un argumento inapropiado o no permitido a un método**
- **NullPointerException → lanzado por la JVM cuando la referencia a un objeto es nula al momento de requerir el objeto**

// **Excepciones no chequeadas y chequeadas**

```java
int valor = 0;

double resultado = 10 / valor;
```

```java
try {
   FileInputStream fileInputStream = new FileInputStream("prueba.txt");
} catch (FileNotFoundException exception) {
   System.out.println("El archivo indicado no existe");
}
```

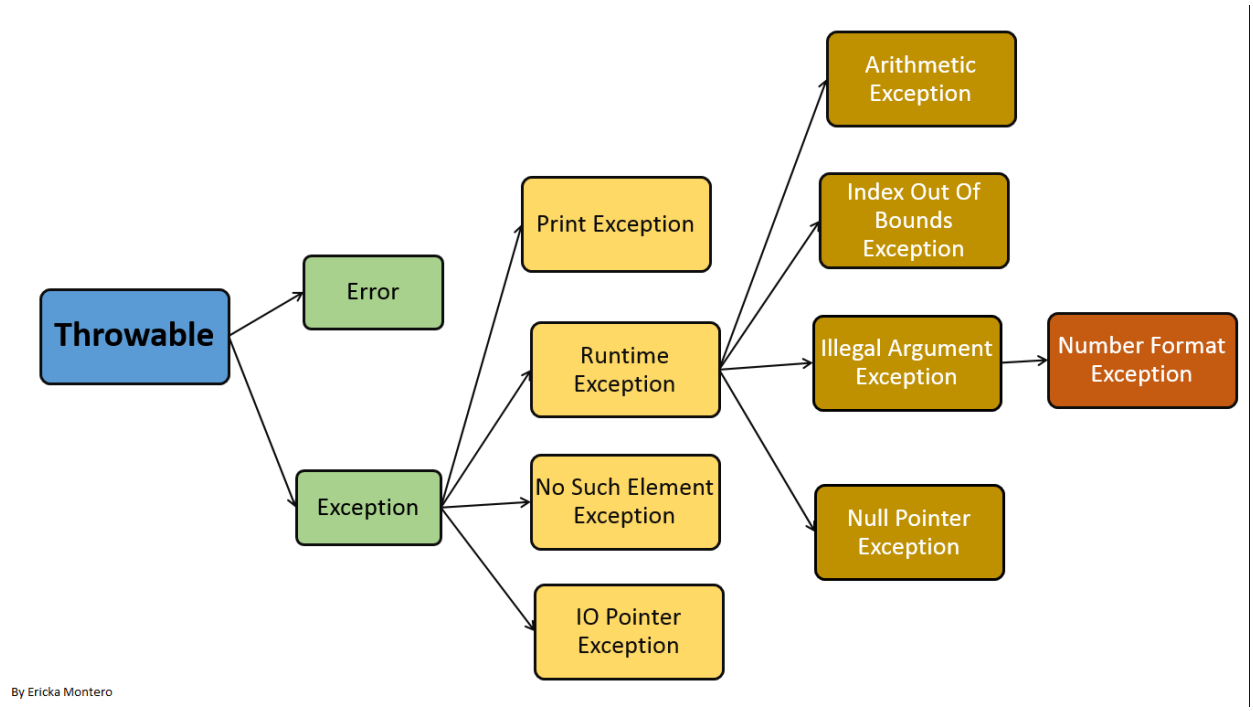
Errores

Los errores derivan de la clase **Error**, son conocidos por ser lanzados por la JVM, y no pueden ser resueltos o solucionados, por lo que el programa normalmente se detiene. Indican problemas graves en nuestra aplicación, suelen ser poco comunes, pero podrías llegar a ver alguno de los siguientes:

- **ExceptionInInitializerError → lanzado por la JVM cuando un inicializador estático lanza una excepción y el mismo no se maneja**
- **StackOverflowError → lanzado por la JVM cuando un método se llama a sí mismo demasiadas veces (esto es llamado recursividad infinita porque el método suele llamarse a sí mismo sin fin)**
- **NoClassDefFoundError** → lanzado por la JVM cuando una clase que utiliza el código está disponible en tiempo de compilación pero no en tiempo de ejecución

Jerarquía de excepciones

En Java, todas las excepciones están representadas por **clases**. Las clases de excepción derivan de una clase llamada **Throwable**.



Stack Trace

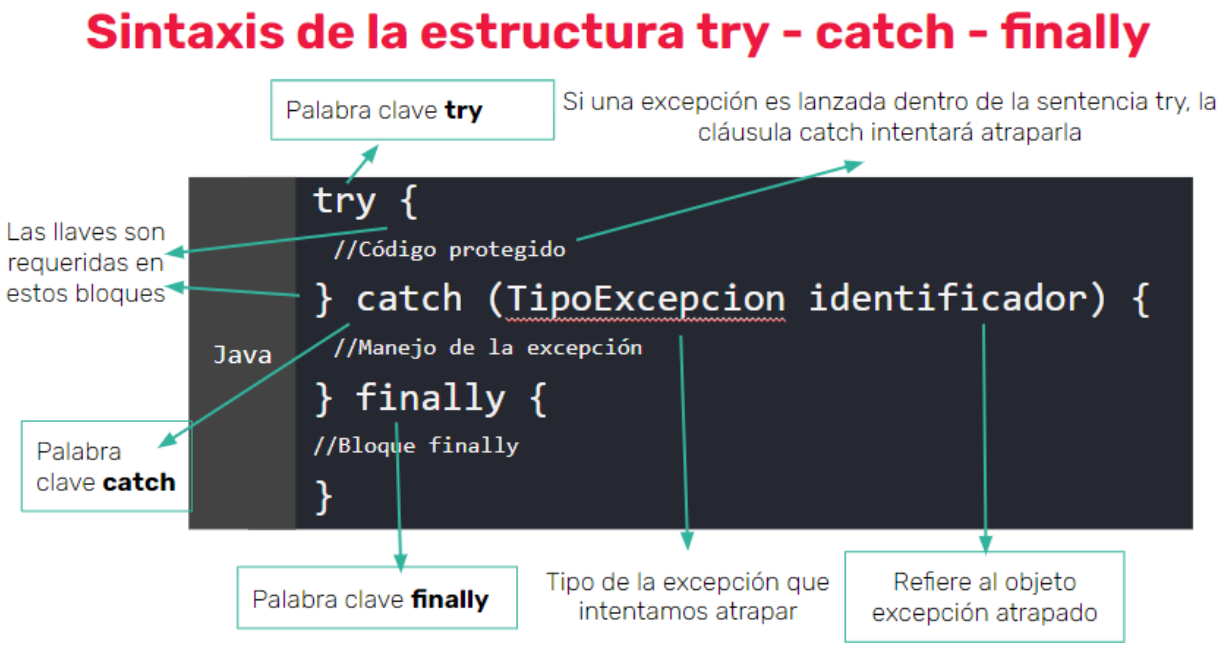
El stack trace o la **pila de llamadas**, es una lista de llamadas a métodos que se han realizado en la aplicación cuando una excepción fue lanzada. Se muestra desde el método más reciente donde se generó la excepción. Este orden nos permite identificar la causa principal del fallo de manera más fácil, ya que es más probable que encontremos la misma en los métodos más recientes ejecutados.

```
try {
   FileInputStream fileInputStream = new FileInputStream("prueba.txt");
} catch (FileNotFoundException exception) {
   exception.printStackTrace();
}
```

Manejo de excepciones

Java nos permite hacer un control de excepciones para que nuestro programa **continúe su ejecución** aunque se produzca una excepción. Para ello contamos con la estructura *try-catch-finally*.

- Bloque **try** → Significa intentar en inglés, todo el código que vaya dentro de esta sentencia será el código sobre el que se intentará capturar una excepción en caso de que ocurra.
- Bloque **catch** → Aquí definimos el conjunto de instrucciones necesarias o de tratamiento del problema capturado que ha sido lanzado en el bloque try.
- Bloque **finally** → En este bloque podremos definir un conjunto de instrucciones necesarias que se ejecutarán tanto si se produce la excepción como si no, por lo tanto se ejecuta ***SIEMPRE.***



El código en el bloque try corre normalmente, si cualquiera de sus líneas lanza una excepción, el bloque try se detiene y comienza la ejecución de las sentencias en el bloque catch.

Si ninguna de las sentencias en el bloque try lanzan una excepción que pueda ser atrapada, la cláusula catch no se ejecuta.

Hay dos caminos posibles cuando existe un bloque catch y uno finally. Si se lanza una excepción, el bloque finally se ejecuta luego del bloque catch. Si no se lanza ninguna excepción, el bloque finally se ejecuta luego que el bloque try.

Throw

La palabra reservada **throw** nos permite lanzar una excepción, debe estar seguida del operador **new** y el **tipo de excepción** que queremos lanzar. La ejecución se detiene inmediatamente después de la sentencia throw, por lo que no se ejecutará ninguna de las sentencias.

Clases Utilitarias

Las Clases Utilitarias son clases que definen un conjunto de métodos que realizan funciones que tienen que ver entre sí, normalmente muy utilizadas. La mayoría de estas clases definen métodos estáticos.

```
public class Math {


public static double cos(double a) {...}
public static double tan(double a) {...}
public static double pow(double a, double b)
{...}
public static long round(double a) {...}


}
```

```
//Haciendo uso de los métodos

double coseno = Math.cos(30);
double tangente = Math.tan(15);
double potencia = Math.pow(2, 5);
long redondeo = Math.round(20.5);
```

```
package java.time;


public final class LocalDateTime ... {
public static LocalDateTime now() {...}
public static LocalDateTime of(int year, Month
month, int dayOfMonth, int hour, int minute, int
second, int nanoOfSecond) {...}
public static LocalDateTime parse(CharSequence
text) {...}
```

```
//Haciendo uso de los métodos

LocalDateTime ldt1 = LocalDateTime.now();

LocalDateTime ldt2 = LocalDateTime.of(2021,
Month.MAY, 20, 8, 20, 15, 11);

LocalDateTime ldt3 =
LocalDateTime.parse("2021-08-03T10:15:30");
```