

Java Standard Web Programming

Módulo 4



Bloque *try-catch-finally*

Introducción

En los sistemas pueden ocurrir eventos excepcionales que corten el flujo correcto y provoquen comportamientos inesperados.

Una excepción es un error que se presenta en nuestro software. Puede ser más o menos grave.

En Java, no se pueden evitar pero sí gestionar. Así se evita la interrupción abrupta del sistema, mediante el tratamiento adecuado del problema. De esta manera, se tendrá un software más robusto y estable.



```
System.out.println("Inicio del Programa");

Integer a = 10;
Integer b = 0;

System.out.println("El resultado de la division es: " + (a / b));

System.out.println("Fin del Programa");
```

```
Inicio del Programa
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Try, catch y finally

Este bloque está conformado para capturar los errores y ejecutar las instrucciones deseadas.

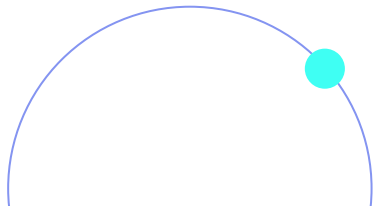
try

Contendrá las instrucciones que pueden **provocar el error**. De este tipo de bloque sólo se puede crear **uno por grupo**.

catch

Contendrá el código necesario para **gestionar el error**. Si se presenta un error en el bloque **try** el **catch** lo capturará creando un objeto según el tipo de excepción esperada, es por ello que este bloque contiene un parámetro y **se pueden crear tantos bloques catch crea conveniente**.

No es obligatorio tener un catch pero es recomendable hacerlo.



finally

Contendrá el código que **se ejecutará suceda o no un error**. No es obligatorio, y de ser requerido **solo puede existir uno** al final del try, en el caso de que no posea bloque catch. Si lo posee, **finally** debe ubicarse al final de todo el grupo.

Existen muchos tipo de excepciones y es imposible enumerarlas todas, aprenderemos poco a poco de cuales existen y cuándo utilizarlas según van apareciendo y obtenemos experiencia.



```
System.out.println("Inicio del Programa");

Integer a = 15;
Integer b = 0;
try {
    System.out.println("El resultado de la division es: " + (a / b));
} catch (ArithmeticException e) {
    System.out.println("Error: No se puede dividir por cero un numero entero");
} catch (NullPointerException e) {
    System.out.println("Error: No se puede dividir con un valor nulo");
} finally {
    System.out.println("Fin del Programa");
}
```

```
Inicio del Programa
No se puede dividir por cero numeros de tipo entero
Fin del Programa
```

Métodos

Cuando se captura un error, se crea un objeto del tipo indicado en la cláusula **catch**. Esto nos permite usar métodos de esa clase para tener un detalle más amplio del error.

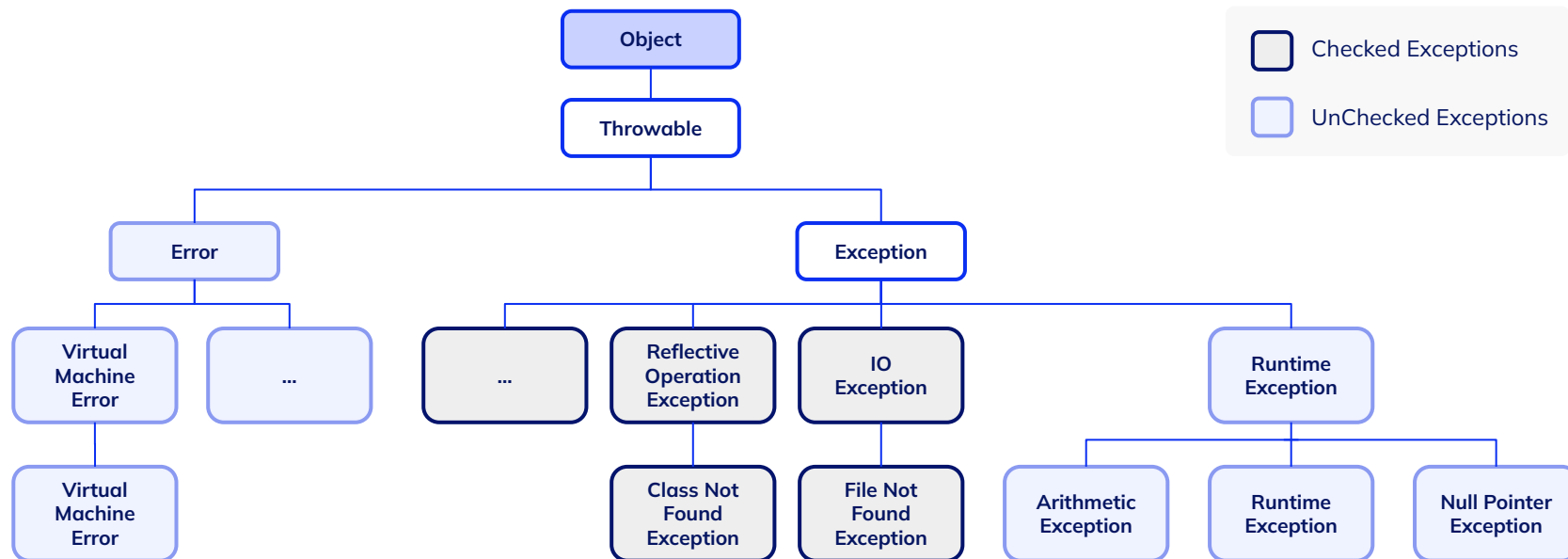
Tipo	Método	Descripción
String	toString()	Devuelve una breve descripción del objeto que capturó el error.
String	getClass()	Devuelve la clase que capturó el error.
String	getMessage()	Devuelve un mensaje con un detalle del error capturado.
void	printStackTrace()	Imprime la pila de errores que se produjo.
String	getCause()	Devuelve la causa del error o null si la causa es inexistente o desconocida.


```
System.out.println("Inicio del Programa");

Integer a = 15;
Integer b = 0;
try {
    System.out.println("El resultado de la division es: " + (a / b));
} catch (ArithmeticException e) {
    System.out.println("Error: " + e.getMessage() + ", Causa: " + e.getCause() + ", Clase: " + e.getClass());
} catch (NullPointerException e) {
    e.printStackTrace();
} finally {
    System.out.println("Fin del Programa");
}
```

```
Inicio del Programa
Error: / by zero, Causa: null, Clase: class java.lang.ArithmeticException
Fin del Programa
```

Jerarquía de las Excepciones



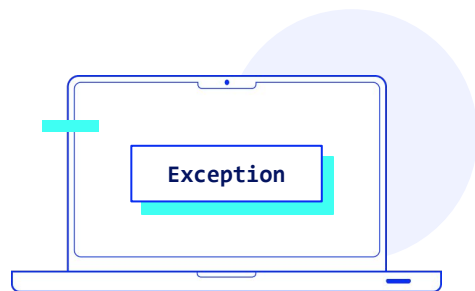
Checked Exceptions

Son las excepciones que deben ser capturadas o delegadas (punto que veremos más adelante) obligatoriamente por nosotros ya que, de lo contrario, dará un error en tiempo de compilación.

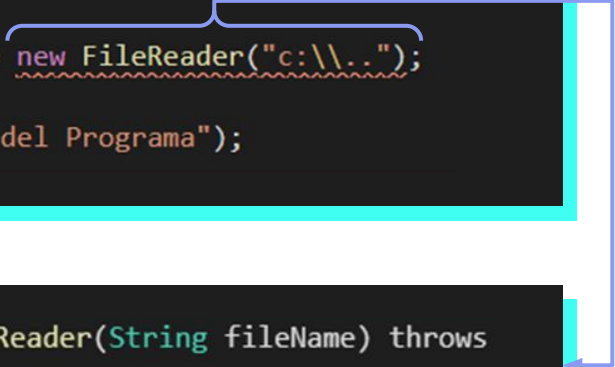
Todas estas excepciones tienen como padre la superclase **Exception**.

Es posible observar, en el ejemplo de la siguiente pantalla, que el IDE muestra un error al tratar de instanciar un objeto del tipo `FileReader` con el constructor que recibe como argumento la dirección de un archivo, en este caso el JDK indica qué tipo de problema puede ocurrir (**`FileNotFoundException`**).

Esta excepción tiene sentido, la clase **`FileReader`** es una clase que trata de leer un archivo y el JDK nos sugiere capturar un error en el caso que la dirección indicada esté errada o no exista el archivo a leer.



```
System.out.println("Inicio del Programa");  
Reader leerArchivo = new FileReader("c:\\\\..");  
System.out.println("Fin del Programa");
```



```
java.io.FileReader.FileReader(String fileName) throws  
FileNotFoundException
```

Unchecked Exceptions

Son las excepciones que tienen como superclase a la clase **RuntimeException**.

No es necesario capturarlas, pero al saltar una excepción de este tipo -como todas las excepciones-, corta el flujo de ejecución. Se suelen llamar **excepciones del programador** ya que siempre debemos estar al tanto de que ciertos errores pueden suceder.

Por ejemplo: error al capturar un dato de tipo numérico porque el usuario ingresa una cadena, o en un ciclo que recorre un arreglo porque se intenta acceder a un índice fuera del tamaño establecido.

Las excepciones de la superclase **Error** son las que el sistema no puede hacer nada con ellas. Se clasifican como errores irreversibles y, en su mayoría, provienen de la **JVM**.

Veamos el ejemplo de la siguiente pantalla.



```
String[] nombres = {"Octavio", "Macarena"};

for (int i = 0; i < 5; i++) {
    System.out.println(nombres[i]);
}
```

Octavio
Macarena

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2

**¡Sigamos
trabajando!**

