UNIDAD 9: GESTIÓN DE EXCEPCIONES

Profesor: José Ramón Simó Martínez

Contenido

1. Introducción

Hasta ahora hemos estudiado los fundamentos de la programación estructurada y la programación orientada a objetos. Sin embargo, en los diversos programas desarrollados hemos tenido que lidiar con los siguientes inconvenientes:

- **Errores de compilación:** mensajes que te indicaban que no podías compilar el programa.

  Por ejemplo:

   "Syntax error, insert ";" to complete BlockStatements"

- **Advertencias (*Warnings*):** puedes compilar el programa, pero ten en cuenta que algo puede ir mal.

  Por ejemplo:

   "the value of the local variable X is never read"

- **Excepciones:** cuando el programa terminaba de forma abrupta.

  Por ejemplo:

  - "Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException".

Los errores de compilación y los conocidos como *Warnings* los hemos resuelto rectificando las partes del código requeridas, normalmente siguiendo los consejos del entorno de desarrollo correspondiente. No obstante, a veces nuestro programa terminaba de forma abrupta con algún mensaje relacionado con la palabra "Exception"; la mayoría de las veces no sabíamos a que se referían dichos mensajes, pero resolvíamos el inconveniente de alguna forma u otra.

En esta unidad nos centraremos en la gestión de las excepciones en el lenguaje de programación Java.

Al terminar esta unidad deberás ser capaz de:

- Conocer la jerarquía y los tipos de excepciones.
- Identificar las posibles excepciones en el código.
- Saber lanzar excepciones y propagarlas desde métodos.
- Saber tratar y capturar excepciones con los bloques *try-catch-finally.*
- Saber crear excepciones propias.
- Desarrollar programas más robustos aplicando la gestión de excepciones en Java

2. Concepto de excepción

En programación se conoce como *excepción* al error que se produce en tiempo de ejecución del programa. Como indica el propio termino, es un problema que ocurre con poca frecuencia; se debe a un dato o instrucción que está fuera de contexto del funcionamiento normal del programa.

El objetivo principal de la gestión de excepciones es crear programas robustos que permitan controlar estas excepciones y seguir con la ejecución del programa sin verse afectados por el problema.

A continuación, veamos unos ejemplos de programas que producen excepciones durante su ejecución.

Ejemplo 1: Dividiendo entre cero

Código:

```
1: /* Ejemplo 1: Dividiendo entre cero */
2: public class Ejemplo1 {
3:   public static void main(String[] args) {
4
5:         int resultado = 5/0; // Error en ejecución
6:         System.out.println(resultado);
7:   }
8:}
```

Salida:

```
<terminated> Test (7) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (11 mar 2023 16:54:26 – 16:54:26) [pid: 13340]
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Test.main(Test.java:5)
```

Explicación del mensaje de error:

- La máquina virtual de Java (JVM) ha detectado la división por 0 (error) y ha creado un objeto de la clase *java.lang.ArithmeticException*. El método *main* no es capaz de tratar dicha excepción y por tanto la JVM finaliza el programa en la línea 5 y muestra un mensaje de error con la información sobre el tipo de excepción que se ha producido.

Ejemplo 2: Conversión de cadena a entero

Código:

```
1: /* Ejemplo 2: conversión de cadena a entero */
2: public class Ejemplo2 {
3:   public static void main(String[] args) {
4:         int num = Integer.parseInt("23b");
5:         System.out.println(num);
6:}}
```

Salida:

```
<terminated> Test (7) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (11 mar 2023 17:15:47 – 17:15:48) [pid: 452]
Exception in thread "main" java.lang.NumberFormatException: For input string: "23b"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:668)
        at java.base/java.lang.Integer.parseInt(Integer.java:786)
        at Test.main(Test.java:4)
```

Explicación del mensaje de error:

- El método *Integer.parseInt(…)* no puede convertir una cadena a entero ya que el formato no es el adecuado, "23b". Por tanto se lanza una excepción de tipo *java.lang.NumberFormatException*. La JVM termina el programa en la línea 4 y muestra por pantalla la información sobre la excepción que se ha producido. En este caso fíjate que se indican también el lugar y la línea de cada método donde se ha producido el error, por ejemplo *"java.lang.Integer.parseInt(Integer.java:668)"* hasta llegar a la clase que maneja la Excepción.

Ejemplo 3: Rango de índices del array

Código:

```
1: /* Ejemplo 3: Rango de índices del array */
2: public class Ejemplo2 {
3:   public static void main(String[] args) {
4:         int miArray = {1,2};
5:         int valor = miArray[4];
6:   }
7:}
```

Salida:

```
<terminated> Test (7) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (11 mar 2023 17:24:49 – 17:24:51) [pid: 14828]
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 2
        at Test.main(Test.java:5)
```

Explicación del mensaje de error:

- En este programa intentamos acceder al elemento que se encuentra en la cuarta posición de un array de 2 elementos. La JVM finaliza el programa en la línea 5 y muestra un mensaje de error sobre la excepción que se ha producido.

Ejemplo 4: Entrada de datos incompatible

Código:

```
1: /* Ejemplo 4: Entrada de datos incompatible */
2: public class Ejemplo2 {
3:   public static void main(String[] args) {
4:         Scanner sc = new Scanner(System.in);
5:         System.out.println("Introduce un número entero: ");
6:         sc.nextInt(); // Si introduces un carácter, lanza excepción.
7:   }
8:}
```

Salida:

```
<terminated> Test (2) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (13 mar 2023 10:46:28 – 10:46:31) [pid: 4540]
Introduce un número entero:
B
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at Test.main(Test.java:6)
```

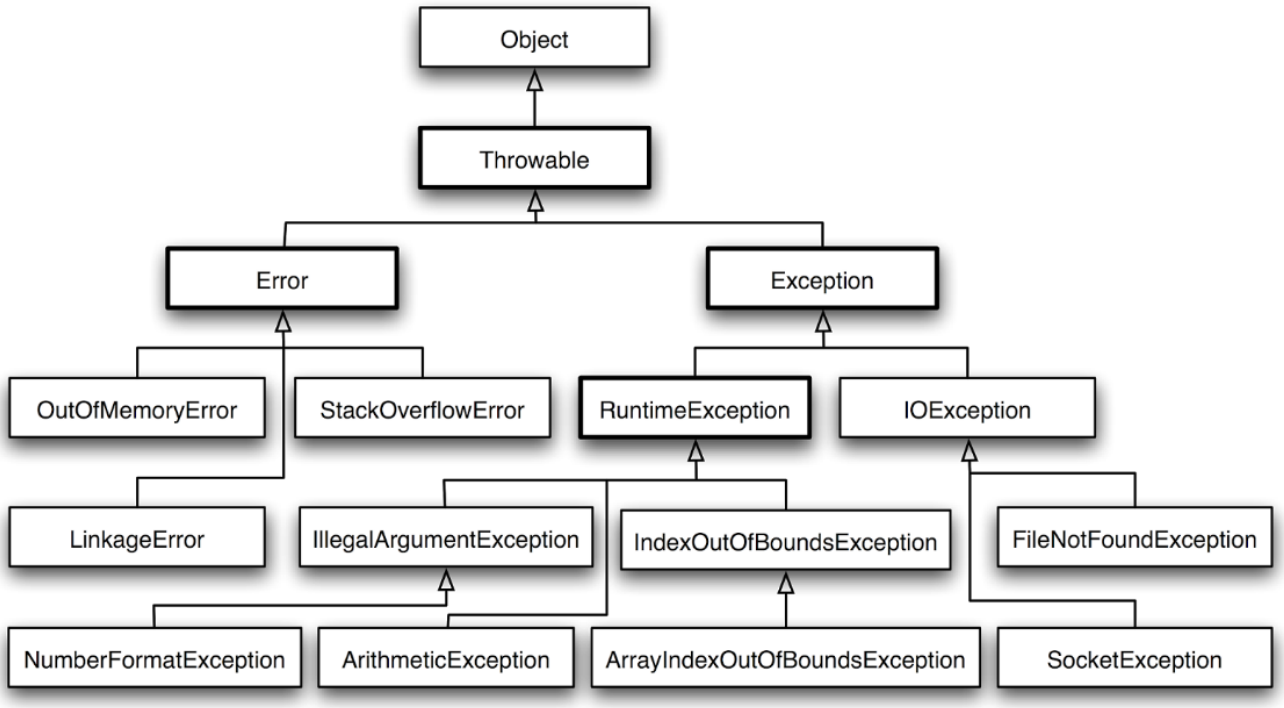
Explicación del mensaje de error:

- En la entrada de datos se ha detectado un dato no compatible con el esperado por el método nextInt() de la clase Scanner (un número entero); en cambio, se introduce un caracter. La JVM finaliza el programa en la línea 6 y muestra un mensaje de error sobre la excepción que se ha producido, así como la traza de llamadas a métodos implicados en la excepción.

3. Jerarquía de excepciones en Java

Java, como lenguaje de programación orientada a objetos, permite gestionar las excepciones a través de una serie de objetos de clases que heredan, en última instancia, de la clase *java.lang.Throwable*. Dicha clase representa cualquier fallo de ejecución independientemente de su tipo.

A continuación, veamos el esquema completo de la jerarquía:



Visto que una excepción es un fallo de ejecución (recuperable) y que también un error es un fallo de ejecución (irrecuperable), las clases *Error* y *Exception* se diseñan como derivadas directas de *Throwable*. Por otra parte, tanto de la clase *Error* como *Exception* derivan otras subclases que especifican en mayor medida el tipo de error o excepción que se puede dar.

> **Nota**
>
> En Java, la clase *Error* hace referencia a problemas serios fuera del control de la aplicación, como por ejemplo memoria insuficiente para alojar un objeto (*OutOfMemoryError*)

Las excepciones se pueden clasificar en:

- **Checked:** representan errores de los que puede recuperar el programa. Todas estas excepciones deben ser capturadas y manejadas en tiempo de compilación. Las clases *Throwable*, *Exception* y sus derivadas son de este tipo.
- **Unchecked:** representan errores de programación. Estas excepciones no deben ser forzosamente declaradas ni capturadas, aunque el programa puede terminar erróneamente. Las clases *Error* y *RuntimeException* son de este tipo.

3.1. Tabla de excepciones más frecuentes

A continuación, veamos la tabla de algunas excepciones de Java (en negrita las más comunes):

Excepción	Descripción	Tipo
IOException	Fallo en operación de entrada/salida.	Checked
ParseException	Parseo de datos incompatible.	Checked
InterruptedException	Un hilo (thread) interrumpe a otro.	Checked
ClassNotFoundException	No se encuentra la clase especificada.	Checked
InputMismatchException	Lanzada por Scanner indicando entrada no compatible.	Unchecked
NullPointerException	Intento de usar un objeto a *null*.	UnChecked
ArrayIndexOutOfBoundsException	Acceso a un índice del array fuera de rango.	UnChecked
StringIndexOutOfBoundsException	Acceso a un índice de la cadena fuera de rango.	UnChecked
NumberFormatException	Al convertir una cadena a número.	UnChecked
ArithmeticException	Al evaluar una operación aritmética.	UnChecked
ClasCastException	Al hacer casting hacia una subclase incompatible.	UnChecked
IllegalArgumentException	Parámetros del método incorrectos.	UnChecked

4. Lanzar excepciones: *throw vs throws*

Una buena práctica en la programación de aplicaciones es lanzar excepciones cuando intentamos realizar acciones incorrectas o inesperadas. Recordemos que en la programación orientada a objetos es la clase la responsable de la validez de sus datos y de las acciones que se pueden o no realizar.

Por ejemplo, en una clase Coche:

- Instanciar un objeto con un identificador de matrícula de coche incorrecta.
- Valor negativo en el número de kilómetros.
- Número de ocupantes mayor a la capacidad del coche.

En este sentido, es apropiado lanzar las excepciones en los *setters* o en cualquier otro método de la clase que reciba datos para realizar acciones no permitidas o que violen la integridad del objeto; como por ejemplo cambiar el valor del kilometraje directamente.

> **Nota**
>
> Lanzar una excepción no implica necesariamente que el programa termine; es una simple forma de avisar de un error.

Para lanzar una excepción utilizaremos la palabra reservada ***throw*** seguido de la instanciación un objeto de tipo *Exception* o alguna de sus subclases.

Por ejemplo, para lanzar una excepción genérica tal y como conocemos los objetos en Java sería:

```
Exception e = new Exception();
throw e;
```

Sin embargo, la forma más adecuada de hacerlo es la siguiente:

```
throw new Exception();
```

Además, el constructor de la clase *Exception* puede recibir una cadena de texto (*String*) para indicar cuál es el problema. Si la excepción no se gestiona y el programa se para, el mensaje de error se mostrará por la consola.

```
throw new Exception("El kilometraje no puede ser negativo");
```

Por otra parte, también podemos lanzar excepciones específicas de Java como las que se indican en el anterior apartado de jerarquía de excepciones:

```
throw new NumberFormatException("mensaje…");
```

4.1. Lanzar excepciones desde métodos: *throws*

Para crear un método que lance excepciones debemos añadir a la cabecera del método la palabra reservada ***throws***:

```
public static void setKilometros(int km) throws Exception {
      if(km < 0)
            new throw Exception("Kilómetros negativos no permitidos.");
      else
            this.km = km;
}
```

Hay que tener en cuenta que al lanzar una excepción se parará la ejecución de dicho método (no se ejecutará el resto del código del método) y se lanzará la excepción al método que lo llamó. Si por ejemplo, desde la función *main* llamamos a *setKilometros()*, y *setKilometros()* es candidato a que lance un excepción, entonces en la práctica es posible que el *main* lance una excepción (no directamente con un *throw*, sino por la excepción que nos lanza *setEdad()*). Por lo tanto, también tenemos que especificar en el *main* que se puede lanzar una excepción:

```
public static void main(String[] args) throws Exception {
      Coche c = new Coche("Toyota");
      c.setKilometros(10000);
      // … más código
}
```

Nota

El anterior código es un ejemplo para explicar el concepto de cómo funciona el lanzamiento de excepciones desde un método. No es nada común indicar que el método *main* lanza una excepción añadiendo *throws* en su cabecera. Lo más adecuado es capturar las excepciones en el método *main* o antes. En siguientes apartados estudiaremos la captura de excepciones.

Puede que un método necesite lanzar más de una excepción. Para ello, indicaremos en la cabecera del método tantas excepciones como sean necesarias separadas por comas:

```
public Coche(String matricula, int kilometros) throws ArithmeticException,
MatriculaException {
      if(km < 0)
            new throw Exception("Kilómetros negativos no permitidos.");
      if(!esMatriculaValida(matricula))
            new throw MatriculaException("Matricula incorrecta");
      this.km = km;
      this.matricula = matricula;
}
```

Nota

MatriculaException sería una excepción de usuario que estudiaremos en próximos apartados.

4.2. Propagación de excepciones

Recordemos la salida del ejemplo 2:

```
<terminated> Test (7) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe  (11 mar 2023 17:15:47 – 17:15:48) [pid: 452]
Exception in thread "main" java.lang.NumberFormatException: For input string: "23b"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:668)
        at java.base/java.lang.Integer.parseInt(Integer.java:786)
        at Test.main(Test.java:4)
```

Podemos observar la pila de llamadas a métodos desde donde se produce la excepción. De manera genérica, imaginemos que en un método A llamamos a un método B, que llama a un método C, etc, hasta llegar a E.

A → B → C → D → E (secuencia de llamadas de métodos)

Si el método E lanza una excepción, esta le llegará a D que a su vez se la lanzará a C, etc, recorriendo el camino hasta llegar al método inicial A.

A ← B ← C ← D ← E (secuencia de lanzamiento de excepciones)

Por lo tanto, como todos estos métodos pueden acabar lanzando una excepción, en sus cabeceras habrá que incluir el *throws Exception* (o el que corresponda según el tipo de excepción).

> **Nota**
>
> Quiero volver a recordar que no es recomendable dejar que las excepciones del programa lleguen de forma descontrolada hasta el *main* y termine el programa. La idea es manejar las excepciones tal y como veremos en el siguiente apartado.

5. Captura y manejo de excepciones: try-catch-finally

Para la gestión de excepciones en el programa se utilizan mecanismos que funcionan en tres bloques de código:

- **try{}**: bloque donde están las instrucciones que pueden provocar alguna excepción.
- **catch{}**: bloque donde se capturará la excepción lanzada en el bloque *try {}* que tiene asociado.
- **finally{}**: bloque opcional. Se ejecutará tanto si se lanza o no una excepción. Se utiliza normalmente para tareas de limpieza (cerrar ficheros, entrada/salida, etc).

Estructura del bloque *try-catch-finally*:

```
try {
     // Instrucciones que pueden provocar alguna excepción
}
catch (NombreDeExcepcion1 e) {
     // Instrucción a realizar si se produce la excepción NombreDeExcepcion1
}
catch (NombreDeExcepcion2 e) {
     // Instrucción a realizar si se produce la excepción NombreDeExcepcion2
}
// Aquí pueden haber más bloques catch
catch (NombreDeExcepcionX e) {
     // Instrucción a realizar si se produce la excepción NombreDeExcepcionX
}
finally {
     // Bloque opcional, se ejecuta tanto si se produce como si no una excepción
}
```

Pueden haber de uno a varios bloques *catch* para captura cada una de las excepciones que se produzcan en el bloque try.

Si reescribimos el código del *ejemplo 1 (división entre cero)*, visto en el apartado 2, para tratar la excepción de forma general quedaría así:

```
/* Ejemplo 1: Dividiendo entre cero con try-catch y excepción genérica*/
public class Ejemplo1 {
     public static void main(String[] args) {
          try {
               int resultado = 5/0; // Lanza excepción
               System.out.println(resultado);
          }
          catch(Exception e) {
               System.out.println("Se ha capturado excepción.");
          }
     }
}
```

Sin embargo, el tratamiento de excepciones debe hacerse de la más concreta a la más genérica dentro del bloque catch. Si observamos la jerarquía de excepciones, encontramos la clase *java.lang.ArithmeticException,* la cual es la más adecuada para el ejemplo anterior:

```
/* Ejemplo 1: Dividiendo entre cero con try-catch y excepción concreta*/
public class Ejemplo1 {
	public static void main(String[] args) {
		try {
			int resultado = 5/0; // Lanza excepción
		}
		catch(ArithmeticException e) {
			System.out.println("Se ha capturado excepción aritmética.");
		}
	}
}
```

También podemos capturar varias excepciones; en este caso, la primera que se produzca será la primera que se capture. Si añadimos el caso del *ejemplo 3 (rango de índices del array),* visto en el apartado 2, tendremos el siguiente código para tratar varias excepciones:

```
public class Ejemplo1 {
	public static void main(String[] args) {
		int miArray = {1,2};
		try {
			int resultado = 5/0; // Lanza excepción
			int valor = miArray[4];

		}
		catch(ArithmeticException e) {
			System.out.println("Se ha capturado excepción aritmética.");
		}
		catch(IndexOutOfBoundsException e) {
			System.out.println("Se ha capturado excepción de índice fuera de
			rango");
		}
	}
}
```

5.1. Mensajes de la excepción

Una excepción es un objeto que, como toda clase, contiene ciertos miembros a los que podemos acceder para obtener información. Algunos de los más destacados son:

- **String toString():** devuelve "TipoExcepción: mensaje".
- **Class<?> getClass():** devuelve la clase a la que pertenece la excepción.
- **String getMessage():** devuelve el mensaje con el que se crea la excepción.

- **printStackTrace():** Imprime por la salida de error el objeto desde el que se invoca con una traza de las llamadas a los miembros desde los que se ha producido la excepción. Es muy útil para depurar programas.

Un ejemplo de uso de alguno de estos métodos:

```
public class Ejemplo1 {
     public static void main(String[] args) {
           int miArray = {1,2};
           try {
                 int resultado = 5/0; // Lanza excepción
                 int valor = miArray[4];

           }
           catch(ArithmeticException e) {
                 System.out.println("Se ha capturado excepción aritmética.");
           }
           catch(IndexOutOfBoundsException e) {
                 System.out.println("Se ha capturado excepción de índice fuera de
                 rango");
                 System.out.println(e.getMessage());
                 System.out.println(e.getStackTrace());
           }
     }
}
```

6. Creación de excepciones de usuario

Existe otro tipo de excepciones que no pueden ser advertidas por el compilador. Para que podamos tratarlas será necesario añadir excepciones propias, es decir, clases que extiendan de la clase *Exception*: a estas excepciones se les conoce como excepciones de usuario.

Para crear excepciones de usuario podemos seguir los siguientes pasos:

1. Definir una clase propia que herede de la clase *Exception*.
2. Crear un constructor con un *String* como argumento.
3. Dentro del constructor llamar al constructor *super()*, pasándole el *String* recibido.

A continuación, vamos a detallar un ejemplo de creación de una clase que llamaremos *ExcepcionIntervalo* para tratar excepciones que se produzcan cuando haya un entero que esté fuera de un rango determinado de valores:

```
public class ExcepcionIntervalo extends Exception {

	// Constructor
	public ExcepcionIntervalo(String msg) {
		super(msg);
	}
}
```

Y con esto ya tendríamos nuestra excepción de usuario lista para utilizarse.

Para poder utilizar la excepción de usuario, debemos utilizarla o lanzarla. Así que en nuestro código debemos detectar cuándo se produce la situación anómala y lanzarla. El siguiente esquema de código refleja esta idea:

```
public void algunMetodo() throws nuevaExcepcion {

	// con throws propagamos la excepción hacia donde se ha llamado el método

	// código del método

	if (situaciónAnómala) // La comprobación estará dentro de un método
		throw new nuevaExcepcion("Descripcion del Error");
		// con throw se lanza la Excepción.

	// más código
}
```

Concretamente, el código para lanzar la excepción *ExcepcionIntervalo* que hemos creado sería el siguiente

```
public static void rango(int num) throws ExcepcionIntervalo {
	if (num < 0 || num > 100) {
		throw new ExcepcionIntervalo("Número fuera del intérvalo");
	}
}
```

Debemos tener en cuenta lo siguiente:

- El método que puede lanzar (*throw*) la excepción debe dejarla salir (*throws*).
- El lanzamiento de las excepciones siempre estará dentro de las sentencias condicionales.
- La descripción (mensaje) debe ser breve y clarificadora.

Por otra parte, si intentamos llamar a este método *rango(…)* sin más, el IDE dará error, diciéndonos que la excepción no está tratada (*unhandled*); esto se debe a que el método la puede lanzar, pero desde donde llamamos al método no estamos tratando la excepción.

Por ejemplo:

```
public class Test {
	public static void main(String args[]) {
		rango(200); // Dará error "unhandled exception type"
	}

	// Método rango…
}
```

En la llamada a *rango(10)* dará error y por tanto debemos tratar la excepción:

```
public class Test {
	public static void main(String args[]) {
		try {
			rango(200);
		}
		catch(ExcepcionIntervalo e) {
			System.out.println(e.getMessage());
		}
	}

	// Método rango…
}
```

Salida por pantalla:

```
Número fuera del intérvalo
```

7. Recomendaciones

A continuación, una serie de recomendaciones sobre la buena práctica en la gestión de excepciones en Java:

- No escatimar en el uso de la propagación de excepciones.
  Moraleja: lanza pronto, captura tarde.
- Usar la excepción más adecuada en cada situación, evitar las excepciones genéricas.
- Las excepciones tienen un coste muy alto respecto a las comprobaciones sencillas (con condicionales).
  Moraleja: usar excepciones para circunstancias excepcionales.
- Evitar bloques *catch* vacíos; siempre incluir al menos un mensaje informando del error o imprimir un rastro completo de este.
- Liberar recursos en el bloque ***finally***.
- Si ya existe una excepción no es necesario personalizarla.

8. Bibliografía

Libro “Core Java Volumen I: Fundamentals”, Cary S. Horstmann

Apuntes de Programación del CEEDCV (Licencia Creative Common)

Apuntes de José Chamorro del CFGS DAW del IES Sant Vicent Ferrer (Algemesí)

https://docs.oracle.com/