Capítulo 6: Excepciones

1. Qué son las excepciones?

Una excepción es un error excepcional, infrecuente, raro, que ocurre en un porcentaje pequeño de veces en la ejecución de un programa bajo condiciones normales esperadas. Es importante distinguir aquí lo ambiguo de esta definición: ¿A partir de qué porcentaje de frecuencia se le considera a un error infrecuente?, ¿qué condiciones se consideran normales?. A los largo de este capítulo se darán mostrarán ejemplos comunes de aplicación de excepciones así como algunos criterios de decisión que pueden ayudar a decidir cuándo tratar a un error como una excepción y cuando no.

Algunos ejemplos típicos de excepciones son:

- Acceso a un elemento de un arreglo fuera de los límites de este.
- Una división por cero.
- Una llamada a un método con parámetros inválidos.
- Una falla en la reserva de memoria (la sentencia *new* falla).

Los errores tratados como excepciones suelen ser también aquellos para los que, dentro del ámbito del programa donde ocurre dicho error, no es posible darle una solución satisfactoria. Note que aquí también hay ambigüedad: ¿Qué es una solución satisfactoria?
Si dicho ámbito abarca a todo el programa, este por consiguiente no puede continuar ejecutándose de la manera esperada, por lo que debería finalizar. Si en un ámbito que engloba al ámbito de la excepción, se cuenta con los elementos necesarios para darle una solución satisfactoria, el manejo de dicha excepción en el ámbito englobante podría permitir *estabilizar* el programa, de forma que vuelva a un estado de ejecución *normal*. Si ningún ámbito englobante, en todo el programa, puede solucionar la excepción, el programa podría notificar al usuario de lo ocurrido y finalizar ordenadamente, liberando los recursos que fueron reservados. En resumen, el tratamiento de excepciones en un programa permite:

- Que el programa se recupere de la excepción y siga ejecutándose normalmente.
- Que el programa notifique la excepción y finalice de manera controlada.

Ahora bien, dado que las excepciones son básicamente errores infrecuentes, podrían tratarse con las mismas técnicas tradicionales que se utilizan para los errores frecuentes. El siguiente pseudo-código muestra una técnica típica de tratamiento de errores con estructuras de control de flujo.

```
Ejecutar una acción
Si ocurrió un error
        Reportar el error y finalizar
Ejecutar una acción
Si ocurrió un error
        Reportar el error y finalizar
```

...

Bajo esta técnica se tiene las siguientes ventajas:

- El tratamiento de un error se realiza en la vecindad donde ocurre.
- Es fácil reconocer el código que maneja cada error en el programa, y por tanto entenderlo. Como contraparte, es fácil reconocer la fuente de un error.
- Es fácil realizar un seguimiento a la ejecución del programa.

Las desventajas son:

- Es fácil que el código del programa termine minado con código de manejo de error, lo que hace difícil distinguir la tarea limpia del programa, esto es, la tarea que se intenta realizar independientemente de los errores que puedan ocurrir o asumiendo que no ocurren. Un código así es difícil de mantener.
- Si un error es infrecuente pero potencialmente puede ocurrir en distintas partes del programa, el programa final contendrá mucho código repetitivo de manejo de dichos errores.
- Si un error es infrecuente, el programador podría pasarlo por alto inadvertidamente o bien tendrá la tendencia a dejar su tratamiento para después, siendo dicho error olvidado o bien tratado no sin un esfuerzo mucho mayor del que hubiese sido necesario en un inicio. Este tipo de errores son un motivo común de la falta de robustez (tolerancia a fallos) de los programas.
- Si un error es infrecuente pero potencialmente puede ocurrir en distintas partes del programa, todas las verificaciones correspondientes a dicho error se realizarán, aun cuando este no ocurra, lo que le resta eficiencia (mayor tiempo de CPU y recursos) al programa.

Como puede apreciarse, el tratamiento tradicional de errores es adecuado cuando estos son frecuentes, pero no cuando son infrecuentes, como las excepciones.

Las excepciones pueden ser generadas por errores detectados por el hardware (como una división entera entre cero) y capturados por el sistema operativo, producidos por el propio sistema operativo debido a un error de lógica interno (detección de un nivel de memoria disponible debajo de un límite de seguridad), producidos por alguna librería utilizada por nuestro programa por nuestro mismo programa.

Los lenguajes de programación que distinguen el concepto de excepción, como C++, Java y C#, utilizan una técnica muy diferente al tratamiento tradicional de errores. Esta técnica se basa en la idea de separar el código principal del programa (el código propio de la tarea que el programa desea realizar, un código limpio de verificación de excepciones), del código de manejo de las excepciones. Como puede entenderse, este código limpio seguirá conteniendo el código de manejo de los errores frecuentes, bajo las técnicas tradicionales.

En las siguientes secciones veremos cómo se implementa el tratamiento de excepciones en Java y C#, así como las ventajas y desventajas de su uso.

2. Implementación

Tanto Java como C# se basan en el modelo de C++ para el tratamiento de excepciones, por lo que revisaremos primero esta para después tratar las diferencias que existen entre los tres lenguajes.

2.1. C++

Para manejar excepciones que pueden ocurrir dentro de un bloque de código, es necesario "delimitar" dicho bloque. Para esto se utiliza la palabra reservada *try*. La sintaxis a utilizar es la siguiente:

```
try {
        // Aquí va el código del que se desea controlar las excepciones
        // que produzca.
}
```

Dentro del bloque *try* se colocará el código del que se espera monitorear las excepciones que produzca. Cuando se produce una excepción, se ejecutará un determinado bloque de código llamado *manejador de excepción*. Estos bloques de código deben de ir inmediatamente después del bloque *try* e igualmente deben ser "delimitados" para cada tipo en particular de excepción. Para esto se utiliza la palabra reservada *catch*. La sintaxis a utilizar será la siguiente:

```
catch( TipoDeExcepcion e ) {
        // Aquí va el código que se ejecutara en caso que se produzca
        // una excepción del tipo "TipoDeExcepcion".
}
```

Un bloque *try* puede ir seguido por tantos bloques *catch* como tipos de excepciones se desee manejar. El tipo de variable *TipoDeExcepcion* determina el tipo de excepción que manejara un bloque *catch*.

Dentro del bloque *try*, una excepción puede ser producida por:

- Una sentencia *trow* que explícitamente dispare la excepción.
- Una llamada a una función que dispare la excepción. Dicha función podría disparar la excepción explícitamente o llamar a otra función (y esta a otra y así sucesivamente) la cual sea quien realmente dispare la excepción.
- La creación de un objeto, debido a un error dentro del constructor utilizado.
- Un error del sistema operativo durante la ejecución de cualquier sentencia dentro del bloque *try*.
- Una interrupción capturada por el sistema operativo y notificada al programa en ejecución a manera de una excepción.

2.2. Java

Al igual que C++, Java comparte el mismo modelo de manejo de excepciones que C++, pero a diferencia de este, Java cuenta con un soporte más completo.

La siguiente es una lista de diferencias:

a) Todos los tipos de excepciones heredan de una clase base.

El siguiente es el árbol de herencia de las clases para excepciones.

```
java.lang.Object
        java.lang.Throwable
                java.lang.Error
                java.lang.Exception
                        java.lang.RuntimeException
```

La clase *Throwable* es la clase base de todos los errores y excepciones de Java manejados por el mecanismo de manejo de excepciones. El intérprete de Java y el programador solo pueden disparar excepciones de un tipo que derive de *Throwable*.

La clase *Error* es la clase base de excepciones generadas por el intérprete de Java, por lo que no deben ser manejadas por el programador, dado que son errores para los que solo el intérprete puede dar una solución satisfactoria.

La clase *Exception* es la clase base de todas excepciones con las que el programador si debe lidiar, a excepción de las que derivan de *RuntimeException*, las que tienen un tratamiento especial.

Si un programa desea crear sus propias excepciones, heredaría de la clase Exception.

b) Las excepciones deben ser tratadas por el programa.

Con *lidiar* nos referimos a que el programador *debe capturar explícitamente* las excepciones de dichos tipos, o bien *indicar explícitamente* que no desea manejarlas. Las excepciones que derivan de *RuntimeException* corresponden a errores de lógica en la programación, por lo que el lenguaje acepta que no sean explícitamente tratadas, dado que solo son usadas para depurar los programas y eliminar dichos errores. Una vez eliminados estos errores, esas excepciones ya no se presentarán, y el código escrito para manejarlas dejaría de ser útil. A continuación, un ejemplo ilustra este aspecto.

```
public class Ejemplo1 {
	public static void main(String args[]) {
		// sentencias fuera del bloque de control de excepciones
		int Arreglo[] = { 1, 2, 3, 4 };

		// definición del bloque try
		try {
			for( int i = 0; i < 5; i++ ) {
				int iValor = Arreglo[ i ];
				System.out.println("Valor " + i + " = " + iValor );
			}
		}
		// definición del bloque catch
		catch( ArrayIndexOutOfBoundsException e ) {
			System.out.println("Ocurrió la siguiente excepción = " + e );
		}
		// definición del bloque finally
		finally {
			System.out.println("Ejecución del bloque finally" );
		}

		// sentencias fuera del bloque de control de excepciones
		System.out.println("Fin del programa." );
	}
}
```

La excepción *ArrayIndexOutOfBounds* hereda de *RuntimeException* y es generada cuando se accede a un elemento de un arreglo utilizando un índice inválido. Esto es justamente lo que sucede en el código anterior, donde se intenta acceder al elemento 5to. del arreglo (indice *i = 4*) produciendose una excepción, dado que el arreglo solo tiene cuatro elementos. Este error se puede subsanar reemplazando la declaración del bucle, por ejemplo, de la siguiente manera:

```
			for( int i = 0; i < Arreglo.length; i++ ) {
```

Por tanto, una vez hecho esto dicha excepción nunca volverá a ocurrir, por lo que todo el código escrito para el manejo de esta pasa a ser inútil. Todo el resto de excepciones que heredan de *Exception* y no de *RuntimeException* *deben ser manejadas explícitamente* por el programador. El siguiente programa muestra este caso.

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

class Ejemplo3 {

	public static void main(String args[]) {
		BufferedReader input
		   = new BufferedReader(new InputStreamReader(System.in));

		try {
			System.out.print("Ingrese un primer entero : ");
			String cadena1 = input.readLine();
			System.out.print("Ingrese un segundo entero : ");
			String cadena2 = input.readLine();

			int entero1 = Integer.parseInt(cadena1);
			int entero2 = Integer.parseInt(cadena2);
			System.out.println("La suma da : " + (entero1 + entero2));
		}
		catch(IOException ex) {
			System.out.println(
				"Ocurrio la sgte. excepcion durante la lectura " + ex);
		}
```

```
        }
}
```

El código anterior utiliza el método *readLine* de la clase *BufferedReader* para leer el texto ingresado por teclado desde la ventana de comandos del programa Dicho método puede producir una excepción del tipo *IOExcepcion*, la cual no deriva de *RuntimeException*, por lo que tenemos la obligación de capturarla. Otra opción es indicar explícitamente que no deseamos manejar dicha excepción mediante la cláusula *throws*. Esta cláusula se coloca luego de la declaración de parámetros de un método. El siguiente código modifica el anterior para utilizar una cláusula *throws*.

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

class Ejemplo3 {

        public static void main(String args[]) throws IOException {
                BufferedReader input
                   = new BufferedReader(new InputStreamReader(System.in));

                System.out.print("Ingrese un primer entero : ");
                String cadena1 = input.readLine();
                System.out.print("Ingrese un segundo entero : ");
                String cadena2 = input.readLine();

                int entero1 = Integer.parseInt(cadena1);
                int entero2 = Integer.parseInt(cadena2);
                System.out.println("La suma da : " + (entero1 + entero2));
        }
}
```

La cláusula *throws* puede ir precedida de una lista de tipos de excepciones, separadas por comas. Esta cláusula le dice al compilador de Java *"se que el código en este método produce estas excepciones, pero no deseo manejarlas aquí,* ***sino en el método que llame a este****"*.

Si el código en nuestro programa produce excepciones que deben ser manejadas explícitamente, ya sea mediante una secuencia *try-catch* o con una cláusula *throws*, no lo son, se generarán errores durante la compilación indicando que esto debe hacerse.

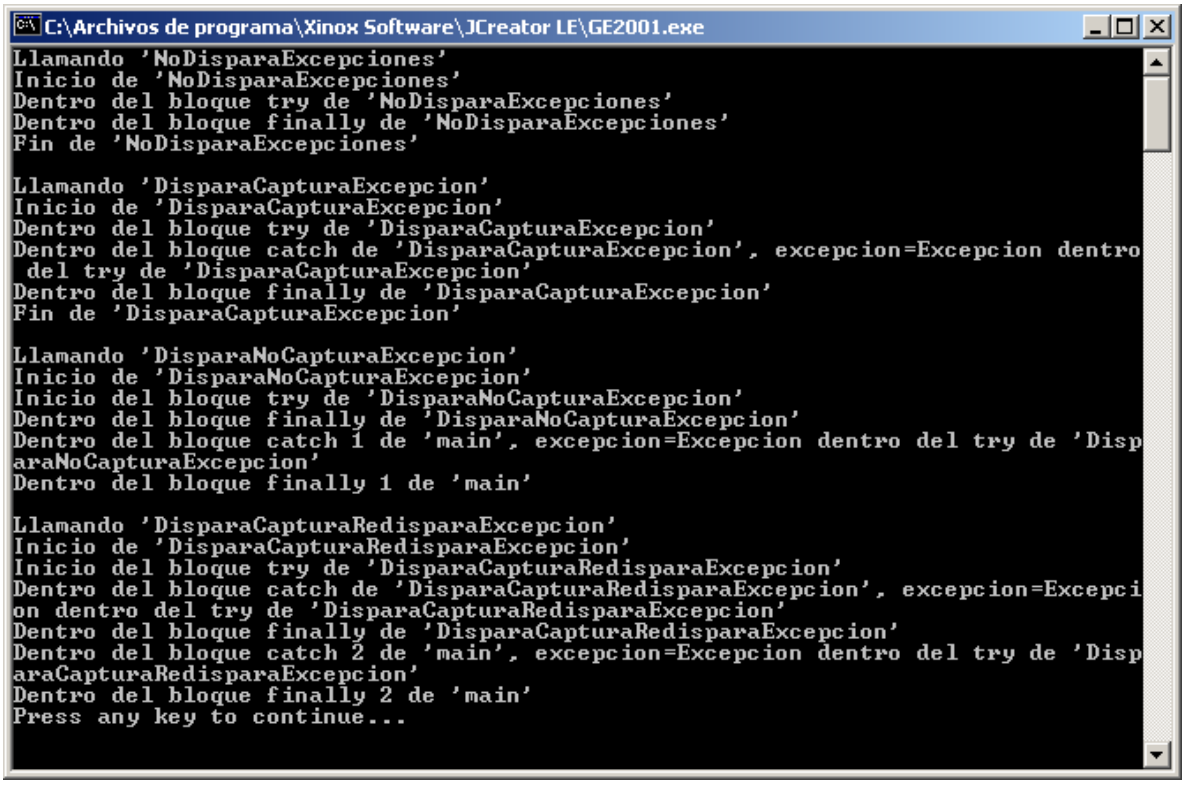
c) Se define un bloque especial que *siempre se ejecuta*.

Existe un bloque llamado *finally*, que puede ir a continuación de un bloque *try* o el último bloque *catch*, y del que se tiene la garantía de que, una vez ingresada la ejecución al código dentro del bloque *try*, siempre se ejecutará, ocurra o no una excepción dentro de *try* y en caso de que sí ocurra, sea o no capturada por uno de los bloques *catch*. El siguiente código muestra un ejemplo del funcionamiento de *finally*.

```
class Ejemplo3 {

    static public void NoDisparaExcepciones() {
        System.out.println("Inicio de 'NoDisparaExcepciones'");
        try {
            System.out.println("Dentro del bloque try de 'NoDisparaExcepciones'");
        }
        finally {
            System.out.println("Dentro del bloque finally de 'NoDisparaExcepciones'");
        }
        System.out.println("Fin de 'NoDisparaExcepciones'");
    }

    static public void DisparaCapturaExcepcion() {
        System.out.println("Inicio de 'DisparaCapturaExcepcion'");
        try {
            System.out.println("Dentro del bloque try de 'DisparaCapturaExcepcion'");
            throw new Exception("Excepcion en try de 'DisparaCapturaExcepcion'");
            // el codigo restante de este try nunca se ejecutara
        }
        catch(Exception ex) {
            System.out.println("Dentro del bloque catch de " +
                "'DisparaCapturaExcepcion', excepcion=" + ex.getMessage());
        }
        finally {
            System.out.println("Dentro del bloque finally de 'DisparaCapturaExcepcion'");
        }
        System.out.println("Fin de 'DisparaCapturaExcepcion'");
    }

    static public void DisparaNoCapturaExcepcion() throws Exception {
        System.out.println("Inicio de 'DisparaNoCapturaExcepcion'");
        try {
            System.out.println("Inicio del bloque try de 'DisparaNoCapturaExcepcion'");
            throw new Exception("Excepcion dentro del try de 'DisparaNoCapturaExcepcion'");
            // el codigo restante de este try nunca se ejecutara
        }
        finally {
            System.out.println("Dentro del bloque finally de 'DisparaNoCapturaExcepcion'");
        }
        // el codigo restante de este metodo nunca se ejecutara
    }

    static public void DisparaCapturaRedisparaExcepcion() throws Exception {
        System.out.println("Inicio de 'DisparaCapturaRedisparaExcepcion'");
        try {
            System.out.println("Inicio del try de DisparaCapturaRedisparaExcepcion'");
            throw new Exception("Excepcion en try de 'DisparaCapturaRedisparaExcepcion'");
            // el codigo restante de este try nunca se ejecutara
        }
        catch(Exception ex) {
            System.out.println("Dentro del bloque catch de " +
                "'DisparaCapturaRedisparaExcepcion', excepcion=" + ex.getMessage());
            throw ex;
        }
        finally {
            System.out.println("Dentro del bloque finally de " +
                "'DisparaCapturaRedisparaExcepcion'");
        }
        // el codigo restante de este try nunca se ejecutara
    }

    public static void main(String args[]) {
        System.out.println("Llamando 'NoDisparaExcepciones'");
        NoDisparaExcepciones();

        System.out.println("\nLlamando 'DisparaCapturaExcepcion'");
        DisparaCapturaExcepcion();

        System.out.println("\nLlamando 'DisparaNoCapturaExcepcion'");
        try {
```

```
            DisparaNoCapturaExcepcion();
        }
        catch(Exception ex) {
            System.out.println("Dentro del bloque catch 1 de 'main', excepcion=" +
                ex.getMessage());
        }
        finally {
            System.out.println("Dentro del bloque finally 1 de 'main'");
        }

        System.out.println("\nLlamando 'DisparaCapturaRedisparaExcepcion'");
        try {
            DisparaCapturaRedisparaExcepcion();
        }
        catch(Exception ex) {
            System.out.println("Dentro del bloque catch 2 de 'main', excepcion=" +
                ex.getMessage());
        }
        finally {
            System.out.println("Dentro del bloque finally 2 de 'main'");
        }
    }
}
```

El código anterior genera una salida semejante a la siguiente:



Esto da una clara idea de que el bloque *finally* se ejecuta siempre que se haya entrado a ejecutar el bloque *try* correspondiente. La siguiente es la lista de combinaciones *try-catch-finally* que se pueden tener:

- Un bloque *try* seguido de uno o más bloques *catch*.
- Un bloque *try* seguido de uno o más bloques *catch*, seguidos de un bloque *finally*.
- Un bloque *try* seguido de un bloque *finally*.

2.3. C#

C# cuenta con un soporte completo, muy similar al de Java.
El siguiente es el árbol de herencia de las clases para excepciones.

System.Object
    System.Exception
        System.ApplicationException
        System.SystemException

La clase *Exception* es la clase base de todos los tipos de excepciones, al igual que Java.

La clase *SystemException* es la clase base que utiliza el intérprete de .NET para todas las excepciones que puede generar.

La clase *ApplicationException* es la clase base que debe utilizar el programador para definir sus propias excepciones.

A diferencia de Java, C# no diferencia entre excepciones que deben tratarse y las que solo sirven para depuración, es decir, si un determinado código puede generar una excepción, no es obligatorio colocarlo dentro de un bloque *try* seguido de un *catch* que lo capture. Por el mismo motivo, tampoco existe una cláusula *throws* con la que se indique que una excepción no se desea tratar en el método donde ocurre.

Fuera de las diferencias antes indicadas, la implementación en Java y C# es igual. El siguiente es un código de ejemplo en C#.

```
using System;

class ExcepcionFormatoInvalido : ApplicationException {
        public ExcepcionFormatoInvalido(string mensaje) : base(mensaje) {}
}

class ExcepcionFechaInvalida : ApplicationException {
        public ExcepcionFechaInvalida(string mensaje) : base(mensaje) {}
}

class Fecha {
        int dia, mes, anho;
        public Fecha(string sFecha) {
                char[] delimitadores = {'-','/'};
                string[] tokens = sFecha.Split(delimitadores);
                if(tokens.Length != 3)
                        throw new ExcepcionFormatoInvalido(
                                "Los delimitadores son inválidos");
                dia = Convert.ToInt32(tokens[0]);
                mes = Convert.ToInt32(tokens[1]);
                anho = Convert.ToInt32(tokens[2]);
                if(anho < 0 || dia < 1 || 31 < dia || mes < 1 || 12 < mes)
```

```
                        throw new ExcepcionFechaInvalida("La fecha no existe");
        }
        public void Imprimir() {
                Console.WriteLine("[" + dia + "/" + mes + "/" + anho + "]");
        }
};

class MainClass
{
        public static void Main(string[] args)
        {
                Console.WriteLine("Inicio");
                try {
                        Console.Write("Ingrese una fecha : ");
                        string sFecha = Console.ReadLine();
                        Fecha fecha = new Fecha(sFecha);
                        Console.Write("La fecha ingresada es : ");
                        fecha.Imprimir();
                }
                catch(Exception ex) {
                        Console.WriteLine("Excepción = " + ex);
                }
                finally {
                        Console.WriteLine("Ejecutandose el bloque finally");
                }
                Console.WriteLine("Fin");
        }
}
```