

C# II

Instructor: Ing. Gabriel Roa
Email: gabrielrb14a@gmail.com
Teléfono: 0424-7592768
Fecha: Enero del 2022

AGENDA

- Presentación.
- Plan de evaluación.
- Excepciones.
 - Qué son.
 - Manejo con Try – Catch – Finally
- Excepciones propias.
- Log de excepciones.

El Instructor

- Mi nombre es Gabriel Roa. Tengo 24 años, Ingeniero en Informática de la UCLA.
- Co-Fundador y Director Ejecutivo de KuroDev. Profesor de Fundamentos de Algoritmos en la Universidad de Carabobo.
- Programador en Python, C#, Java, Javascript, con experiencia en desarrollo de aplicaciones web y en gestión de proyectos.
- Amante de los gatos, la tecnología, el cine y el ciclismo.

Objetivos del Curso

- Comprender cómo utilizar C# para desarrollar aplicaciones de escritorio de mediana complejidad.
- Aprender a manejar, capturar y logear excepciones.
- Aprender a manejar las distintas herramientas de depuración que nos ofrece Visual Studio.
- Conocer y entender qué es SQL Server, cómo funciona y cuáles son sus particularidades.

Objetivos del Curso

- Desarrollar funciones y expresiones lambda que permitan resolver problemas sencillos y aplicarlas para escribir sentencias LINQ.
- Estudiar los ORM, su existencia, ventajas, desventajas, y por qué debemos utilizarlos.
- Conocer el ORM Entity Framework Core.
 - Definir modelos y sus relaciones utilizando EF Core y Fluent API.
 - Escribir programas que interactúen con una base de datos a través de EF Core.

Indicaciones

- Prestar atención en los momentos de explicación.
- Conversar ordenadamente, sin interrumpirnos los unos a los otros.
- Tomar notas de las cosas que consideren importantes, tanto teóricas como prácticas.
- ¡Preguntar! TODAS las preguntas son absolutamente válidas.
- Usaremos el aula virtual de gracosoft.com para envío del material y de asignaciones, y un grupo de Telegram para comunicarnos más de cerca.
- Pueden escribirme al WhatsApp o al Telegram: 0424 759 2768 - @gaboroa14.

Plan de evaluación


Fecha	Tipo	Contenido	Ponderación
26-ene	Quiz teórico	Excepciones y depuración	15%
28-ene	Ejercicio Práctico (Laboratorio)	Creación de BD y tablas con SQL Server Definición de MER para resolver el problema	20%
01-feb	Ejercicio Práctico (Laboratorio)	Funciones Lambda	25%
03-feb	Quiz teórico	ORM	15%
04-feb	Ejercicio Práctico (Laboratorio)	Definición de modelos y CRUD con EF Core	25%

Comencemos



¿Qué es una excepción?

- Consideremos el siguiente fragmento de código:

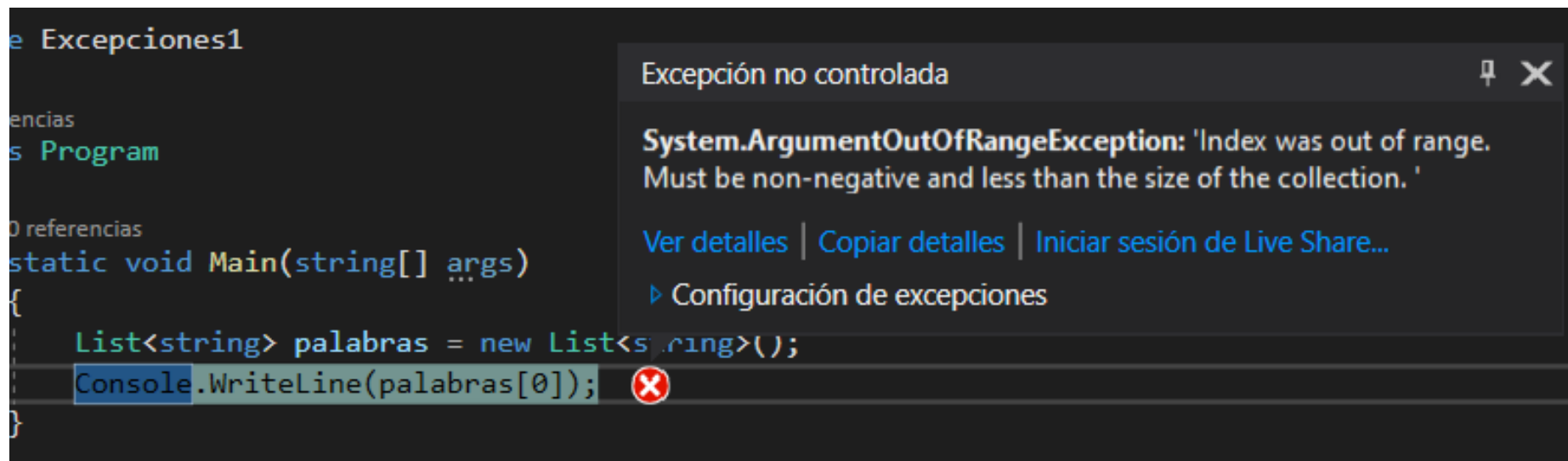


```
static void Main(string[] args)
{
    List<string> palabras = new List<string>();
    Console.WriteLine(palabras[0]);
}
```

- ¿Qué pasa si lo intentamos ejecutar?

¿Qué es una excepción?

- Ocurre un error que detiene la ejecución del programa.



```
e Excepciones1
encias
s Program
0 referencias
static void Main(string[] args)
{
    List<string> palabras = new List<string>();
    Console.WriteLine(palabras[0]);
}
```

Excepción no controlada

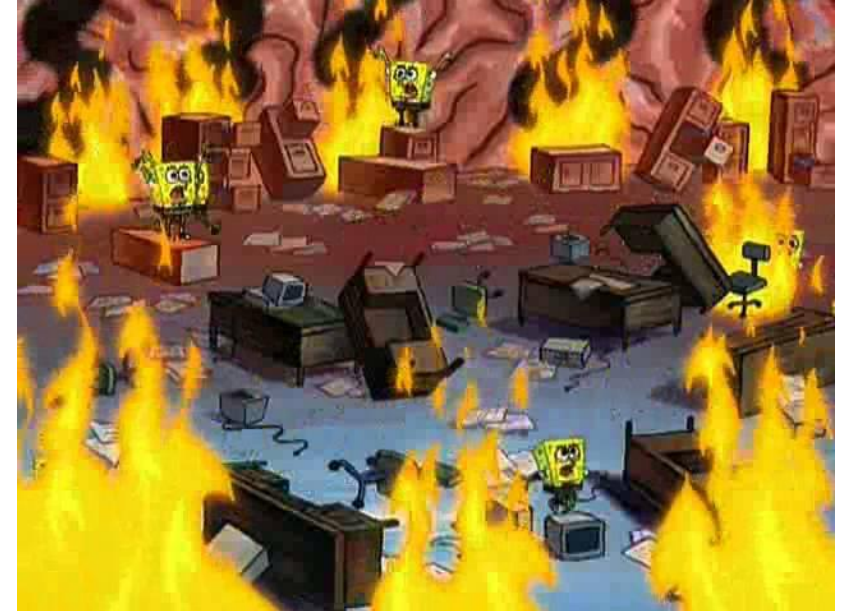
System.ArgumentOutOfRangeException: 'Index was out of range. Must be non-negative and less than the size of the collection.'

[Ver detalles](#) | [Copiar detalles](#) | [Iniciar sesión de Live Share...](#)

▸ Configuración de excepciones

¿Qué es una excepción?

- Es la indicación de que ocurrió un error en el programa.
- Estos errores ocurren de manera excepcional – de allí su nombre.
- Suelen ser inesperados y errores surgidos de problemas con nuestra lógica de programación.
- Pueden ser controladas y no controladas.



¿Qué es una excepción?

Controladas

- Hay un fragmento de código que evita que el error finalice la ejecución del programa.
- Se suele hacer un log del error.
- Se controlan a través del try / catch.

No controladas

- El error finaliza por la fuerza la ejecución del programa.
- Si estamos en modo de depuración, podremos ver una descripción de la misma. En caso contrario, sólo aparecerá una ventana de error sin más.

Try – Catch – Finally

- Cláusula que nos permite *intentar* ejecutar un fragmento de código.
- En caso de que se lance una excepción, nos permite *capturar* la misma y ejecutar una serie de acciones para controlarla.
- Tras cualquier intento, ejecuta *finalmente* un cierto fragmento de código para finalizar.

```
try {  
    //Código a intentar ejecutar  
}  
catch (Exception e)  
{  
    //Código a ejecutar si ocurre una excepción  
}  
finally  
{  
    //Código a ejecutar al finalizar el try y/o el catch  
}
```

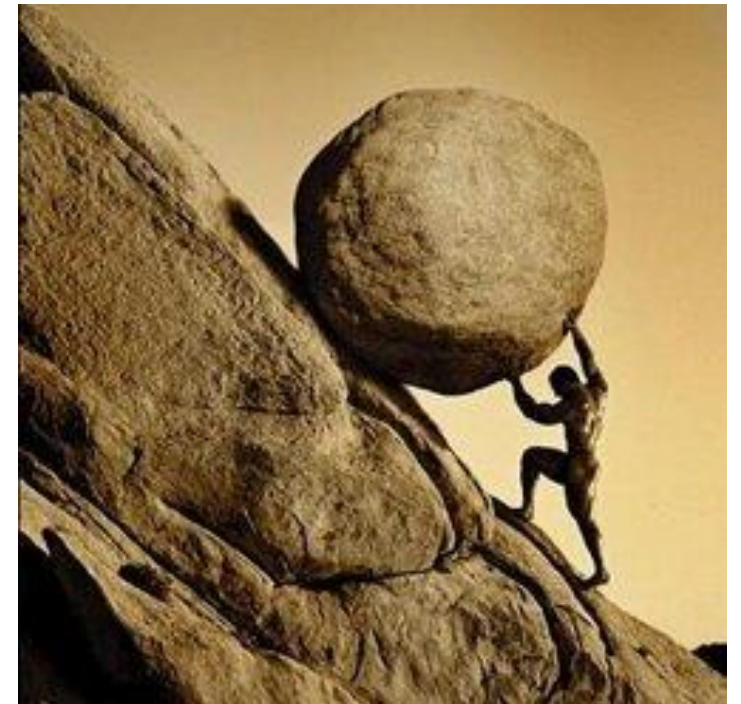
Try – Catch – Finally

- Para saber si debemos usar un try – catch – finally, debemos primeramente preguntarnos: ¿hay alguna posibilidad de que el código falle? En cuyo caso, procedemos a envolver nuestro código con este.



Try

- Contiene el fragmento de código que quiero intentar ejecutar y es susceptible a cualquier tipo de error.
- En él, va el código que normalmente programaríamos si no estuviésemos aplicando el try – catch.



Catch

- Contiene el código a ejecutar en caso de detectar una excepción.
- Normalmente registra la información del error, notifica al usuario del mismo, reintenta la acción del error o ejecuta otro fragmento de código enteramente distinto.

```
try
{
} catch (ClaseExcepcion nombreExcepcion)
{
}
```


Catch

- Funciona similar a un método que recibe un parámetro: un objeto derivado de Exception que contiene la información de la excepción que acaba de ocurrir.
- Se pueden concatenar catches, considerando que cada uno de ellos recibirá como parámetro un objeto de una clase distinta (sobrecarga)

```
catch (ApplicationException e){  
    //Código a ejecutar cuando suceda una excepción de tipo ApplicationException  
}  
catch (ArgumentException e)  
{  
    //Código a ejecutar cuando suceda una excepción ArgumentException  
}  
catch (DivideByZeroException e)  
{  
    //Código a ejecutar cuando suceda una excepción DivideByZeroException  
} //etc
```

Catch

- Todas las excepciones derivan de una clase base, la clase Exception, y define una serie de propiedades y métodos por defecto que podemos utilizar para evaluar la excepción dentro del catch.
- Podemos capturar las excepciones Exception, en cuyo caso todas las excepciones serán manejadas por un mismo Catch.



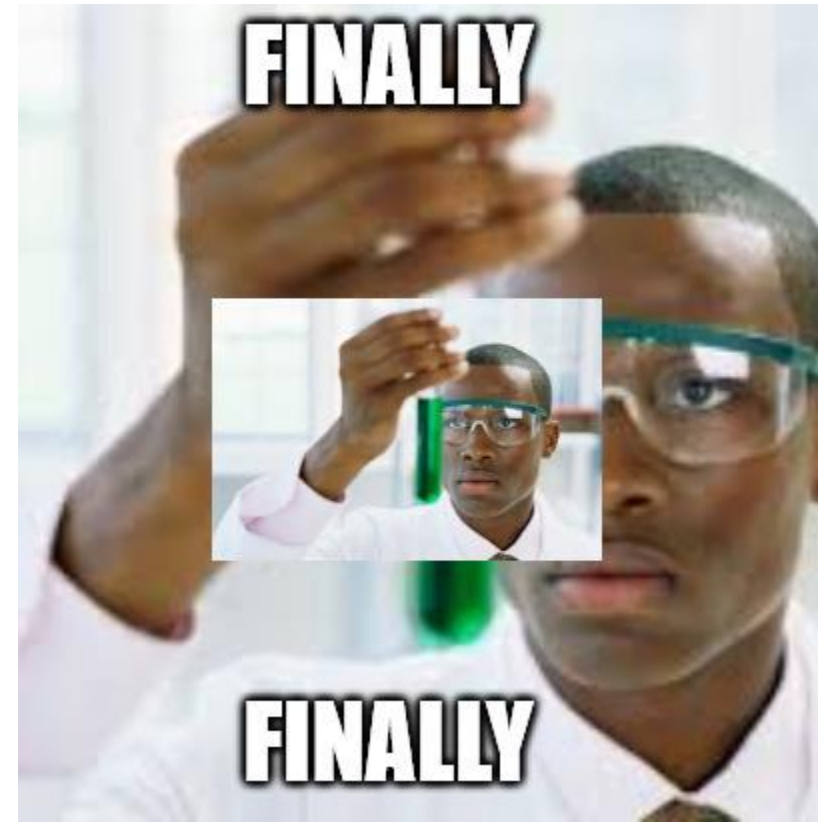
Ejemplo

- Escriba el código de un programa que lea por consola dos números decimales e imprima en pantalla la suma de los mismos. Utilice las cláusulas try – catch para capturar todas las posibles excepciones de este código.



Finally

- Ejecuta un fragmento de código al final de la ejecución del bloque try – catch.
- Este fragmento se ejecuta sin importar si ocurrió una excepción o no.
- Útil para liberar recursos o efectuar acciones que deshagan lo que hayamos llevado a cabo dentro del try, en caso de que sea necesario.



Excepciones propias

- C# nos brinda la posibilidad de *arrojar* nuestras propias excepciones en cualquier momento que deseemos a través de la palabra *throw*.
- Podemos hacer uso de esta posibilidad para escribir un código que emita y capture errores propios nuestros de manera controlada.



Ejemplo

- Escriba un programa que defina una clase Estudiante, lea por pantalla de consola los datos del mismo y valide a través de una excepción `ArgumentOutOfRangeException` si el mismo es menor de edad o no.



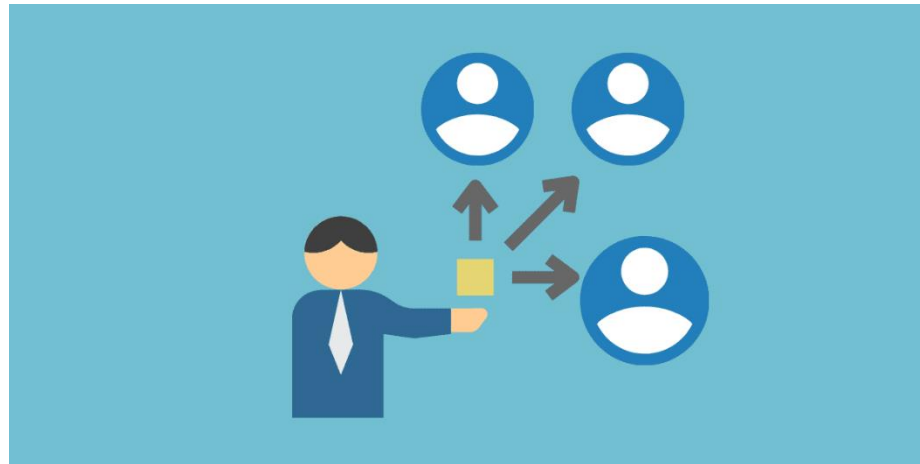
Construyendo nuestras propias Excepciones

- Es posible escribir una clase que defina nuestras propias excepciones, esto es, una clase derivada de Exception que podremos utilizar para arrojar nuestras propias excepciones.
- Esta clase debe cumplir con las siguientes características:



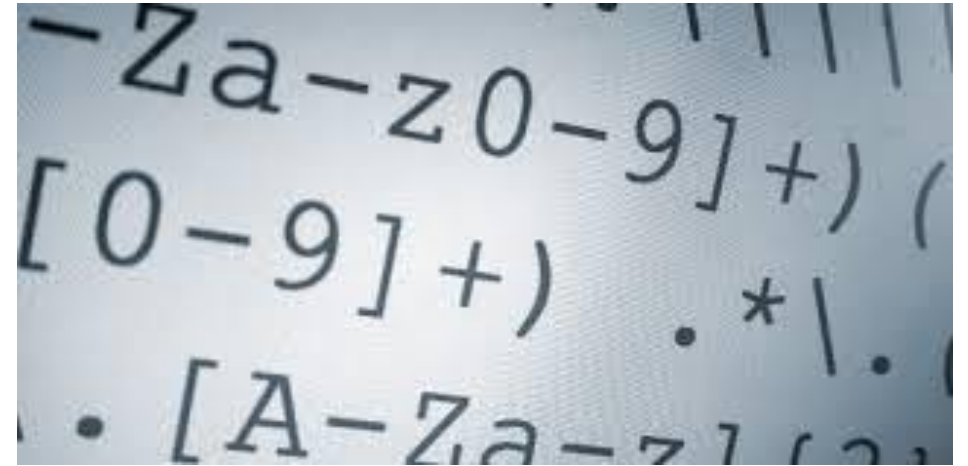
Construyendo nuestras propias Excepciones

- Debe heredar de la clase Exception.
- Su nombre debe finalizar con la palabra Exception.
- Debe implementar los tres constructores de la clase Exception.



Ejemplo

- Escriba una excepción que pueda arrojar cuando se intente crear un Estudiante, según el ejemplo anterior, y alguna de sus propiedades tenga un valor inválido. Esta excepción deberá tener cuál fue el campo que recibió, y un mensaje de error explicando qué sucedió.



Log en archivos .log

- Normalmente, los errores suelen registrarse para que el programador pueda verificarlos.
- Esto sobre todo cuando suceden errores que no están *planificados* ni forman parte de lo que el programador tenía en mente cuando quiso programar.
- Este registro se suele hacer en una base de datos, o en su defecto, en un archivo .log.



Log en archivos .log

- Para efectos de este curso, haremos uso de una clase llamada Logger, que nos permitirá registrar en un archivo error.log los errores que ocurran en el programa, junto a una información básica que nos permitirá depurar el error en una futura ocasión.

```
56,001] DEBUG [views.py:15] ==> Debug Log
56,001] INFO [views.py:16] ==> Info Log
56,001] WARNING [views.py:17] ==> Warning Log
56,005] DEBUG [views.py:15] ==> Debug Log
56,005] INFO [views.py:16] ==> Info Log
56,005] WARNING [views.py:17] ==> Warning Log
56,005] ERROR [views.py:18] ==> Error Log
56,005] CRITICAL [views.py:19] ==> Critical Log
56,005] CRITICAL [views.py:20] =====
56,002] ERROR [views.py:18] ==> Error Log
56,012] CRITICAL [views.py:19] ==> Critical Log
56,012] CRITICAL [views.py:20] =====
39,731] CRITICAL [views.py:19] ==> Critical Log
39,732] CRITICAL [views.py:20] =====
```

Ejemplo

- Escribe un programa de consola que lea los datos de un conjunto de estudiantes y los guarde en un `List<Estudiante>` para luego imprimirlos. Valide y registre las excepciones utilizando `try – catch – finally` y haciendo uso de la clase `Logger` para registrar los errores.



Referencias Bibliográficas

- Farrell, J. (2018). *Microsoft Visual C# 2017: An Introduction to Object-Oriented Programming*. Boston: Cengage Learning.
- Griffiths, I. (2019). *Programming C# 8.0*. Sebastopol: O'Reilly Media, Inc.
- Microsoft. (28 de 01 de 2021). *C# documentation*. Obtenido de Microsoft Docs: <https://docs.microsoft.com/en-us/dotnet/csharp/>