# Clase 13 - 13/10/2020

Tuesday, October 13, 2020

## -Excepciones-

*Las excepciones se generan en tiempo de ejecucion porque cuando ocurra el error tiene que estar en ejecucion el programa, de otra forma no podria ser, tambien porque las excepciones son objetos y los objetos se crean en tiempo de ejecucion. Tambien son resultado de la propia ejecucion o flujo de ejecucion del programa y suceden en tiempo de ejecucion.

*Pueden ser lanzadas desde el codigo de la aplicacion y la plataforma de .NET. Hay excepciones que ya vienen con las bibliotecas propias de la plataforma de .NET o alguna biblioteca que usamos, pero tambien las podemos crear nosotros y lanzarlas con el throw.

*La propiedad InnerException la tienen todas las excepciones, si o si. Porque lo heredan de exception. Las clases que representan a las clases, TODAS, derivan de exception, lo que no heredan son los constructores. La propiedad la tienen por mas de que no se pueda inicializar.

*Dentro del bloque Try se ubica el codigo que podria generar una excepcion.
-El bloque Try: lo que esta adentro del try es todo el codigo que podia llegar a emitir a un tipo de excepcion. Todo el codigo que puede generar la excepcion esta en el bloque TRY.
-El bloque Catch: captura la excepcion. Contiene el codigo para manejar la excepcion.

*En una estructura con multiples bloques catch solo se puede ejecutar un bloque catch. Porque funciona como el switch, va a empezar por el bloque catch que este mas arriba en el codigo y va a empezar a comparar el tipo de la excepcion que captura con el tipo de la excepcion que se lanzo. Un catch puede lanzar una excepcion pero dentro de un mismo try catch un bloque catch que este mas abajo no deberia capturarla de nuevo.

*Cuando se maneja una excepcion, la ejecucion sigue a continuacion del bloque Try-Catch que la manejó.
Una vez que se capturo una excepcion, se manejó, la ejecucion continua justo despues del bloque try catch que la manejo porque hay se corta la ejecucion hasta que encuentra un bloque try catch.

*Que es un stack trace: Una coleccion de los metodos invocados previa al lanzamiento de la excepcion.
Es una coleccion lifo porque es una pila, donde cuando se lanzo una excepcion y va desde el ultimo metodo ejecutado al primero y tiene la coleccion de todos los metodos que se fueron invocando. Empezando por el ultimmo en ejecutarse cuando se lanzo la excepcion hasta el primero que se ejecuto que fue cuando se inicio la aplicacion. Todo es esta guardado dentro de la excepcion.
Una coleccion de las excepciones producidas previamente seria la InnerException. Si bien no es una collecion, es una propiedad de tipo exception es como una excepcion anidada dentro de otra.
Se puede llegar a ver el historial de excepciones producidas previamente, no es una coleccion, son propiedades anidadas.

*La expresion Throw ex; reinicia el stack trace.
Cuando hacemos throw ex, throw y la variable del catch que almacena el objeto de la excepcion o su referencia, se reiniciar el stack trace. Volvemos a lanzar la excepcion pero toda la pila de llamadas previas se pierde. Si se hace throw solo la pila se conserva.

*El bloque finally se ejecuta siempre, si se ejecuto bien el codigo y no se lanzo ninguna excepcion, si se capturo una excepcion en el catch o hay una nueva, no importa lo que suceda se va a ejecutar siempre.

# Clase 14 - 14/10/2020

Wednesday, October 14, 2020

-Test Unitarios-

## Pruebas:

*Prueba integral: Prueba que todos los elementos que componen el software funcionan juntos correctamente.

*Prueba funcional: Prueba que el sistema hace lo que se pidio (desde el punto de vista del cliente/usuario).

*Prueba unitaria: Prueban induvidualmente y de forma aislada cada servicio o funcion de un modulo.

De la prueba mas cara (dificil de automatizar) a la mas barata:
Funcionales - Integrales - Unitarias.

---

## Fases del patron AAA:

En la fase ARRANGE del patron AAA:
Se inicializan los objetos y establece el valor de los datos que se pasa al metodo en pruebas.
Es donde se generan los casos de prueba en forma de obejtos, se le dan valores y se prepara todo para hacer a prueba en especifico.

En la fase ASSERT del patron AAA:
Se compueba si la accion del codigo de pruebas se comporta de la forma prevista.

En la fase ACT del patron AAA:
Es cuando ejecuto el metodo a probar.

# Clase 15 - 20/10/2020

Tuesday, October 20, 2020

## -Generics-

*Una clase generia es una clase que encapsula operaciones que no son especificas de un tipo particular de dato, permite crear clases flexibles que permiten reutilizar el codigo.

*Se puede tener más de un parámetro/comodín/tipo genérico en una clase o método. Por ejemplo: los diccionarios.

Una restriccion o constraint de un tipo generico sirve para que se le indique a la clase que es lo que puede aceptar, que es lo que puede hacer ese tipo de dato y en caso de que no este no pasa nada, entraria cualquier tipo de dato. Lo que hacen las restricciones es informar al compilador las caracteristicas que debe de tener el tipo de dato para que sea valido para ese tipo de clase generica. Si no se le pone restriccion puede ser de cualquier tipo.

*Si intentamos instanciar una clase genérica pasando como argumento un tipo que no cumple con las restricciones tira error en tiempo de diseño, el IntelliSense avisa. Si no, cuando intentamos compilar no deja. Las clases genericas se resuelven en tiempo de compilacion.

*Se puede aplicar más de una restricción por tipo genérico, seria el tipo de dato : (dos puntos) las restricciones que se quieran. Con , (coma) se aplicaria mas de una restriccion por parametro.

*Puedo declarar métodos genéricos en clases no-genéricas.

*Los comodines o parámetros genéricos pueden tener cualquier nombre/identificador que siga las reglas del lenguaje.

---

*Tabla de restricciones:

| Restricciones | Descripcion |
|---|---|
| where T : struct | El dato tiene que ser un tipo por valor. |
| where T : class | Tiene que ser un tipo por referencia. Puede ser una interfaz. |
| where T : notnull | Que no sea null. Es de c#8 y no lo vimos. Es que un tipo sea nulleable. |
| where T : new() | Que tenga un constructor por defecto. Un contructor publico sin parametros. |
| where T : <clase base> | Que sea del tipo de la clase que se le esta pasando o sus derivadas. |
| where T : <interface> | Lo mismo que el 5 pero con interfaz. Su interfaz  y sus derivadas. |
| where T : U | Que T sea el mismo tipo que U o derivado. |