Excepciones

15

Programación II y Laboratorio de Computación II

Edición 2018 – Rev. 2022

Gestión de errores

- La gestión de errores es la técnica que permite interceptar con éxito errores en tiempo de ejecución esperados y no esperados.
- En C# la gestión de errores se controla por medio de excepciones.
- Cuando se produce un error se ***lanza*** una excepción.
- El programa debe construirse usando diferentes técnicas de gestión de errores para ***atrapar*** las excepciones y administrarlas de manera conveniente.

Excepciones

- Cuando algo va mal mientras un programa de C# se está ejecutando, se inicia una excepción.
- Las excepciones detienen el flujo actual del programa, y si no se hace nada, el programa dejará de funcionar.
- Se producen por un error en el programa, por ejemplo, si se divide un número por cero, o pueden ser el resultado de alguna entrada inesperada, por ejemplo, cuando un usuario selecciona un archivo que no existe.
- El programador debe habilitar su programa para que resuelva estos problemas sin bloquearse.

Objeto Exception

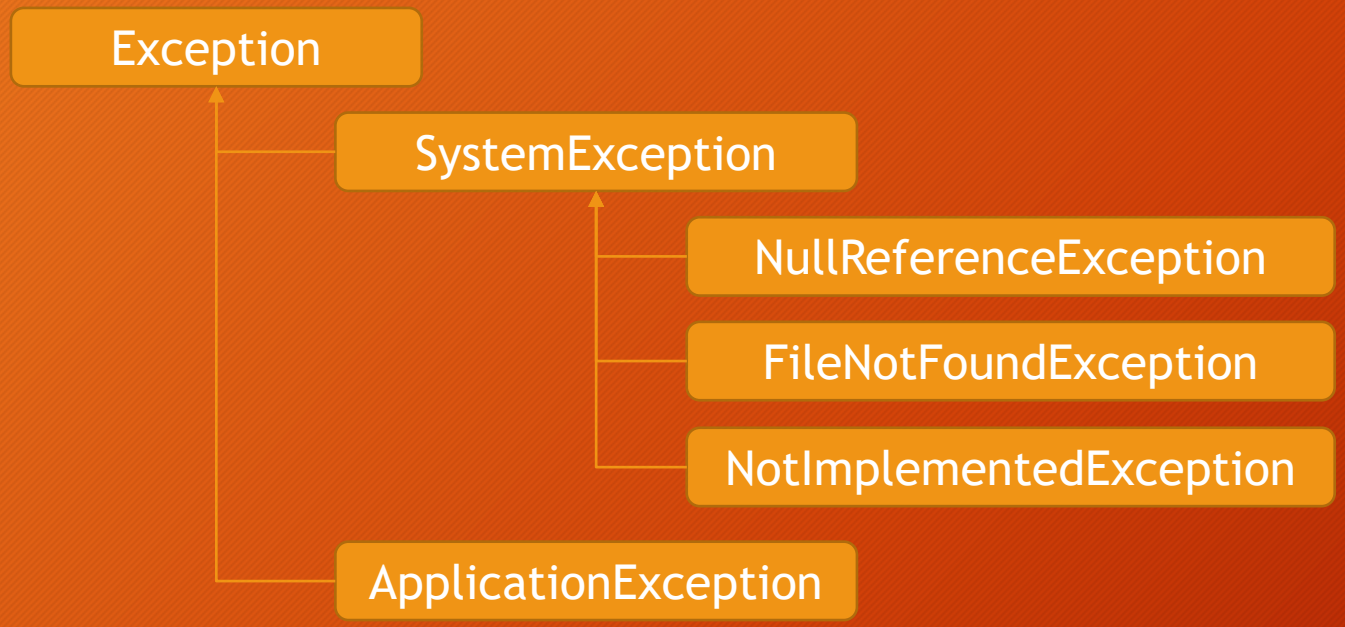
- Todas las excepciones derivan de la clase **Exception**, que es parte del runtime de lenguaje común (CLR).

- Ventajas:
  - Los mensajes de error no están representados por valores enteros o enumeraciones.
    - Los valores enteros de programación, como -3, desparecen y en su lugar se utilizan clases concretas, como **OutOfMemoryException**.
  - Cada clase de excepción puede residir dentro de su propio archivo de origen y no está vinculada con las demás clases de excepción.

Objeto Exception

- Se generan mensajes de error significativos.
- Cada clase de excepción es descriptiva y representa un error concreto de forma clara y evidente.
  - En lugar de un –3, se utiliza una clase llamada **OutOfMemoryException**.
- Cada clase de excepción contiene también información específica.
  - Por ejemplo, una clase **FileNotFoundException** podría contener el nombre del archivo no encontrado.

Objeto Exception



Bloque Try - Catch

- Los bloques **try-catch** son la solución que ofrece la orientación a objetos a los problemas de tratamiento de errores.
- La idea consiste en separar físicamente las instrucciones básicas del programa para el flujo de control normal de las instrucciones para tratamiento de errores.
- Así, las partes del código que podrían lanzar excepciones se colocan en un bloque **try**, mientras que el código para tratamiento de excepciones en el bloque **try** se pone en un bloque **catch** aparte.

Bloque Try - Catch

```
try
{
    // Código a controlar
}
catch (Exception identificador)
{
    // Control de la excepción, en caso de que se produzca.
    // ClaseException tiene que ser de System.Exception
    //o una clase derivada.
    // El identificador, que es opcional, es una variable
    //local de sólo lectura en el ámbito del bloque catch.
}
```

Ejemplo

```
void MetodoExcepcion()
{
    try
    {
        Console.WriteLine("Escriba un número");
        int i = int.Parse(Console.ReadLine());
    }
    catch (OverflowException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Bloque Try - Catch

- El bloque **try** contiene una expresión que puede generar la excepción.
- En caso de producirse la excepción, el runtime detiene la ejecución normal y empieza a buscar un bloque **catch** que pueda capturar la excepción pendiente (basándose en su tipo).
- Si en la función inmediata no se encuentra un bloque **catch** adecuado, el runtime desenreda la pila de llamadas en busca de la función de llamada.
- Si tampoco ahí encuentra un bloque **catch** apropiado, busca la función que llamó a la función de llamada y así sucesivamente hasta encontrar un bloque **catch** (o hasta llegar al final, en cuyo caso se cerrará el programa).

Bloque Try - Catch

- Si encuentra un bloque **catch**, se considera que la excepción ha sido capturada y se reanuda la ejecución normal desde el cuerpo del bloque **catch** (que, en el caso de la diapositiva, escribe el mensaje contenido en el objeto excepción **OverflowException**).

- Por lo tanto, el uso de bloques **try-catch** hace que las instrucciones para tratamiento de errores no se mezclen con las instrucciones lógicas básicas, por lo que el programa es más fácil de interpretar.

Múltiples Catch

- Un bloque de código en una instancia **try** puede contener muchas instrucciones, cada una de las cuales puede producir una o más clases diferentes de excepción.
- Al haber muchas clases de excepciones distintas, es posible que haya muchos bloques **catch** y que cada uno de ellos capture un tipo específico de excepción.
- La captura de una excepción se basa únicamente en su tipo.
- El runtime captura automáticamente objetos excepción de un tipo concreto en un bloque **catch** para ese tipo.

Ejemplo

```
try
{
    Console.WriteLine("Escriba el primer número");
    int i = int.Parse(Console.ReadLine());
    Console.WriteLine("Escriba el segundo número");
    int j = int.Parse(Console.ReadLine());
    int k = i / j;
}
catch (OverflowException e)
{
    Console.WriteLine(e.Message);
}
catch (DivideByZeroException e)
{
    Console.WriteLine(e.Message);
}
```

Catch genérico

- Un bloque **catch** general (Exception), puede capturar cualquier excepción independientemente de su clase y se utiliza con frecuencia para capturar cualquier posible excepción que se pudiera producir por la falta de un controlador adecuado.

- Un bloque **try** no puede tener más que un bloque **catch** general.

- En caso de existir, un bloque **catch** general debe ser el último bloque **catch** en el programa.

Throw

- Cuando necesita lanzar una excepción, el runtime ejecuta una instrucción **throw** y lanza una excepción definida por el sistema.
- Esto interrumpe inmediatamente la secuencia de ejecución normal del programa y transfiere el control al primer bloque **catch** que pueda hacerse cargo de la excepción en función de su clase.
- Es posible utilizar la instrucción **throw** para lanzar excepciones propias.
- Pueden generar excepciones Common Language Runtime (CLR), .NET Framework, las bibliotecas de otros fabricantes o el código de aplicación.

Throw

```
if (minuto < 1 || minuto >= 60)
{
    string fallo = minuto + " no es un minuto válido";
    throw new TiempoInvalidoException(fallo);
}
```

- En este ejemplo se emplea la instrucción **throw** para lanzar una excepción definida por el usuario, TiempoInvalidoException, si el tiempo analizado no es válido.
- En general, las excepciones esperan como parámetro una cadena con un mensaje significativo que se puede mostrar o quedar registrado cuando se captura la excepción.
- También es conveniente lanzar una clase adecuada de excepción.

Throw

```
catch (Exception e)
{
    throw e;
}
```

- Sólo es posible lanzar un objeto si el tipo de ese objeto deriva directa o indirectamente de **System.Exception**.
- Se puede utilizar una instrucción **throw** en un bloque **catch** para volver a lanzar el mismo objeto excepción u otro nuevo.

Throw

```
catch (IOException e)
{
    throw new FileNotFoundException(fallo);
}
```

- En el ejemplo anterior, el objeto **IOException** y toda la información que contiene se pierde cuando la excepción se convierte en un objeto **FileNotFoundException.**
- Es más conveniente ajustar la excepción, añadiendo nueva información pero conservando la que tiene en la propiedad InnerException.

```
catch (IOException e)
{
    throw new FileNotFoundException(fallo, e);
}
```

Bloque Finally

- La cláusula **finally** de C# contiene un conjunto de instrucciones que es necesario ejecutar sea cual sea el flujo de control.
- Las instrucciones del bloque **finally** se ejecutarán aunque el control abandone un bucle **try** como resultado de la ejecución normal porque el flujo de control llega al final del bloque **try**.
- Del mismo modo, también se ejecutarán las instrucciones del bloque **finally** si el control abandona un bucle **try** como resultado de una instrucción **throw** o una instrucción de salto como **break** o **continue**.
- El bloque **finally** es útil en dos casos: para evitar la repetición de instrucciones y para liberar recursos tras el lanzamiento de una excepción.

Bloque Finally

```
try
{
    // Código
}
catch (OverflowException e)
{
    Console.WriteLine(e.Message);
}
catch (DivideByZeroException e)
{
    Console.WriteLine(e.Message);
}
finally
{
    Console.WriteLine("Pulse una tecla para continuar...");
}
```