Исключения среди исключений

Кто я?

- Носов Роман
- Team Lead
- Более 5 лет в энтерпрайз разработке
- Работаю в Аркадии
- Full stack: Angular + .NET

О чем говорим сегодня?

I. Немного теории
II. Немного практики
III. Уроним try-catch-finally?
IV. Заключение

.NET Fremawork ***VS*** .NET Core *****

***** Где есть раззличие

Начнем

I. ***Немного теории***
II. Немного практики
III. Уроним try-catch-finally?
IV. Заключение

.NET Fremawork ***VS*** .NET Core *****

***** Где есть различие

Exceptions

- Represent errors that occur during application execution.
- After an exception is thrown, it is handled by the application or by the default exception handler.
- System.Exception is the base class for all exceptions
- Throwing or handling an exception consumes a significant amount of system resources and execution time.

Любая аргументация выглядит убедительнее если вставлять цитаты мудреца

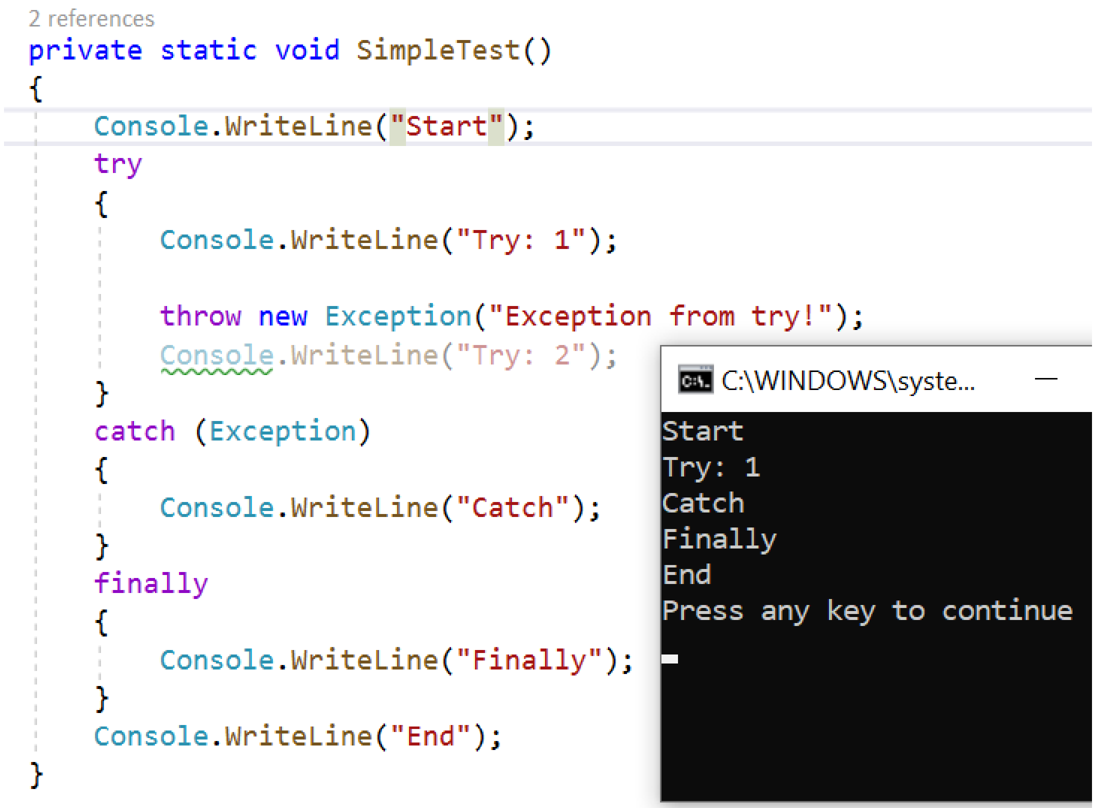
Если мы говорим о .NET, то…

MSDN – и есть наш мудрец

Try-catch-finally

- ***try*** is used with one or more ***catch*** and/or ***finally*** block
- The ***try*** statement provides a mechanism for catching exceptions that occur during execution of a block
- The statements of a ***finally*** block are always executed when control leaves a try statement.

© MSDN



```
2 references
private static void SimpleTest()
{
    Console.WriteLine("Start");
    try
    {
        Console.WriteLine("Try: 1");

        throw new Exception("Exception from try!");
        Console.WriteLine("Try: 2");
    }
    catch (Exception)
    {
        Console.WriteLine("Catch");
    }
    finally
    {
        Console.WriteLine("Finally");
    }
    Console.WriteLine("End");
}
```

C:\WINDOWS\syste...

```
Start
Try: 1
Catch
Finally
End
Press any key to continue
```

Not only try-catch-finally

using (var reader = new StringReader())

```
var reader = new StringReader(manyLines);
try {
    string? item;
    do {
        item = reader.ReadLine();
        Console.WriteLine(item);
    } while(item != null);
} finally
{
    reader?.Dispose();
}
```

lock (x) { /* Your code... */ }

```
object __lockObj = x;
bool __lockWasTaken = false;
try
{
    System.Threading.Monitor.Enter(__lockObj, ref __lockWasTaken);
    // Your code...
}
finally
{
    if (__lockWasTaken) System.Threading.Monitor.Exit(__lockObj);
}
```

Случай из жизни

I. Немного теории
II. *Немного практики*
III. Уроним try-catch-finally?
IV. Заключение

.NET Fremawork ***VS*** .NET Core *****

***** Где есть раззличие

А теперь самое интересное

I. Немного теории

II. Немного практики

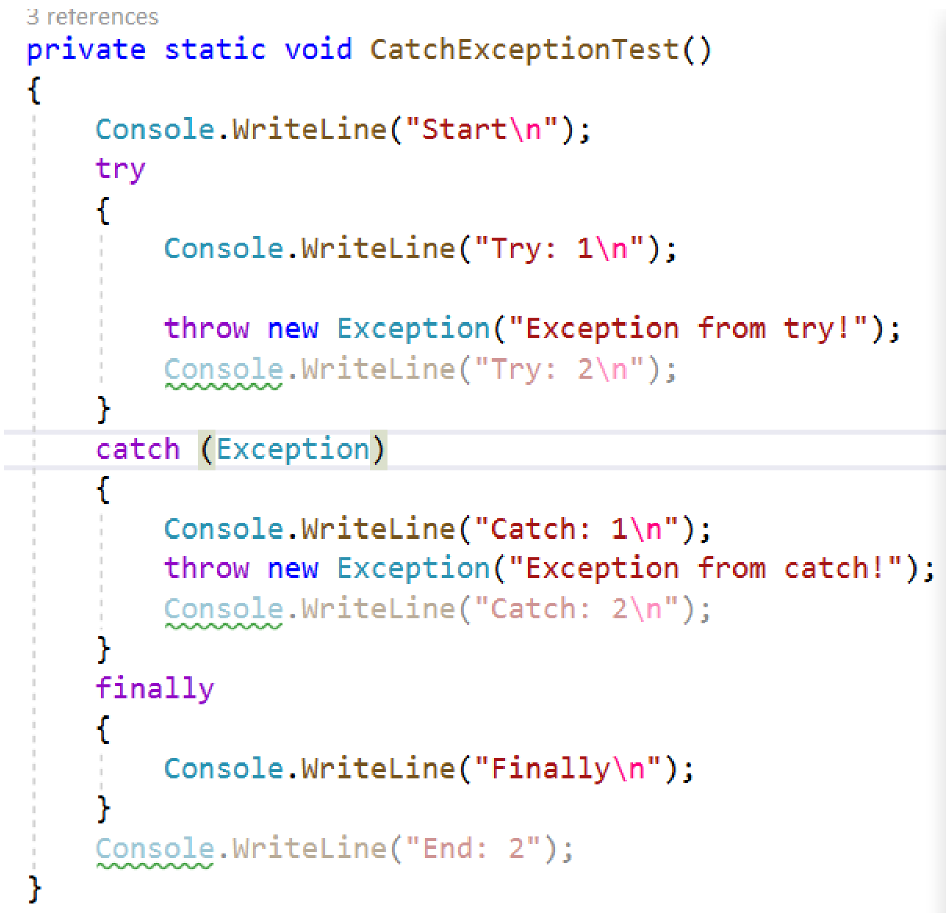
III. *Уроним try-catch-finally?*

IV. Заключение

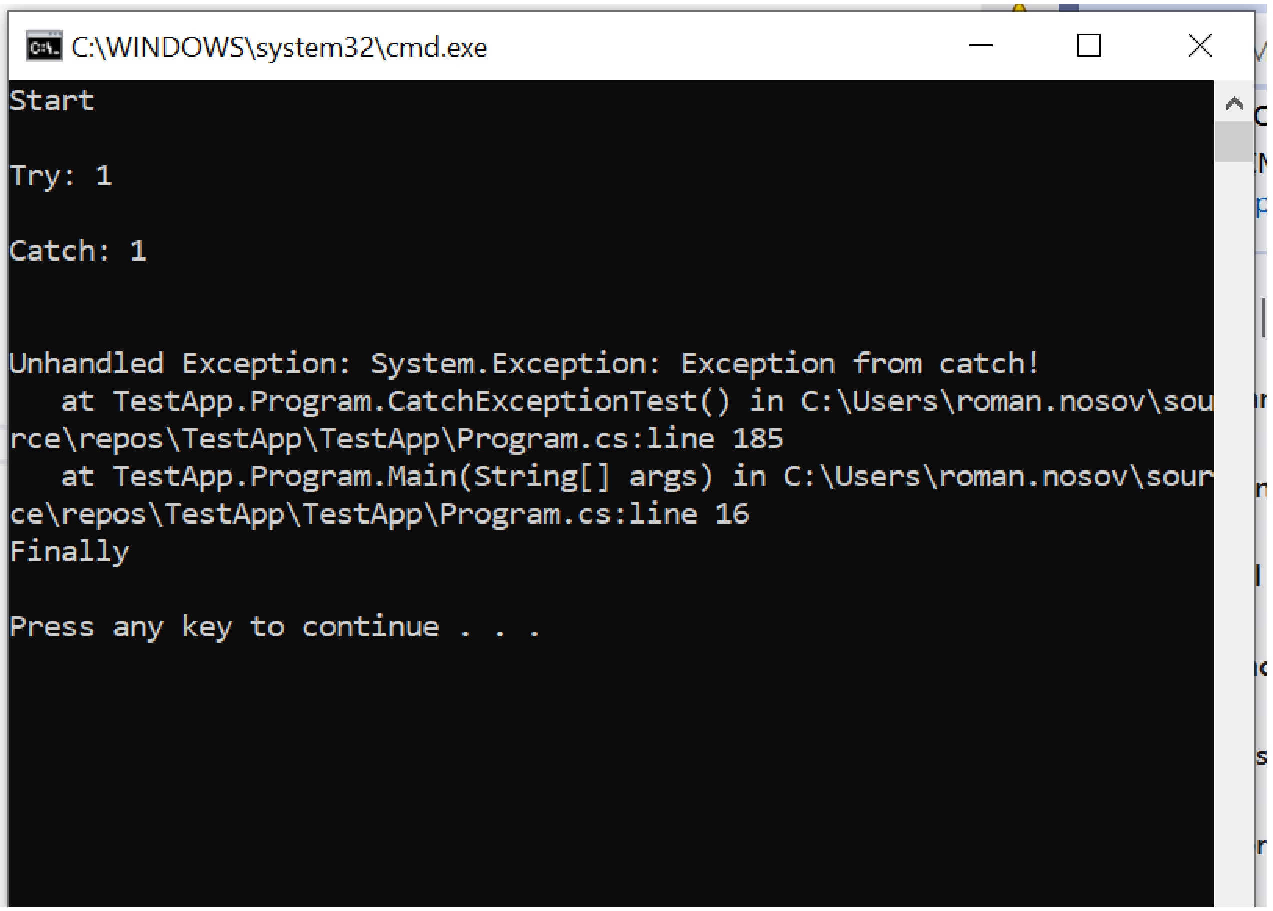
.NET Fremawork ***VS*** .NET Core *****

***** Где есть разлличие

Exception from catch?



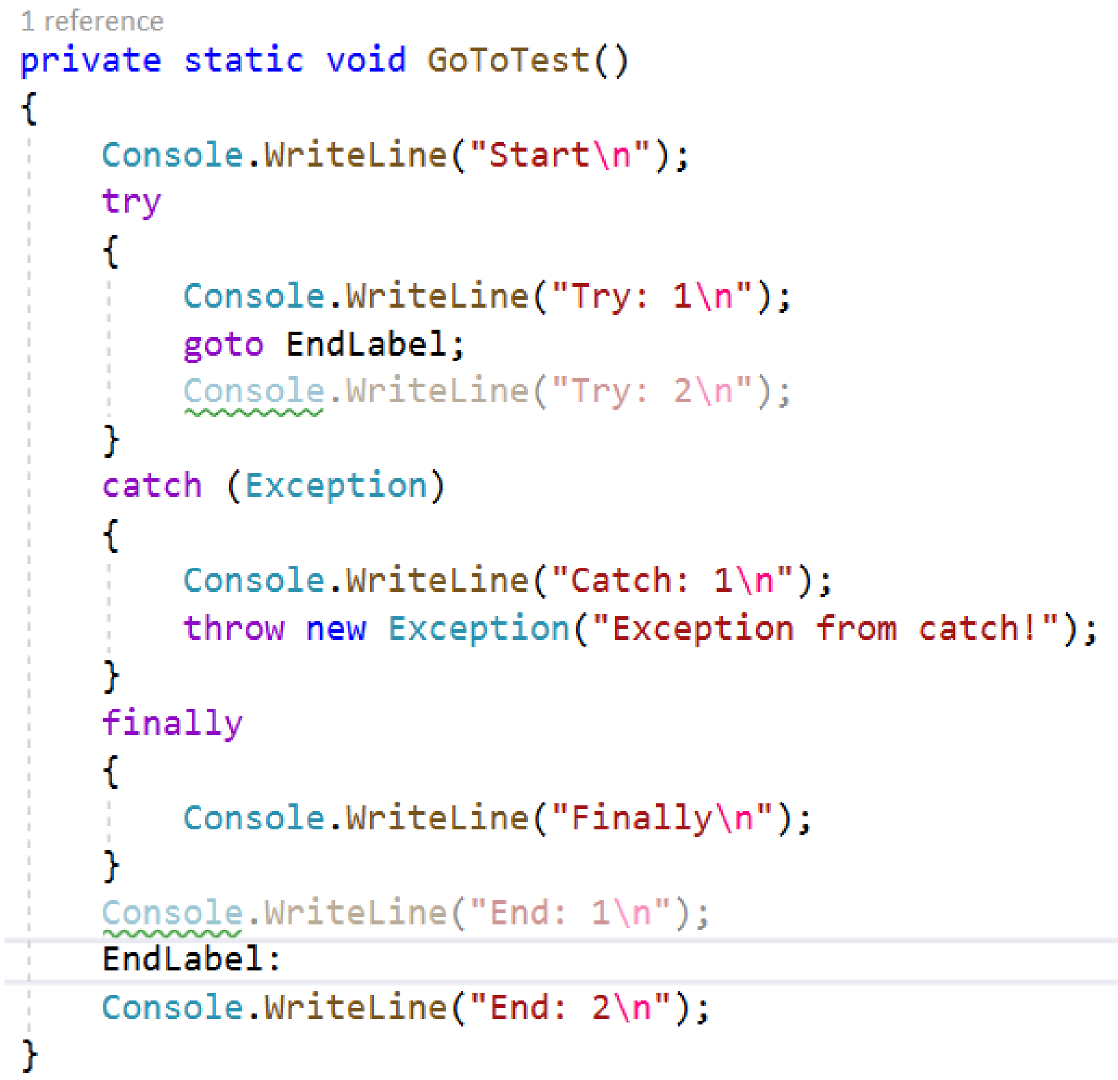
```
3 references
private static void CatchExceptionTest()
{
    Console.WriteLine("Start\n");
    try
    {
        Console.WriteLine("Try: 1\n");

        throw new Exception("Exception from try!");
        Console.WriteLine("Try: 2\n");
    }
    catch (Exception)
    {
        Console.WriteLine("Catch: 1\n");
        throw new Exception("Exception from catch!");
        Console.WriteLine("Catch: 2\n");
    }
    finally
    {
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End: 2");
}
```



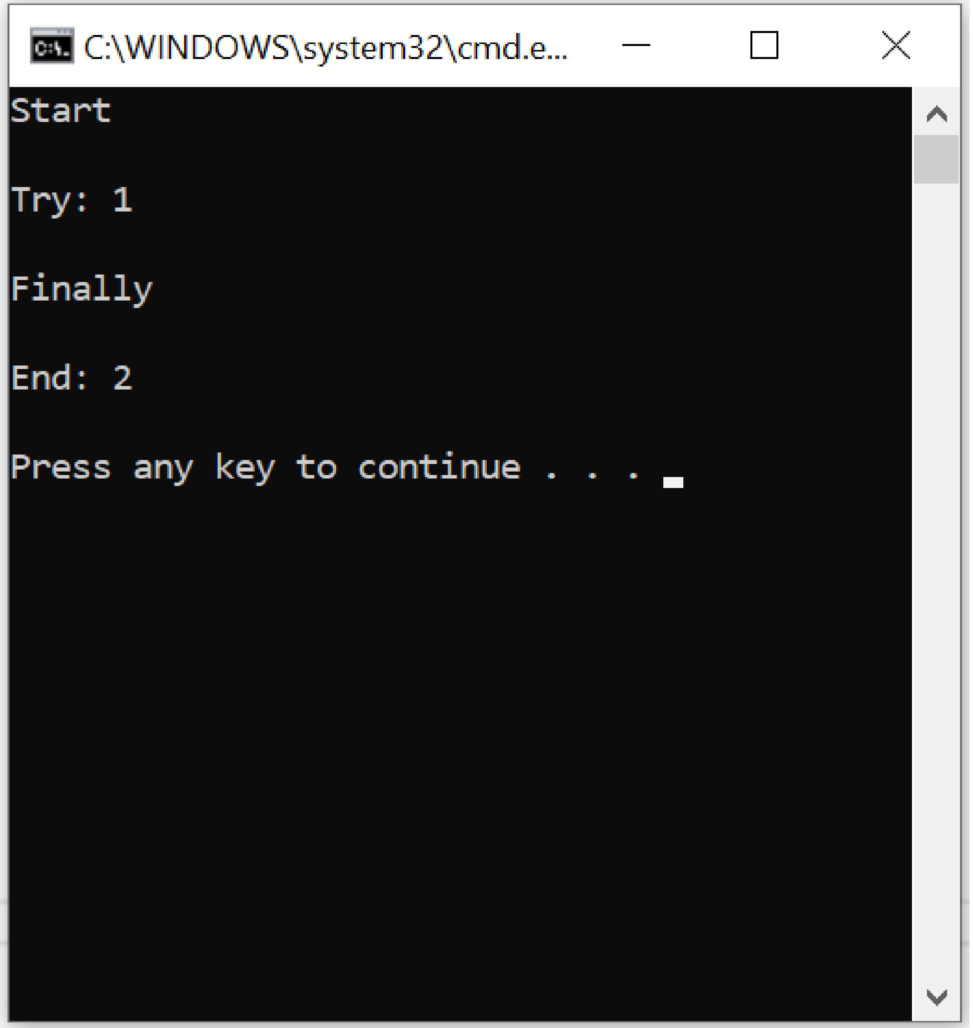
C:\WINDOWS\system32\cmd.exe

```
Start

Try: 1

Catch: 1


Unhandled Exception: System.Exception: Exception from catch!
   at TestApp.Program.CatchExceptionTest() in C:\Users\roman.nosov\sou
rce\repos\TestApp\TestApp\Program.cs:line 185
   at TestApp.Program.Main(String[] args) in C:\Users\roman.nosov\sour
ce\repos\TestApp\TestApp\Program.cs:line 16
Finally

Press any key to continue . . .
```

Maybe GoTo?



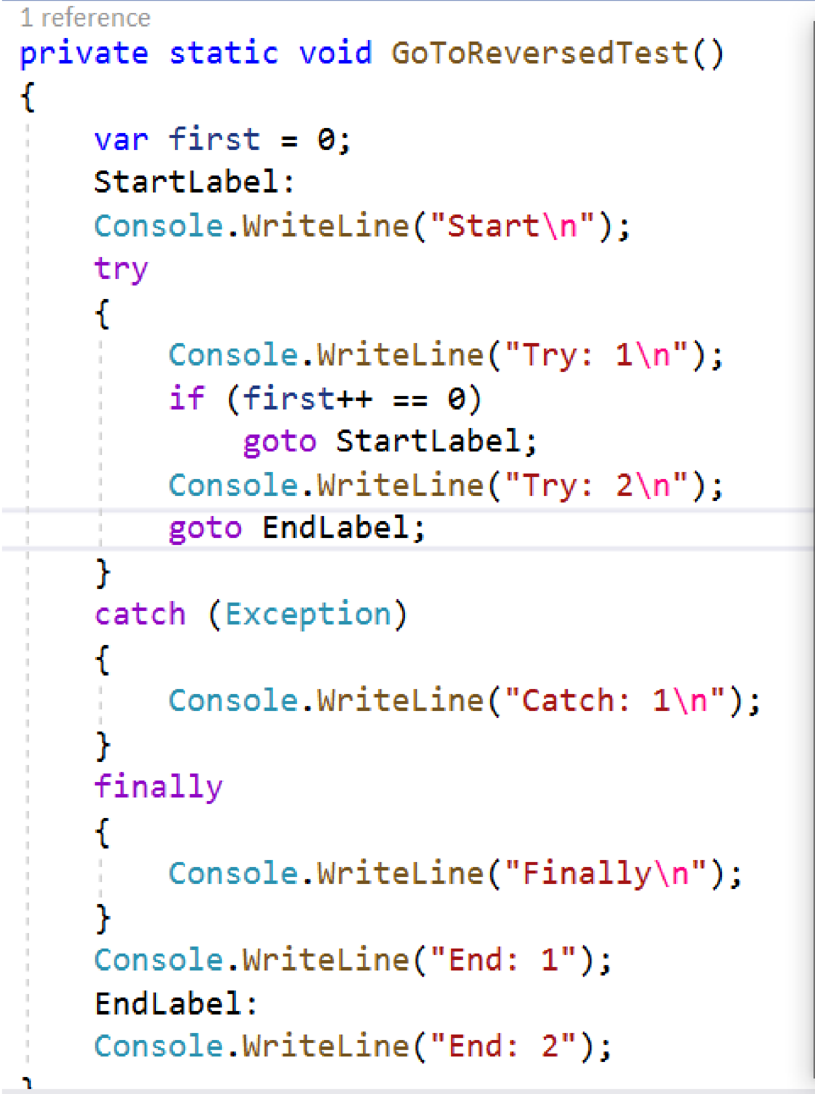
```
1 reference
private static void GoToTest()
{
    Console.WriteLine("Start\n");
    try
    {
        Console.WriteLine("Try: 1\n");
        goto EndLabel;
        Console.WriteLine("Try: 2\n");
    }
    catch (Exception)
    {
        Console.WriteLine("Catch: 1\n");
        throw new Exception("Exception from catch!");
    }
    finally
    {
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End: 1\n");
    EndLabel:
    Console.WriteLine("End: 2\n");
}
```



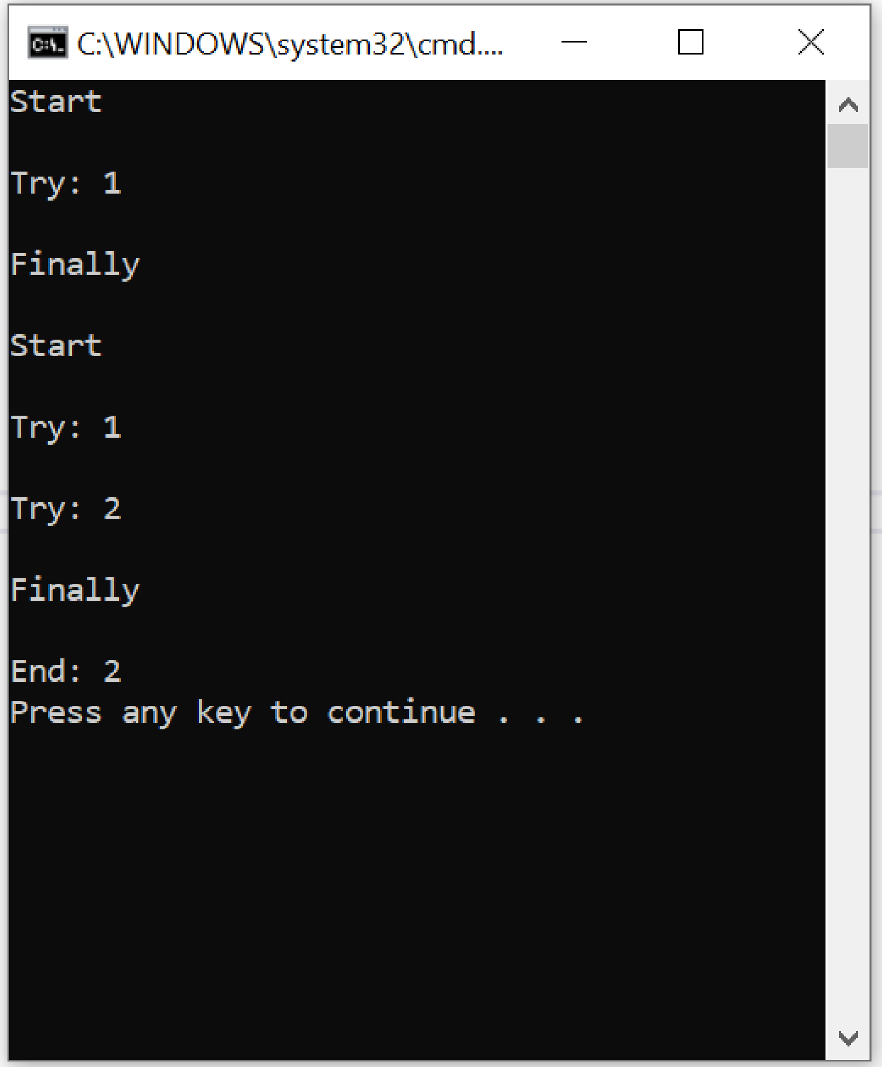
C:\WINDOWS\system32\cmd.e...

```
Start

Try: 1

Finally

End: 2

Press any key to continue . . .
```

GoTo reversed

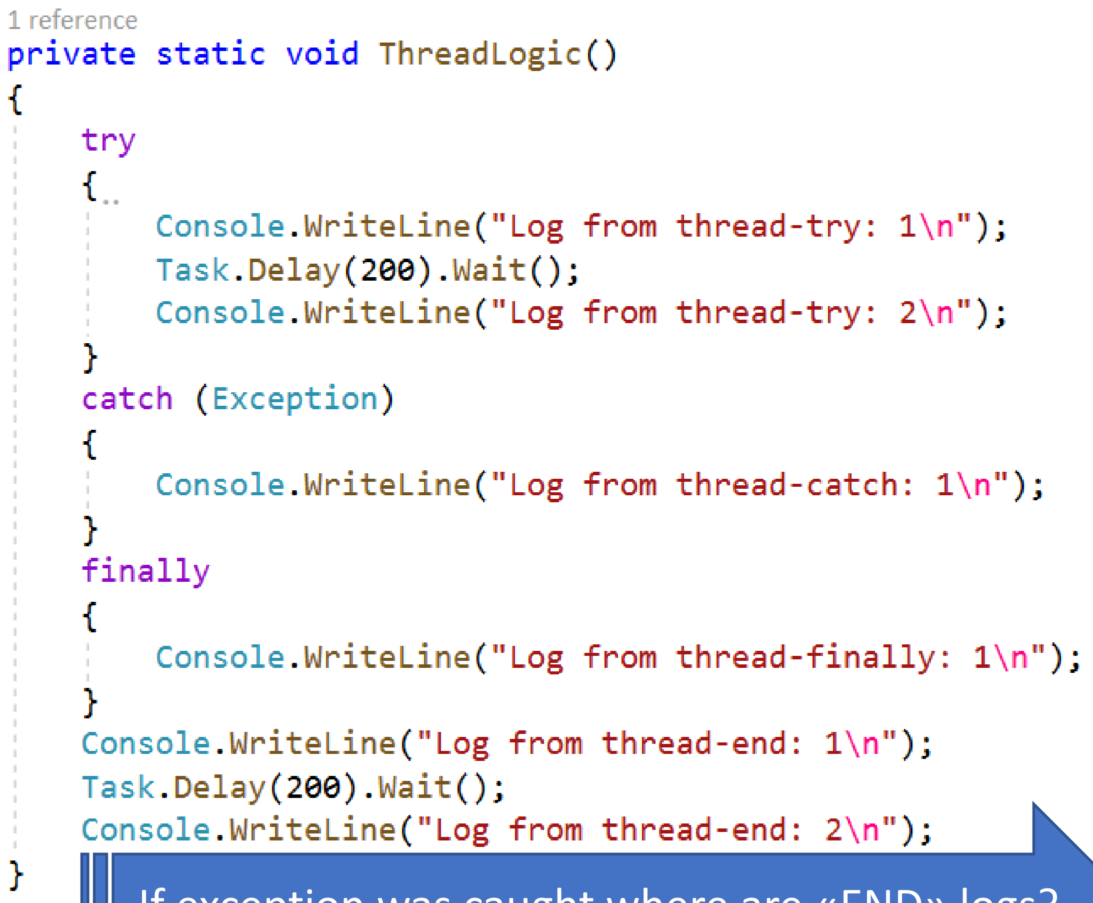


```
1 reference
private static void GoToReversedTest()
{
    var first = 0;
    StartLabel:
    Console.WriteLine("Start\n");
    try
    {
        Console.WriteLine("Try: 1\n");
        if (first++ == 0)
            goto StartLabel;
        Console.WriteLine("Try: 2\n");
        goto EndLabel;
    }
    catch (Exception)
    {
        Console.WriteLine("Catch: 1\n");
    }
    finally
    {
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End: 1");
    EndLabel:
    Console.WriteLine("End: 2");
}
```

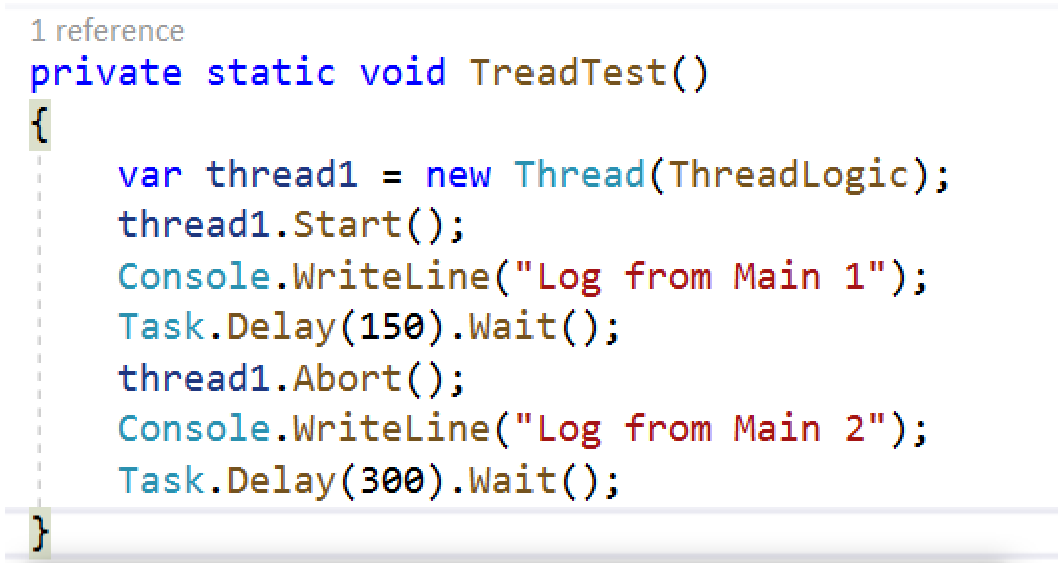


```
Start

Try: 1

Finally

Start

Try: 1

Try: 2

Finally

End: 2
Press any key to continue . . .
```

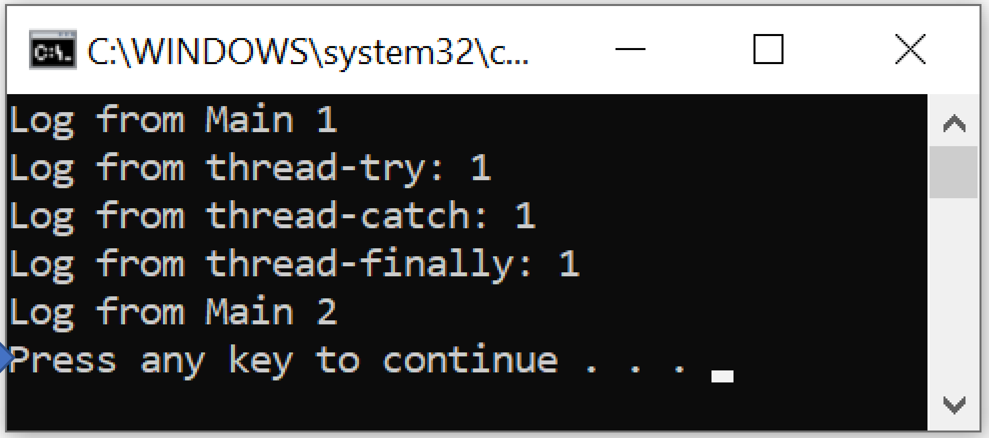
Kill another thread?



```
1 reference
private static void ThreadLogic()
{
    try
    {
        Console.WriteLine("Log from thread-try: 1\n");
        Task.Delay(200).Wait();
        Console.WriteLine("Log from thread-try: 2\n");
    }
    catch (Exception)
    {
        Console.WriteLine("Log from thread-catch: 1\n");
    }
    finally
    {
        Console.WriteLine("Log from thread-finally: 1\n");
    }
    Console.WriteLine("Log from thread-end: 1\n");
    Task.Delay(200).Wait();
    Console.WriteLine("Log from thread-end: 2\n");
}
```



```
1 reference
private static void TreadTest()
{
    var thread1 = new Thread(ThreadLogic);
    thread1.Start();
    Console.WriteLine("Log from Main 1");
    Task.Delay(150).Wait();
    thread1.Abort();
    Console.WriteLine("Log from Main 2");
    Task.Delay(300).Wait();
}
```



C:\WINDOWS\system32\c...

```
Log from Main 1
Log from thread-try: 1
Log from thread-catch: 1
Log from thread-finally: 1
Log from Main 2
Press any key to continue . . .
```

If exception was caught where are «END» logs?

ThreadAbortException

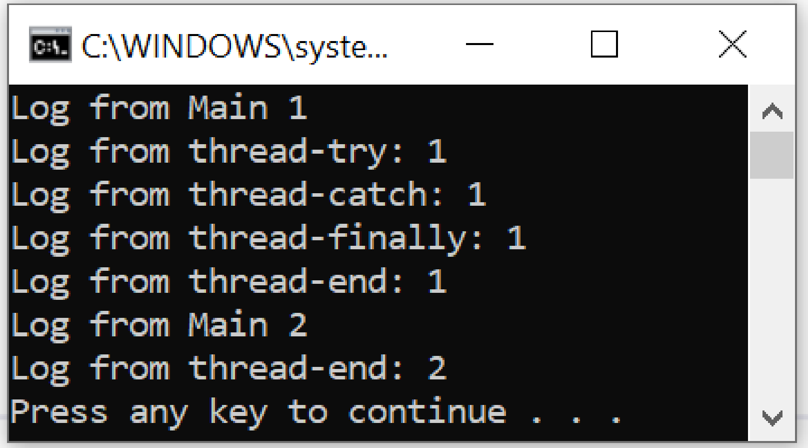
- ***ThreadAbortException*** is a special exception that can be caught, but it will automatically be raised again at the end of the catch block.
- When this exception is raised, the runtime executes all the ***finally*** blocks before ending the thread.
- ***Thread.ResetAbort*** to cancel the abort, there is no guarantee that the thread will ever end.

© MSDN

Kill another thread? Part 2.

```csharp
private static void ThreadLogic()
{
    try
    {
        Console.WriteLine("Log from thread-try: 1");
        Task.Delay(200).Wait();
        Console.WriteLine("Log from thread-try: 2");
    }
    catch (ThreadAbortException)
    {
        Console.WriteLine("Log from thread-catch: 1");
        Thread.ResetAbort();
    }
    finally
    {
        Console.WriteLine("Log from thread-finally: 1");
    }
    Console.WriteLine("Log from thread-end: 1");
    Task.Delay(200).Wait();
    Console.WriteLine("Log from thread-end: 2");
}
```

```csharp
1 reference
private static void TreadTest()
{
    var thread1 = new Thread(ThreadLogic);
    thread1.Start();
    Console.WriteLine("Log from Main 1");
    Task.Delay(150).Wait();
    thread1.Abort();
    Console.WriteLine("Log from Main 2");
    Task.Delay(300).Wait();
}
```



```
Log from Main 1
Log from thread-try: 1
Log from thread-catch: 1
Log from thread-finally: 1
Log from thread-end: 1
Log from Main 2
Log from thread-end: 2
Press any key to continue . . .
```

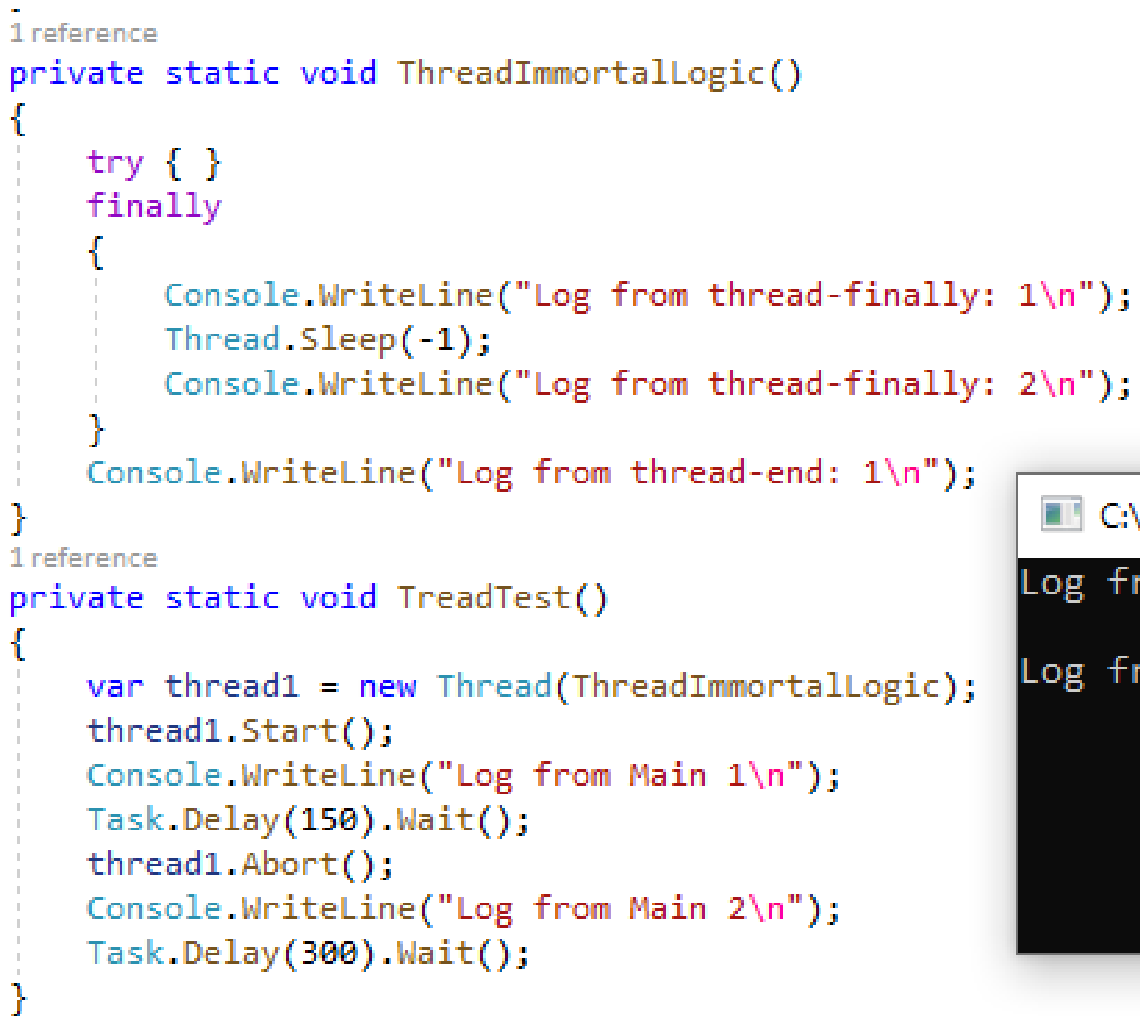
Why Thread.Abort is dangerous?

- If one thread calls Abort on another thread, the abort interrupts whatever code is running.
- The thread is not guaranteed to abort immediately, or at all.

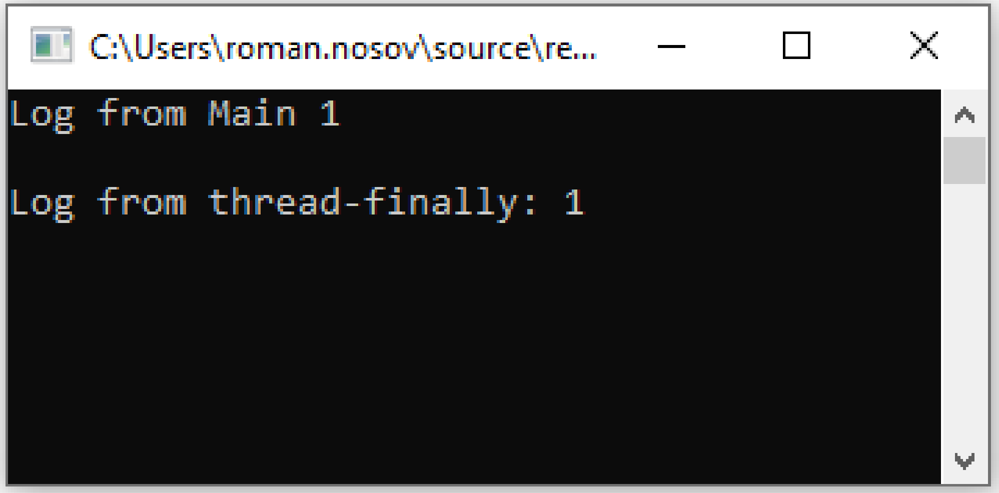
- .NET Core and .NET 5+ only: in all cases: PlatformNotSupportedException

© MSDN

Why Thread.Abort is dangerous? Part 2



```
1 reference
private static void ThreadImmortalLogic()
{
    try { }
    finally
    {
        Console.WriteLine("Log from thread-finally: 1\n");
        Thread.Sleep(-1);
        Console.WriteLine("Log from thread-finally: 2\n");
    }
    Console.WriteLine("Log from thread-end: 1\n");
}
1 reference
private static void TreadTest()
{
    var thread1 = new Thread(ThreadImmortalLogic);
    thread1.Start();
    Console.WriteLine("Log from Main 1\n");
    Task.Delay(150).Wait();
    thread1.Abort();
    Console.WriteLine("Log from Main 2\n");
    Task.Delay(300).Wait();
}
```



C:\Users\roman.nosov\source\re...

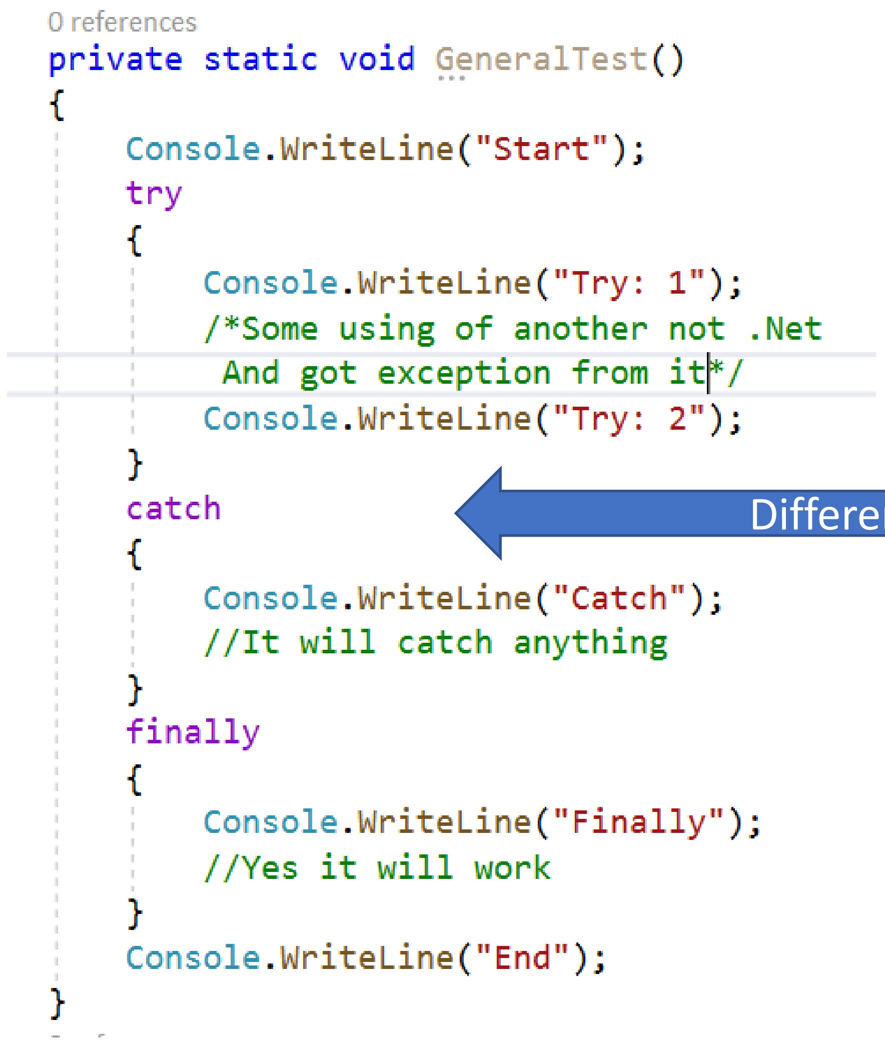
```
Log from Main 1

Log from thread-finally: 1
```

Exceptions that are not child of Exception?

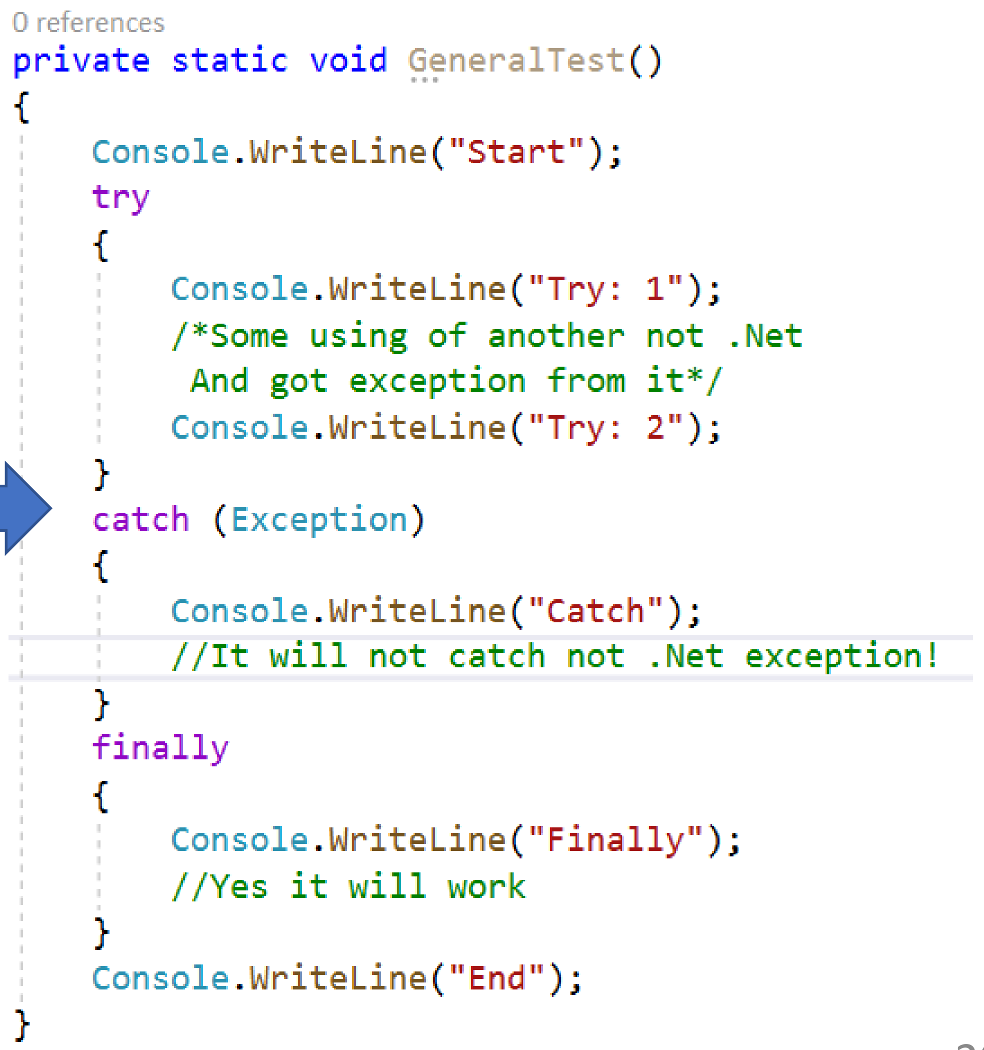
- A catch clause that does not specify an exception_specifier is called a **general catch** clause.
- Some programming languages may support exceptions that are not representable as an object derived from **System.Exception**, although such exceptions could never be generated by C# code. A general catch clause may be used to catch such exceptions. Thus, a general catch clause is semantically different from one that specifies the type **System.Exception**, in that the former may also catch exceptions from other languages.

© MSDN

It’s not the same!


```
0 references
private static void GeneralTest()
{
    Console.WriteLine("Start");
    try
    {
        Console.WriteLine("Try: 1");
        /*Some using of another not .Net
         And got exception from it*/
        Console.WriteLine("Try: 2");
    }
    catch
    {
        Console.WriteLine("Catch");
        //It will catch anything
    }
    finally
    {
        Console.WriteLine("Finally");
        //Yes it will work
    }
    Console.WriteLine("End");
}
```

Difference


```
0 references
private static void GeneralTest()
{
    Console.WriteLine("Start");
    try
    {
        Console.WriteLine("Try: 1");
        /*Some using of another not .Net
         And got exception from it*/
        Console.WriteLine("Try: 2");
    }
    catch (Exception)
    {
        Console.WriteLine("Catch");
        //It will not catch not .Net exception!
    }
    finally
    {
        Console.WriteLine("Finally");
        //Yes it will work
    }
    Console.WriteLine("End");
}
```

Неординарные задачи требуют неординарных решений

I. Немного теории
II. Немного практики
III. Уроним try-catch-finally?
IV. Уроним try-catch-finally? (А теперь серьезно)
V. Заключение

.NET Fremawork ***VS*** .NET Core *****

***** Где есть раззличие

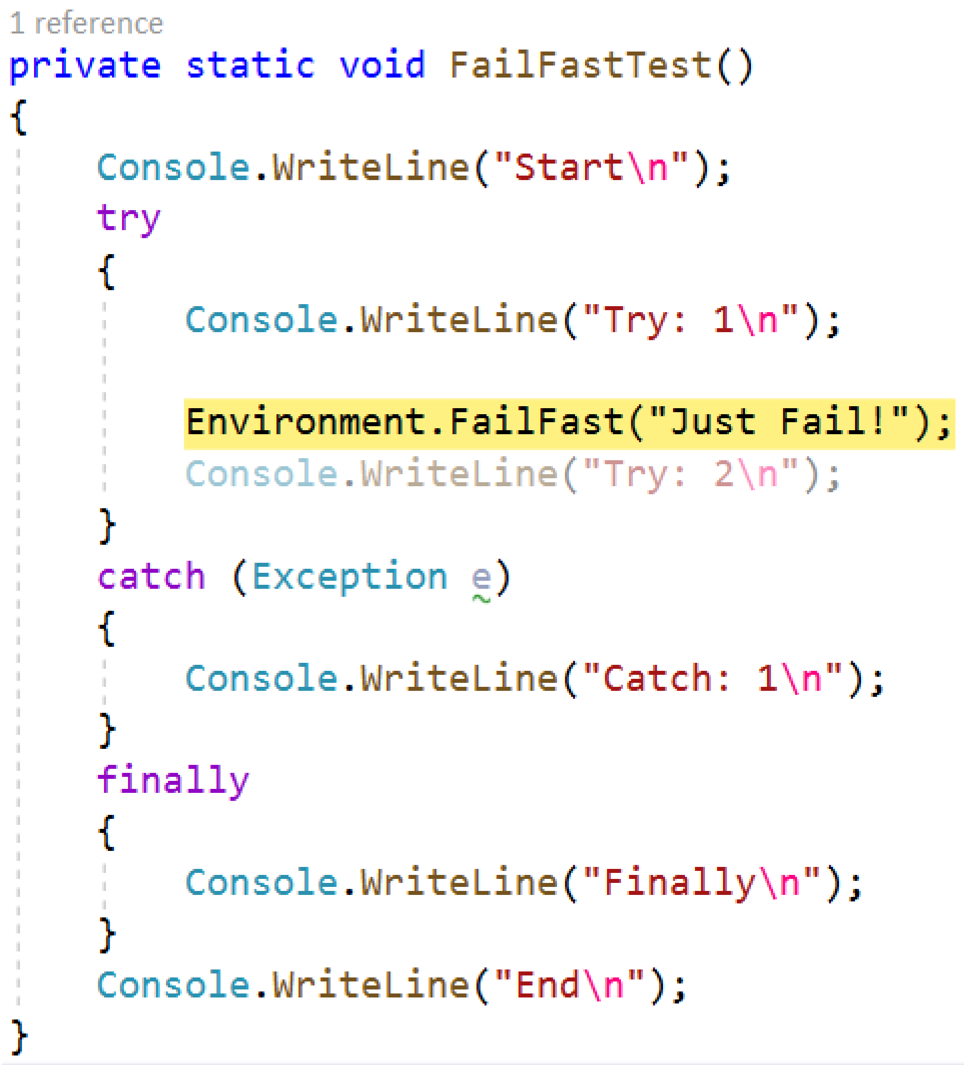
I. Just switch computer off

Reason:
- Process was eliminated
- Server shut down
- Docker shut down

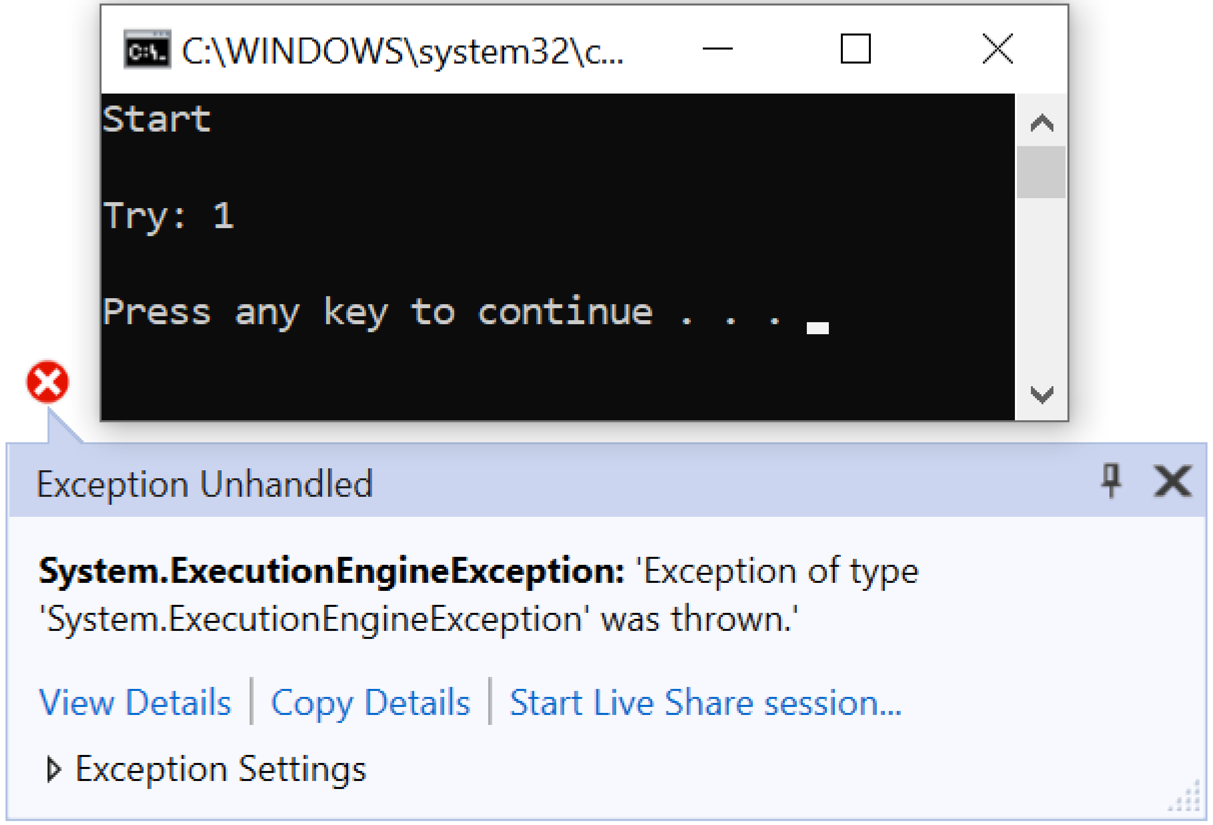
Places:
- Microservices
- Phone

Expectation:
- Inform another microservice
- Send something via queue
- Close connections/transactions

II. FailFast



```
1 reference
private static void FailFastTest()
{
    Console.WriteLine("Start\n");
    try
    {
        Console.WriteLine("Try: 1\n");

        Environment.FailFast("Just Fail!");
        Console.WriteLine("Try: 2\n");
    }
    catch (Exception e)
    {
        Console.WriteLine("Catch: 1\n");
    }
    finally
    {
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End\n");
}
```



C:\WINDOWS\system32\c...

```
Start

Try: 1

Press any key to continue . . .
```

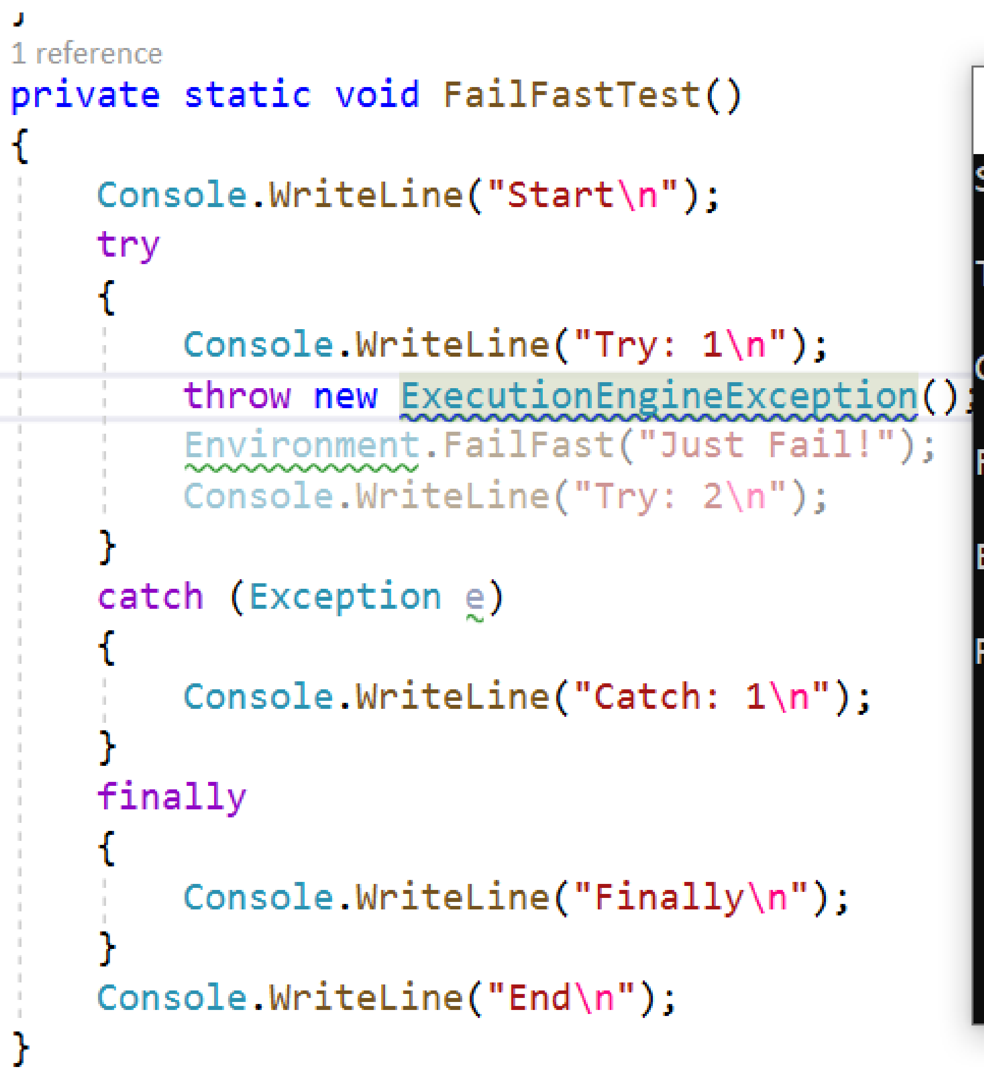
Exception Unhandled

System.ExecutionEngineException: 'Exception of type 'System.ExecutionEngineException' was thrown.'

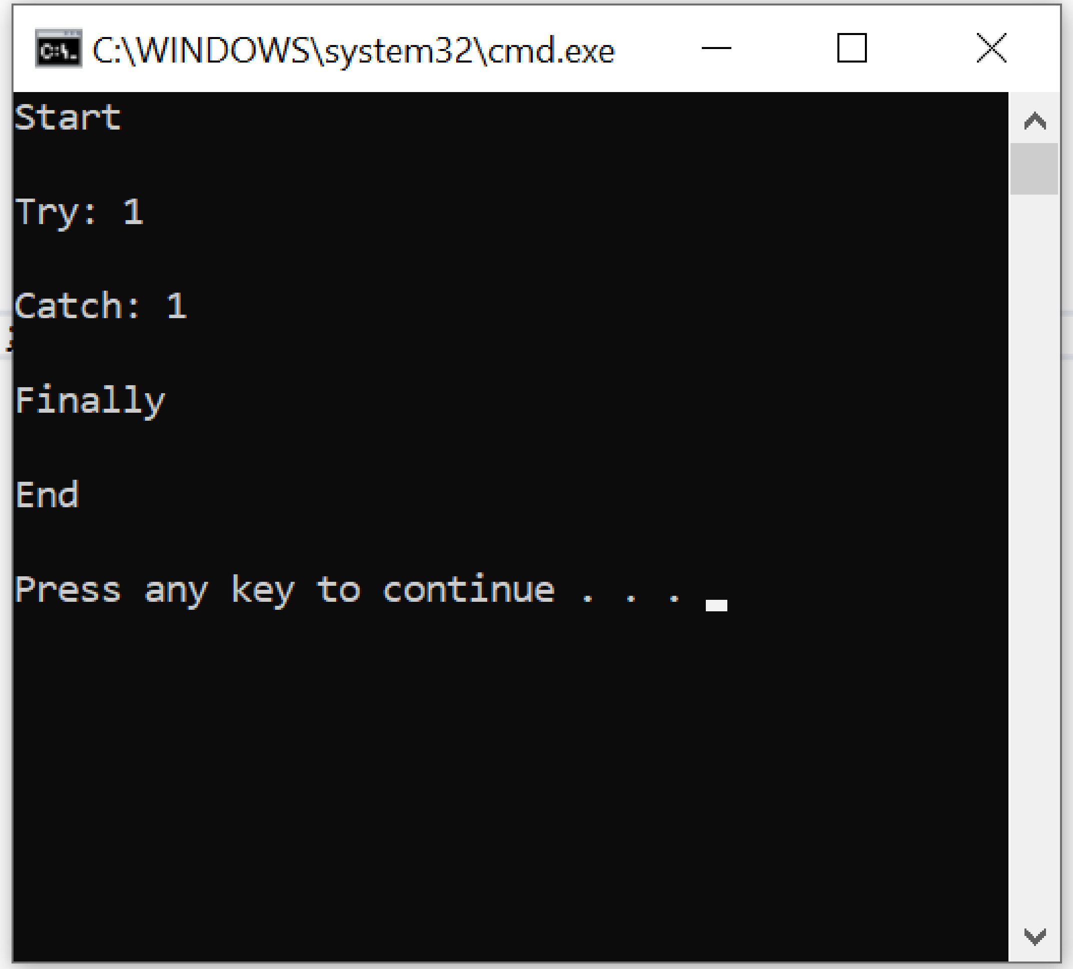
View Details | Copy Details | Start Live Share session...

Exception Settings

II. FailFast. Part 2. ExecutionEngineException



```
1 reference
private static void FailFastTest()
{
    Console.WriteLine("Start\n");
    try
    {
        Console.WriteLine("Try: 1\n");
        throw new ExecutionEngineException();
        Environment.FailFast("Just Fail!");
        Console.WriteLine("Try: 2\n");
    }
    catch (Exception e)
    {
        Console.WriteLine("Catch: 1\n");
    }
    finally
    {
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End\n");
}
```



C:\WINDOWS\system32\cmd.exe

```
Start

Try: 1

Catch: 1

Finally

End

Press any key to continue . . .
```

III. Corrupted State Exception

- It’s part of SEH (Structured Exception Handling)
- And can be handled by CLR
- catch and finally do not work

CLR has stopped handling it since 4.0

[HandleProcessCorruptedStateExceptions]

- Something wrong with Windows
- Something wrong with CLR
- Something wrong because of unsafe code

III. Are there any differences?

legacyCorruptedStateExceptionsPolicy=true
[HandleProcessCorruptedStateExceptions]

Even though this attribute exists in .NET Core, since the recovery from corrupted process state exceptions is not supported, this attribute is ignored. The CLR doesn't deliver corrupted process state exceptions to the managed code.

© MSDN

IV. InvalidProgramException

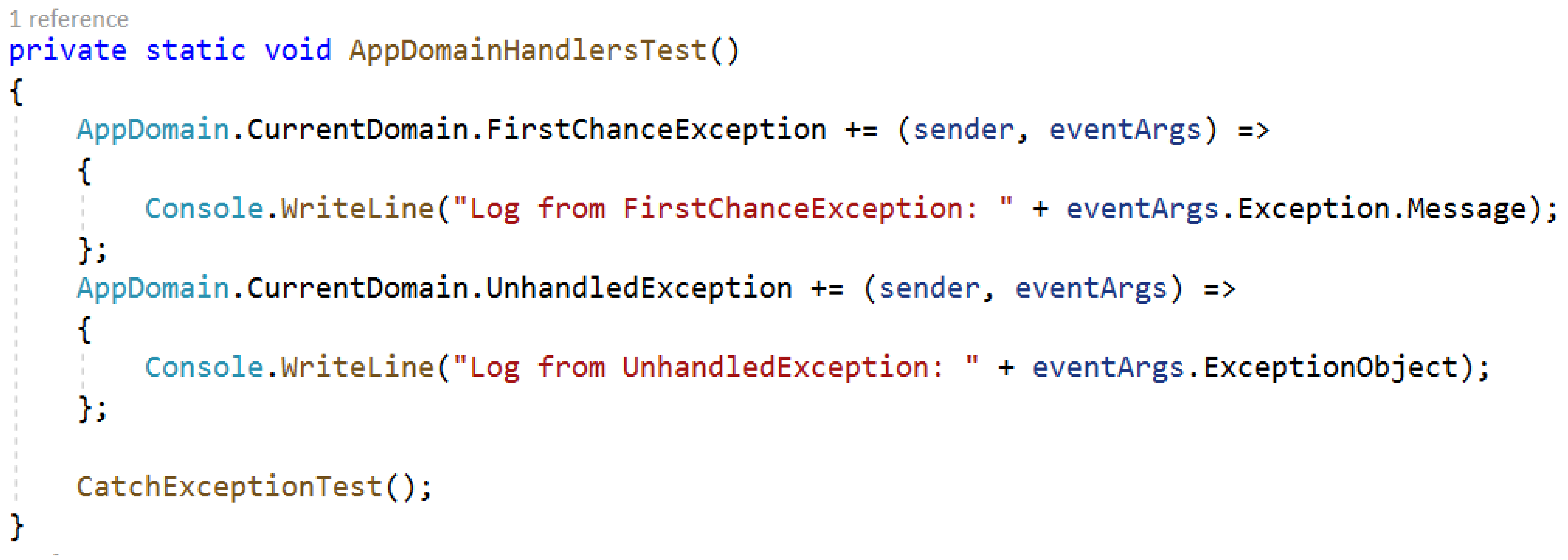
The exception that is thrown when a program contains invalid Microsoft intermediate language (MSIL) or metadata. Generally, this indicates a bug in the compiler that generated the program.

© MSDN

IV. InvalidProgramException. Part 2

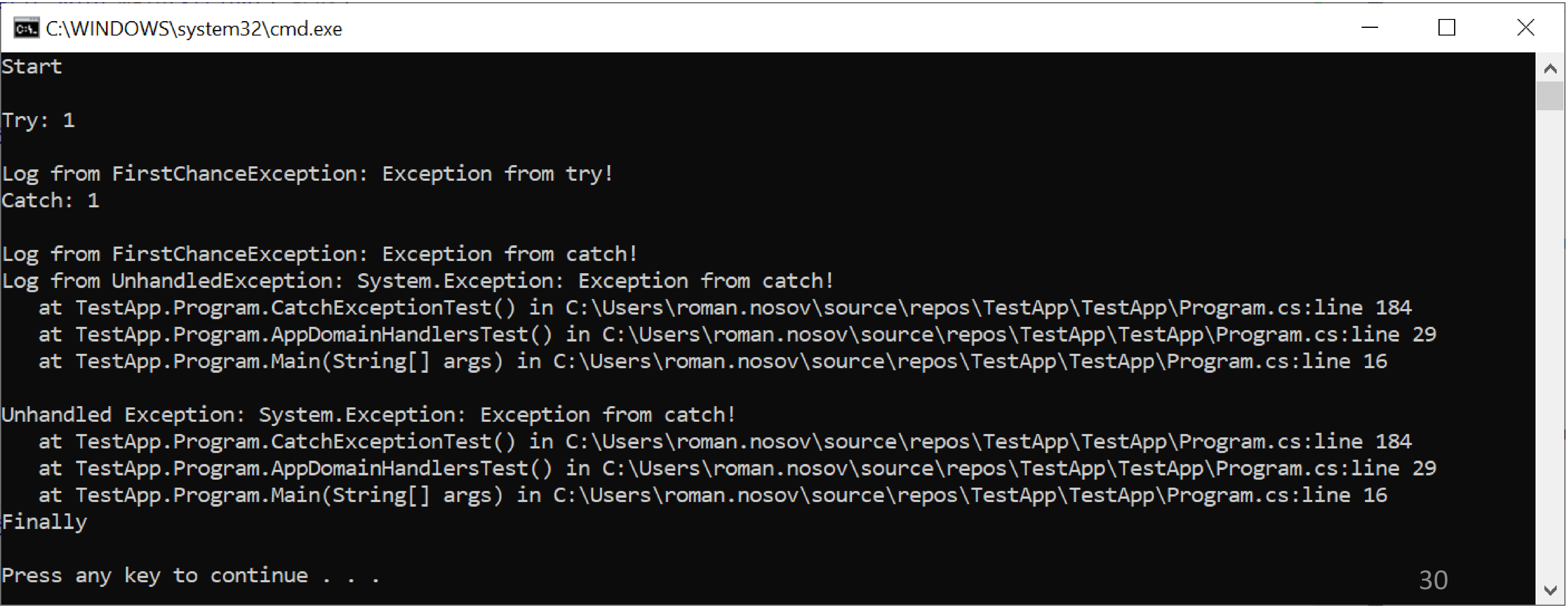
ILGenerator is used to generate method bodies for methods and constructors in dynamic assemblies (represented by the MethodBuilder and ConstructorBuilder classes) and for standalone dynamic methods (represented by the DynamicMethod class).

V. App domain exception subscriptions



1 reference
```
private static void AppDomainHandlersTest()
{
    AppDomain.CurrentDomain.FirstChanceException += (sender, eventArgs) =>
    {
        Console.WriteLine("Log from FirstChanceException: " + eventArgs.Exception.Message);
    };
    AppDomain.CurrentDomain.UnhandledException += (sender, eventArgs) =>
    {
        Console.WriteLine("Log from UnhandledException: " + eventArgs.ExceptionObject);
    };

    CatchExceptionTest();
}
```

V. FirstChanceException

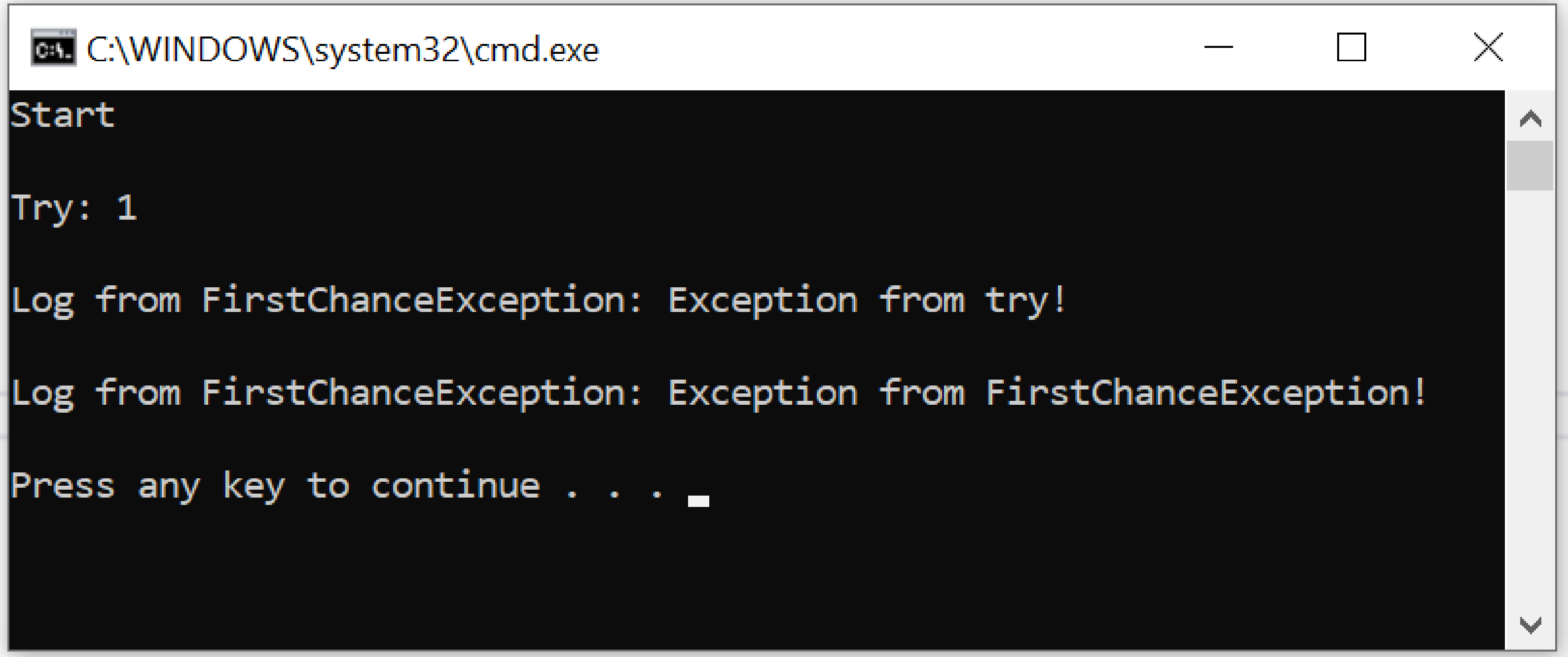


C:\WINDOWS\system32\cmd.exe

```
Start

Try: 1

Log from FirstChanceException: Exception from try!
Catch: 1

Log from FirstChanceException: Exception from catch!
Log from UnhandledException: System.Exception: Exception from catch!
   at TestApp.Program.CatchExceptionTest() in C:\Users\roman.nosov\source\repos\TestApp\TestApp\Program.cs:line 184
   at TestApp.Program.AppDomainHandlersTest() in C:\Users\roman.nosov\source\repos\TestApp\TestApp\Program.cs:line 29
   at TestApp.Program.Main(String[] args) in C:\Users\roman.nosov\source\repos\TestApp\TestApp\Program.cs:line 16

Unhandled Exception: System.Exception: Exception from catch!
   at TestApp.Program.CatchExceptionTest() in C:\Users\roman.nosov\source\repos\TestApp\TestApp\Program.cs:line 184
   at TestApp.Program.AppDomainHandlersTest() in C:\Users\roman.nosov\source\repos\TestApp\TestApp\Program.cs:line 29
   at TestApp.Program.Main(String[] args) in C:\Users\roman.nosov\source\repos\TestApp\TestApp\Program.cs:line 16
Finally

Press any key to continue . . .
```

V. FirstChanceException. Part 2

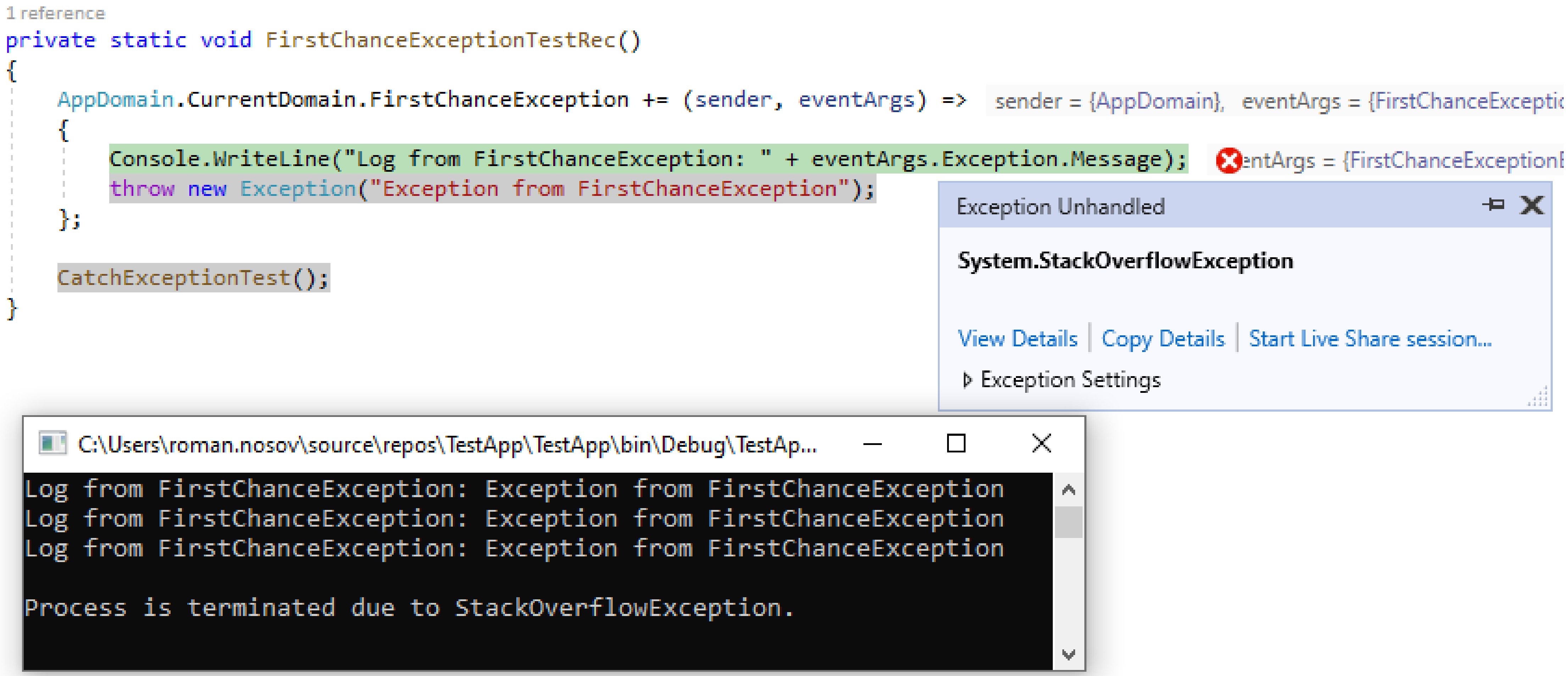
```
1 reference
private static void FirstChanceExceptionTest()
{
    AppDomain.CurrentDomain.FirstChanceException += (sender, eventArgs) =>
    {
        Console.WriteLine($"Log from FirstChanceException: {eventArgs.Exception.Message}\n");
        if (eventArgs.Exception.Message == "Exception from try!")
            throw new Exception("Exception from FirstChanceException!");
    };

    SimpleTest();
}
```



C:\WINDOWS\system32\cmd.exe

```
Start

Try: 1

Log from FirstChanceException: Exception from try!

Log from FirstChanceException: Exception from FirstChanceException!

Press any key to continue . . .
```

V. FirstChanceException. Recursion



```
1 reference
private static void FirstChanceExceptionTestRec()
{
    AppDomain.CurrentDomain.FirstChanceException += (sender, eventArgs) =>   sender = {AppDomain},  eventArgs = {FirstChanceExceptic
    {
        Console.WriteLine("Log from FirstChanceException: " + eventArgs.Exception.Message);   entArgs = {FirstChanceExceptionE
        throw new Exception("Exception from FirstChanceException");
    };

    CatchExceptionTest();
}
```

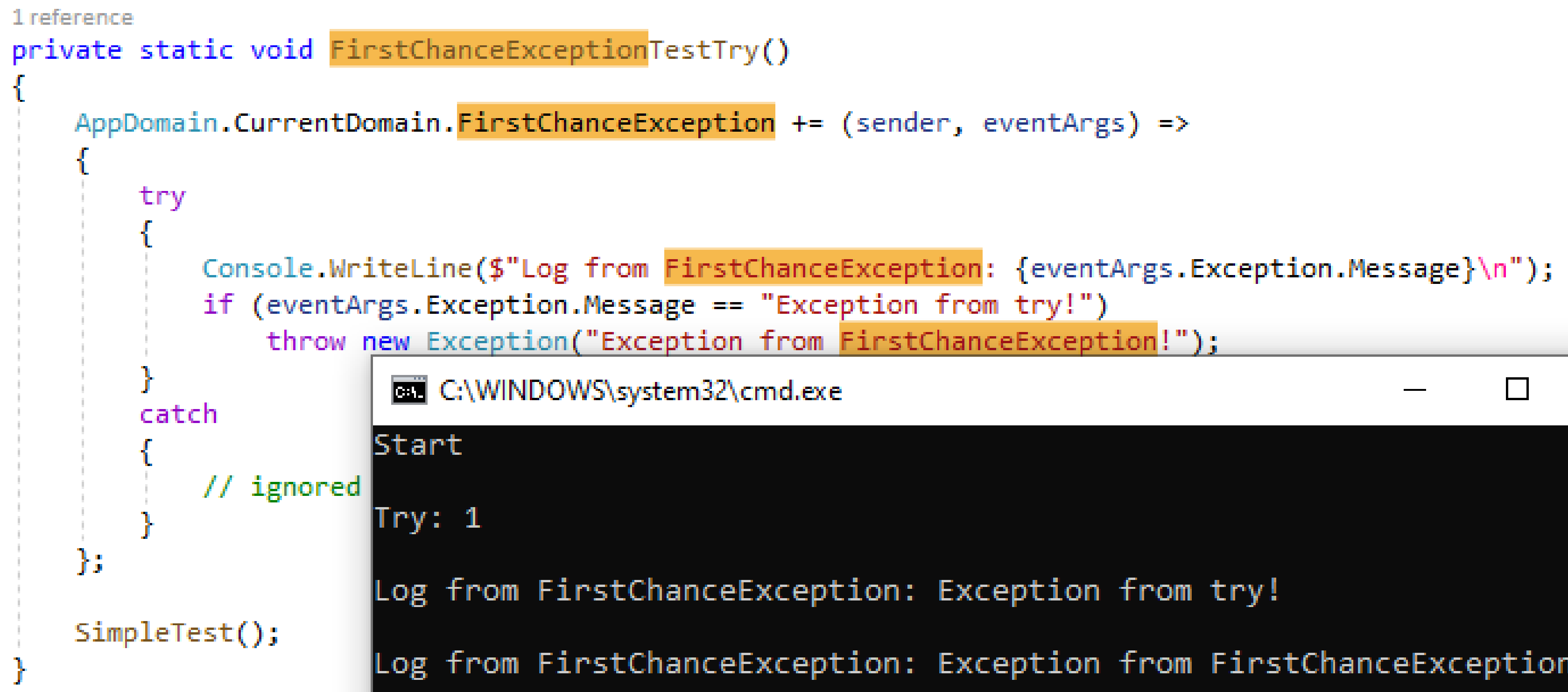
Exception Unhandled

System.StackOverflowException

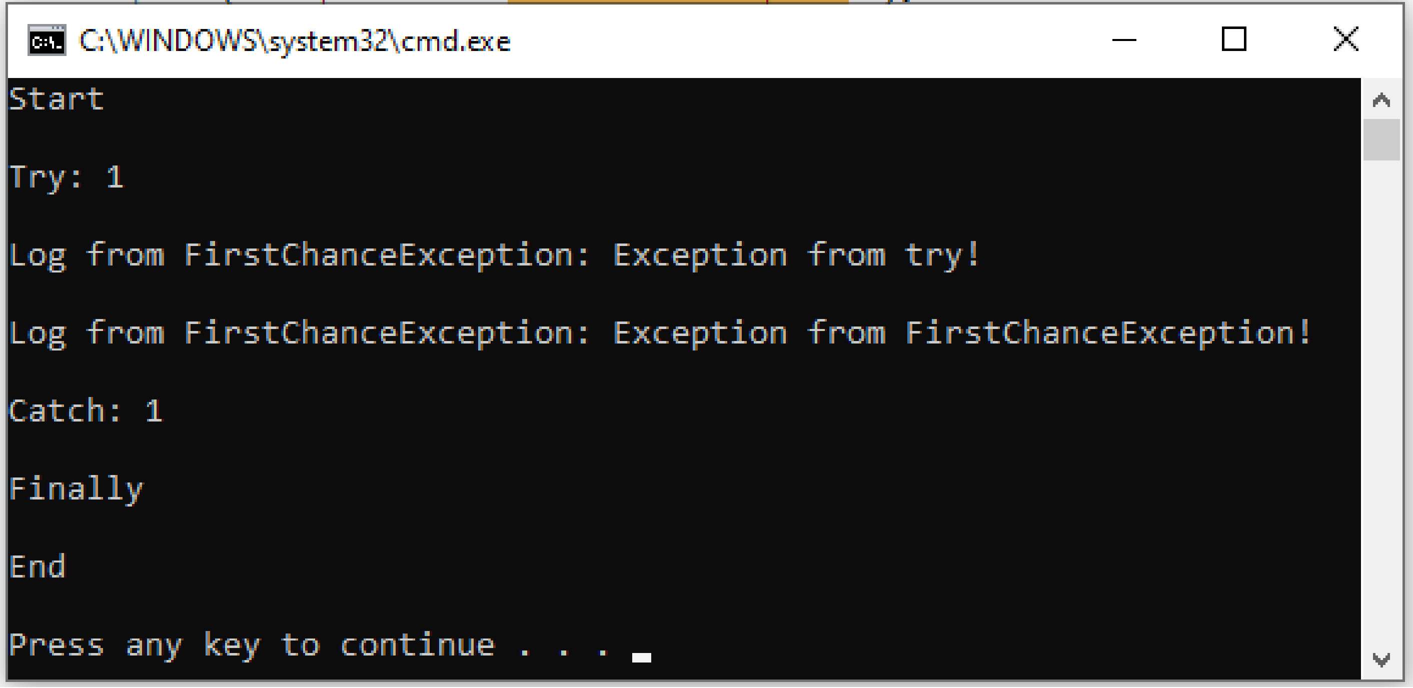
View Details | Copy Details | Start Live Share session...
Exception Settings
C:\Users\roman.nosov\source\repos\TestApp\TestApp\bin\Debug\TestAp...

```
Log from FirstChanceException: Exception from FirstChanceException
Log from FirstChanceException: Exception from FirstChanceException
Log from FirstChanceException: Exception from FirstChanceException

Process is terminated due to StackOverflowException.
```

V. FirstChanceException. Part 3



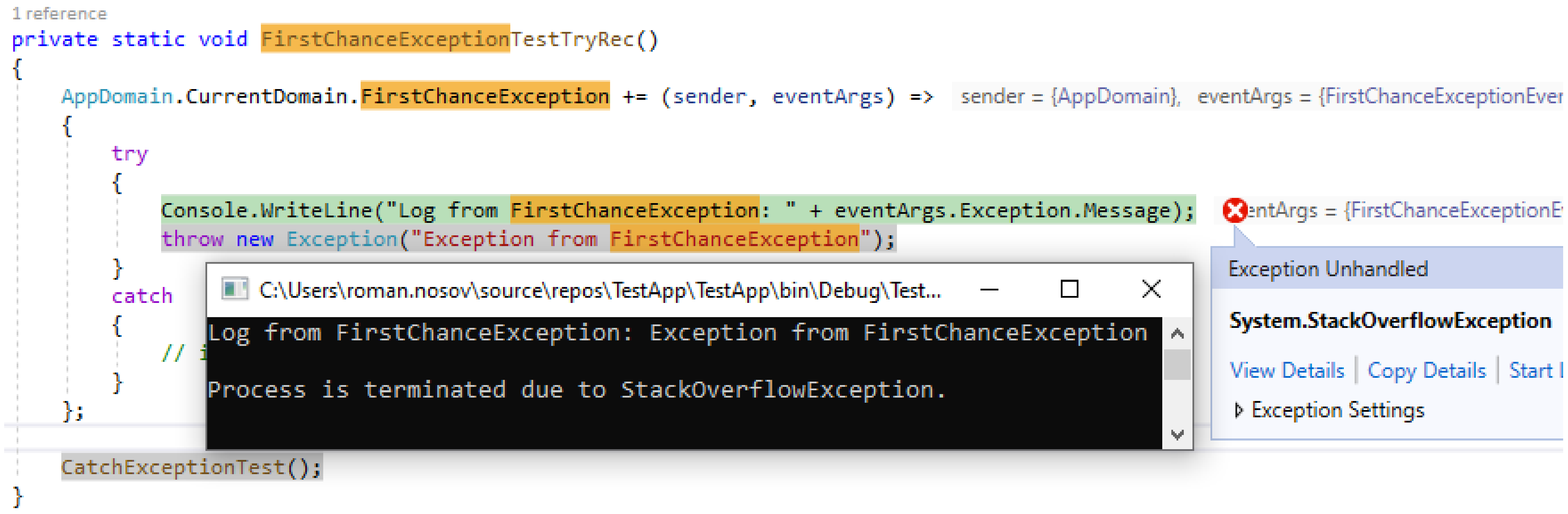
```
1 reference
private static void FirstChanceExceptionTestTry()
{
    AppDomain.CurrentDomain.FirstChanceException += (sender, eventArgs) =>
    {
        try
        {
            Console.WriteLine($"Log from FirstChanceException: {eventArgs.Exception.Message}\n");
            if (eventArgs.Exception.Message == "Exception from try!")
                throw new Exception("Exception from FirstChanceException!");
        }
        catch
        {
            // ignored
        }
    };

    SimpleTest();
}
```



C:\WINDOWS\system32\cmd.exe

```
Start

Try: 1

Log from FirstChanceException: Exception from try!

Log from FirstChanceException: Exception from FirstChanceException!

Catch: 1

Finally

End

Press any key to continue . . .
```

V. FirstChanceException. Recursion. Part 2



```
1 reference
private static void FirstChanceExceptionTestTryRec()
{
    AppDomain.CurrentDomain.FirstChanceException += (sender, eventArgs) =>   sender = {AppDomain},   eventArgs = {FirstChanceExceptionEver
    {
        try
        {
            Console.WriteLine("Log from FirstChanceException: " + eventArgs.Exception.Message);   entArgs = {FirstChanceExceptionE
            throw new Exception("Exception from FirstChanceException");
        }
        catch
        {
            // i
        }
    };

    CatchExceptionTest();
}
```

C:\Users\roman.nosov\source\repos\TestApp\TestApp\bin\Debug\Test...

```
Log from FirstChanceException: Exception from FirstChanceException

Process is terminated due to StackOverflowException.
```

Exception Unhandled

System.StackOverflowException

View Details | Copy Details | Start l

Exception Settings

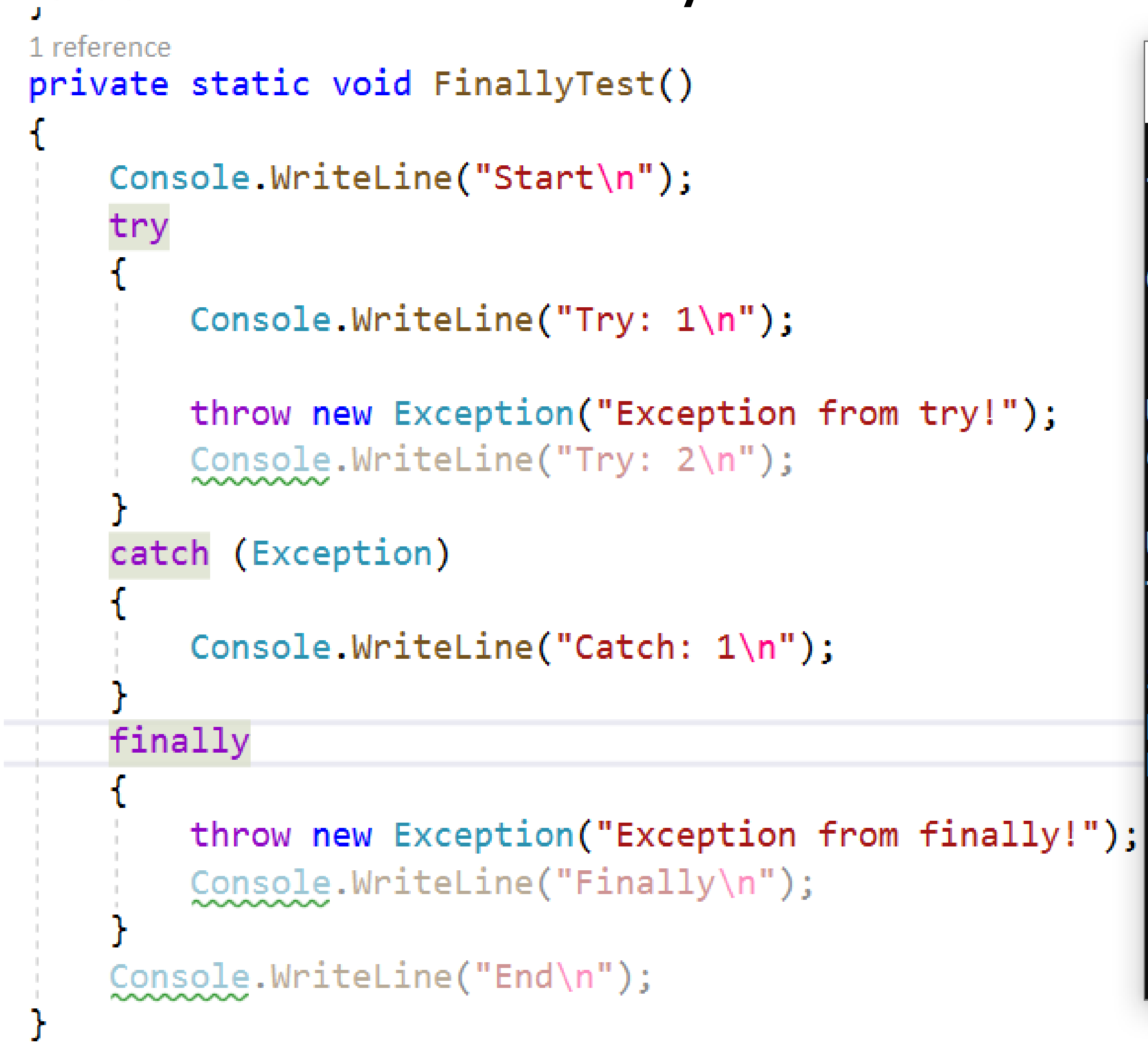
V. App domain.Difference?

AppDomain

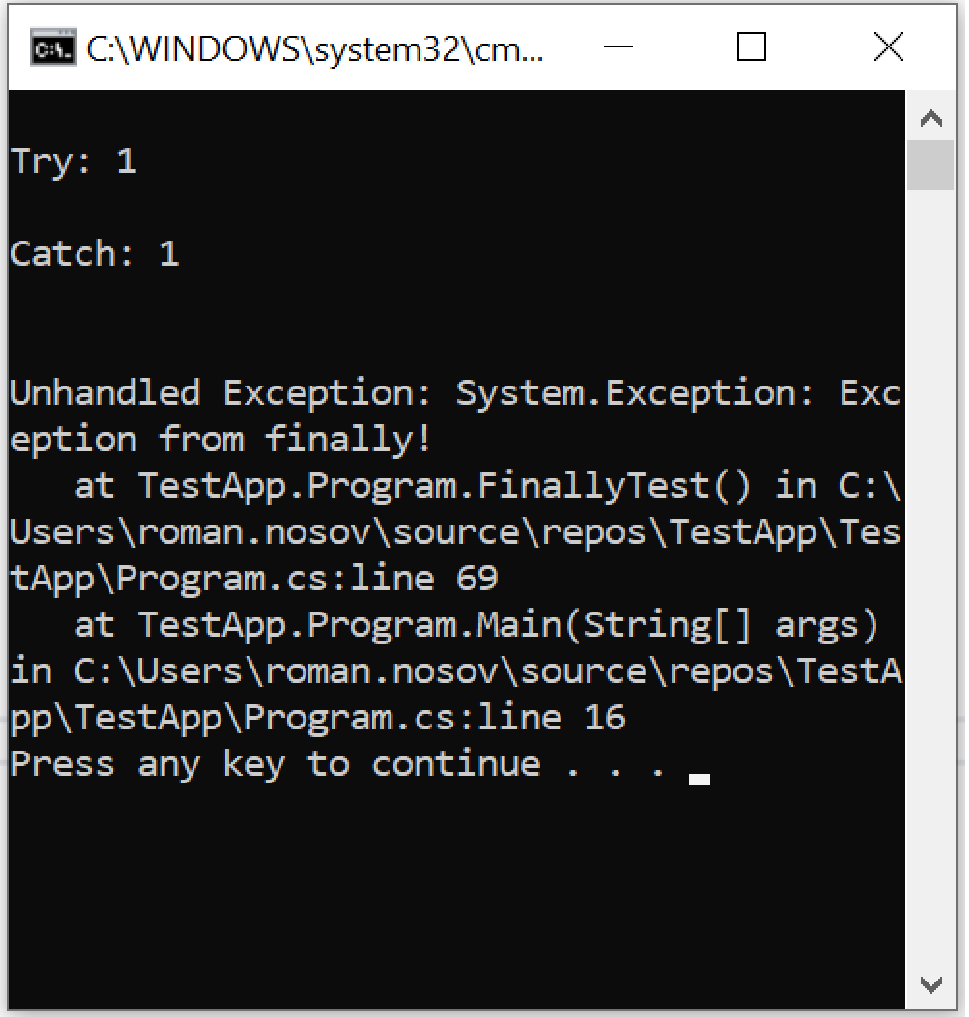
On .NET Core, the AppDomain implementation is limited by design and does not provide isolation, unloading, or security boundaries. For .NET Core, there is exactly one AppDomain.

© MSDN

VI. Could finally launch but not executed?

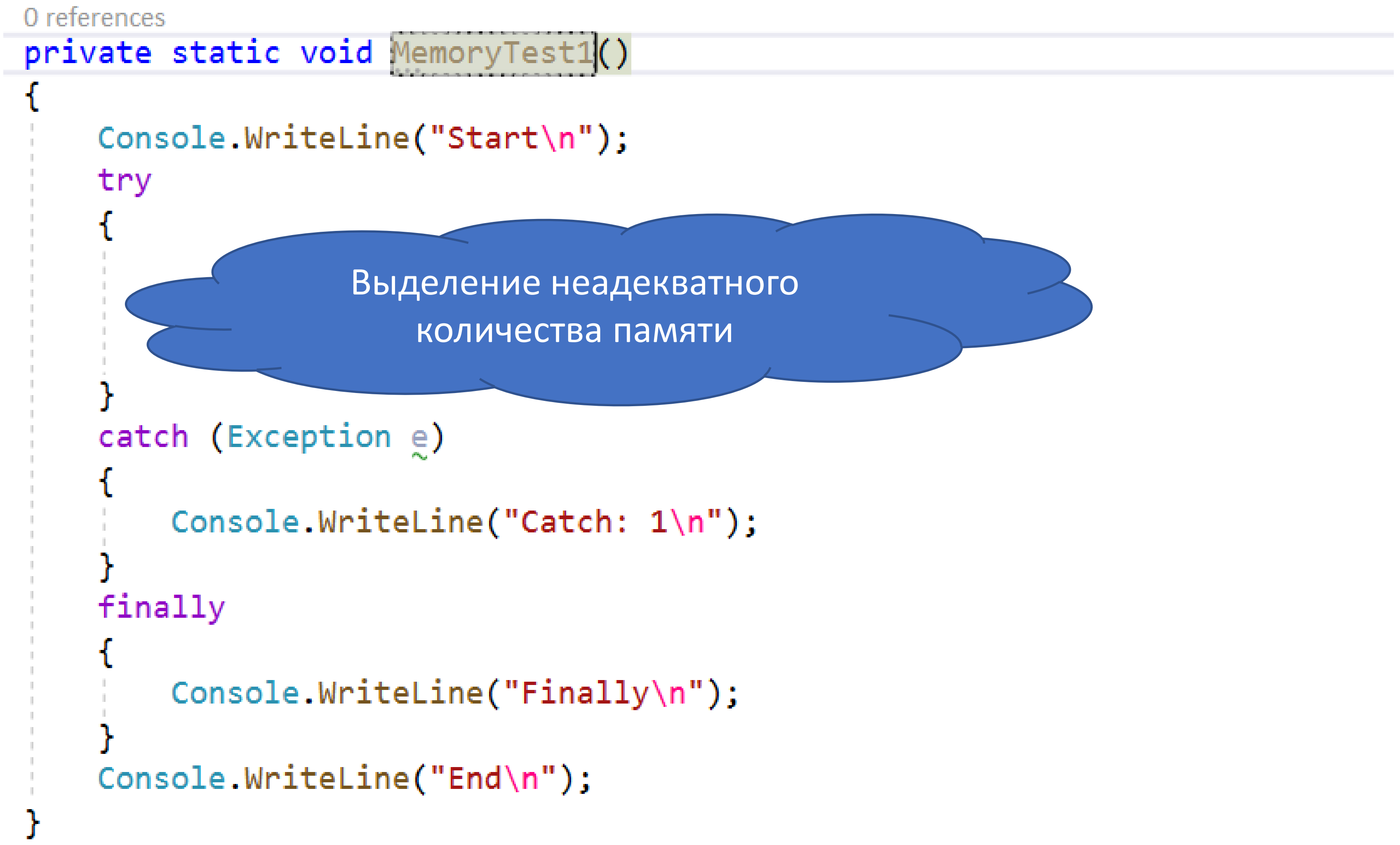


```
1 reference
private static void FinallyTest()
{
    Console.WriteLine("Start\n");
    try
    {
        Console.WriteLine("Try: 1\n");

        throw new Exception("Exception from try!");
        Console.WriteLine("Try: 2\n");
    }
    catch (Exception)
    {
        Console.WriteLine("Catch: 1\n");
    }
    finally
    {
        throw new Exception("Exception from finally!");
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End\n");
}
```



```
Try: 1

Catch: 1


Unhandled Exception: System.Exception: Exc
eption from finally!
   at TestApp.Program.FinallyTest() in C:\
Users\roman.nosov\source\repos\TestApp\Tes
tApp\Program.cs:line 69
   at TestApp.Program.Main(String[] args)
in C:\Users\roman.nosov\source\repos\TestA
pp\TestApp\Program.cs:line 16
Press any key to continue . . .
```

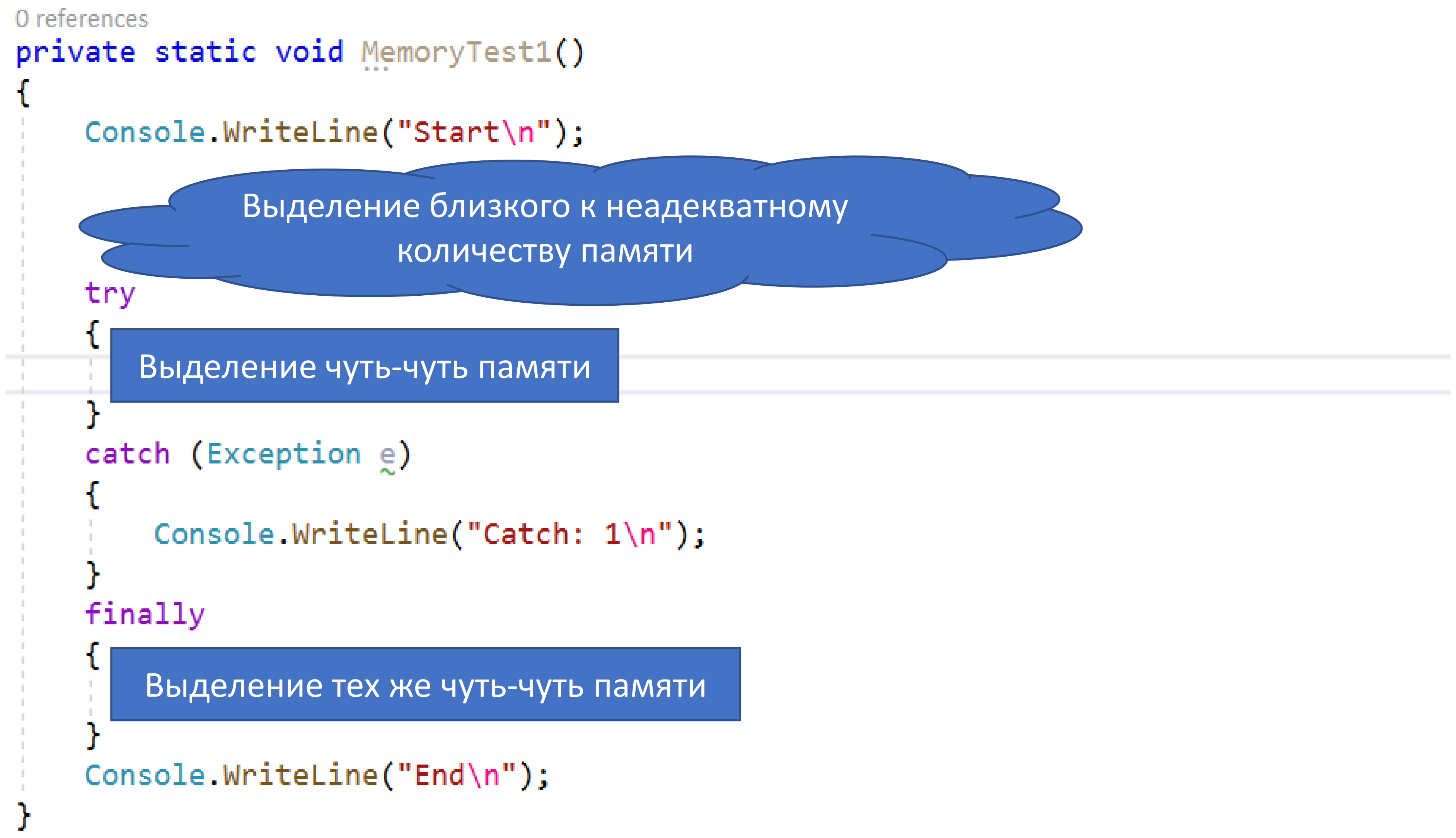
VI. OutOfMemoryException



```
0 references
private static void MemoryTest1()
{
    Console.WriteLine("Start\n");
    try
    {



    }
    catch (Exception e)
    {
        Console.WriteLine("Catch: 1\n");
    }
    finally
    {
        Console.WriteLine("Finally\n");
    }
    Console.WriteLine("End\n");
}
```

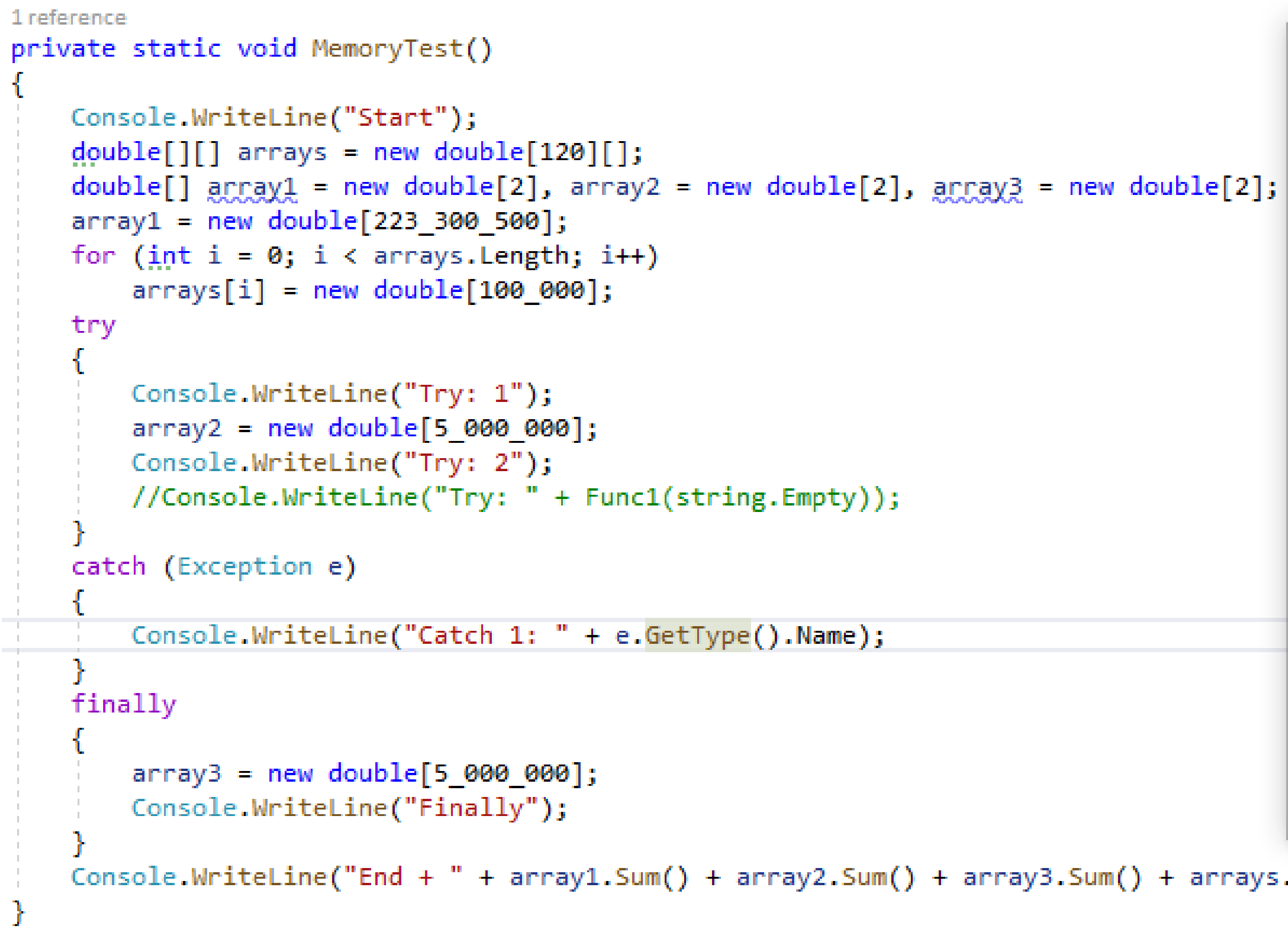
Выделение неадекватного количества памяти

VI. OutOfMemoryException. Part 2

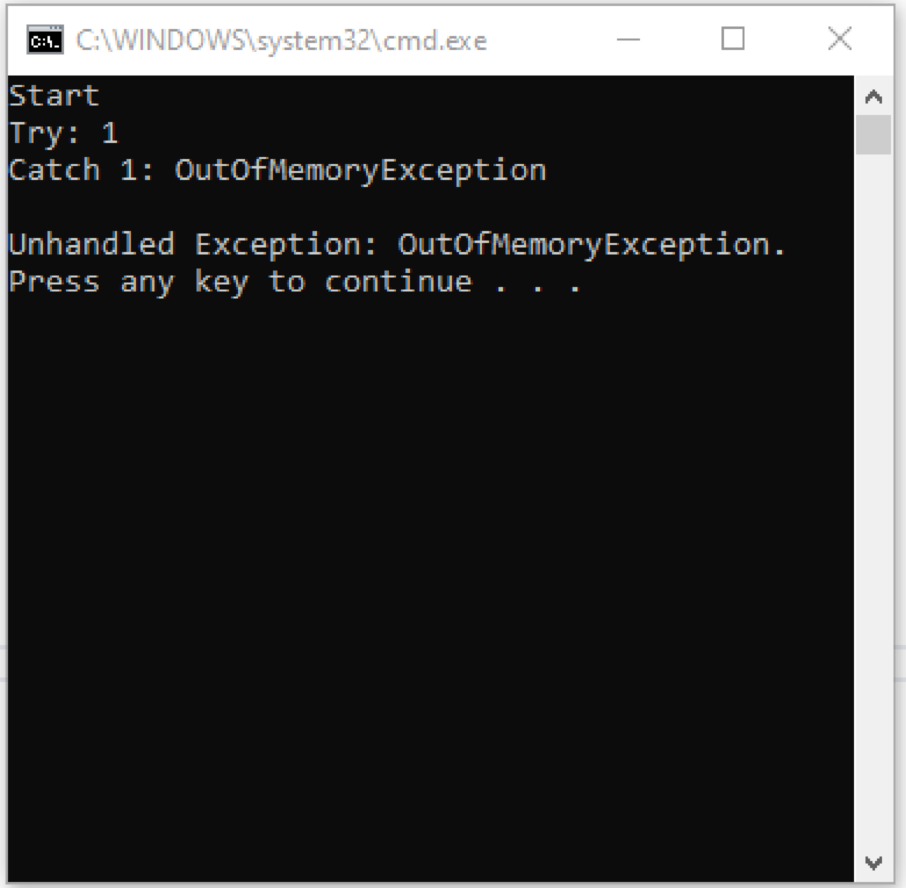


```
0 references
private static void MemoryTest1()
{
    Console.WriteLine("Start\n");

      Выделение близкого к неадекватному
             количеству памяти

    try
    {
      Выделение чуть-чуть памяти
    }
    catch (Exception e)
    {
        Console.WriteLine("Catch: 1\n");
    }
    finally
    {
      Выделение тех же чуть-чуть памяти
    }
    Console.WriteLine("End\n");
}
```

VI. OutOfMemoryException. Part 3



```
1 reference
private static void MemoryTest()
{
    Console.WriteLine("Start");
    double[][] arrays = new double[120][];
    double[] array1 = new double[2], array2 = new double[2], array3 = new double[2];
    array1 = new double[223_300_500];
    for (int i = 0; i < arrays.Length; i++)
        arrays[i] = new double[100_000];
    try
    {
        Console.WriteLine("Try: 1");
        array2 = new double[5_000_000];
        Console.WriteLine("Try: 2");
        //Console.WriteLine("Try: " + Func1(string.Empty));
    }
    catch (Exception e)
    {
        Console.WriteLine("Catch 1: " + e.GetType().Name);
    }
    finally
    {
        array3 = new double[5_000_000];
        Console.WriteLine("Finally");
    }
    Console.WriteLine("End + " + array1.Sum() + array2.Sum() + array3.Sum() + arrays.Sum(a => a.Sum()));
}
```



```
Start
Try: 1
Catch 1: OutOfMemoryException

Unhandled Exception: OutOfMemoryException.
Press any key to continue . . .
```

VI. Memory. Difference?

Max object size:

- .NET Framework – 2GB
- Cab be overridden by <gcAllowVeryLargeObjects> configuration file setting
- .Net Core – no limit by default

Total max allocate virtual memory

- 32-bit process + 32-bit system – 2GB
- 32-bit process + 64-bit system – 4GB
- 64-bit process + 64-bit system – 8TB

© MSDN

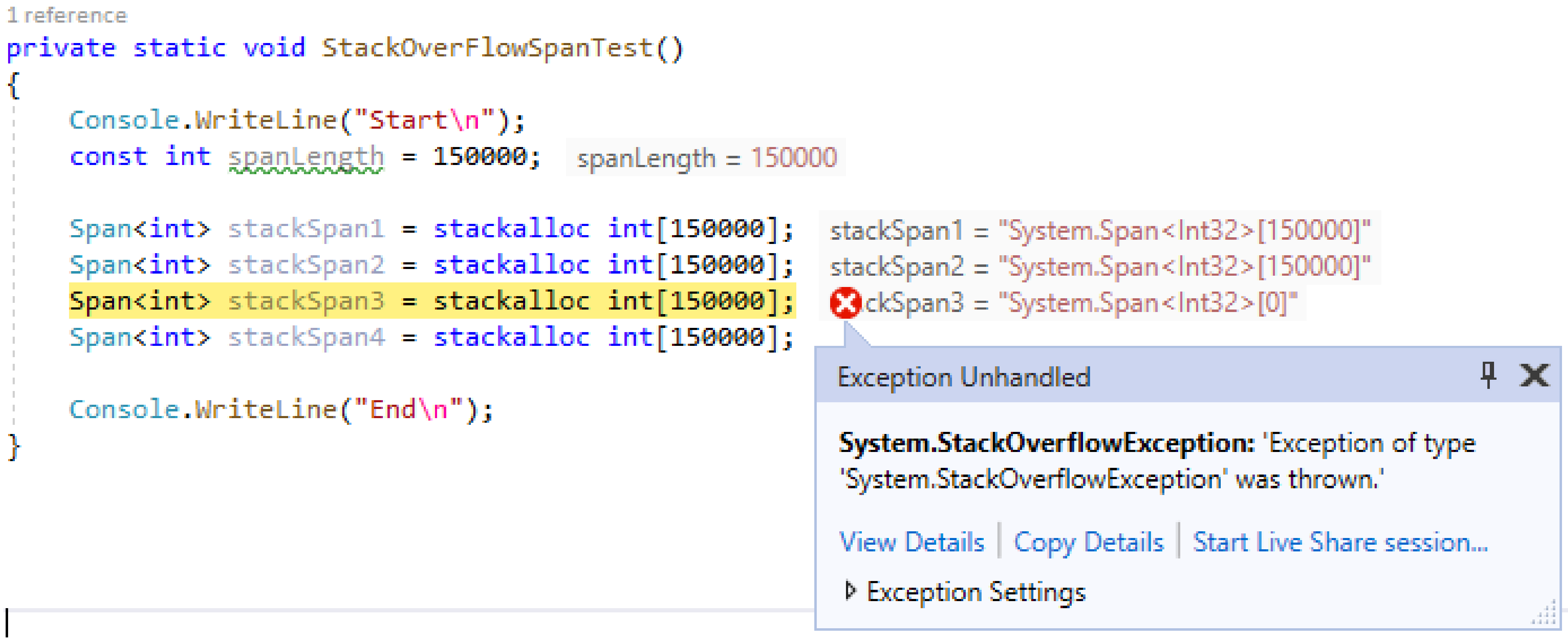
VII. StackOverflowException

Starting with the .NET Framework 2.0, you can't catch a ***StackOverflowException*** object with a try/catch block,

and the corresponding process is terminated by default.

© MSDN

VII. StackOverflowException. Part 2



```
1 reference
private static void StackOverFlowSpanTest()
{
    Console.WriteLine("Start\n");
    const int spanLength = 150000;  spanLength = 150000

    Span<int> stackSpan1 = stackalloc int[150000];  stackSpan1 = "System.Span<Int32>[150000]"
    Span<int> stackSpan2 = stackalloc int[150000];  stackSpan2 = "System.Span<Int32>[150000]"
    Span<int> stackSpan3 = stackalloc int[150000];  ckSpan3 = "System.Span<Int32>[0]"
    Span<int> stackSpan4 = stackalloc int[150000];

    Console.WriteLine("End\n");
}
```

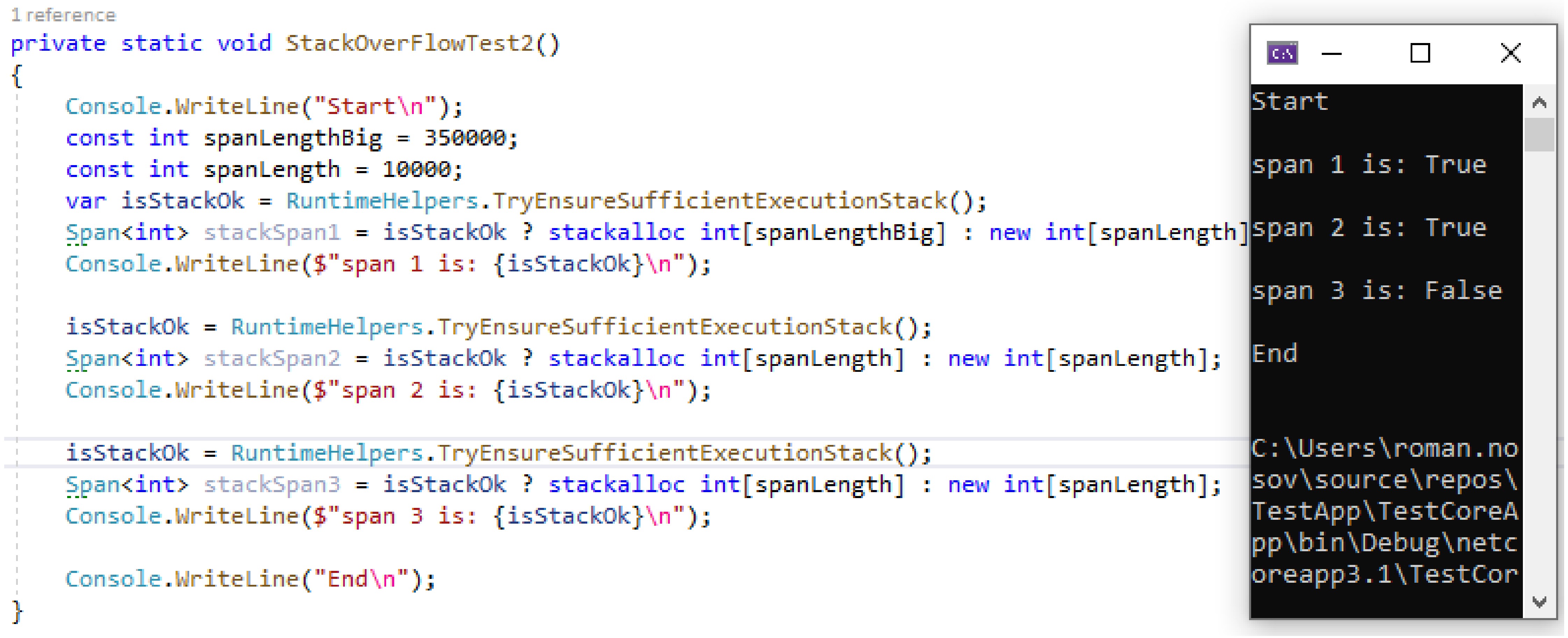
Exception Unhandled

System.StackOverflowException: 'Exception of type 'System.StackOverflowException' was thrown.'

View Details | Copy Details | Start Live Share session...

Exception Settings

VII. StackOverflowException. Part 3



```
1 reference
private static void StackOverFlowTest2()
{
    Console.WriteLine("Start\n");
    const int spanLengthBig = 350000;
    const int spanLength = 10000;
    var isStackOk = RuntimeHelpers.TryEnsureSufficientExecutionStack();
    Span<int> stackSpan1 = isStackOk ? stackalloc int[spanLengthBig] : new int[spanLength];
    Console.WriteLine($"span 1 is: {isStackOk}\n");

    isStackOk = RuntimeHelpers.TryEnsureSufficientExecutionStack();
    Span<int> stackSpan2 = isStackOk ? stackalloc int[spanLength] : new int[spanLength];
    Console.WriteLine($"span 2 is: {isStackOk}\n");

    isStackOk = RuntimeHelpers.TryEnsureSufficientExecutionStack();
    Span<int> stackSpan3 = isStackOk ? stackalloc int[spanLength] : new int[spanLength];
    Console.WriteLine($"span 3 is: {isStackOk}\n");

    Console.WriteLine("End\n");
}
```

```
Start

span 1 is: True

span 2 is: True

span 3 is: False

End


C:\Users\roman.no
sov\source\repos\
TestApp\TestCoreA
pp\bin\Debug\netc
oreapp3.1\TestCor
```

VII. Stack. Difference?

RuntimeHelpers

- EnsureSufficientExecutionStack();
- TryEnsureSufficientExecutionStack();

Ensures that the remaining stack space is large enough to execute the average .NET function.

© MSDN

What is “average .NET function”?

- .Net Framework x86 – 512 KB (half of stack size)
- .Net Framework x64 – 2 MB (half of stack size)
- .Net Core – 64 KB

И под конец

I. Немного теории
II. Немного практики
III. Уроним try-catch-finally?
IV. Уроним try-catch-finally? *(А теперь серьезно)*
V. Заключение

.NET Fremawork ***VS*** .NET Core *****

***** Где есть раззличие

Вместо заключения

Это все интересно, но?

I. Process was eliminated
II. Environment.FastFail()
III. Corrupted state exception
IV. InvalidProgramException
V. Exceptions in FirstChanceException
VI. OutOfMemoryExceptions
VII. StackOverflowException

Вместо заключения

Это все интересно, но?

I. Process was eliminated
~~II. Environment.FastFail()~~
III. Corrupted state exception
IV. InvalidProgramException
~~V. Exceptions in FirstChanceException~~
VI. OutOfMemoryExceptions
VII. StackOverflowException

Спасибо за внимание