
ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР
ПРОГРАММИРОВАНИЯ И ВЫСОКИХ
ТЕХНОЛОГИЙ

2020г.

# ИСКЛЮЧЕНИЯ

# AGENDA

- ✓ Понятие исключительной ситуации
- ✓ Структурная обработка исключений в .NET
- ✓ Проектирование собственных типов исключений

# Исключения

**Исключения** – ненормальные события, препятствующие успешному выполнению конкретной задачи.

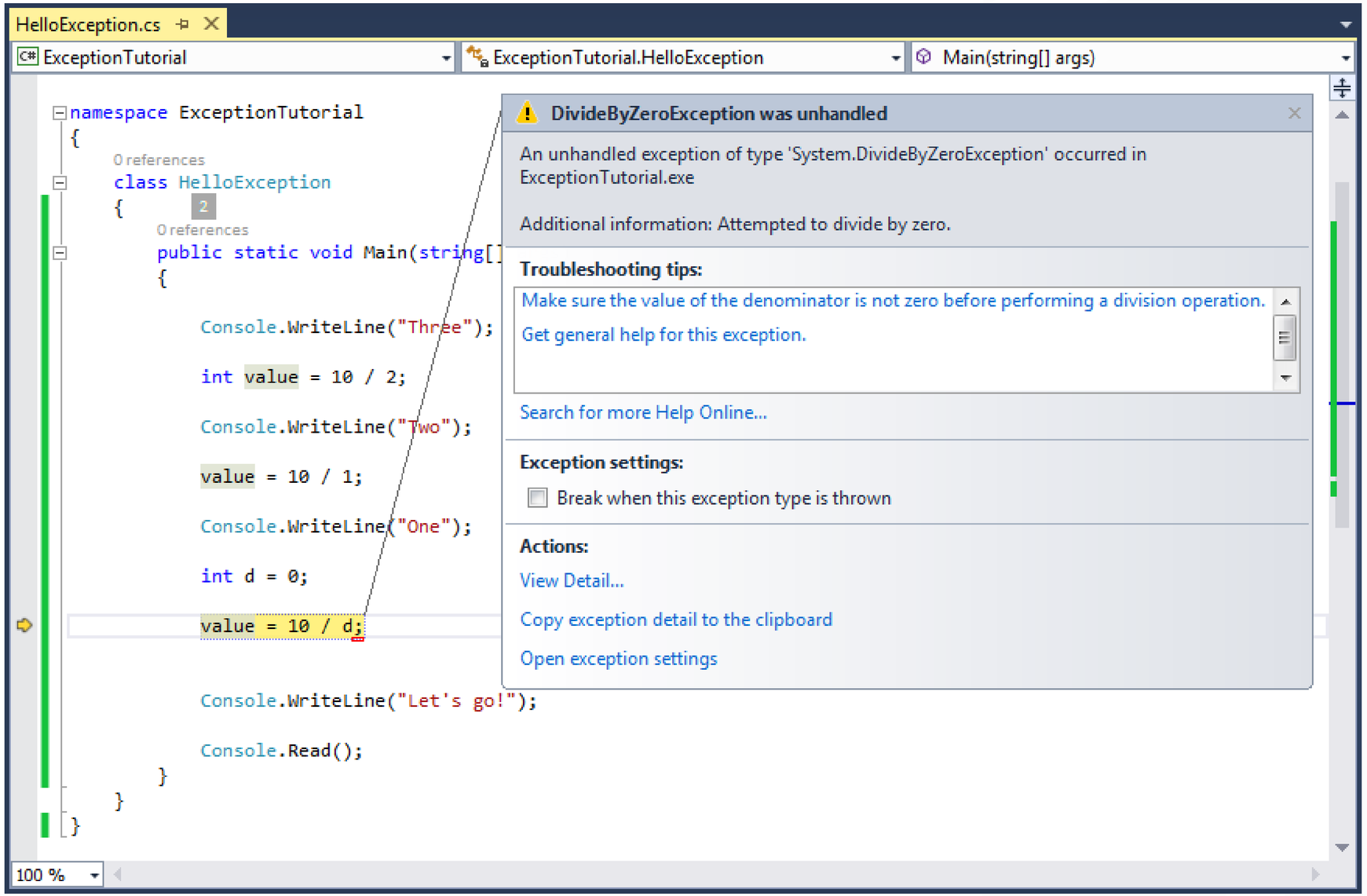
**Исключения** – ошибки времени выполнения, которые могут привести к непредвиденному завершению программы.

Исключения в C# разрушают нормальное течение программы.

**Обработка исключений** – процесс обработки таких ошибок времени выполнения.

Обработка исключения указывает на действие, которое необходимо выполнить в случае возникновения ошибки, чтобы предотвратить принудительное завершение программы.

# Исключения


HelloException.cs
ExceptionTutorial
ExceptionTutorial.HelloException
Main(string[] args)
namespace ExceptionTutorial
{
0 references
class HelloException
{
2
0 references
public static void Main(string[]
{
Console.WriteLine("Three");
int value = 10 / 2;
Console.WriteLine("Two");
value = 10 / 1;
Console.WriteLine("One");
int d = 0;
value = 10 / d;
Console.WriteLine("Let's go!");
Console.Read();
}
}
}
DivideByZeroException was unhandled
An unhandled exception of type 'System.DivideByZeroException' occurred in ExceptionTutorial.exe
Additional information: Attempted to divide by zero.
Troubleshooting tips:
Make sure the value of the denominator is not zero before performing a division operation.
Get general help for this exception.
Search for more Help Online...
Exception settings:
Break when this exception type is thrown
Actions:
View Detail...
Copy exception detail to the clipboard
Open exception settings
100 %

# Исключения

C# может обрабатывать различные типы исключений с помощью соответствующих выражений обработки исключений.

Это позволяет обрабатывать два основных типа исключений:

- **Исключения системного уровня** – это исключения, генерируемые CLR. Эти исключения считаются неустранимыми, фатальными ошибками. Исключения системного уровня получаются непосредственно из базового класса **System.SystemException**.

- **Исключения уровня приложения** – исключения генерируемые пользовательскими приложениями. Платформа .NET позволяет создавать свои собственные исключения, учитывающие специфику приложения. Исключения уровня приложения получаются непосредственно из базового класса **System.ApplicationException**.

**SystemException** и **ApplicationException** не определяют никаких дополнительных членов, кроме конструкторов. Единственной их цель - идентификация источника ошибки. Оба являются производным от **System.Exception**.

# Исключения

Класс **System.Exception** – стандартный класс для представления исключительных ситуаций.

| Свойства | Описания |
|---|---|
| **Message** | Выводит сообщение с причиной исключения. |
| **Source** | Указывает имя объекта, вызвавшего исключение. |
| **StackTrace** | Указывает содержимое стека вызовов. |
| **InnerException** | Возвращает экземпляр **Exception**. |

Библиотека классов **.NET** содержит большое число разнообразных классов, порожденных от **System.Exception** и описывающих конкретные исключительные ситуации.

# Часто используемые исключения

| Исключения | Описания |
|---|---|
| System.<br>NullReferenceException | Попытка использовать пустую ссылку (null). |
| System.<br>OutOfMemoryException | Недостаточно памяти для дальнейшего выполнения приложения, например для размещения объекта. |
| System.<br>OverflowException | Арифметическое переполнение, результат операции арифметических вычислений, приведения типов или конвертации слишком велик, чтобы поместиться в целевой объект или переменную. |
| System.<br>StackOverflowException | Переполнение стека выполнения из-за того, что он содержит слишком много вызовов вложенных методов. |
| System.<br>FormatException | Несоответствие формата аргумента формату типа данных вызванного метода. |

# Часто используемые исключения

| Исключения | Описания |
|---|---|
| System.<br>ArgumentException | Один из агруметов метода не соответствует спецификациям. |
| System.<br>ArrayTypeMismatchException | Тип сохраняемого значения несовместим с типом массива. |
| System.<br>DivideByZeroException | Попытка деления на ноль. |
| System.<br>IndexOutOfRangeException | Индекс выходит за границы массива или меньше нуля. |
| System.<br>InvalidCastException | Неверно выполнено приведение типа. |

# Обработка исключений

Исключения, возникающие при работе конкретных программ, необходимо перехватывать соответствующими процедурами обработки.

Язык **C#** предоставляет разработчикам возможность для обработки исключениий - конструкция **try...catch...finally**.

При использовании блока **try...catch..finally** вначале выполняются все инструкции в блоке **try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок **finally**. И затем конструкция **try..catch..finally** завершает свою работу.

Если же в блоке **try** вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда **CLR** начинает искать блок **catch**, который может обработать данное исключение. Если нужный блок **catch** найден, то он выполняется, и после его завершения выполняется блок **finally**.

Если нужный блок **catch** не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

# Обработка исключений

```
try
{
    // код программы
}
catch[(<ExceptionClass> <objException>)]
{
    // механизм обработки исключения
}
finally
{
    // код программы
}
```

Если исключение не перехватывается в программе, то оно будет перехвачено исполняющей системой. А исполняющая система выдаст сообщение об ошибке и прервет выполнение программы.

# Обработка исключений

Блок **try** может вызывать несколько типов исключений, необходимых для обработки блоком **catch**.

C# позволяет определять несколько блоков **catch** для обработки различных типов исключений. Блоки **catch** для различных типов необходимо располагать в строгой иерархии от узкоспециализированных к общим, следуя иерархии их наследования.

В зависимости от типа исключения, вызываемого блоком **try**, выполняется соответствующий (если есть) блок **catch**.

Общий блок **catch** может обрабатывать все типы исключений. Недостаток общего блока **catch** в том, что нет экземпляра исключения, так что неясно, какое действие необходимо выполнить для обработки исключения.

**catch (Exception)** – позволяет перехватывать все исключительные ситуации, генерируемые **CLR**.

**catch { }** – Блок будет обрабатывать любые исключительные ситуации, в том числе и не связанные с исполняющей средой.

# Обработка исключений

Выражение **throw** в **C#** позволяет программно вызывать исключения.

Выражение **throw** принимает экземпляр конкретного исключения в качестве параметра.

При вызове исключения с помощью ключевого слова **throw**, исключение обрабатывается блоком **catch**.

**Важно:** Объект, указанный после **throw**, должен быть объектом класса исключительной ситуации. В **C#** классами исключительных ситуаций являются класс **System.Exception** и все его наследники. В некоторых языках для **.NET** можно (хотя и не рекомендуется) генерировать исключения, не являющиеся производными от **Exception**. В таком случае **CLR** автоматически поместит объект исключения в оболочку класса **RuntimeWrappedException**, который наследуется от **Exception**.

# Пользовательские исключения

Пользователь может создать собственный класс для представления информации об исключительной ситуации. Единственным условием является прямое или косвенное наследование этого класса от класса **System.Exception**.

Эти исключения позволяют распознавать случаи неожиданных событий в заданных программах и выводить пользовательские сообщения.

Они позволяют упрощать и улучшать процесс обработки ошибок в программе.

Пользовательские исключения могут быть созданы наследованием от класса **Exception**, класса **SystemException** или класса **ApplicationException**.

Производные классы должны определять, по меньшей мере, четыре конструктора: один конструктор по умолчанию, один конструктор, задающий свойство сообщения, и еще один, задающий свойства **Message** и **InnerException**. Четвертый конструктор служит для сериализации исключения. Новые классы исключений должны быть сериализуемыми.

# Пользовательские исключения

```
public class CustomException : AplicationException
{
    public CustomException (string message) : base(message)
    {   }
}
...
    static void Main(string[] args)
    {
        try{
            throw new CustomException (“custom exception”);
        }
        catch(CustomException objCustom){
            Console.WriteLine(objCustom.Message);
        }
    }
```

# Итог

**Исключения:**

- Исключения – это ошибки времени выполнения приложения.

**Вызов и перехват исключений:**

- Выражение **throw** позволяет программно вызывать исключения.
- Конструкция **try-catch-finally** предназначена для перехвата и обработки исключений.

**Вложенные блоки try и использование нескольких блоков catch:**

- Вложенные блоки **try** позволяют иметь в блоке **try** конструкцию **try-catch-finally**.
- Несколько блоков **catch** могут быть реализованы при возникновении в блоке **try** нескольких типов исключений.
- Блоки **catch** для разных типов исключений располагаются в соответствии с иерархией наследования данных типов, от специфических к базовым.

**Пользовательские исключения:**

- Пользовательские исключения позволяют обрабатывать системные и специфичные для приложений ошибки времени выполнения.

# Задание

Разработать класс описывающий уникальное для вашего приложения исключение.

(Методы нашего приложения вызывают исключение, когда в качестве аргумента типа **string** приходит пустая строка.)

Разработать класс, в котором будет вызываться разработанное ранее исключение.

В классе **Program** реализовать обработку исключение таким образом, чтобы в зависимости от перехваченного исключения выводилось соответстующее сообщение.

Создать ситуацию при которой будет обработано созданное ранее исключение.