

# Исключения



# ***Исключительные ситуации в С#***

Structured Exception Handling – SHE  
унифицированный подход для языков  
ориентированных на платформу .NET

System.Exception

# Исключительная ситуация exception -

Это состояние ошибки, обнаруженное в программе в ходе ее выполнения (деление на ноль, невозможность выделения памяти при создании нового объекта и т.д. )

Генерируется оператором

**throw( <выражение> )**

аргумент - объект типа исключение

# Генерация исключения

```
if (b==0)
    throw new Exception("Zero devision");
else
    c = a / b;
```


# ГЕНЕРИРОВАНИЕ И РАСПОЗНАВАНИЕ ИСКЛЮЧЕНИЙ

- ▶ try – контролируемый блок
- ▶ throw - генерация искл. ситуации  
внутри try
- ▶ catch – обработчики исключений,  
идут за try (несколько)
- ▶ finally - код, очищающий ресурсы и  
др. действия (выполняется всегда)  
(один на один try)

```
try
{
    // Блок кода, проверяемый на наличие ошибок.
}

catch (ExceptionType1 exOb)
{
    // Обработчик исключения типа ExceptionType1.
}

catch (ExceptionType2 exOb)
{
    // Обработчик исключения типа ExceptionType2.
}
```



тип исключения (catch type)

Если код в блоке try не порождает исключение, CLR никогда не переходит к выполнению кода в соответствующем блоке catch

```
FileStream fs = null;
try
{
    fs = new FileStream(pathname, FileMode.Open);
    // Обработка данных
}
catch (IOException)
{
    // Код восстановления
}
finally
{
    // Файл следует закрыть
    if (fs != null) fs.Close();
}
```

по возможности коротким

только один блок finally

источник исключения → catch или finally, CLR продолжает работу, теряется информация о первом исключении, вброшенном в блоке try. Скорее всего новое исключение останется необработанным → CLR завершает процесс

DebugAny CPUПродолжить

События жизненного циклаПоток: [6964] Основной поток

Кадр стека: CSharp2017\_lection.StatProgram.Test.M

program.cs

CSharp2017\_lection.StatProgram.TestMain(string[] args)

```
class Test
{
    //ссылка: 0
    static void Main(string[] args)
    {
        int input = Convert.ToInt32(Console.ReadLine());
        int i = 100 / input;
    }
}
```

Средства диагностики

Выбор средствУвеличитьУменьшить

Сеанс диагностики: 4 с (Выбрано: 4,642 с)

Память процесса мусораСнимок

События

FormatException не обработано

Необработанное исключение типа "System.FormatException" в mscorlib.dll

Дополнительные сведения: Входная строка имела неверный формат.

Советы по устранению неполадок:

При преобразовании строки в дату/время (DateTime) выполните синтаксический разбор строки, чтобы получить дату перед помещением "Дата/время" (DateTime).

Убедитесь, что ваши аргументы метода имеют правильный формат.

Получить общие справочные сведения об этом исключении.

Поиск дополнительных справочных сведений в сети...

Параметры исключений:

☐ Остановить при возникновении исключения этого типа

Действия:

Просмотр сведений...

Скопировать сведения исключения в буфер обмена

Открыть параметры исключений

//static void Main

Контрольные значения 1 Операция



## Сведения об исключении:

System.FormatException	{ "Входная строка имела неверный формат." }
Data	{ System.Collections.ListDictionaryInternal }
HelpLink	null
HResult	-2146233033
InnerException	null
Message	Входная строка имела неверный формат.
Source	mscorlib
StackTrace	в System.Number.StringToNumber(String str, N
TargetSite	{ Void StringToNumber(System.String, System.Glo
Attributes	Private   Static   HideBySig
CallingConvention	Standard
ContainsGenericParameters	false
CustomAttributes	Count = 1
DeclaringType	{ Name = "Number" FullName = "System.Number
IsAbstract	false
IsAssembly	false
IsConstructor	false
IsFamily	false
IsFamilyAndAssembly	false
IsFamilyOrAssembly	false
IsFinal	false
IsGenericMethod	false
IsGenericMethodDefinition	false
IsHideBySig	true
IsPrivate	true
IsPublic	false
IsSecurityCritical	true
IsSecuritySafeCritical	true
IsSecurityTransparent	false

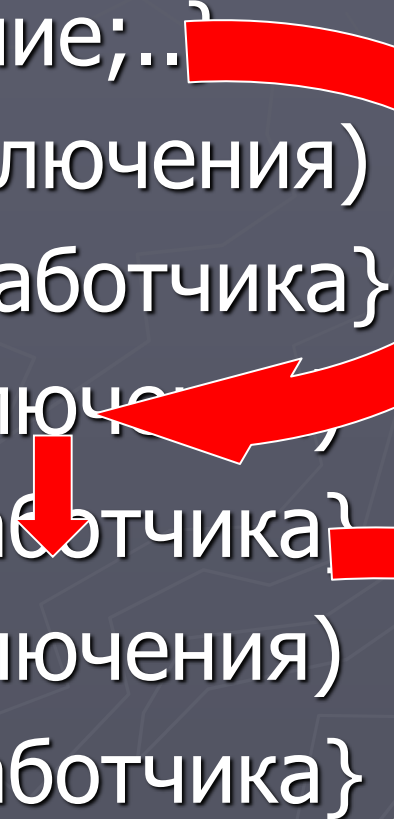
OK

# try - catch

catch (тип\_исключения имя\_переменной)

Поиск подходящего блока catch в CLR осуществляется сверху вниз, поэтому наиболее конкретные обработчики должны находиться в начале списка.

```
try { ...throw выражение; ... }  
catch( объявление исключения )  
    { операторы обработчика }  
catch(объявление исключения)  
    { операторы обработчика }  
catch(объявление исключения)  
    { операторы обработчика }
```



Сначала  
следуют потомки  
с наибольшей  
глубиной  
наследования,  
потом — их  
базовые классы  
(если таковые  
имеются) и,  
наконец, —  
класс  
System.Exception

try-catch,

try-finally,

try-catch-finally

```
public class Exception : ISerializable, _Exception
{ // Общедоступные конструкторы
    public Exception(string message, Exception innerException);
    public Exception(string message);
    public Exception(); ...
// Методы
    public virtual Exception GetBaseException() ;
    public virtual void GetObjectData(SerializationInfo info,
    StreamingContext context);
// Свойства
    public virtual IDictionary Data { get; }
    public virtual string HelpLink { get; set; }
    public Exception InnerException { get; }
    public virtual string Message { get; }
    public virtual string Source { get; set; }
    public virtual string StackTrace { get; }
    public MethodBase TargetSite { get; } ... }
```

CLR позволяет  
генерировать в  
качестве  
исключений  
экземпляры любого  
типа

```
static int ExceptionExample(int x, int y)
{
    if (y == 0 )
    {
        Exception a = new Exception();
        a.HelpLink = "http://www.belstu.by";
        a.Data.Add("Время возникновения: ", DateTime.Now);
        throw a;
    }
    return x / y;
}
```

```
static void Main()
{
    try
    {
        int x = int.Parse(Console.ReadLine());
        int y = int.Parse(Console.ReadLine());
        ExceptionExample(x, y);
    }
```

```
    catch (Exception ex)
    {
        Console.Write(ex.Message + "\n\n");
        Console.Write(ex.TargetSite + "\n\n");
        Console.Write(ex.StackTrace + "\n\n");
        Console.Write(ex.HelpLink + "\n\n");
        if (ex.Data != null)
        {
            Console.WriteLine("Сведения: \n");
            foreach (DictionaryEntry d in ex.Data)
                Console.WriteLine("-> {0} {1}", d.Key, d.Value);
            Console.WriteLine("\n\n");
        }
    }
}
```

Имена и  
сигнатуры  
методов, вызов  
которых стал  
источником  
исключения

Текст с  
описанием  
причины  
исключения

Адрес  
документации с  
информацией  
об исключении

C:\Windows\system32\cmd.exe

9  
0

Выдано исключение типа "System.Exception".

Int32 ExceptionExample(Int32, Int32)

в CSharp2017\_lection.StatProgram.ExceptionExample(Int32 x, Int32 y) в C:\NATALIA\Лекции\ООП\_2\2016\_лекции\Проекты\_к\_лекции\CSharp2016\_lection\CSharp2016\_lection\Program.cs:строка 76

в CSharp2017\_lection.StatProgram.Main() в C:\NATALIA\Лекции\ООП\_2\2016\_лекции\Проекты\_к\_лекции\CSharp2016\_lection\CSharp2016\_lection\Program.cs:строка 86

<http://www.belstu.by>

Сведения:

-> Время возникновения: 05.03.2017 23:48:41

Для продолжения нажмите любую клавишу . . . \_

имена всех методов от точки, в которой было вброшено исключение, до точки, где оно было перехвачено.

System.Exception

System.AggregateException

System.ApplicationException

System.Reflection.InvalidFilterCriteriaException

System.Reflection.TargetException

System.Reflection.TargetInvocationException

System.Reflection.TargetParameterCountException

System.Threading.WaitHandleCannotBeOpenedException

System.Diagnostics.Tracing.EventSourceException

System.InvalidTimeZoneException

System.IO.IsolatedStorage.IsolatedStorageException

System.Runtime.CompilerServices.RuntimeWrappedException

System.SystemException

System.Threading.AbandonedMutexException

System.AccessViolationException

System.Reflection.AmbiguousMatchException

System.AppDomainUnloadedException

System.ArgumentException

System.ArithmeticException

System.ArrayTypeMismatchException

System.BadImageFormatException

System.CannotUnloadAppDomainException

System.ContextMarshalException

System.Security.Cryptography.CryptographicException

mscorlib.dll,



System.Data.MisalignedException

System.ExecutionEngineException

System.Runtime.InteropServices.ExternalException

System.Runtime.InteropServices.COMException

System.Runtime.InteropServices.SEHException

System.FormatException

System.Reflection.CustomAttributeFormatException

System.Security.HostProtectionException

System.Security.Principal.IdentityNotMappedException

System.IndexOutOfRangeException

System.InvalidOperationException

System.InvalidCastException

System.Runtime.InteropServices.InvalidComObjectException

System.Runtime.InteropServices.InvalidOleVariantTypeException

System.InvalidOperationException

System.ObjectDisposedException

System.InvalidProgramException

System.IO.IOException

System.IO.DirectoryNotFoundException

System.IO.DriveNotFoundException

System.IO.EndOfStreamException

System.IO.FileLoadException

System.IO.FileNotFoundException

System.Exception

```
graph TD; A[System.Exception] --> B[System.SystemException]; A --> C[System.ApplicationException];
```

System.SystemException  
***ИСКЛЮЧЕНИЯ УРОВНЯ  
СИСТЕМЫ***

System.ApplicationException  
***ИСКЛЮЧЕНИЯ УРОВНЯ  
ПРИЛОЖЕНИЯ***

Создание исключений:

- 1) наследование от ApplicationException;
- 2) атрибут [System.Serializable];
- 3) конструктор по умолчанию;
- 4) конструктор с установкой значение Message;
- 4) конструктор для обработки "внутренних исключений";
- 5) конструктор для сериализации типа



```
catch (OverflowException ex)
{
    Console.Write("Данное число не входит в диапазон" +
        ex.Message);
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Деление на ноль "+ ex.Message);
}

catch (IndexOutOfRangeException )
{
    Console.WriteLine("Индекс выходит за пределы\n");
}
catch //универсальный обработчик
{ }
```

предпочтительней

```
catch (Exception ex) // это общий обработчик
исключений
{
    //...
}
```

# Генерация исключений

```
try
{
    if (i > 255)
        // Генерируем исключение
        throw new OverflowException();
}
```

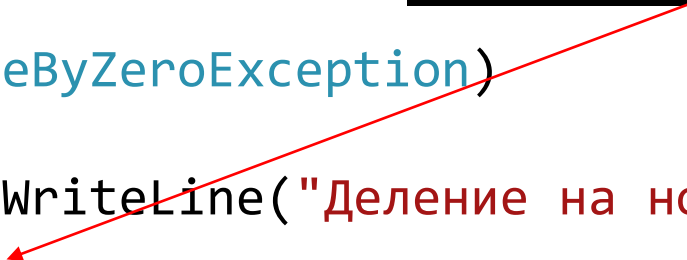
Должен существовать catch

# Повторная генерация исключения

создание нового объекта посредством повторного использования старого с помощью оператора `throw` без параметров

```
static void MathOp(int x, int y)
{
    try
    {
        int result = x / y;
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("Деление на ноль!");
        throw;
    }
}
```

при повторном вызове перехваченного исключения с помощью ключевого слова `throw` удаления из стека информации о начальной точке не происходит

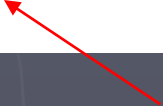


# Фильтры исключений

Фильтр исключения позволяет указать дополнительные условия, при которых используется обработчик исключения.

Эти условия принимают форму булева выражения, перед которым ставится ключевое слово `when`.

```
catch (Exception ex) when
    (ex.GetType() != typeof(System.FormatException))
{
    // Обработка всех ранее не перехваченных исключений,
    // кроме того, которое называется FormatException
}
```



этот обработчик будет проигнорирован для `FormatException`

# Механизм

1) Исключение не произошло

1.1. `try` выполняем до конца

1.2. `catch` пропускаем

1.3. `finally` выполняем

## 2) Исключение произошло

2.1. выполнение try прекращается (все что идет за возникшим исключением игнорируется)

2.2 ищем блок catch на соответствие по типу исключения

2.2.1. если нет catch

2.2.1.1 разматывает стек, локальные объекты, выходят из области видимости

2.2.1.2 снова генерируется исключение в точке вызова метода

2.2.1.3. если блока не найдено, то сообщение - необработанное исключение → дальнейшее выполнение программы останавливается

## 2.2.2 catch найден

2.2.2.1 Передается управление ближайшему **catch-обработчику, совместимому** с типом выброшенного исключения

2.2.2.2. объект-исключения передается, если это предусмотрено, обработчику в качестве параметра.

## 2.3. переходим в finally

### 2.3.1. если нет finally

2.3.1.1. выполнение программы продолжается начиная с позиции , след. за последним обработчиком данного блока try

# Свойства и правила


- ▶ try могут быть вложенные
- ▶ более специфичные исключения обрабатываются первыми
- ▶ Свои классы исключений должны наследоваться от `System.Exception` или `System.ApplicationException`
- ▶ может иметь одну конструкцию catch без аргументов ( нежелательно)
- ▶ finally выполняется всегда
  - ▶ ( не выполняется в случае выброса `StackOverflowException` или `System.exit(0)`)
  - ▶ используйте блоки finally



- ▶ Может быть трансляция исключения
- ▶ при использовании инструкций `lock`, `using` и `foreach` блоки `try/finally` создаются автоматически
- ▶ Генерация исключений в `finally` (нежелательно - код восстановления или очистки будет выполнен не полностью)
- ▶ процедура обработки исключений медленная

# Проверяемые и непроверяемые операции для примитивных типов

```
Byte b8 = 100;  
b8 = (Byte)(b8 + 200);
```



по умолчанию проверка переполнения отключена

```
Byte b8u = unchecked((Byte)(b8 + 400)); //OK
```



команды без проверки переполнения

```
Byte b8c = checked((Byte)(b8 + 300)); //Ex
```



команды с проверкой

Необработанное исключение: System.OverflowException: Переполнение в результате выполнения арифметической операции.