Обработка исключительных ситуаций

Лекция №6

Обработка исключительных ситуаций

Все исключения являются подклассами класса Exception пространства имен System.

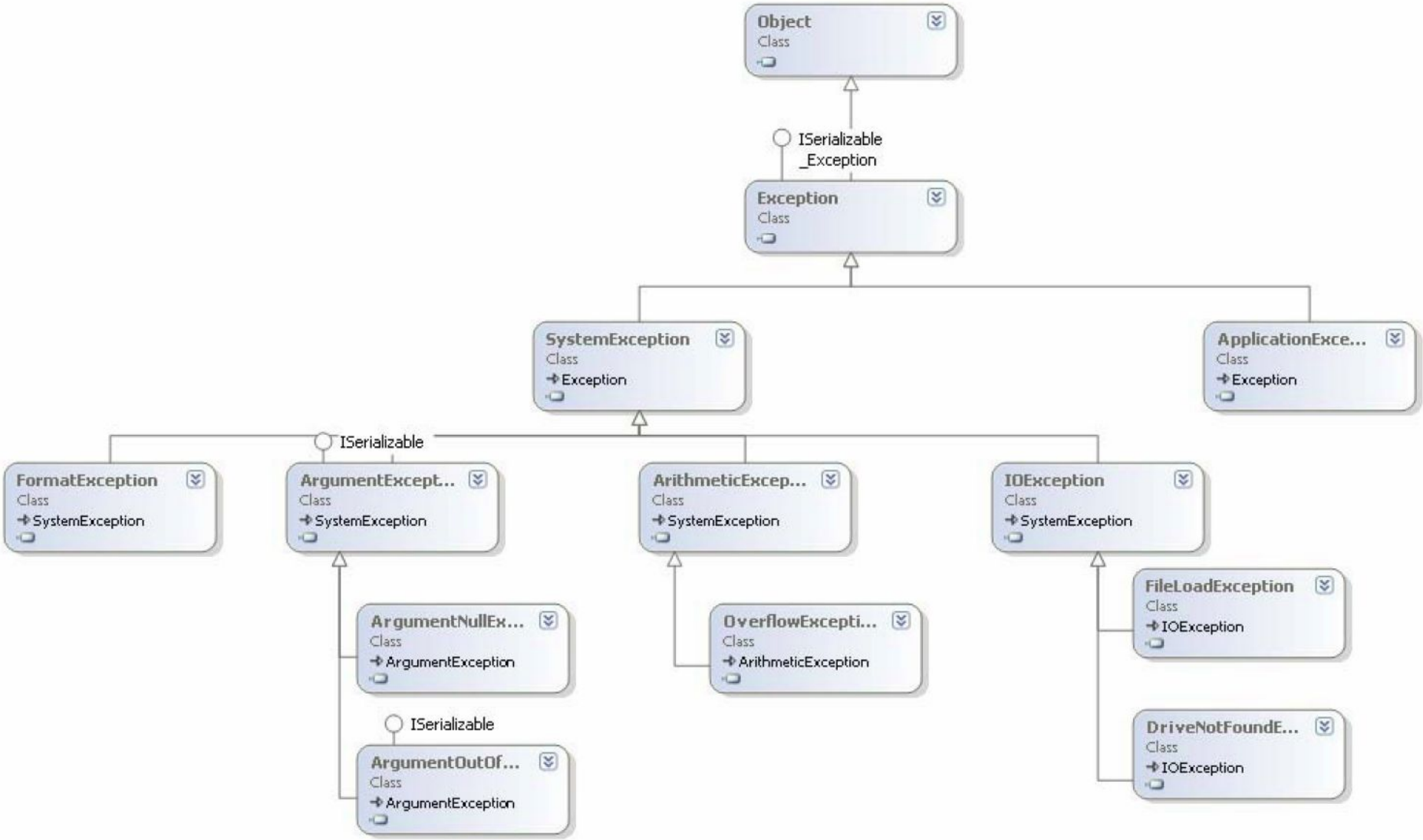
Исключения генерирует среда программирования или программист.

Часто используемые исключения пространства имен System:

Исключение	***Значение***
ArrayTypeMismatchException	Тип сохраняемого значения несовместим с типом массива
DivideByZeroException	Попытка деления на ноль
IndexOutOfRangeException	Индекс массива оказался вне диапазона

Outof MemoryException	Обращение к оператору **new** оказалось неудачным из-за недостаточного объема свободной памяти
OverFlowException	Имеет место арифметическое переполнение
FormatException	Попытка передать в метод аргумент неверного формата
InvalidCastException	Ошибка преобразования типа

Иерархия исключений



Название свойства	Описание
string Message	Содержит текст сообщения с указанием причины возникновения исключения.
IDictionary Data	Ссылка на набор пар «параметр-значение». Обычно код, генерирующий исключение, добавляет записи в этот набор. Код, перехвативший исключение, может использовать эти данные для получения дополнительной информации о причине возникновения исключения.
string Source	Содержит имя сборки, сгенерировавшей исключение.
string StackTrace	Содержит имена и сигнатуры методов, вызов которых привел к возникновению исключения.
MethodBase TargetSite	Содержит метод, сгенерировавший исключение.
string HelpLink	Содержит URL документа с описанием исключения
Exception InnerException	Указывает предыдущее исключение, если текущее было сгенерировано при обработке предыдущего исключения

Исключения перехватываются и обрабатываются оператором **try.**

```
try
 {контролируемый блок}
catch (тип1 [имя1]) { обработчик исключения1 }
catch (тип2 [имя2]) { обработчик исключения2 }
. . .
catch { обработчик исключения }
finally  {блок завершения}
```

В контролируемый блок включаются операторы, выполнение которых может привести к ошибке.

С try-блоком можно связать не одну, а несколько catch-инструкций. Однако все catch-инструкции должны перехватывать исключения различного типа.

При возникновении ошибки при выполнении операторов контролируемого блока генерируется исключение.

Выполнение текущего блока прекращается, находится обработчик исключения соответствующего типа, которому и передается выполнение.

После выполнения обработчика выполняется блок **finally.**

Блок **finally** будет выполнен после выхода из try/catch-блока, независимо от условий его выполнения.

Если подходящий обработчик не найден, вызывается стандартный обработчик, который обычно выводит сообщение и останавливает работу программы.

Форма обработчика

catch (тип) { обработчик исключения }

используется если важен только тип исключения, а свойства исключения не используются.

Например:

```
 try
    {
          int y=a/b;
    }
 catch (DivideByZeroException)
{  Console.WriteLine(" Деление на 0"); }
finally   {Console.ReadKey();}
```

Форма обработчика

catch (тип имя) { обработчик исключения }

используется когда имя параметра используется в теле обработчика.

Например:

```
try
   { int y=a/b;}
 catch (DivideByZeroException  f)
{  Console.WriteLine(f.Message+": Деление на 0"); }
```

При попытке деления на 0 выведется сообщение:

Attempted to divide by zero. Деление на 0

Форма обработчика

catch { обработчик исключения }

применяется для перехвата всех исключений, независимо от их типа.

Он может быть только один в операторе try и должен быть помещен после остальных catch-блоков.

```
try
    { int v = Convert.ToInt32(Console.ReadLine()); }
catch { Console.WriteLine("Ошибка!!!"); }
```

В этом примере и в случае ввода очень большого числа и в случае ввода недопустимых в целых константах символов выводится сообщение **"Ошибка!!!"**

Генерирование исключений вручную

Исключение можно сгенерировать вручную, используя инструкцию **throw.**

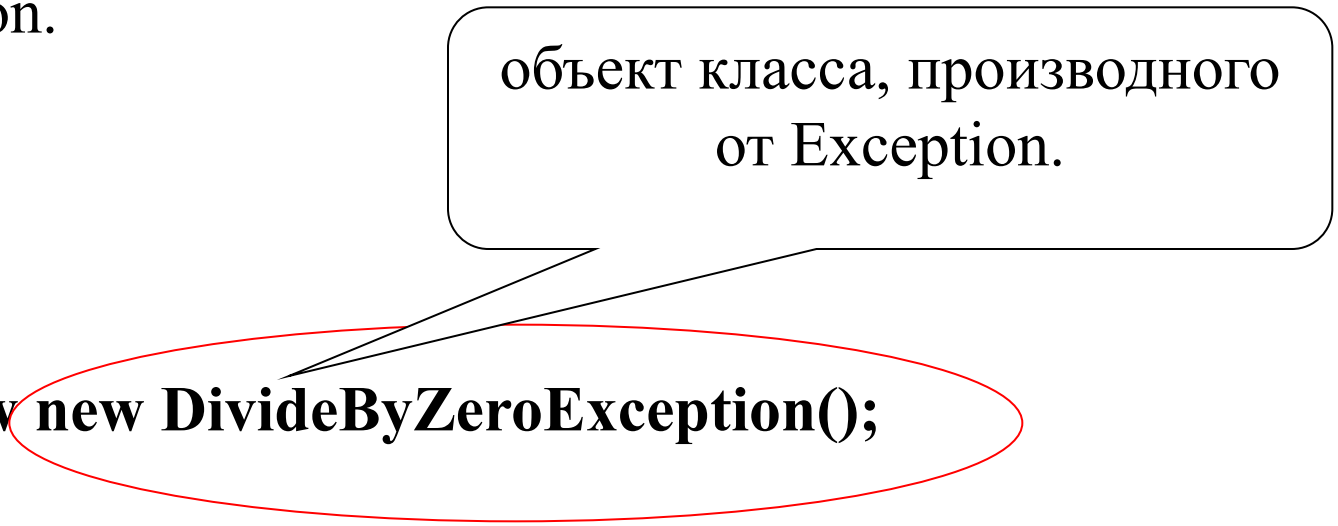
Формат ее записан таков:

throw [параметр];

Параметр - это объект класса исключений, производного от класса Exception.

Например:

```
double x;
if (x == 0) throw new DivideByZeroException();
```



```
class Program
    {
        static void D_0(double x)
        { if (x == 0) throw new Exception("Деление на 0"); }
        static void Main(string[] args)
        {
            try
            {   string s = Console.ReadLine();
                double d = Convert.ToDouble(s);
                int v = Convert.ToInt32(Console.ReadLine());
                double dd = Convert.ToDouble(Console.ReadLine());


  D_0(dd);
                    double y; y = d / dd;
                    Console.WriteLine("y=" + y);
                    // D_0(v);
                    double r = 5 / v;
              }
```

```
catch (FormatException)
            { Console.WriteLine("Неверный ввод"); }

        catch (DivideByZeroException f)
             { Console.WriteLine(f.Message+
                    " Деление на 0_для_целых"); }

        catch (Exception gg)
        { Console.WriteLine("Ошибка:    "+gg.Message+" "
                                  +gg.TargetSite); }

        finally {Console.ReadKey();}
```

Исключение, перехваченное одной catch-инструкцией, можно перегенерировать, чтобы обеспечить возможность его перехвата другой (внешней) catch-инструкцией.

Чтобы повторно сгенерировать исключение, достаточно использовать ключевое слово throw, не указывая параметра:

throw ;

Например:

```
try
   {
    try
     {
      int v = Convert.ToInt32(Console.ReadLine());

      Console.WriteLine("v=" + v);

      }

     catch (FormatException)
          { Console.WriteLine("Неверный ввод"); throw; }
      }
catch (FormatException)
     { Console.WriteLine("Это очень плохо"); }
```

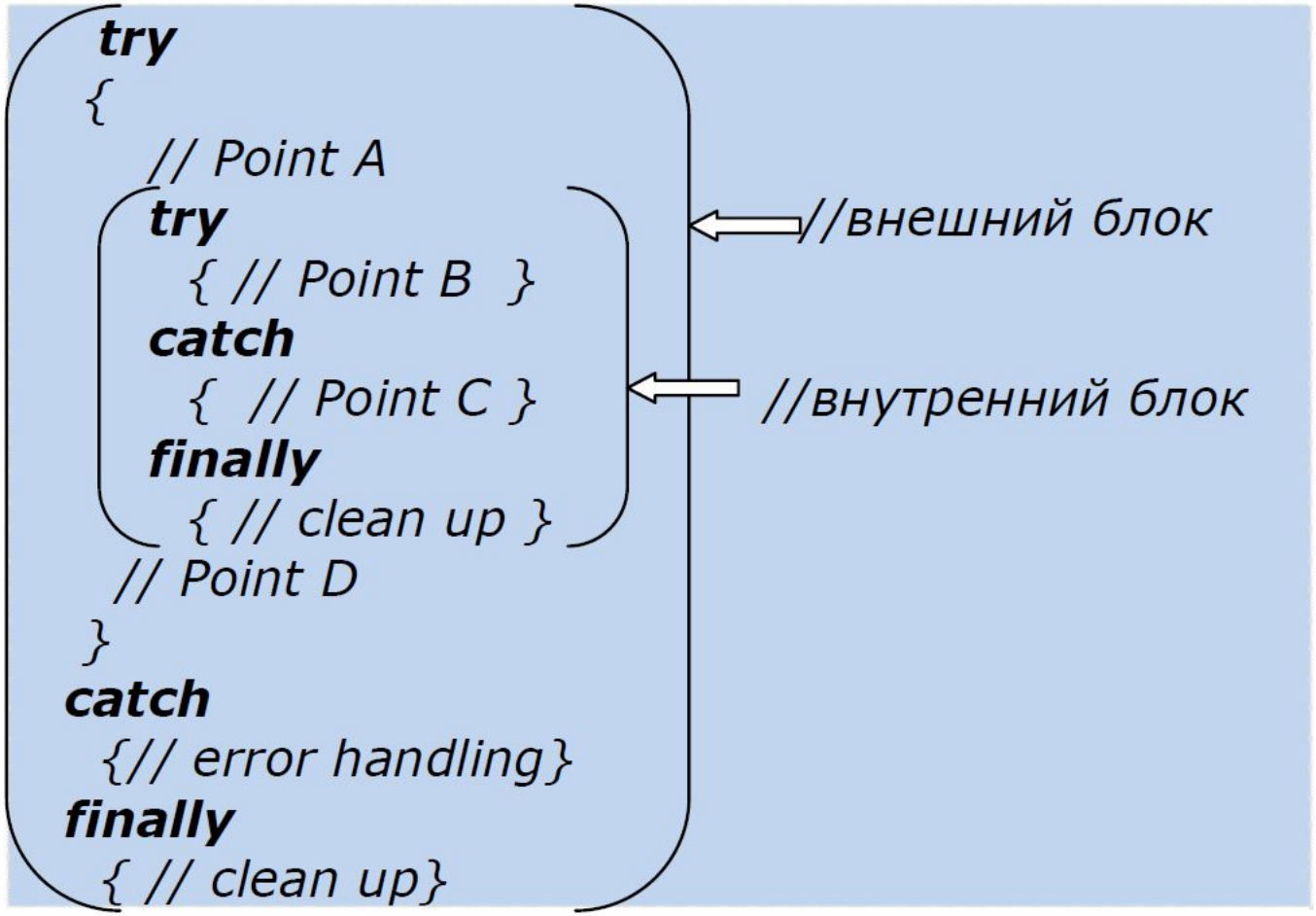
Один try-блок можно вложить в другой.

Исключение, сгенерированное во внутреннем try-блоке и не перехваченное catch-инструкцией, которая связана с этим try-блоком, передается во внешний try-блок.

Часто внешний try-блок используют для перехвата самых серьезных ошибок, позволяя внутренним try-блокам обрабатывать менее опасные.

Вложенные блоки *try*

Блоки ***try*** могут быть вложенными:

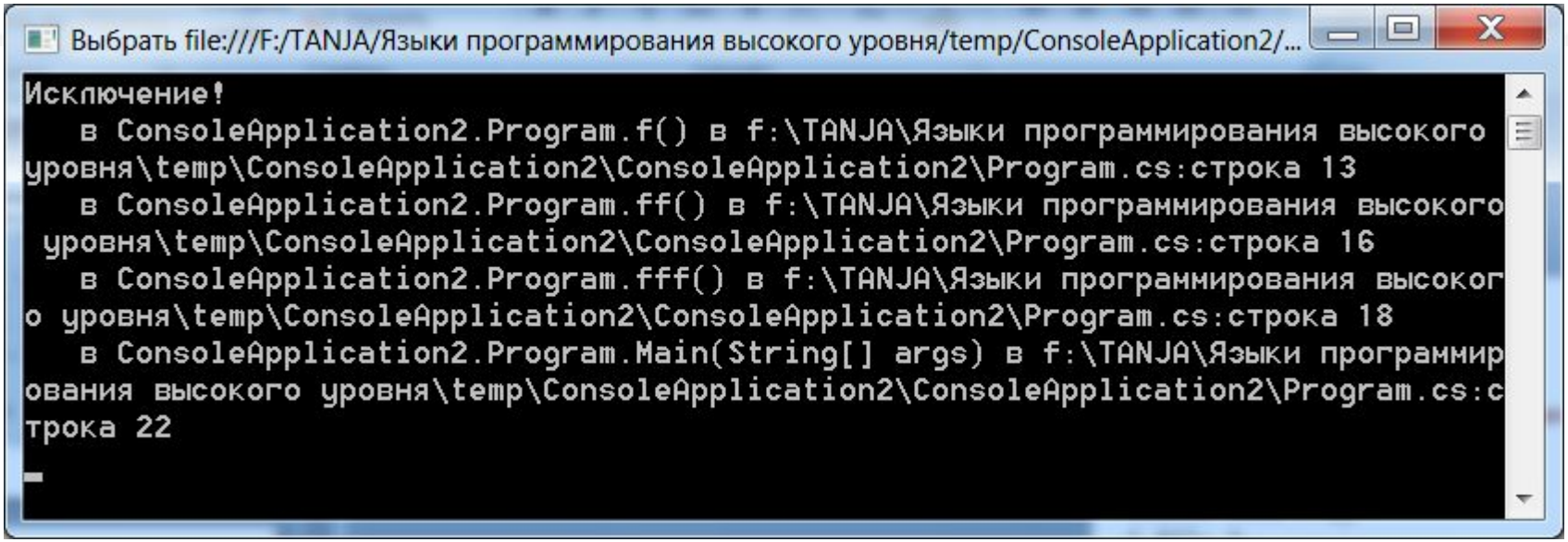


Например:

```
try
            {
                try
                {
                  int v = Convert.ToInt32(Console.ReadLine());

                  Console.WriteLine("v=" + v);

                }

                catch (FormatException)
                { Console.WriteLine("Неверный ввод");}
            }
      catch (OverflowException)
   { Console.WriteLine("Слишком большое целое число"); }
```

```
using System;

namespace NestedTryBlocks
{
    class Program
    {
        static void Main()
        {
            int[] a = new int[5];
            int cnt = 0;
            try //внешний блок try
            {
                for (int i = -3; i <= 3; i++)
                {
                    //при делении на 0 не происходит выход из цикла:
                    //это исключение перехватывается
                    //и обрабатывается вложенным блоком try
                    try //вложенный блок try
                    {
                        a[cnt] = 100 / i;
                        Console.WriteLine(a[cnt]);
                        cnt++;
                    }
                    catch (DivideByZeroException e)
                    {
                        Console.WriteLine("In inner catch");
                        Console.WriteLine(e.Message);
                    }
                }
            }
            catch (IndexOutOfRangeException e)
            {
                Console.WriteLine("In outer catch");
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

```
ссылка 1
static void f()
{
    throw new Exception("Исключение!");
}
ссылка 1
static void ff()
{ f(); }
ссылка 1
static void fff()
{ ff(); }
ссылок 0
static void Main(string[] args)
{
    try
    { fff();}
    catch (Exception e) { Console.WriteLine(e.Message+"\n"+e.StackTrace); }
    Console.ReadKey();
}
```



```
 try
            {
                    int x = 567;
                    byte b = (byte)x;
                    Console.WriteLine(b);

            }
catch (Exception e)
            { Console.WriteLine(e.Message); }
```

```
try
            {
                checked
                {
                    int x = 567;
                    byte b = (byte)x;
                    Console.WriteLine(b);
                }
            }
catch (Exception e)
            { Console.WriteLine(e.Message); }
```

```
try
            {
                unchecked
                {
                    int x = 567;
                    byte b = (byte)x;
                    Console.WriteLine(b);
                }
            }
catch (Exception e)
            { Console.WriteLine(e.Message); }
```

```
int[] arr = new int[3];
int i=0;
int current;
while(i<arr.Length)
{
    try
    {
        Console.Write("arr["+i+"]= ");
        current = Convert.ToInt32(Console.ReadLine());
        arr[i] = current;
        i++;
    }
    catch { }
}
```

```
public class CarIsDeadException : ApplicationException
{
  private string messageDetails = String.Empty;
  public DateTime ErrorTimeStamp {get; set;}
  public string CauseOfError {get; set;}

  public CarIsDeadException(){}
  public CarIsDeadException(string message,
    string cause, DateTime time)
  {
    messageDetails = message;
    CauseOfError = cause;
    ErrorTimeStamp = time;
  }
  // Переопределение свойства Exception.Message.
  public override string Message
  {
    get
    {
      return string.Format("Car Error Message: {0}", messageDetails);
    }
  }
}
```

Фильтры исключений

```
try
{
   //Вызываем исключение
}
catch (ArgumentNullException ex) if (ex.Source == "EmployeeCreation")
{
   //Нотификация об ошибке
}
catch (InvalidOperationException ex) if (ex.InnerException != null)
{
   //Нотификация об ошибке
}
catch (Exception ex) if (ex.InnerException != null)
{
    //Сохраняем данные в лог
}
```