Исключения (Exceptions)

Исключительная ситуация (исключение) — это ошибка, которая возникает во время выполнения программы.

Исключением называется ситуация, когда член класса не в состоянии решить возложенную на него задачу.

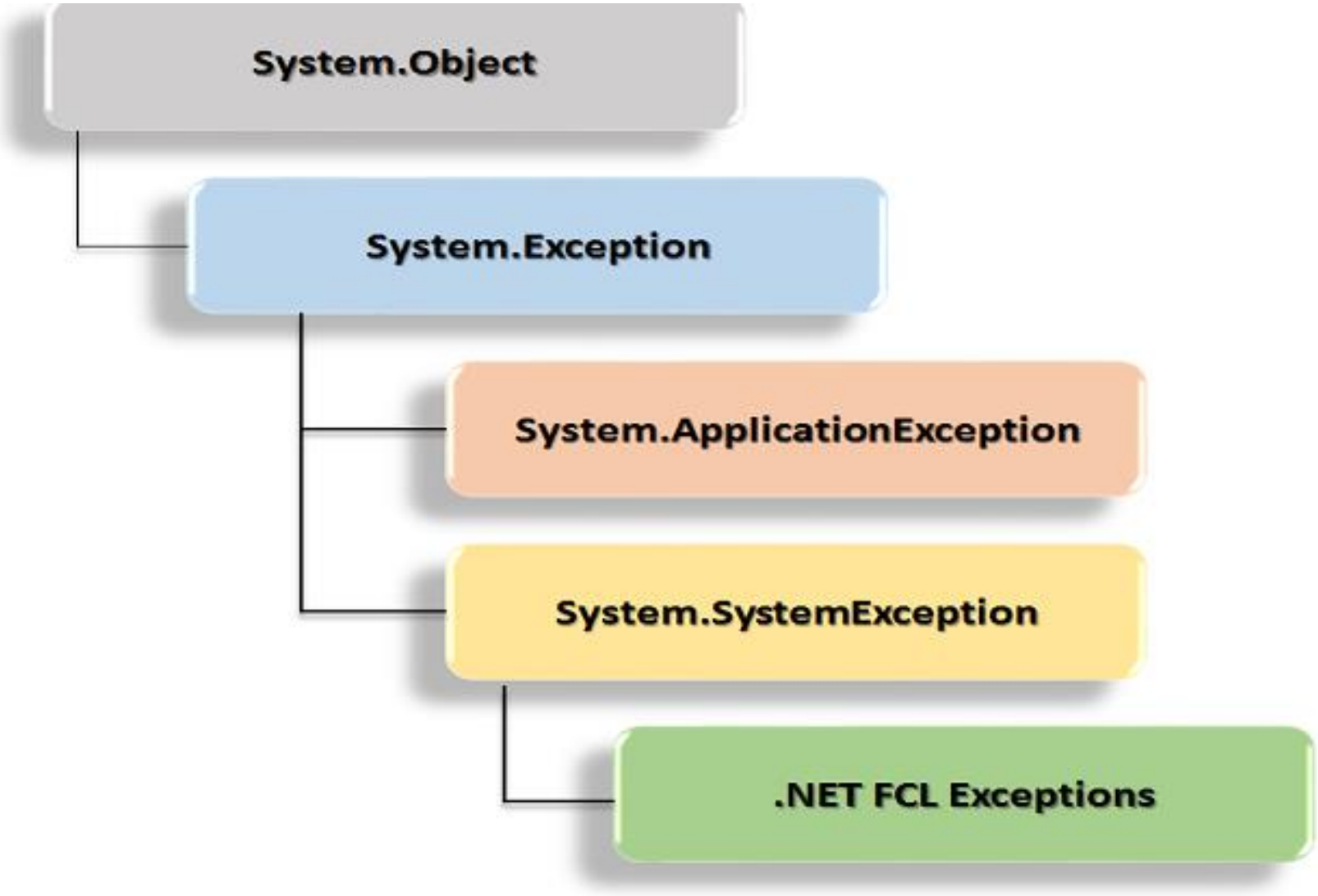
Особенности обработки исключительных ситуаций на C#

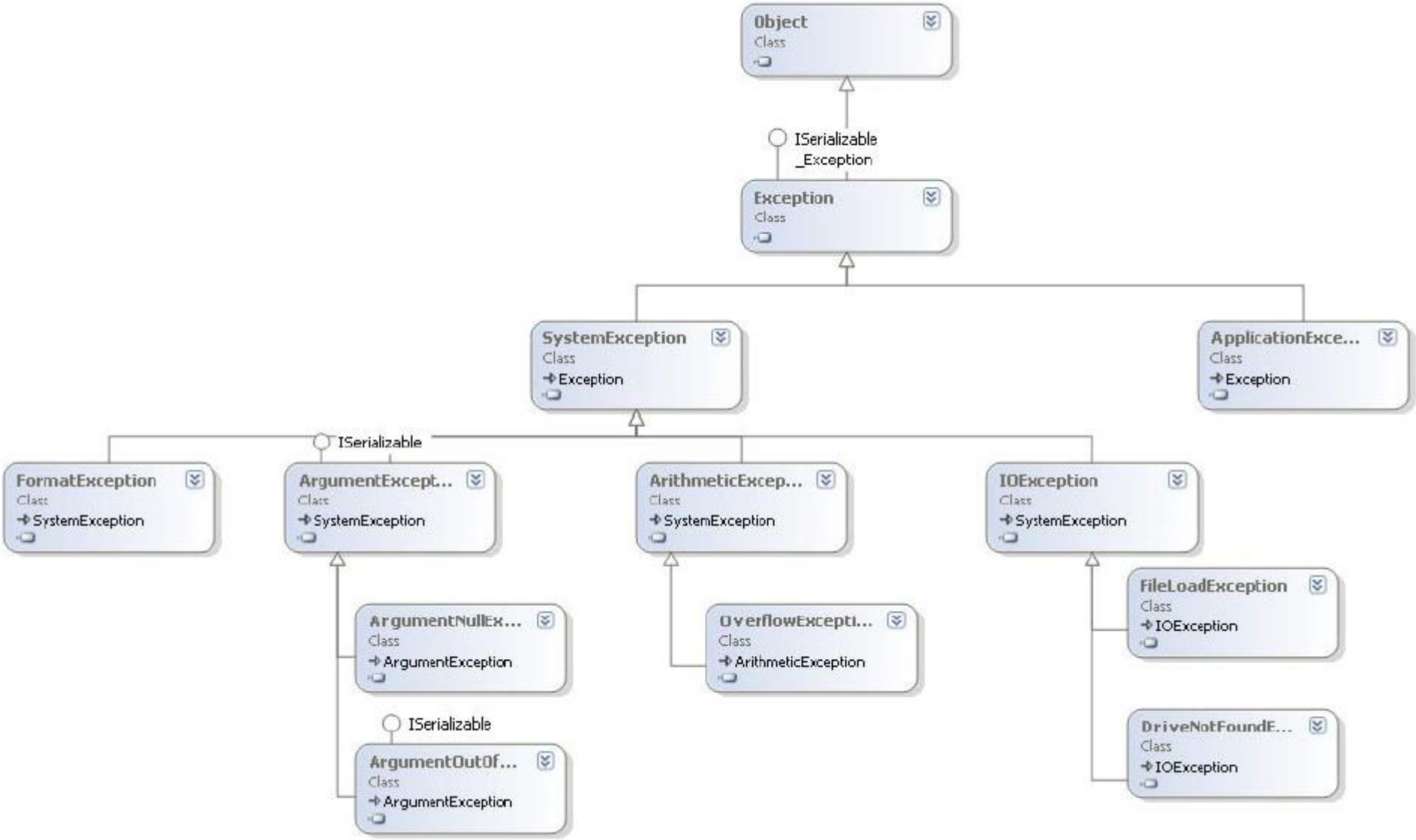
- В C# исключения представляются **классами**.
- Все классы исключений являются потомками класса исключений **Exception**, который является частью пространства имен **System**.

Иерархия исключений

Базовый класс System.Exception.

Анализ иерархии стандартных исключений.


System.Object
System.Exception
System.ApplicationException
System.SystemException
.NET FCL Exceptions



Некоторые классы исключений

Основы обработки исключений

Ключевое слово try.

Ключевое слово catch.

Ключевое слово throw.

Ключевое слово finally.

Формат записи try/catch-блоков обработки исключений:

```
try
{
      //  Блок кода, в котором может
      //  возникнуть исключение
}
catch [(тип исключения)]
{
     // Обработчик исключения.
}
```

```
Console.WriteLine("Простое исключение");

try
// Блок кода, где может произойти
// исключительная ситуация
{
 int a=Int32.Parse(Console.ReadLine());
 int b=Convert.ToInt32(Console.ReadLine());
 Console.WriteLine("Сумма чисел равна "+(a + b));
}
catch
{
  // Блок кода, который должен обработать
  // исключительную ситуацию
Console.WriteLine("Проверьте тип введенных данных!");
}
```

```
Console.Write(“Использование TryParse()");

int a;
if (!Int32.TryParse(Console.ReadLine(), out a))
// сокращенная запись
    {
      Console.Write("тип введенных данных!");
       return;
    }

int b;
Int32.TryParse(Console.ReadLine(),out b);
Console.WriteLine(a+b);
```

Свойства класса System.Exception

Название свойства	Описание
string Message	Содержит текст сообщения с указанием причины возникновения исключения
string Source	Содержит имя сборки, сгенерировавшей исключение.
string StackTrace	Содержит имена и сигнатуры методов, вызов которых привел к возникновению исключения.
string HelpLink	Содержит URL документа с описанием исключения
MethodBase TargetSite	Содержит метод, сгенерировавший исключение.

Порядок использования свойств класса System.Exception

```
try
{
  int a=Int32.Parse(Console.ReadLine());
  int b=Convert.ToInt32(Console.ReadLine());
}
catch
{
  Console.WriteLine(“Ошибка формата данных”);
}
```

Порядок использования свойств класса System.Exception

```
try
{
  int a=Int32.Parse(Console.ReadLine());
  int b=Convert.ToInt32(Console.ReadLine());
}
catch (Exception ex) // перехват исключений любого характера
{
  Console.WriteLine(ex.Message);
  // Содержит имя сборки, сгенерировавшей исключение.
  Console.WriteLine(ex.Source);
  // Содержит URL документа с описанием исключения
  Console.WriteLine(ex.HelpLink);
  // Содержит имена и сигнатуры методов, вызов которых привел к
  // возникновению исключения.
 Console.WriteLine(ex.StackTrace);
  //Содержит метод, сгенерировавший исключение.
 Console.WriteLine(ex.TargetSite);
}
```

```
Вывод сообщений об исключении
oi
--------------------------------------------------------------------------------

Входная строка имела неверный формат.
------------------------------------------
mscorlib
------------------------------------------

------------------------------------------
   в System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer
& number, NumberFormatInfo info, Boolean parseDecimal)
   в System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo inf
o)
   в System.Int32.Parse(String s)
   в Example3.Program.Main(String[] args) в g:\_Работа (нов)\_ITstep\_Уч. матери
ал\1. C#\Day5 Обработка исключений\_Examples(Exceptions)\Exceptions\Example\Prog
ram.cs:строка 14
------------------------------------------
Void StringToNumber(System.String, System.Globalization.NumberStyles, NumberBuff
er ByRef, System.Globalization.NumberFormatInfo, Boolean)
------------------------------------------
Press any key to continue . . .
```

Порядок использования свойств класса System.Exception

```
class MyClass
{  public void MyMethod()
    {
        Exception exception = new Exception("Мое исключение");
        exception.HelpLink = "http://MyCompany.com/ErrorService";
        exception.Data.Add("Причина исключения: ", "Тестовое исключение");
        exception.Data.Add("Время возникновения исключения: ", DateTime.Now);
        throw exception;
    }
}
class Program
{ static void Main()
    {  try
        {   MyClass instance = new MyClass();
            instance.MyMethod();
        }
        catch (Exception e)
        {
            Console.WriteLine("Имя члена:                {0}", e.TargetSite);
            Console.WriteLine("Класс :                   {0}", e.TargetSite.DeclaringType);
            Console.WriteLine("Тип члена:                {0}", e.TargetSite.MemberType);
            Console.WriteLine("Message:                  {0}", e.Message);
            Console.WriteLine("Source:                   {0}", e.Source);
            Console.WriteLine("Help Link:                {0}", e.HelpLink);
            Console.WriteLine("Stack:                    {0}", e.StackTrace);
            foreach (DictionaryEntry de in e.Data)
                Console.WriteLine("{0} : {1}", de.Key, de.Value);
        }
}}
```

```
Имя члена:                    Void MyMethod()
Класс определяющий член: Exceptions.MyClass
Тип члена:                    Method
Message:                      Мое исключение
Source:                       ConsoleApplication19
Help Link:                    http://MyCompany.com/ErrorService
Stack:                           в Exceptions.MyClass.MyMethod() в e:\!kin\!Other\Con
soleApplication19\ConsoleApplication19\Program.cs:строка 21
   в Exceptions.Program.Main() в e:\!kin\!Other\ConsoleApplication19\ConsoleAppl
ication19\Program.cs:строка 32
Причина исключения:  : Тестовое исключение
Время возникновения исключения:  : 12.09.2016 17:51:57
Для продолжения нажмите любую клавишу . . . _
```

```
try
{
      //  Блок кода, в котором может
      //  возникнуть исключение
}
catch (тип исключения)
{
     // Обработчик исключения.
}
finally
{
     // освобождение ресурсов
}   …
```

Блок finally

```
try
// Блок кода, где может произойти исключительная ситуация
{
     Console.WriteLine("Блок try до генерации исключения");
     Console.Write("Генерировать исключение? (1-да/0-нет):");
     if (Console.ReadLine() == "1")
         throw new IndexOutOfRangeException();
     Console.WriteLine("Блок try после генерации исключения");
}
catch(Exception )
// Блок кода, который должен обработать исключительную ситуацию
{
    Console.WriteLine("Блок catch ");
}
finally
{
// Блок кода. который выполняется как при наличии исключения,
// так и без исключения!
    Console.WriteLine("Блок finally ");
}
```

Использование блока try/catch

```
try
{
    [Try block.]
}
catch ([catch specification 1])
{
    [Catch block 1.]
}
...
catch ([catch specification n])
{
    [Catch block n.]
}
finally
{
    [Finally block.]
}
```

В блок **try** обычно помещается ***код, требующий корректного восстановления, генерации ошибки и/или очистки неуправляемых ресурсов.***

1. Код обработки определенного исключения
2. Логгирования
3. Восстановлении
4. Проброс ошибки

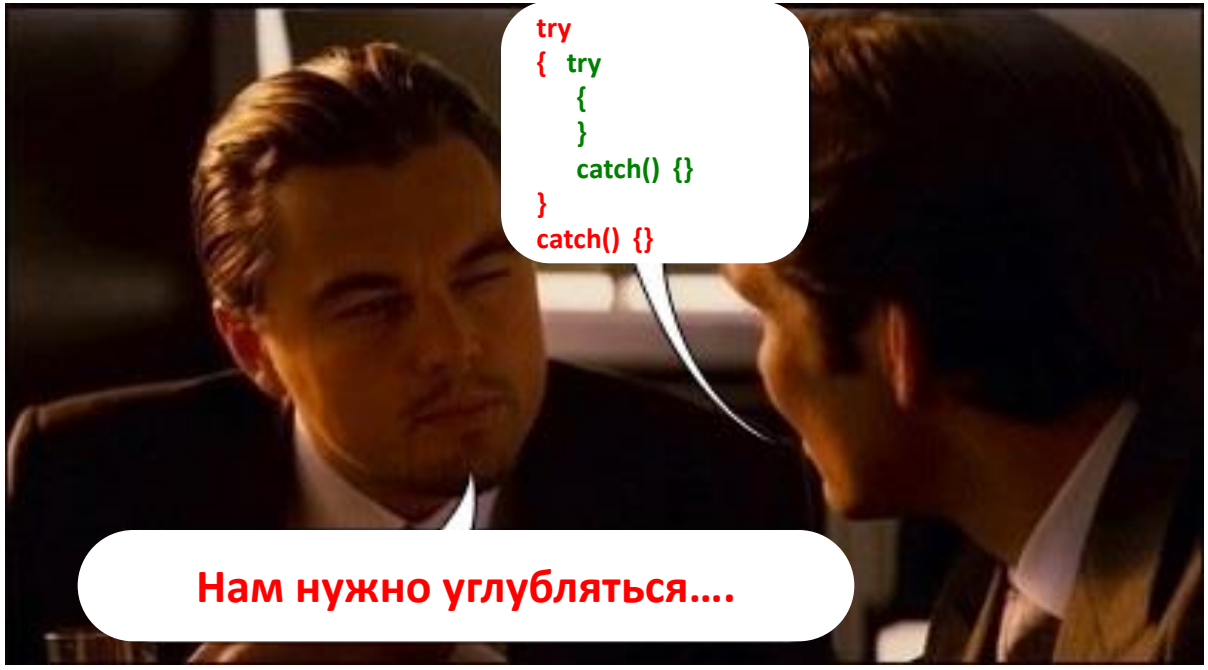
Код выполняющий очистку не управляемых ресурсов после операций, начатых в блоке try.

Тонкости обработки исключений

Перехват всех исключений.

Вложенные блоки try.

Повторное генерирование исключений.



```
try
{
  // Блок кода, проверяемый на наличие ошибок.
}
catch (ExcepType1 exOb)
{
  // Обработчик исключения типа ExcepType1.
}
catch (ExcepType2 exOb)
{
    // Обработчик исключения типа ExcepType2.
}
...
```

```
try
{  Console.Write("Введите число :  ");
   string s = Console.ReadLine();
   int n = Convert.ToInt32(s);
   if (n <= 0)
   {   // генерируется исключение ArgumentOutOfRangeException
       // "n <= 0" - сообщение, которое выведется при исключении
      throw new ArgumentOutOfRangeException("n <= 0");
   }
   double f = Math.Log(n);
   int d = 100 / (int)f;
   Console.WriteLine("d = {0} f = {1}", d, f);
}
catch (FormatException)
   {   //происходит, если введенное пользователем значение некорректно
       Console.WriteLine("FormatException");
   }
catch (DivideByZeroException)
   {
      //происходит, если  Log(1) = 0  - 100/0 Деление на ноль
      Console.WriteLine("DivideByZeroException");
   }
catch (Exception e)
   {
      // перехват всех  остальных  исключений
      Console.WriteLine("Exception: {0}", e.Message);
   }
```

Вложенные блоки try-catch

```
try
{      // Вложенный блок try-catch
        try
        {
            //…
        }
       catch (ExcepType ex1)
       {    //…        }
      finally
       {                }
}
catch (ExcepType ex2)
{
   //…
}
```

Вложенные блоки try-catch

```
try
{
Console.WriteLine("Блок try до генерации исключения");
  try
  {
    Console.WriteLine(" ---- Влож. блок try  до исключения");
    switch (Console.ReadLine())
     {
        case "1": throw new IndexOutOfRangeException();          break;
        case "2": throw new DivideByZeroException();             break;
        default: Console.WriteLine("Нет генерации исключений"); break;
     }
    Console.WriteLine(" ---- Влож. блок try  - конец");
  }
  catch (IndexOutOfRangeException ex)
  {   Console.WriteLine(" ---- Влож. блок catch ");   }
  finally
  {   Console.WriteLine(" ---- Влож. блок finally "); }
  Console.WriteLine("Блок try  - конец");
}
catch (Exception ex)
{  Console.WriteLine("Блок catch ");   }
finally
{  Console.WriteLine("Блок finally "); }
```

Повторная генерация исключений Часть 1

```
static void Main(string[] args)
{   try
    {       Test1();    }
    catch (Exception ex)
    {       Console.WriteLine(ex.Message);    }
}
```

```
static void Test1()
{
 try
 {
 Console.WriteLine(" * * * Генерация IndexOutOfRangeException");
 throw new IndexOutOfRangeException();
 }
 catch (IndexOutOfRangeException ex)
 {
   Console.WriteLine(" * * * Обработка IndexOutOfRangeException");
   // повторная генерация исключения (типа IndexOutOfRangeException)
   // для обработки catch более высокого уровня!
   throw;
 } }
```

```
 * * * Генерация исключения  IndexOutOfRangeException
 * * * Обработка исключения  IndexOutOfRangeException
Индекс находился вне границ массива.
```

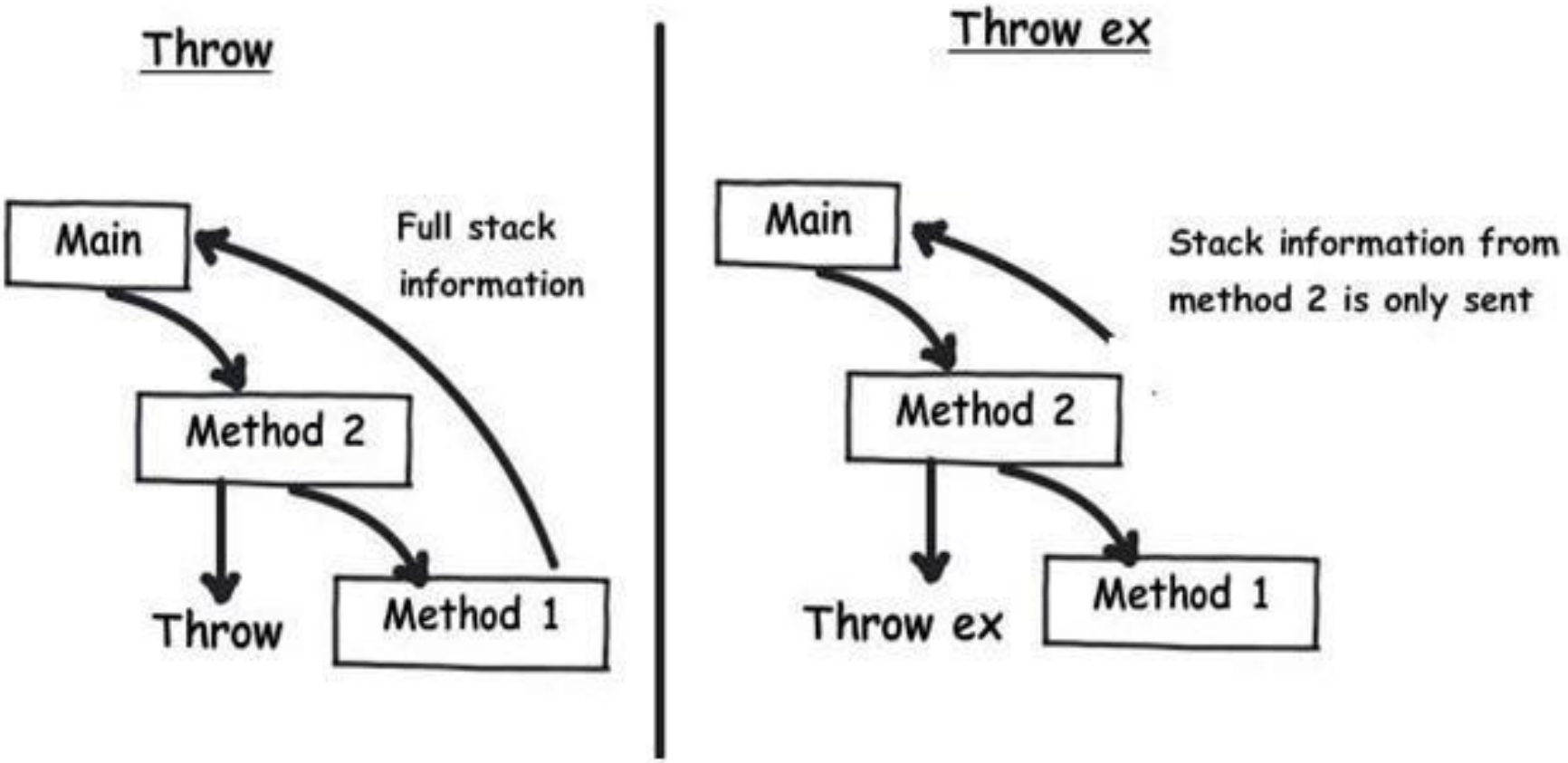
Повторная генерация исключений Часть 2

```
static void Main(string[] args)
{   try
    {       Test1();    }
    catch (Exception ex)
    {       Console.WriteLine(ex.Message);    }
}
```

```
static void Test1()
{
 try
 {
   Console.WriteLine(" * * * Генерация IndexOutOfRangeException");
   throw new IndexOutOfRangeException();
 }
 catch (IndexOutOfRangeException ex)
 {
   Console.WriteLine(" * * * Обработка IndexOutOfRangeException");
   // повторная генерация исключения
   throw new DivideByZeroException("Деление на ноль");
} }
```

```
 * * * Генерация исключения  IndexOutOfRangeException
 * * * Обработка исключения  IndexOutOfRangeException
Деление на ноль
```

Передача по стеку вызовов (отличие throw от throw ex)



Передача по стеку вызовов (отличие throw от throw ex)

```
class Program
   {  static void Main(string[] args)
       {
           try
           {
               Method2();
           }
           catch (Exception ex)
           {   Console.Write(ex.StackTrace.ToString());
               Console.ReadLine();
           }
       }
       static void Method2()
       {
           try
           {   Method1();     }
           catch (Exception ex)
           { throw ex;          // throw;
            }
       }
       public  static void Method1()
       {
           try
           { throw new Exception("This is thrown from Method1");
           }
           catch (Exception ex)
           { throw;    }
       }
   }
```

```
   в DiffBetweenThrowAndThrowEx.Program.Method2() в E:\ex\DiffBetweenThrowAndThr
owEx\Program.cs:строка 30
   в DiffBetweenThrowAndThrowEx.Program.Main(String[] args) в E:\ex\DiffBetweenT
hrowAndThrowEx\Program.cs:строка 14_
```

```
   в DiffBetweenThrowAndThrowEx.Program.Method1() в E:\ex\DiffBetweenThrowAndThr
owEx\Program.cs:строка 41
   в DiffBetweenThrowAndThrowEx.Program.Method2() в E:\ex\DiffBetweenThrowAndThr
owEx\Program.cs:строка 30
   в DiffBetweenThrowAndThrowEx.Program.Main(String[] args) в E:\ex\DiffBetweenT
hrowAndThrowEx\Program.cs:строка 14
```

Вложенные исключения

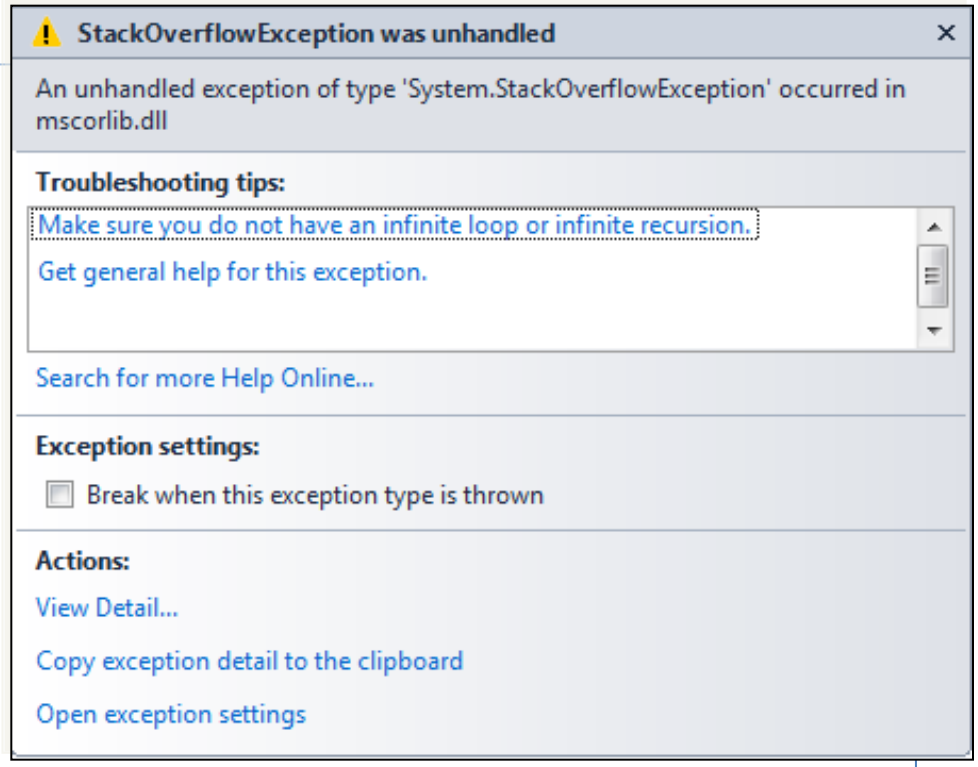
```
public class ClassWithException
    {
        public void ThrowInner()
        { throw new Exception("Это внутреннее исключение!"); }

        public void CatchInner()
        {
            try
            {
                this.ThrowInner();
            }
            catch (Exception e)
            {  throw new Exception("Это внешнее исключение!", e);   }
} }
```

```
static void Main()
 {
   ClassWithException instance = new ClassWithException();
   //instance.CatchInner(); // Попробовать вызвать.
   try
   {      instance.CatchInner(); }
   catch (Exception ex)
   {  Console.WriteLine("Exception : {0}", ex.Message);
      Console.WriteLine("Inner : {0}",     ex.InnerException.Message);
 }}
```

Необрабатываемые исключения

```
class Program
{
    public static void Method()
    {
        int[] arr = new int[10];
        Console.WriteLine(arr);
        Method(); // рекурсия
    }
    static void Main()
    {   try
        {
            Method();
        }
        catch (Exception ex)
        {  Console.WriteLine(ex.Message); }
        finally
        {
            // finally - не сработает при ошибке.
            Console.WriteLine("Finally");
        }
}}
```



Исключение OutOfMemoryException.

```
class Program
{  public static void Method()
    {
        int[] arr = new int[100000000];
        Console.WriteLine(arr);
        Method();
    }

static void Main()
    {   try
        {
            Method();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        finally
        {
            // finally - сработает.
            Console.WriteLine("Finally");
        }
}}
```

Создание пользовательского исключения.

```
public class UserException:Exception
{
 // конструкторы
 public UserException ()
  {
     // ...
  }

   // конструкторы для инициализации св-ва Message
 public UserException (string message) : base(message)
 {
   // ...
 }
}
```

Применение конструкций checked и unchecked.

```
Byte b1 = 100; // byte хранит числа от 0 до 255 (всего 256)
//b1 =300-256= 44 – разряды, которые  не попадают отбрасываются
b1 = (Byte)(b1 + 200);
Console.WriteLine("Результат: "+b1);

Byte b2 = 100;
try
{ // контроль за переполнением и генерация исключения
   b2 = checked((Byte)(b2 + 200));
   Console.WriteLine("Результат checked: " + b2);
}
catch(OverflowException ex)
{    Console.WriteLine(ex.Message);            }

Byte b3 = 100;
try
{  // игнорирование переполнения
   b3 = unchecked((Byte)(b3 + 200));
   Console.WriteLine("Результат unchecked: " + b3);
}
catch (OverflowException ex)
{    Console.WriteLine(ex.Message);            }
```

```
Использование checked/unchecked
Результат: 44
Переполнение в результате выполнения арифметической операции.
Результат: 44
Press any key to continue . . .
```

Фильтры исключений

```
try
{
    throw new Exception("1");
}
catch (Exception e) when (e.Message == "0")
{
    Console.WriteLine("This is 0 block");
}
catch (Exception e) when(e.Message =="1")
{
    Console.WriteLine("This is 1 block");  ≤1ms elapsed
}
catch (Exception e) when (e.Message == "2")
{
    Console.WriteLine("This is 2 block");
}
```

Генерация исключений

«**Принцип самурая**» это принцип разработки, призванный описать «контракт» между функцией и вызывающим ее кодом и заключается в следующем. Любая функция, реализующая некоторую единицу работы должна следовать тому же кодексу чести «бусидо», по которому живет любой самурай. Так, самурай не будет выполнять никаких заданий, противоречащих его «кодексу чести» и если к нему подойти с «непристойным» предложением, то он снесет вам башку раньше, чем вы успеете глазом моргнуть. Но если уж самурай возьмется за дело, то можно быть уверенным в том, что он доведет его до конца «сделай или умри».

Генерация исключений

Генерировать исключения необходимо при выполнении одного или нескольких из следующих условий

Метод не способен выполнить свои определенные функции

```
static void CopyObject(SampleClass original)
{
    if (original == null)
    {
       throw new ArgumentNullException("original");
    }
}
```

Генерация исключений

Аргумент метода вызывает исключение

```
static int GetValueFromArray(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (IndexOutOfRangeException ex)
    {
        ArgumentException argEx = new ArgumentException("Index is out of
range", "index", ex);
        throw argEx;
    }
}
```

Генерация исключений

На основе состояния объекта выполнен неправильный вызов объекта

```
class ProgramLog
{
    FileStream logFile = null;
    void OpenLog(FileInfo fileName, FileMode mode) { }
    void WriteLog()
    {
        if (!this.logFile.CanWrite)
        {
            throw new InvalidOperationException("Logfile cannot be read-
only");
        }
        // Else write data to the log and return.
    }
}
```

Как правильно использовать исключения

Проверяйте аргументы своих методов

Для типов, которые являются частью библиотеки повторно используемых классов, настоятельно рекомендуется, чтобы в открытых типах присутствовала проверка правильности параметров открытых и защищенных методов, до начала выполнения какой-либо операции методом

System.ArgumentException
- **System.ArgumentNullException**
- **System.ArgumentOutOfRangeException**
- **System.DuplicateWaitObjectException**

Рекомендации по обработке и генерации исключений

Следует генерировать исключение, соответствующее обнаруженному ошибочному условию

Логика приложения не должна полагаться на логику блоков try и catch для работы в рамках неисключительных условий

При определении нескольких блоков catch, следует упорядочивать их спецификации от наиболее конкретных к наименее конкретным

Следует прехватывать исключения и детализировать сообщения для диагностических целей, а затем отображать удобные для пользователя сообщения

Не желательно показывать пользователю слишком подробную информацию об исключении, поскольку она может быть использована злонамеренными пользователями для вывода приложения из строя или получения доступа к защищенной информации

Эффективная обработка исключения должна позволить приложению восстановиться после возникновения исключения, а пользователю – продолжить пользоваться приложением