Исключения

Исключительные ситуации в C#

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
{

}
catch
{

}
finally
{

}
```

- 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 (ExcepType1 exOb)
        {
    // Обработчик исключения типа ExcepType1.
        }
catch (ExcepType2 exOb)
        {
   // Обработчик исключения типа ExcepType2.
        }
```

тип исключения (catch type)

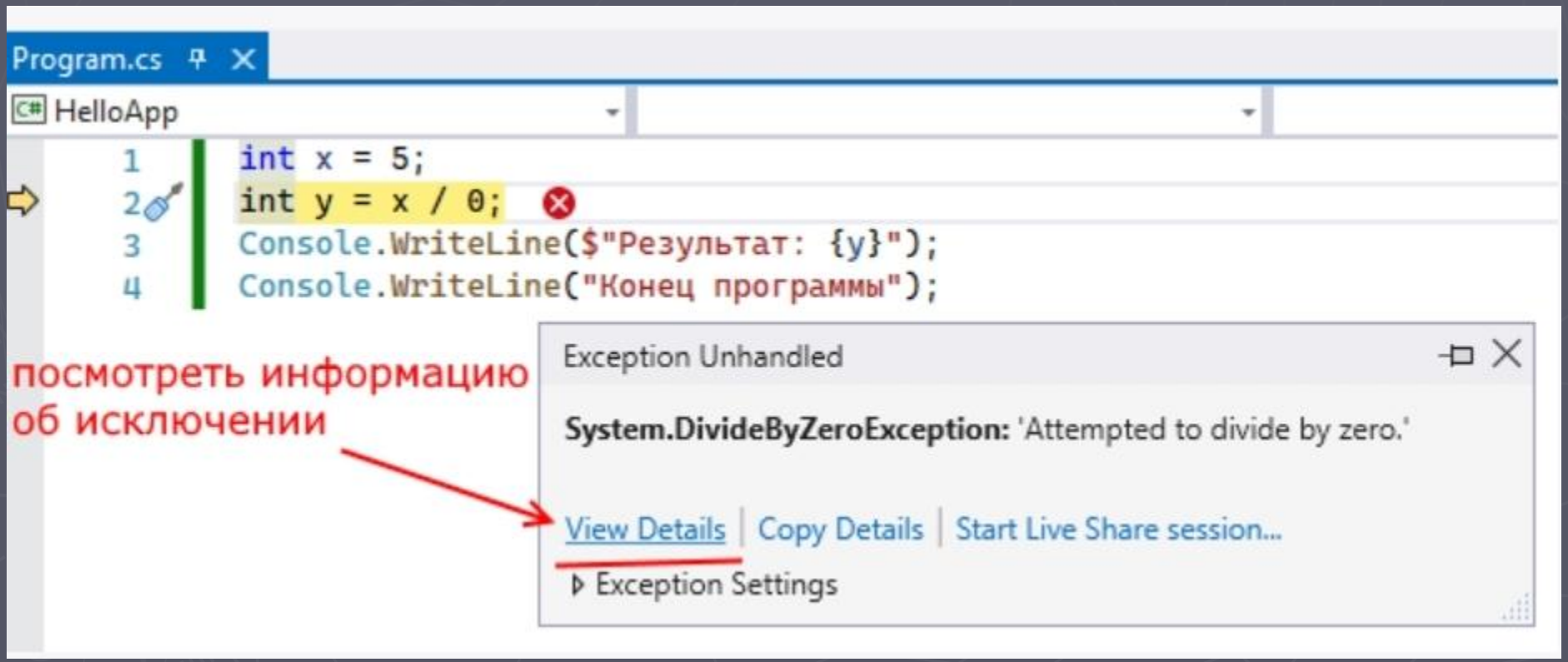
Если код в блоке try не порождает исключение, CLR никогда не переходит к выполнению кода в соответствующем блоке catch

```
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();
           }
```

источник исключения -> catch или finally: CLR продолжает работу, теряется информация о первом исключении, вброшенном в блоке try. Если исключение останется необработанным CLR завершает процесс

только один блок finally

```
int x = 5;
int y = x / 0;
Console.WriteLine($"Результат: {y}");
Console.WriteLine("Конец программы");
```



С помощью пункта View Details можно посмотреть более детальную информацию об исключении

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch
{
    Console.WriteLine("Возникло исключение!");
}
finally
{
    Console.WriteLine("Блок finally");
}
Console.WriteLine("Конец программы")
```

```
Возникло исключение!
Блок finally
Конец программы
```

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch
{
    Console.WriteLine("Возникло исключение!");
}
```

обязателен блок try. При наличии блока catch мы можем опустить блок finally

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
finally
{
    Console.WriteLine("Блок finally");
}
```

Если CLR не сможет найти нужный блок catch, то исключение не будет обработано, и программа аварийно завершится

при наличии блока finally мы не можем опустить блок catch

```
catch
{
    // выполняемые инструкции
}
```

Обрабатывает любое исключение, которое возникло в блоке try

```
catch (тип_исключения)
{
    // выполняемые инструкции
}
```

Обрабатывает только те исключения, которые соответствуют типу, указаному в скобках после оператора catch.

```
catch (тип_исключения имя_переменной)
{
    // выполняемые инструкции
}
```

вся информация об исключении помещается в переменную данного типа

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

Имя метода

Текст с описанием причины исключения

Адрес документации с информацией об исключении

Exception

InnerException: хранит информацию об исключении, которое послужило причиной текущего исключения

Message: хранит сообщение об исключении, текст ошибки

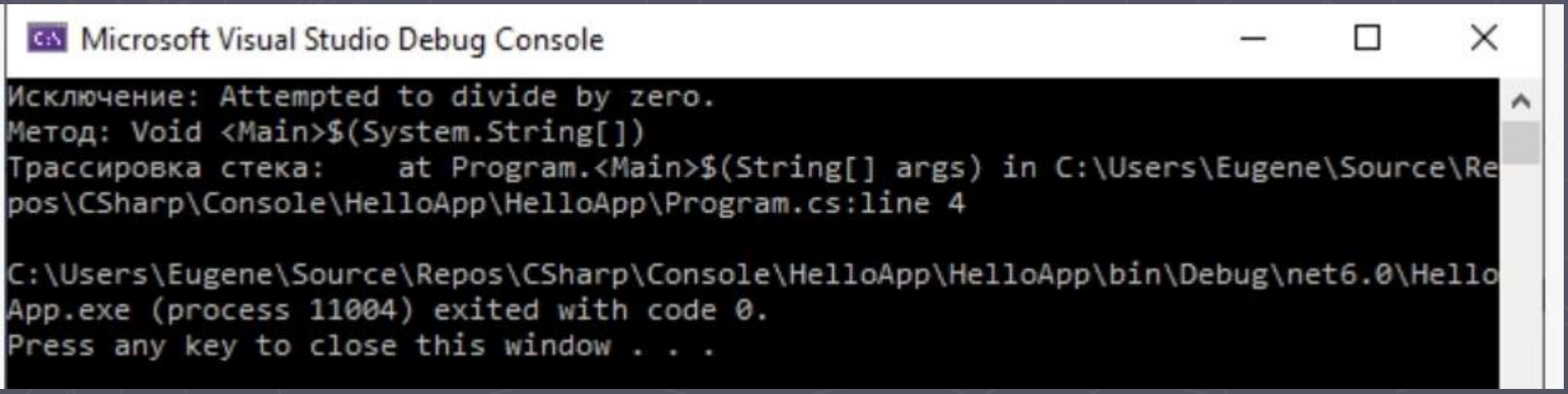
Source: хранит имя объекта или сборки, которое вызвало исключение

StackTrace: возвращает строковое представление стека вызывов, которые привели к возникновению исключения

TargetSite: возвращает метод, в котором и было вызвано исключение

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch (Exception ex)
{
    Console.WriteLine($"Исключение: {ex.Message}");
    Console.WriteLine($"Метод: {ex.TargetSite}");
    Console.WriteLine($"Трассировка стека: {ex.StackTrace}");
}
```

Exception является базовым типом для всех исключений, то выражение catch (Exception ex) будет обрабатывать все исключения, которые могут возникнуть



DivideByZeroException: представляет исключение, которое генерируется при делении на ноль

ArgumentOutOfRangeException: генерируется, если значение аргумента находится вне диапазона допустимых значений

ArgumentException: генерируется, если в метод для параметра передается некорректное значение

IndexOutOfRangeException: генерируется, если индекс элемента массива или коллекции находится вне диапазона допустимых значений

InvalidCastException: генерируется при попытке произвести недопустимые преобразования типов

NullReferenceException: генерируется при попытке обращения к объекту, который равен null (то есть по сути неопределен)

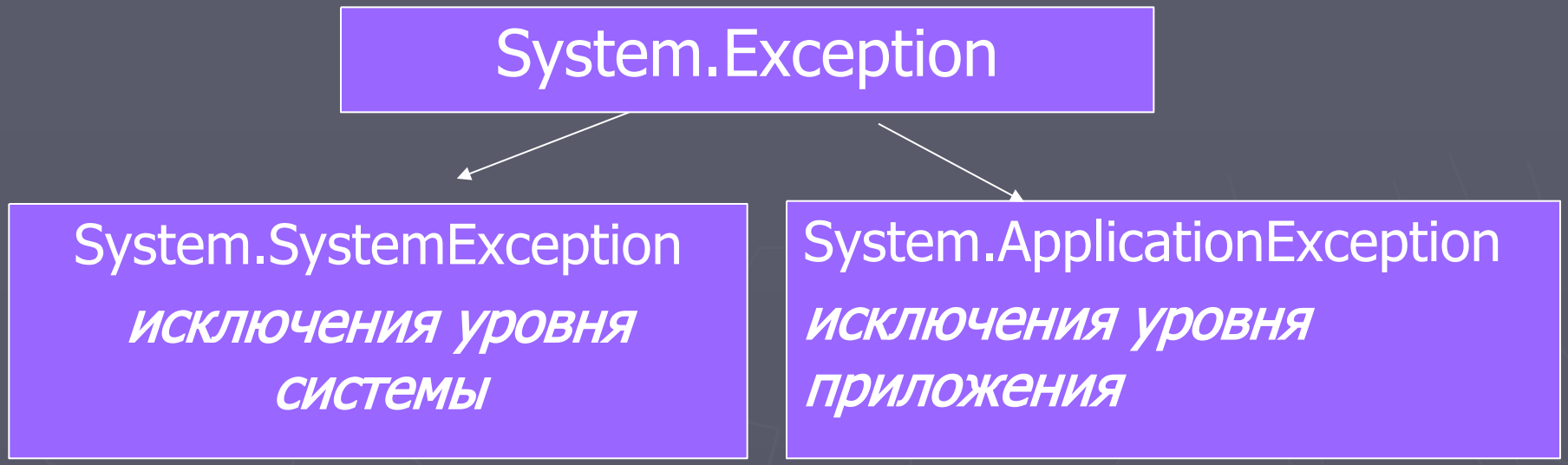
```
static void Main(string[] args)
{
    try
    {
        int[] numbers = new int[4];
        numbers[7] = 9;      // IndexOutOfRangeException

        int x = 5;
        int y = x / 0;  // DivideByZeroException
        Console.WriteLine($"Результат: {y}");
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("Возникло исключение DivideByZeroException");
    }
    catch (IndexOutOfRangeException ex)
    {
        Console.WriteLine(ex.Message);
    }

    Console.Read();
}
```

исключение типа DivideByZeroException никогда не будет сгенерировано.

```
try
{
    object obj = "you";
    int num = (int)obj;      // System.InvalidCastException
    Console.WriteLine($"Результат: {num}");
}
catch (DivideByZeroException)
{
    Console.WriteLine("Возникло исключение DivideByZeroException");
}
catch (IndexOutOfRangeException)
{
    Console.WriteLine("Возникло исключение IndexOutOfRangeException");
}
catch (Exception ex)
{
    Console.WriteLine($"Исключение: {ex.Message}");
}
```

```
try
{
    object obj = "you";
    int num = (int)obj;      // System.InvalidCastException
    Console.WriteLine($"Результат: {num}");
}
catch (DivideByZeroException)
{
    Console.WriteLine("Возникло исключение DivideByZeroException");
}
catch (IndexOutOfRangeException)
{
    Console.WriteLine("Возникло исключение IndexOutOfRangeException");
}
catch (Exception ex)
{
    Console.WriteLine($"Исключение: {ex.Message}");
}
```

обрабатывает все исключения кроме DivideByZeroException и IndexOutOfRangeException



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

```
class PersonException : Exception
{
    public PersonException(string message)
        : base(message) { }
}
```

```
class Person
{
    private int age;
    public string Name { get; set; } = "";
    public int Age
    {
        get => age;
        set
        {
            if (value < 18)
                throw new PersonException("Лицам до 18 регистрация запрещена");
            else
                age = value;
        }
    }
}
```

```
try
{
    Person person = new Person { Name = "Tom", Age = 17 };
}
catch (PersonException ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
}
```

```
class PersonException : ArgumentException
{
    public int Value { get;}
    public PersonException(string message, int val)
        : base(message)
    {
        Value = val;
    }
}
```

```
try
{
    Person person = new Person { Name = "Tom", Age = 17 };
}
catch (PersonException ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
    Console.WriteLine($"Некорректное значение: {ex.Value}");
}

class Person
{
    private int age;
    public string Name { get; set; } = "";
    public int Age
    {
        get => age;
        set
        {
            if (value < 18)
                throw new PersonException("Лицам до 18 регистрация запрещена", value
            else
                age = value;
        }
    }
}
```

```
 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
                {
                    //...
                }
```

```
public class Exx{
        public void Move(){

                int x = 5;
                int y = x / 0;
     }
        public void Copy(){
               int x = 5;
               int y = x / 5;
               Console.WriteLine(y);}
   }
   private static void Main(string[] args)
   {
      try
       {
           Exx q= new Exx();
           q.Move();
           q.Copy();
       }
       catch { Console.WriteLine("Catch"); }
       finally { Console.WriteLine("Finally"); }
   }
```

```
Catch
Finally
```

```
public class Exx{
        public void Move(){
              try{
                 int x = 5;
                 int y = x / 0;}
            catch { Console.WriteLine("Exception in Move");}}
        public void Copy(){
               int x = 5;
                int y = x / 5;
                Console.WriteLine(y);}
    }
    private static void Main(string[] args)
    {
       try
        {
            Exx q= new Exx();
            q.Move();
            q.Copy();
        }
        catch { Console.WriteLine("Catch"); }
        finally { Console.WriteLine("Finally"); }
    }
```

```
Exception in Move
1
Finally
```

Повторная генерация исключения

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

В этом виде оператор throw может использоваться **исключительно в блоке catch**

```
try
{
    try
    {
        Console.Write("Введите строку: ");
        string message = Console.ReadLine();
        if (message.Length > 3)
        {
            throw new Exception("Длина строки больше 6
символов");//генерируем исключение
        }
    }
    catch
    {
        Console.WriteLine("Возникло исключение");
        throw;
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

исключение будет передано дальше внешнему блоку catch.

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`

```
int x = 1;
int y = 0;

try
{
    int result1 = x / y;
    int result2 = y / x;
}
catch (DivideByZeroException) when (y == 0)
{
    Console.WriteLine("y не должен быть равен 0");
}
catch(DivideByZeroException ex)
{
    Console.WriteLine(ex.Message);
}
```

В этом случае обработка исключения в блоке catch производится только в том случае, если условие в выражении when истинно

Противоположная ситуация

```
int x = 0;
int y = 1;

try
{
    int result1 = x / y;
    int result2 = y / x;
}
catch (DivideByZeroException) when (y == 0)
{
    Console.WriteLine("y не должен быть равен 0");
}
catch(DivideByZeroException ex)
{
    Console.WriteLine(ex.Message);
}
```

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 исключение 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 исключении 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/ищем 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