Исключительные ситуации

Исключение

- Исключительная ситуации или исключение (*exception*) – это возникновение в программе непредвиденного события, которое может порождаться некорректным использованием аппаратуры или неправильной работой программы.
- Типичные исключительные ситуации:
  - деление на ноль,
  - достижение конца файла,
  - переполнение в арифметических операциях,
  - обращение к несуществующему участку памяти.

Исключение

- Обычно эти ситуации приводят к завершению выполнения программы с системным сообщением об ошибке.
- Механизм обработки исключений дает программисту возможность, определить каким образом программа может продолжить выполняться после обнаружения исключения.

Суть механизма исключений

- Логическое разделение вычислительного процесса на две части - обнаружение аварийной ситуации и ее обработку:
  - функция, обнаружившая ошибку, может «не знать» что делать для ее исправления
  - функция, вызывающая ошибку, может «знать» что делать, но «не уметь» определить место обнаружения ошибки.

Достоинства

- Улучшение структуры программы.
- Облегчение работы с библиотечными функциями и многомодульными программами.
- Для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение, параметры и глобальные переменные
  - это особенно важно для конструкторов!

Исключения в C#

- В C# практически любое состояние, достигнутое в процессе выполнения программы, можно опередить как исключительную ситуацию (не только аварийную ситуацию).
  - if (a!=b) генерировать исключение???
- Однако это не имеет преимуществ перед другими решениями и не улучшает структуру и читаемость программы.
- Исключения C# не поддерживают обработку асинхронных событий: ошибки оборудования или прерывания (Ctrl+C) и др.

Способы генерации исключений

- Стандартные исключения генерирует среда выполнения, они являются потомками класса **System.Exception.**

- Исключение может определить программист с помощью конструкции throw.

Примеры стандартных исключений

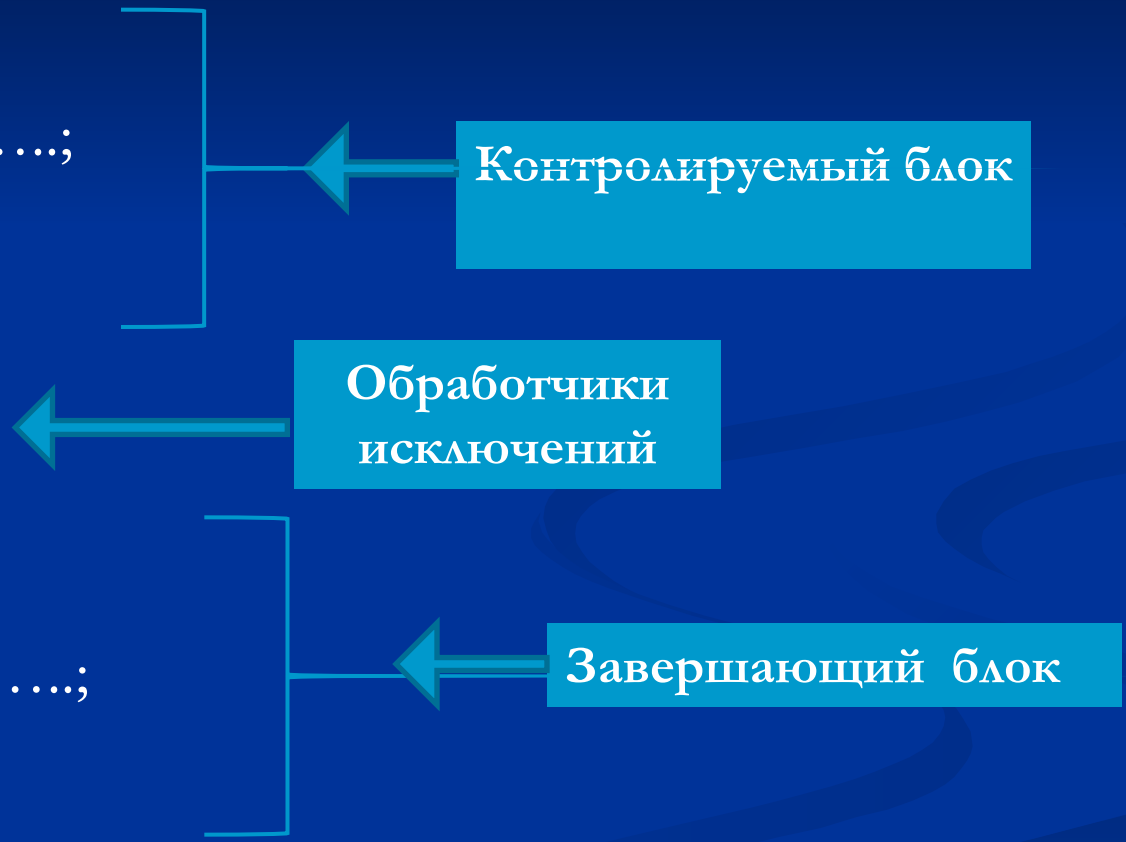
Имя	Описание
AritheticException	Ошибка в арифметических операциях или преобразованиях (является предком DivideByZeroException и OverFlowException)
ArrayTypeMismatchException	Попытка сохранения в массиве несовместимого типа
DivideByZeroException	Попытка деления на ноль
FormatException	Попытка передать в метод аргумент неверного формата
IndexOutOfRangeException	Индекс массива выходит за пределы диапазона
InvalidCastException	Ошибка преобразования типа
OutOfMemoryException	Недостаточно памяти для создания нового объекта
OverFlowException	Переполнение при выполнении арифметической операции
StackOverFlowException	Переполнение стека

Ключевые слова

- Для реализации механизма обработки исключений в C# используются следующие ключевые слова
  - try (контролировать)
  - catch (ловить)
  - finally (завершить)
  - throw (генерировать, порождать)

Синтаксис конструкции try

```
try
{
      оператор_1; …..;
      оператор_n;
}
 [блоки catch]
[finally
{
      оператор_1; …..;
      оператор_n;
}]
```



Блоки конструкции try

- **Блок try** - *контролируемого блок* - блок программы, в котором может генерироваться исключение.
- **Блоки catch** –*обработчики исключений* - определяет действия, которые должны выполниться в случае возникновения исключения.
- **Блок finally** – *завершающий блок* – выполняется вне зависимости от того возникло исключение или нет.

Контролируемый блок

- Любые описания
- Любые определения
- Любые операции
- Вызовы методов

которые потенциально могут генерировать исключения

- Специальные конструкции генерации исключений:

throw [выражение];

Конструкция throw [выражение];

- Выражение (параметр) - любой **объект, порожденный от стандартного класса Exception**.
- Именно этот объект используется для передачи информации об исключении его обработчику.
- Например,
  - throw new Exception ();
  - throw new Exception (“Исключение в функции F”);
- Выражение может отсутствовать только, если исключение генерируется внутри блока - обработчика исключения catch.

Свойства класса Exception

Имя	Описание
HelpLink	Возвращает или задает ссылку на файл справки, связанный с этим исключением.
HResult	Возвращает или задает HRESULT — кодированное числовое значение, присвоенное определенному исключению.
InnerException	Возвращает экземпляр объекта **Exception**, который вызвал текущее исключение.
Message	Возвращает сообщение, описывающее текущее исключение.
Source	Возвращает или задает имя приложения или объекта, вызывавшего ошибку.
StackTrace	Получает строковое представление непосредственных кадров в стеке вызова.
TargetSite	Возвращает метод, создавший текущее исключение.

Конструкция throw

- На практике часто конструкция throw находятся не в блоке try, а описывается в методах, прямо или косвенно, вызываемых в этом блоке

```
static void f () { … throw new Exception(“F”); …}
static void Main
{
try {… f(); ….}
catch () {….}

}
```

Генерация исключения

- выполнение контролируемого блока прекращается
- управление передается непосредственно за пределы контролируемого блока на соответствующий обработчик исключений (catch).

Обработка исключений

- Обработчик исключений должен непосредственно располагаться за try-блоком.
- Допускается наличие как одного, так и нескольких обработчиков (расположенных подряд).
- Если обработчиков несколько они должны отличаться типами исключений.
- В соответствии с типом переданного из контролируемого блока объекта-исключения и выбирается нужный обработчик.

Формы записи обработчика

1. catch (тип_исключения имя) {…}

- Роль имени подобна роли имени формального параметра в функции.
- Предполагается, что имя будет использоваться в операторах обработки исключения.

2. catch (тип_исключения) {…}

- В данном варианте не преподается использования значения исключения.
- Для обработчика важен только его тип и факт получения.

Формы записи обработчика

3. catch {…}

- Данный обработчик реагирует на любое исключение независимо от его типа.
- Так как сравнение сгенерированного исключения со спецификациями обработчиков проводится последовательно, данный тип обработчика следует помещать только в конец списка обработчиков.
- В противном случае он перехватил бы все исключение и написание последующих обработчиков было бы бессмысленно.

Обработка исключений

- обработчик исключения отсутствует + в процессе выполнения блока try возникло исключение

- метод, сгенерировавший исключение, вызван вне блока try

завершение программы с выводом системного сообщения.

Обработка исключений

- После обработки исключения

- Если исключение не возникло

Управление предается

• первому оператору блока finally (при наличии)

или

•первому оператору находящемуся непосредственно за обработчиками исключений

Блок finally

- обычно содержит операторы, которые выполнятся вне зависимости от генерации исключений:
  - закрытие файлов, которые были открыты в контролируемом блоке,
  - вывод информации.

Пример 1. Обработка стандартных исключений

```
//program_exc1.cs
using System;
 class ExceptionTestClass
{
   static void Main()
  {
     int x;
     Console.Write("Enter int x : ");
     string buf=Console.ReadLine();
```

Пример 1. Обработка стандартных исключений

```
try
    {
      x = int.Parse(buf);
      int y = 100 / x;
      Console.WriteLine("Result : " + y);
    }
    catch (FormatException e)
    {
       Console.WriteLine("FormatException : {0}\nFile:
{1}\nLocation : {2} ", e.Message, e.Source, e.TargetSite);
    }
```

Пример 1. Обработка стандартных исключений

```
catch (ArithmeticException e)
      {
        Console.WriteLine("ArithmeticException : {0}\nFile:
{1}\nLocation : {2} ", e.Message, e.Source, e.TargetSite);
      }
      catch (Exception e)
      {
        Console.WriteLine("Generic Exception : {0}\nFile:
{1}\nLocation : {2} ", e.Message, e.Source, e.TargetSite);
      }
      Console.ReadKey();
    }
}
```

Варианты работы программы

- Например, при вводе буквы на экране появиться сообщение следующего вида:

  FormatException: Попытка передать в метод параметр неверного формата

  File : mscorlib

  Location : Void StringToNumber(…)

Варианты работы программы

- При вводе нуля на экране появиться сообщение следующего вида:

ArithmeticException : Попытка деления на ноль.

File : program_exc1

Location : Void Main()

Варианты работы программы

- Третий из обработчиков перехватит любое из исключений, возникших в контролируемом блоке.
- Если исключение в данной программе не будет сгенерировано, то на экран будет выведен результат деления 100/x.

Пример 2. Метод НОД

```
using System;
 class ExceptionTestClass1
 {
      static int GCM(int x, int y)
       {
            if (x==0 || y==0) throw new Exception("\nZero!");
            if (x<0) throw new Exception ("\nNegative parametr 1");
            if (y<0) throw new Exception ("\nNegative perametr 2");
          while (x!=y)
           {
                    if (x>y) x-=y;
                    else y-=x;
             }
          return x;
 }
```

Пример 2. Метод НОД

```
static void Main()
        {
            try
            {
                Console.WriteLine("\nCGM(66,44) = "+ GCM(66, 44));
                Console.WriteLine("\nCGM(0,7) = " + GCM(0, 7));
                Console.WriteLine("\nCGM(-12,8) = " + GCM(-12, 8));
            }
            catch (Exception exc) { Console.WriteLine(exc.Message); }
            finally {Console.WriteLine("\nFinally"); }

            Console.ReadKey();
        }
 }
```

Результат работы программы

CGM(66,44)=22

Zero!

Finally

Распространение исключений

- Блоки try могут вкладываться друг в друга (без ограничений на уровень вложенности).
- Исключение, сгенерированное во внутреннем блоке и не перехваченное соответствующем оператором catch, передается на верхний уровень, где продолжается поиск обработчика.
- Этот процесс называется распространением исключения.

Механизм обработки исключений

- Когда с помощью throw генерируется исключение выполняются следующие действия:
  - создается копия параметра throw в виде статического объекта, который существует до тех пор, пока исключение не будет обработано;
  - в поисках походящего обработчика «раскручивается» стек, вызывая деструкторы локальных объектов, выходящих из области действия;
  - объект и управление передается обработчику, имеющему параметр, совместимый по типу с типом объекта.

Механизм обработки исключений

- Раскручиванием стека называется процесс освобождения памяти из-под локальных переменных и возврата управления вызывающей функции.
- При раскручивании стека все обработчики на каждом уровне просматриваются последовательно, от внутреннего блока к внешнему, пока не будет найден походящий обработчик.

Механизм обработки исключений

- Обработчик считается найденным, если тип объекта указанного после throw
  - такой же как и указанный в параметре catch
  - является производным от указанного в параметре catch
- Обработчики производных классов следует размещать до обработчиков базовых, поскольку в противном случае им никогда не будет передано управление.

Исключения при переполнении

- Процессом их генерации можно управлять помощью ключевых слов checked и unchecked.
- Данную проверку можно реализовать как для отдельного выражения, так и для блока операторов.
  - a=checked (b+c);
  - unchecked {a=b+c;}
- Данная проверка не распространяется на функции, вызванные в блоке.

Пример 3. Переполнение

```
using System;
 class ExceptionTestClass
 {
    static byte s(byte a, byte b)
    {
        return checked((byte)(a + b));
    }
```

Пример 3. Переполнение

```
static void Main()
   {
      try
      {
         byte n=255, m=1;
         Console.WriteLine(s(n, m));
      }
      catch (OverflowException e) { Console.WriteLine(e.Message);
}

      Console.ReadKey();
   }
 }
```

Результат

- n=255 и m=1 сгенерируется исключение и обработается
- n=10,m=5, то исключение не будет сгенерирована (так как их сумма меньше 255) на экран будет выведено 15.
- Если в функции s удалить слово checked, то исключение не будет сгенерировано и на экран будет выведено 0.

Исключение переполнения

- Проверку переполнения обычно выключают только в тех случаях, когда усечение результата необходимо в соответствии с алгоритмом.
- Можно задать проверку переполнения во всей программе с помощью ключа компилятора /checked.
- Так как это замедляет работу программы, то данный прием обычно используют при отладке.

Выводы

- все потенциально опасные фрагменты программы следует заключать в блок try
- следует обрабатывать хотя бы одно исключение типа Exception
- генерировать исключения рекомендуется в тех случаях, когда в месте возникновения ошибки недостаточно данных для ее обработки.