Обработка исключений

Непредусмотренные ошибки вызывают *исключения*.

Исключение, или *исключительная ситуация* – это аномалии, которые могут возникнуть во время выполнения приложения.

Основные типы:

- Ошибки при работе с ресурсами компьютера. Например, ошибки чтения/записи данных или выделения памяти для работы программы.
- Ошибки логики приложения – возникают, если разработчик допустил оплошность при разработке логики программы. Пример: деление на ноль, выход за пределы коллекции/массива.
- Ошибки пользователя – ошибки ввода данных или некорректные действия, которые нельзя предусмотреть.

Некоторые стандартные типы исключений

Базовым для всех типов исключений является тип Exception.

Имя	Описание
ArithmeticException	Ошибка в арифметических операциях и преобразованиях
DivideByZeroException	Попытка деления на ноль
FormatException	Попытка передать в метод аргумент неверного формата
IndexOutOfRangeException	Индекс массива выходит за границы диапазона
InvalidCastException	Ошибка преобразования типа
OutOfMemoryException	Недостаточно памяти для создания нового объекта
OverFlowException	Переполнение при выполнении арифметических операций
StackOverFlowException	Переполнение стека

Конструкция try-catch-finally

```
try
{
    // контролируемый блок (является обязательным)
}
catch
{
      // обработчик исключения (их может быть несколько)
}
finally
{
    // блок завершения (можно опустить)
}
```

- В *контролируемый блок* включаются операторы, которые потенциально могут вызвать ошибку.
- В *обработчике исключения* описывается то, как обрабатываются ошибки различных типов.
- *Блок завершения* выполняется независимо от того, возникла ли ошибка в блоке *try*.

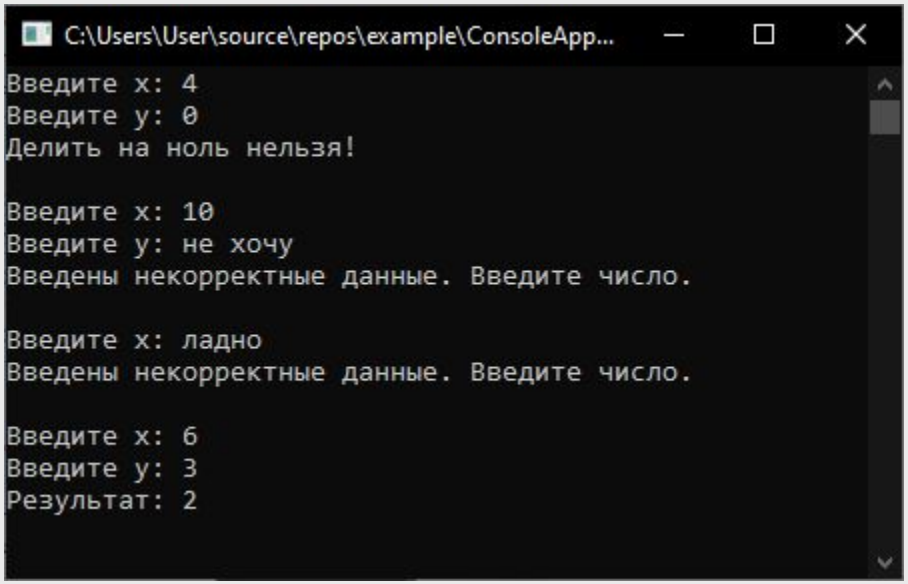
Конструкция try-catch-finally

- Вначале выполняются инструкции в блоке ***try***.
- Если в блоке *try* **возникло** исключение, выполнение текущего блока прекращается и выполняется поиск соответствующего обработчика исключения **в блоке *catch***, управление передается данному блоку.
- **В любом случае** (была ошибка или нет) **выполняется блок *finally***, если он присутствует.
- Если обработчик исключения не найден, вызывается стандартный обработчик исключения. Исполняющая система перехватит исключение, выдаст сообщение об ошибке и аварийно завершит работу программы.

Конструкция try-catch-finally

Пример 1.

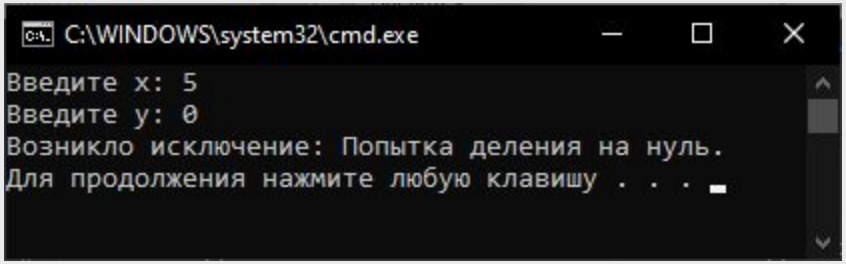
```
using System;
namespace ConsoleApp1
{
   class Program
   {
      static void Main()
      {
         try
         {
            Console.Write("Введите x: ");
            int x = int.Parse(Console.ReadLine());
            Console.Write("Введите y: ");
            int y = int.Parse(Console.ReadLine());
            double result = x / y;
            Console.WriteLine("Результат: " + result);
            Console.ReadKey();
         }
         // Обработка исключения, возникающего при делении на ноль
         catch (DivideByZeroException)
         {
            Console.WriteLine("Делить на ноль нельзя!\n");
            Main();
         }
         // Обработка исключения при некорректном вводе числа
         catch (FormatException)
         {
            Console.WriteLine("Введены некорректные данные. Введите число.\n");
            Main();
         }
      }
   }
}
```



Конструкция try-catch-finally

Пример 2.

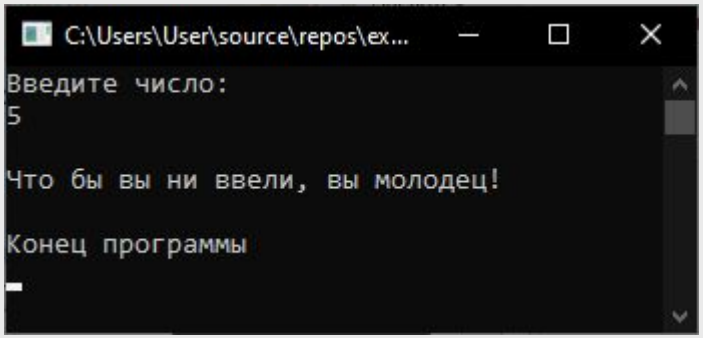
```
try
{
    Console.Write("Введите x: ");
    int x = int.Parse(Console.ReadLine());
    Console.Write("Введите y: ");
    int y = int.Parse(Console.ReadLine());
    double result = x / y;
    Console.WriteLine("Результат: " + result);
}
// Обработка исключения, возникающего при делении на ноль
catch (DivideByZeroException ex)
{
    Console.WriteLine($"Возникло исключение: {ex.Message}");
}
```

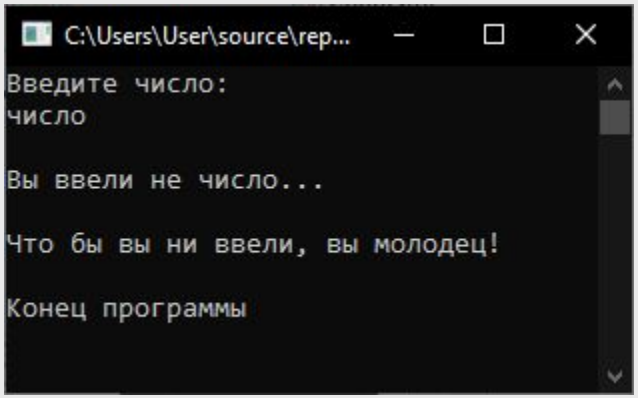


Конструкция try-catch-finally

Пример 3.

```
try
{
	Console.WriteLine("Введите число: ");
	int number = int.Parse(Console.ReadLine());
}
catch
{
	Console.WriteLine("\nВы ввели не число...");
}
finally
{
	Console.WriteLine("\nЧто бы вы ни ввели, вы молодец!");
}
Console.WriteLine("\nКонец программы");
Console.ReadLine();
```





Конструкция try-catch-finally

Блок catch может иметь следующие формы:

- Обрабатывает любое исключение из блока try:

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

- Обрабатывает только те исключения, которые соответствуют типу, указанному в скобках:

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

- Обрабатывает только те исключения, которые соответствуют типу, указанному в скобках, вся информация об исключении помещается в переменную данного типа:

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

Конструкция try-catch-finally

Фильтры исключений позволяют обрабатывать исключения в зависимости от определенных условий. Для их применения после выражения catch идет выражение when, после которого в скобках указывается условие:

```
catch when (условие)
{
    // выполняемые инструкции
}
```

Обработка в блоке catch происходит только в том случае, если условие в выражении when истинно.

Условные конструкции при обработке исключений

В ряде случаев более оптимально будет применить условные конструкции в тех местах, где можно применить блок try-catch.

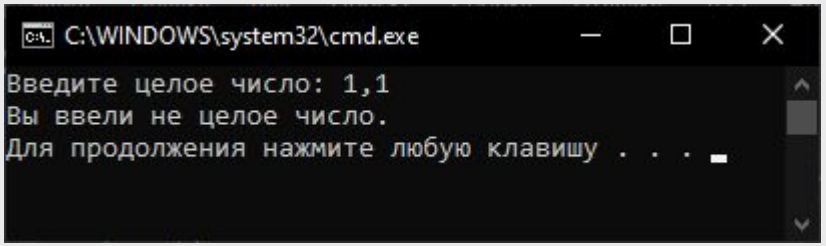
С точки зрения производительности использование блоков try-catch более накладно, чем применение условных конструкций. По возможности лучше использовать условные конструкции на проверку исключительных ситуаций вместо try-catch.

Условные конструкции при обработке исключений

Пример.

```
// При вводе нецелого числа возникнет исключение  и будет совершен аварийный выход из программы
Console.Write("Введите целое число: ");
int x = int.Parse(Console.ReadLine());
```

```
Console.Write("Введите целое число: ");
int x1;
if (int.TryParse(Console.ReadLine(), out x1))
{
   Console.WriteLine($"Квадрат числа {x1} = " + Math.Pow(x1, 2));
}
else
{
   Console.WriteLine("Вы ввели не целое число.");
}
```



Оператор throw

позволяет самостоятельно генерировать исключительные ситуации:

throw [объект_класса_исключений];

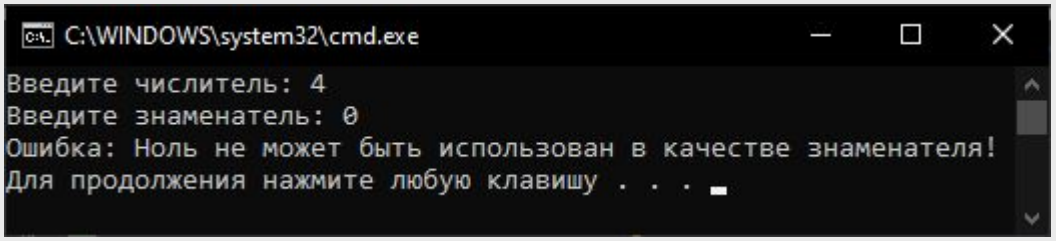
Например:

throw new DivideByZeroException();

В качестве параметра должен быть объект, порожденный стандартным классом System.Exception. Далее он используется для передачи информации об исключении его обработчику.

Оператор throw

Пример.




```
try
{
    Console.Write("Введите числитель: ");
    int a = int.Parse(Console.ReadLine());
    Console.Write("Введите знаменатель: ");
    int b = int.Parse(Console.ReadLine());
    if (b == 0)
    {
        throw new Exception("Ноль не может быть использован в качестве знаменателя!");
    }
    else
    {
        Console.WriteLine("Частное : " + (double)(a / b));
    }
}
catch (Exception e)
{
    Console.WriteLine($"Ошибка: {e.Message}");
}
```

Массивы

Массив – набор элементов одного типа, объединенных общим именем.

Одномерный массив – это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер.

Нумерация элементов массива в C# начинается с нуля: если массив состоит из 5 элементов, то они будут иметь следующие номера: 0, 1, 2, 3, 4.

Одномерные массивы

Одномерный массив в C# реализуется как объект, поэтому его создание представляет собой двухступенчатый процесс: сначала объявляется ссылочная переменная типа массив, затем выделяется память под требуемое количество элементов базового типа, и ссылочной переменной присваивается адрес нулевого элемента в массиве.

Базовый тип определяет тип данных каждого элемента массива.

Количество элементов, которые будут храниться в массиве, определяется размером массива. Размерность задается при выделении памяти и не может быть изменена впоследствии.

Способы объявления:

1) базовый_тип[] имя_массива;

Например: int[] numbers;

2) базовый_тип[] имя_массива = new базовый_тип[размер];

Например: int[] numbers = new int[10];

Одномерные массивы

В C# элементам массива присваиваются начальные значения по умолчанию в зависимости от базового типа. Для арифметических типов – нули, для ссылочных типов – null, для символов – символ с кодом ноль.

Третий вариант – инициализация массива сразу при объявлении.

Например:

```
int[] nums1 = new int[4] { 1, 2, 3, 4 };
int[] nums2 = new int[] { 1, 2, 3, 4 };
int[] nums3 = new[] { 1, 2, 3, 4 };
int[] nums4 = { 1, 2, 3, 4 };

string[] colors = { "Red", "Orange", "Yellow" };
```

Одномерные массивы

Обращение к элементу массива происходит с помощью *индекса* – номера элемента в массиве (нумерация начинается с нуля!).

Например: arr[0], a[9], b[i].

Получение элемента массива:

```
int[] numbers = { 1, 2, 3, 5 };
// вывод значения элемента массива на консоль
Console.WriteLine(numbers[3]);  // 5
// получение элемента массива в переменную
var n = numbers[1];     // 2
Console.WriteLine(n);  // 2
```

Одномерные массивы

Изменение элемента массива:

```
int[] numbers = { 1, 2, 3, 5 };

numbers[1] = 0;

Console.WriteLine(numbers[1]);  // 0
```

Каждый массив имеет свойство *Length*, которое хранит длину (размер) массива. Для получения длины массива необходимо обратиться к свойству Length, указав его через точку: numbers.Lentgh.

Например, получим последний элемент массива:

```
Console.WriteLine(numbers[numbers.Length - 1]); // 5
```

При работе с массивом автоматически выполняется контроль выхода за его границы: если значение индекса выходит за границы массива, генерируется исключение IndexOutOfRangeException.

Одномерные массивы

Так как массив представляет собой набор элементов, обработка массива обычно производится в цикле.

Вывод массива на экран

- При помощи цикла foreach:

```
int[] numbers = { 1, 2, 3, 4, 5 };          // Последовательно и
foreach (int i in numbers)                  // только для чтения
{
    Console.WriteLine(i);
}
```

- При помощи цикла for:

```
int[] numbers = { 1, 2, 3, 4, 5 };          // Можно менять приращение
for (int i = 0; i < numbers.Length; i++)// счетчика и изменять
{                                   // элементы
    Console.WriteLine(numbers[i]);
}
```

Одномерные массивы

Вывод массива на экран

- При помощи цикла while:

```
int[] numbers = { 1, 2, 3, 4, 5 };
int i = 0;
while(i < numbers.Length)
{
    Console.WriteLine(numbers[i]);
    i++;
}
```

Базовый класс Array

Все массивы в C# имеют общий базовый класс Array, определенный в пространстве имен System.

Некоторые элементы класса Array:

- Clear (Статический метод) – Присваивает элементам массива значения по умолчанию (для арифметических типов нули и т. д.);
- Copy (Статический метод) – Копирует элементы одного массива в другой массив;
- IndexOf (Статический метод) – Осуществляет поиск первого вхождения элемента в одномерный массив, если найден – возвращает индекс, иначе -1;
- Length (Свойство) - Количество элементов массива;
- Reverse (Статический метод) – Изменяет порядок следования элементов в массиве на обратный;
- Sort (Статический метод) – Упорядочивание элементов одномерного массива.

Двумерные массивы

Многомерные массивы имеют более одного измерения. Чаще всего используются *двумерные массивы*, которые представляют собой таблицы. Каждый элемент массива имеет два индекса, первый определяет номер строки, второй – номер столбца, на пересечении которых находится элемент. Нумерация строк и столбцов начинается с нуля.

Объявить двумерный массив можно одним из предложенных способов:

1) базовый_тип[,] имя_массива;

Например: int[,] a;

2) базовый тип[,] имя_массива = new базовый_тип[размер1, размер2];

Например: float[,] a= new float[3, 4];

Элементы массива инициализируются по умолчанию нулями.

Двумерные массивы

Объявление с инициализацией:

3) базовый_тип[,] имя_массива = {{элементы 1-ой строки}, … ,

{элементы n-ой строки}};

Например: int[,] a= new int[,] {{0, 1, 2}, {3, 4, 5}};

Все способы:

```
int[,] nums1;
int[,] nums2 = new int[2, 3];
int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums5 = new [,]{ { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Двумерные массивы

Обращение к элементу массива происходит с помощью индексов: указывается имя массива, в квадратных скобках номер строки и через запятую номер столбца, на пересечении которых находится данный элемент. Например, a[0, 0], b[2, 3], c[i, j].

Так как массив представляет собой набор элементов, объединенных общим именем, то обработка массива обычно производится с помощью вложенных циклов.

При обращении к свойству Length для двумерного массива получаем общее количество элементов в массиве.

Чтобы получить количество строк, нужно обратиться к методу GetLength с параметром 0. Чтобы получить количество столбцов – к методу GetLength с параметром 1.