

# **Обработка исключений**

- **Исключение** - это объект, генерирующий информацию о «необычном программном происшествии»
- *Ошибка в программе* допускается программистом при ее разработке.
- *Ошибочная ситуация* вызвана действиями пользователя.
- *Исключительная ситуация* - непредсказуемая и неотвратимая проблема.

# Оператор try

- В C# исключения представляются классами.
- Все классы исключений порождены от встроенного класса исключений *Exception*, который определен в пространстве имен *System*.

```
try // контролируемый блок
{ /*Программные инструкции, которые нужно
проконтролировать на предмет исключений*/
...}
catch //один или несколько блоков обработки исключений
{ /*Если исключение возникает в этом блоке, оно дает знать о
себе выбросом определенного рода информации. Выброшенная
информация может быть перехвачена и обработана*/
...}
finally //блок завершения
{ /*Любой код, который должен быть обязательно выполнен
при выходе из блока try*/
...}
```

```
static void Main()
{
    int x = int.Parse(Console.ReadLine());
    int y = 1 / x;
    Console.WriteLine(y);
}
```

- **Исключительные ситуации :**

- 1) пользователь может ввести нечисловое значение
- 2) если ввести значение 0, то произойдет деление на 0.

```

class Program{
    static void Main()    {
        try
        { int x = int.Parse(Console.ReadLine());
          int y = 1 / x;
          Console.WriteLine("y={0}", y);
          Console.WriteLine("блок try выполнен успешно");
        }
        catch
        { Console.WriteLine("возникла какая-то ошибка");    }
          Console.WriteLine("конец программы");    }
    }
}

```

```

0
возникла какая-то ошибка
конец программы
Для продолжения нажмите любую клавишу . .

```

```

к
возникла какая-то ошибка
конец программы
Для продолжения нажмите любую клавишу .

```

Когда возникает исключение, выполнение программы останавливается и управление передается блоку *catch*. Этот блок *никогда* не возвращает управление в то место программы, где возникло исключение.

Команды из блока *try*, расположенные ниже строки, в которой возникло исключение, никогда не будут выполнены.

Блок *catch* обрабатывает исключение, и выполнение программы продолжается с оператора, следующего за этим блоком.

# Вывод информации об ошибке

```
catch (Exception error)
{
    Console.WriteLine("Возникла ошибка
{0}", error);
}
```

- «выброшенная» информация будет записана в идентификатор **error**. Её можно просмотреть с помощью метода **WriteLine**.

# Стандартные классы исключений

<i>Имя</i>	<i>Описание</i>
<b>ArithmeticException</b>	Ошибка в арифметических операциях или преобразованиях
<b>ArrayTypeMismatchException</b>	Попытка сохранения в массиве элемента несовместимого типа
<b>DivideByZeroException</b>	Попытка деления на ноль
<b>FormatException</b>	Попытка передать в метод аргумент неверного формата
<b>IndexOutOfRangeException</b>	Индекс массива выходит за границу диапазона
<b>InvalidCastException</b>	Ошибка преобразования типа
<b>OutOfMemoryException</b>	Недостаточно памяти для нового объекта
<b>OverflowException</b>	Перевыполнение при выполнении арифметических операций
<b>StackOverflowException</b>	Переполнение стека

# Операторы *checked* и *unchecked*

- В C# предусмотрено специальное средство, которое связано с генерированием исключений, вызванных переполнением результата в арифметических вычислениях.

```
static void Main()  
{  
    byte x = 200; byte y = 200;  
    byte result = (byte) (x + y);  
    Console.WriteLine(result);  
}
```

произведение значений *a* и *b*  
превышает диапазон представления  
значений типа *byte*



# Операторы *checked* и *unchecked*

- *checked* - для указания, что некоторое выражение должно быть проконтролировано на предмет переполнения
- *unchecked* – игнорирование переполнения

- Можно **задать (или отключить) проверку переполнения** сразу для всего проекта. Для этого необходимо выполнить следующие действия:
  1. Щелкнуть правой кнопкой мыши на имени проекта
  2. В выпадающем меню выбрать Properties
  3. В появившемся окне (см. рис.) выбрать слева страницу Build
  4. Щелкнуть на кнопке Advanced
  5. В появившемся окошке поставить или убрать галочку напротив Check for arithmetic overflow/underflow property.

Solution Explorer - pr47

- Solution 'pr47' (1 project)
  - pr47
    - Properties
    - References
    - Program.cs

pr47 Program.cs Start Page Object Browser

Application

**Build**

Build Events

Debug

Resources

Settings

Reference Paths

Signing

Security

Publish

Configuration: Active (Debug) Platform: Active (Any CPU)

Allow unsafe code

Optimize code

Errors and warnings

Warning level: 4

Suppress warnings:

Treat warnings as errors

None

Specific warnings:

All

Output

Output path: bin\Debug\ Browse...

XML documentation file:

Register for COM interop

Generate serialization assembly: Auto

Advanced...

# Оператор *checked* имеет две формы:

- 1) проверяет конкретное выражение и называется *операторной checked-формой*

*checked* ((тип-выражения) *expr*)

где *expr* — выражение, значение которого необходимо контролировать.

Если значение контролируемого выражения переполнилось, генерируется исключение типа *OverflowException*.

- 2) проверяет блок инструкций

*checked*

```
{  
// Инструкции, подлежащие проверке  
}
```

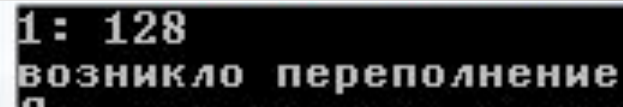
# проверка конкретного выражения

```
static void Main()
{
    byte x = 200; byte y = 200;
    try
    {
        byte result = unchecked((byte)(x + y));
        Console.WriteLine("1: {0}", result);
        result = checked((byte)(x + y));
        Console.WriteLine("2: ", result);
    }
    catch (OverflowException)
    {
        Console.WriteLine("возникло переполнение");
    }
}
```

```
1: 144
возникло переполнение
Для продолжения нажмите любую клавишу
```

# Контроль за блоком инструкций

```
class Program {
    static void Main() {
        byte n = 1; byte i;
        try
        {
            unchecked //блок без проверки
            {
                for (i = 1; i < 10; i++) n *= i;
                Console.WriteLine("1: {0}", n);
            }
            checked //блок с проверкой
            {
                n = 1;
                for (i = 1; i < 10; i++) n *= i;
                Console.WriteLine("2: ", n);
            }
        }
        catch (OverflowException)
        {
            Console.WriteLine("возникло переполнение");
        }
    }
}
```

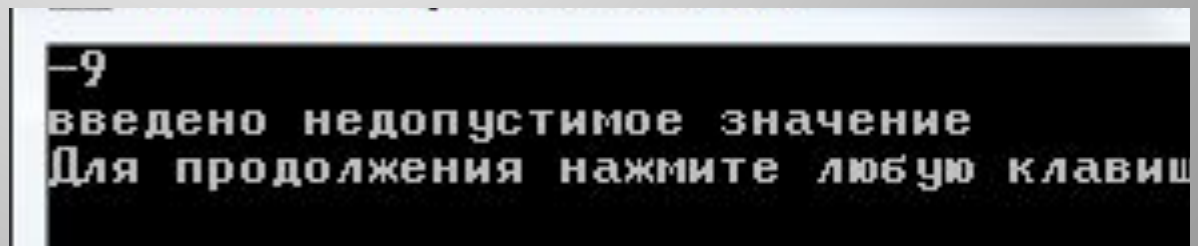


1: 128  
возникло переполнение

# Генерация собственных исключений

- Используя оператор *throw*, указав параметры, определяющие вид исключения (параметром должен быть объект, порожденный от стандартного класса *System.Exception*).
- Этот объект используется для передачи информации об исключении обработчику.

```
static void Main()    {
    try
    {
        int x = int.Parse(Console.ReadLine());
        if (x < 0) throw new Exception(); /* с помощью
команды new был создан объект исключения типа
Exception*/
        Console.WriteLine("ok");
    }
    catch
    {
        Console.WriteLine("введено недопустимое
значение");
    }
}
```

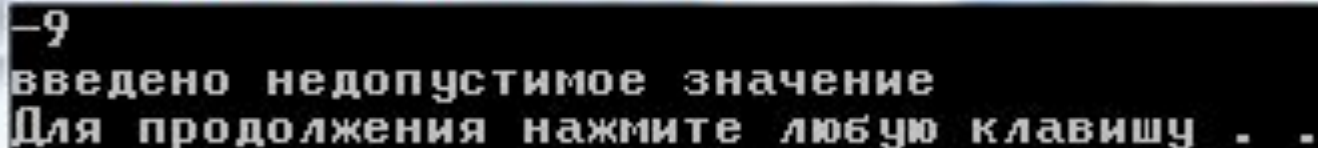


```
-9
введено недопустимое значение
Для продолжения нажмите любую клавишу
```



## При генерации исключения можно определить сообщение, которое будет «выбрасываться» обработчиком исключений

```
static void Main()
{
    try
    {
        int x = int.Parse(Console.ReadLine());
        if (x < 0) throw new Exception("введено недопустимое значение");
        Console.WriteLine("ok");
    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message);
    }
}
```



```
-9
введено недопустимое значение
Для продолжения нажмите любую клавишу . .
```

- 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**.
- Исключение, **перехваченное** одной catch-инструкцией, можно **сгенерировать повторно**, чтобы обеспечить возможность его перехвата другой (внешней) catch-инструкцией. Это позволяет нескольким обработчикам получить доступ к исключению.
- Тип исключения **должен совпадать** с типом, заданным в catch-инструкции. В противном случае это исключение не будет перехвачено.
- Можно **перехватывать все исключения**, используя catch-инструкцию **без параметров**.
- С try-блоком **можно связать не одну**, а несколько catch-инструкций. В этом случае все catch-инструкции должны перехватывать **исключения различного типа**.