

ЛЕКЦИЯ 11

- 11.10.2023
- Изменяемые строки StringBuilder
- Возникновение и обработка исключений

ЦЕЛИ ЛЕКЦИИ

- Познакомится с классом изменяемых строк `StringBuilder`
- Изучить методы работы со строками `StringBuilder C#`
- Обсудить варианты использования `String` и `StringBuilder` в программах
- Обобщить представления об ошибках в программах
- Познакомится с обработкой исключений подробнее



Это изображение, автор: Неизвестный автор, лицензия: CC BY-NC

ИЗМЕНЯЕМЫЕ СТРОКИ В C#

- Изменяемую строку символов в C# реализуют объекты типа `StringBuilder`
- Класс `StringBuilder` принадлежит пространству имён `System.Text`
- Не является перечислимой коллекцией, то есть не доступен для перебора `foreach`

ОБЪЕКТЫ STRINGBUILDER И ИХ СВЯЗЬ СО STRING

```
StringBuilder sb = "ABCD";
```

L-value в этом примере с типом **String** и **НЕЯВНО** не преобразуется в **StringBuilder**

```
StringBuilder sb = (StringBuilder)"ABCD";
```

L-value в этом примере с типом **String** и **ЯВНО** не преобразуется в **StringBuilder**

Строки **StringBuilder** создаются при помощи конструктора

ЕМКОСТЬ И ДЛИНА ОБЪЕКТА STRINGBUILDER

```
StringBuilder sb = new StringBuilder();
```

```
Console.WriteLine(sb.Capacity);
```

```
Console.WriteLine(sb.Length);
```

Ёмкость объекта StringBuilder –
максимальное значение содержащихся
СИМВОЛОВ

Длина содержащейся в StringBuilder **строки**
– фактическое значение количества
содержащихся в ней СИМВОЛОВ

При добавлении символов, пока значение
максимальной ёмкости не достигнуто,
память не перераспределяется

При достижении максимальной ёмкости,
автоматически выделяется память,
значение ёмкости удваивается

СОЗДАНИЕ ОБЪЕКТОВ STRINGBUILDER

```
// Пустой конструктор.  
StringBuilder sb = new StringBuilder();  
// Исходно задаем емкость 25.  
StringBuilder sb2 = new StringBuilder(25);  
// Корректный вариант инициализацией строкой.  
StringBuilder sb3 = new StringBuilder("ABCD");  
// Задаём исходную емкость и лимит емкости.  
// Именованные параметры использованы для ясности.  
StringBuilder sb4 = new StringBuilder(capacity: 20, maxCapacity: 35);  
// Инициализируем строкой и сразу устанавливаем емкость.  
StringBuilder sb5 = new StringBuilder("ABCD", 20);  
// Инициализируем подстрокой строки и устанавливаем емкость.  
StringBuilder sb6 = new StringBuilder("Let eat bee...",  
    startIndex: 4, length: 3, capacity: 30);
```

В поздних версиях .NET Core и .NET Framework это можно обойти

РАБОТА С ЕМКОСТЬЮ

```
StringBuilder sb4 = new StringBuilder(capacity: 20, maxCapacity: 35);  
sb4.Capacity = 40;
```

```
Unhandled exception. System.ArgumentOutOfRangeException: Capacity exceeds maximum capacity. (Parameter 'value')  
at System.Text.StringBuilder.set_Capacity(Int32 value)  
at Program.Main() in F:\Work\Work-HSE\Maxi-HSE\Prog\2023-10-04\LecEx\Program.cs:line 10
```

```
int i = 1024_0000;  
StringBuilder sb4 = new StringBuilder(capacity: 20, maxCapacity: 26);  
Console.WriteLine(sb4.Capacity);  
  
sb4.Append("abcdefghijklmnopqrstuvxyz").Append(i.ToString());  
Console.WriteLine(sb4.Capacity);
```

.NET Core .NET Framework 4.0

ДОПИСЫВАНИЕ В КОНЕЦ СТРОКИ

```
string[] song = { "When I find myself in times of trouble",
  "Mother Mary comes to me",
  "Speaking words of wisdom",
  "Let it be",
  "And in my hour of darkness",
  "She is standing right in front of me",
  "Speaking words of wisdom",
  "Let it be"};
```

```
StringBuilder sb = new StringBuilder();
for(int i = 0; i < song.Length; i++)
{
    sb.Append(song[i]);
    Console.WriteLine($"Cap:: {sb.Capacity} Length:: {sb.Length} Added: {song[i]}");
}
Console.WriteLine(sb.ToString());
```

```
Cap:: 38 Length:: 38 Added: When I find myself in times of trouble
Cap:: 76 Length:: 61 Added: Mother Mary comes to me
Cap:: 152 Length:: 85 Added: Speaking words of wisdom
Cap:: 152 Length:: 94 Added: Let it be
Cap:: 152 Length:: 120 Added: And in my hour of darkness
Cap:: 304 Length:: 156 Added: She is standing right in front of me
Cap:: 304 Length:: 180 Added: Speaking words of wisdom
Cap:: 304 Length:: 189 Added: Let it be
When I find myself in times of troubleMother Mary comes to
meSpeaking words of wisdomLet it beAnd in my hour of darknessShe is
standing right in front of meSpeaking words of wisdomLet it be
```


ДОПИСЫВАНИЕ В КОНЕЦ СТРОКИ С ФОРМАТИРОВАНИЕМ

```
When I find myself in times of
troubleMother Mary comes to meSpeaking
words of wisdomLet it beAnd in my hour
of darknessShe is standing right in
front of meSpeaking words of wisdomLet
it be Mean Lenght:: 23,625
```

```
StringBuilder sb = new StringBuilder();
int totalLength = 0;
for (int i = 0; i < song.Length; i++)
{
    sb.Append(song[i]);
    totalLength += song[i].Length;
    Console.WriteLine($" Cap:: {sb.Capacity} Length:: {sb.Length} Added: {song[i]}");
}
sb.AppendFormat(" Mean Lenght:: {0:f3}", (double)totalLength / song.Length);
Console.WriteLine(sb.ToString());
```

```
string[] song = { "When I find myself in times of
trouble",
"Mother Mary comes to me",
"Speaking words of wisdom",
"Let it be",
"And in my hour of darkness",
"She is standing right in front of me",
"Speaking words of wisdom",
"Let it be" };
```

УДАЛЕНИЕ

```
StringBuilder[] sb = new StringBuilder[song.Length];  
for (int i = 0; i < song.Length; i++) {  
    sb[i] = (new StringBuilder(song[i])).Remove(0,1);  
}  
for (int i = 0; i < sb.Length; i++)  
{  
    Console.WriteLine(sb[i].ToString());  
}
```

hen I find myself in times of trouble
other Mary comes to me
peaking words of wisdom
et it be
nd in my hour of darkness
he is standing right in front of me
peaking words of wisdom
et it be

```
string[] song = { "When I find myself in times of  
trouble",  
"Mother Mary comes to me",  
"Speaking words of wisdom",  
"Let it be",  
"And in my hour of darkness",  
"She is standing right in front of me",  
"Speaking words of wisdom",  
"Let it be" };
```

BCTABKA

```
StringBuilder[] sb = new StringBuilder[song.Length];  
for (int i = 0; i < song.Length; i++) {  
    sb[i] = (new StringBuilder(song[i])).Insert(0, "*", 3);  
}  
for (int i = 0; i < sb.Length; i++)  
{  
    Console.WriteLine(sb[i].ToString());  
}
```

```
string[] song = { "When I find myself in times of trouble",  
"Mother Mary comes to me",  
"Speaking words of wisdom",  
"Let it be",  
"And in my hour of darkness",  
"She is standing right in front of me",  
"Speaking words of wisdom",  
"Let it be" };
```

```
***When I find myself in times of trouble  
***Mother Mary comes to me  
***Speaking words of wisdom  
***Let it be  
***And in my hour of darkness  
***She is standing right in front of me  
***Speaking words of wisdom  
***Let it be
```

ПОИСК В СТРОКАХ: ПРОВЕРКА СОДЕРЖИМОГО СТРОКИ

```
for (int i = 0; i < song.Length; i++)  
{  
    if (song[i].Contains('I') || song[i].Contains("in"))  
    {  
        Console.WriteLine(song[i]);  
    }  
}
```

Поиск реализован методами класса String

When I find myself in times of trouble
Speaking words of wisdom
And in my hour of darkness
She is standing right in front of me
Speaking words of wisdom

ПОИСК В СТРОКАХ: ОПРЕДЕЛЕНИЕ ИНДЕКСА

String.LastIndexOf
String.LastIndexOfAny

String.IndexOf
String.IndexOfAny

```
for (int i = 0; i < song.Length; i++)  
{  
    if (song[i].Contains('I') || song[i].Contains("in"))  
    {  
        Console.WriteLine($"{song[i]} " +  
            $"{song[i].IndexOf('I')} " +  
            $"{song[i].IndexOfAny(new char[]{'i', 'n'})}");  
    }  
}
```

```
When I find myself in times of trouble 5 3  
Speaking words of wisdom -1 5  
And in my hour of darkness -1 1  
She is standing right in front of me -1 4  
Speaking words of wisdom -1 5
```

ПОИСК В СТРОКАХ: ПРОВЕРКА НАЧАЛА СТРОКИ

```
for (int i = 0; i < song.Length; i++)  
{  
    if (song[i].StartsWith('A'))  
    {  
        Console.WriteLine($"{song[i]}");  
    }  
}
```

And in my hour of darkness

```
for (int i = 0; i < song.Length; i++)  
{  
    if (song[i].StartsWith("sh", true, System.Globalization.CultureInfo.CurrentCulture))  
    {  
        Console.WriteLine($"{song[i]}");  
    }  
}
```

She is standing right in front of me

ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ STRING И STRINGBUILDER

String

Заранее известное малое
количество изменений в
строке

Фиксированное число
объединений

Строковые литералы

Разнообразные действия,
связанные с поиском

StringBuilder

Количество изменений
неизвестно

Данные
произвольные, в т.ч.
пользовательские

Количество изменений
велико

Данные
произвольные, в т.ч.
пользовательские

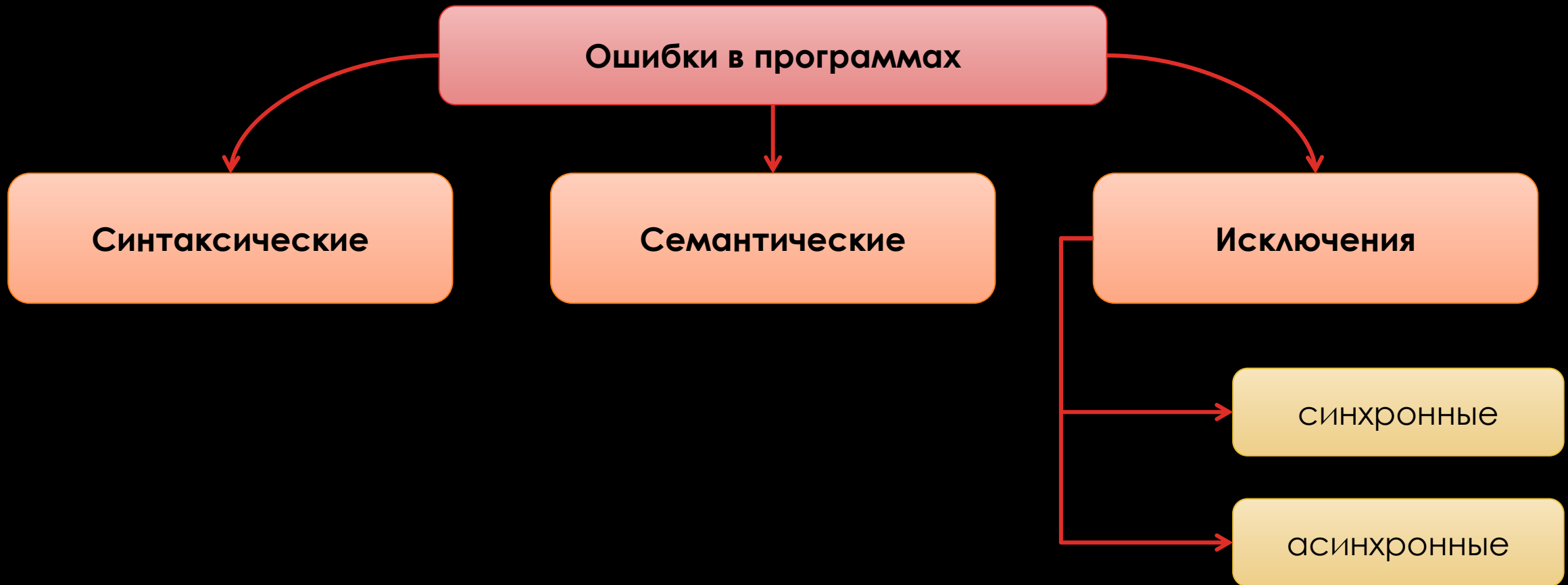
РАБОТА С ОШИБКАМИ ПРОГРАММ

Виды ошибок

Работа с исключениями



ОШИБКИ В ПРОГРАММАХ



ЧТО ТАКОЕ ИСКЛЮЧЕНИЯ?

- Исключения в .Net (C#) - классическая реализация модели исключений в ООП
- являются мощным механизмом для централизованной обработки исключений (исключительных событий)
- замена процедурно-ориентированному подходу
 - каждый метод возвращал код ошибки
- упрощают разработку кода и поддержку
- позволяют обрабатывать проблемные ситуации на множестве уровней
- исключение – это объект типа Exception (для управляемого кода)

EXCEPTIONS – ИСКЛЮЧЕНИЯ

```
int x = 10;  
int y = 0;
```

```
x /= y;
```

При некоторых вариантах
входных данных строка
приведёт к возникновению
исключения

```
Console.WriteLine("And we are here");
```

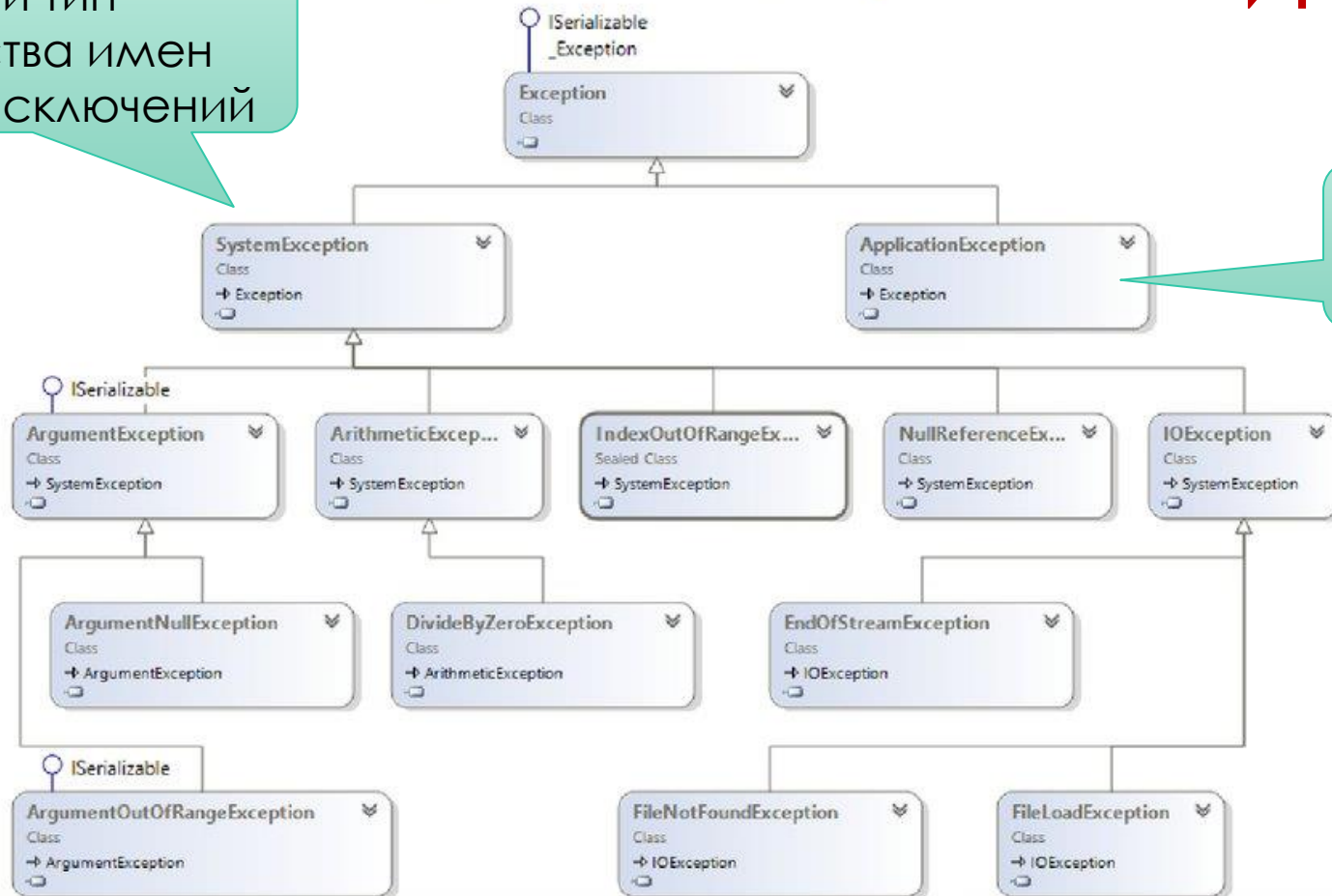
Это значит, что сюда можно
никогда не попасть

```
Unhandled exception. System.DivideByZeroException: Attempted to divide by zero.  
at Program.<Main>$(String[] args) in D:\Programming\ConsoleApp1\ConsoleApp1\Program.cs:line 4
```

```
D:\Programming\ConsoleApp1\ConsoleApp1\bin\Debug\net6.0\ConsoleApp1.exe (процесс 20540) завершил работу с кодом -1073741676.
```

ИЕРАРХИЯ КЛАССОВ ИСКЛЮЧЕНИЙ

Базовый тип
пространства имен
системных исключений



Исключения,
определяемые
приложением

ОПЕРАТОР TRY

```
try
{
    // Код, потенциально содержащий исключения.
}
```

Обязательный блок

```
catch (...)
{
    // Вариант обработки исключения.
}
catch(...)
{
    // Ещё вариант обработки исключения.
}
catch ...
```

Может отсутствовать,
если есть `finally` блок

```
finally
{
    //
}
```

Может отсутствовать,
если есть хотя бы один
`catch` блок

ПРИМЕР

```
int x = 10;
```

```
try
```

```
{
```

```
    int y = 0;
```

```
    x /= y;
```

```
}
```

```
catch
```

```
{
```

```
    Console.WriteLine($"x hasn't changed value: {x} ");
```

```
}
```

```
Console.WriteLine("And we are here");
```

Место, где может быть
исключение, размещаем в блок
операторов при try

Размещаем реакцию
на возникновение
исключения в блоке за
catch

x hasn't changed value: 10
And we are here

СВОЙСТВА ОБЪЕКТА ТИПА EXCEPTION

Свойство	Тип	Описание
Message	string	Строка с сообщением, объясняющим причину возникновения исключения
StackTrace	string	Информация о месте возникновения исключения (стек вызовов)
InnerException	Exception	Если исключение появилось из-за возникновения другого исключения, то это свойство содержит ссылку на предыдущее исключение
Source	string	Содержит информацию о сборке (assembly), в которой возникло исключение

```
catch { операторы }
catch (тип_исключения) { операторы }
catch (тип_исключения имя) { операторы }
catch (тип_исключения имя) when (<предикат>) { операторы } // C# 6.0
```

```

        catch ( IndexOutOfRangeException e )
        {
            Console.WriteLine( "Message: {0}", e.Message );
            Console.WriteLine( "Source: {0}", e.Source );
            Console.WriteLine( "Stack: {0}", e.StackTrace );
        }
    }
}

```

ПРИМЕР

```
int x = 10;
try
{
    int y = 0;
    x /= y;
}
catch (DivideByZeroException e)
{
    Console.WriteLine("Message: {0}", e.Message);
    Console.WriteLine("Source: {0}", e.Source);
    Console.WriteLine("Stack: {0}", e.StackTrace);
}
```

Указываем тип
перехватываемого
исключения явно

Имя объекта-исключения,
необходимо для обращения к
свойствам

```
Message: Attempted to divide by zero.
Source: ConsoleApp1
Stack:   at Program.<Main>$(String[] args) in D:\Programming\ConsoleApp1\ConsoleApp1\Program.cs:line 5
```

БЛОКИ CATCH

```
try
{
    // Код, потенциально содержащий исключения.
}
```

```
catch (...)
{
    // Вариант обработки исключения.
}
catch(...)
{
    // Ещё вариант обработки исключения.
}
catch ...
```

```
finally
{
    //
}
```

- Типы исключений идут от частного к общему, ориентируясь на иерархию наследования
- Секций с конкретными типами может быть более одной; секция с базовым (наиболее общим типом) только одна
- **catch { операторы }** и **catch (Exception) { операторы }** - СИНОНИМЫ

БЛОК FINALLY

```
try
{
    // Код, потенциально содержащий исключения.
}
```


```
catch (...)
{
    // Вариант обработки исключения.
}
catch(...)
{
    // Ещё вариант обработки исключения.
}
catch ...
```

```
finally
{
    //
}
```


- Всегда не более одного
- Может отсутствовать, если присутствует хотя бы один catch
- Код блока запускается, если исключение не произошло в try
- Если исключение произошло, то сначала отработает код в блоке catch с наиболее подходящим типом исключения, а затем произойдёт переход к finally

КОММЕНТАРИИ ПО СТИЛЮ КОДИРОВАНИЯ

- Логика расположения блоков catch опирается на наследование и полиморфизм



```
int x = 0;
try
{
    int.TryParse(Console.ReadLine(), out x);
    Console.WriteLine(10 / x);
}
catch (Exception ex)
{
    if (ex is ArithmeticException)
    {
        // logic 1
    }
    if (ex is ArgumentNullException)
    {
        // logic 2
    }
}
```



```
int x = 0;
try
{
    int.TryParse(Console.ReadLine(), out x);
    Console.WriteLine(10 / x);
}
catch( ArithmeticException arEx)
{
    // logic for Arithmetic Exception
}
catch( ArgumentNullException nullEx)
{
    // logic for ArgumentNullException
}
catch (Exception ex)
{
    // logic for the most general Exception
}
```

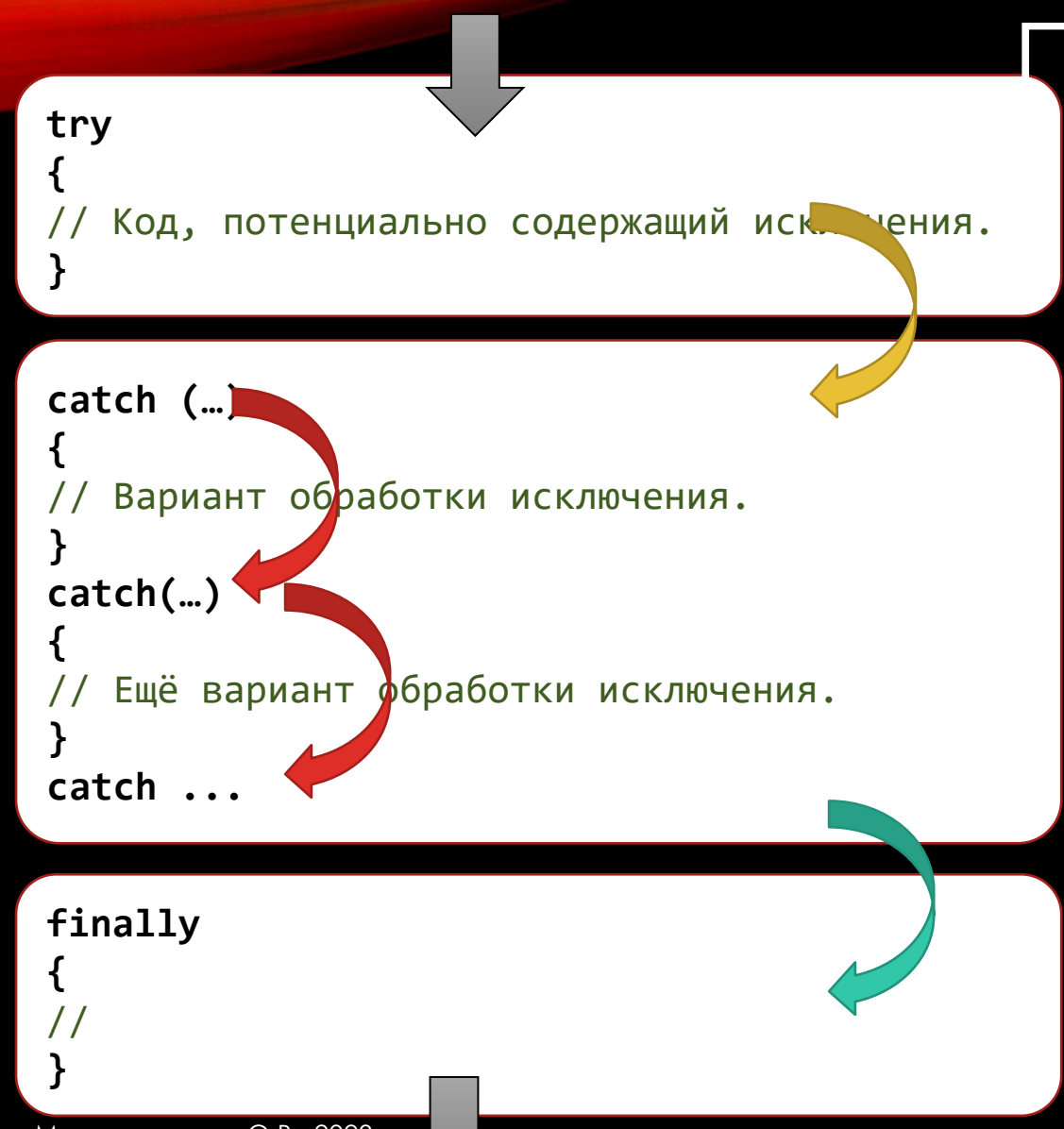
ПРИМЕР С RETURN В TRY-БЛОКЕ ВНУТРИ МЕТОДА

```
int inVal = 0;
try
{
    if (inVal < 10)
    {
        Console.WriteLine("Меньше 10 ");
        return;
    }
    else
        Console.WriteLine("Больше 10 ");
}
finally
{
    Console.WriteLine("Это выполняется всегда!");
}
```

Оператор безусловного
перехода

Выполняется
всегда

ПОИСК ОБРАБОТЧИКА ИСКЛЮЧЕНИЯ

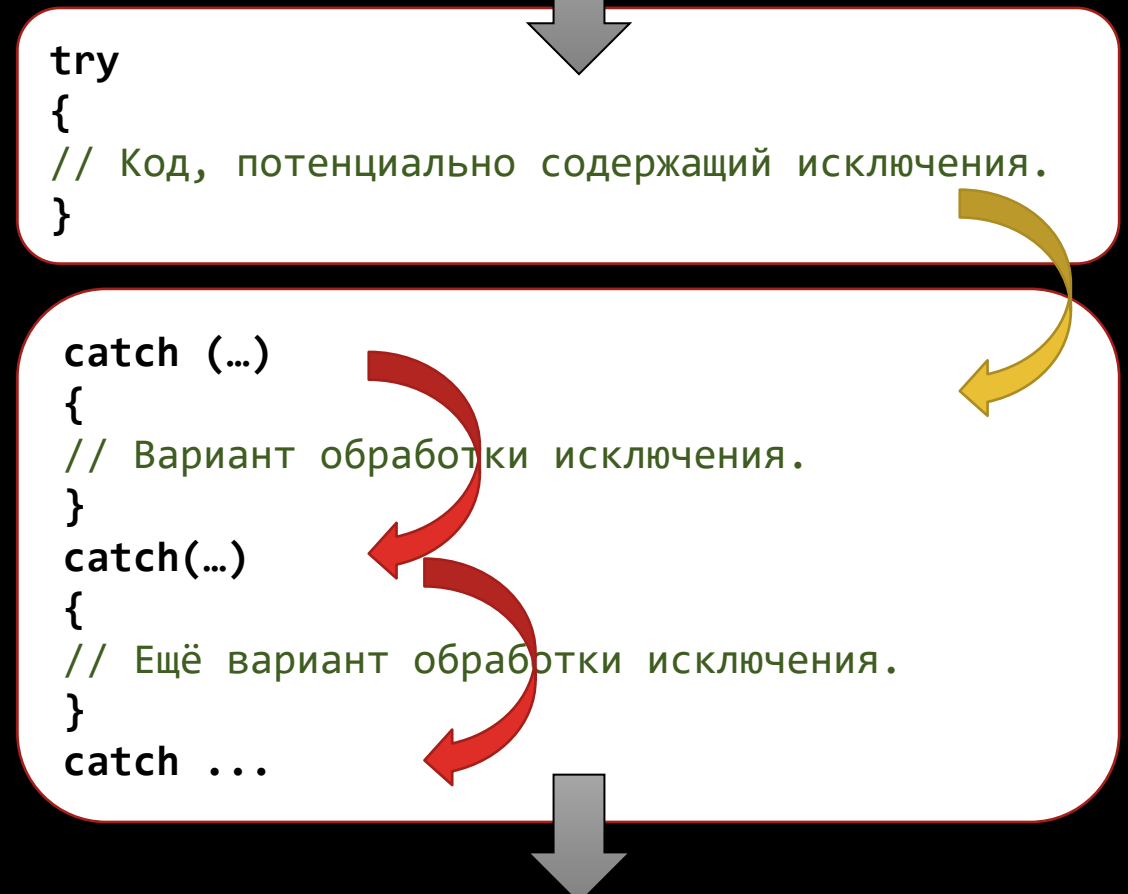


```
try
{
    // Код, потенциально содержащий исключения.
}
```

A flowchart illustrating the search for an exception handler. It starts with a 'try' block containing code that might throw an exception. A yellow arrow points from the 'try' block to a 'catch' block. The 'catch' block contains a 'catch (...)' statement followed by a comment indicating it's a variant of exception handling. A red arrow points from the 'catch (...)' statement to another 'catch (...)' statement, which also has a comment. A teal arrow points from the 'catch ...' statement to a 'finally' block. The 'finally' block contains a comment. A large grey arrow points down from the 'try' block, and another large grey arrow points down from the 'finally' block.

```
catch (...)  
{  
    // Вариант обработки исключения.  
}  
catch(...)  
{  
    // Ещё вариант обработки исключения.  
}  
catch ...
```

```
finally  
{  
    //  
}
```



```
try  
{  
    // Код, потенциально содержащий исключения.  
}
```

A flowchart illustrating the search for an exception handler. It starts with a 'try' block containing code that might throw an exception. A yellow arrow points from the 'try' block to a 'catch' block. The 'catch' block contains a 'catch (...)' statement followed by a comment indicating it's a variant of exception handling. A red arrow points from the 'catch (...)' statement to another 'catch (...)' statement, which also has a comment. A teal arrow points from the 'catch ...' statement to a 'finally' block. The 'finally' block contains a comment. A large grey arrow points down from the 'try' block, and another large grey arrow points down from the 'finally' block.

```
catch (...)  
{  
    // Вариант обработки исключения.  
}  
catch(...)  
{  
    // Ещё вариант обработки исключения.  
}  
catch ...
```

ПОИСК ОБРАБОТЧИКА ИСКЛЮЧЕНИЯ

```
public static void Foo()
{
    try
    {
        Bar();
    }
    catch ...
    finally ...
}
```

1

ВЫЗОВ

5

Ищем подходящий обработчик

6

Выполняем finally

Вопрос: можно ли “потерять” исключение?

Ответ:

13.10.6 The throw statement (ECMA-334 5th Edition / December 2017)

If the **finally** block throws another exception, processing of the current exception is terminated.

→ Да, можно, если в **finally** возникнет новое исключение.

```
public static void Bar()
{
    try
    {
        ...
    }
    catch ...
    finally ...
}
```

2

Произошло исключение

3

Ищем подходящий обработчик

4

Выполняем finally

Вызов

Foo();

Описание

```
static void Foo()
{
    try { Bar(); }
    catch (DivideByZeroException e)
    {
        Console.WriteLine("Обработали исключение в Foo");
    }
    finally
    {
        Console.WriteLine("Foo");
    }
}
```

Вызов

Описание

```
static void Bar()
{
    int x = 1, y = 0;
    try { x /= y; }
    catch (IndexOutOfRangeException)
    {
        Console.WriteLine("Обработали исключение в Bar");
    }
    finally
    {
        Console.WriteLine("Finally в Bar");
    }
}
```

Finally в Bar
Обработали исключение в Foo
Foo

АЛГОРИТМ ПОИСКА ОБРАБОТЧИКА



ВЫБРАСЫВАНИЕ ИСКЛЮЧЕНИЙ В МЕТОДЕ

```
class MyClass
{
    public static void PrintArg(string arg)
    {
        try
        {
            if (arg is null)
            {
                throw new ArgumentNullException("arg");
            }
            Console.WriteLine(arg);
        }
        catch (ArgumentNullException e)
        {
            Console.WriteLine($"Message: {e.Message}");
        }
    } // end of PrintArg()
} // end of class MyClass
```

```
class Program
{
    static void Main()
    {
        string? s = null;
        MyClass.PrintArg(s);
        MyClass.PrintArg("Hi there!");
    }
}
```

throw *ExceptionObject*;

Message: Value cannot be null. (Parameter 'arg')
Hi there!

ОПЕРАТОР THROW БЕЗ ОБЪЕКТА ИСКЛЮЧЕНИЯ

За обработку будет
отвечать внешний try

```
class Program {
    static void Main() {
        string? s = null;
        try {
            MyClass.PrintArg(s);
            MyClass.PrintArg("Hi there!");
        }
        catch (ArgumentNullException e) {
            Console.WriteLine($"Outer Catch: Handling '{e.Message}'");
        }
    }
}
```

```
class MyClass {
    public static void PrintArg(string arg) {
        try {
            if (arg is null) {
                throw new ArgumentNullException("arg");
            }
            Console.WriteLine(arg);
        }
        catch (ArgumentNullException e) {
            Console.WriteLine($"Inner Catch: {e.Message}");
            throw; // Пробрасывание текущего исключения.
        }
        // end of PrintArg()
    }
    // end of class MyClass
}
```

Ретранслированное
исключение будет
обработано тут

```

class Method {
    public static int[] ArrayRead() {
        int k = 0, dimAr = 2, x;
        int[] row = new int[dimAr];
        while (true) {
            do
                Console.Write("x = ");
            while (!int.TryParse(Console.ReadLine(), out x));
            if (x == 0) break;
            try { row[k] = x; k++; }
            catch (IndexOutOfRangeException) {
                dimAr *= 2;
                row = Method.VaryArray(row, dimAr);
                row[k++] = x;
            }
        } //end while
        row = Method.VaryArray(row, k);
        return row;
    }
    static int[] VaryArray(int[] ar, int newSize) {
        int[] temp = new int[newSize];
        Array.Copy(ar, temp, newSize < ar.Length ? newSize : ar.Length);
        return temp;
    }
}

```

```

static int[] VaryArray(int[] ar, int newSize)
{
    int[] temp = new int[newSize];
    Array.Copy(ar, temp, newSize < ar.Length
                ? newSize : ar.Length);
    return temp;
}

```

```

static void Main()
{
    int[] res = Method.ArrayRead();
    foreach (int memb in res)
        Console.Write(memb + " ");
}

```

Результат выполнения программы:

```

x = 1<ENTER>
x = 2<ENTER>
x = 3<ENTER>
x = 4<ENTER>
x = 5<ENTER>
x = 0<ENTER>
1 2 3 4 5

```

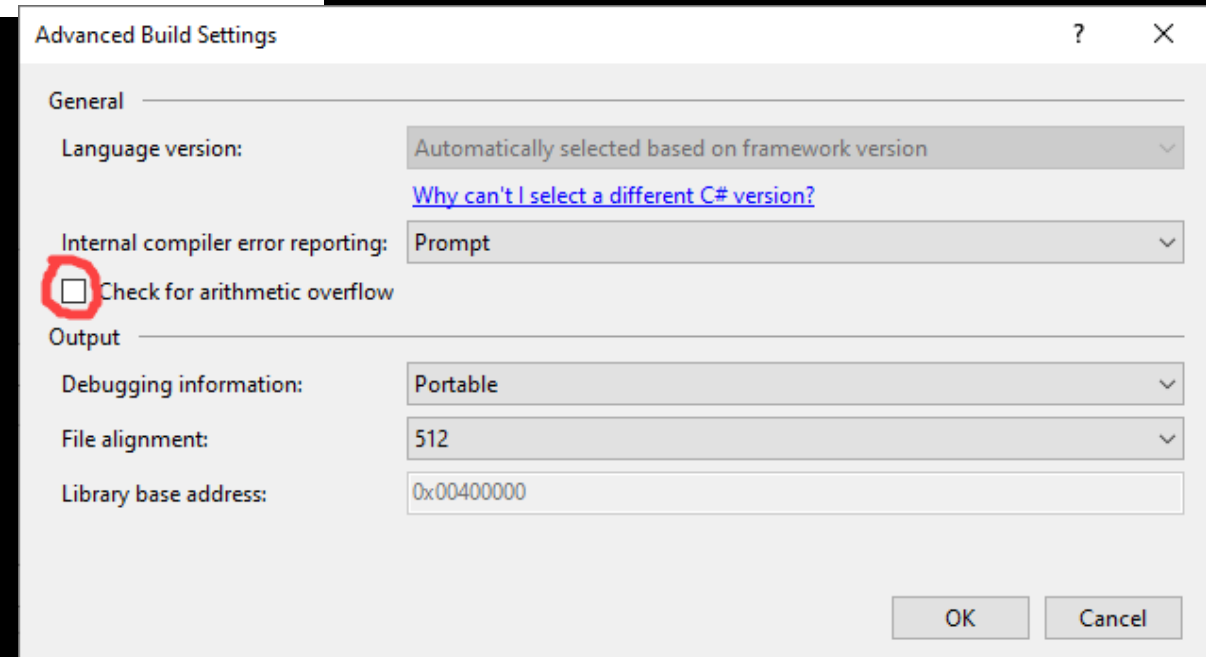
ИСКЛЮЧЕНИЯ В АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЯХ

```
int x = 111111, y = 111111, z = 0;  
double a = x / 0.0; // Результат: "бесконечность".  
double b = x / 0;    // Ошибка компиляции.  
double c = x / z;    // Исключение DivideByZeroException.  
double d = x * y;    // Результат: -539247567.
```

checked (*выражение*)
checked {*операторы*}

```
double e = checked(x * y);
```

unchecked



КОНТРОЛЬ ЗА ПЕРЕПОЛНЕНИЕМ

```
// тестовые данные
int[] x = { 0, 1, 2};
int[] y = { -int.MaxValue, int.MinValue, int.MaxValue};
// перебор вариантов пар тестовых данных без checked
foreach (int i in x)
{
    foreach(int j in y)
    {
        Console.WriteLine($"{i} + {j} = {i+j}");
    }
}
```

0 + -2147483647 = -2147483647
 0 + -2147483648 = -2147483648
 0 + 2147483647 = 2147483647
 1 + -2147483647 = -2147483646
 1 + -2147483648 = -2147483647
 1 + 2147483647 = -2147483648
 2 + -2147483647 = -2147483645
 2 + -2147483648 = -2147483646
 2 + 2147483647 = -2147483647

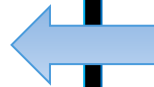
0 + -2147483647 = -2147483647
 0 + -2147483648 = -2147483648
 0 + 2147483647 = 2147483647
 1 + -2147483647 = -2147483646
 1 + -2147483648 = -2147483647
 Unhandled exception. **System.OverflowException**: Arithmetic operation resulted in an overflow.
 at ConsoleApp3.Program.Main(String[] args)

```
// тестовые данные
int[] x = { 0, 1, 2};
int [] y = { -int.MaxValue, int.MinValue, int.MaxValue};
// перебор вариантов пар тестовых данных с checked
foreach (int i in x)
{
    foreach (int j in y)
    {
        checked
        {
            Console.WriteLine($"{i} + {j} = {i + j}");
        }
    }
}
```


КОНТРОЛЬ ЗА ПЕРЕПОЛНЕНИЕМ

- Блок checked разместили в try
- Исключения обрабатываем
- Пользователь знает, что не все его данные корректно были обработаны

0 + -2147483647 = -2147483647
 0 + -2147483648 = -2147483648
 0 + 2147483647 = 2147483647
 1 + -2147483647 = -2147483646
 1 + -2147483648 = -2147483647
 Arithmetic operation resulted in an overflow.
 2 + -2147483647 = -2147483645
 2 + -2147483648 = -2147483646
 Arithmetic operation resulted in an overflow.



```

// тестовые данные
int[] x = { 0, 1, 2};
int [] y = { -int.MaxValue, int.MinValue, int.MaxValue};

// перебор вариантов пар тестовых данных с checked
foreach (int i in x)
{
    foreach (int j in y)
    {
        try
        {
            checked
            {
                Console.WriteLine($"{i} + {j} = {i + j}");
            }
        }
        catch (OverflowException e)
        {
            Console.WriteLine(e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
  
```

ЧТО КОНТРОЛИРУЕТСЯ В CHECKED / UNCHECKED?

- Встроенные арифметические операции:
 - Унарные ++ -- -
 - Бинарные - + * / (для символов, целых и перечислений)
- Явные преобразования между целыми арифметическими типами или приведением вещественных к целым
- Пользовательские версии приведения типов и операторов контроля за переполнением (начиная с C# 11)

ВЫБОР ТИПА ИСКЛЮЧЕНИЯ

1

При передаче неверного значения параметра в метод:

- `ArgumentException`,
`ArgumentNullException`
- `ArgumentOutOfRangeException`

2

При отсутствии поддержки операции:

- `NotSupportedException`

3

Когда отсутствует реализация функционального члена:

- `NotImplementedException`

4

Если нет подходящего стандартного типа исключения:

- наследуемся от `Exception`

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

- StringBuilder Класс (<http://learn.microsoft.com/ru-ru/dotnet/api/system.text.stringbuilder?view=net-7.0>)
- Операторы checked и unchecked (справочник по C#) (<https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/statements/checked-and-unchecked>)
- Лучшие практики (<https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions>)