В.В. Подбельский

Иллюстрации к курсу лекций по дисциплине «Программирование»

C#

Использованы материалы пособия Daniel Solis, Illustrated C#

Exceptions – исключения

Классификация ошибок в программах

- **Синтаксические (компиляции);**
- **Семантические (логические);**
- **Исключения:**
  - **Синхронные;**
  - **Асинхронные.**

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

```
static void Main()
{
    int x = 10, y = 0;
    x /= y; // попытка деления на 0 => исключение
}
```

Сообщение:

Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero. at Exceptions_1.Program.Main() in C:\Progs\Exceptions\Program.cs:line 12

Оператор try

```
try
{
  statements
}
```

This section is required.

```
catch( ... )
{
   statements
}
catch( ... )
{
   statements
}
catch ...
```

```
finally
{
   statements
}
```

One or both of these sections must be present. If both sections are present, the `finally` block must be placed last.

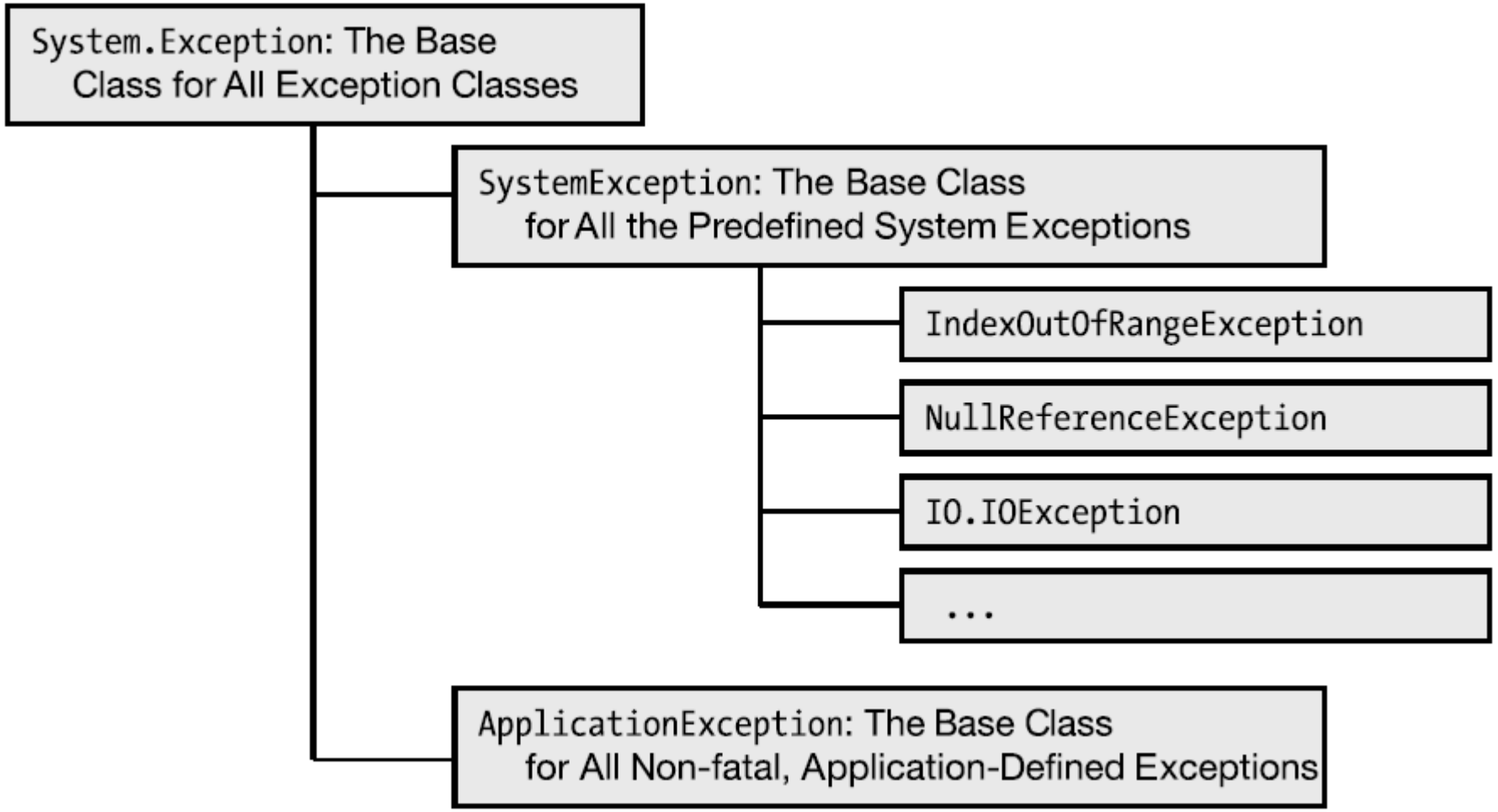
Пример кода с try-оператором

```
static void Main()
{
    int x = 10;
    try
    {
        int y = 0;
        x /= y; // возникновение исключения
    }
    catch  {
        // код для обработки исключения
        Console.WriteLine("'Обрабатываем' все – продолжаем работать");
    }
}
```

Результат выполнения программы:

Handling all exceptions - Keep on Running

Иерархия классов исключений



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.

Классы системных исключений

ArgumentException	Недопустимое значение аргумента при вызове метода
ArithmeticException	Базовый класс для исключений, которые возникают при выполнении арифметических операций. Например, DivideByZeroException или OverflowException.
ArrayTypeMismatchException	Посылается при попытке присвоить элементу массива значение, тип которого не совместим с типом элементов массива.
DivideByZeroException	Посылается при попытке деления на нуль целочисленного значения.
FormatException	Несоответствие параметров спецификации форматирования
IndexOutOfRangeException	Посылается при выходе индекса массива за пределы граничной пары (индекс отрицателен или больше верхней границы).
InvalidCastException	Посылается при неверном преобразовании от базового типа или базового интерфейса к производному типу.
NullReferenceException	Посылается при попытке применить ссылку со значением **null** для обращения к объекту.
OutOfMemoryException	Посылается при неудачной попытке выделить память с помощью операции **new**. (Недостаточно свободной памяти.)
OverflowException	Посылается при переполнениях в арифметических выражениях, выполняемых в контексте, определенном служебным словом **checked**.
RanException	Несоответствие размерностей параметра и аргумента при вызове метода.
StackOverflowException	Посылается при переполнении рабочего стека. Возникает, когда вызвано слишком много методов, например, при очень глубокой или бесконечной рекурсии.
TypeInitializationException	Посылается, когда исключение посылает статический конструктор и отсутствует обработчик исключений (catch-блок) для перехвата исключения

Четыре формы catch-инструкции

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

тип исключения ↓ — переменная со ссылкой на исключение ↓

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

Доступ к переменной исключения ↓

Пример перехвата исключения

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

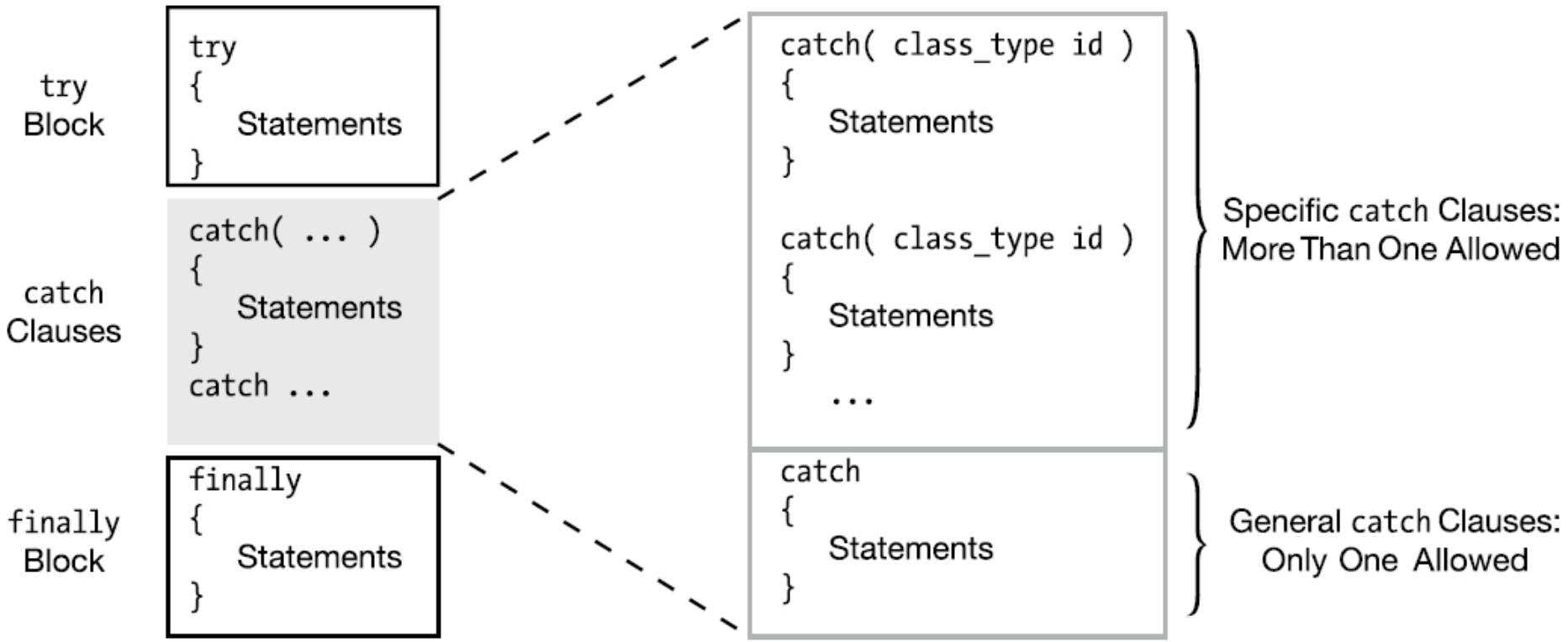
Результат выполнения программы:

Message: Attempted to divide by zero.
Source: Exceptions 1
Stack: at Exceptions_1.Program.Main() in C:\Progs\Exceptions 1\
Exceptions 1\Program.cs:line 14

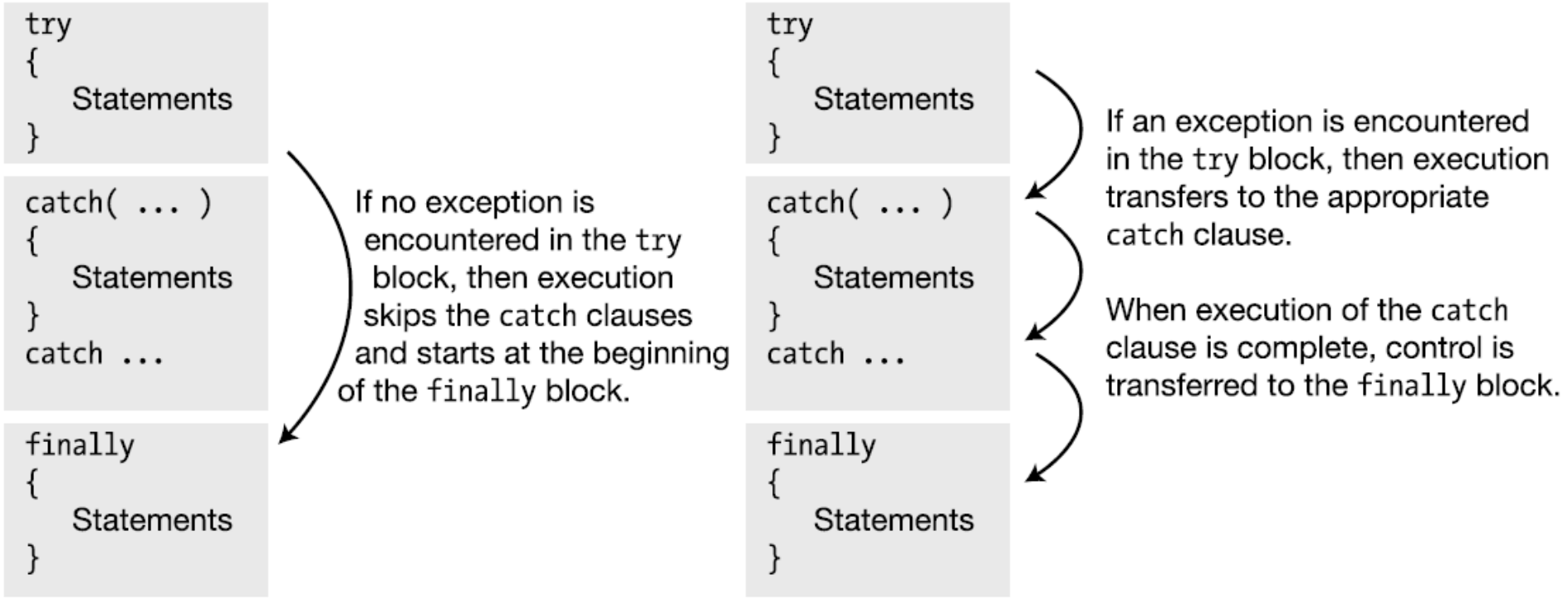
Пример фильтрации исключений (C# 6.0)

```
try {
        // Выполнить веб-запрос
}
catch (HttpRequestException e) when (e.Message.Contains("500"))
{
        // Выполнить действие при ошибке сервера
}
catch (HttpRequestException e) when (e.Message.Contains("404"))
{
        // Выполнить действие при отсутствии документа
}
catch (HttpRequestException e) when (e.Message.Contains("301"))
{
        // Выполнить действие при переадресации
}
```

Структура try-оператора



Блок finally



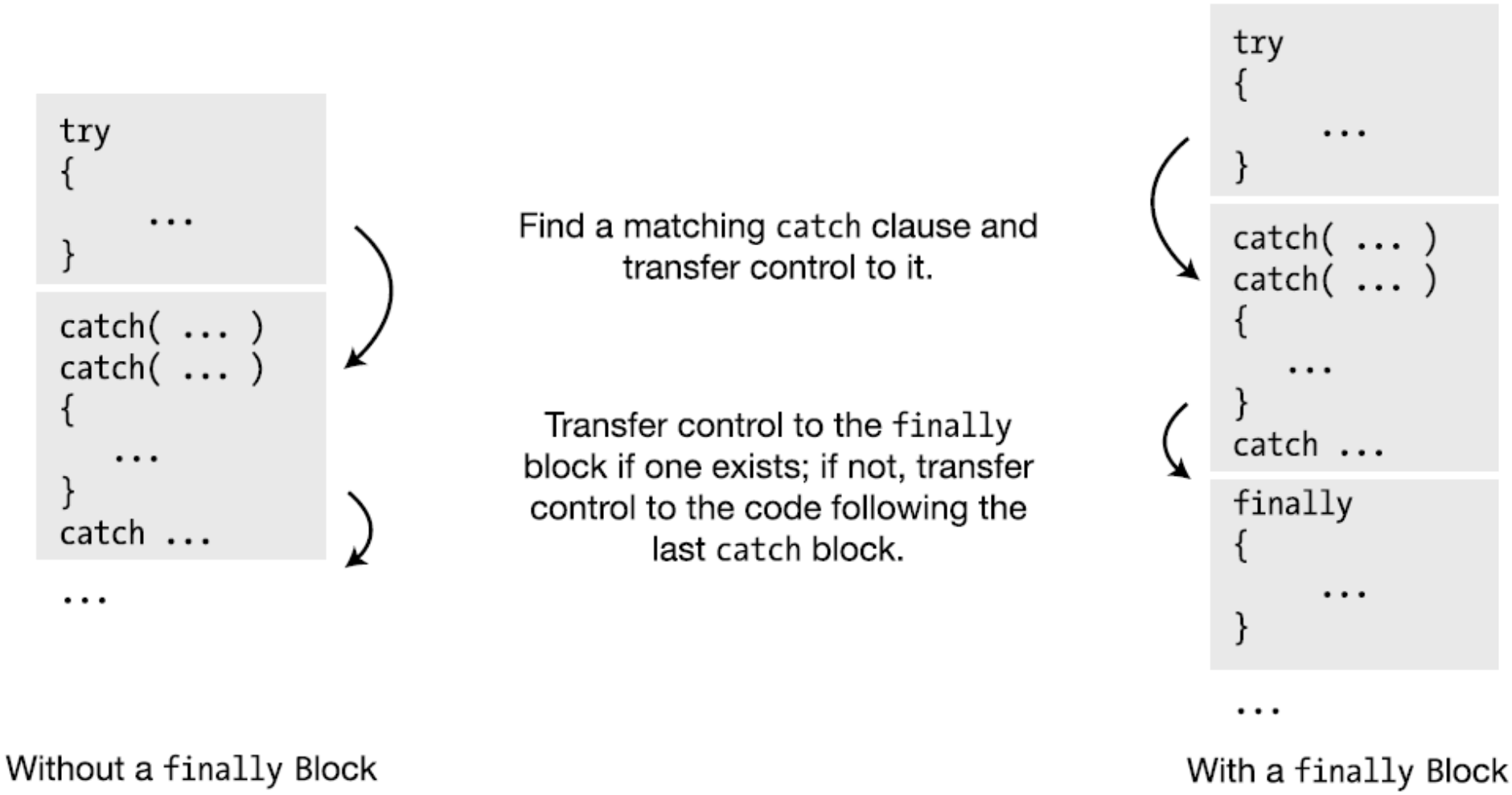
Пример с return в try-блоке внутри метода

```
try
{
    if (inVal < 10)
    {
        Console.Write("First Branch - ");
        return;
    }
    else
        Console.Write("Second Branch - ");
}
finally
{
    Console.WriteLine("In finally statement");
}
```

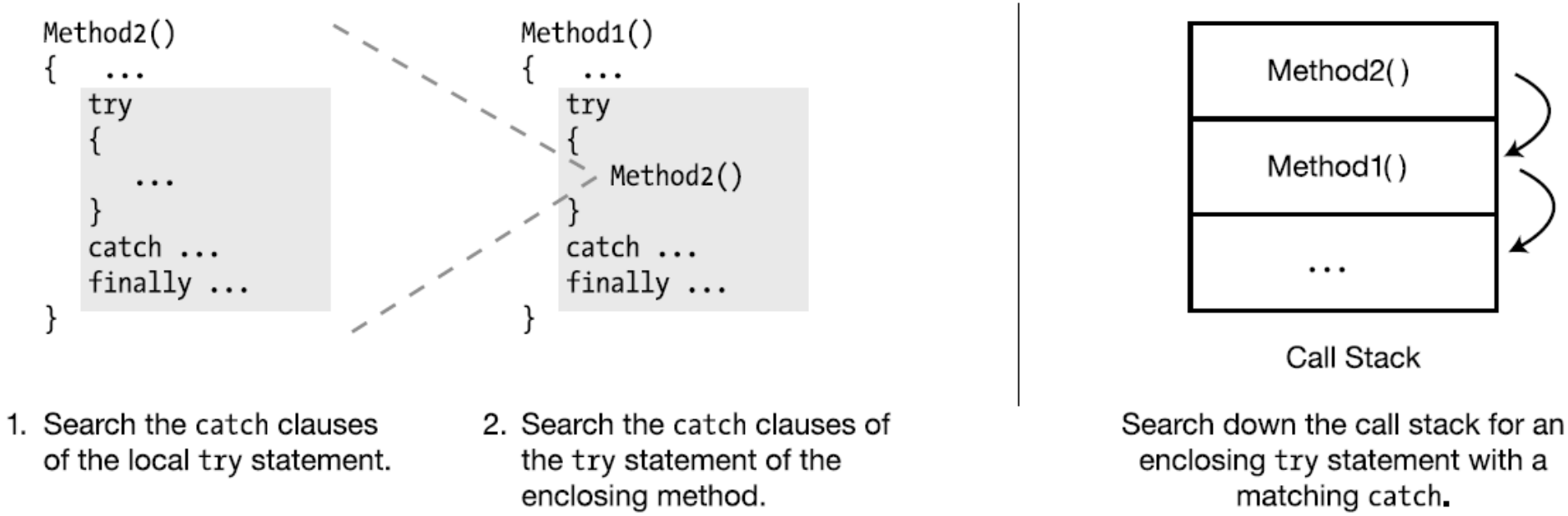
Если inVal равно 5, то результат такой:

First Branch - In finally statement

Поиск обработчика исключения



Дальнейший поиск обработчика исключений



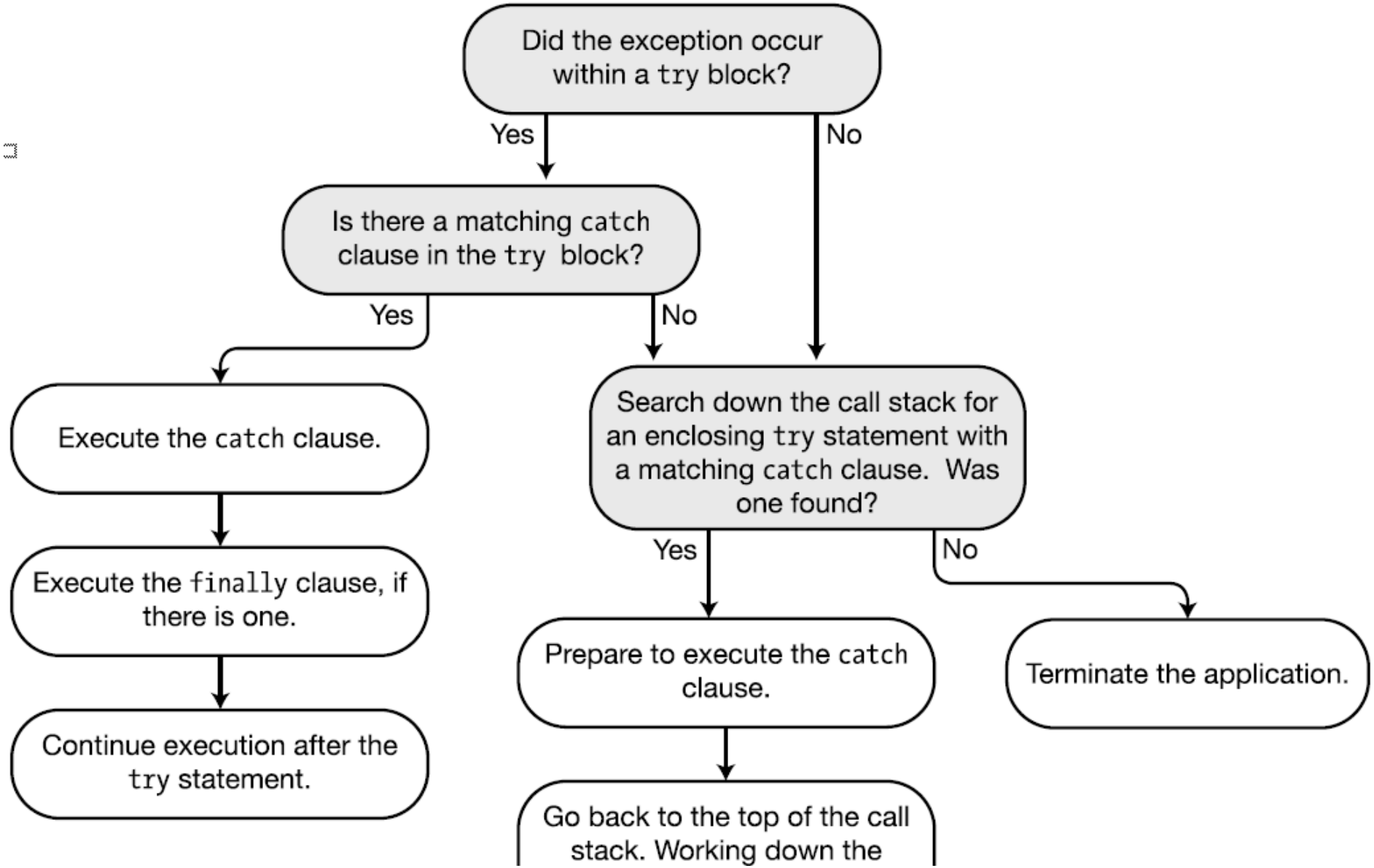
Вопрос: можно ли “потерять” исключение?

Ответ:

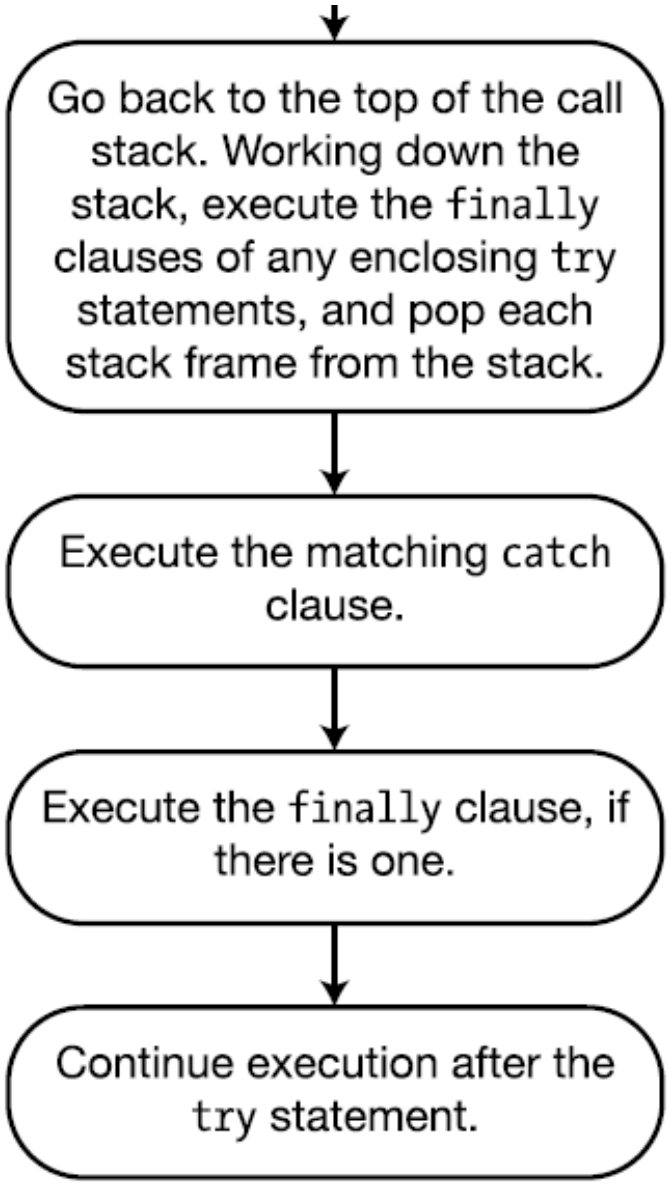
13.10.6 The throw statement (ECMA-334 5th Edition / December 2017)

If the **finally** block throws another exception, processing of the current exception is terminated. ➔ Да, можно, если в **finally** возникнет новое исключение.

Алгоритм поиска обработчика (1)



Алгоритм поиска обработчика (2)



Выбрасывание исключений (class)

throw *ExceptionObject*;

```
class MyClass
{
    public static void PrintArg(string arg)
    {
        try
        {
            if (arg == null)
            {
                ArgumentNullException myEx = new ArgumentNullException("arg");
                throw myEx;
            }
            Console.WriteLine(arg);
        }
        catch (ArgumentNullException e)
        {
            Console.WriteLine("Message: {0}", e.Message);
        }
    }   // PrintArg
}   // class MyClass
```

Выбрасывание исключений (Main)

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

Результат работы программы:
Message: Value cannot be null.
Parameter name: arg
Hi there!

Оператор throw без объекта исключения

```
class MyClass {
    public static void PrintArg(string arg) {
        try {
            try {
                if (arg == null) {
                    ArgumentNullException myEx = new ArgumentNullException("arg");
                    throw myEx;
                }
                Console.WriteLine(arg);
            } // try
            catch (ArgumentNullException e) {
                Console.WriteLine("Inner Catch: {0}", e.Message);
                throw; // пробрасывание текущего исключения
            }
        }   // try
        catch {
            Console.WriteLine("Outer Catch: Handling an Exception.");
        }
    }
}
```

Ретрансляция исключений

```
class Program
{
    static void Main()
    {
        string s = null;
        MyClass.PrintArg(s);
    }
}
```

Результат работы программы :

Inner Catch: Value cannot be null.
Parameter name: arg
Outer Catch: Handling an Exception.

Исключения при вводе

```
double x;
while (true) {
    try {
        Console.Write("x = ");
        x = double.Parse(Console.ReadLine());
        Console.WriteLine("res = " + x);
    }
    catch (FormatException) {
        Console.WriteLine("Ошибка в формате данных!");
        continue;
    }
    catch (ArithmeticException ex) {
        Console.WriteLine("ex.Message=" + ex.Message);
        Console.WriteLine("ex.Source=" + ex.Source);
        continue;
    }
    break;
} // while (true)
```

Исключения при вводе

Результат диалога с программой:

x = 5n6<ENTER>
Ошибка в формате данных!
x = 1e999999<ENTER>
ex.Message=Значение было недопустимо малым или
недопустимо большим для Double.
ex.Source=mscorlib
x = 543<ENTER>
res = 543

Управление программой с помощью исключений – VaryArray()

Вспомогательный метод:

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

newSize – требуемая длина нового массива

Управление программой

```
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;
}
```

Управление программой с помощью исключений – Main()

```
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

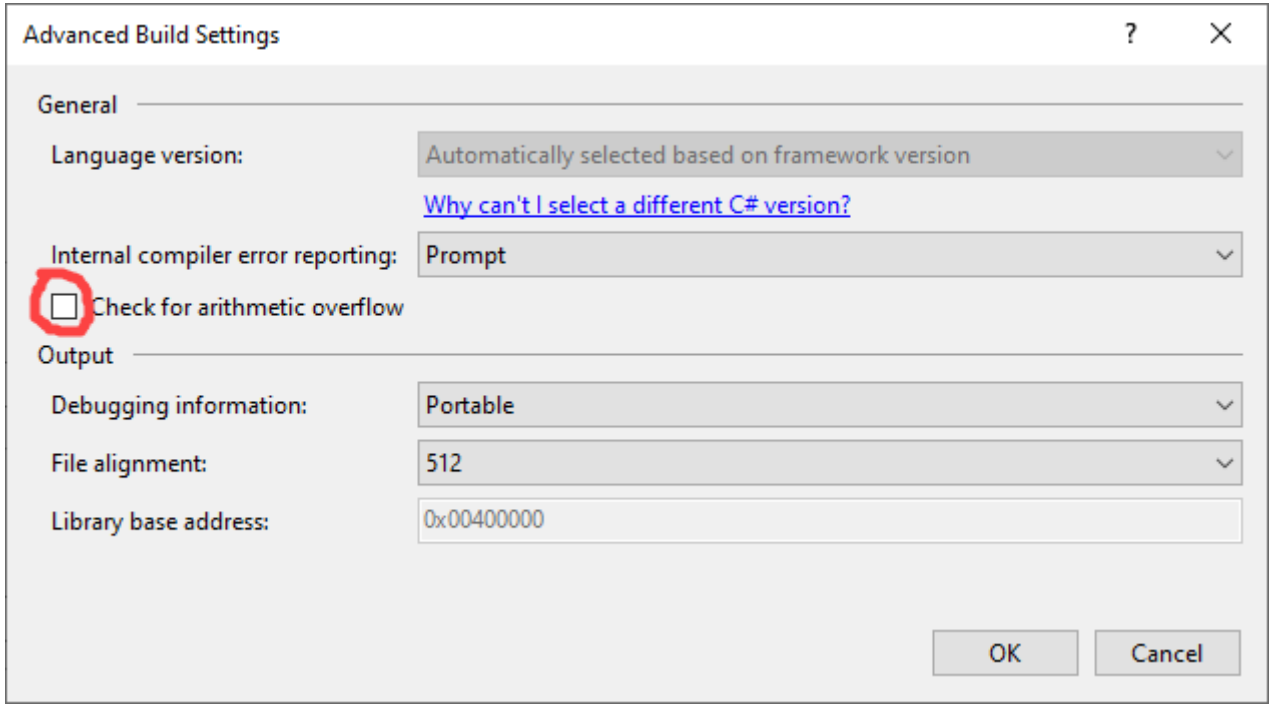
Исключения в арифметических выражениях

```
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



Исключения в библиотечном методе

Конструкторы класса System.Exception:

- Exception()
- Exception(String)
- Exception(String, Exception)
- Exception(SerializationInfo, StreamingContext)

Помним про сниппеты!

```
public static double Scalar(double[] x, double[] y)
{
    if (x.Length != y.Length)
        throw new ArgumentException("Неверные размеры векторов!");
    double result = 0;
    for (int i = 0; i < x.Length; i++)
        result += x[i] * y[i];
    return Math.Sqrt(result);
}
```

Исключения в методе

```
static void Main()
{
    try
    {
        Console.WriteLine("Scalar1 = " +
            Scalar(new double[] { 1, 2, 3, 4 }, new double[] { 1, 2, 3, 4 }));
        Console.WriteLine("Scalar2 = " +
            Scalar(new double[] { 1, 2, 3, 4 }, new double[] { 1, 2, 3 }));
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Результат выполнения программы:

scalar1=5,47722557505166
Неверные размеры векторов!

try-finally для очистки ресурсов

```
static string ReadFile1(string fileName) {
    System.IO.StreamReader reader = null;
    try {
        reader = System.IO.File.OpenText(fileName);
        if (reader.EndOfStream)
            return null;
        return reader.ReadToEnd();
    }
    finally {
        //if (reader != null) reader.Dispose();
        (reader as IDisposable)?.Dispose();
        Console.WriteLine("Выполнили очистку.");
    }
}
```

using для очистки ресурсов

```
static string ReadFile2(string fileName)
{
    using (System.IO.StreamReader reader =
           System.IO.File.OpenText(fileName))
    {
        if (reader.EndOfStream)
            return null;
        return reader.ReadToEnd();
    }
}
```

Выбор типа исключения

- При передаче неверного значения параметра в метод:
  ArgumentException, ArgumentNullException, ArgumentOutOfRangeException;
- При отсутствии поддержки операции:
  NotSupportedException;
- Когда отсутствует реализация функционального члена:
  NotImplementedException;
- Если нет подходящего стандартного типа исключения:
  наследуемся от **Exception**.

Базовые классы для пользовательских типов исключений

- System.Exception
- System.SystemException
- System.ApplicationException // не рекомендуется

Лучшие практики:
https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions

Базовые классы для пользовательских типов исключений

```
[Serializable]
public class TriangleException : Exception          // System.ApplicationException
{
    public TriangleException() : base() { }

    public TriangleException(string message) : base(message) { }

    public TriangleException(string message, Exception innerException)
        : base(message, innerException) { }

    public TriangleException(System.Runtime.Serialization.SerializationInfo info,
                             System.Runtime.Serialization.StreamingContext context)
        : base(info, context) { }

    public override string ToString() {
        return $"ОШИБКА ВРЕМЕНИ ИСПОЛНЕНИЯ({GetType()}): {Message} в {StackTrace}";
    }
}
```