iTechArt

Software
Engineering
Services

ОБРАБОТКА ИСКЛЮЧЕНИЙ

MARYIA DOLIA

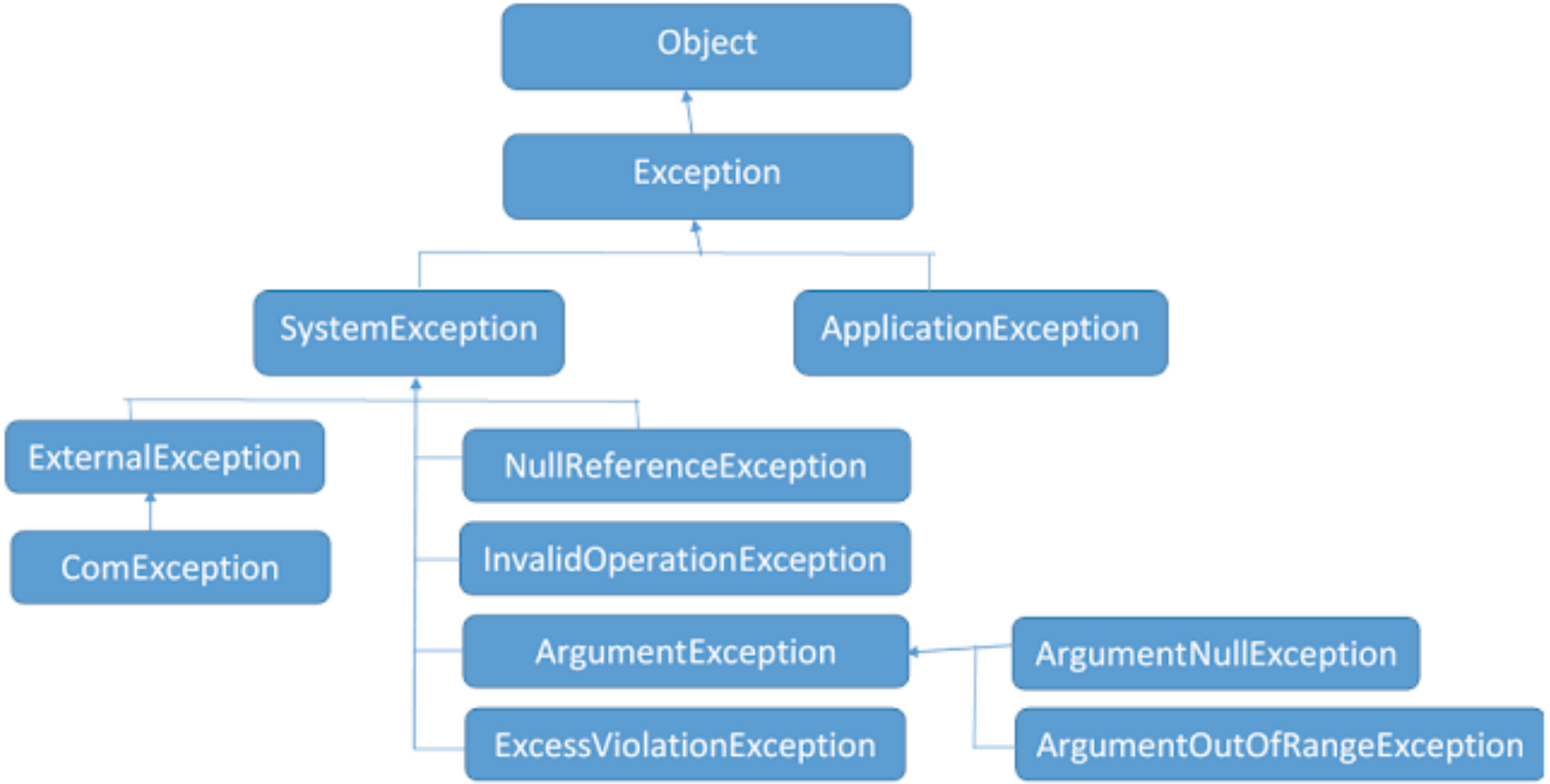
ИСКЛЮЧЕНИЯ

Исключение — это то, вероятность (возможность) чего исключается системой… это то что в условиях программы произойти не может. Например, при передачи файла по сети может неожиданно оборваться сетевое подключение.

Строго говоря, **исключения — это конструкция языка позволяющая управлять потоком выполнения**.

Исключение != ошибка

ИЕРАРХИЯ ИСКЛЮЧЕНИЙ

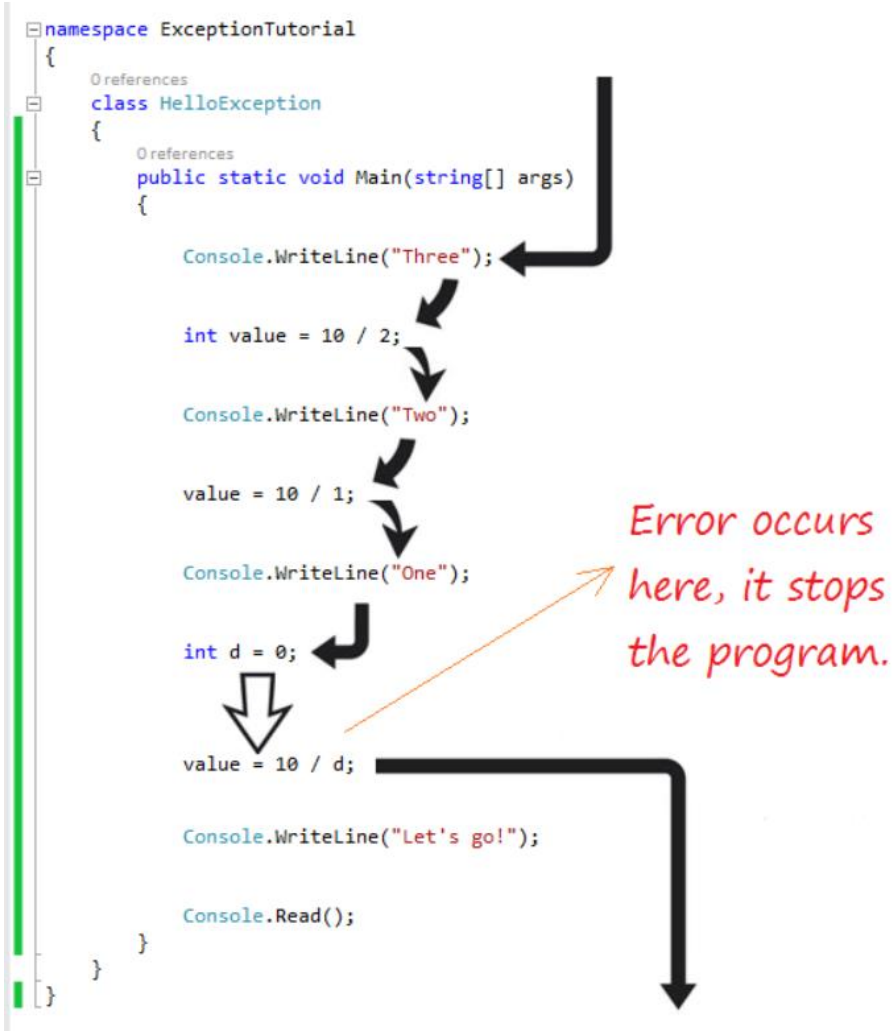


КЛАСС EXCEPTION

Базовым для всех типов исключений является тип **Exception**. Этот тип определяет ряд свойств, с помощью которых можно получить информацию об исключении.

- **Data** — подробная информация по исключению.
- **InnerException** — подробная информация по исключению
- **Message** — хранит сообщение об исключении, текст ошибки
- **Source** — хранит имя объекта или сборки, которое вызвало исключение
- **StackTrace** — последовательность вызовов методов, в результате которой возникла ошибка.

ОБРАБОТКА ИСКЛЮЧЕНИЙ



ОБРАБОТКА ИСКЛЮЧЕНИЙ

Для обработки исключений в C# предназначена конструкция **try...catch...finally**

```
static void Main(string[] args)
{
    try
    {
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Исключение: {ex.Message}");
        Console.WriteLine($"Метод: {ex.TargetSite}");
        Console.WriteLine($"Трассировка стека: {ex.StackTrace}");
    }

    Console.Read();
}
```

ОПРЕДЕЛЕНИЕ БЛОКА CATCH

За обработку исключения отвечает блок **catch**, который может иметь следующие формы:

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

Обрабатывает любое исключение, которое возникло в блоке try.

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

Обрабатывает только те исключения, которые соответствуют типу, указанному в скобках после оператора catch.

ОПРЕДЕЛЕНИЕ БЛОКА CATCH


```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch(DivideByZeroException)
{
    Console.WriteLine("Возникло исключение DivideByZeroException");
}
```

ОПРЕДЕЛЕНИЕ БЛОКА CATCH

Обрабатывает только те исключения, которые соответствуют типу, указанному в скобках после оператора catch. А вся информация об исключении помещается в переменную данного типа:

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

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch(DivideByZeroException ex)
{
    Console.WriteLine($"Возникло исключение {ex.Message}");
}
```

ФИЛЬТРЫ ИСКЛЮЧЕНИЙ

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.

```
int x = 1;
int y = 0;

try
{
    int result = x / y;
}
catch(DivideByZeroException) when (y==0 && x == 0)
{
    Console.WriteLine("y не должен быть равен 0");
}
catch(DivideByZeroException ex)
{
    Console.WriteLine(ex.Message);
}
```

ГЕНЕРАЦИЯ ИСКЛЮЧЕНИЙ И ОПЕРАТОР THROW

C# также позволяет генерировать исключения вручную с помощью оператора **throw**. То есть с помощью этого оператора мы сами можем создать исключение и вызвать его в процессе выполнения.

```
static void Main(string[] args)
{
    try
    {
        Console.Write("Введите строку: ");
        string message = Console.ReadLine();
        if (message.Length > 6)
        {
            throw new Exception("Длина строки больше 6 символов");
        }
    }
    catch (Exception e)
    {
        Console.WriteLine($"Ошибка: {e.Message}");
    }
    Console.Read();
}
```

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ИСКЛЮЧЕНИЙ

Если ни одно из стандартных исключений не подходит, можно создать собственные классы исключений путем наследования от класса Exception.

```
using System;

public class EmployeeListNotFoundException : Exception
{
    public EmployeeListNotFoundException()
    {
    }

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

    public EmployeeListNotFoundException(string message, Exception inner)
        : base(message, inner)
    {
    }
}
```

ЛУЧШИЕ МЕТОДИКИ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Обработка общих условий без выдачи исключений

Для условий, которые могут возникнуть, но способны вызвать исключение, рекомендуется реализовать обработку таким способом, который позволит избежать исключения. Например, при попытке закрыть уже закрытое подключение возникает *InvalidOperationException*. Этого можно избежать, используя оператор if для проверки состояния подключения перед попыткой закрыть его.

```
if (conn.State != ConnectionState.Closed)
{
    conn.Close();
}
```

```
try
{
    conn.Close();
}
catch (InvalidOperationException ex)
{
    Console.WriteLine(ex.GetType().FullName);
    Console.WriteLine(ex.Message);
}
```

ЛУЧШИЕ МЕТОДИКИ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Использование предопределенных типов исключений .NET

Создавайте новый класс исключений, только если предопределенное исключение не подходит. Пример:

1. Вызывайте исключение InvalidOperationException, если значение свойства или вызов метода не соответствуют текущему состоянию объекта.

2. Порождайте исключение ArgumentException или одного из предварительно определенных классов, которые являются производными от ArgumentException, если передаются недопустимые параметры.

ЛУЧШИЕ МЕТОДИКИ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Завершайте имена классов исключений словом Exception.

Если требуется пользовательское исключение, присвойте ему соответствующее имя и сделайте его производным от класса Exception.

```
public class MyFileNotFoundException : Exception
{
}
```

ЛУЧШИЕ МЕТОДИКИ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Включение трех конструкторов в пользовательские классы исключений.

- **Exception(),** использующий значения по умолчанию.
- **Exception(String),** принимающий строковое сообщение.
- **Exception(String, Exception),** принимающий строковое сообщение и внутреннее исключение.

ЛУЧШИЕ МЕТОДИКИ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Использование грамматически правильных сообщений об ошибке.

Составляйте понятные предложения, указывая в конце знаки препинания. Каждое предложение в строке, назначенной свойству **Exception.Message**, должно заканчиваться точкой.

Например***, "Таблица журнала переполнена."*** будет подходящей строкой сообщения.

ЛУЧШИЕ МЕТОДИКИ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

Предоставление дополнительных свойств в пользовательских исключениях по мере необходимости

Дополнительные сведения (кроме строки настраиваемого сообщения) включайте в исключение только в случаях, когда в соответствии со сценарием программирования такие дополнительные сведения могут оказаться полезными.

Например,

исключение FileNotFoundException предоставляет свойство FileName.

```
catch (FileNotFoundException)
{
    Console.WriteLine($"File {fileName} was not found.");
}
```