**Лабораторная работа №13.1**

**Тема:** «Разработка программ с использованием исключениями»

**Цель работы**: Сформировать умения работать с исключениями.

Пусть в классе объявляются методы A и B. При этом из метода A вызывается метод B, который выполняет свою работу, возможно, возвращает результаты. В теле метода A есть точка вызова метода B, и точка возврата, в которой оказывается управление после успешного возвращения из метода B. Если всё хорошо, метод A, возможно, анализирует полученные результаты и продолжает свою работу непосредственно из точки возврата. Если при выполнении метода B возникла исключительная ситуация (например, целочисленное деление на 0), возможно, что метод A узнает об этом, анализируя возвращаемое из B значение. Таким может быть один из сценариев “обратной связи”, при котором вызывающий метод узнаёт о результатах деятельности вызывающего метода. Недостатки этого сценария заключаются в том, что: метод B может в принципе не возвращать никаких значений, среди множества возвращаемых методом B значений невозможно выделить подмножество значений, которые можно было бы воспринимать как уведомление об ошибке, работа по подготовке уведомления об ошибке требует неоправданно больших усилий.

Решение проблемы состоит в том, что в среде выполнения поддерживается модель обработки исключений, основанная на понятиях объектов исключения и защищенных блоков кода. Следует отметить, что схеме обработки исключений не нова и успешно реализована во многих языках и системах программирования.

Некорректная ситуация в ходе выполнения программы (деление на нуль, выход за пределы массива) рассматривается как исключительная ситуация, на которую метод, в котором она произошла, реагирует

ГЕНЕРАЦИЕЙ ИСКЛЮЧЕНИЯ, а не обычным возвращением значения, пусть даже изначально ассоциированного с ошибкой. Среда выполнения создает объект для представления исключения при его возникновении. Одновременно с этим прерывается обычный ход выполнения программы. Происходит так называемое разматывание стека, при котором управление НЕ оказывается в точке возврата и если ни в одном из методов, предшествующих вызову, не было предпринято предварительных усилий по ПЕРЕХВАТУ ИСКЛЮЧЕНИЯ, приложение аварийно завершается.

Можно писать код, обеспечивающий корректный перехват исключений, можно создать собственные классы исключений, получив производные классы из соответствующего базового исключения. Все языки программирования, использующие среду выполнения, обрабатывают исключения одинаково. В каждом языке используется форма try/catch/finally для структурированной обработки исключений. Следующий пример демонстрирует основные принципы организации генераторов и перехватчиков исключений.

```
using System;
// Объявление собственного исключения.
// Наследуется базовый класс Exception.
public class xException : Exception
{ // Собственное исключение имеет специальную строчку
  // и в этом её отличие.
    public string xMessage;
    // Кроме того, в поле её базового элемента
    // также фиксируется особое сообщение.
    public xException(string str) : base("xException is here...")
    {
        xMessage = str;
    }
}
// Объявление собственного исключения. // Наследуется базовый класс Exception.
public class MyException : Exception
{
    // Собственное исключение имеет специальную строчку
    // и в этом её отличие.
    public string MyMessage;
    // Кроме того, в поле её базового элемента
    // также фиксируется особое сообщение.
    public MyException(string str) : base("MyException is here...")
    {
        MyMessage = str;
    }
}
public class StupidCalcule
{
    public int Div(int x1, int x2)
    {
```

```
        // Вот здесь метод проверяет корректность операндов и с помощью оператора
        // throw возбуждает исключение.
        if (x1 != 0 && x2 == 0)
            throw new Exception("message from Exception: Incorrect x2!");
        else if (x1 == 0 && x2 == 0)
            throw new MyException("message from MyException: Incorrect x1 &&
x2!");
        else if (x1 == -1 && x2 == -1)
            throw new xException("message from xException: @#$%*&^???");
        // Если же ВСЁ ХОРОШО, счастливый заказчик получит ожидаемый результат.
        return (int)(x1 / x2);
    }
}
public class Runner
{
    public static void Main()
    {
        int ret = 0;
        StupidCalcule sc = new StupidCalcule();
        // Наше приложение специально подготовлено к обработке исключений!
        // Потенциально опасное место (КРИТИЧЕСКАЯ СЕКЦИЯ) ограждено
        // (заключено в блок try)
        try
        { // Вызов...  ret = sc.Div(-1,-1);
            // Если ВСЁ ХОРОШО - будет выполнен оператор
            Console.WriteLine();
            // Потом операторы блока finally, затем операторы за пределами
            // пределами блоков try, catch, finally.
            Console.WriteLine($"OK, ret == {ret}");
        }
        catch (MyException e)
        {
            // Здесь перехватывается MyException.  Console.WriteLine(Exception e);
            Console.WriteLine(e.MyMessage);
        }
        // Если этот блок будет закомментирован -
        // возможные исключения типа Exception и xException
        // окажутся неперехваченными. И после блока
        // finally приложение аварийно завершится.
        catch (Exception e)
        { // А перехватчика xException у нас нет!
          // Поэтому в этом блоке будут перехватываться
          // все ранее неперехваченные потомки исключения Exception.
          // Это последний рубеж.
          // Ещё один вариант построения последнего рубежа
          // выглядит так: //
        }
        catch
        {
            //{ //  Операторы обработки - сюда. //}
            Console.WriteLine("Без параметров");
        }
        finally
        {  // Операторы в блоке finally выполняются ВСЕГДА, тогда как
           // операторы, расположенные за пределами блоков try, catch, finally,
           // могут и не выполниться вовсе.
            Console.WriteLine("finally block is here!");
        } // Вот если блоки перехвата исключения окажутся не
          // соответствующими возникшему исключению - нормальное
          // выполнение приложения будет прервано - и мы никогда не увидим // этой
надписи.
        Console.WriteLine("Good Bye!");
    }
}
```

Основные системные исключения:

*DivideByZeroException* Предпринята попытка деления на ноль

*IndexOutOfRangeException* Индекс массива выходит за пределы диапазона

*InvalidCastException* Некорректное преобразование в процессе выполнения

*OutOfMemoryException* Вызов new был неудачным из-за недостатка памяти

*Overflow*/*Exception* Переполнение при выполнении арифметической операции

*StackOverflowException* Переполнение стека

*NullReferanceExeption* ссылается на пустую (null) ссылку.

**Порядок выполнения работы:**

1) Разработать класс, обязательными членами которого должны являться: конструктор, поля, методы, свойства;

2) Создать свой собственный тип исключения и сгенерировать в заданной ситуации. Обеспечить возможность её возникновения.

3) Произвести перехват исключения в методе Main, а также разработать еще один статический метод в классе Program для перехвата исключения.

4) Реализовать программу на C# в соответствии с вариантом исполнения.

5) Выдать значения StackTrace для первого перехваченного исключения и для второго. Проанализировать результаты.

**Варианты**

1. Калькулятор 1 байтный знаковый. **Операции** +,-.*,/ . Свои исключения: переполнения разрядов, заем из старшего разряда.
2. Арифметическая прогрессия. Можно вводить элементы, считать сумму членов. Получать N-й член последовательности. Свои исключения: не более 10 членов.
3. Геометрическая прогрессия. Можно вводить элементы, считать сумму членов. Свои исключения: q не должен быть 0м.
4. Плоские фигуры. Содержит массив пространственных точек. Можно менять их количество. Можно менять заданную точку. Метод нахождения площади. Свои исключения: сделать возможным передачи 3й координаты, но выводить исключение.
5. Драйвер устройство получает двоичный код выдает соответствующий ему символ либо если указана другая настройка 16ричное число, либо 10е число. Генерировать исключения, если получает не двоичный код.

6. Функциональный калькулятор умеющий считать Sin,Cos, tg,ctg, логорифм по заданному основанию, натуральный и десятичный, возведение в любую степень, отдельно корень любой степени. +,-,*,/ . Разработать меню. Выдавать исключения, когда математические операции выполнить невозможно.

7. Класс круг. Свойства центр и радиус. Операции ++ и --, увеличивают и уменьшают радиус круга. Выдавать исключения если радиус меньше либо равен 0, так же при установке свойства радиус.

8. Класс человек. Свойства :Возраст имя пол(перечисления). Генерировать свои исключения при отрицательном возрасте, при несоответствии пола, при передачи в имя цифр.

Контрольные вопросы:

1) Объясните работу механизма исключительных ситуаций в C#?

2) При помощи, каких ключевых слов можно обработать исключение?

3)Условия перехвата исключения?

4) Перечислите базовые исключения.