Лабораторная работа №12

Тема: Обработка исключительных ситуаций.

Цель: Научиться обрабатывать исключительные ситуации, возникновение которых возможно в процессе работы программы.

Теоретические сведения:

Исключения (Exceptions) это тип ошибки, которая происходит при выполнении приложения. Ошибки обычно означают появление неожиданных проблем. Тогда как исключения, обработка которых организована в коде, являются ожидаемыми, они происходят в коде приложений по различным причинам.

Исключения позволяют передать управление из одной части кода в другую часть. Когда срабатывает/выбрасывается исключение (exception thrown), текущий поток выполнения кода секции try прерывается, и запускается выполнение секции catch.

Обработка исключений C# осуществляется следующими ключевыми словами**: try, catch, finally** и **throw**.

try – блок try инкапсулирует проверяемый на исключение регион кода. Если любая строка кода в этом блоке вызывает срабатывание исключения, то исключение будет обработано соответствующим блоком catch.

catch – когда происходит исключение, запускается блок кода catch. Это то место, где можно обработать исключения и предпринять адекватные для него действия, например записать событие ошибки в лог, прервать работу программы, или может просто игнорировать исключение (когда блок catch пустой).

finally – блок finally позволяет выполнить какой-то определенный код приложения, если исключение сработало, или если оно не сработало.

Часто блок finally опускается, когда обработка исключения подразумевает нормальное дальнейшее выполнение программы – потому блок finally может быть просто заменен обычным кодом, который следует за блоками try и catch.

throw – ключевое слово throw используется для реального создания нового исключения, в результате чего выполнение кода попадет в соответствующие блоки catch и finally.

Пример 1. Обработка деления на ноль.

Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение. Рассмотрим код:

```
int x = 5;
int y = 0;
int z = x / y;
Console.WriteLine($"Результат: {z}");
Console.WriteLine("Конец программы");
Console.Read();
```

Чтобы избежать аварийного завершения программы, следует использовать для обработки исключений конструкцию try…catch:

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

            Console.WriteLine("Конец программы");
            Console.Read();
```

На языке C# ключевые слова try и catch используются для определения блока проверяемого кода (блок try catch). Блок try catch помещается вокруг кода, который потенциально может выбросить исключение.

Если произошло исключение, то этот блок try catch обработает исключение, гарантируя тем самым, что приложение не выдаст ошибку необработанного исключения (unhandled exception), ошибки пользователя, и эта ошибка не разрушит процесс выполнения приложения.

Ряд исключительных ситуаций может быть предвиден разработчиком. Например, пусть программа предусматривает ввод числа и вывод его квадрата:

```
Console.WriteLine(“Введите число”);
int x = Int32.Parse(Console.ReadLine());
 x *= x;
Console.WriteLine(“Квадрат числа: ” + x);
Console.Read();
```

Если пользователь введет не число, а строку, какие-то другие символы, то программа выпадет в ошибку.

С одной стороны, здесь как раз та ситуация, когда можно применить блок try..catch, чтобы обработать возможную ошибку. Однако гораздо оптимальнее было бы проверить допустимость преобразования:

```
Console.WriteLine("Введите число");
            int x;
            string input = Console.ReadLine();
            if (Int32.TryParse(input, out x))
            {
                x *= x;
                Console.WriteLine("Квадрат числа: " + x);
            }
            else
            {
                Console.WriteLine("Некорректный ввод");
            }
            Console.Read();
```

Метод Int32.TryParse() возвращает true, если преобразование можно осуществить, и false – если нельзя.

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:

```
catch when(условие)
{

}
```

В этом случае обработка исключения в блоке catch производится только в том случае, если условие в выражении when истинно. Например:

```
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);
            }
```

В данном случае будет выброшено исключение, так как y=0. Здесь два блока catch, и оба они обрабатывают исключения типа DivideByZeroException, то есть по сути все исключения, генерируемые при делении на ноль.

Но поскольку для первого блока указано условие y == 0 && x == 0, то оно не будет обрабатывать исключение – условие, указанное после оператора when возвращает false.

Поэтому CLR будет дальше искать соответствующие блоки catch далее и для обработки исключения выберет второй блок catch. В итоге если мы уберем второй блок catch, то исключение вообще не будет обрабатываться.

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

InnerException: хранит информацию об исключении, которое послужило причиной текущего исключения;

Message: хранит сообщение об исключении, текст ошибки;

Source: хранит имя объекта или сборки, которое вызвало исключение;

StackTrace: возвращает строковое представление стека вызовов, которые привели к возникновению исключения;

TargetSite: возвращает метод, в котором и было вызвано исключение.

Например, обработаем исключения типа Exception:

```
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();
```

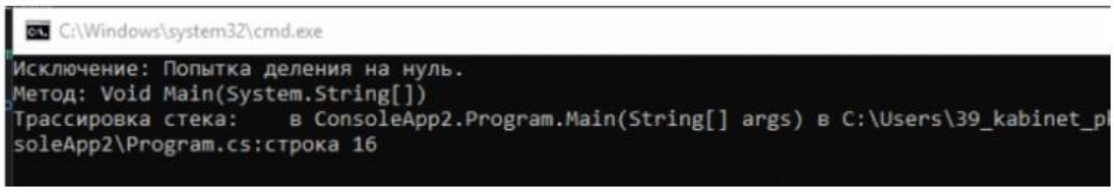

Рисунок 1 – Окно выполнения программы

Если нужно определить код, который будет выполняться после выхода из блока try/catch, тогда нужно использовать блок finally. Использование этого блока гарантирует, что некоторый набор операторов будет выполняться всегда, независимо от того, возникло исключение (любого типа) или нет.

Блок finally выполняется и в том случае, если любой код в блоке try или в связанном с ним блоках catch приводит к возврату их метода. Пример:

```
int x = 10, y = 0, z;
            try
            {
                z = x / y;
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Деление на 0");
            }
            finally
            {
                Console.WriteLine("Конец программы");
            }
```

Язык C# позволяет генерировать исключения вручную с помощью оператора throw. То есть с помощью этого оператора можно создать исключение и вызвать его в процессе выполнения.

Например, в нашей программе происходит ввод строки, и мы хотим, чтобы, если длина строки будет больше 8 символов, возникало исключение:

```
try
            {
                Console.Write("Введите строку: ");
                string message = Console.ReadLine();
                if (message.Length > 8)
                {
                    throw new Exception("Длина строки больше 8 символо
в");
                }
            }
            catch (Exception e)
            {
```

```
            Console.WriteLine($"Ошибка: {e.Message}");
        }
        Console.Read();
```

Подобным образом можно генерировать исключения в любом месте программы. Но существует также и другая форма использования оператора throw, когда после данного оператора не указывается объект исключения. В подобном виде оператор throw может использоваться только в блоке catch.

```
try
        {
            try
            {
                Console.Write("Введите строку: ");
                string message = Console.ReadLine();
                if (message.Length > 8)
                {
                    throw new Exception("Длина строки больше 8 сим
волов");
                }
            }
            catch
            {
                Console.WriteLine("Возникло исключение");
                throw;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
```

Выполнение работы:

Модернизируйте программу из лабораторной работы №10, предусмотрев все исключительные ситуации и обработав их.

Содержание отчета:

1. Номер и тема лабораторной работы.
2. Цель лабораторной работы.
3. Техническое оснащение.
4. Скриншоты выполнения примера.
5. При выполнении индивидуальных заданий в отчет внести изображение кода программы и окно выполнения программы.
6. Вывод по лабораторной работе.
7. Краткий конспект теоретических сведений.