

Обработка исключений в Си-шарп. Оператор try-catch

В предыдущих уроках, в некоторых программах мы не учитывали непредвиденные ситуации, которые могут приводить к ошибкам. Например, когда нам необходимо было ввести число. Если вместо числа мы ввели бы строку, то при конвертации этой строки в численный тип программа бы аварийно завершила работу, и мы получили бы ошибку.

Такие ошибки и другие непредвиденные ситуации в Си-шарп называются **исключениями**.

Обработка исключений – это описание реакции программы на подобные события (исключения) во время выполнения программы. Реакцией программы может быть корректное завершение работы программы, вывод информации об ошибке и запрос повторения действия (при вводе данных).

Примерами исключений может быть:

- деление на ноль;
- конвертация некорректных данных из одного типа в другой;
- попытка открыть файл, которого не существует;
- доступ к элементу вне рамок массива;
- исчерпывание памяти программы;
- другое.

Для обработки исключений в Си-шарп используется оператор **try-catch**. Он имеет следующую структуру:

```
try
{
    //блок кода, в котором возможно исключение
}
catch ([тип исключения] [имя])
{
    //блок кода – обработка исключения
}
```

Работает это все очень просто. Выполняется код в блоке try, и, если в нем происходит исключение типа, соответствующего типу, указанному в catch, то управление передается блоку catch. При этом, весь оставшийся код от момента выбрасывания исключения до конца блока try не будет выполнен. После выполнения блока catch, оператор try-catch завершает работу.

Указывать имя исключения не обязательно. Исключение представляет собою

объект, и к нему мы имеем доступ через это имя. С этого объекта мы можем получить, например, стандартное сообщение об ошибке (Message), или трассировку стека (StackTrace), которая поможет узнать место возникновения ошибки. В этом объекте хранится детальная информации об исключении.

Если тип выброшенного исключения не будет соответствовать типу, указанному в catch – исключение не обрабатывается, и программа завершит работу аварийно.

Ниже приведен пример программы, в которой используется обработка исключения некорректного формата данных:

```
static void Main(string[] args)
{
    string result = "";
    Console.WriteLine("Введите число:");
    try
    {
        int a = Convert.ToInt32(Console.ReadLine()); //вводим данные, и
        конвертируем в целое число
        result = "Вы ввели число " + a;
    }
    catch (FormatException)
    {
        result = "Ошибка. Вы ввели не число";
    }
    Console.WriteLine(result);
    Console.ReadLine();
}
```

Типы исключений

Ниже приведены некоторые из часто встречаемых типов исключений.

Exception – базовый тип всех исключений. Блок catch, в котором указан тип Exception будет «ловить» все исключения.

FormatException – некорректный формат операнда или аргумента (при передаче в метод).

NullReferenceException - В экземпляре объекта не задана ссылка на объект, объект не создан

IndexOutOfRangeException – индекс вне рамок коллекции

FileNotFoundException – файл не найден.

DivideByZeroException – деление на ноль

Несколько блоков catch

Одному блоку try может соответствовать несколько блоков catch:

```
try
{
    //блок1
}
catch (FormatException)
{
    //блок-обработка исключения 1
}
catch (FileNotFoundException)
{
    //блок-обработка исключения 2
}
```

В зависимости от того или другого типа исключения в блоке try, выполнение будет передано соответствующему блоку catch.

Блок finally

Оператор try-catch также может содержать блок finally. Особенность блока finally в том, что код внутри этого блока выполнится в любом случае, в независимости от того, было ли исключение или нет.

```
try
{
    //блок1
}
catch (Exception)
{
    //обработка исключения
}
finally
{
    //блок кода, который выполнится обязательно
}
```

Выполнение кода программы в блоке finally происходит в последнюю очередь. Сначала try затем finally или catch-finally (если было исключение).

Обычно, он используется для освобождения ресурсов. Классическим примером использования блока finally является закрытие файла.

Зачем блок finally?

Очень часто можно услышать вопрос, для чего нужен этот блок? Ведь, кажется,

можно освободить ресурсы просто после оператора try-catch, без использования finally. А ответ очень прост. Finally **гарантирует** выполнение кода, несмотря ни на что. Даже если в блоках try или catch будет происходить выход из метода с помощью оператора return – finally выполнится.

Операторы try-catch также могут быть вложенными. Внутри блока try либо catch может быть еще один try-catch.

Обработка исключений, в первую очередь, нам понадобится при работе с файлами. Работе с файлами будет посвящен следующий урок.