Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Например, при передачи файла по сети может неожиданно оборваться сетевое подключение. такие ситуации называются **исключениями**. Язык C# предоставляет разработчикам возможности для обработки таких ситуаций. Для этого в C# предназначена конструкция **try...catch...finally**.

```
try
{

}
catch
{

}
finally
{

}
```

При использовании блока **try...catch..finally** вначале выполняются все инструкции в блоке **try**. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок **finally**. И затем конструкция try..catch..finally завершает свою работу.

Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок **catch**, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.

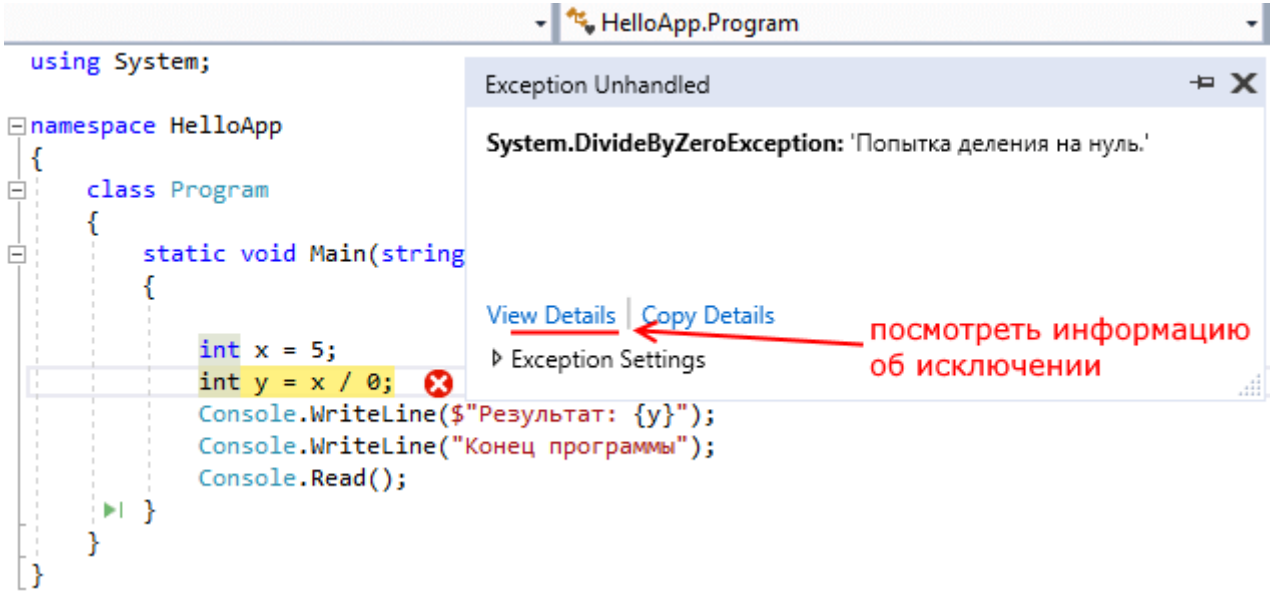
Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

Рассмотрим следующий пример:

```
class Program
{
    static void Main(string[] args)
    {
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
        Console.WriteLine("Конец программы");
        Console.Read();
    }
```

```
11  }
```

В данном случае происходит деление числа на 0, что приведет к генерации исключения. И при запуске приложения в режиме отладки мы увидим в Visual Studio окошко, которое информирует об исключении:



В этом окошке мы видим, что возникло исключение, которое представляет тип **System.DivideByZeroException**, то есть попытка деления на ноль. С помощью пункта **View Details** можно посмотреть более детальную информацию об исключении.

И в этом случае единственное, что нам остается, это завершить выполнение программы.

Чтобы избежать подобного аварийного завершения программы, следует использовать для обработки исключений конструкцию**try...catch...finally**. Так, перепишем пример следующим образом:

```
1   class Program
2   {
3       static void Main(string[] args)
4       {
5           try
6           {
7               int x = 5;
8               int y = x / 0;
9               Console.WriteLine($"Результат: {y}");
10          }
```

```
        catch
        {
            Console.WriteLine("Возникло исключение!");
        }
        finally
        {
            Console.WriteLine("Блок finally");
        }
        Console.WriteLine("Конец программы");
        Console.Read();
    }
```

В данном случае у нас опять же возникнет исключение в блоке try, так как мы пытаемся разделить на ноль. И дойдя до строки

```
int y = x / 0;
```

выполнение программы остановится. CLR найдет блок **catch** и передаст управление этому блоку.

После блока catch будет выполняться блок finally.

```
Возникло исключение!
Блок finally
Конец программы
```

Таким образом, программа по-прежнему не будет выполнять деление на ноль и соответственно не будет выводить результат этого деления, но теперь она не будет аварийно завершаться, а исключение будет обрабатываться в блоке catch.

Следует отметить, что в этой конструкции обязателен блок **try**. При наличии блока catch мы можем опустить блок finally:

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

И, наоборот, при наличии блока finally мы можем опустить блок catch и не обрабатывать исключение:

```
try
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
finally
{
    Console.WriteLine("Блок finally");
}
```

Однако, хотя с точки зрения синтаксиса C# такая конструкция вполне корректна, тем не менее, поскольку CLR не сможет найти нужный блок catch, то исключение не будет обработано, и программа аварийно завершится.

Обработка исключений и условные конструкции

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

```
static void Main(string[] args)
{
    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:

```
static void Main(string[] args)
{
    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. При допустимости преобразования переменная x будет содержать введенное число. Так, не используя `try...catch` можно обработать возможную исключительную ситуацию.

С точки зрения производительности использование блоков `try..catch` более накладно, чем применение условных конструкций. Поэтому по возможности вместо try..catch лучше использовать условные конструкции на проверку исключительных ситуаций.