Глава 3. Обработка исключений: проект EXCEP

Проект EXCEP знакомит с приемами обработки исключительных ситуаций (исключений). Описывается структура `try`-блоков и демонстрируются особенности, связанные с использованием вложенных `try`-блоков. Дается обзор исключений, связанных с арифметическими операциями и связанные с ними операторы `checked` и `unchecked`. Описываются действия, которые можно выполнить при возникновении исключения в различных режимах запуска приложения. Рассматривается метод `Parse` преобразования строки в число и оператор `throw` возбуждения исключения (в том числе повторного). Описывается вариант `try`-блока, использующий раздел `finally`.

3.1. Обработка конкретного исключения и групп исключений

В этом примере, как и в предыдущем, будет разрабатываться консольное приложение. Создайте заготовку проекта для консольного приложения (как в разд. 2.1) и измените описание класса `Program` в файле Program.cs (листинг 3.1).

Листинг 3.1. Описание класса `Program`

```
class Program
{
  static void M1(int x, int y, int z)
  {
    try
    {
      int a = checked((int)Math.Pow(x, y));
      Console.WriteLine("x ^ y / z = {0}", a / z);
    }
    catch (DivideByZeroException)
    {
      Console.WriteLine("DivideByZero Exception");
    }
    Console.WriteLine("M1 finished");
  }
  static void M2(int x, int y, int z)
  {
    try
    {
      M1(x, y, z);
    }
    catch (ArithmeticException)
    {
      Console.WriteLine("Arithmetic Exception");
    }
    Console.WriteLine("M2 finished");
  }
  static void Main(string[] args)
  {
    Console.Write("x = ");
    int x = int.Parse(Console.ReadLine());
    Console.Write("y = ");
    int y = int.Parse(Console.ReadLine());
    Console.Write("z = ");
    int z = int.Parse(Console.ReadLine());
    M2(x, y, z);
    Console.ReadLine();
  }
}
```

Результат: для трех введенных целых чисел *x*, *y*, *z* программа вычисляет выражение x^y/z, обрабатывая возникающие при этом *исключительные ситуации* (для выхода из программы надо нажать <Enter>). Опишем различные варианты работы программы.

Вариант А. Обработка допустимых значений (листинг 3.2). Вычисления выполняются успешно; ни один обработчик исключений не активизируется.

Листинг 3.2. Содержимое консольного окна при обработке допустимых значений

```
x = 9
y = 2
z = 3
```

```
x ^ y / z = 27
M1 finished
M2 finished
```

Вариант B. Деление на 0 (листинг 3.3). Активизируется обработчик `try`-блока метода `M1`, обрабатывающий исключение типа `DivideByZeroException`, после чего выполнение программы продолжается с оператора, следующего за данным `try`-блоком.

Листинг 3.3. Содержимое консольного окна при делении на 0

```
x = 1
y = 1
z = 0
DivideByZero Exception
M1 finished
M2 finished
```

Вариант C. Целочисленное переполнение (листинг 3.4). При попытке возведения числа 10 в степень 10 (и последующего преобразования результата к целому типу) возникает исключение `OverflowException`. Поскольку в разделе `catch` `try`-блока метода `M1` обработка исключения `OverflowException` не предусмотрена, происходит немедленный переход в обработчик `try`-блока следующего уровня (то есть в раздел `catch` `try`-блока метода `M2`). Здесь исключение `OverflowException` обрабатывается, так как оно является *потомком* исключения `ArithmeticException` — предка всех исключений, порожденных ошибками при выполнении арифметических операций. После этой обработки выполнение программы продолжается с оператора, следующего за `try`-блоком метода `M2`.

Листинг 3.4. Содержимое консольного окна при целочисленном переполнении

```
x = 10
y = 10
z = 1
Arithmetic Exception
M2 finished
```

D. Ввод недопустимого символа. После ввода недопустимого символа (например, звездочки `*`) выполнение программы *немедленно прерывается,* происходит возврат в среду Visual C# и в тексте программы выделяется оператор, выполнение которого привело к возникновению исключения (в нашем случае это будет первый из операторов метода `Main`, содержащих вызов функции `Parse`). Такое поведение программы объясняется тем, что в методе `Main` не предусмотрена обработка возникшего исключения `FormatException`, и поэтому активизируется режим обработки исключения по умолчанию.

В данной ситуации возможны два варианта действий:

1. Немедленно прервать выполнение программы, нажав комбинацию <Shift>+<F5> или кнопку с изображением синего квадратика  .
2. Продолжить выполнение программы, пропустив ошибочный оператор и, возможно, несколько следующих операторов. Для этого надо зацепить мышью желтую стрелку  , расположенную рядом с ошибочным оператором, и перетащить ее к тому оператору, с которого надо продолжить выполнение программы, после чего нажать клавишу <F5> или кнопку с изображением зеленого треугольника  . Можно также перейти к *пошаговому выполнению программы*, нажимая кнопки  (или клавишу <F11>) и  (или клавишу <F10>). Любая из этих кнопок обеспечивает выполнение текущего оператора (то есть оператора, на который указывает желтая стрелка). Отличие между ними состоит в том, что если текущий оператор является вызовом функции и код этой функции доступен, то первая кнопка (кнопка **Step Into**) обеспечивает переход на начало этой функции, а вторая (кнопка **Step Over**) немедленно выполняет функцию и переходит к следующему за ней оператору.

Примечание

Поведение программы, описанное в ситуации D, соответствует обработке исключений по умолчанию в режиме включенного отладчика (то есть при запуске программы с помощью клавиши <F5>). Если программа запущена в режиме с отключенным отладчиком (то есть с помощью комбинации <Ctrl>+<F5>), то при возникновении исключения на экране появляется диалоговое окно с информацией о возникшем исключении, содержащее две кнопки: **Continue** (для продолжения выполнения программы) и **Quit** (для ее немедленного завершения). В подобной ситуации не происходит перехода в редактор среды Visual C# и подсвечивания ошибочного оператора.

Комментарии

1. Использованный в программе метод `Parse` для типа `int` (см. листинг 3.1) позволяет преобразовать указанную строку к типу `int` (для успешности подобного преобразования необходимо, чтобы в строке содержалось изображение некоторого целого числа, возможно, дополненное слева и справа пробелами). Аналогичный метод имеется и у других числовых типов, в частности, у вещественного типа `double`. Следует иметь в виду, что при преобразовании строки в вещественное число символ разделителя дробной

части определяется из настроек операционной системы Windows, поэтому в программе, выполняемой в русской версии Windows, в качестве десятичного разделителя необходимо вводить запятую. Дополнительные сведения о языковых настройках и их изменении приводятся в комментарии 3 в разд. 6.5.

Если параметр метода `Parse` невозможно преобразовать к указанному числовому типу, то возбуждается исключение типа `FormatException` (см. выше описание варианта D).

2. Добиться исключения `OverflowException` при выполнении операций с вещественными числами невозможно: если получающееся число окажется слишком большим, то будет возвращено особое значение типа `double` —`double.PositiveInfinity` (положительная бесконечность), с которым можно выполнять различные действия так же, как и с обычными числами. Имеются еще два особых вещественных значения: `double.NegativeInfinity` — отрицательная бесконечность и `double.NaN` — не число. Стандартные математические функции, определенные в классе `Math`, могут возвращать как "обычные", так и особые числовые значения. Например, `Math.Sqrt(-1)` (квадратный корень) вернет `double.NaN`, а `Math.Log(0)` (натуральный логарифм) вернет `double.NegativeInfinity`. Заметим, что в нашей программе мы воспользовались функцией `Pow` из класса `Math`, позволяющей выполнить возведение в степень, причем переполнение, описанное в варианте C, возникало не при вычислении этой функции, а при попытке преобразования полученного результата к типу `int` (то есть имело место *целочисленное переполнение*).

3. Целочисленное переполнение тоже не всегда приводит к возбуждению исключения. При стандартных настройках проекта в подобной ситуации исключение не возбуждается, а слишком большое целочисленное значение урезается посредством отбрасывания лишних старших байтов. Если такое поведение при целочисленном переполнении нежелательно (как в нашем случае), то можно включить явный контроль за переполнением, заключив опасное выражение в конструкцию `checked` с круглыми скобками (см. листинг 3.1, метод `M1`). Эту конструкцию можно использовать для защиты не только выражений, но и группы операторов; в подобной ситуации ее синтаксис следующий:

`checked { `*операторы*` }`

Следует, однако, иметь в виду, что если среди указанных операторов имеются вызовы методов, то внутри этих методов контроль за целочисленным переполнением выполняться не будет.

Предусмотрена также парная конструкция `unchecked`, отключающая контроль за целочисленным переполнением. Благодаря отключенному контролю, операции над целочисленными данными выполняются существенно быстрее.

Установить контроль за целочисленным переполнением на уровне всего проекта можно, изменив его настройки. Для этого надо выполнить команду меню **Project** | *<имя проекта>* **Properties…**, перейти в появившемся вкладке с именем проекта в раздел **Build**, нажать кнопку **Advanced** и установить в появившемся окне флажок **Check for arithmetic overflow/underflow**.

3.2. Обработка любого исключения

Измените метод `Main` в файле Program.cs (листинг 3.5). Теперь программа содержит три вложенных `try`-блока.

Листинг 3.5. Новый вариант метода `Main`

```
static void Main(string[] args)
{
  try
  {
    Console.Write("x = ");
    int x = int.Parse(Console.ReadLine());
    Console.Write("y = ");
    int y = int.Parse(Console.ReadLine());
    Console.Write("z = ");
    int z = int.Parse(Console.ReadLine());
    M2(x, y, z);
  }
  catch
  {
    Console.WriteLine("Other exception");
  }
  Console.ReadLine();
}
```

Результат: в любом из вариантов A, B, C, рассмотренных в разд. 3.1, результат работы программы будет тем же самым. В варианте D (ввод недопустимого символа) в окне будет выведено

```
x = *
Other exception
```

и программа будет ожидать нажатия <Enter> для своего завершения. Таким образом, никакое исключение теперь не приведет к аварийному завершению программы.

Недочет: информация, выводимая на экран, не позволяет определить, какое именно исключение возникло в ходе выполнения программы.

Исправление: измените раздел `catch` в методе `Main` на следующий:

```
catch(Exception ex)
{
  Console.WriteLine(ex.GetType().Name + ":\n  " +
          ↳ ex.Message);
}
```

Результат: теперь в случае ввода недопустимого символа будет выведена более содержательная информация:

```
x = *
FormatException:
  Input string was not in a correct format.
```

Комментарий

В последнем варианте программы был определен обработчик для исключения `Exception` — общего предка всех исключений. Поэтому он активизируется при возникновении любого исключения, не обработанного в предыдущих `try`-блоках. Использование переменной `ex` типа `Exception` позволяет получить доступ к методам и свойствам возникшего исключения: имя класса-исключения можно получить с помощью выражения `ex.GetType().Name` (вызывается метод `GetType`, который возвращает объект типа `Type`, и для этого объекта вызывается свойство `Name`), краткое описание ошибки доступно с помощью свойства `Message` объекта `ex`.

3.3. Повторное возбуждение обработанного исключения

Дополните раздел `catch` для `try`-блока метода `M2`:

```
catch (ArithmeticException)
{
  Console.WriteLine("Arithmetic Exception");
  throw;
}
```

Результат: в любом из вариантов A, B, D результат работы программы будет тем же самым. В варианте C (целочисленное переполнение) в окне будет выведена более содержательная информация (листинг 3.6). Это связано с тем, что добавленный в раздел `catch` метода `M2` оператор `throw` выполнил *повторное возбуждение* только что обработанного исключения. Повторно возбужденное исключение было окончательно обработано в разделе `catch` метода `Main` (см. разд. 3.2).

Листинг 3.6. Содержимое консольного окна при целочисленном переполнении

```
x = 10
y = 10
z = 1
Arithmetic Exception
OverflowException:
  Arithmetic operation resulted in an overflow.
```

Комментарии

1. Оператор `throw` используется также для явного возбуждения исключения в программе. Например, если в некотором методе `M` параметр `n` целого типа может принимать только значения от `1` до `nMax`, то при нарушении этого условия следует возбудить в методе `M` исключение `ArgumentOutOfRangeException`:

```
if (n < 1 || n > nMax)
  throw new ArgumentOutOfRangeException("n");
```

Использованный вариант конструктора класса `ArgumentOutOfRangeException` позволяет указать имя ошибочного параметра в свойстве `Message` созданного исключения. В нашем случае данное свойство будет содержать текст: `Specified argument was out of the range of valid values. Parameter name: n` ("указанный аргумент находится вне диапазона допустимых значений. Имя параметра: `n`"). Пример использования оператора `throw` приводится также в разд. 30.1.

2. В `try`-блоке может содержаться несколько разделов `catch`, каждый из которых будет обрабатывать исключения определенного типа. Предусмотрен также дополнительный раздел `try`-блока с именем `finally`, который располагается после всех разделов `catch` и содержит код, выполняющий очистку ресурсов, выделенных в программе, и другие завершающие действия. Раздел `finally` выполняется *всегда*: и при нормальной работе операторов в блоке `try`, и при возбуждении исключения, причем даже в ситуации, если это исключение не было перехвачено в предыдущих разделах `catch`. Варианты `try`-блока с разделом `finally` приводятся в разд. 13.6—13.7.