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.

За обработку исключения отвечает блок **catch**, который может иметь следующие формы:

|  |  |
| --- | --- |
| 1  2  3  4 | catch  {      // выполняемые инструкции  } |

Обрабатывает любое исключение, которое возникло в блоке try. Выше уже был продемонстрирован пример подобного блока.

|  |  |
| --- | --- |
| 1  2  3  4 | catch (тип\_исключения)  {      // выполняемые инструкции  } |

Обрабатывает только те исключения, которые соответствуют типу, указаному в скобках после оператора catch.

Например, обработаем только исключения типа DivideByZeroException:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | try  {      int x = 5;      int y = x / 0;      Console.WriteLine($"Результат: {y}");  }  catch(DivideByZeroException)  {      Console.WriteLine("Возникло исключение DivideByZeroException");  } |

Однако если в блоке try возникнут исключения каких-то других типов, отличных от DivideByZeroException, то они не будут обработаны.

|  |  |
| --- | --- |
| 1  2  3  4 | catch (тип\_исключения имя\_переменной)  {      // выполняемые инструкции  } |

Обрабатывает только те исключения, которые соответствуют типу, указаному в скобках после оператора catch. А вся информация об исключении помещается в переменную данного типа. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | try  {      int x = 5;      int y = x / 0;      Console.WriteLine($"Результат: {y}");  }  catch(DivideByZeroException ex)  {      Console.WriteLine($"Возникло исключение {ex.Message}");  } |

Фактически этот случай аналогичен предыдущему за тем исключением, что здесь используется переменная. В данном случае в переменную ex, которая представляет тип DivideByZeroException, помещается информация о возникшем исключени. И с помощью свойства Message мы можем получить сообщение об ошибке.

Если нам не нужна информация об исключении, то переменную можно не использовать как в предыдущем случае.

**Фильтры исключений**

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:

|  |  |
| --- | --- |
| 1  2  3  4 | catch when(условие)  {    } |

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

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int x = 1;  int y = 0;    try  {      int result1 = x / y;      int result2 = y / x;  }  catch (DivideByZeroException) when (y == 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**, то именно этот блок будет обрабатывать данное исключение - условие, указанное после оператора **when** возвращает true.

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 вызывов, которые привели к возникновению исключения
* **TargetSite**: возвращает метод, в котором и было вызвано исключение

Например, обработаем исключения типа Exception:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | 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}");  } |

Однако так как тип Exception является базовым типом для всех исключений, то выражение catch (Exception ex) будет обрабатывать все исключения, которые могут возникнуть.

Но также есть более специализированные типы исключений, которые предназначены для обработки каких-то определенных видов исключений. Их довольно много, я приведу лишь некоторые:

* **DivideByZeroException**: представляет исключение, которое генерируется при делении на ноль
* **ArgumentOutOfRangeException**: генерируется, если значение аргумента находится вне диапазона допустимых значений
* **ArgumentException**: генерируется, если в метод для параметра передается некорректное значение
* **IndexOutOfRangeException**: генерируется, если индекс элемента массива или коллекции находится вне диапазона допустимых значений
* **InvalidCastException**: генерируется при попытке произвести недопустимые преобразования типов
* **NullReferenceException**: генерируется при попытке обращения к объекту, который равен null (то есть по сути неопределен)

И при необходимости мы можем разграничить обработку различных типов исключений, включив дополнительные блоки catch:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | static void Main(string[] args)  {      try      {          int[] numbers = new int[4];          numbers[7] = 9;     // IndexOutOfRangeException            int x = 5;          int y = x / 0;  // DivideByZeroException          Console.WriteLine($"Результат: {y}");      }      catch (DivideByZeroException)      {          Console.WriteLine("Возникло исключение DivideByZeroException");      }      catch (IndexOutOfRangeException ex)      {          Console.WriteLine(ex.Message);      }        Console.Read();  } |

В данном случае блоки catch обрабатывают исключения типов IndexOutOfRangeException и DivideByZeroException. Когда в блоке try возникнет исключение, то CLR будет искать нужный блок catch для обработки исключения. Так, в данном случае на строке

|  |  |
| --- | --- |
| 1 | numbers[7] = 9; |

происходит обращение к 7-му элементу массива. Однако поскольку в массиве только 4 элемента, то мы получим исключение типа IndexOutOfRangeException. CLR найдет блок catch, который обрабатывает данное исключение, и передаст ему управление.

Следует отметить, что в данном случае в блоке try есть ситуация для генерации второго исключения - деление на ноль. Однако поскольку после генерации IndexOutOfRangeException управление переходит в соответствующий блок catch, то деление на ноль int y = x / 0 в принципе не будет выполняться, поэтому исключение типа DivideByZeroException никогда не будет сгенерировано.

Однако рассмотрим другую ситуацию:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | try  {      object obj = "you";      int num = (int)obj;     // System.InvalidCastException      Console.WriteLine($"Результат: {num}");  }  catch (DivideByZeroException)  {      Console.WriteLine("Возникло исключение DivideByZeroException");  }  catch (IndexOutOfRangeException)  {      Console.WriteLine("Возникло исключение IndexOutOfRangeException");  } |

В данном случае в блоке try генерируется исключение типа InvalidCastException, однако соответствующего блока catch для обработки данного исключения нет. Поэтому программа аварийно завершит свое выполнение.

Мы также можем определить для InvalidCastException свой блок catch, однако суть в том, что теоретически в коде могут быть сгенерированы самыи разные типы исключений. А определять для всех типов исключений блоки catch, если обработка исключений однотипна, не имеет смысла. И в этом случае мы можем определить блок catch для базового типа Exception:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | try  {      object obj = "you";      int num = (int)obj;     // System.InvalidCastException      Console.WriteLine($"Результат: {num}");  }  catch (DivideByZeroException)  {      Console.WriteLine("Возникло исключение DivideByZeroException");  }  catch (IndexOutOfRangeException)  {      Console.WriteLine("Возникло исключение IndexOutOfRangeException");  }  catch (Exception ex)  {      Console.WriteLine($"Исключение: {ex.Message}");  } |

И в данном случае блок catch (Exception ex){} будет обрабатывать все исключения кроме DivideByZeroException и IndexOutOfRangeException. При этом блоки catch для более общих, более базовых исключений следует помещать в конце - после блоков catch для более конкретный, специализированных типов. Так как CLR выбирает для обработки исключения первый блок catch, который соответствует типу сгенерированного исключения. Поэтому в данном случае сначала обрабатывается исключение DivideByZeroException и IndexOutOfRangeException, и только потом Exception (так как DivideByZeroException и IndexOutOfRangeException наследуется от класса Exception).

Обычно система сама генерирует исключения при определенных ситуациях, например, при делении числа на ноль. Но язык C# также позволяет генерировать исключения вручную с помощью оператора **throw**. То есть с помощью этого оператора мы сами можем создать исключение и вызвать его в процессе выполнения.

Например, в нашей программе происходит ввод имени пользователя, и мы хотим, чтобы, если длина имени меньше 2 символов, то возникало исключение:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | try  {      Console.Write("Введите имя: ");      string? name = Console.ReadLine();      if (name== null || name.Length < 2)      {          throw new Exception("Длина имени меньше 2 символов");      }      else      {          Console.WriteLine($"Ваше имя: {name}");      }  }  catch (Exception e)  {      Console.WriteLine($"Ошибка: {e.Message}");  } |

После оператора **throw** указывается объект исключения, через конструктор которого мы можем передать сообщение об ошибке. Естественно вместо типа Exception мы можем использовать объект любого другого типа исключений.

Затем в блоке catch сгенерированное нами исключение будет обработано.

Подобным образом мы можем 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:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | try  {      try      {          Console.Write("Введите имя: ");          string? name = Console.ReadLine();          if (name == null || name.Length < 2)          {              throw new Exception("Длина имени меньше 2 символов");          }          else          {              Console.WriteLine($"Ваше имя: {name}");          }      }      catch (Exception e)      {          Console.WriteLine($"Ошибка: {e.Message}");          throw;      }  }  catch (Exception ex)  {      Console.WriteLine(ex.Message);  } |

В данном случае при вводе имени с длиной меньше 2 символов возникнет исключение, которое будет обработано внутренним блоком catch. Однако поскольку в этом блоке используется оператор throw, то исключение будет передано дальше внешнему блоку catch, который получит то же самое исключение и выведет то же самое сообщение на консоль.