Перечисления

Перечисление представляет собой множество именованных целочисленных констант. Перечислимый тип данных объявляется с помощью ключевого слова `enum`. Ниже приведена общая форма объявления перечисления:

```
enum имя { список_перечисления} ;
```

где *имя* – это имя типа перечисления, а *список_перечисления* – список идентификаторов, разделяемый запятыми.

В приведенном ниже примере объявляется перечисление `Apple` различных сортов яблок.

```
enum Apple { Jonathan,  GoldenDel,  RedDel, Winesap,
             Cortland,  Mcintosh };
```

Следует особо подчеркнуть, что каждая символически обозначаемая константа в перечислении имеет целое значение. Тем не менее неявные преобразования перечислимого типа во встроенные целочисленные типы и обратно в C# не определены, а значит, в подобных случаях требуется явное приведение типов. Кроме того, приведение типов требуется при преобразовании двух перечислимых типов. Но поскольку перечисления обозначают целые значения, то их можно, например, использовать для управления оператором выбора `switch` или же оператором цикла `for`.

Для каждой последующей символически обозначаемой константы в перечислении задается целое значение, которое на единицу больше, чем у предыдущей константы. По умолчанию значение первой символически обозначаемой константы в перечислении равно нулю. Следовательно, в приведенном выше примере перечисления `Apple` константа `Jonathan` равна нулю, константа `GoldenDel` – 1, константа `RedDel` – 2 и т.д.

Доступ к членам перечисления осуществляется по имени их типа, после которого следует оператор-точка. Например, при выполнении фрагмента кода

```
Console.WriteLine(Apple.RedDel + " имеет значение " +
                  (int)Apple.RedDel) ;
```

выводится следующий результат.

```
RedDel имеет значение 2
```

Как показывает результат выполнения приведенного выше фрагмента кода, для вывода перечислимого значения используется его имя. Но для получения этого значения требуется предварительно привести его к типу `int`.

Ниже приведен пример программы, демонстрирующий применение перечисления `Apple`.

Листинг 1

```
// Продемонстрировать применение перечисления.

using System;

class EnumDemo {
  enum Apple { Jonathan, GoldenDel, RedDel, Winesap,
               Cortland, McIntosh };

  static void Main() {
    string[] color = {
      "красный",
      "желтый",
      "красный",
      "красный ",
      "красный ",
      "красновато-зеленый"
```

```
    };

    Apple i; // объявить переменную перечислимого типа

    // Использовать переменную i для циклического
    // обращения к членам перечисления.
    for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
      Console.WriteLine(i + " имеет значение " + (int)i);

    Console.WriteLine();

    // Использовать перечисление для индексирования массива.
    for(i = Apple.Jonathan; i <= Apple.McIntosh; i++)
      Console.WriteLine("Цвет сорта " + i + " is " +
                        color[(int)i]);
  }
}
```

Ниже приведен результат выполнения этой программы.

```
Jonathan имеет значение 0
GoldenDel имеет значение 1
RedDel имеет значение 2
Winsap имеет- значение 3
Cortland имеет значение 4
Mcintosh имеет значение 5

Цвет сорта Jonathan - красный
Цвет сорта GoldenDel - желтый
Цвет сорта RedDel - красный
Цвет сорта Winsap - красный
Цвет сорта Cortland - красный
Цвет сорта Mcintosh - красновато-зеленый
```

Обратите внимание на то, как переменная типа `Apple` управляет циклами `for`. Значения символически обозначаемых констант в перечислении `Apple` начинаются с нуля, поэтому их можно использовать для индексирования массива, чтобы получить цвет каждого сорта яблок. Обратите также внимание на необходимость производить приведение типов, когда перечислимое значение используется для индексирования массива. Как упоминалось выше, в C# не предусмотрены неявные преобразования перечислимых типов в целочисленные и обратно, поэтому для этой цели требуется явное приведение типов.

И еще одно замечание: все перечисления неявно наследуют от класса `System.Enum`, который наследует от класса `System.ValueType`, а тот, в свою очередь, – от класса `object`.

1 Инициализация перечисления

Значение одной или нескольких символически обозначаемых констант в перечислении можно задать с помощью инициализатора. Для этого достаточно указать после символического обозначения отдельной константы знак равенства и целое значение. Каждой последующей константе присваивается значение, которое на единицу больше значения предыдущей инициализированной константы. Например, в приведенном ниже фрагменте кода константе `RedDel` присваивается значение 10.

```
enum Apple { Jonathan,  GoldenDel,  RedDel = 10, Winesap,
             Cortland,  Mcintosh };
```

В итоге все константы в перечислении принимают приведенные ниже значения.

```
Jonathan          0
GoldenDel         1
RedDel            10
```

```
Winesap          11
Cortland         12
Mcintosh         13
```

2 Указание базового типа перечисления

По умолчанию в качестве базового для перечислений выбирается тип `int`, тем не менее перечисление может быть создано любого целочисленного типа, за исключением `char`. Для того чтобы указать другой тип, кроме `int`, достаточно поместить этот тип после имени перечисления, отделив его двоеточием. В качестве примера ниже задается тип `byte` для перечисления Apple.

```
enum Apple  : byte { Jonathan,  GoldenDel, RedDel,
                     Winesap,  Cortland,  Mcintosh };
```

Теперь константа `Apple.Winesap`, например, имеет количественное значение типа `byte`.

3 Применение перечислений

На первый взгляд перечисления могут показаться любопытным, но не очень нужным элементом C#, но на самом деле это не так. Перечисления очень полезны, когда в программе требуется одна или несколько специальных символически обозначаемых констант. Допустим, что требуется написать программу для управления лентой конвейера на фабрике. Для этой цели можно создать метод `Conveyor()`, принимающий в качестве параметров следующие команды: "старт", "стоп", "вперед" и "назад". Вместо того чтобы передавать методу `Conveyor()` целые значения, например, 1 – в качестве команды "старт", 2 – в качестве команды "стоп" и так далее, что чревато ошибками, можно создать перечисление, чтобы присвоить этим значениям содержательные символические обозначения. Ниже приведен пример применения такого подхода.

Листинг 2

```
// Сымитировать управление лентой конвейера

using System;

class ConveyorControl
{
  // Перечислить команды конвейера.
  public enum Action { Start, Stop, Forward, Reverse };

  public void Conveyor(Action com) {
    switch(com) {
      case Action.Start:
        Console.WriteLine("Запустить конвейер.");
        break;
      case Action.Stop:
        Console.WriteLine("Остановить конвейер.");
        break;
      case Action.Forward:
        Console.WriteLine("Переместить конвейер вперед.");
        break;
      case Action.Reverse:
        Console.WriteLine("Переместить конвейер назад.");
        break;
    }
  }
}
```

```
class ConveyorDemo
{
  static void Main()
  {
    ConveyorControl c = new ConveyorControl();

    c.Conveyor(ConveyorControl.Action.Start);
    c.Conveyor(ConveyorControl.Action.Forward);
    c.Conveyor(ConveyorControl.Action.Reverse);
    c.Conveyor(ConveyorControl.Action.Stop);
  }
}
```

Вот к какому результату приводит выполнение этого кода.

```
Запустить конвейер.
Переместить конвейер вперед.
Переместить конвейер назад.
Остановить конвейер.
```

Метод `Conveyor()` принимает аргумент типа `Action`, и поэтому ему могут быть переданы только значения, определяемые в перечислении `Action`. Например, ниже приведена попытка передать методу `Conveyor()` значение 22.

```
c.Conveyor(22);  // Ошибка!
```

Эта строка кода не будет скомпилирована, поскольку отсутствует предварительно заданное преобразование типа `int` в перечислимый тип `Action`. Именно это и препятствует передаче неправильных команд методу `Conveyor()`. Конечно, такое преобразование можно организовать принудительно с помощью приведения типов, но это было бы преднамеренным, а не случайным или неумышленным действием. Кроме того, вероятность неумышленной передачи пользователем неправильных команд методу `Conveyor()` сводится к минимуму благодаря тому, что эти команды обозначены символическими именами в перечислении.

В приведенном выше примере обращает на себя внимание еще одно интересное обстоятельство: перечислимый тип используется для управления оператором `switch`. Как упоминалось выше, перечисления относятся к целочисленным типам данных, и поэтому их вполне допустимо использовать в операторе `switch`.