

Лабораторная работа

6. Классы и файлы

Варианты заданий

Выполнить задания 1-4 по предложенным вариантам:

	№ Задания		
	1	2	3
Вариант 1	1.II	2.II	3.III
Вариант 2	1.III	2.III	3.I
Вариант 3	1.I	2.I	3.II
Вариант 4	1.II	2.III	3.III
Вариант 5	1.I	2.II	3.II
Вариант 6	1.II	2.III	3.III
Вариант 7	1.II	2.I	3.II
Вариант 8	1.III	2.II	3.I
Вариант 9	1.I	2.II	3.III
Вариант 10	1.II	2.I	3.II

Задание 1.

I Составить описание класса для передачи пакетных данных по протоколу IPv4. Разработать методы установки следующих закрытых полей: размера передаваемого сообщения в байтах, размера одного пакета в байтах (минимум – 32 байт, максимум – 32 Кбайт) с проверкой допустимости вводимых значений (в том числе и при инициализации объекта конструктором). Случаи вне допустимых диапазонов необходимо обработать исключениями. Создать методы: изменения размера пакета на заданное количество байт, вычисления необходимого количества пакетов для передачи сообщения.

Примечание: пользователь может вводить данные с указанием единиц измерения: KB, MB, GB. Например: “25 MB”. Если единица измерения не указана, то считать, что введено значение в байтах. Реализовать метод преобразования всех этих величин в байты.

Разработать программу, демонстрирующую работу методов класса (необходимо создать несколько объектов). Все значения представлены целыми числами.

Например: Пользователь вводит размер сообщения: 34051, размер пакета: 2 KB. Число пакетов для передачи такого сообщения будет равно 17. Увеличиваем размер пакета на 1 KB. Тогда число пакетов для передачи будет равно 12.

II Составить описание класса для представления времени в 24-часовом формате. Разработать методы установки даты и отдельных ее полей (часы, минуты, секунды – закрытые поля) с проверкой допустимости вводимых значений (в том числе и при инициализации объекта конструктором). Случаи вне допустимых диапазонов необходимо обработать исключениями. Создать методы изменения даты на заданное количество часов, минут, секунд.

Разработать программу, демонстрирующую работу методов класса (необходимо создать несколько объектов).

III Составить описание класса для представления положения объекта в пространстве. Разработать методы установки положения объекта и отдельных его географических координат (широта, долгота, высота – закрытые поля) с проверкой допустимости вводимых значений (в том числе и при инициализации объекта конструктором):

- $-90^{\circ} \leq \phi \leq 90^{\circ}$ - широта: $\phi \geq 0$ – северная широта (N), $\phi < 0$ – южная широта (S);
- $-180^{\circ} \leq \lambda \leq 180^{\circ}$ - долгота: $\lambda \geq 0$ – восточная долгота (E), $\lambda < 0$ западная долгота (W).
- $-10000 \leq h \leq 10000$ – высота, м над уровнем моря.

Случаи вне допустимых диапазонов необходимо обработать исключениями. Создать методы изменения положения объекта в пространстве, методы преобразования градусов с десятичной частью в градусы, минуты и секунды:

$1^{\circ} = 60'$ минутам, $1' \text{ минута} = 60''$ секундам.

Разработать программу, демонстрирующую работу методов класса (необходимо создать несколько объектов).

Примеры координат:

- N55.755831, E37.617673 — градусы
- 55 45'20.9916"N, 37 37'3.6228"E — градусы, минуты и секунды

Задание 2.

I "Группы". Выполняя программу "Обмен студентами", университету необходимо создать n групп студентов. В каждой группе должны быть представлены студенты разных специальностей, по возможности равномерно. Факультет в соответствии с численностью студентов на каждой специальности выделил для поездки лучших своих студентов. Составлен общий список студентов, упорядоченный по их фамилиям. Необходимо распределить студентов по группам.

Указание: названия специальностей университета следует задать перечислением. Следует определить класс Student, среди полей которого будет поле spes, заданное перечислением.

Например, итоговый результат может быть таким для 2х групп и 3х специальностей:

Группа 1: Иванов (РПИС), Петров (ПО), Сидоров (УИТС);

Группа 2: Смирнов (РПИС), Кузнецов (ПО).

II "Книги". Библиотечному коллектору необходимо создать n наборов книг для рассылки в библиотеки. В каждом наборе должны быть представлены книги разной тематики, по возможности равномерно. В коллекторе составлен общий список книг, упорядоченный по фамилиям авторов. Необходимо распределить книги по наборам.

Указание: тематику книг следует задать перечислением. Следует определить класс Book, среди полей которого будет поле title, заданное перечислением.

Пример реализации – аналогично заданию 2.I.

III "Тесты". Преподавателю необходимо создать n наборов тестов. В каждом наборе должны быть представлены вопросы разной тематики, по возможности равномерно. У преподавателя составлен общий список вопросов, упорядоченный по их названиям. Помогите преподавателю распределить вопросы по тестам.

Указание: тематику вопросов следует задать перечислением. Следует определить класс Question, среди полей которого будет поле theme, заданное перечислением.

Пример реализации – аналогично заданию 2.I.

Задание 3.

Дан текстовый файл (in.txt) содержащий список учащихся и их оценки по трем предметам: математике, физике, информатике.

Формат файла: сначала количество учащихся n , затем n строк, каждая из которых содержит фамилию, имя и три числа. Данные в строке разделены одним пробелом.

Оценки принимают значение от 1 до 5.

Пример входного файла:

```
4
Ivanov Vasilii 4 3 4
Petrov Sergey 5 3 5
Konstantinov Nikolay 4 4 5
Kuznetsov Ivan 5 5 5
```

I Определите средний балл каждого из учащихся. Выведите в файл out.txt фамилии и имена учащихся, не имеющих троек (а также двоек и колов).

II Выведите три действительных числа: средний балл всех учащихся по математике, по физике, по информатике. Определите учащихся с наилучшей успеваемостью, то есть с максимальным средним баллом по трем предметам. Выведите в файл out.txt одного или нескольких учащихся, имеющих максимальный средний балл.

III Выведите в файл out.txt фамилии и имена учащихся в порядке убывания их среднего балла.

Методические указания

Оглавление

6. Классы и файлы	1
Варианты заданий.....	1
Методические указания.....	4
6.1. Классы	4
6.1.1. Поля, методы и конструкторы	5
6.1.2. Обработка исключений (Exception)	7
6.2. Классы-перечисления	10
6.3. Работа с файлами и строками	14
6.3.1. Статические и динамические методы и свойства класса String.	14
6.3.2. Форматирование строк.	16
6.1.1. Ввод-вывод в файлы.	18

6.1. Классы

У класса две различные роли: модуля и типа данных.

- Класс - это **модуль**, архитектурная единица построения программной системы. Модульность построения - основное свойство программных систем.
- Класс - это **тип данных**, задающий реализацию некоторой абстракции данных, характерной для проблемной области, в интересах которой создается программная система.

Синтаксис описания класса:

```
[атрибуты][модификаторы]class имя_класса[:список_родителей]
{тело_класса}
```

В простых случаях объявление класса может выглядеть так:

```
public class Class1 {тело_класса}
```

В теле класса могут быть объявлены:

- константы;
- поля;
- конструкторы и деструкторы;
- методы;
- события;
- классы (структуры, делегаты, интерфейсы, перечисления).

Из синтаксиса следует, что классы могут быть вложенными. Такая ситуация довольно редкая. Ее стоит использовать, когда некоторый класс носит вспомогательный характер, разрабатывается в интересах другого класса, и есть полная уверенность, что внутренний

класс никому не понадобится, кроме класса, в который он вложен и, возможно, его потомков. Внутренние классы обычно имеют модификатор доступа `private` или `protected`.

6.1.1. Поля, методы и конструкторы

Поля класса синтаксически являются обычными переменными (объектами) языка. Их описание удовлетворяет обычным правилам объявления переменных. Содержательно поля задают представление абстракции данных, которую реализует класс. Поля характеризуют свойства объектов класса. Если проектируется класс `Car`, то поля задают свойства автомобилей, для класса `Person` поля задают свойства личности.

Каждое поле имеет **модификатор доступа**, принимающий одно из четырех значений: `public`, `private`, `protected`, `internal`. Возможно совместное задание двух атрибутов `protected` и `internal`.

- **Модификатор `public`** обеспечивает доступ к полю класса из любого созданного в коде программы объекта.
- **Модификатор `private`** является атрибутом доступа по умолчанию. Он закрывает поля от всех других классов, разрешая прямой доступ к ним (чтение и запись) только методам самого класса. Помните, все поля всегда доступны всем методам класса. Они являются для методов класса глобальной информацией, с которой работают все методы, извлекая из полей нужные им данные и изменяя их значения в ходе работы.
- **Модификатор `protected`** открывает поля классам наследникам. Если класс `A` объявил некоторое поле с модификатором `protected`, то методы класса `B`, который является наследником класса `A` и, следовательно, наследует поля класса `A`, могут непосредственно работать с наследуемыми полями.
- **Модификатор `internal`** открывает поля дружественным классам. Два класса `A` и `B` называются дружественными, если они принадлежат одной сборке - одному проекту. Если класс `A` объявил некоторое поле с модификатором `internal`, то методы дружественного класса `B`, являющегося клиентом класса `A`, могут непосредственно работать с таким полем.

Все процедуры и функции, объявленные в классе, являются **методами класса**. Их описание удовлетворяет обычным правилам синтаксиса. Методы содержат описания операций, доступных над объектами класса, определяя, тем самым, поведение объектов. Все объекты одного класса имеют один и тот же набор методов

Каждый метод имеет **модификатор доступа**, принимающий одно из четырех значений: `public`, `private`, `protected`, `internal`. Модификатором доступа по умолчанию является модификатор `private`. Независимо от значения модификатора доступа, все методы доступны для вызова при выполнении метода класса. Если методы имеют модификатор доступа `private`, возможно, опущенный, то тогда они доступны для вызова только внутри методов самого класса. Такие методы считаются **закрытыми**.

Конструктор - неотъемлемый компонент класса. Нет классов, задающих тип данных и не имеющих конструкторов. Конструктор представляет собой специальный метод класса, позволяющий создавать объекты класса. У конструкторов две синтаксические особенности:

- имя конструктора фиксировано и совпадает с именем класса,
- для конструктора не задается возвращаемое значение.

Из первого свойства следует, что конструкторы класса представляют собой множество перегруженных методов и должны отличаться сигнатурой - числом аргументов либо типами аргументов. Чтобы справиться с этим ограничением, иногда приходится в один из конструкторов добавлять фиктивный аргумент, изменяя тем самым его сигнатуру.

Если в классе явно не задан ни один конструктор, то к классу автоматически добавляется конструктор по умолчанию - конструктор без аргументов. Заметьте, что если в классе явно определен один или несколько конструкторов, то автоматического добавления конструктора без аргументов не происходит.

Рассмотрим следующий пример. Определяем класс Worker. Внутри этого класса существуют две переменные — целая age для возраста и name строкового типа для имени:

```
using System;
namespace test
{
    //Начало класса
    class Worker
    {
        int age;
        string name;

        Worker()//конструктор
        {
            age = 0;
        }

        Worker(int age, string name)//конструктор с параметрами
        {
            this.age = age; //присваиваем полю класса this.age значение параметра
            this.name = name;
        }

        public void setAge(int age) //метод задания возраста
        {
            this.age = age;
        }

        public void setName(string name) //метод задания имени
        {
            this.name = name;
        }

        public int getAge() //метод возврата возраста
        {
            return age;
        }

        public string getName() //метод возврата имени
        {
            return name;
        }
    }//Конец класса

    class Test
```

```
{
    static void Main(string[] args)
    {
        //Создаем новый экземпляр класса (объект)
        Worker wrk1 = new Worker();
        wrk1.setAge(30);
        wrk1.setName("Петров Иван Сергеевич");
        Console.WriteLine(wrk1.getName() + " - " + wrk1.getAge() + " года");
    }
}
```

Поля класса обычно имеют модификатор доступа `private`, а доступ к ним осуществляется через публичные методы класса.

Класс может быть описан в коде файла `Program.cs`. Обычно для описания класса создается отдельный файл. Чтобы добавить новый класс в проект необходимо в окне `Solution Explorer` кликнуть правой кнопкой на проекте, выбрать `Add->Class`. После чего необходимо указать имя класса.

6.1.2. Обработка исключений (Exception)

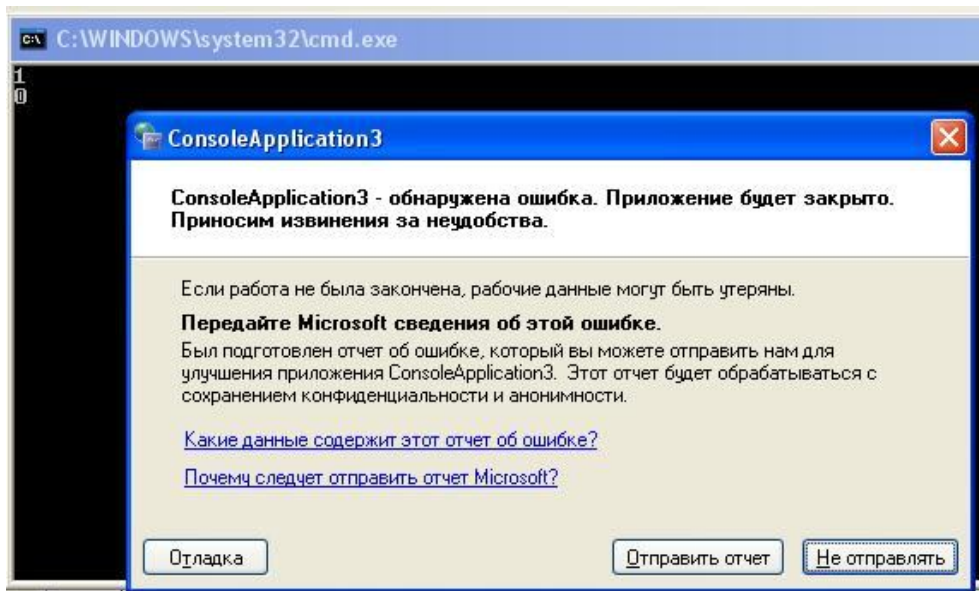
Как правило, программист в своем проекте не может предсказать все действия пользователя, входные значения и многие другие параметры. А чем крупнее и серьезнее проект, тем таких мест в приложении становится все больше и больше, поэтому перед программистом встает задача отловить и по возможности обработать все возможные ситуации неправильного выполнения код.

Любое действие, которое не может быть выполнено по той или иной причине, называется `Exception` (исключение). Данное исключение и будем отлавливать. Для этого в `C#` есть конструкция `try .. catch`.

Рассмотрим небольшой пример

```
int a = Convert.ToInt32(Console.ReadLine());
int b = Convert.ToInt32(Console.ReadLine());
Console.WriteLine(a / b);
```

В данном примере вводятся два числа, далее идет деление и вывод результата. Все бы ничего, а что если пользователь введет во втором случае 0? Ведь на ноль делить нельзя - будет выведено исключение



Возможно, пользователь даже не сможет понять, что произошло. Отловим исключение, выдав ошибку.

Заклучим критическую область в try, а в catch выведем ошибку

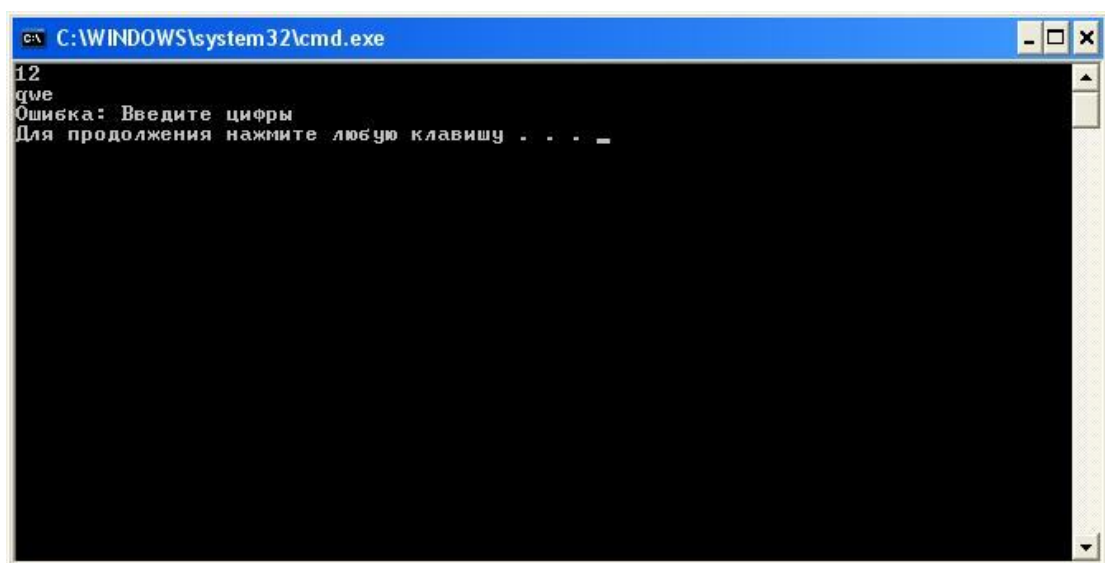
```
try
{
    int a = Convert.ToInt32(Console.ReadLine());
    int b = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine(a / b);
}
catch
{
    Console.WriteLine("Ошибка: деление на ноль");
}
```



Теперь, как видите, ошибка очевидна. Но часто так бывает, что в критической секции могут возникнуть разные исключения и их надо по-разному обработать, тогда нам надо как можно точнее описать секцию catch, то есть для какого именно исключения она будет обрабатывать. Для этого рядом с Catch пишем имя исключения. Пользователь может ввести не только цифры, но и буквы, поэтому обработаем и это исключение. Оно будет происходить в момент конвертации string в int


```
try
{
    int a = Convert.ToInt32(Console.ReadLine());
    int b = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine(a / b);
}
catch (System.DivideByZeroException)
{
    Console.WriteLine("Ошибка: деление на ноль");
}
catch (System.FormatException)
{
    Console.WriteLine("Ошибка: Введите цифры");
}
```

Теперь будут все исключения обрабатываться как надо.



Осталось только рассмотреть последний блок данной конструкции – блок `finally`. Он выполняется всегда: произошла ошибка в `catch` или нет.

Пример может быть следующим: вы открываете файл что то делаете и в любом случае произошла или нет ошибка вы должны закрыть файл это можно как раз сделать в `finally`

```
try
{
    Открываем файл
}
catch
{
    Console.WriteLine("Ошибка: что то с файлом");
}
finally
{
    Закрываем файл
}
```

Исключения обнаруживаются и обрабатываются в операторе `try`.

Исключения генерирует либо среда выполнения, либо программист с помощью оператора `throw`. В таблице 1 приведены наиболее часто используемые стандартные

исключения, генерируемые средой. Они определены в пространстве имен System. Все они являются потомками класса Exception, а точнее, потомками его потомка SystemException.

Таблица 1. Часто используемые стандартные исключения

Имя	Описание
DivideByZeroException	Попытка деления на ноль
FormatException	Попытка передать в метод аргумент неверного формата
IndexOutOfRangeException	Индекс массива выходит за границы диапазона
InvalidCastException	Ошибка преобразования типа
OutOfMemoryException	Недостаточно памяти для создания нового объекта
OverflowException	Переполнение при выполнении арифметических операций
StackOverflowException	Переполнение стека

Оператор throw

До сих пор мы рассматривали исключения, которые генерирует среда выполнения C#, но это может сделать и сам программист. Для генерации исключения используется оператор throw с параметром, определяющим вид исключения. Параметр должен быть объектом, порожденным от стандартного класса System.Exception. Этот объект используется для передачи информации об исключении его обработчику.

Синтаксис оператора throw:

```
throw [ выражение ];
```

Форма без параметра применяется только внутри блока catch для повторной генерации исключения. Тип выражения, стоящего после throw, определяет тип исключения, например:

```
throw new DivideByZeroException();
```

Здесь после слова throw записано выражение, создающее объект стандартного класса "ошибка при делении на 0" с помощью операции new.

При генерации исключения выполнение текущего блока прекращается и происходит поиск соответствующего обработчика с передачей ему управления. Обработчик считается найденным, если тип объекта, указанного после throw, либо тот же, что задан в параметре catch, либо является производным от него.

6.2. Классы-перечисления

Перечисление – это частный случай класса, класс, заданный без собственных методов. Перечисление задает конечное множество возможных значений, которые могут получать объекты класса перечисления. Поскольку у перечислений нет собственных методов, то синтаксис объявления этого класса упрощается, остается обычный заголовок и тело класса, содержащее список возможных значений. Вот формальное определение синтаксиса перечислений:

```
[атрибуты][модификаторы]enum имя_перечисления[:базовый класс]  
{список_возможных_значений}
```

Приведем примеры объявлений классов-перечислений:

```
public enum Profession {teacher, engineer, businessman};
public enum MyColors {red, blue, yellow, black, white};
public enum TwoColors {black, white};
public enum Rainbow {красный, оранжевый, желтый, зеленый, голубой, синий,
фиолетовый};
public enum Sex: byte {man=1, woman};
public enum Days:long {Sun, Mon, Tue, Wed, Thu, Fri, Sat};
```

Рассмотрим теперь пример работы с объектами – экземплярами различных перечислений. С этой целью введем в программный проект класс Testing - он является клиентом выше определенных перечислений, методы которого реализуют тесты, позволяющие анализировать работу с перечислениями.

```
class Testing
{
    /// <summary>
    /// Создание объектов перечислений,
    /// присваивание значений и вывод на печать
    /// </summary>
    public void TestEnum()
    {
        const string COLOR_EQUAL =
            "Цвета совпадают!";
        const string COLOR_DIFFERENT =
            "Цвета не совпадают!";
        const string ENUM_RAINBOW =
            "Цвета перечисления Rainbow:";

        Rainbow color = new Rainbow();
        //MyColors color1 = MyColors(MyColors.blue);
        MyColors color1 = MyColors.white;
        TwoColors color2;
        color2 = TwoColors.white;
        Console.WriteLine("цвет1 = {0}, цвет2 = {1}",
            color1, color2);

        //if(color1 != color2) color2 = color1;
        if (color1.ToString() == color2.ToString())
            Console.WriteLine(COLOR_EQUAL);
        else Console.WriteLine(COLOR_DIFFERENT);

        Rainbow color3;
        color3 = (Rainbow)4;
        color1 = MyColors.blue;
        Console.WriteLine("цвет1 = {0}, цвет2 = {1}",
            color1, color3);
        if (color3 == Rainbow.голубой)
            Console.WriteLine(COLOR_EQUAL);
        else Console.WriteLine(COLOR_DIFFERENT);

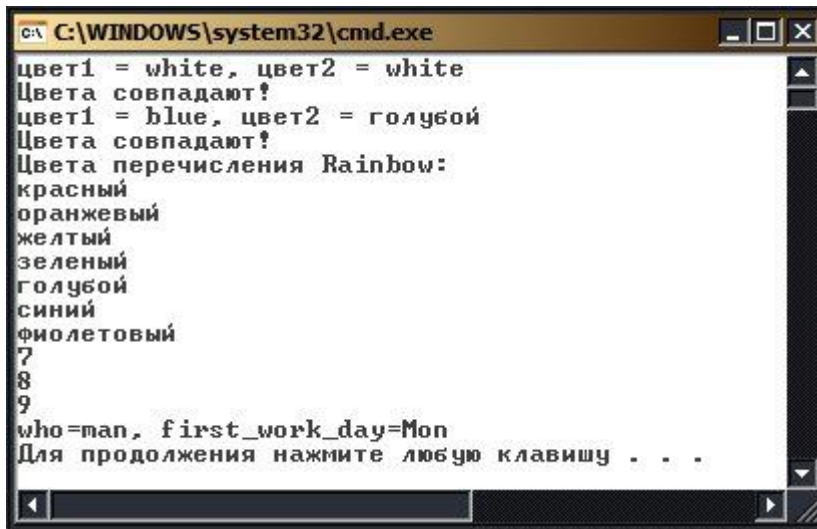
        Console.WriteLine(ENUM_RAINBOW);
        for (int num = 0; num < 10; num++)
        {
            color = (Rainbow)num;
            Console.WriteLine(color.ToString());
        }
    }
}
```

```
    }  
  
    Sex who = Sex.man;  
    Days first_work_day = (Days)(long)1;  
  
    Console.WriteLine("who={0}, first_work_day={1}",  
        who, first_work_day);  
    }  
}
```

Данный пример иллюстрирует следующие особенности работы с объектами перечислений.

- Объект перечисления можно создать в объектном стиле с использованием операции new. Но у перечислений есть только единственный конструктор без параметров, инициализирующий объект первым по порядку следования значением перечисления. В примере так создается объект color класса Rainbow, получающий значение "красный".
- Попытка создать объект color1 класса MyColors закомментирована, поскольку у перечислений нет конструкторов с параметрами.
- Объекты можно объявлять с явной инициализацией, как color1, или с отложенной инициализацией, как color2. При объявлении без явной инициализации объект получает значение первой константы перечисления, так что color2 в момент объявления получает значение black.
- Объекту можно присвоить значение, которое задается константой перечисления, уточненной именем перечисления, как для color1 и color2.
- Нельзя сравнивать объекты разных перечислений, например, color1 и color2. Это разные классы, для которых не определены операции преобразования типа. Но, заметьте, можно сравнивать строки, возвращаемые методом ToString, например, color1.ToString() и color2.ToString().
- Метод ToString, наследованный от класса object, для перечислений переопределен. Если для числового значения объекта существует константа перечисления, отображаемая на числовое значение, то в качестве результата возвращается соответствующий идентификатор как строка. В противном случае, когда такого отображения нет, возвращается как строка само числовое значение.
- Существуют явные взаимно обратные преобразования констант базового типа и констант перечисления. Цикл, печатающий все значения перечисления Rainbow, демонстрирует преобразование объектов типа int в объекты Rainbow. Заметьте, в объекты Rainbow преобразуются числа, выходящие за интервал, на который проецируются константы перечисления.

Результаты работы метода TestEnum:



```

C:\WINDOWS\system32\cmd.exe
цвет1 = white, цвет2 = white
Цвета совпадают!
цвет1 = blue, цвет2 = голубой
Цвета совпадают!
Цвета перечисления Rainbow:
красный
оранжевый
желтый
зеленый
голубой
синий
фиолетовый
7
8
9
who=man, first_work_day=Mon
Для продолжения нажмите любую клавишу . . .

```

Персоны и профессии

Рассмотрим класс Person с полями, типичными для классов, которые описывают личность: имя, фамилия, возраст и так далее. Добавим в этот класс поле, определяющее профессию персоны. Вполне разумно иметь перечисление, например, Profession, задающее список возможных профессий:

```

string name;
public enum Profession { businessman, teacher, engineer };
public Profession prof;
Person(string s)
{
    name = s;
}

```

Добавим еще в класс Person метод Analysis, который анализирует профессию, организовав традиционный разбор случаев и принимая решение на каждой ветви, в данном примере – выводя соответствующий текст:

```

public void Analysis()
{
    switch (prof)
    {
        case Profession.businessman:
            Console.WriteLine(name + ". профессия: бизнесмен");
            break;
        case Profession.teacher:
            Console.WriteLine(name + ". профессия: учитель");
            break;
        case Profession.engineer:
            Console.WriteLine(name + ". профессия: инженер");
            break;
        default:
            Console.WriteLine(name + ". профессия: неизвестна");
            break;
    }
}

```

Приведем простой тестирующий пример работы с объектом Person и его профессией:

```

public static void TestProfession()
{

```

```

    Person[] persons = new Person[3];
    persons[0] = new Person("Петров");
    persons[0].prof=Profession.businessman;
    persons[1] = new Person("Фролов");
    persons[1].prof = Profession.engineer;
    persons[2] = new Person("Климов");
    persons[1].prof = Profession.teacher;

    foreach (Person p in persons)
    {
        p.Analysis();
    }
}

```

6.3. Работа с файлами и строками

6.3.1. Статические и динамические методы и свойства класса String.

Статические методы и свойства класса string	
Метод	Описание
Empty	Возвращается пустая строка. Свойство со статусом read only.
Compare	Сравнение двух строк. Метод перегружен. Реализации метода позволяют сравнивать как строки, так и подстроки. При этом можно учитывать или не учитывать регистр, особенности национального форматирования дат, чисел и т.д.
CompareOrdinal	Сравнение двух строк. Метод перегружен. Реализации метода позволяют сравнивать как строки, так и подстроки. Сравниваются коды символов.
Concat	Конкатенация строк. Метод перегружен, допускает сцепление произвольного числа строк.
Copy	Создается копия строки.
Format	Выполняет форматирование в соответствии с заданными спецификациями формата.
Join	Конкатенация массива строк в единую строку. При конкатенации между элементами массива вставляются разделители. Операция, заданная методом Join, является обратной к операции, заданной методом Split. Последний является динамическим методом и, используя разделители, осуществляет разделение строки на элементы.

Методы Join и Split выполняют над строкой текста взаимно обратные преобразования. Динамический метод Split позволяет осуществить разбор текста на элементы. Статический метод Join выполняет обратную операцию, собирая строку из элементов.

Наиболее часто используемая реализация имеет следующий синтаксис:

```
public string[] Split(params char[])
```

На вход методу Split передается один или несколько символов, интерпретируемых как разделители. Объект string, вызвавший метод, разделяется на подстроки, ограниченные этими разделителями. Из этих подстрок создается массив, возвращаемый в качестве результата метода.

Синтаксис статического метода Join таков:

```
public static string Join(string delimiters, string[] items )
```

В качестве результата метод возвращает строку, полученную конкатенацией элементов массива `items`, между которыми вставляется строка разделителей `delimiters`. Как правило, строка `delimiters` состоит из одного символа, который и разделяет в результирующей строке элементы массива `items`; но в отдельных случаях ограничителем может быть строка из нескольких символов, например, запятая и следующий за ней пробел.

Рассмотрим примеры применения этих методов. В первом из них строка представляет сложноподчиненное предложение, которое разбивается на простые предложения. Во втором предложение разделяется на слова. Затем производится обратная сборка разобранного текста. Вот код соответствующей процедуры:

```
public void TestSplitAndJoin()
{
    string txt = "А это пшеница, которая в темном чулане хранится,"
    + " в доме, который построил Джек!";
    Console.WriteLine("txt={0}", txt);
    Console.WriteLine("Разделение текста на простые предложения:");
    string[] SimpleSentences, Words;
    //размерность массивов SimpleSentences и Words
    //устанавливается автоматически в соответствии с
    //размерностью массива, возвращаемого методом Split
    SimpleSentences = txt.Split(',');
    for(int i=0;i< SimpleSentences.Length; i++)
        Console.WriteLine("SimpleSentences[{0}]= {1}", i, SimpleSentences[i]);
    string txtjoin = string.Join(",", SimpleSentences);
    Console.WriteLine("txtjoin={0}", txtjoin);
    Words = txt.Split(' ', ' ');
    for(int i=0;i< Words.Length; i++)
        Console.WriteLine("Words[{0}]= {1}", i, Words[i]);
    txtjoin = string.Join(" ", Words);
    Console.WriteLine("txtjoin={0}", txtjoin);
}
```

Результаты выполнения этой процедуры показаны на рис. 1.


```

E:\from_D\C#BookProjects\Strings\bin\Debug\Strings.exe
txt=Я это пшеница, которая в темном чулане хранится, в доме, который построил Джек!
Разделение текста на простые предложения:
SimpleSentences[0]= Я это пшеница
SimpleSentences[1]= которая в темном чулане хранится
SimpleSentences[2]= в доме
SimpleSentences[3]= который построил Джек!
txtjoin=Я это пшеница, которая в темном чулане хранится, в доме, который построил Джек!
Words[0]= Я
Words[1]= это
Words[2]= пшеница
Words[3]= которая
Words[4]= в
Words[5]= темном
Words[6]= чулане
Words[7]= хранится
Words[8]= в
Words[9]= доме
Words[10]= который
Words[11]= построил
Words[12]= Джек!
txtjoin=Я это пшеница которая в темном чулане хранится в доме который построил Джек!
Press any key to continue

```

Рис. 1. Разбор и сборка строки текста

Рассмотрим наиболее характерные динамические методы при работе со строками

Динамические методы и свойства класса string

Метод	Описание
Insert	Вставляет подстроку в заданную позицию.
Remove	Удаляет подстроку в заданной позиции.
Replace	Заменяет подстроку в заданной позиции на новую подстроку.
Substring	Выделяет подстроку в заданной позиции.
IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny	Определяются индексы первого и последнего вхождения заданной подстроки или любого символа из заданного набора
StartsWith, EndsWith	Возвращается true или false, в зависимости от того, начинается или заканчивается строка заданной подстрокой.
PadLeft, PadRight	Выполняет набивку нужным числом пробелов в начале и в конце строки.
Trim, TrimStart, TrimEnd	Обратные операции к методам Pad. Удаляются пробелы в начале и в конце строки, или только с одного ее конца.
ToCharArray	Преобразование строки в массив символов.

6.3.2. Форматирование строк.

Спецификатор формата определяет форму представления выводимого значения. В общем виде параметр задается следующим образом:

```
{n [,m[:спецификатор_формата]]}
```

Здесь n — номер параметра. Параметры нумеруются с нуля, нулевой параметр заменяется значением первой переменной из списка вывода, первый параметр — второй переменной, и т. д. Параметр m определяет минимальную ширину поля, которое отводится под выводимое значение. Если выводимому числу достаточно меньшего

количества позиций, неиспользуемые позиции заполняются пробелами. Если числу требуется больше позиций, параметр игнорируется.

Спецификатор формата, как явствует из его названия, определяет формат вывода значения. Например, спецификатор C (Currency) означает, что параметр должен форматироваться как валюта с учетом национальных особенностей представления, а спецификатор X (Hexadecimal) задает шестнадцатеричную форму представления выводимого значения.

Пользовательские шаблоны задают вид выводимого значения посимвольно, причем на месте каждого символа может стоять либо #, либо 0. Если указан знак #, на этом месте будет выведена цифра числа, если она не равна нулю. Если указан 0, будет выведена любая цифра, в том числе и 0.

Консольный вывод всегда можно свести к форме Console.WriteLine(s), если строку s предварительно отформатировать, используя явный вызов метода Format. Этот метод полезно вызывать и в Windows-проектах при выводе специфических данных - денежных сумм, процентов, дат и времени.

```
enum Rainbow {красный, желтый, голубой};  
/// <summary>  
/// Форматирование чисел, дат, перечислений  
/// </summary>  
public void TestFormat()  
{  
    int x = 77;  
    double p = 0.52;  
    double d = -151.17;  
    DateTime today = DateTime.Now;  
    //Форматирование чисел  
    string s = string.Format("Итого:{0:P}\n" +  
        "Сумма_1 = {1:C}\n" +  
        "x = {1:#####} рублей\n" +  
        "d = {2,-10:F} рублей\n" +  
        "d = {2, 10:F} рублей\n" +  
        "d = {2:E}\n", p, x, d);  
    Console.WriteLine(s);  
    //Форматирование дат  
    s = string.Format("Время: {0:t}, Дата: {0:d}\n" +  
        "Дата и время - {0:F}", today);  
    Console.WriteLine(s);  
    //Форматирование перечислений  
    s = string.Format("Цвет1: {0:G}, Цвет2: {1:F}\n",  
        Rainbow.голубой, Rainbow.красный);  
    Console.WriteLine(s);  
    //Национальные особенности  
    System.Globalization.CultureInfo ci =  
        new System.Globalization.CultureInfo("en-US");  
    s = string.Format(ci, "Итого:{0,4:C} ", 77.77);  
    Console.WriteLine(s);  
}
```

В примере показано использование различных спецификаций формата с разными кодами форматирования для таких данных. В заключительном фрагменте кода

демонстрируется задание провайдером национальных особенностей. С этой целью создается объект класса `CultureInfo`, инициализированный так, чтобы он задавал особенности форматирования, принятые в США. Класс `CultureInfo` наследует интерфейс `IFormatProvider`. Российские национальные особенности форматирования установлены по умолчанию. При необходимости их можно установить таким же образом, как это сделано для США, задав соответственно константу `"ru-RU"`. Результаты работы метода показаны на рис.2.

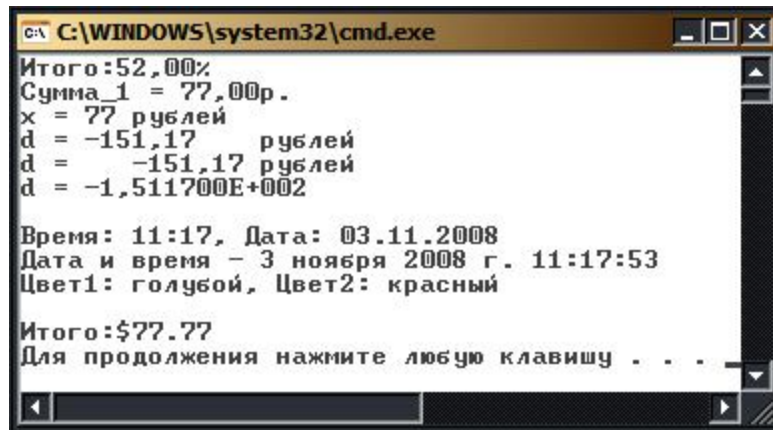


Рис. 2. Результаты работы метода `Format`

6.1.1. Ввод-вывод в файлы.

Часто бывает удобно заранее подготовить исходные данные в текстовом файле и считывать их в программе. Вывод из программы тоже бывает полезно выполнить не на экран, а в текстовый файл. Работа с файлами подробно рассматривается позже, а сейчас приведем лишь образцы для использования в программах. В примере рассмотрена программа, выводящая данные в текстовый файл с именем `output.txt`. Файл создается в том же каталоге, что и исполняемый файл программы, по умолчанию — `...\ConsoleApplication1\bin\Debug`.

```

using System;
using System.IO;                                     // 1
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            StreamWriter f = new StreamWriter( "output.txt" ); // 2
            int i = 3;
            double y = 4.12;
            decimal d = 600m;
            string s = "Вася";

            f.WriteLine( "i = " + i );                // 3
            f.WriteLine( "s = " + s );                // 4
            f.WriteLine( "y = {0} \nd = {1}", y, d ); // 5

            f.Close();                                // 6
        }
    }
}
  
```

Для того чтобы использовать в программе файлы, необходимо:

1. Подключить пространство имен, в котором описываются стандартные классы для работы с файлами (оператор 1).
2. Объявить файловую переменную и связать ее с файлом на диске (оператор 2).
3. Выполнить операции ввода-вывода (операторы 3–5).
4. Закрыть файл (оператор 6).

Ввод данных из файла выполняется аналогично. В следующем примере приведена программа, в которой ввод выполняется из файла с именем input.txt, расположенного в каталоге D:\C#\.

```
using System;
using System.IO;
namespace ConsoleApplication1
{
    class Class1
    {
        static void Main()
        {
            StreamReader f = new StreamReader( "d:\\C#\\input.txt" );
            string s = f.ReadLine(); //1. ввести строку
            Console.WriteLine( "s = " + s );

            char c = (char)f.Read(); //2 ввести символ
            f.ReadLine();
            Console.WriteLine( "c = " + c );

            string buf;
            buf = f.ReadLine();           //3 ввести целое число
            int i = Convert.ToInt32( buf );
            Console.WriteLine( i );

            buf = f.ReadLine();
            double x = Convert.ToDouble( buf );//4 вещественное число
            Console.WriteLine( x );

            buf = f.ReadLine();
            double y = double.Parse( buf ); //5 вещественное число
            Console.WriteLine( y );

            buf = f.ReadLine();
            decimal z = decimal.Parse( buf ); //6 вещественное число
            Console.WriteLine( z );
            f.Close();
        }
    }
}
```

Источники:

1. Биллиг В. Основы программирования на С#. Режим доступа: <http://www.intuit.ru/studies/courses/2247/18/info>
2. Павловская Т.А. Программирование на С#. Учебный курс. Режим доступа: <http://ips.ifmo.ru/courses/csharp/index.html>