ПРАКТИЧЕСКАЯ РАБОТА 1.2

Тема: Использование интерфейсов и параметризованных коллекций. Работа с классами файлового и потокового ввода/вывода при разработке приложений.

Цель: Научиться использовать классы файлового и потокового ввода/вывода при разработке приложений на языке C#.

Время выполнения: 6 часов.

Содержание работы и последовательность ее исполнения.

I. Входной контроль.

1) Дайте определение понятия «класс».
2) В чем преимущества использования объектно-ориентированного программирования по сравнению с императивным программированием?
3) Опишите основные классы пространства имен System.IO.

II. Задания для выполнения работы.

Задание №1. *Разработка классов согласно индивидуальному варианту задания.*

Разработать класс для хранения списка объектов классов, используя коллекцию. Класс должен содержать следующие методы:

- конструктор по умолчанию;
- помещение элемента в список;
- удаление текущего элемента из списка;
- перемещение по списку (в начало, в конец, на следующий элемент, на предыдущий элемент);
- получение текущего элемента из списка;
- установка нового значения выбранного элемента;
- загрузить объекты из коллекции в файл;
- сохранить объекты из коллекции в файл.

Реализовать глобальный класс MyApplication, реализующий главное меню для управления классами.

Задание №2. *Тестирование разработанных классов.*

Реализовать консольное приложение для тестирования разработанных классов, в котором пользователь может добавить, удалить или изменить значения в списке. Кроме того, пользователь должен иметь возможность выполнения сортировки списка.

Для ввода и вывода информации использовать классы файлового и потокового ввода/вывода библиотеки FCL.

III. Краткие теоретические сведения и примеры программного кода

MicrosoftVisualStudio *.NET* - это интегрированная *среда разработки* для создания, документирования, запуска и отладки программ, написанных

на языках *.NET*.

Эта *среда разработки* является открытой языковой средой. Наряду с языками программирования, изначально включенными в среду - *C++, C#, J#, Visual Basic,* - в нее могут добавляться любые языки программирования, компиляторы которых создаются сторонними разработчиками. Необходимым условием для включения языков в среду *Visual Studio.Net* является использование единого каркаса – платформы *Framework.Net*.

Платформа *Framework. Net* позволяет:

– легко использовать компоненты, разработанных на различных языках;

– разрабатывать единое приложение из нескольких частей на разных языках;

Платформа *Framework.Net* содержит две сновных компоненты:

– FCL (Framework Class Library) - библиотеку классов каркаса;

– CLR (Common Language Runtime) - общеязыковую исполнительную среду.

Платформа *.NET* предоставляет в распоряжение программиста библиотеку базовых классов, доступную из любого языка программирования .NET. Поскольку число *классов библиотеки FCL* достигает нескольких тысяч, то в целях структуризации функционально близкие классы объединяются в группы, называемые *пространством имен* (Namespace).

Основным пространством имен *библиотеки FCL* является *пространство* System, содержащее как классы, так и другие вложенные пространства имен.

Например, в пространстве System.Collections находятся классы и интерфейсы, поддерживающие работу с коллекциями объектов - списками, очередями, словарями.

Пространство System.Windows.Forms содержит классы, используемые при создании *windows*- приложений.

.NET Framework Class library (FCL)

В *.NET* включены сборки библиотеки классов *.NET* Framework *Class library* (*FCL*), содержащие определения нескольких тысяч типов, каждый из которых предоставляет некоторую функциональность. Наборы "родственных" типов собраны в отдельные пространства имен.

Так, *пространство* имен *System* содержит базовый *класс Object*, из которого в конечном счете порождаются все остальные типы.

Таким образом, всякая сущность в *.NET* является объектом со своими полями и методами.

Кроме того, *System* содержит типы для целых чисел, символов, строк, обработки исключений, консольного ввода/вывода, группу типов для

безопасного преобразования одних типов в другие, форматирования данных, *генерации случайных чисел* и выполнения математических операций.

Типами из пространства имен *System* пользуются все приложения.

Для изменения существующего *FCL*-типа можно создать свой собственный *производный тип*. Можно создавать свои собственные пространства имен. Все это будет четко соответствовать принципам, предлагаемым платформой *.NET*

Приведем некоторые наиболее распространенные пространства имен и краткое описание содержащихся там типов (табл. 1.1).

Таблица 1.1 – Основные пространства имен FCL

Пространство имен	**Содержание**
System	Фундаментальные типы данных и вспомогательные классы
System.Collections	Хэш-таблицы, массивы переменной размерности и другие контейнеры
System.Data	Классы ADO .NET для доступа к данным
System.Drawing	Классы для вывода графики (GDI+)
System.IO	Классы файлового и потокового ввода/вывода
System.Net	Классы для работы с сетевыми протоколами, например с HTTP
System.Reflection	Классы для чтения и записи метаданных
System.Runtime.Remoting	Классы для распределенных приложений
System.ServiceProcess	Классы для создания служб Wndows
System.Threading	Классы для создания и управления потоками
System.Web	Классы для поддержки HTTP
System.Web.Services	Классы для разработки web-сервисов
System.Web.Services.	Классы для разработки клиентов web-сервисов Protocols
System.Web.UI	Основные классы, используемые ASP .NET
System.Web.UI. WebControls	Серверные элементы управления ASP .NET
System.Wndows.Forms	Классы для приложений с графическим интерфейсом пользователя
System.Xml	Классы для чтения и ввода данных в формате XML

Все управляемые приложения используют *библиотеку классов FCL*. Это упрощает интеграцию приложений и позволяет легче переносить приложения с одного языка на другой.

Пример 1. Простейшее приложение на C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
     class Program
     {
          static void Main(string[] args)
          {
               Console.WriteLine("Hello, World!");
               Console.ReadKey();
          }
     }
}
```

Поток – это абстрактное представление данных (в байтах), которое облегчает работу с ними. В качестве источника данных может быть файл, устройство ввода-вывода, принтер. Класс Stream является абстрактным базовым классом для всех потоковых классов в Си-шарп. Для работы с файлами нам понадобится класс FileStream (файловый поток). FileStream - представляет поток, который позволяет выполнять операции чтения/записи в файл.

```
static void Main(string[] args)
{
     //открывает файл только на чтение
     FileStream file = new FileStream("d:\\test.txt",
      FileMode.Open, FileAccess.Read);
}
```

Режимы открытия FileMode:

– Append – открывает файл (если существует) и переводит указатель в конец файла (данные будут дописываться в конец), или создает новый файл. Данный режим возможен только при режиме доступа FileAccess.Write.

– Create - создает новый файл(если существует – заменяет)

– CreateNew – создает новый файл (если существует – генерируется исключение)

– Open - открывает файл (если не существует – генерируется исключение)

– OpenOrCreate – открывает файл, либо создает новый, если его не существует

– Truncate – открывает файл, но все данные внутри файла затирает (если файла не существует – генерируется исключение)

```
static void Main(string[] args)
{
     //создание нового файла
     FileStream file1 = new FileStream("d:\\file1.txt",
       FileMode.CreateNew);
     //открытие существующего файла
     FileStream file2 = new FileStream("d:\\file2.txt",
       FileMode.Open);
     //открытие файла на дозапись в конец файла
     FileStream file3 = new FileStream("d:\\file3.txt",
      FileMode.Append);
}
```

Режим доступа FileAccess:

– Read – открытие файла только на чтение. При попытке записи генерируется исключение

– Write - открытие файла только на запись. При попытке чтения генерируется исключение

– ReadWrite - открытие файла на чтение и запись. Чтение из файла

Для чтения данных из потока нам понадобится класс StreamReader. В нем реализовано множество методов для удобного считывания данных. Ниже приведена программа, которая выводит содержимое файла на экран:

```
static void Main(string[] args)
{
     //создаем файловый поток
     FileStream file1 = new FileStream("d:\\test.txt",
       FileMode.Open);
     // создаем «потоковый читатель» и связываем его с файловым
     потоком
     StreamReader reader = new StreamReader(file1);
     //считываем все данные с потока и выводим на экран
     Console.WriteLine(reader.ReadToEnd());
     //закрываем поток
     reader.Close();
     Console.ReadLine();
}
```

Метод ReadToEnd() считывает все данные из файла. ReadLine() – считывает одну строку (указатель потока при этом переходит на новую строку, и при следующем вызове метода будет считана следующая строка).

Свойство EndOfStream указывает, находится ли текущая позиция в потоке в конце потока (достигнут ли конец файла). Возвращает true или false.

Запись в файл

Для записи данных в поток используется класс StreamWriter. Пример записи в файл:

```
static void Main(string[] args)
{
     //создаем файловый поток
     FileStream file1 = new FileStream("d:\\test.txt",
       FileMode.Create);
     //создаем «потоковый писатель» и связываем его с файловым
     потоком
     StreamWriter writer = new StreamWriter(file1);
     //записываем в файл
     writer.Write("текст");
     //закрываем поток. Не закрыв поток, в файл ничего не
     запишется
     writer.Close();
}
```

Метод WriteLine() записывает в файл построчно (то же самое, что и простая запись с помощью Write(), только в конце добавляется новая строка).

Нужно всегда помнить, что после работы с потоком, его нужно закрыть (освободить ресурсы), использовав метод Close().

Кодировка, в которой будут считываться/записываться данные указывается при создании StreamReader/StreamWriter:

```
static void Main(string[] args)
{
     FileStream file1 = new FileStream("d:\\test.txt",
       FileMode.Open);
     StreamReader reader = new StreamReader(file1,
       Encoding.Unicode);
     StreamWriter writer = new StreamWriter(file1,
       Encoding.UTF8);
}
```

Кроме того, при использовании StreamReader и StreamWriter можно не создавать отдельно файловый поток FileStream, а сделать это сразу при создании StreamReader/StreamWriter:

```
static void Main(string[] args)
{
     //указываем путь к файлу, а не поток
     StreamWriter writer = new StreamWriter("d:\\test.txt");
     writer.WriteLine("текст");
     writer.Close();
}
```

Пример 2.

В этом примере класс Employee содержит два закрытых элемента данных – name и salary. Как к закрытым членам, к ним нельзя получить доступ кроме как через методы членов. Для получения управляемого доступа к закрытым членам можно использовать методы с именем GetName и Salary. Доступ к методу name можно получить через открытый метод, а к методу salary – через открытое свойство только для чтения.

```
class Employee2
{
    private string name = "FirstName,
    LastName"; private double salary =
    100.0;

    public string GetName()
    {
        return name;
    }

    public double Salary
    {
        get { return salary; }
    }
}

class PrivateTest
{
    static void Main()
    {
        Employee2 e = new Employee2();

        // The data members are inaccessible (private), so
        // they can't be accessed like this:
        //    string n = e.name;
        //    double s = e.salary;

        // 'name' is indirectly accessed
        via method: string n =
```

```
    e.GetName();

    // 'salary' is indirectly accessed
    via property double s = e.Salary;
}}
```

IV. Выходной контроль.

Разработанные вами приложения согласно индивидуальным вариантам задания продемонстрировать преподавателю. Оформить отчет, в котором указать вариант индивидуального задания, формулировка задания, листинги программ с подробными комментариями, оформленные согласно соглашений по оформлению исходного кода, результаты работы программы в виде скриншотов, общие выводы по работе.

Преподаватель:_____Курносова И.Г..

ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

Вариант	Название класса	Атрибуты	Коллекция
1	Экзамен	имя студента – string дата – string оценка – byte	List<T>
2	Квитанция	номер – uint дата – string сумма – double	LinkedList<T>
3	Автомобиль	марка – string мощность – double стоимость – double	List<T>
4	Корабль	имя – string водоизмещение – double тип – string	LinkedList<T>
5	Животное	имя – string класс – string средний вес – double	List<T>
6	Кадры	имя – string номер цеха – uint разряд – byte	LinkedList<T>
7	Товар	наименование – string количество – uint стоимость – double	List<T>
8	Персона	имя – string возраст – uint пол – char	LinkedList<T>
9	Книга	автор – string название книги – string год издания – double	List<T>
10	Собака	кличка – string порода – string возраст – double	LinkedList<T>
11	Компьютер	процессор – string оперативная память – uint частота – double	List<T>
12	Преподаватель	фамилия – string цикловая комиссия – string педагогический стаж – double	LinkedList<T>
13	Город	название – string площадь – double численность – uint	List<T>
14	Обувь	производитель – string размер – double цена – double	LinkedList<T>
15	Учащийся	фамилия – string группа – string средний балл – double	List<T>