Лабораторная работа 1

Объектно-ориентированное программирование в C #

C# является объектно-ориентированным языком, а это значит, что программа может быть представлена в виде взаимосвязанных объектов, которые взаимодействуют между собой. Описанием объекта является класс, в то время как объект - экземпляр этого класса. Класс определяется с помощью ключевого слова Class:

Class Book {

}

Всю функциональность класса обеспечивают его члены - поля, свойства, методы, конструкторы, события. Поля представляют обычные переменные. Обычным образом также определяются процедуры и функции (в этом плане классы во многом похожи на структуры).

Модификаторы доступа.

Модификаторы доступа позволяют задать область видимости для членов класса. Разделить элементы класса на элементы, которые реализуют класс, и интерфейсную часть, предназначенная для взаимодействия с программой. Такое сокрытие элементов класса называется инкапсуляцией.

Public - публичный или открытый класс или член класса. Доступ к члену класса можно получить из любого места в коде, а к открытому классу - из других программ.

Private - закрытый класс или член класса. Доступ к данному классу или члену класса, можно получить только из кода в том же классе.

Protected - Такие классы или члены класа только из самого класса, либо из наследующих классов.

Кроме обычных методов в классах существуют специальные методы - конструкторы. Конструкторы вызываются при создании нового объекта класса. Имя конструктора должно совпадать с именем класса. Зачем нужен конструктор? Обычно конструктор выполняет инициализацию членов класса. Объявим в классе конструктор, который будет инициализировать поля нашего класса Book:

```
class Book
    {
        public string name;
        public string author;
        public int year;

       public  Book(string name, string author, int year)
        {
           this.name = name;
           this.author = author;
           this.year = year;
        }


     public void GetInformation()
        {
Console.WriteLine("книга '{0}' автор {1} была издана в {2} году",name,
author, year);
```

```
        }

    }
```

Здесь мы создали конструктор, который принимает три параметра - название книги, ее автора и год издания. Затем в конструкторе мы присваиваем значения параметров полям класса. Поскольку параметры и поля класса называются одинаково, то нам надо использовать ключевое слово **this**, которое предоставляет ссылку на данный класс. Если бы параметры имели другие имена, то использование слова **this** было бы необязательным. Если мы не создадим свой конструктор, тогда будет использоваться конструктор по умолчанию:

```
Sub New()

End Sub
```

Теперь используем класс. Создайте новое консольное приложение. Затем нажмите правой кнопкой мыши на название проекта в окне Solution Explorer (Обозреватель решений) и в появившемся меню выберите пункт Add (Добавить), в другом появившемся меню выберите пункт Class (Класс). В окне создания нового класса присвойте ему имя Book и нажмите кнопку Add (Добавить). В проект будет добавлен новый класс Book, который будет находиться в файле Book.cs. Перенесите в этот класс следующий код:

```
class Book
    {
        public string name;
        public string author;
        public int year;

       public  Book(string name, string author, int year)
        {
           this.name = name;
           this.author = author;
           this.year = year;
        }

        public Book()
        {
          name = "Евгений Онегин";
          author = "А.С. Пушкин";
          year = 1833;
        }

     public void GetInformation()
        {
Console.WriteLine("книга '{0}' автор {1} была издана в {2} году",name,
author, year);

        }

    }
```

Теперь в основной файл приложения - в модуль добавим следующий код:

```
class Program
    {
        static void Main(string[] args)
        {
            Book b1 = new Book("Война и мир", "Л.Н. Толстой", 1869);
            Book b2 = new Book();

        //    b1.GetInformation();
            b2.GetInformation();

            Console.ReadKey();


        }
    }
```

Результат:

```
Книга 'Евгений Онегин' (автор А. С. Пушкин) была издана в 1833 году
```

Коллекции объектов

Коллекция является объектом, используемым для группировки связанных объектов и управления ими.
После создания объекта коллекции можно добавлять и удалять элементы, а также получать доступ к элементам коллекции.

Существует несколько типов коллекций, и они отличаются друг от друга по назначению и функциональным возможностям.

Например, Класс List(Of T) представляет простейший список однотипных объектов.

Для добавления в коллекцию объектов используются методы коллекции:

- Add(item As T): добавляет в список новый элемент
- AddRange(collection As ICollection): добавляет в список коллекцию или массив

Рассмотрим использование коллекции на примере:

Создадим еще один класс. Назовем его library. Данный класс будет предназначен для хранения книг в коллекции books.

```
class library
    {
        public List<Book> books = new List<Book> { };
    }
```

Добавим в модуль выделенный фрагмент

```
static void Main(string[] args)
        {
            Book b1 = new Book("Война и мир", "Л.Н. Толстой", 1869);
            Book b2 = new Book();
            Book b3 = new Book("Филипок", "Л.Н. Толстой", 1875);

            library bibl1 = new library();    // создание новой библиотеки
            bibl1.books.Add(b1);              // добавление книг в
коллекцию библиотеки
            bibl1.books.AddRange(new Book[] {b2,b3});

            foreach (Book b in bibl1.books)
            {
                b.GetInformation();

            }
            Console.ReadKey();
```

Результат:

```
Книга 'Война и мир' (автор Л.Н. Толстой) была издана в 1869 году
Книга 'Евгений Онегин' (автор А. С. Пушкин) была издана в 1833 году
Книга 'Филипок' (автор Л.Н. Толстой) была издана в 1875 году
```

Использование делегатов

Наряду со свойствами и методами классы могут иметь делегаты. Делегат (англ. delegate) - такой класс, который позволяет хранить в себе ссылку на метод с определённой сигнатурой.

Расширим функциональные возможности класса library. Добавим в него функцию ***fnd*** поиска книг по ФИО автора. Функция будет возвращать коллекцию книг указанного автора.

```
class library
    {
        public List<Book> books = new List<Book> { };
        private string filterAuthor;

        public List<Book> fnd(string author)
        {
            filterAuthor = author;
            List<Book> res = books.FindAll(PredicateAuthor);
            return res;

        }
        private bool PredicateAuthor(Book bk)
        {
            if (bk.author == filterAuthor)
                return true;
```

```
            else
                return false;
        }


    }
```

В этой функции используется метод **FindAll** коллекции books.

Метод **FindAll** возвращает подмножество объектов исходной коллекции, удовлетворяющих некоторым условиям.

Условия поиска определяются в предикате, на который ссылается делегат, передаваемый в качестве входного параметра функции **FindAll.**

Объекты, для которых предикат возвращает True,
метод **FindAll** помещает в выходную коллекцию.

Использование делегата в данном случае позволяет формировать условия поиска любой сложности.

Предикат представлен приватной функцией **PredicateAuthor**.

Для передачи в предикат условия фильтра используется приватное свойство **filterAuthor.**

```
class Program
    {
        static void Main(string[] args)
        {
            Book b1 = new Book("Война и мир", "Л.Н. Толстой", 1869);
            Book b2 = new Book();
            Book b3 = new Book("Филипок", "Л.Н. Толстой", 1875);

            library bibl1 = new library();    // создание новой библиотеки
            bibl1.books.Add(b1);              // добавление книг в
коллекцию библиотеки
            bibl1.books.AddRange(new Book[] {b2,b3});


            List<Book> res = bibl1.fnd("Л.Н. Толстой");


            foreach (Book b in res)
            {
                b.GetInformation();

            }
            Console.ReadKey();

        }
    }
```

Результат:

Добавьте в класс library метод удаления книги:

```
public void del(string author)
        {
            filterAuthor = author;
            List<Book> res = books.RemoveAll(PredicateAuthor);
            }
```

События

Добавим в класс library событие, которое он будет генерировать в случае удаления книг.

Далее, мы создаем событие при помощи ключевого слова event и связываем его делегатом (MethodContainer), а, следовательно, с методами, имеющими сигнатуру void (void). Событие должно быть public, т.к. его должны использовать разные классы, например, класс Handler1.
Событие имеет синтаксис: public event <НазваниеДелегата> <НазваниеСобытия>;
Название делегата — это имя делегата, на который «ссылаются» наши методы.

```
public event MethodContainer onDel;
```

```
class library
    {
        public List<Book> books = new List<Book> { };
        private string filterAuthor;


        public delegate void MethodContainer();
        //Событие OnDel с типом делегата MethodContainer
        public event MethodContainer onDel;


        public List<Book> fnd(string author)
        {
            filterAuthor = author;
            List<Book> res = books.FindAll(PredicateAuthor);
            return res;

        }
```

```
        public  void del(string author)
        {
            filterAuthor = author;
            books.RemoveAll(PredicateAuthor);
            onDel();

        }

        private bool PredicateAuthor(Book bk)
        {
            if (bk.author == filterAuthor)
                return true;
            else
                return false;
        }

    }
```

Создадим класс, который должен реагировать на возникновение события “УДАЛЕНИЕ КНИГ”.

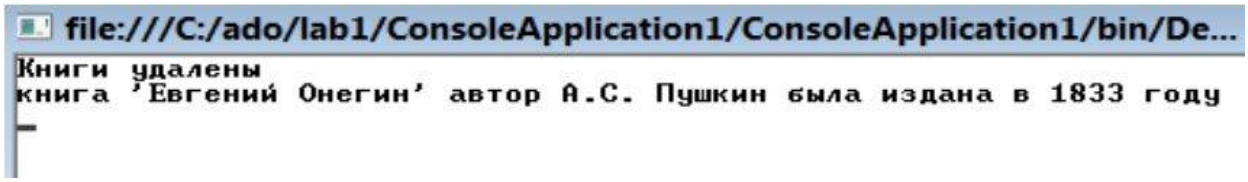
```
class Handler1 //Это класс, реагирующий на событие (удаляются книги)
записью строки в консоли.
    {
        public void Message()
        {

            //для вывода в консольном приложении
            Console.WriteLine("Книги удалены");
        }
    }
```

Вернемся в точку входа программы **main** и создадим экземпляр класса **Handler1**, который должен реагировать на возникновение события, а также укажем событию onCount, метод, который должен запуститься.

```
static void Main(string[] args)
        {
            Book b1 = new Book("Война и мир", "Л.Н. Толстой", 1869);
            Book b2 = new Book();
            Book b3 = new Book("Филипок", "Л.Н. Толстой", 1875);

            library bibl1 = new library();    // создание новой библиотеки

            Handler1 h1 = new Handler1();
            bibl1.onDel += h1.Message; // метод сигнатура которого
совпадает с сигнатурой делегата
```

```
            //добавление книг в коллекцию библиотеки
            bibl1.books.AddRange(new Book[] {b1,b2,b3});

            bibl1.del("Л.Н. Толстой");   // удаление книг


            foreach (Book b in bibl1.books)
            {
                b.GetInformation();

            }

            Console.ReadKey();

        }
```

Результат:



Задание

1. Для каждой книги нужно хранить количество экземпляров. Включить в класс Library метод добавления книг. Если оказывается, что добавляемая книга уже есть в библиотеке, нужно увеличить количество экземпляров, если нет — добавить новую книгу.
2. Добавить в программу обработку события “Книги добавлены”