**Лабораторная работа № 7.2**

**Классы**

**Цель работы:**

Приобретение навыков практического применения, закрепление знаний при работе со свойствами классов.

**Задачи:**

1. Модифицировать созданное в Л.р. №7 приложение, таким образом, чтобы работа с созданным списком (List) производилась с помощью индексатора;
2. Организовать доступ к полям класса через свойства;
3. Создать библиотеку dll, в которой будет находиться созданные класс и структура, с библиотекой будет продолжена работа в следующей лабораторной работе (как создать библиотеку см. Павловская «ПЯВУ C#», Создание библиотеки стр. 275 - 279);
4. Выполнить требования к лабораторным работам (см. файл «Требования к лабораторным»);
5. Ознакомиться с литературой по теме лабораторной работы;
6. Ответить на контрольные вопросы.

**Контрольные вопросы:**

1. Для чего используется ключевое слово this?
2. Что такое свойства? Для чего они нужны?
3. Что такое индексаторы?

**Пояснения и примеры:**

**Свойство** — это член класса, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного поля. Свойства можно использовать, как если бы они являлись открытыми членами данных, хотя в действительности они являются специальными методами, называемыми методами доступа. Это обеспечивает простой доступ к данным и позволяет повысить уровень безопасности и гибкости методов.

Свойства объединяют функции полей и методов. Для объекта, использующего какой-либо объект, свойство является полем, поэтому для доступа к свойству требуется тот же синтаксис, что и для поля. Для средства реализации класса свойство является одним или двумя блоками кода, представляющими метод доступа [get](http://msdn.microsoft.com/ru-ru/library/ms228503.aspx) и/или метод доступа [set](http://msdn.microsoft.com/ru-ru/library/ms228368.aspx). Блок кода для метода доступа get выполняется, когда осуществляется чтение свойства; блок кода для метода доступа set выполняется, когда свойству присваивается новое значение. Свойство без метода доступа set считается доступным только для чтения. Свойство без метода доступа get считается доступным только для чтения. Свойство с обоими методами доступа доступно для чтения и для записи.

В отличие от полей свойства не классифицируются как переменные. Поэтому свойство нельзя передать в качестве параметра [ref](http://msdn.microsoft.com/ru-ru/library/14akc2c7.aspx) или [out](http://msdn.microsoft.com/ru-ru/library/t3c3bfhx.aspx).

Свойства имеют множество применений: с их помощью можно проверить данные перед разрешением изменения, они могут прозрачно представлять данные в классе, куда эти данные извлекаются из какого-либо другого источника, например базы данных, они могут выполнять действие при изменении данных, например вызов события или изменение значения в других полях.

Свойства объявляются в блоке класса с помощью последовательного указания уровня доступа для поля, типа свойства, имени свойства и блока кода, в котором объявляется метод доступа get и/или set. Пример:

public class Date

{

private int month = 7;

public int Month

{

get

{

return month;

}

set

{

if ((value > 0) && (value < 13))

{

month = value;

}

}

}

}

В данном примере Month объявляется как свойство, поэтому метод доступа set может обеспечить задание для свойства Month значения от 1 до 12. Свойство Month использует частное поле для отслеживания фактического значения. Фактическое местоположение данных свойства часто называется "резервным хранилищем" этого свойства. Обычно в качестве резервного хранилища свойств используются частные поля. Поле помечается как частное для того, чтобы предотвратить изменение других полей при вызове данного свойства.

[**Метод доступа get**](javascript:void(0))

Основная часть метода доступа get похожа на основную часть метода. Она должна возвращать значение типа свойства. Выполнение метода доступа get эквивалентно считыванию значения поля. Например, когда возвращается частная переменная из метода доступа get и разрешена оптимизация, вызов метода доступа get встраивается компилятором, что позволяет избежать ненужных затрат на вызов метода. Однако виртуальный метод доступа get не может быть встроен, поскольку во время компиляции у компилятора нет данных о том, какой метод может быть вызван во время выполнения. Ниже приведен метод доступа get, который возвращает значение частного поля name:

class Person

{

private string name; // поле name

public string Name // свойство Name

{

get

{

return name;

}

}

}

При создании ссылки на свойство, кроме случая присвоения ему значения, для чтения значения свойства вызывается метод доступа get. Примеры.

Person person = new Person();

//...

System.Console.Write(person.Name); // здесь вызывается метод доступа get

Метод доступа get должен заканчиваться оператором [return](http://msdn.microsoft.com/ru-ru/library/1h3swy84.aspx) или [throw](http://msdn.microsoft.com/ru-ru/library/1ah5wsex.aspx), а элемент управления не должен выходить за основную часть метода доступа.

[**Метод доступа set**](javascript:void(0))

Метод доступа set похож на метод, имеющий тип возвращаемого значения [void](http://msdn.microsoft.com/ru-ru/library/yah0tteb.aspx). В нем используется неявный параметр value, тип которого соответствует типу свойства. В следующем примере метод доступа set добавляется в свойство Name:

class Person

{

private string name; // поле name

public string Name // свойство Name

{

get

{

return name;

}

set

{

name = value;

}

}

}

Когда свойству присваивается значение, выполняется вызов метода доступа set с помощью аргумента, предоставляющего новое значение. Примеры.

Person person = new Person();

person.Name = "Joe"; // здесь вызывается метод доступа set

System.Console.Write(person.Name); // здесь вызывается метод доступа get

**Индексаторы**

Индексаторы являются синтаксическим удобством, позволяющим создавать [класс](http://msdn.microsoft.com/ru-ru/library/0b0thckt.aspx), [структуру](http://msdn.microsoft.com/ru-ru/library/ah19swz4.aspx) или [интерфейс](http://msdn.microsoft.com/ru-ru/library/87d83y5b.aspx), доступ к которому клиентские приложения получают, как к массиву. Чаще всего индексаторы реализуются в типах, главная цель которых — инкапсуляция внутренней коллекции или массива. Например, предположим, что имеется класс с именем "TempRecord", представляющий набор температур по шкале Фаренгейта, полученных в 10 различных моментов в течение 24 часов. Класс содержит массив с именем "temps" типа "float", представляющий температуры, и [DateTime](http://msdn.microsoft.com/ru-ru/library/system.datetime.aspx), представляющий дату регистрации температур. Путем внедрения в этот класс индексатора клиенты получат доступ к температурам в экземпляре TempRecord с помощью float temp = tr[4], а не float temp = tr.temps[4]. Использование индексатора не только упрощает синтаксис для клиентских приложений, но и делает класс и его назначение интуитивно понятными для других разработчиков.

Чтобы объявить индексатор для класса или структуры, используйте ключевое слово [this](http://msdn.microsoft.com/ru-ru/library/dk1507sz.aspx) как показано в следующем примере:

public int this[int index]

{

// методы доступа set и get

}

**Описание**

В следующем примере показано, как объявить закрытое поле массива temps и индексатор. Индексатор обеспечивает прямой доступ к экземпляру tempRecord[i]. В качестве альтернативы применению индексатора можно объявить массив как член типа [public](http://msdn.microsoft.com/ru-ru/library/yzh058ae.aspx) осуществлять прямой доступ к его членам tempRecord.temps[i].

Обратите внимание, что при вычислении доступа индексатора, например, в инструкции Console.Write вызывается метод доступа [get](http://msdn.microsoft.com/ru-ru/library/ms228503.aspx). Таким образом, если не существует метода доступа get, происходит ошибка времени компиляции.

class TempRecord

{

// Array of temperature values

private float[] temps = new float[10] { 56.2F, 56.7F, 56.5F, 56.9F, 58.8F,

61.3F, 65.9F, 62.1F, 59.2F, 57.5F };

// To enable client code to validate input

// when accessing your indexer.

public int Length

{

get { return temps.Length; }

}

// Indexer declaration.

// If index is out of range, the temps array will throw the exception.

public float this[int index]

{

get

{

return temps[index];

}

set

{

temps[index] = value;

}

}

}

class MainClass

{

static void Main()

{

TempRecord tempRecord = new TempRecord();

// Use the indexer's set accessor

tempRecord[3] = 58.3F;

tempRecord[5] = 60.1F;

// Use the indexer's get accessor

for (int i = 0; i < 10; i++)

{

System.Console.WriteLine("Element #{0} = {1}", i, tempRecord[i]);

}

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

}

/\* Output:

Element #0 = 56.2

Element #1 = 56.7

Element #2 = 56.5

Element #3 = 58.3

Element #4 = 58.8

Element #5 = 60.1

Element #6 = 65.9

Element #7 = 62.1

Element #8 = 59.2

Element #9 = 57.5

\*/

**Литература:**

1. Павловская Т.А. С#. Программирование на языке высокого уровня. Ключевое слово this 114 стр., Свойства 120 – 124, Индексаторы 157 – 161;
2. Статья «Использование индексаторов (Руководство по программированию в C#)»;
3. Рихтер C# via .Net. Глава 9 Свойства 204 – 216 стр.;
4. Троелсен - Язык программирования С# 2005 и платформа .NET 3 е издание. Инкапсуляция на основе свойств класса 215 – 222 стр.