

Лекция 7. ПОНЯТИЕ КЛАССА

7.1 Понятие класса

Мы уже отмечали, что язык С# является объектно-ориентированным языком программирования, основным понятием которого является понятие класса. Во всех примерах программ при изучении языка С# мы постоянно использовали структуры типа класс и даже делали робкие попытки дать определение класса, но всегда при этом говорили, что изучением класса мы займемся в одной из следующих лекций.

В этой лекции мы будем заниматься только классами.

Начнем с понятия класса. Естественно начнем с определений, которые приводят в своих книгах по программированию на языке С# преподаватели ВУЗов.

Фаронов В.В. определяет класс как фрагмент кода, перед которым стоит зарезервированное слово `class` (2 стр. 36). Или «Класс – это всего лишь тип данных, то есть «схема», по которой изготавливаются объекты – реальные экземпляры класса.»(стр. 45).

Павловская Т.А. приводит следующее определение класса «Класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых экземплярами класса.»(3 стр. 13).

Необходимо отметить, что ООП существовало до появления языка С# и понятие класса существует уже давно. Самое короткое определение, которое я встречал в литературе это следующее определение класса – «Классы – это типы, определяемые программистом».

В этом определении отображена очень важная особенность класса – это тип данных, новый по сравнению с массивами, записями или структурами. Однако не всякий тип данных, определяемый программистом, является классом. Второй важной особенностью в определении класса должен быть состав класса или хотя бы намек на его состав и в тоже время определение класса должно быть кратким и легко запоминаемым, например.

Класс это тип данных, содержащий поля, методы и события.

Тип данных - это семантическая единица, которая описывает свойства и поведение множества объектов, называемых экземплярами класса.

Синтаксически класс представляет описание данных, называемых полями класса, описание методов класса и описание событий класса.

Некоторые авторы выделяют классы представленные модулями в самостоятельные группы, например, классы элементов управления при визуальном программировании. Такие классы несут дополнительную нагрузку. Они являются самостоятельными архитектурными единицами построения проектов.

Изучение класса начнем с изучения его формата записи.

Обязательный формат записи класса содержит служебное слово **class**, за которым, располагается его имя и далее в фигурных скобках находится тело класса. Это так называемый минимальный состав описания класса.

Общее описание класса, включающее необязательные элементы (они выделены квадратными скобками), имеет следующий формат записи:

[атрибуты] [спецификаторы] **class** имя_класса [: родители]
{ тело_класса } где,

атрибуты – задают дополнительную информацию о классе;

спецификаторы – определяют условие доступа к составляющим класса.

родители – базовые классы, которые наследует наш класс:

тело класса – определяет состав элементов класса.

Возможными спецификаторами в объявлении класса могут быть **abstract**, **sealed** и **protected**, о которых подробно будет говориться при рассмотрении наследования. Спецификаторы **private**, **public**, **static** и **internal** определяют доступность класса для программы. Говорят, что спецификатор **private** полностью закрывает видимость класса, а **public** делает класс видимым (доступным) для любого фрагмента программы. По умолчанию класс имеет спецификатор доступа **internal** – класс доступен в сборке, в которой он определен. Спецификатор **static** позволяет использовать класс и его элементы, не создавая переменной этого класса (объекта класса).

Все перечисленные спецификаторы применимы как для класса, так и для отдельных его членов, например, полей, методов.

Некоторые необязательные элементы формата описания класса мы будем рассматривать по мере изучения дисциплины.

Класс это тип данных – шаблон, который можно «наполнить» некоторыми значениями, т.е. получить экземпляр класса – переменную типа **класс** или объект.

Подставляя в программу (в модель решения задачи) различные значения, мы будем получать разные объекты класса (различный результат работы программы), но тип класса (код программы) останется неизменным.

Класс можно описывать непосредственно внутри пространства имен или внутри другого класса. В последнем случае класс называется вложенным.

В языке **C#** класс является ссылочным типом и для размещения объекта класса в памяти компьютера необходимо использовать оператор **new**.

7.2 Состав класса

Как мы уже отмечали, тело класса может содержать данные, методы их обработки и события – все эти составляющие часто называют элементами класса.

Рассмотрим основные элементы класса и их назначение:

- константы класса хранят неизменяемые значения;
- поля класса (типы и имена переменных класса);

- методы класса это поименованный фрагмент кода программы, предназначенный для работы с данными класса;
- свойства класса это совокупность методов, позволяющих классу обмениваться (читать или записывать) значениями полей класса с другими классами программы;
- конструкторы класса это специальные методы класса, которые предназначены для создания объектов класса и присваивания начальных значений полям класса;
- деструкторы класса это специальные методы, определяющие порядок действий при освобождении ресурсов, выделенных объекту;
- события класса это специальные методы, позволяющие классу реагировать на действия пользователя или на определенные изменения в программе;
- типы это типы данных, внутренние по отношению к классу. Например, перечисления, структуры, классы, делегаты, интерфейсы.
- индексы это средство доступа к элементам данных класса (обычно массивам) по их порядковому номеру;
- операции это специальные действия с объектами класса с помощью знаков операций.

Данными класса могут быть константы или переменные (поля) класса. При объявлении данных в классе обычно указывается спецификатор доступа к нему, например,

private int a;

Общий формат записи данных класса при их объявлении имеет следующий вид:

[атрибуты] [спецификаторы] [const] тип имя [= начальное_значение].

Обычно данные класса «закрывают для программы» – используют спецификатор **private**. Если перед данными используются спецификатор **public**, то они являются доступными «программе».

По умолчанию, как для данных, так и для методов применяется спецификатор **private**.

Объект – это переменная типа класс и при его создании в памяти компьютера выделяется отдельная область, в которой хранятся значения элементов класса.

Однако в классе могут присутствовать статические элементы класса, которые существуют в единственном экземпляре для всех объектов класса. Часто статические данные называют данными класса, а остальные – данными экземпляра класса, т.е. объекта.

Доступ к некоторым элементам класса (методам) и полям возможен только после создания объекта. Если доступ разрешен, то для обращения к ним используется оператор «точка», например, для некоторого объекта **stud** доступ к полю **name** возможен следующим образом **stud.name = “Иванов”;**.

Аналогичным образом для объекта можно вызывать метод его класса, например, **stud.poisk(a)**;, где **poisk(int a)** метод класса, для которого создан объект **stud**.

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**.

7.3 Методы класса

Метод — это поименованный функциональный элемент класса, предназначенный для работы с данными и методами данного класса.

Методы определяют набор действий, которые доступны классу (часто говорят, что они определяют поведение класса).

Метод описывается один раз, а может вызываться для различных объектов класса столько раз, сколько необходимо.

Общий формат записи методов класса имеет следующий вид:

[атрибуты] [спецификаторы] тип метода имя метода ([параметры])
{тело метода}

Например,

```
static void Main(string[] args)
{ }
```

Наиболее часто встречаемые спецификаторы это **private**, **public** и **static**.

Любые методы класса, объявленные со спецификатором **private**, доступны только в методах данного класса.

Спецификатором **public** делает метод доступным в любом месте программы.

Спецификатор **static** означает, что к методу можно обращаться «на уровне класса» не создавая объект класса — это очень важно, так как в данной дисциплине мы будем очень часто использовать статические методы.

Другими методами доступными программе без создания объекта класса являются конструкторы класса (они и создают объект).

Доступ к остальным методам возможен только после создания объекта класса.

Если спецификатор не указан то (по умолчанию) считается, что данный метод класса имеет спецификатор **private**.

Тип метода может задаваться любым определенным в программе или стандартным типом языка **C#** или **void** — без типа. Например:

```
int kol(int a) { ... }  
public double sym(out float r) { ... }  
public void poisk(ref float s) { ... }  
public int funkcij( int a, out int b, params int[] c) { ... }
```

Если задан тип метода (кроме **void**), то последним оператором тела метода должен быть оператор **return**, возвращающий результат работы метода. При этом метод необходимо присваивать некоторой переменной или использовать как выражение в операторах языка C#. Часто такие методы называют функциями.

Если перед методом указан тип **void**, то метод не должен возвращать результат своей работы с помощью оператора **return** (оператор **return** в этом случае отсутствует в теле метода). Часто такой метод называют процедурой – ее не надо присваивать переменной, а можно записывать как отдельную подпрограмму – процедуру (имя метода с указанием в круглых скобках ее параметров).

Имя метода – идентификатор, определяемый программистом. Желательно в имя метода закладывать смысловое назначение метода, например, **sym**, **max**, **poisk** и т.д.

Параметры метода (формальные параметры) предназначены для обмена данными между методом и программой. Часто параметры метода называют средством «настройки» метода на выполнение необходимого алгоритма.

В языке C# различают следующие параметры методов:

- параметры-значения (входные параметры, т.е. получаемые методом);
- выходные-параметры (помечаются служебным словом **out**);
- параметры-ссылки (помечаются служебным словом **ref**);
- параметры-массивы (помечаются служебным словом **params**).

Параметры-значения не имеют помечающего служебного слова.

Параметров метода класса разделяются запятыми. Параметр-массив в методе может быть только один и должен быть последним в списке параметров.

Если в методе объявлены параметры-значения, то это означает, что метод получает в свое распоряжение копии некоторых переменных. Метод может изменять значения этих копий, но их оригинал (в программе) остается неизменным. По окончании работы метода параметры-значения удаляются из памяти компьютера.

Выходные-параметры метода предназначены для передачи результатов работы метода в программу. В теле метода обязательно должны находиться операторы присваивания выходным-параметрам некоторых значений, иначе сообщение об ошибке во время компиляции программы.

Если в методе объявлены параметры-ссылки, то фактически метод получает в свое распоряжение адреса соответствующих переменных и может их использовать по своему алгоритму (читать или писать новые значения).

Объявленные в методе параметры-массивы предназначены для работы с произвольным числом фактических переменных. При этом формальному

параметру, стоящему за служебным словом **params** ставится в соответствие массив произвольной длины данных.

Таким образом, через свои параметры метод может, как получать необходимые значения (параметры-значения и параметры-ссылки), так и возвращать результаты своей работы (выходные-параметры и параметры-ссылки).

Тело метода содержит фрагмент кода программы, реализующий некоторый алгоритм. При этом метод выступает как некоторый шаблон действия с формальными параметрами. В программе вместо формальных параметров необходимо использовать реальные переменные – фактические параметры и шаблон действия метода будет применяться для реальных переменных.

7.4 Структура объекта

Рассматривая классы, мы отмечали, что переменной типа класс является объект. Объект переменная, следовательно, ей выделяется место в памяти компьютера, в которой хранится информация объекта.

Рассмотрим, что конкретно хранится в памяти, соответствующей объекту.

Естественно, это значения всех полей данных класса.

Специальное поле **this** (параметр по ссылке), который формируется автоматически при создании объекта и содержит адрес объекта.

Фактически связь объекта с методами класса происходит через параметр **this**. Каждый метод класса может непосредственно обращаться к параметру **this** для работы с элементами текущего объекта. Поскольку значение **this** всегда соответствует текущему объекту (объекту, с которым в текущий момент работает программа), то методы класса будут работать с элементами текущего объекта.

Использование указателя **this** позволяет не создавать в каждом объекте копии методов класса, для работы с объектом. Таким образом, методы класса не тиражируются для каждого объекта.

7.5 Пример учебной программы

Рассмотреть чисто учебный пример по созданию класса треугольник (на основе примера из первой лекции).

На этапе визуального программирования мы будем использовать те же три стандартных элемента управления из окна Toolbox: статический текст или метка (Label), поле ввода или окно редактирования (TextBox) и командную кнопку (Button), но расположи их на форме в другом порядке.

Исходный код файла Form1.cs:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;
```

```

using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public class treyg
        {
            private int a, b, c, p;
            public string ss;

            public void vvod(int sa, int sb, int sc)
            {
                if (sa > 0 && sb > 0 && sc > 0)
                {
                    if (sa + sb > sc && sa + sc > sb && sb + sc > sa)
                    {
                        a = sa; b = sb; c = sc;
                        p = a + b + c;
                        ss = "Периметр треугольника = " + p.ToString();
                    }
                    else
                        ss = "Одна из сторон треугольника больше суммы двух других Повторите ввод ";
                }
                else
                    ss = "Одна из сторон треугольника меньше 0! Повторите ввод ";
            }
        }

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int A, B, C;
            treyg t = new treyg();
            A = Convert.ToInt32(textBox1.Text);
            B = Convert.ToInt32(textBox2.Text);
            C = Convert.ToInt32(textBox3.Text);
            t.vvod(A, B, C);
            textBox4.Text = t.ss;
        }
    }
}

```

Рассмотрим подробнее некоторые элементы класса и их использование в программе.

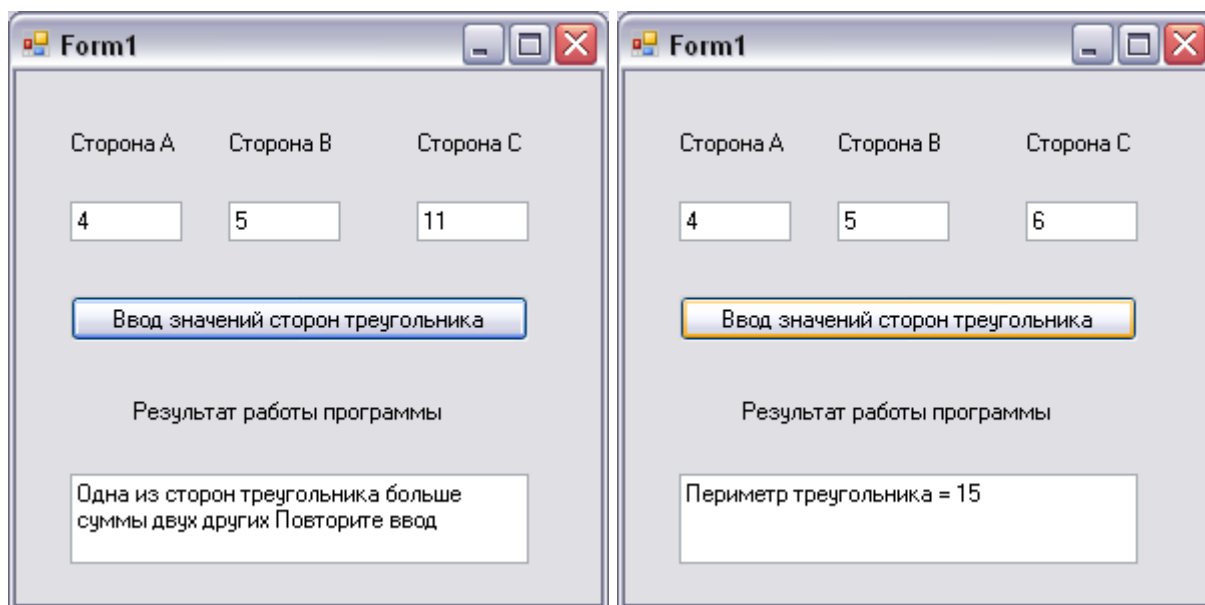


Рисунок 7.1 Окна программы «Треугольник Класс».

В первую очередь для работы с данными и методами класса `class treyg` необходимо создать объект этого класса – переменная `t`

```
treyg t = new treyg();
```

Данные класса `private int a, b, c, p;` являются закрытыми. Это означает, что доступ к элементам данных класса возможен только с помощью его методов. Например, если после создания объекта `t` попытаться присвоить новое значение элементу данных `b` (`t.b = 3;`), то это действие вызовет сообщение об ошибке т.к. непосредственное обращение к элементам данных класса запрещено спецификатором доступа **private**.

В классе `treyg` использованы два метода – конструктор (по умолчанию) и метод ввода значений сторон треугольника.

Конструктор создает объект с «нулевыми» значениями его полей данных.

Задание значений полям данных класса `treyg` осуществляется методом `public void vvod(int sa, int sb, int sc)`, которому в качестве фактических параметров задаются значения переменных А, В и С, введенные в режиме диалога.

В программе рассмотрены варианты «неправильного» задания значений сторон треугольника и печать соответствующих комментариев. Однако в ней не предусмотрена защита от нажатия кнопки «Ввод значений сторон треугольника» с «пустыми» окнами ввода.

7.6 Доступ к полям

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

Модификатор `private`

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`

Этот модификатор открывает поля классам наследникам. Если класс А объявил некоторое поле с модификатором `protected`, то методы класса В, который является наследником класса А и, следовательно, наследует поля класса А, могут непосредственно работать с наследуемыми полями.

Модификатор `internal`

Этот модификатор открывает поля дружественным классам. Два класса А и В называются дружественными, если они принадлежат одной сборке - одному проекту. Если класс А объявил некоторое поле с модификатором `internal`, то методы дружественного класса В, являющегося клиентом класса А, могут непосредственно работать с таким полем.

Комбинация атрибутов `protected` и `internal`

Эта комбинация открывает поле тем классам, которые являются либо наследниками, либо дружественными классами. Если требуется более строгое ограничение доступа к полю, чтобы оно было доступно только тем наследникам, которые являются дружественными классами, то сам класс нужно объявить с модификатором `internal`, а соответствующее поле - с модификатором `protected`.

Если поля доступны только для методов класса, то они имеют модификатор доступа `private`, который можно опускать. Такие поля считаются закрытыми, но часто желательно, чтобы некоторые из них были доступны в более широком контексте. Если некоторые поля класса А должны быть доступны для методов класса В, являющегося потомком класса А, то эти поля следует снабдить модификатором `protected`. Такие поля называются защищенными. Если некоторые поля должны быть доступны для методов классов В1, В2, и так далее, дружественных по отношению к классу А, то эти поля следует снабдить модификатором `internal`, а все дружественные классы В поместить в один проект (`assembly`). Такие поля называются дружественными. Наконец, если некоторые поля должны быть доступны для методов любого класса В, которому доступен сам класс А, то эти поля следует снабдить модификатором `public`. Такие поля называются общедоступными или открытыми.