

Лекция 8 ЭЛЕМЕНТЫ КЛАССОВ

8.1 Конструкторы

Основное назначение конструктора – создать объект и элементам данных этого объекта присвоить некоторые начальные значения. Иногда этот процесс называют инициализацией объекта.

Создание объекта является основной задачей конструктора.

Присваивание элементам данных объекта некоторых значений может осуществляться различными способами.

В зависимости от способа определения значений элементов данных объекта различают:

- конструкторы с умолчанием;
- конструкторы с заданием параметров;
- множественные конструкторы.

Конструктор с умолчанием вызывается в программе, без каких либо параметров. Например,

```
treug t1 = new treug();
```

Это означает, что элементам данных объекта t1 присваиваются некоторые фиксированные значения – обычно нулевые.

Реализация такого конструктора может включать операторы присваивания полям данным класс некоторых фиксированных значений.

Например, в наш класс можно добавить конструктор с умолчанием, который присваивает (по умолчанию) сторонам треугольника фиксированные значения 3, 4 и 5. В этом можно убедиться, если добавить метод печати периметра объекта:

```
public treug()
{
    a = 3; b = 4; c = 5;
}
public void printO()
{
    p = a + b + c;
    ss = "Периметр треугольника = " + p.ToString();
}
```

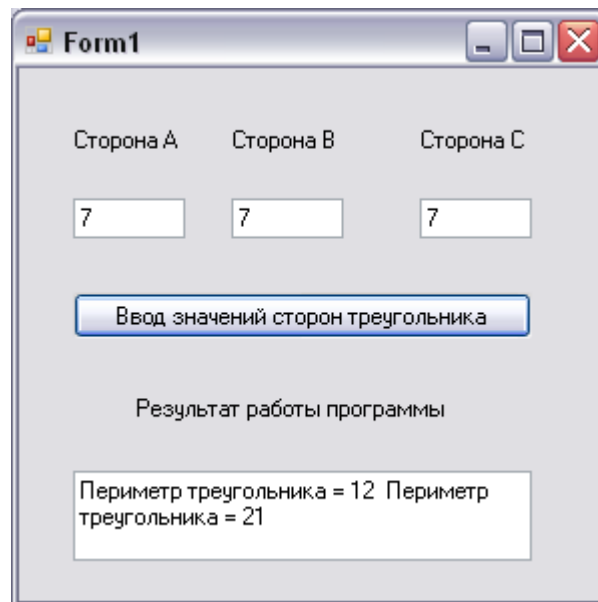


Рисунок 8.1 – Использование в программе конструктора с фиксированными начальными значениями

Конструкторы с заданием параметров позволяют определять начальные значения элементов данных объекта во время создания объекта. Объявление таких конструкторов в классе выглядит следующим образом:

```
public treyg(int sa, int sb, int sc)
{
    a = sa; b = sb; c = sc;
}
```

Многие программисты включают в реализацию конструктора проверку на допустимые значения элементов данных объектов, создавая так называемые «умные» конструкторы. Например, в реализацию конструктора класса треугольник, можно включить следующие проверки элементов данных:

- все стороны треугольника данных должны быть больше 0;
- сумма любых двух сторон треугольника должна быть больше третьей стороны.

Если условия не выполняются необходимо выдать сообщение и присвоить элементам данных объекта некоторые фиксированные значения или повторить ввод.

Объявление такого «умного» конструктора ни чем не отличается от объявления обычного конструктора, а реализация включает все необходимые проверки в функции ввода данных класса, например:

```
public treyg(int sa, int sb, int sc)
{
    vvod(sa, sb, sc);
}
```

Изменим код программы нашего обработчика события следующим образом – умышленно введем в конструкторе неправильные значения сторон треугольника:

```
private void button1_Click(object sender, EventArgs e)
{
    int A, B, C;
    treyg t = new treyg(3,5,9);
    t.printO();
    A = Convert.ToInt32(textBox1.Text);
    B = Convert.ToInt32(textBox2.Text);
    C = Convert.ToInt32(textBox3.Text);
    t.vvod(A,B,C);
    textBox4.Text = t.ss;
}
```

Работа программы на рисунке 1.6

Рисунок 8.2 – Работа программы с заведомо неправильными значениями в конструкторе

Множественные конструкторы используют перегрузку функций при обработке аргументов разного типа.

Такие конструкторы используются, когда значения данных могут задаваться в различных формах, например, целым или вещественным числами или строкой.

Обычно такие конструкторы применяются для обработки дат.

Используя перегрузку функций можно в описание класса включать несколько конструкторов, «понимающих» данные различного типа.

Например, текущую дату можно задавать по умолчанию или вводить разными способами. Возможны, например, следующие три варианта значений ввода даты:

целыми числами (месяц, день и год – 23 12 07);

некоторым текстом (23 декабря 2007 года);

набором с использованием дополнительных символов (20.10.07 или 17/09/2007).

```

class date
{
    ...
    date() { ... }
    date(int mm, int dd, int gg) { ... }
    date (string tekct) { ... }
    ...
};

```

При инициализации объекта возможен любой из предложенных вариантов.

8.2 Деструкторы

В языке C# классы могут содержать специальные методы деструкторы. В функциональном плане они должны осуществлять действия, обратные тем, что реализуют конструкторы. Однако особенностью деструкторов языка C# является то, что они не занимаются освобождением памяти, выделяемой конструктором соответствующим объектом (за этим следит сборщик мусора), а освобождают ресурсы, выделенные объекту, например, закрывают связанные с ним файлы, связь с сервером базы данных, связь с другим компьютером и т.д.

Также как у конструктора имя деструктора совпадает с именем класса, но перед именем деструктора устанавливается символ «~» – тильда. Например, если бы наш класс `treyg` содержал деструктор, то его запись выглядела бы следующим образом:

```

public ~treyg()
{ тело_деструктора }

```

Деструктор нельзя вызвать непосредственно в программе — он автоматически вызывается сборщиком мусора при удалении объекта из кучи. Освобождение ресурсов, выделяемых объекту, на практике осуществляется автоматически после того, как надобность в них отпадает, поэтому деструкторы конструктивно присутствуют в структуре класса, но, как правило, не создаются – либо используются по умолчанию.

8.3 Свойства

Один из принципов объектно-ориентированного программирования является инкапсуляция. Формально инкапсуляция это объединение полей и методов класса с целью защиты данных от непосредственного доступа из программы. Поля объекта используется через его интерфейс – совокупность правил доступа или свойства. Скрытие полей объекта – их инкапсуляция (от слова «капсула») осуществляется через свойства.

Свойства представляют собой специальные методы, которые на самом деле состоят из двух методов – **get()** и **set()** и некоторого поля объекта. Метод **get()** возвращает текущее значение соответствующего поля объекта, а метод **set()** помещает в поле объекта новое значение. Эти методы вызываются автоматически всякий раз, когда программа хочет получить значение «свойства» или изменить его. С точки зрения синтаксиса свойства ничем не отличаются от полей: они могут стоять в левой части оператора присваивания либо являться членом выражения в его правой части. Например, закрытому полю `int` `a` можно записать следующее свойство целого типа `Aa`.

```
public class treyg
{
    private int a, b, c, p;
    public int Aa
    {
        get { return a; }
        set { a = value; }
    }
    public string ss;
    . . .
}
```

В программе, после создания объекта `t` полю `a` можно присвоить новое значение через свойство `Aa` следующим образом:

```
int A, B, C;
treyp t = new treyp();
A = Convert.ToInt32(textBox1.Text);
B = Convert.ToInt32(textBox2.Text);
C = Convert.ToInt32(textBox3.Text);
t.Aa = A;
```

или переменной программы `A` присвоить значение поля `a` объекта `t` следующим образом:

```
A = t.Aa;
```

где `t.Aa` – это свойство класса `treyp`.

В приведенном примере свойством `t.Aa` мы фактически закрытое поле `a` класса `treyp` превратили в открытое поле – это можно сделать проще объявив в классе вместо спецификатора `private` спецификатор `public`.

Естественно свойства предназначены для реализации более сложной логики доступом к закрытому полю класса. В нашем примере в свойство установки нового значения полю класса мы можем включить проверку значения на условие `> 0` и на реализацию только одной попытки записи, например:

```
set { if (a == 0 && value > 0) a = value; }
```

Любой из методов доступа свойства (но не оба сразу) может отсутствовать. В этом случае мы получаем свойства, предназначенные только для чтения или только для записи.

8.4 Параметр по ссылке **this**

Специальное поле указатель (**this**), который формируется автоматически при создании объекта и содержит адрес созданного объекта.

Фактически связь полей 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 параметра **this** применяется во многих конструкторах для инициализации полей объектов. Многие авторы при описании конструкторов классов в качестве имен формальных параметров используют имена полей классов и для того чтобы различать поля классов и имена формальных параметров перед именами полей классов указывается параметр **this**. Например, наш конструктор `treys` в этом случае необходимо было бы записать следующим образом:

```
public treys(int a, int b, int c)
{
    this.a = a; this.b = b; this.c = c;
}
```

Избежать конфликта имен можно более простым способом – просто выбрав другие имена для формальных параметров (что мы и сделали в программе).

Параметр **this** нельзя использовать при обращении к статическим элементам класса, так как они принадлежат не конкретному объекту, а классу в целом.

В языке C# применяется еще один параметр по ссылке **base**, который используется для работы с базовым (родительским) объектом. Если Вы внимательно посмотрите исходный код файла **Form1.Designer.cs**, а именно метод **void Dispose(bool disposing)**, то в последних строках кода этого метода обычно находится запись **base.Dispose(disposing);**, которая позволяет при удалении объекта из памяти удалять и родительский (базовый) объект.

8.5 События класса

Важной составной частью ООП является реализованный в классах механизм событий. С помощью этого механизма один объект (источник события) может сообщить другому объекту (получателю события) об изменении своего состояния.

Обычно механизм события используется в многопоточных процессах при синхронизации – упорядочения очередности работы этих потоков. Однако этот же механизм можно использовать в **Windows**-приложениях, в которых такие элементы, как кнопки, флажки, переключатели и т. п., выдают информацию о взаимодействии с ними пользователя. Например, все объекты-

кнопки (класса **Button**) при щелчке мышью возбуждают событие **OnClick**. Но для одной кнопки это событие приведет, например, к вводу набранных значений, для другой выполнит некоторое преобразование значений (например, их сортировку), а для третьей откроет окно другой формы.

Каждый элемент управления, размещаемый на форме, имеет определенный набор событий, «пустые» обработчики которых можно получить с помощью окна свойств этого элемента. Однако разработчики программ могут разработать свои специальные обработчики событий. Для реализации механизма события – извещение клиентов некоторого класса о факте наступления события класс-источник события должен:

- объявить событие как член класса (наряду с полями, методами, свойствами) — для объявления используется зарезервированное слово **event**;

- передать клиентам класса (получателям события) в нужный момент информацию о наступившем событии, сопроводив ее необходимыми параметрами;

- получить от клиента (клиентов) ответ и, проанализировав его, выполнить связанное с событием действие.

Две последних операции (обмен информацией с клиентами) обычно реализуются с помощью делегатов, которые мы будем рассматривать в следующих лекциях.

8.6 Перегрузка операций класса

В некоторых классах используются фрагменты кода (блоки), в заголовках которых используется имя класса (но не конструктор) после которого, но до круглых скобок формальных параметров, используется служебное слово **operator**. С помощью этого слова обозначается механизм перегрузки операций, который позволяет использовать в обычных математических выражениях переменный типа класс. Например, если для класса «студент» перегружена операция сложения (**operator+**) и в программе созданы два объекта типа «студент» – **ct1** и **ct2**, то можно записать выражение **ct1 + ct2**.

Что означает сложение двух объектов? Может быть, мы объединяем их имена или суммируем возраст?

Для этих целей и используется перегрузка операций, которая четко прописывает, что должно суммироваться при суммировании двух объектов, например, необходимо суммировать их оценки.

Служебное слово **operator** указывает, что осуществляется перегрузка операции, а операция « + », что перегружается операция сложения.

Определение собственных операций класса с помощью указания **operator** называют перегрузкой операций. Перегрузка обычно применяется для классов, описывающих математические или физические понятия, то есть таких классов, для которых семантика операций делает программу более понятной.

Перегрузка операции класса описывается с помощью методов специального вида (функций-операций).

Формат записи перегрузки операции имеет следующий вид:

спецификаторы имя класса operator имя операции (формальные параметры) {тело }

В качестве спецификаторов обычно одновременно используются ключевые слова **public** и **static**. Кроме того, операцию можно объявить как внешнюю (**extern**).

Тело операции определяет действия, которые выполняются при использовании операции в выражении. Тело представляет собой блок, аналогичный телу других методов класса.

Например:

```
public static int operator+ (Student S1, Student S2)
{
    return S1.Oценка + S2.Oценка;
}
```

Если мы определили перегружаемую операцию для « + », то это не означает, что мы определили перегрузку операции «++» или «+=». Это другие операции и их необходимо определять самостоятельно.

Если мы вводим перегружаемую операцию для некоторого класса, то ее действия должны быть ограничены только данными этого класса.

Необходимо отметить, что перегрузка операций не создает новые операции, а только адаптирует существующие к данным типа класс.

При описании перегрузки операций необходимо соблюдать следующие правила:

- операция должна быть описана как открытый статический метод класса (спецификаторы **public static**);
- формальные параметры в операцию должны передаваться по значению (то есть не должны предваряться ключевыми словами **ref** или **out**);
- форматы записей всех операций класса должны различаться.

Перегрузка операций обычно используется с целью обеспечения более естественного синтаксиса выражений для типов, определяемых пользователем.

Пример использования перегрузки операции для подсчета суммы оценок первых двух студентов:

```
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 Stydent
    {
        int Ocenka;
        string Name;
        public string ss;
        public int Aa
        {
            get { return Ocenka; }
            set { if (value >= 2 && value <= 5) Ocenka =
value; else ss="Вводите правильно значение оценки \r\n"; }
        }

        public void Vvod(string name)
        {
            Name = name;
        }
        public static int operator+ (Stydent S1, Stydent S2)
        {
            return S1.Ocenka + S2.Ocenka;
        }
    }
    public Stydent[] styd = new Stydent[5];
    public static int n=0;
    public Form1()
    {
        InitializeComponent();
        textBox1.Text = "";
        n = 0;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        int Oc;
        string fio, In;
        Stydent st = new Stydent();
        fio = textBox2.Text;
        st.Vvod(fio);
        In = textBox3.Text;
        Oc = Convert.ToInt32(In);
        st.Aa = Oc;

        styd[n] = st;
        textBox1.AppendText("Имя студента " + fio + " Его
оценка = " + styd[n].Aa.ToString() + "\r\n");
        if (st.Aa != 0) n++;
        if (n == 2) textBox1.AppendText("Сумма оценок двух
студентов = " + (styd[0] + styd[1]).ToString() + "\r\n");
    }
}

```

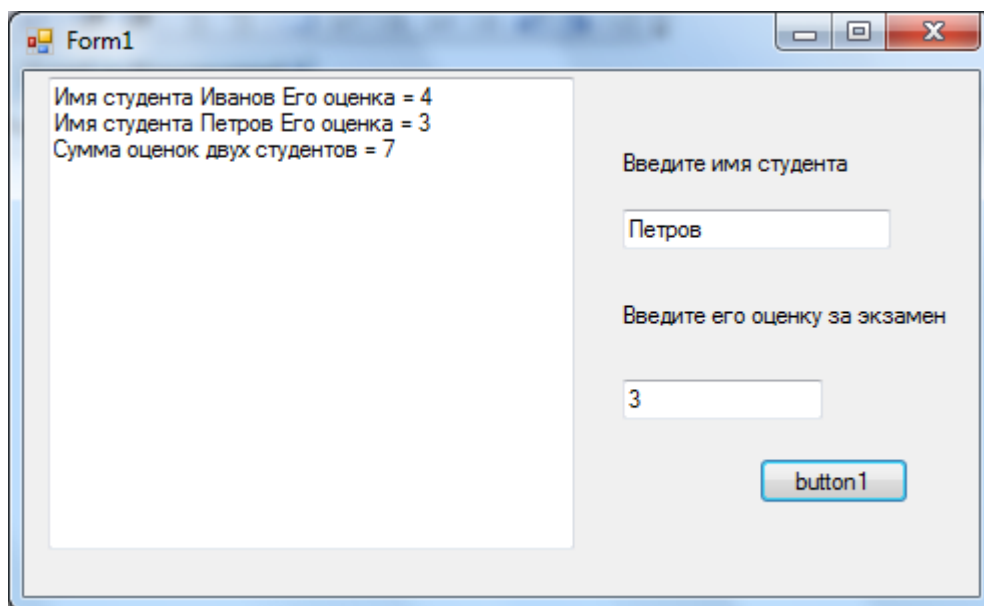


Рисунок 8.3 – Пример перегрузки операции сложения

8.7 Вопросы для самопроверки

- 8.7.1 Для чего предназначен конструктор с умолчанием?
- 8.7.2 Для чего предназначен конструктор с заданием параметров?
- 8.7.3 Как называется процесс определения нескольких методов с одинаковыми именами?
- 8.7.4 Как называются несколько конструкторов одного класса?
- 8.7.5 Когда вызывается деструктор класса `tk`?
- 8.7.6 Как обычно называется метод, если тип его возвращаемого значения объявлен `void`?
- 8.7.7 Как должен заканчиваться в языке `C#` метод если его тип не `void`?
- 8.7.8 Какие формальные параметры метода `C#` называются параметрами-ссылки?
- 8.7.9 Как называется объединение в одной структуре полей и методов их обработки?
- 8.7.10 Какой механизм используется в классах для доступа к «закрытым» полям класса?