УДК 004.42

Т.Б.Казиахмедов
Нижневартовск, Россия

ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ КАК КЛАССОВ И КАК ФУНКЦИОНАЛОВ НА ПРИМЕРЕ АЛГОРИТМА СОРТИРОВКИ

Аннотация: Данная статья описывает концепцию делегатов как классов и как функционалов, основные нотации данной технологии, также здесь приведены примеры использования делегатов в разных нотациях; коды реализованы средствами Microsoft Visual Studio 2010.

Ключевые слова: объектно-ориентированная парадигма; функциональное программирование; делегаты; проектирование; Microsoft Visual Studio.

Сведения об авторе: Казиахмедов Тофик Багаутдинович, зав. кафедрой информатики и методики преподавания информатики.
Место работы: Нижневартовский государственный университет.
Контактная информация: 626448, г. Нижневартовск, ул. Чапаева, д. 91; тел.: 9825314896.
E-mail: ktofik@ yandex.ru

T.B.Kaziakhmedov
Nizhnevartovsk, Russia

USING DELEGATES AS CLASSES AND FUNCTIONALS IN TERMS AS EXEMPLIFIED BY THE SORTING ALGORITHM

Abstract: This article describes the concept of delegates as classes and functional, as well as the basic notations of this technology, and give examples of using delegates in different notations, the codes being developed using Microsoft Visual Studio 2010.

Key words: object-oriented paradigm; functional programming; delegates; design; Microsoft Visual Studio.

About the author: Tofik Bagautdinovich Kaziakhmedov, Head of Department of Informatics and Methods of Teaching Informatics.
Place of employment: Nizhnevartovsk state University.

Прежде чем перейти непосредственно к описанию применения делегатов в различных алгоритмах, необходимо рассмотреть ряд вопросов, касающихся самой концепции программирования с использованием делегатов. Слово «делегат» (delegate) используется в C# для обозначения хорошо известного понятия. Делегат задает определение функционального типа (класса) данных. Экземплярами класса являются функции. Описание делегата в языке C# представляет собой описание еще одного частного случая класса. Каждый делегат описывает множество функций с заданной сигнатурой. Каждая функция (метод), сигнатура которого совпадает с сигнатурой делегата, может рассматриваться как экземпляр класса, заданного делегатом. Синтаксис объявления делегата имеет следующий вид:

[<спецификатор доступа>] delegate <тип результата > <имя класса> (<список аргументов>);

Этим объявлением класса задается функциональный тип — множество функций с заданной сигнатурой, у которых аргументы определяются списком, заданным в объявлении делегата, и тип возвращаемого значения определяется типом результата делегата.

Для объявления делегата, как и у всякого класса, есть две возможности:

- непосредственно в пространстве имен, наряду с объявлениями других классов, структур, интерфейсов;
- внутри другого класса, наряду с объявлениями методов и свойств. Такое объявление рассматривается как объявление вложенного класса.

Так же, как и интерфейсы C#, делегаты не задают реализации. Фактически между некоторыми классами и делегатом заключается контракт на реализацию делегата. Классы, согласные с контрактом, должны объявить у себя статические или динамические функции, сигнатура которых совпадает с сигнатурой делегата. Если контракт выполняется, то можно создать экземпляры делегата, присвоив им в качестве значений функции, удовлетворяющие контракту. Контракт является жестким: не допускается ситуация, при которой у делегата тип параметра — object, а у экземпляра соответствующий параметр имеет тип, согласованный с object, например, int.

Рассмотрим пример объявления 3 делегатов, поместив два из них в пространство имен, третий вложим непосредственно в создаваемый нами класс. Ключевое слово ref используется для передачи аргументов по ссылке. В результате все изменения параметра в методе

будут отражены в переменной при передаче элемента управления обратно в вызывающий метод. Для работы с параметром ref при определении метода, вызывающего этот параметр, должны явно использовать ключевое слово ref.

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

namespace delegates_0
{

//объявление классов — делегатов 2 в пространстве имен
delegate void Proc(ref int x);
delegate void MesToPers(string s);
class OwnDel
{
  public delegate int Fun1(int x);// //объявление класса — делегата внутри класса
  int Plus1( int x){return(x+100);}//Plus1
  int Minus1(int x){return(x-100);}//Minus1
  void Plus(ref int x){x+= 100;}
  void Minus(ref int x){x-=100;}
  //поля класса
  public Proc p1;
  public Fun1 f1;
  char sign;
  //конструктор
  public OwnDel(char sign)
  {
     this.sign = sign;
     if (sign == '+')
     {p1 = new Proc(Plus);
         f1 = new Fun1(Plus1);}
     else
     {p1 = new Proc(Minus);
         f1 = new Fun1(Minus1);}
  }
}//конец описания класс OwnDel
   class Program
   {
      static void Main(string[] args)
       {
         int account = 1000, account1 = 0;
      OwnDel oda = new OwnDel('+');
      Console.WriteLine("account = {0}, account1 = {1}",
              account, account1);
        oda.p1(ref account); account1 = oda.f1(account);
      Console.WriteLine("account = {0}, account1 = {1}",
              account, account1);
           Console.ReadLine();

       }
   }
}
```

Клиент класса OwnDel создает экземпляр класса oda, передавая конструктору знак той операции, которую он хотел бы выполнить над своими счетами — account и account1. Вызов p1 и f1, связанных к моменту вызова с закрытыми методами класса, приводит к выполнению нужных функций.

Одно из наиболее важных применений делегатов связано с функциями высших порядков. **Функцией высшего порядка** называется такая функция (метод) класса, у которой один или несколько аргументов принадлежат к функциональному типу. Без этих функций в программировании обойтись довольно трудно. Классическим примером является функция

вычисления интеграла, у которой один из аргументов задает подынтегральную функцию. Другим примером может служить функция, сортирующая объекты. Аргументом ее является функция Compare, сравнивающая два объекта. В зависимости от того, какая функция сравнения будет передана на вход функции сортировки, объекты будут сортироваться поразному, например, по имени, или по ключу, или по нескольким полям. Вариантов может быть много, и они определяются классом, описывающим сортируемые объекты.

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

namespace delegates_01
{

   public class HighOrderIntegral
   {
       //delegate
       public delegate double SubIntegralFun(double x);
       public double EvalIntegral(double a, double b,
          double eps, SubIntegralFun sif)
       {
           int n = 4;
           double I0 = 0, I1 = I(a, b, n, sif);
           for (n = 8; n < Math.Pow(2.0, 15.0); n *= 2)
           {
               I0 = I1; I1 = I(a, b, n, sif);
               if (Math.Abs(I1 - I0) < eps) break;
           }
           if (Math.Abs(I1 - I0) < eps)
               Console.WriteLine("Требуемая точность достигнута! " +
                  " eps = {0}, достигнутая точность ={1}, n= {2}",
                  eps, Math.Abs(I1 - I0), n);
           else
               Console.WriteLine("Требуемая точность не достигнута! " +
                  " eps = {0}, достигнутая точность ={1}, n= {2}",
                  eps, Math.Abs(I1 - I0), n);
           return (I1);
       }
       private double I(double a, double b, int n,
          SubIntegralFun sif)
       {
           //Вычисляет частную сумму по методу трапеций
           double x = a, sum = sif(x) / 2, dx = (b - a) / n;
           for (int i = 2; i <= n; i++)
           {
               x += dx; sum += sif(x);
           }
           x = b; sum += sif(x) / 2;
           return (sum * dx);
       }
   }//class HighOrderIntegral

   class functions
   {
       //подынтегральные функции
       static public double sif1(double x)
       {
           int k = 1; int b = 2;
           return (double)(k * x + b);
       }
       static public double sif2(double x)
       {
           double a = 1.0; double b = 2.0; double c = 3.0;
           return (double)(a * x * x + b * x + c);
       }
```

```
    }//class functions


    class Program
    {
        static void Main(string[] args)
        {
            double myint1 = 0.0;
            HighOrderIntegral.SubIntegralFun hoisif1 =
               new HighOrderIntegral.SubIntegralFun(functions.sif1);
            HighOrderIntegral hoi = new HighOrderIntegral();
            myint1 = hoi.EvalIntegral(2, 3, 0.1e-5, hoisif1);
            Console.WriteLine("Мой интеграл1 = {0}", myint1);
            HighOrderIntegral.SubIntegralFun hoisif2 =
               new HighOrderIntegral.SubIntegralFun(functions.sif2);
            myint1 = hoi.EvalIntegral(2, 3, 0.1e-5, hoisif2);
            Console.WriteLine("Мой интеграл2 = {0}", myint1);

            Console.ReadLine();
        }
    }
}
```

Приведем некоторые пояснения:

- Класс HighOrderIntegral предназначен для работы с функциями. В него вложено описание функционального класса — делегата SubIntegralFun, задающего класс функций с одним аргументом типа double, возвращающих значение этого же типа.
- Метод EvalIntegral — основной метод класса, позволяет вычислять определенный интеграл. Этот метод есть функция высшего порядка, поскольку одним из его аргументов является подынтегральная функция, принадлежащая классу SubIntegralFun.
- Для вычисления интеграла применяется классическая схема. Интервал интегрирования разбивается на n частей, и вычисляется частичная сумма по методу трапеций, представляющая приближенное значение интеграла. Затем n удваивается, и вычисляется новая сумма. Если разность двух приближений по модулю меньше заданной точности eps, то вычисление интеграла заканчивается, иначе процесс повторяется в цикле. Цикл завершается либо по достижении заданной точности, либо когда n достигнет некоторого предельного значения (в нашем случае — 215).
- Вычисление частичной суммы интеграла по методу трапеций реализовано закрытой процедурой I.
- В классе Functions описано несколько функций, удовлетворяющих контракту, который задан классом SubIntegralFun.

Чаще всего создание экземпляров удобнее возложить на класс, создающий требуемые функции. Более того, в этом классе делегат можно объявить как **свойство** класса. Такой подход, во-первых, с пользователей класса снимает заботу создания делегатов, что требует некоторой квалификации, которой у пользователя может и не быть. Во-вторых, делегаты создаются динамически, в тот момент, когда они требуются. Это важно как при работе с функциями высших порядков, когда реализаций, например, подынтегральных функций, достаточно много, так и при работе с событиями класса, в основе которых лежат делегаты.

Рассмотрим пример, демонстрирующий и поясняющий эту возможность при работе с функциями высших порядков. Спроектируем два класса:

- класс объектов Person с полями: имя, идентификационный номер, зарплата. В этом классе определим различные реализации функции Compare, позволяющие сравнивать два объекта по имени, по номеру, по зарплате, по нескольким полям. Самое интересное, ради чего и строится данный пример: для каждой реализации Compare будет построена процедура-свойство, которая задает реализацию делегата, определенного в классе Persons;
- класс Persons будет играть роль контейнера объектов Person.

В этом классе определены операции над объектами. Среди операций нас, прежде всего, будет интересовать сортировка объектов, реализованная в виде функции высших порядков. Функциональный параметр будет задавать класс функций сравнения объектов, реализации которых находятся в классе Person. Делегат, определяющий класс функций сравнения, будет задан в классе Persons.

Приведем полный код приложения.

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

namespace delegates2
{

   public class Person
   {
       //конструкторы
       public Person() { name = ""; id = 0; salary = 0.0; }
       public Person(string name) { this.name = name; }
       public Person(string name, int id, double salary)
       { this.name = name; this.id = id; this.salary = salary; }
       public Person(Person pers)
       {
           this.name = pers.name; this.id = pers.id;
           this.salary = pers.salary;
       }
       //методы
       public void ToPerson(string mes)
       {
           this.message = mes;
           Console.WriteLine("{0}, {1}", name, message);
       }
       //свойства
       private string name;
       private int id;
       private double salary;
       private string message;
       //доступ к свойствам
       public string Name
       { get { return (name); }
           set { name = value; } }
       public double Salary
       { get { return (salary); }
           set { salary = value; } }
       public int Id
       { get { return (id); } set { id = value; } }

       private static int CompareName(Person obj1, Person obj2)
       {
           return (string.Compare(obj1.name, obj2.name));
       }
       private static int CompareId(Person obj1, Person obj2)
       {
           if (obj1.id > obj2.id) return (1);
           else return (-1);
       }
       private static int CompareSalary(Person obj1, Person obj2)
       {
           if (obj1.salary > obj2.salary) return (1);
           else if (obj1.salary < obj2.salary) return (-1);
           else return (0);
       }
       private static int CompareSalaryName(Person obj1, Person obj2)
```

```
        {
            if (obj1.salary > obj2.salary) return (1);
            else if (obj1.salary < obj2.salary) return (-1);
            else return (string.Compare(obj1.name, obj2.name));
        }
        // реализация делегата CompareItems из класса Persons как свойства
        public static Persons.CompareItems SortByName
        {
            get { return (new Persons.CompareItems(CompareName)); }
        }
        public static Persons.CompareItems SortById
        {
            get { return (new Persons.CompareItems(CompareId)); }
        }
        public static Persons.CompareItems SortBySalary
        {
            get { return (new Persons.CompareItems(CompareSalary)); }
        }
        public static Persons.CompareItems SortBySalaryName
        {
            get { return (new Persons.CompareItems(CompareSalaryName)); }
        }


    }//class Person

    public class Persons
    {//контейнер объектов Person
        //делегат
        public delegate int CompareItems(Person obj1, Person obj2);
        private int freeItem = 0;
        const int n = 100;
        private Person[] persons = new Person[n];

        public void AddPerson(Person pers)
        {
            if (freeItem < n)
            {
                Person p = new Person(pers);
                persons[freeItem++] = p;
            }
            else Console.WriteLine("Не могу добавить Person");
        }
        public void LoadPersons()
        {
            //реально загрузка должна идти из базы данных
            AddPerson(new Person("Петров", 123, 750.0));
            AddPerson(new Person("Петров", 128, 850.0));
            AddPerson(new Person("Сидоров", 223, 750.0));
            AddPerson(new Person("Орлов", 129, 800.0));
            AddPerson(new Person("Соколов", 133, 1750.0));
            AddPerson(new Person("Орлов", 119, 750.0));
            AddPerson(new Person("Орлов", 120, 750.0));
        }//LoadPersons
        public void PrintPersons()
        {
            for (int i = 0; i < freeItem; i++)
            {
                Console.WriteLine("{0,10} {1,5} {2,5}",
                persons[i].Name, persons[i].Id, persons[i].Salary);
            }
        }//PrintPersons
        // Определим метод сортировки записей с функциональным параметром, задающим
тот или иной способ сравнения элементов:
        //сортировка
        public void SimpleSortPerson(CompareItems compare)
        {
            Person temp = new Person();
```

```
            for (int i = 1; i < freeItem; i++)
                for (int j = freeItem - 1; j >= i; j--)
                    if (compare(persons[j], persons[j - 1]) == -1)
                    {
                        temp = persons[j - 1];
                        persons[j - 1] = persons[j];
                        persons[j] = temp;
                    }
        }//SimpleSortObject
    }//Persons

    class Program
    {

        static void Main()
        {
            Persons persons = new Persons();
            persons.LoadPersons();
            Console.WriteLine("Сортировка по имени: ");
            persons.SimpleSortPerson(Person.SortByName);
            persons.PrintPersons();
            Console.WriteLine("Сортировка по идентификатору: ");
            persons.SimpleSortPerson(Person.SortById);
            persons.PrintPersons();
            Console.WriteLine("Сортировка по зарплате: ");
            persons.SimpleSortPerson(Person.SortBySalary);
            persons.PrintPersons();
            Console.WriteLine("Сортировка по зарплате и имени: ");
            persons.SimpleSortPerson(Person.SortBySalaryName);
            persons.PrintPersons();
            Console.ReadLine();
        }
    }
}
```

Таким образом, реализация делегатов дает широкие возможности в интеллектуализации методов, расширяя понятия шаблонных и полиморфных методов в парадигме объектно-ориентированного программирования.

ЛИТЕРАТУРА

1. Раттц-мл. Дж. LINQ: Язык интегрированных запросов в C# 2008 для профессионалов. М., 2008.
2. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4. М., 2010.

REFERENCES

1. Ratts, Jr. J. LINQ: Language of Integrated Query in C# 2008 for Professionals. Moscow, 2008.
2. Troelsen A. C# 2010 Programming Language and .NET 4 Platform. Moscow, 2010.