Как и методы, делегаты также могут быть обобщенными. Ниже приведена общая

форма объявления обобщенного делегата.

delegate возврашаемый\_тип имя\_делегата<список\_параметров\_типа>(список\_аргументов);

Обратите внимание на расположение списка параметров типа. Он следует непосредственно

после имени делегата. Преимущество обобщенных делегатов заключается

в том, что их допускается определять в типизированной обобщенной форме, которую

можно затем согласовать с любым совместимым методом.

В приведенном ниже примере программы демонстрируется применение делегата

SomeOp с одним параметром типа Т. Этот делегат возвращает значение типа Т и принимает

аргумент типа Т.

(***glava18\_14***)

//define generalized delegate

delegate T SomeOp<T>(T v);

class GenDelegatDemo

{

//return result of sum argument

static int Sum(int v)

{

int result = 0;

for (int i = v; i > 0; i--)

result += i;

return result;

}

//return string, contains reversed value of argument

static string Reflect(string str)

{

string result = "";

foreach(char ch in str)

{

result = ch + result;

}

return result;

}

static void Main()

{

//define delegate type int

SomeOp<int> intDel = Sum;

Console.WriteLine(intDel(3));

//define delegate type string

SomeOp<string> strDel = Reflect;

Console.WriteLine(strDel("Hello"));

}

}

Рассмотрим эту программу более подробно. Прежде всего обратите внимание на

следующее объявление делегата SomeOp.

delegate T SomeOp<T>(T v);

Как видите, тип Т может служить в качестве возвращаемого типа, несмотря на то,

что параметр типа Т указывается после имени делегата SomeOp.

В силу присущей обобщениям типовой безопасности обобщенным делегатам нельзя

присваивать несовместимые методы. Так, следующая строка кода оказалась бы ошибочной

в рассматриваемой здесь программе.

SomeOp<int> intDel = Reflect; // Ошибка!

Ведь метод Reflect() принимает аргумент типа string и возвращает результат

типа string, а следовательно, он несовместим с целочисленным экземпляром делегата

SomeOp.

**Обобщенные интерфейсы**

Помимо обобщенных классов и методов, в C# допускаются обобщенные интерфейсы.

Такие интерфейсы указываются аналогично обобщенным классам. Ниже приведен

измененный вариант примера из главы 12, демонстрирующего интерфейс ISeries.

(Напомним, что ISeries является интерфейсом для класса, генерирующего последовательный

ряд числовых значений.) Тип данных, которым оперирует этот интерфейс,

теперь определяется параметром типа.

(***glava18\_15***)

public interface ISeries<T>

{

T GetNext(); //next number in order

void Reset(); //start count from the begining

void SetStart(T v); //set start value

}

//realize ISeries interface

class ByTwos<T> : ISeries<T>

{

T start;

T val;

//this delegate defines method form, called for generate

//another element in order of value

public delegate T IncByTwo(T v);

//this link to delegate, will be assigned method

//passed to constructor of class ByTwos

IncByTwo incr;

public ByTwos(IncByTwo incrMeth)

{

start = default(T);

val = default(T);

incr = incrMeth;

}

public T GetNext()

{

val = incr(val);

return val;

}

public void Reset()

{

val = start;

}

public void SetStart(T v)

{

start = v;

val = start;

}

}

class ThreeD

{

public int x, y, z;

public ThreeD(int a, int b, int c)

{

x = a;

y = b;

z = c;

}

}

class GenlntfDemo

{

//define method of increasing by two

//every next element int type

static int IntPlusTwo(int v)

{

return v + 2;

}

//define method increasing by two every

//value type double

static double DoublePlusTwo(double v)

{

return v + 2.0;

}

//define method increasing by two every

//next value ccordinates of ThreeD type object

static ThreeD ThreeDPlustTwo(ThreeD v)

{

if (v == null) return new ThreeD(0, 0, 0);

else return new ThreeD(v.x + 2, v.y + 2, v.z + 2);

}

static void Main()

{

//demonstrate generating

//order of values type int

ByTwos<int> intBT = new ByTwos<int>(IntPlusTwo);

for (int i = 0; i < 5; i++)

Console.Write(intBT.GetNext() + " ");

Console.WriteLine();

//order ob list value type diuble

ByTwos<double> dblBT =

new ByTwos<double>(DoublePlusTwo);

dblBT.SetStart(11.4);

for (int i = 0; i < 5; i++)

Console.Write(dblBT.GetNext() + " ");

Console.WriteLine();

//order list coodrinates of ThreeD

ByTwos<ThreeD> ThrDBT = new ByTwos<ThreeD>(ThreeDPlustTwo);

ThreeD coord;

for (int i = 0; i < 5; i++)

{

coord = ThrDBT.GetNext();

Console.Write(coord.x + "," + coord.y

+ "," + coord.z + " ");

}

Console.WriteLine();

}

}

В данном примере кода имеется ряд любопытных моментов. Прежде всего обратите

внимание на объявление интерфейса ISeries в следующей строке кода.

public interface ISeries<T>

Как упоминалось выше, для объявления обобщенного интерфейса используется

такой же синтаксис, что и для объявления обобщенного класса.

А теперь обратите внимание на следующее объявление класса ByTwos, реализующего

интерфейс Iseries.

class ByTwos<T> : ISeries<T>

Параметр типа Т указывается не только при объявлении класса ByTwos, но и при

объявлении интерфейса ISeries. И это очень важно. ***Ведь класс, реализующий обобщенный***

***вариант интерфейса, сам должен быть обобщенным.*** Так, приведенное ниже

объявление недопустимо, поскольку параметр типа Т не определен.

class ByTwos : ISeries<T> { // Неверно!

Аргумент типа, требующийся для интерфейса ISeries, должен быть передан классу

ByTwos. В противном случае интерфейс никак не сможет получить аргумент типа.

Далее переменные, хранящие текущее значение в последовательном ряду (val) и

его начальное значение (start), объявляются как объекты обобщенного типа Т. После

этого объявляется делегат IncByTwo. Этот делегат определяет форму метода, используемого

для увеличения на два значения, хранящегося в объекте типа Т. Для того

чтобы в классе ByTwos могли обрабатываться данные любого типа, необходимо каким-

то образом определить порядок увеличения на два значения каждого типа данных.

Для этого конструктору класса ByTwos передается ссылка на метод, выполняющий

увеличение на два. Эта ссылка хранится в переменной экземпляра делегата incr. Когда

требуется сгенерировать следующий элемент в последовательном ряду, этот метод

вызывается с помощью делегата incr.

А теперь обратите внимание на класс ThreeD. В этом классе инкапсулируются координаты

трехмерного пространства (X,Z,Y). Его назначение — продемонстрировать

обработку данных типа класса в классе ByTwos.

Далее в классе GenIntfDemo объявляются три метода увеличения на два для объектов

типа int, double и ThreeD. Все эти методы передаются конструктору класса

ByTwos при создании объектов соответствующих типов. Обратите особое внимание

на приведенный ниже метод ThreeDPlusTwo().

static ThreeD ThreeDPlustTwo(ThreeD v)

{

if (v == null) return new ThreeD(0, 0, 0);

else return new ThreeD(v.x + 2, v.y + 2, v.z + 2);

}

В этом методе сначала проверяется, содержит ли переменная экземпляра v пустое

значение (null). Если она содержит это значение, то метод возвращает новый объект

типа ThreeD со всеми обнуленными полями координат. Ведь дело в том, что переменной

v по умолчанию присваивается значение типа default(Т) в конструкторе класса

ByTwos. Это значение оказывается по умолчанию нулевым для типов значений и пустым

для типов ссылок на объекты. Поэтому если предварительно не был вызван метод

SetStart(), то перед первым увеличением на два переменная v будет содержать

пустое значение вместо ссылки на объект. Это означает, что для первого увеличения на

два требуется новый объект.

На параметр типа в обобщенном интерфейсе могут накладываться ограничения

таким же образом, как и в обобщенном классе. В качестве примера ниже приведен

вариант объявления интерфейса ISeries с ограничением на использование только

ссылочных типов.

public interface ISeries<T> where T : class {

Если реализуется именно такой вариант интерфейса ISeries, в реализующем

его классе следует указать то же самое ограничение на параметр типа Т, как показано

ниже.

class ByTwos<T> : ISeries<T> where T : class {

В силу ограничения ссылочного типа этот вариант интерфейса ISeries нельзя

применять к типам значений. Поэтому если реализовать его в рассматриваемом здесь

примере программы, то допустимым окажется только объявление ByTwos<ThreeD>,

но не объявления ByTwos<int> и ByTwos<double>.