Как и методы, конструкторы также могут перегружаться. Это дает возможность

конструировать объекты самыми разными способами. (***glava8\_11***)

class MyClass

{

public int x;

public MyClass()

{

Console.WriteLine("Inside MyCLass().");

x = 0;

}

public MyClass(int i)

{

Console.WriteLine("Inside MyCLass(int i).");

x = i;

}

public MyClass(double d)

{

Console.WriteLine("Inside MyCLass(double d).");

x = (int)d;

}

public MyClass(int i, int j)

{

Console.WriteLine("Inside MyCLass(int i, int j).");

x = i \* j;

}

}

class OverloadConsDemo

{

static void Main()

{

MyClass t1 = new MyClass();

MyClass t2 = new MyClass(88);

MyClass t3 = new MyClass(17.23);

MyClass t4 = new MyClass(2, 4);

Console.WriteLine("t1.x: " + t1.x);

Console.WriteLine("t2.x: " + t2.x);

Console.WriteLine("t3.x: " + t3.x);

Console.WriteLine("t4.x: " + t4.x);

}

}

В данном примере конструктор MyClass() перегружается четыре раза, всякий раз

конструируя объект по-разному. Подходящий конструктор вызывается каждый раз,

исходя из аргументов, указываемых при выполнении оператора new. Перегрузка конструктора

класса предоставляет пользователю этого класса дополнительные преимущества

в конструировании объектов.

Одна из самых распространенных причин для перегрузки конструкторов заключается

в необходимости предоставить возможность одним объектам инициализировать

другие. В качестве примера ниже приведен усовершенствованный вариант разработанного

ранее класса Stack, позволяющий конструировать один стек из другого. (***glava8\_11.1***)

Короче говоря конструктор копирования. В класс Stack из ***glava8\_2\_stack*** добавлен конструктор перегрузка:

public Stack(Stack ob)

{

stck = new char[ob.stck.Length];

//copy elements

for (int i = 0; i < ob.tos; i++)

stck[i] = ob.stck[i];

tos = ob.tos;

}

**Вызов перегружаемого конструктора с помощью ключевого слова this**

Когда приходится работать с перегружаемыми конструкторами, то иногда очень полезно

предоставить возможность одному конструктору вызывать другой. В С# это дается

с помощью ключевого слова this. Ниже приведена общая форма такого вызова.

**имя\_конструктора(список\_параметров1) : this(список\_параметров2) {**

**// ... Тело конструктора, которое может быть пустым.**

**}**

В исходном конструкторе сначала выполняется перегружаемый конструктор, список

параметров которого соответствует критерию *список\_параметров2,* а затем все

остальные операторы, если таковые имеются в исходном конструкторе. Ниже приведен

соответствующий пример. (***glava8\_11.2***)

class XYCoord

{

public int x, y;

public XYCoord() : this(0, 0)

{

Console.WriteLine("In XYCoord().");

}

public XYCoord(XYCoord obj) : this(obj.x, obj.y)

{

Console.WriteLine("In XYCoord(XYCoord obj).");

}

public XYCoord(int i, int j)

{

Console.WriteLine("In XYCoord(int i, int j).");

x = i;

y = j;

}

}

class OverloadConsDemo

{

static void Main()

{

XYCoord t1 = new XYCoord();

XYCoord t2 = new XYCoord(8, 9);

XYCoord t3 = new XYCoord(t2);

Console.WriteLine("t1.x, t1.y: " + t1.x + " " + t1.y);

Console.WriteLine("t2.x, t2.y: " + t2.x + " " + t2.y);

Console.WriteLine("t3.x, t3.y: " + t3.x + " " + t3.y); }}

Код в приведенном выше примере работает следующим образом. Единственным

конструктором, фактически инициализирующим поля х и у в классе XYCoord, является

конструктор XYCoord(int, int). А два других конструктора просто вызывают

этот конструктор с помощью ключевого слова this. Например, когда создается объект

t1, то вызывается его конструктор XYCoord(), что приводит к вызову this(0, 0),

который в данном случае преобразуется в вызов конструктора XYCoord(0, 0). То же

самое происходит и при создании объекта t2.

Вызывать перегружаемый конструктор с помощью ключевого слова this полезно,

в частности, потому, что он позволяет исключить ненужное дублирование кода.

В приведенном выше примере нет никакой необходимости дублировать во всех трех

конструкторах одну и ту же последовательность инициализации, и благодаря применению

ключевого слова this такое дублирование исключается. Другое преимущество

организации подобного вызова перезагружаемого конструктора заключается в

возможности создавать конструкторы с задаваемыми "по умолчанию" аргументами,

когда эти аргументы не указаны явно. Ниже приведен пример создания еще одного

конструктора XYCoord.

***public XYCoord(int х) : this(х, х) { }***

По умолчанию в этом конструкторе для координаты у автоматически устанавливается

то же значение, что и для координаты у. Конечно, пользоваться такими конструкциями

с задаваемыми *"*по умолчанию*"* аргументами следует благоразумно и

осторожно, чтобы не ввести в заблуждение пользователей классов.