***Конструктор***инициализирует объект при его создании. У конструктора такое же

имя, как и у его класса, а с точки зрения синтаксиса он подобен методу. Но у конструкторов

нет возвращаемого типа, указываемого явно.

Как правило, конструктор используется для задания первоначальных значений переменных

экземпляра, определенных в классе, или же для выполнения любых других

установочных процедур, которые требуются для создания полностью сформированного

объекта. Кроме того, *доступ* обычно представляет собой модификатор доступа

типа public, поскольку конструкторы зачастую вызываются в классе. А *список\_па-*

*раметров* может быть, как пустым, так и состоящим из одного иди более указываемых

параметров.

У всех классов имеются конструкторы, независимо от того, определите вы их или

нет, поскольку в C# автоматически предоставляется конструктор, используемый по

умолчанию и инициализирующий все переменные экземпляра их значениями по

умолчанию. Для большинства типов данных значением по умолчанию является нулевое,

для типа bool — значение false, а для ссылочных типов — пустое значение.

Но как только вы определите свой собственный конструктор, то конструктор по умолчанию

больше не используется.

class MyClass

{

public int x;

public MyClass() //constructor

{

x = 10;

}

}

**Параметризированные конструкторы**

Зачастую конструктор, должен принимать один или несколько параметров. В конструктор параметры вводятся таким же образом, как и в метод. Для этого достаточно объявить их в скобках после имени конструктора. (glava6\_7)

class MyClass

{

public int x;

public MyClass(int i)//constructor w parameter

{

x = i;

}

}

class Building

{

public int Floors;

public int Area;

public int Occupants;

//constructor w parameters

public Building(int f, int a, int o)

{

Floors = f;

Area = a;

Occupants = o;

}

…

}

**Деструкторы**

В языке C# имеется возможность определить метод, который будет вызываться непосредственно

перед окончательным уничтожением объекта системой "сборки мусора".

Такой метод называется *деструктором* и может использоваться в ряде особых

случаев, чтобы гарантировать четкое окончание срока действия объекта. Например,

деструктор может быть использован для гарантированного освобождения системного

ресурса, задействованного освобождаемым объектом. Следует, однако, сразу же подчеркнуть,

что деструкторы — весьма специфические средства, применяемые только

в редких, особых случаях. И, как правило, они не нужны. Но здесь они рассматриваются

вкратце ради полноты представления о возможностях языка С#.

Ниже приведена общая форма деструктора:

***~имя\_класса() {***

***// код деструктора***

***}***

class Destruct

{

public int x;

public Destruct(int i)

{

x = i;

}

//calls to delete

~Destruct()

{

Console.WriteLine("Destroy " + x);

}

public void Generator(int i)

{

Destruct o = new Destruct(i);

}

}

В разные моменты этого процесса происходит "сборка мусора".

Насколько часто она происходит — зависит от нескольких факторов, в том числе

от первоначального объема свободной памяти, типа используемой операционной системы

и т.д.