Міністерство освіти і науки України
Криворізький коледж Національного авіаційного університету

Курс лекцій

з предмету

“Об’єктно-орієнтоване програмування”

Спеціальність 5.050103

м. Кривий Ріг

Лекция №4

Тема: Классы в языке C#

Вопросы, рассматриваемые на лекции

- Понятие класса
- Объявление класса в C# (C Sharp)
- Члены класса
- Экземпляры класса
- Методы
- Конструкторы

Понятие класса

Класс **- это определяемый пользователем тип, который содержит данные(константы и переменные), а также операции(функции-члены или методы), выполняемые над ними.**

Хранение в одной структуре и данных, и методов называется *инкапсуляцией*. Инкапсуляция позволяет в максимальной степени изолировать объект от внешнего окружения. Она существенно повышает надежность разрабатываемых программ, т.к. локализованные в объекте функции обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются.

Все классы .NET имеют общего предка — класс object, и организованы в единую иерархическую структуру.

Классы логически сгруппированы в пространства имен, которые служат для упорядочивания имен классов и предотвращения конфликтов имен: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными.

Любая программа использует пространство имен System.

Понятие объекта

Объекты — это *экземпляры* класса.

- В реальном мире каждый предмет или процесс обладает набором статических и динамических характеристик (свойствами и поведением). *Поведение объекта* зависит от его *состояния* и *внешних воздействий*.
- Понятие объекта в программе совпадает с обыденным смыслом этого слова: *объект представляется как совокупность данных, характеризующих его состояние, и функций их обработки, моделирующих его поведение*. Вызов функции на выполнение часто называют *посылкой сообщения* объекту.
- При создании объектно-ориентированной программы предметная область представляется в виде совокупности объектов. Выполнение программы состоит в том, что объекты обмениваются сообщениями.

Абстрагирование и инкапсуляция

- При представлении реального объекта с помощью программного необходимо выделить в первом его существенные особенности и игнорировать несущественные. Это называется *абстрагированием*.
- Таким образом, программный объект — это абстракция.
- Детали реализации объекта скрыты, он используется через его *интерфейс* — совокупность правил доступа.
- Скрытие деталей реализации называется *инкапсуляцией.* Это позволяет представить программу в укрупненном виде — на уровне объектов и их взаимосвязей, а следовательно, управлять большим объемом информации.
- *Итак,* объект — это инкапсулированная абстракция с четко определенным интерфейсом.
- Хранение в одной структуре и данных, и методов называется *инкапсуляцией*. Инкапсуляция позволяет в максимальной степени изолировать объект от внешнего окружения. Она существенно повышает надежность разрабатываемых программ, т.к. локализованные в объекте функции обмениваются с программой сравнительно небольшими объемами данных, причем количество и тип этих данных обычно тщательно контролируются.

Наследование

- Важное значение имеет возможность многократного использования кода. Для объекта можно определить наследников, корректирующих или дополняющих его поведение.
- *Наследование* применяется для:
  - исключения из программы повторяющихся фрагментов кода;
  - упрощения модификации программы;
  - упрощения создания новых программ на основе существующих.
- Благодаря наследованию появляется возможность использовать объекты, исходный код которых недоступен, но в которые требуется внести изменения.
- Наследование позволяет создавать *иерархии объектов*. Иерархия представляется в виде дерева, в котором более общие объекты располагаются ближе к корню, а более специализированные — на ветвях и листьях.

Полиморфизм

- ООП позволяет писать гибкие, расширяемые и читабельные программы.
- Во многом это обеспечивается благодаря полиморфизму, под которым понимается возможность во время выполнения программы с помощью одного и того же имени выполнять разные действия или обращаться к объектам разного типа.
- Чаще всего понятие полиморфизма связывают с механизмом виртуальных методов.

Объявление класса

Форма объявления класса:

[атрибуты] [модификаторы-прав-доступа] class идентификатор [:базовый-класс]

{

// Объявление переменных экземпляров.

доступ тип переменная1;

доступ тип переменная2;

//. . .

доступ тип переменнаяN;

// Объявление методов

доступ тип_возврата метод1 (параметры) { тело метода}

доступ тип_возврата метод2{параметры) { тело метода}

доступ тип_возврата методN {параметры) { тело метода}

}

Спецификаторы класса

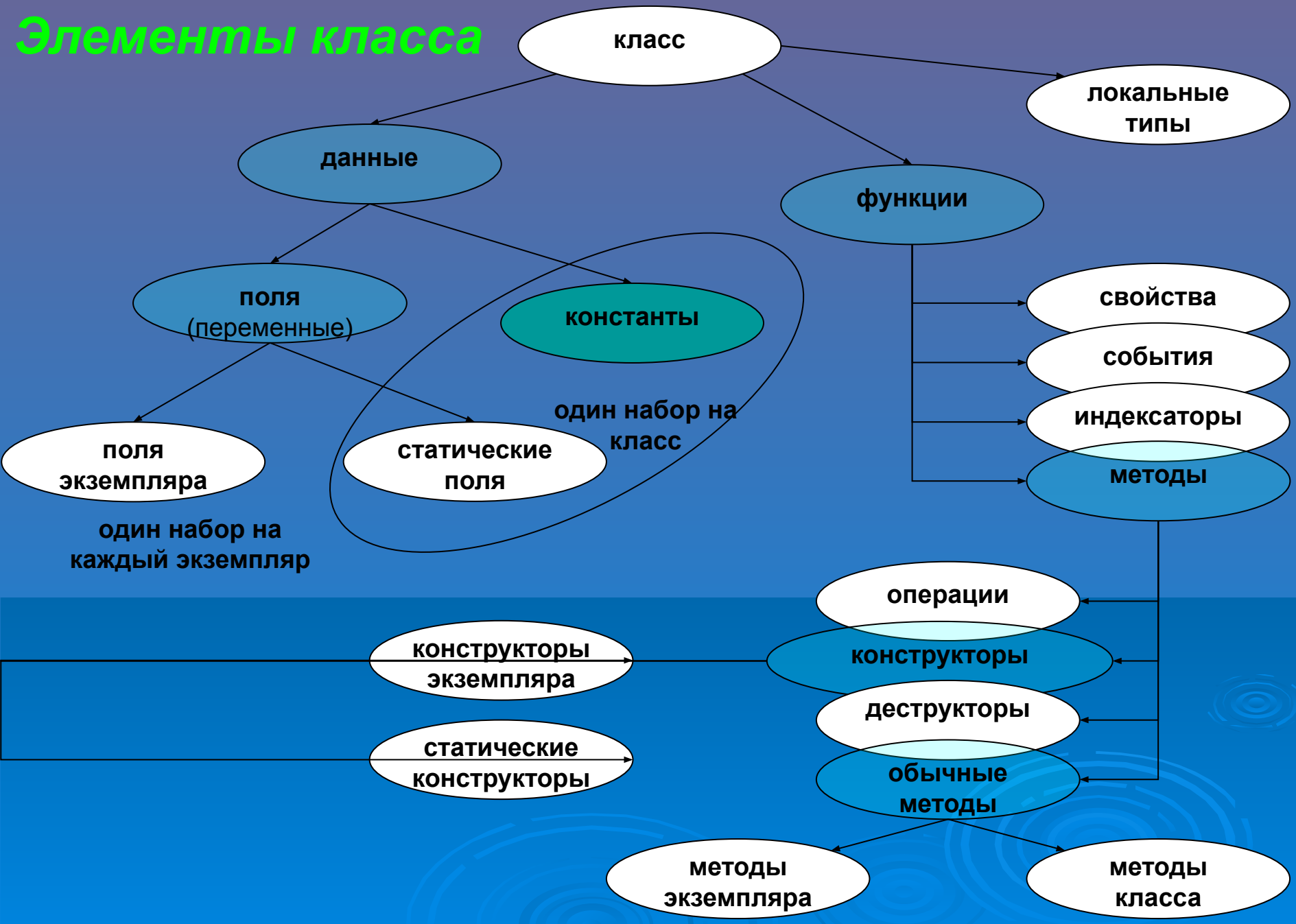
Спецификатор	Описание
new	(для вложенных классов). Задает новое описание класса взамен унаследованного от предка. Применяется в иерархиях
public	Доступ не ограничен
protected	Используется для вложенных классов. Доступ только из элементов данного и производных классов
internal	Доступ только из данной программы (сборки)
protected internal	Доступ только из данного и производных классов или из данной программы (сборки)
private	(для вложенных классов). Доступ только из элементов класса, внутри которого описан данный класс
abstract	Абстрактный класс. Применяется в иерархиях
sealed	Бесплодный класс. Применяется в иерархиях
static	Статический класс.

Члены класса

Классы состоят из членов и могут включать следующие объекты:

- константы,
- события,
- поля,
- финализаторы,
- индексаторы,
- конструкторы экземпляров,
- методы,
- объявления вложенных типов,
- операторы,
- свойства,
- статические конструкторы.

Элементы класса



Члены класса

- Константа — это член класса, который, как и предлагает название, используется для представления постоянного значения. Такое постоянное значение можно либо объявить, либо вычислить во время компиляции. Константы могут зависеть от других констант той же программы
- Полем называется член, используемый для представления переменной, связанной с объектом или классом.

  Формат объявления переменной экземпляра такой:

  доступ тип имя_переменной;

  Поля представляют собой любые переменные, связанные с классом. Если определить переменную на уровне класса, то на самом деле это будет поле класса.

  Доступ к полям осуществляется с помощью синтаксиса

  объект.имя_поля
- События являются членами класса, которые позволяют объекту уведомлять вызывающего о том, что произошли некоторые программные изменения, например, изменилось поле или свойство класса либо имела место некоторая форма взаимодействия с пользователем. Клиент может содержать код, известный как обработчик ошибок, который реагирует на событие.

Члены класса

- Методом называют член, реализующий действие, которое может выполняться объектом или классом.

Методы имеют

- список формальных параметров (который может быть пустым),
- возвращаемое значение (если только возвращаемым типом не является void).

Методы могут быть статическими и нестатическими:

- ***статические*** методы доступны через класс (они предоставляют общую функциональность, не требующую создания экземпляра класса (подобно методу Console. WriteLine ()),
- ***нестатические*** методы (методы экземпляра) доступны через экземпляры класса.

- Свойством называют член, обеспечивающий доступ к конкретной характеристике объекта или класса (например, такой как длина строки). Свойства в определенном смысле аналогичны полям, но, в отличие от полей, не указывают на области в памяти. Свойства имеют аксессоры (средства доступа), указывающие операторы, которые должны выполняться при доступе к свойству для чтения или записи.
- Конструкторы являются функциями, которые вызываются при создании экземпляра объекта. Они обязаны иметь имя, совпадающее с именем класса, и не должны возвращать никаких значений. Конструкторы служат для установки значений полей при создании экземпляра объекта.

Члены класса

- Статические конструкторы

  Статический конструктор является членом, выполняющим действия, необходимые для инициализации класса. Статические конструкторы не могут использовать параметры, модификаторы доступности.

  Статические конструкторы не могут быть вызваны явно, а вызываются автоматически.

- Деструкторы аналогичны конструкторам, но вызываются при уничтожении объекта. Они имеют то же имя, что и класс, но с предшествующим значком тильды (~). Так как сборкой мусора занимается CLR, нельзя сказать точно, когда будет вызван деструктор. В C# деструкторы используются менее часто, чем в C++.

- Операция представляет собой член, определяющий значение выражения, которое можно применять к экземплярам данного класса. Можно определить следующие три вида операций:

  - бинарные операции,
  - операции преобразования,
  - унарные операции.

  Классы могут также содержать определения операций, поэтому можно описать свои собственные операции или указать, как существующие операции будут работать с классом.

Члены класса

- Индексаторы позволяют индексировать объекты точно так же, как массивы и коллекции.
- Финализатор — это член, выполняющий действия, необходимые для завершения использования экземпляра класса. Соответствующие действия выполняются тогда, когда класс больше не нужен.

  Финализаторы не могут использовать параметры, модификаторы доступности.

  Финализаторы не могут быть вызваны явно. Финализатор любого экземпляра вызывается автоматически в процессе сборки мусора, выполняемого средствами .NET Framework.

Создание экземпляра класса

Чтобы реально создать объект класса, используется инструкция:
ИмяКласса Идентификатор= new ИмяКласса(<СписокПараметров>);
или

ИмяКласса Идентификатор;
Идентификатор = new ИмяКласса(<СписокПараметров>);

Оператор new динамически (т.е. во время выполнения программы) выделяет память для объекта и возвращает ссылку на него. Эта ссылка (сохраненная в конкретной переменной) служит адресом объекта в памяти, выделенной для него оператором new. Таким образом, в C# для всех объектов классов должна динамически выделяться память.

После выполнения этой инструкции Идентификатор станет экземпляром класса ИмяКласса, т.е. обретет "физическую" реальность. Идентификатор не определяет объект, а может лишь ссылаться на него. При каждом создании экземпляра класса создается объект, который содержит собственную копию каждой переменной экземпляра, определенной этим классом.

Объявление методов

Формат записи метода:

МодификаторДоступа ТипВозврата имя (СписокФормальныхПараметров)

{ тело метода }

С помощью элемента *тип_возврата* указывается тип значения, возвращаемого методом. Это может быть любой допустимый тип, включая типы классов, создаваемые программистом. Если метод не возвращает никакого значения, необходимо указать тип void. Имя метода, как нетрудно догадаться, задается элементом *имя*. В качестве имени метода можно использовать любой допустимый идентификатор, отличный от тех, которые уже использованы для других элементов программы в пределах текущей области видимости. Элемент *список_параметров* представляет собой последовательность пар (состоящих из типа данных и идентификатора), разделенных запятыми. Параметры — это переменные, которые получают значения аргументов, передаваемых методу при вызове.

Если же у метода отсутствуют аргументы, все равно необходимо указать пустую пару скобок () после имени метода.

При отсутствии модификатора доступа метод является закрытым (private) по умолчанию.

Объявление методов

Метод может принимать один или несколько из следующих модификаторов

Модификатор	Описание
new	Метод скрывает унаследованный метод с такой же сигнатурой
public	Метод доступен отовсюду, в том числе извне класса.
protected	Метод доступен изнутри класса, к которому он принадлежит, или из типа, производного от данного класса.
internal	Метод доступен только из той же программы.
private	Метод доступен только из класса, к которому он принадлежит.
static	Метод не предназначен для работы с определенным экземпляром класса.
virtual	Метод может быть переопределен в производном классе.
abstract	Виртуальный метод, который определяет сигнатуру метода, но не содержит реализацию.
override	Метод переопределяет унаследованный виртуальный или абстрактный метод.
sealed	Метод перекрывает унаследованный виртуальный метод, но не может быть перекрыт классами, которые являются производными от данного. Должен использоваться совместно с override.
extern	Метод реализован вне программы на другом языке.

Возврат значения метода

Для возврата значения метода используется оператор

return Выражение;

В качестве Выражения может быть явно заданная константа указанного в описании метода типа возврата (например, для целочисленных типов 0 или 1), переменная того же типа или выражение. Допускается использовать явное преобразование типа.

Немедленное завершение void-метода можно организовать с помощью следующей формы инструкции return:

return;

Вызов метода

Для вызова (активизации) метода необходимо указать

Имя объекта. Метод (СписокФактическихАргументов)

Если функция возвращает результат, его можно сохранить в переменной, использовать в качестве аргумента другой функции или отбросить.

При вызове статического (static) метода необходимо использовать ИМЯ типа класса этого метода, а не имя экземпляра класса:

Имя класса. Метод (СписокФактическихАргументов)

Аргументы методов

Значение, передаваемое методу, называется *аргументом.*

Переменная внутри метода, которая принимает значение аргумента, называется *параметром.*

Между списком формальных и списком фактических аргументов должно выполняться определенное соответствие по числу, порядку следования, типу и статусу аргументов.

Аргументы могут быть переданы в методы по ссылке или по значению.

Переменная, передаваемая в метод *по ссылке*, подвергается любым изменениям, которые производит с ней вызываемый метод, в то время как *переменная, передаваемая по значению*, не меняет свое значение в результате изменений, сделанных внутри метода. Это происходит потому, что в первом случае метод получает ссылку на саму переменную, а во втором случае он получает ее копию.

Аргументы методов

Синтаксис объявления формального аргумента:

[ref | out | params | in] тип_аргумента имя_аргумента

Если параметр никак не помечен, то по умолчанию считается, что этот параметр — входящий (для передачи методу), передаваемый как значение. Вместо пропуска модификатора можно использовать модификатор in —результат будет тем же самым.

Если в метод передается параметр, а аргумент метода помечен как ref, то любое изменение переменной, сделанное методом, вызовет соответствующее изменение оригинальной переменной (переменные по значению передаются по ссылке). Ключевое слово ref должно указываться и при вызове метода.

Модификатор out подобен модификатору ref за одним исключением: его можно использовать только для передачи значения из метода. Совсем не обязательно (и даже не нужно) присваивать переменной, используемой в качестве out-параметра, начальное значение до вызова метода. Более того, предполагается, что out-параметр всегда "поступает" в метод без начального значения, но метод (до своего завершения) обязательно *должен* присвоить этому параметру значение.

Аргументы методов

Несмотря на фиксированное число формальных аргументов, есть возможность при вызове метода передавать ему произвольное число фактических аргументов. Для реализации этой возможности в списке формальных аргументов необходимо задать ключевое слово params. Оно задается один раз и указывается только для последнего аргумента списка, объявляемого как массив произвольного типа.

Аргументы методов

Пример:

```
 using System;
class XY
{
    public void swap(ref int x, ref int y)
    {
        int k;    k = x;     x = y;      y = k;
    }
}
class Program
{
    static void Main(string[] args)
    {
        XY z = new XY();
        int x = 5, y = 6;
        Console.WriteLine("x=" + x + "y=" + y);
        z.swap(ref x, ref y);
        Console.WriteLine("x=" + x + "y=" + y);
    }
}
```

Примеры

Пример:

```
 using System;
class XY
{
    public void swap(ref int x, ref int y, out int add, out int sub)
    {
        add = x + y;
        sub = x - y;
    }
}
class Program
{
    static void Main(string[] args)
    {
        XY z = new XY();
        int x = 5, y = 6,s,r;
        z.swap(ref x, ref y,out s,out r);
        Console.WriteLine("s=" + s + "  r=" + r);
    }
}
```

Примеры

Пример:

```
 using System;
class XY
{
    public int sum(params int[] abc)
    {
        int s=0;
        for (int i = 0; i < abc.Length; i++)
            s += abc[i];
        return s;
    }
}
class Program
{
    static void Main(string[] args)
    {
        XY z = new XY();
        int x = 5, y = 6, m = 4, k = 7;
        Console.WriteLine("s=" + z.sum(x, y));
        Console.WriteLine("s=" + z.sum(x, y, m));
        Console.WriteLine("s=" + z.sum(x, y, m, k));
    }
}
```

Возвращаемые значения метода

Метод может возвращать данные любого типа, в том числе классового

Пример:

```
 using System;
class XY
{    public int x, y;  }
class op
{
    public XY init()
    {
        XY s = new XY();     s.x = 4;      s.y = 5;
        return s;
    }
}
class Program
{
    static void Main(string[] args)
    {
        op z = new op();
        Console.WriteLine("x=" + z.init().x);
        Console.WriteLine("y=" + z.init().y);
    }
}
```

Возвращаемые значения метода

Метод может возвратить массив

Пример:

```
 using System;
class Zada4a
{
   public int [ ] mas(int n)
    {
        int [ ] m = new int[n];
        Random rnd = new Random();
        for (int i = 0; i < n; i++) m[i] = rnd.Next(1, 100);
        return m;
    }
    public static int Main()
    {
        const int n=5;
        Zada4a z = new Zada4a();
        int[] m = z.mas(n);
        for (int i = 0; i < n; i++)
            Console.WriteLine("m[{0}]={1}", i, m[i]);
        return 0;
    }
}
```

Перегрузка методов

В C# два или больше методов внутри одного класса могут иметь одинаковое имя, но при условии, что их параметры будут различными. Методам для перегрузки недостаточно отличаться лишь типами возвращаемых значений. Они должны отличаться типами или числом параметров.

Это называется *перегрузкой* методов (method overloading), а методы, которые в ней задействованы, называют *перегруженными* (overloaded). Перегрузка методов — один из способов реализации полиморфизма в C#.

Пример:

```
public void f(byte x)
   {
      Console.WriteLine("Внутри метода f(byte): " + x);
   }
   public void f(int x)
   {
      Console.WriteLine("Внутри метода f(int): " + x);
   }
   public void f(double x, double y)
   {
      Console.WriteLine("Внутри метода f(double): " + x+”,”+y);
   }
```

Конструкторы

Конструктор инициализирует объект при его создании. Он имеет такое же имя, что и сам класс, а синтаксически подобен методу. Однако в определении конструкторов не указывается тип возвращаемого значения. Формат записи конструктора такой:

```
доступ имя_класса()
{
// тело конструктора
}
```

Обычно конструктор используется, чтобы придать переменным экземпляра, определенным в классе, начальные значения или выполнить исходные действия, необходимые для создания полностью сформированного объекта. Кроме того, обычно в качестве элемента *доступ* используется модификатор доступа public, поскольку конструкторы, как правило, вызываются вне их класса.

Все классы имеют конструкторы независимо от того, определите вы их или нет.

Пример:

```
class XY
{
  int x, y;
  public XY()  {     x = 4; y = 5;  } // конструктор без параметров
}
```

Параметризованные конструкторы

Конструкторы ***могут принимать один или несколько параметров. Параметры вносятся в конструктор точно так же, как в метод. Конструкторы также можно перегружать.***

Пример:

```
using System;
class XY
{ public int x, y;
    public XY ()
    {       x = 4; y = 5;    }
    public XY(int x, int y)
    {       this.x = x; this.y = y;    }
}
class Program
{
    static void Main(string[] args)
    {
        XY z = new XY();
        XY z1 = new XY(3,4);
        Console.WriteLine("x=" + z.x+ " y="+z.y);
        Console.WriteLine("x1=" + z1.x + " y1=" + z1.y);
    }
}
```

Статические конструкторы

Статический конструктор обычно используется для инициализации атрибутов, которые применяются к классу в целом, а не к конкретному его экземпляру. Таким образом, статический конструктор служит для инициализации аспектов класса до создания объектов этого класса.

Использование оператора new

Формат записи:

переменная_типа_класса = new имя_класса ( ) ;

Здесь элемент *переменная_типа_класса* означает имя создаваемой переменной типа класса. Под элементом *имя_класса* понимается имя реализуемого в объекте класса. Имя класса вместе со следующей за ним парой круглых скобок — это ни что иное, как *конструктор* реализуемого класса с параметрами или без них. Если в классе конструктор не определен явным образом, оператор new будет использовать конструктор по умолчанию, который предоставляется средствами языка C#. Таким образом, оператор new можно использовать для создания объекта любого "классового" типа.

Статические конструкторы

Пример:

```
using System;
class Cons
{   public static int alpha;
    public int beta;
    static Cons() // Статический конструктор
    {   alpha = 99; Console.WriteLine("Внутри статического конструктора.");   }
    public Cons() // Конструктор экземпляра
    {   beta = 100;    Console.WriteLine("Внутри конструктора экземпляра.");   }
}
class ConsDemo
{
    public static void Main()
    {   Cons ob = new Cons();
        Console.WriteLine("Cons.alpha: " + Cons.alpha);
        Console.WriteLine("ob.beta: " + ob.beta);   }
}
```

Результат выполнения программы:

```
Внутри статического конструктора.
Внутри конструктора экземпляра.
Cons.alpha: 99
ob.beta: 100
```

Контрольные вопросы по теме

1. Почему язык C/C++ называют языком системного программирования?
2. Из чего состоит алфавит языка C/C++?
3. Какие управляющие последовательности Вы запомнили?
4. Что такое лексема?
5. Перечислите шесть классов лексем.
6. Дайте определение комментария.
7. Перечислите виды комментариев.
8. Приведите примеры записи комментариев.
9. Дайте определение идентификатору.
10. Перечислите требования к идентификатору.
11. Приведите примеры идентификаторов.
12. Дайте определение ключевым словам.
13. Приведите пример ключевых слов.

Контрольные вопросы по теме

14. Что относится к элементам данных?
15. Дайте определение константе.
16. Какие типы констант Вы знаете?
17. Напишите форму записи типизированных констант.
18. Приведите примеры типизированных констант.
19. Напишите форму записи нетипизированных констант.
20. Приведите примеры нетипизированных констант.
21. Дайте определение переменной.
22. Какие типы переменных Вы знаете?
23. Напишите форму записи объявления переменных без явной инициализации.
24. Приведите примеры объявления переменных без явной инициализации.
25. Напишите форму записи объявления переменных с явной инициализацией.
26. Приведите примеры объявления переменных с явной инициализацией.

Контрольные задания по теме

1. Какая запись верна? Почему?

   а) m a x = 0; б) max = 0;

2. Верна ли запись? Если да, то почему?

   c = a + 3 ;

3. В какой из строк, где вычисляется сумма 2-х чисел, комментарий используется верно?

   а) //c=a+b; сумма чисел

   б) c=a+b; //сумма чисел

4. Где идентификаторы записаны верно? Почему?

   а) макс; б) 243c; в) s_pr; г) _k_1;

5. Являются ли все записи одним и тем же идентификатором? Почему?

   s_pr; S_pr; S_Pr; S_PR;

6. Верна ли запись?

   int a, b=3; int c=a+b;

7. Верна ли запись?

   const int a=2, b=3; int A=a+b;

8. Верна ли запись?

   const c=7;

Письменные контрольные задания по теме

Вариант №1	Вариант №2	Вариант №3
Понятие лексемы. Перечислить классы лексем.	Что такое ключевые слова? Примеры	Понятие идентификатора. Требования к нему. Примеры.
Форма записи однострочного комментария. Пример	Форма записи многострочного комментария. Пример	Приведите примеры всех возможных комментариев
Понятие константы. Формы записи. Примеры	Требования к переменным и константам. Примеры. Перечислите все формы записи переменных и констант.	Понятие переменной. Формы записи. Примеры