# Понятие класса. Структуры. Перечисления.

1. Классы.
2. Структуры.
3. Перечисления.
4. Упаковка и распаковка

## Классы.

Объектно-ориентированное программирование построено на классах. Любую программную систему, построенную в объектном стиле, можно рассматривать как совокупность классов, возможно объединенных в пространства имен, проекты, решения, как это делается при программировании в Visual Studio .Net.

У класса две различные роли: модуля и типа данных.

**Класс – это модуль**, архитектурная единица построения программной системы. Модульность построения – основное свойство программных систем. В ООП программная система, строящаяся по модульному принципу, состоит из классов, являющихся основным видом модуля. Модуль может не представлять собой содержательную единицу, его размер и содержание определяется архитектурными соображениями, а не семантическими. Теоретически можно построить монолитную систему, состоящую из одного модуля, решающую ту же задачу, что и система, состоящая из многих модулей. Практически большую систему, создаваемую коллективом разработчиком, без разделения системы на модули построить не удается. Модульность построения – основное средство борьбы со сложностью системы.

**Класс – это тип данных**, задающий реализацию некоторой абстракции данных, характерной для проблемной области, в интересах которой создается программная система. Т.е. класс является обобщенным понятием, определяющим характеристики и поведение некоторого множества конкретных объектов этого класса, называемых *экземплярами класса.* В общем случае класс содержит *данные,* задающие свойства объектов класса, и *функции,* определяющие их поведение. В класс также могут добавляться *события,* на которые может реагировать объект класса (приложения, построенные на основе событийно-управляемой модели, например, при программировании для Windows.).

Примеры: Студенты, Преподаватели, Аудитории, Дисциплины.

**Пространство имен.**

Все классы библиотеки .NET, а также все классы, которые создает программист в среде .NET, имеют одного общего предка — класс object и организованы в единую иерархическую структуру. Внутри нее классы логически сгруппированы в так называемые пространства имен, которые служат для упорядочивания имен классов и предотвращения конфликтов имен: в разных пространствах имена могут совпадать. Пространства имен могут быть вложенными, их идея аналогична иерархической структуре каталогов на компьютере.

Любая программа, создаваемая в .NET, использует пространство имен System. В нем определены классы, которые обеспечивают базовую функциональность, например, поддерживают выполнение математических операций, управление памятью и ввод-вывод.

Обычно в одно пространство имен объединяют взаимосвязанные классы.

**Упрощенное описание класса.**

class имя\_класса тело-класса

Данные, содержащиеся в классе, могут быть *переменными (поля)* или *константами.*

Некоторые спецификаторы полей и констант класса.

* public - доступ к элементу не ограничен.
* protected - доступ только из данного и производных классов.
* private - доступ только из данного класса.
* static - одно поле для всех экземпляров класса

По умолчанию элементы класса считаются закрытыми (private). Все методы класса имеют непосредственный доступ к его закрытым полям.

**Методы.**

*Метод —* это функциональный элемент класса, который реализует вычисления или другие действия, выполняемые классом или экземпляром. Методы определяют поведение класса.

Метод представляет собой законченный фрагмент кода, к которому можно обратиться по имени. Он описывается один раз, а вызываться может столько раз, сколько необходимо. Один и тот же метод может обрабатывать различные данные, переданные ему в качестве аргументов.

**Синтаксис метода:**

[ спецификаторы ] тип имя\_метода ( [ параметры ] ) тело метода

Чаше всего для методов задается спецификатор доступа public, ведь методы составляют интерфейс класса - то, с чем работает пользователь, поэтому они должны быть доступны .

Статические (static) методы, или методы класса, можно вызывать, не создавая экземпляр объекта. Именно таким образом используется метод Main.

Для каждого объекта при его создании в памяти выделяется отдельная область, в которой хранятся его данные. Объекты создаются явным или неявным образом, то есть либо программистом, либо системой. Программист создает экземпляр класса с помощью операции new. При этом для создания объекта и инициализации его данных вызывается специальный метод – конструктор. Имя конструктора совпадает с именем класса.

**Свойства конструкторов**:

1. Конструктор *не возвращает значение,* даже типа void.
2. Класс может иметь *несколько конструкторов* с разными параметрами для разных видов инициализации.
3. Если программист не указал ни одного конструктора или какие-то поля не были инициализированы, полям значимых типов присваивается нуль, полям ссылочных типов – значение null .

Конструктор, вызываемый без параметров, называется *конструктором по умолчанию.*

**Пример 1.**

class Person

{

public string name;

public int age;

public Person(string strName, int iAge)

{

name = strName; age = iAge;

}

public Person()//конструктор по умолчанию

{

name = "NoName";

age = 20;

}

public override string ToString()

{

return name + "," + age.ToString();

}

public void Show()

{

Console.WriteLine(ToString());//this.ToString()

}

}

class Program

{

static void Main(string[] args)

{

Person p1 = new Person();

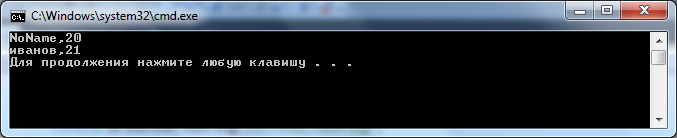
p1.Show();

Person p2 = new Person("Иванов", 21);

p2.Show();

}

}



Классы - это ссылочные типы, т. е. доступ к объектам классов осуществляется по адресу (через ссылку). Доступ к объектам классов через ссылки увеличивает расходы системных ресурсов, в том числе и памяти.

p

age

name

Person p1 = new Person();

p1.Show();

Person p2 = new Person("Иванов", 21);

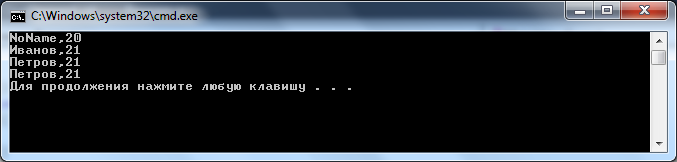
p2.Show();

p1 = p2;

p2.name = "Петров";

p1.Show();

p2.Show();



## 2. Структуры.

*Структура* подобна классу, но она является значимым, а не ссылочным типом.

**Описание структуры**

struct имя\_структуры тело\_структуры

Структуры могут содержать:

* + методы,
  + поля,
  + конструкторы **с параметрами**,
  + события.

Конструктор по умолчанию автоматически определяется для всех структур, и его изменить нельзя.

**Пример 2.**

struct Date

{

public int year, month, day;

public Date(int d, int m, int y)

{

year = y; month = m; day = d;

}

public override string ToString()

{

string result;

if (day < 10) result = "0" + day.ToString() + ":";

else result = day.ToString() + ":";

if (month < 10) result = result + "0" + month.ToString() + ":";

else result = result + month.ToString() + ":";

result = result + year.ToString();

return result;

}

public void Show()

{

Console.WriteLine(this.ToString());

}

}

class Program

{

static void Main(string[] args)

{

Date d1=new Date (1,1,2012);

Console.WriteLine(d1.ToString());

Date d2=new Date();

d2.Show();

//создали без использования new, но поля непроинициализрованы

Date d3;

//d3.Show();//Error;

d3.day = 5;

d3.month = 10;

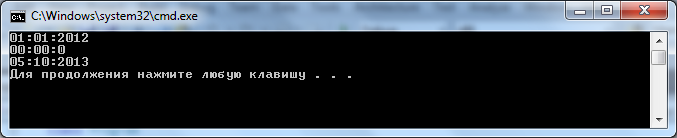
d3.year = 2013;

d3.Show();

}

}

Объекты типа Структура можно создавать с помощью операции new (также как и классы), в этом случае поля инициализируются конструктором. Если объект создается без использования конструктора (d3 в примере 1), то поля останутся не проинициализированы и им нужно будет присвоить значения вручную.



При присваивании одной структуры другой создается копия этого объекта.

d3 = d1;//копия

d3.day = 31;

Console.Write("d3= "); d3.Show();

Console.Write("d1= "); d1.Show();

d3

d3

d1

31

1

2012

1

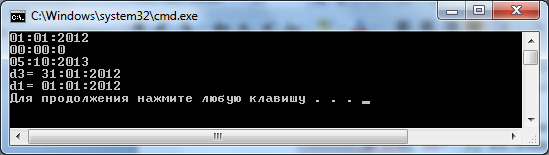
1

2012

1

1

2012



## 3. Перечисления

Перечисление (enumeration) – это множество именованных целочисленных констант.

**Описание перечисления**

**enum** имя {список\_перечисления}

**Пример перечисления**

**enum** month{Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

Каждый элемент перечисления – это целочисленная константа, список констант по умолчанию начинается с 0, каждая следующая константа на единицу больше предыдущей. Можно менять значения констант.

enum month{Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

Доступ к элементам перечисления осуществляется с помощью имен перечисления: Month.Jan

**Пример 3.**

enum Month { Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };

struct Date

{

public int year, day;

public Month month;

public Date(int d, Month m, int y)

{

year = y; month = m; day = d;

}

public override string ToString()

{

string result;

if (day < 10) result = "0" + day.ToString() + ":";

else result = day.ToString() + ":";

result = result + month.ToString() + ":";

result = result + year.ToString();

return result;

}

. . . .

}

class Program

{

static void Main(string[] args)

{

Date d1=new Date (1,Month.Jan,2012);

Console.WriteLine(d1.ToString());

Date d2=new Date(2,Month.Mar,2012);

d2.Show();

Date d3;

d3.day = 5;

d3.month = Month.Nov;

d3.year = 2013;

d3.Show();

}

Константу перечисления можно использовать везде, где допустимо целочисленное значение. Но при преобразовании константы перечисления в целое число или другое перечисление нужно использовать явное преобразование типов.

struct Date

{

. . . . .

public string ToMonthInt()

{

string result;

if (day < 10) result = "0" + day.ToString() + ":";

else result = day.ToString() + ":";

**result = result + ((int)month).ToString() + ":";**

result = result + year.ToString();

return result;

}

}

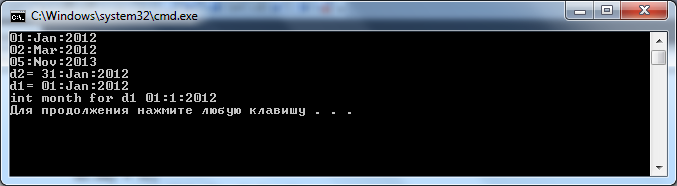
static void Main(string[] args)

{

. . . . . .

Console.WriteLine("int month for d1 " + d1.ToMonthInt());

}



По умолчанию перечисления используют тип int , но можно также создать перечисление любого другого целочисленного типа, за исключением типа char.

enum month:**byte** {Jan=1, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

Использование переменной из перечисления в качестве параметра цикла:

for (Month m = Month.Jan; m < Month.Dec; m++) Console.WriteLine(m);

Поскольку перечислимые типы представляют собой целочисленные значения, их также можно использовать для управления switch-инструкцией.

## Упаковка и распаковка

**Упаковка** – процесс преобразования из типа значений в тип ссылок. При упаковке в динамической памяти создается объект, которому присваивается значение объекта значимого типа. Результатом упаковки является адрес созданного объекта в динамической памяти (ссылка на объект). Упаковка выполняется автоматически.

Т. к. все классы C# имеют общий базовый класс object, то ссылке типа object можно присвоить значение структуры (упаковка будет выполняться автоматически).

**Распаковка** – обратная процедура, при которой значение объекта присваивается переменной значимого типа. Распаковка автоматически не выполняется. Для распаковки требуется применять операцию приведения типа.

class Person

{

public string name;

public int age;

public Person(string N, int A)

{

name = N; age = A;

}

public void Init(string N, int A)

{

name = N; age = A;

}

public void Show()

{

Console.WriteLine(name + ", возраст " + age);

}

}

struct Student

{

public string name;

public int age;

public Student(string N, int A)

{

name = N; age = A;

}

public void Init(string N, int A)

{

name = N; age = A;

}

public void Show()

{

Console.WriteLine(name + ", возраст " + age);

}

}

class Program

{

static void Main(string[] args)

{

Person p = new Person("Иванов", 20);

p.Show();

Student s;

s.name = "Петров"; s.age = 21;

s.Show();//динамическая память не выделяется

object[] mas = new object[5];

mas[0] = p;

mas[1] = s;//упаковка

foreach (object x in mas)

{

if (x is Person)

{

Person xp = x as Person;

xp.Show();

}

if (x is Student)

{

Student xs = (Student)x;//распаковка, as нельзя использовать!

xs.Show();

}

}

}

Упаковка и распаковка выполняются не только при присваивании, но и при передаче параметров, и к возвращению функцией результата с помощью оператора return.