

Классы и ООП

Лекция 3

Виденин Сергей Александрович

Гради Буч дает следующее определение объекта:

- Объект - это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области



Category	Ratings August 2014	Delta August 2015
Object-Oriented Languages	67.1%	+1.8%
Procedural Languages	27.8%	-0.4%
Functional Languages	3.4%	-1.5%
Logical Languages	1.7%	+0.1%



Понятие объекта

- В реальном мире каждый предмет или процесс обладает свойствами и поведением (набором статических и динамических характеристик).
*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.*



Достоинства ООП

- использование при программировании понятий, близких к предметной области;
- возможность успешно управлять большими объемами исходного кода благодаря инкапсуляции, то есть скрытию деталей реализации объектов и упрощению структуры программы;
- возможность многократного использования кода за счет наследования;
- сравнительно простая возможность модификации программ;
- возможность создания и использования библиотек объектов.



Недостатки ООП

- идеи ООП не просты для понимания и в особенности для практического использования
- для эффективного использования существующих объектно-ориентированных систем и библиотек требуется **большой объем первоначальных знаний**
- **неграмотное применение ООП может привести к значительному ухудшению характеристик разрабатываемой программы**
- некоторое снижение быстродействия программы, связанное с использованием виртуальных методов



Роли класса

- ▣ **Класс - это модуль**, архитектурная единица построения программной системы.
- ▣ **Класс - это тип данных**, задающий реализацию некоторой абстракции данных, характерной для задачи, в интересах которой создается программная система.

Состав класса, его размер определяется не архитектурными соображениями, а той абстракцией данных, которую должен реализовать класс. Если вы создаете класс Account, реализующий такую абстракцию как банковский счет, то в этот класс нельзя добавить поля из класса Car, задающего автомобиль.



Синтаксис класса

```
[атрибуты][модификаторы]class имя_класса[:список_родителей]
{тело_класса}
```

```
public class Rational {тело_класса}
```

В теле класса могут быть объявлены:

- *константы ;*
 - *поля ;*
 - *конструкторы и деструкторы ;*
 - *методы ;*
 - *события;*
 - *делегаты;*
 - *классы (структуры, интерфейсы, перечисления).*
-



Поля класса

- *Поля класса* синтаксически являются обычными переменными (объектами) языка.
- Содержательно *поля* задают представление той самой абстракции данных, которую реализует класс.
- *Поля* характеризуют *свойства объектов класса*.



Доступ к полям

- Каждое *поле* имеет *модификатор доступа*, принимающий одно из четырех значений: **public**, **private**, **protected**, **internal**.
- Независимо от значения атрибута доступа, все *поля* доступны для всех *методов класса*.
- Если некоторые *поля* должны быть доступны для *методов классов* В1, В2 и так далее, дружественных по отношению к классу А, то эти *поля* следует снабдить **атрибутом internal**, а все дружественные классы В **поместить в один проект (assembly)**.



Методы класса

- ▣ *Методы класса* синтаксически являются обычными процедурами и функциями языка.
- ▣ *Методы* содержат описания *операций*, доступных над объектами класса.
- ▣ Два объекта одного класса имеют один и тот же набор *методов*.



Доступ к методам

- Каждый метод имеет модификатор доступа, принимающий одно из четырех значений: `public`, `private`, `protected`, `internal`.
- Понятно, что класс, у которого все методы закрыты, абсурден, поскольку никто не смог бы вызвать ни один из его методов.
- **Интерфейс - это лицо класса** и именно он определяет, чем класс интересен своим клиентам, что он может делать, какие сервисы предоставляет клиентам.
- **Закрытые методы** составляют важную часть класса, позволяя клиентам не вникать во многие детали реализации

изменения в закрытых методах никак не отражаются на клиентах класса при условии корректной работы открытых методов.



Методы-свойства

- **Методы**, называемые **свойствами (Properties)**, представляют специальную синтаксическую конструкцию, предназначенную для обеспечения эффективной работы со свойствами.

Пять наиболее употребительных стратегий:

- чтение, запись (Read, Write);
 - чтение, запись при первом обращении (Read, Write-once);
 - только чтение (Read-only);
 - только запись (Write-only);
 - ни чтения, ни записи (Not Read, Not Write).
-



Рассмотрим класс Person

□ Пять *полей*: **fam, status, salary, age, health**

Для каждого из этих *полей* может быть разумной своя стратегия доступа. Возраст доступен для чтения и записи, фамилию можно задать только один раз, статус можно только читать, зарплата недоступна для чтения, а здоровье закрыто для доступа и только специальные *методы* класса могут сообщать некоторую информацию о здоровье персоны



Индексаторы

- Свойства являются частным случаем *метода класса* с особым синтаксисом.

Добавим в класс `Person` свойство `children`, задающее детей персоны, сделаем это свойство закрытым, а доступ к нему обеспечит *индексатор*

```
const int Child_Max = 20; //максимальное число детей
Person[] children = new Person[Child_Max];
int count_children=0; //число детей
public Person this[int i] //индексатор
{
    get {if (i>=0 && i< count_children)return(children[i]);
        else return(children[0]);}
    set
    {
        if (i==count_children && i< Child_Max)
            {children[i] = value; count_children++;}
    }
}
```

Статические поля и методы класса

- У класса могут быть *поля*, связанные не с объектами, а с самим классом. Эти *поля* объявляются как *статические* с модификатором `static`.
- ▣ ***Статические поля*** доступны всем *методам класса*.
- Например, у класса `Person` может быть *статическое поле* `message`, в котором каждый объект может оставить сообщение для других объектов класса.
- Аналогично *полям*, у класса могут быть и *статические методы*, объявленные с модификатором `static`.



Константы

- В нашем классе Person была задана *константа* Child_Max, характеризующая максимальное число детей у персоны.



Конструкторы класса

- ▣ **Конструктор** представляет собой специальный *метод класса*, позволяющий создавать объекты класса. Одна из синтаксических особенностей этого *метода* в том, что его имя должно совпадать с именем класса.
- ▣ `Person pers1 = new Person(), pers2 = new Person();
Person pers3 = new Person("Петрова");`



Разберем в деталях процесс создания

- первым делом для сущности `pers` создается ссылка, пока висячая, со значением `null` ;
 - затем в динамической памяти создается объект - структура данных с полями, определяемыми классом `Person`. Поля объекта инициализируются значениями по умолчанию: ссылочные поля - значением `null`, арифметические - нулями, строковые - пустой строкой. Эту работу выполняет *конструктор* по умолчанию, который, можно считать, всегда вызывается в начале процесса создания. Заметьте, если инициализируется переменная значимого типа, то все происходит аналогичным образом, за исключением того, что объект создается в стеке;
 - если поля класса проинициализированы, как в нашем примере, то выполняется инициализация полей заданными значениями;
 - если вызван *конструктор* с аргументами, то начинает выполняться тело этого *конструктора*. Как правило, при этом происходит инициализация отдельных полей класса значениями, переданными *конструктору*. Так, поле `fat` объекта `pers3` получает значение "Петрова";
 - На заключительном этапе ссылка связывается с созданным объектом.
-



Статический конструктор

- Такой конструктор может выполнять некоторую предварительную работу, которую нужно выполнить один раз, например, связаться с базой данных, заполнить значения *статических полей* класса, создать *константы* класса, выполнить другие подобные действия.

```
static Person()  
{  
    Console.WriteLine("Выполняется статический конструктор!");  
}
```

