Объектно-ориентированное программирование

основные понятия

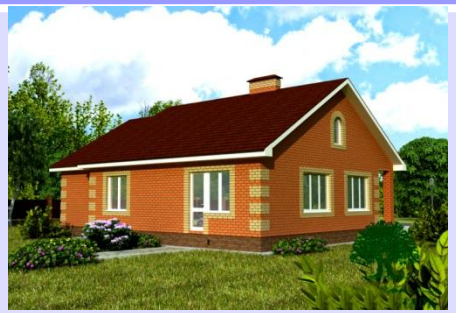
Объектно-ориентированное программирование (ООП)

- Это методика разработки программ, в основе которой лежит понятие объекта как некоторой структуры, соответствующей объекту реального мира, его поведению.

Информационная модель объекта

- Объект-оригинал заменяется набором его характеристик (свойств) и их значений

Объект - Дом

Свойство	Значение
Внешний вид	
Длина	8 м
Ширина	6 м
Высота	4,5 м
Количество этажей	1
Материал	Кирпич

Класс

- Это тип, описывающий устройство объектов (их поведение и способ представления)
- Класс можно сравнить с чертежом, согласно которому создаются объекты

Объявление класса

```
class Person
{
   string Name;
   string Address;
   int Year;
}
```

- Объекты как представители класса создаются при помощи операции new

```
Person Men = new Person();
Person Women = new Person();
```

Инкапсуляция

- Это объединение внутри класса его данных с функциями, обрабатывающими эти данные (методами)

Под инкапсуляцией понимают скрытие полей объекта с целью обеспечения доступа к ним только посредством методов класса

Методы

- Методы класса выполняют действия над объектами класса
- Включая объявление метода в объявление класса, программист явно указывает, какие действия могут быть выполнены над объектами этого класса

Объявление методов

```
class Person
{       string Name;
        string Address;
        int Year;
   public Person(string N, string A, int Y)
     {
        Name = N;
        Address = A;
        Year = Y;
      }
```

```
public void Show()
   {
    Console.WriteLine(Name);
    Console.WriteLine(Address);
    Console.WriteLine(Year);
    }
 public int GetYear()
    {
    return Year;
     }
}
```

Конструктор

В C# объект – это динамическая структура.

Переменная-объект содержит не данные, а ссылку на данные объекта.

Поэтому программист должен выделить память для этих данных при помощи специального метода класса – *конструктора.*

Имя конструктора совпадает с именем класса.

Описание конструктора

```
public Person(string N, string A, int Y)
    {
        Name = N;
        Address = A;
        Year = Y;
     }
```

Конструктор вызывается при создании объекта с помощью операции **new**

```
Person Men = new
Person(“Иванов”,“СПб”,1976);
```

Вызов методов

Для обычных (не статических) методов вызов осуществляется через **имя экземпляра класса**.

```
Person Men = new
Person(“Иванов”,“СПб”,1976);
Men.Show();
int myYear = Men.GetYear();
```

Для защиты полей объекта используют директиву private (закрытый)

- Директива private используется также для ограничения использования некоторых методов объекта
- Поля и методы, объявленные как private, не доступны за пределами класса, которому они принадлежат, даже в порожденных классах

Сокрытие данных

- Неотделимая часть ООП, управляющая областями видимости
- Целью сокрытия данных является предельная локализация изменений в коде программы
- Вместо прямого доступа к полям объекта следует использовать методы

Выводы:

- Инкапсуляция - это принцип, согласно которому любой класс должен рассматриваться как черный ящик
- Пользователь класса должен видеть и использовать только интерфейсную часть класса (список декларируемых свойств и методов)

Наследование

- Это механизм, позволяющий создавать *иерархии объектов*
- Наследованием называется возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка, добавляя при необходимости новые свойства и методы

Иерархия

- Это набор классов, связанных отношением наследования

- **Человек (класс Person)**
  - **Студент (класс Student)**
    - Очная форма (класс RealStudent)
    - Заочная форма (класс VirtualStudent)
  - **Профессор (класс Professor)**

Пусть определены три класса, один из которых является базовым для двух других

```
class Person
    {  string Name;
       string Address;
       int Year;
    public Person(string N, string A, int Y)
    {
       Name = N;
       Address = A;
       Year = Y;
     }
}
```

Класс-потомок должен иметь свой конструктор

```
class Student: Person
{
    int Group;

public Student(string N, string A, int Y, int G): base (N, A, Y)
    {
        Group = G;
    }
}
```

```
class Professor: Person
{
 string Kafedra;

public Professor(string N, string A, int Y, string K):
                                                    base (N, A, Y)
    {
       Kafedra = K;
    }
}
```

Переопределение методов

- Объект-потомок наследует не только поля родителя-объекта, но и методы
- Использование имени метода родительского типа в объявлении дочернего типа называется *переопределением метода*

Полиморфизм

- «поли» значит «много», а «морфизм» — «изменение» или «вариативность»
- таким образом, «полиморфизм» — это свойство одних и тех же объектов и методов принимать разные формы.

Полиморфизм

- Полиморфизмом называют явление, при котором один и тот же программный код (полиморфный код) выполняется по-разному, в зависимости от того, объект какого класса используется при вызове данного кода

Виртуальный метод

virtual - виртуальный (возможный – лат.)

Объявление метода виртуальным дает возможность дочернему классу произвести замену виртуального метода своим собственным

Метод порожденного класса, замещающий метод родительского класса, помечается директивой override

Определение метода Show для каждого класса

```
class Person
{
…
virtual public void Show()
      {
       Console.WriteLine(Name);
       Console.WriteLine(Address);
       Console.WriteLine(Year);
       }
}
```

```
class Student: Person
{
…
override public void Show()
      {
       base.Show();
       Console.WriteLine(Group);
      }
}
```

```
class Professor: Person
{
…
override public void Show()
      {
       base.Show();
       Console.WriteLine(Kafedra);
      }
}
```

Программа демонстрации наследования

```
namespace ConsoleApplication1
{
    class DecTo
    {
        string State;
        int Base;

        public DecTo(string s)
        {
            this.State = s;
            this.Base = s.Length;
        }
```

```
// метод перевода чисел
        public string Trans(int Dec)
        {
            string r = "\0";
            int Ost;
            do
            {
                Ost = Dec % Base;
                r = State[Ost] + r;
                Dec /= Base;
            }
            while (Dec > 0);
            return r;
        }
    }
```

```
class DecToBin : DecTo
    {
        public DecToBin() : base ("01");
    }

    class DecToOct : DecTo
    {
        public DecToOct(): base ("01234567");
    }

    class DecToHex : DecTo
    {
        public DecToHex(): base("0123456789ABCDEF");
    }
```

```
class Program
{
    static void Main()
    {
        int D;
        Console.WriteLine("Input value in 10 system:");
        D = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine();
        DecToBin P1 = new DecToBin();
        DecToOct P2 = new DecToOct();
        DecToHex P3 = new DecToHex();
        Console.WriteLine("2 system -> " + P1.Trans(D));
        Console.WriteLine("8 system -> " + P2.Trans(D));
        Console.WriteLine("16 system -> " + P3.Trans(D));
        Console.ReadKey();
    }
}
```

Результаты работы программы:

file:///C:/Users/229281/Desktop/Перевод в разные системы счисления...

```
Input value in 10 system:
209

2 system -> 11010001
8 system -> 321
16 system -> D1
```

Основные концепции ООП

- Система состоит из объектов
- Объекты взаимодействуют между собой, могут быть связаны отношением наследования
- Каждый объект характеризуется своим состоянием и поведением
- Состояние объекта задается значениями полей данных
- Поведение объекта задается методами
- Дочерние классы наследуют поля и методы своего родительского класса, с возможностью добавления новых свойств и методов
- Есть возможность переопределить в дочернем классе некоторые методы(не виртуальные методы переопределять не рекомендуется!)

Проверочная работа № 1

I вариант

Базовый класс Товар (поля **Название, Цена, Производитель**), дочерний класс Продукт (доп. поле **Срок реализации** - дней)

II вариант

Базовый класс Автомобиль (поля **Производитель, Модель, Год выпуска**), дочерний класс Автобус (доп. поле **Вместимость**)

План работы:

- Описать поля базового класса
- Описать конструктор базового класса
- Описать метод Show для вывода на экран значений полей
- Описать метод Get, возвращающий значение одного из полей
- Аналогично для дочернего класса
- В главной функции создать по 1 объекту каждого класса
- Показать вызов методов