ПЯТОЕ ЗАНЯТИЕ

Наследование

Наследование (inheritance) является одним из ключевых моментов ООП. Его смысл состоит в том, что мы можем расширить функциональность уже существующих классов за счет добавления нового функционала или изменения старого.

Простой класс

Класс представляет обычную модель человека

Только имя, ну и пусть

```
class Person
{
    private string _name;
    public string Name
    {
        get { return _name; }
    }
}
```

Класс работник

Теперь класс Employee «Является» человеком. То есть мы можем работать с классом Employee так же, как и с Person

```
class Employee : Person
{

}
```

ПРИМЕР

И поскольку объект Employee является также и объектом Person, то мы можем так определить переменную:

Person p = new Employee().

```
static void Main(string[] args)
{
    Person p = new Person { FirstName = "Bill", LastName = "Gates" };
    p.Display();
    p = new Employee { FirstName = "Denis",   LastName = "Ritchi" };
    p.Display();
    Console.Read();
}
```

Ограничения наследования

- Не поддерживается множественное наследование, класс может наследоваться только от одного класса. Хотя проблема множественного наследования реализуется с помощью концепции интерфейсов, о которых мы поговорим позже.
- При создании производного класса надо учитывать тип доступа к базовому классу – тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа internal, то производный класс может иметь тип доступа internal или private, но не public.
- Если класс объявлен с модификатором **sealed**, то от этого класса нельзя наследовать и создавать производные классы. Например, следующий класс не допускает создание наследников

Обращение к полям родителя

Мы не можем обратиться к приватным полям родителя, можем обращаться только к public, protected, internal, protected internal

```
class Employee : Person
{
    public void Display()
    {
        Console.WriteLine(_firstName);
    }
}
```

Ключевое слово base

Слово base используется для доступа к полям и методам класса – родителя.

К его конструкторам, поведению.

ДОБАВИМ КОНСТРУКТОР В PERSON

```
public Person(string name)
    {
        _name = name;
    }
```

ВЫЗОВ БАЗОВОГО КОНСТРУКТОРА

```
class Employee : Person
{
    public string Company {get; set;}

    public Employee(string name, string company) :
                    base(name)
    {
     Company = company;
     }
}
```

Соответственно, если мы в базовом классе создали конструктор с параметрами, мы в производных конструкторах всегда должны его вызывать.

Конструктор без параметров, если он есть, выполнится сам, без нашего вмешательства

Полиморфизм

Полиморфизм является третьим ключевым аспектом объектно-ориентированного программирования и предполагает способность к изменению функционала, унаследованного от базового класса. Полиморфизм предполагает определение полиморфного интерфейса в базовом классе - набор членов класса, которые могут быть переопределены в классе-наследнике. Методы, которые мы хотим сделать доступными для переопределения, в базовом классе помечается модификатором virtual. Такие методы называют виртуальными. Они и представляют полиморфный интерфейс.

Допустим в базовом классе есть метод **виртуальный**

```
class A
{
    public virtual void Smth(int arg)
    {
        Console.WriteLine("Base: " + arg);
    }
}
```

Теперь мы можем изменить поведение данного метода в производных

Необходимо слово override

```
class B : A
{
    public override void Smth(int arg)
    {
        Console.WriteLine("Child: " + arg);
    }
}
```