

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 помечаются 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);
    }
}
```