C# Наследование

Мухортова Н.Н.

Цель

Освоить один из самых мощных инструментов объектно-ориентированного программирования

Наследование

Наследование (inheritance) является одним из ключевых моментов ООП. Благодаря наследованию один класс может унаследовать функциональность другого класса.

Пример

```
class Person
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    public void Display()
    {
        Console.WriteLine(Name);
    }
}
```

Пример

Но вдруг нам потребовался класс, описывающий сотрудника предприятия - класс Employee.

Поскольку этот класс будет реализовывать тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником, или подклассом) от класса Person, который, в свою очередь, называется базовым классом или родителем (или суперклассом):

Наследник

```
class Employee : Person
{

}
```

После двоеточия мы указываем базовый класс для данного класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же свойства, методы, поля, которые есть в классе Person. Единственное, что не передается при наследовании, это конструкторы базового класса.

Main

```
static void Main(string[] args)
{
    Person p = new Person { Name = "Tom"};
    p.Display();
    p = new Employee { Name = "Sam" };
    p.Display();
    Console.Read();
}
```

И поскольку объект Employee является также и объектом Person, то мы можем так определить переменную: Person p = new Employee().

Класс Object

По умолчанию все классы наследуются от базового класса Object, даже если мы явным образом не устанавливаем наследование. Поэтому выше определенные классы Person и Employee кроме своих собственных методов, также будут иметь и методы класса Object: ToString(), Equals(), GetHashCode() и GetType().

Ограничения на наследование

1. Не поддерживается множественное наследование, класс может наследоваться только от одного класса.
2. 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.
3. Если базовый и производный класс находятся в разных сборках (проектах), то в этом случае производый класс может наследовать только от класса, который имеет модификатор public.
4. Если класс объявлен с модификатором sealed, то от этого класса нельзя наследовать и создавать производные классы.
5. Нельзя унаследовать класс от статического класса.

Доступ к членам базового класса из класса-наследника

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

Этот код не сработает и выдаст ошибку, так как переменная _name объявлена с модификатором private и поэтому к ней доступ имеет только класс Person.

Доступ к членам базового класса из класса-наследника

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

Но зато в классе Person определено общедоступное свойство Name, которое мы можем использовать

Ключевое слово base

```
class Person
{
    public string Name { get;  set; }

    public Person(string name)
    {
        Name = name;
    }

    public void Display()
    {
        Console.WriteLine(Name);
    }
}
```

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

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

Конструкторы

При вызове конструктора класса сначала отрабатывают конструкторы базовых классов и только затем конструкторы производных.

Выводы

Один класс может быть получен из другого, используя наследование.

Производный класс, также известный как дочерний класс или подкласс, наследует функциональные возможности от базового класса, также известного как родительский класс или суперкласс.

Подкласс может добавлять методы и свойства или изменять функциональные возможности, которые он унаследовал, чтобы обеспечить более специализированную версию базового класса.