

## Лабораторная работа 1.3

### «Виртуальные методы»

В этой лабораторной работе будут рассмотрены возможности позднего связывания в языке C# в рамках концепции виртуальных функций.

В предыдущей лабораторной работе, мы столкнулись с ситуацией, когда объект класса наследника, переопределяющего один из методов базового класса, будучи приведенным к этому базовому классу терял доступ к переопределенным методам.

Происходит это из-за того, что по умолчанию, связь между переменными и кодом, исполняемым над ними, устанавливается в момент компиляции программы, это называется ранним связыванием. Поскольку в момент компиляции про объект, находящийся под переменной типа `BaseClass`, известно только то, что он приводим к типу `BaseClass`, то логичным будет связать весь код с объектом базового класса.

При позднем связывании, связывание между идентификатором и кодом не происходит на этапе компиляции, во время исполнения программа должна установить эту связь сама. Т.е. в нашем случае вызова `Method()` у объекта `DerivedClass`, находящегося под переменной типа `BaseClass`, вызовется версия `DerivedClass.Method`, потому что связь между переменной и кодом будет устанавливаться в соответствии с тем объектом, который фактически лежит под этой переменной.

Для позднего связывания методов в классах наследниках, существует концепция виртуальных методов – метод, который может быть переопределен в классе наследнике так, что конкретная реализация метода для вызова будет определяться во время исполнения, т.е. как раз то, что нам и нужно. В C# есть ключевое слово `virtual`, которым можно пометить метод базового класса, чтобы сделать его виртуальным. При определении такого метода в классе наследнике необходимо указать ключевое слово `override`.

Напишем код аналогичный написанному в предыдущей лабораторной, единственное добавим ключевые слова `virtual` и `override`:

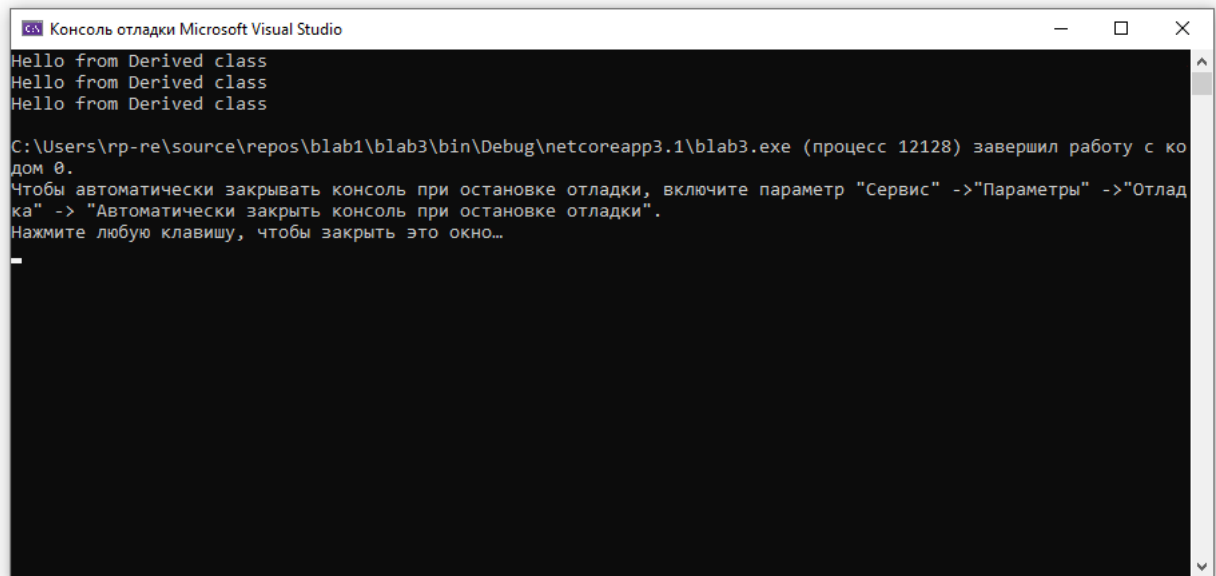
```
class BaseClass
{
    public virtual void Method()
    {
        Console.WriteLine("Hello from Base class");
    }
}

class DerivedClass : BaseClass
{
    public override void Method()
    {
        Console.WriteLine("Hello from Derived class");
    }
}
```

Теперь исполним тот же самый код, что и в предыдущей лабе и убедимся в правильности результата:

```
DerivedClass dInstance = new DerivedClass();  
dInstance.Method(); // Hello from Derived class  
  
// Down cast  
BaseClass bInstance = dInstance;  
bInstance.Method(); // Hello from Derived class  
  
// Up cast  
DerivedClass uInstance = (bInstance as DerivedClass);  
uInstance.Method(); // Hello from Derived class
```

Скриншот работы программы:



Консоль отладки Microsoft Visual Studio

```
Hello from Derived class  
Hello from Derived class  
Hello from Derived class  
  
C:\Users\rp-re\source\repos\blab1\blab3\bin\Debug\netcoreapp3.1\blab3.exe (процесс 12128) завершил работу с кодом 0.  
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".  
Нажмите любую клавишу, чтобы закрыть это окно...
```

## Приложение

1. Исходный код программы: <https://github.com/proxodilka/csharp-labs/tree/master/blab3>