УДК 004.42

НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ В C#

Вишняков В. В., студент гр. ИТм-171, II курс
Научный руководитель: Сыркин И.С., к.т.н., доцент
Кузбасский государственный технический университет имени Т.Ф.Горбачева
г. Кемерово

Наследование – это механизм объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), который позволяет описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом [1].

Наследование обозначается знаком двоеточие «:».

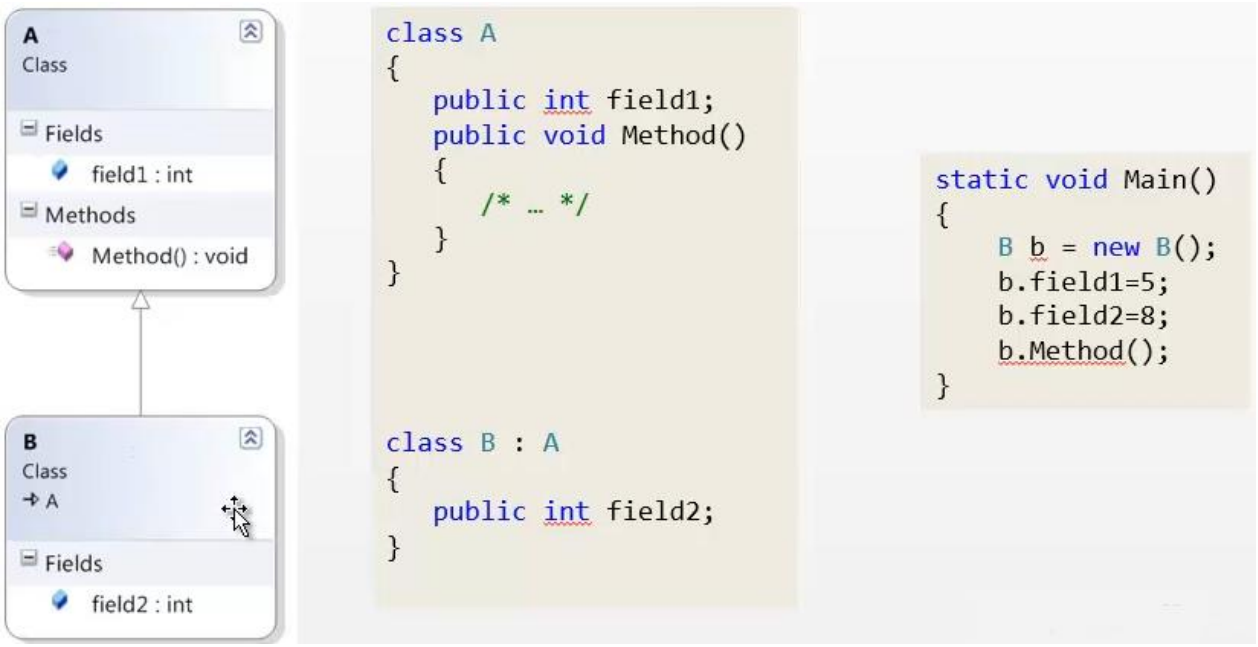

Рис. 1. UML диаграмма с примером наследования и примеры кода.

На Рисунке 1 на UML диаграмме мы видим произвольно созданные классы **A** и **B** с полями и методами. В примере кода класс B наследует от класса **A** поля и методы, также имея свое поле field2. При объявлении класса B в теле программы, мы видим, что можем обращаться к методам и полям, которые класс **B** унаследовал от класса **A**.

Модификаторы доступа – это ключевые слова, задающие доступность члена или типа. При помощи модификаторов доступа можно задавать уровни доступа к членам.

- **public** – доступ к типу или члену возможен из любого другого кода в той же сборке или другой сборке, ссылающейся на него.
- **protected** – доступ к типу или элементу можно получить только из кода в том же классе или структуре, либо производном классе.
- **private** – доступ к члену или типу можно получить только из кода в том же классе или структуре.

Полиморфизм – в парадигме ООП его можно описать как «один интерфейс, несколько функций».

Полиморфизм можно разделить на классический(принудительный) и «ad-hoc».

Классический имеет 2 формы:

- с использованием виртуальных членов
- с использованием привидения к базовому(или от базового) типов.

```
class BaseClass
{
    public int field1;
    public int field2;
    public int field3;
}

class DerivedClass : BaseClass
{
    public int field4;
    public int field5;
}
```

```
DerivedClass instance = new DerivedClass();

instance.field1 = 1;
instance.field2 = 2;
instance.field3 = 3;

instance.field4 = 4;
instance.field5 = 5;

// Приведение экземпляра класса DerivedClass к базовому типу BaseClass.
BaseClass newInstance = (BaseClass)instance; // Upcast

Console.WriteLine(newInstance.field1);
Console.WriteLine(newInstance.field2);
Console.WriteLine(newInstance.field3);

//Console.WriteLine(newInstance.field4);
//Console.WriteLine(newInstance.field5);
```

Рис. 2. Полиморфизм с использованием привидения типов.

На рисунке 2 можно наблюдать пример полиморфизма с использованием привидения типов. Создается DerivedClass, который наследуется от BaseClass и каждый класс имеет свои поля.

Далее в примере мы создаем экемпляр класса DerivedClass с именем instance. Как видим, мы получаем доступ к полям базового класса и класса наследника.

Далее используем операцию, называемую Upcast, т.е. приведение типа к базовому классу на экземпляре класса DerivedClass и, как следствие, получа-

ем доступ только к элементам базового класса. При попытке обратиться к полям DerivedClass получим ошибку уровня компиляции, т.к. мы произвели приведение к базовому классу. Обратное привидение базового класса к классу– наследнику называется Downcast.

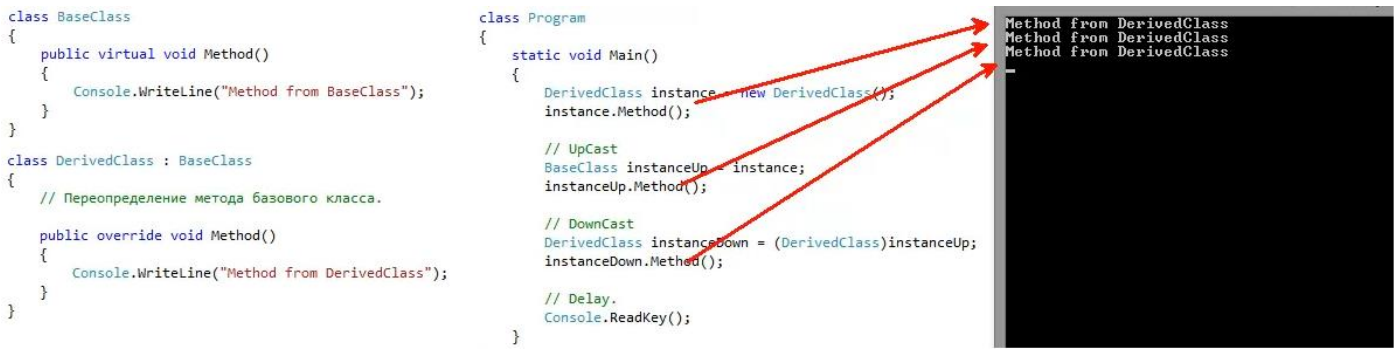

Рис. 3. Полиморфизм с использованием виртуальных членов.

На рисунке можно увидеть, что также создаются базовый и наследующий классы. Метод базового класса содержит модификатор virtual, в то время как в наследуемом классе такой же метод содержит модификатор override.

В теле программы создаем экемпляр класса DerivedClass с именем instance и вызываем в нем метод, при попытке провести Upcast или Downcast текст выводимый методом не меняется, что говорит о том, что полиморфизм с использованием виртуальных членов доминирует над полиморфизмом с использованием привидения типов (выполнится даже если было осуществлено привидение типов).

Однако привидение типов также сработает, и если бы в классах были методы без модификатора virtual/override, они бы отработали согласно предыдущего примера на рисунке 2.

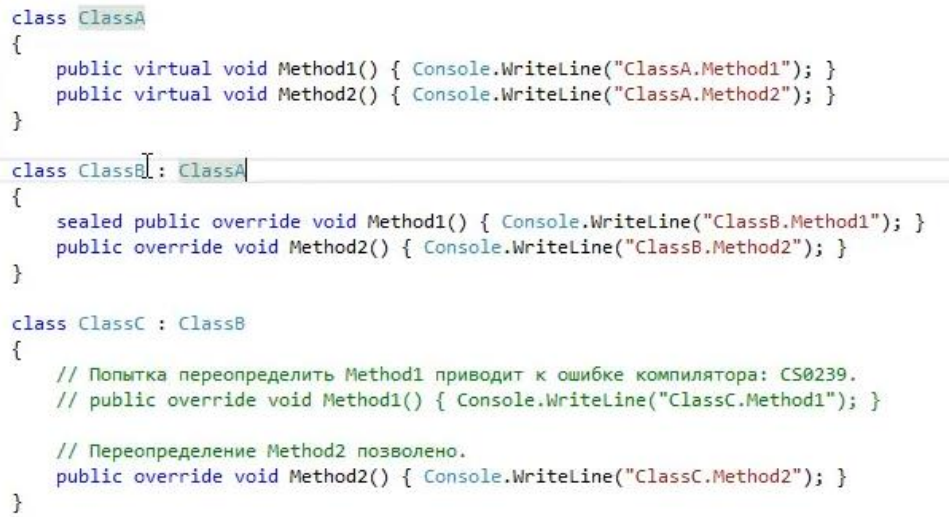

Рис. 4. Модификатор sealed.

Модификатор sealed запрещает наследовать класс или же, если данный модификатор применяется только к методам класса, то он запрещает переопределять метод в дальнейшем. На рисунке 4 все наглядно описано.

Всё вышеописанное является одной из основ объектно-ориентированного программирования на языке C# и активно применяется при создании программистами своих проектов.

Список литературы:

1. Курсы по C# на портале ITVDN. Режим доступа: https://itvdn.com/ru