Модификатор exclusive: наследуется всё, что не запрещено

admin
January 11 edited March 22

В OverScript наследуются все методы, а не только public и protected, как в C# и других языках. Запрет наследования метода делается модификатором exclusive. Соответственно, модификатора protected, который делает метод видимым в производных классах, в OverScript нет. Применять exclusive можно только к методам, а к переменным (полям) пока нельзя.
Это неконвенциональный подход, и я не уверен в его практичности. Но он представляется мне более понятным для начинающих программистов. Если же по мнению пользователей у этой концепции окажется больше минусов, чем плюсов, то я верну традиционные protected-методы.
Далее приведу примеры использования модификатора exclusive и потом покажу, почему он кажется мне удобнее, чем protected.
Допустим, мы хотим исключить метод класса-родителя из производного класса:

```
Bar b=new Bar();
b.Test2(); //Test2 at Bar
//b.Test(); //ошибка: функция не найдена

class Foo{
    public exclusive Test(){WriteLine("Test at Foo");}
}
class Bar:Foo{
    public Test2(){WriteLine("Test2 at Bar");}
}
```

Думаю, всё понятно. Теперь пример с конструктором:

```
new Bar(5L); //New at Bar. А без exclusive было бы: New at Foo (два конструктора было бы, и тот, который с long был бы подходящим).

class Foo{
    exclusive New(long x){
```

```
        WriteLine("New at Foo");
    }
}
class Bar:Foo{
    New(int x){
        WriteLine("New at Bar");
    }
}
```

Конструкторы в OverScript наследуются как обычные методы. Запрет наследования модификатором exclusive может помочь избежать возможных накладок, которые являются одной из причин отсутствия наследования конструкторов в других языках.
Теперь покажу, почему с protected было бы не очень удобно.

```
new Bar().Test(); //123

class Foo{
    public Test(){WriteLine(GetNum());}
    int GetNum(){return 123;}
}

class Bar:Foo{}
```

Представим, что для наследования нужно использовать модификатор protected. Тогда нужно было бы добавить protected к GetNum(), либо добавить fixed к Test():

```
new Bar().Test(); //123

class Foo{
    public Test(){WriteLine(GetNum());}
    protected int GetNum(){return 123;}
}

class Bar:Foo{}
```

Если GetNum() не protected, то её не будет в Bar и функция Test() в Bar не сможет скомпилироваться, т.к. будет ошибка "функция не найдена". В OverScript наследуемые функции компилируются для каждого класса, если не помечены модификатором fixed. В C# не так, а fixed имеет другое назначение. Добавить к одной GetNum() модификатор несложно, но в Test() может использоваться много функций, которые в свою очередь тоже могут вызывать другие функции, а значит, нам пришлось бы добавлять protected всем (если вариант с fixed для главной функции Test() нас не устраивает).
Приведу пример, чем полезно отдельное компилирование функции для каждого класса, где она унаследована:

```
new Bar().Test(); //555

class Foo{
    public Test(){WriteLine(GetNum());}
    int GetNum(){return 123;}
}

class Bar:Foo{
    int GetNum(){return 555;}
}
```

Теперь в Bar есть своя GetNum(), и Test() из Foo будет вызывать именно её. В C# пришлось бы сделать GetNum() виртуальной, а вызов виртуальных методов происходит значительно дольше обычных. В Python поведение как в OverScript, т.е. функции родительского класса вызывают перекрывающие функции из производного.

Говоря по-простому, в OverScript проще запрещать наследование отдельных методов, чем добавлять protected ко всем тем, которые нужно наследовать. Если мы написали класс без учёта, что его члены могут быть унаследованы производными классами, то, когда мы всё-таки решим использовать его в качестве родительского, добавлять модификаторы не потребуется.