# Почему объектно-ориентированное программирование провалилось

ООП как и каждая технология, проходит несколько этапов развития: от неприятия, через всеобщее восхваление до охлаждения и обыденного использования. Вам не надо — не используйте. Есть масса задач, где ООП не нужно. Только при достаточно обьёмных задачах без ООП можно завязнуть.
*—из интернета* http://otvet.mail.ru/question/96086478

К глубокому сожалению провалилось, по крайней мере, не сбылись предсказания об единоличном царствовании ООП. ООП оказалось забавной финтиклюшкой, которая не сделала революции в программировании.
*— из интернета*

чем проще язык программирования, тем трудней сделать на нем семантическую ошибку
*— очевидная истина.*

В 2000 году доктор компьютерных наук Стэнфорда, старший архитектор по разработке ПО сначала Sun, а потом и IBM, Ричард Гэбриел опубликовал нашумевшую статью (название вынесено в заголовок раздела), в которой он изложил свое скептическое отношение к парадигме объектно-ориентированного программирования (ООП). Два года спустя на ежегодной конференции IT-специалистов по объектно-ориентированным языкам и методологиям разработки ПО (г. Сиэтл) состоялась горячая дискуссия между сторонниками и противниками ООП[1]. Вводные выступления представителей обеих сторон можно почитать в русском переводе[2,3].

Появившаяся в 80-ых годах парадигма ООП была провозглашена «серебряной пулей» в борьбе со сложностью программ. Но, по мнению Ричарда Гэбриела, «она была искусственно и безальтернативно навязана в академической среде, причем мифы по поводу ООП, которые кочуют из учебника в учебник часто забавны и высосаны буквально из пальца».

Крупнейший специалист по Лиспу Пол Грэм утверждал, «что половина всех концепций ООП являются скорее плохими, чем хорошими, в связи с чем он искренне сочувствует ООП-программистам».

Другой крупный критик ООП, известный специалист по программированию Александр Степанов, автор известной библиотеки STL, который участвовал в создании C++, полностью разочаровался в парадигме ООП и писал так: «Я уверен, что парадигма ООП методологически неверна..».

Чем же закончилась дискуссия? Стороны так и не пришли к единому мнению, но показательно следующее свидетельство очевидцев: «…сторона, представлявшая объектно-ориентированное программирование, во время открытой дискуссии с противниками под смех зала даже запуталась в своих же концепциях. Люди вспоминают, что у всех создалось стойкое впечатление, что аргументация Lisp'еров (*критиков ООП – Л.Ч.*) была куда убедительней и последовательней, чем сторонников ООП».

Мы(я) склонны(ен) скорее к критикам и полностью согласны(ен) с высказыванием из первого эпиграфа к разделу. Хотя, конечно, мы писали и учили (и продолжаем учить) писать программы в стиле ООП. Но на практике не так уж часто возникала потребность в ООП.

Однажды, участвуя в одном из проектов, где базовым языком был C++, я был крайне удивлен реализацией одной из компонент, написанной моими коллегами. Я переписал её без всякого ООП, код получился в 3 раза короче и стал более понятен. Задаю вопрос: «Зачем такие сложности?». В ответ я не получила никаких аргументов, кроме «Но это же объектно-ориентированный язык!». Мне тогда очень хотелось написать об этом статью.

В ООП есть понятие *класса* и три ключевых его свойства: *инкапсуляция*, *наследование* и *полиморфизм*. Однако в развитии этих понятий в разных языках были созданы разные модели. Как говорится, вся суть в деталях. Не случайно, сторонники ООП в дискуссии запуталась в своих концепциях. Одно только понятие наследования имеет разные трактовки. Например, в [4] различается семь форм наследования: композиция, обобщение, расширение, ограничение, варьирование, конструирование, специализация и спецификация.

Между типами и классами много общего. Фактически класс – это тип с методами. В некоторых языках эти понятия почти одинаковые. Например, «в языке C# сглажено различие между

типом и классом. Все типы – встроенные и пользовательские – одновременно являются классами»[4].

Сподвижники Н.Вирта в своих языках последовательно шли по пути упрощения. И им удалось достичь той же мощности языка без добавления новых понятий. Рассмотрим это более подробно на примере Компонентного Паскаля (КП).

Как известно, в Паскале есть тип записи «record». Н. Вирт ввел понятие расширяемой записи: запись, в которую можно добавлять новые поля, объявляя другой тип. Например,

```
TYPE T0 = EXTENSIBLE RECORD X,Y:INTEGER END;
     T1 = RECORD (T0) R:INTEGER  END;
```

Теперь переменные типа T1 будут иметь три поля: первые два они «наследуют» (этого слова в КП не употребляют), а третье объявлено в расширенном типе:

```
VAR p0 : T0; p1 : T1;  . . .  p1.X=1;  p1.Y=2; p1.R=2;
```

Для работы с этими переменным могут быть написаны процедуры, например:

```
PROCEDURE pr0(VAR x:T0;n:INTEGER); BEGIN x.X:=n END pr0;
```

В КП эту же процедуру можно написать по-другому, написав параметр x перед именем процедуры. Такая процедура называется *методом*. При вызове метода имя фактического параметра, называемого *принимающим*, также указывается перед именем процедуры и отделяется точкой:

```
PROCEDURE (VAR x:T0)pr0(n:INTEGER); BEGIN x.X:=n END m0; . . .
p0.pr0(2);  (* вызов метода *)
```

В других языках (C++, C#, Java) процедуры, работающие с данными класса, текстуально объединяются в один блок: «class имя {… данные и методы класса … }», а параметр объекта неявно передается всем методам. В КП объединения не требуется, а параметр передается явно. Это соответствует общему принципу отсутствия умолчаний.

Переменные p0 и p1 являются *статическими* (память выделяется при компиляции) и поскольку их типы разные, они несовместимы по присваиванию. По-другому обстоят дела с *динамическими* типами, память под объекты которых выделяется во время выполнения с помощью процедуру NEW. Для работы с динамическими объектами объявляются типы-указатели:

```
TYPE P0 = POINTER TO T0; P1 = POINTER TO T1; . . .
VAR  px0 : P0; px1 : P1; . . .
NEW(px0);  NEW(px1);  // инициализация указателей – выделение памяти
```

Теперь указателю `px0` можно присваивать значение указателя `px1` и других указателей, если их типы являются расширениями типа T0. Говорят, что `px0` имеет динамический тип. С помощью оператора конкретизации типа (отметим, что в JavaScript и Паскале with имеет совсем другой смысл) мы можем выполнять разные действия:

```
WITH px0: T0 DO S0 | px0: T1 DO S1 ELSE S2 END.
```

Тип T0 и T1 могут иметь одинаковые методы. Если вызвать такой метод, то реально будет вызван метод, соответствующий динамическому типу. Это в ООП называется *динамическим связыванием* и обеспечивает полиморфизм. Например,

```
PROCEDURE (VAR x:T0) m0(n:INTEGER),NEW,EXTENSIBLE; BEGIN x.X:=n END m0;
PROCEDURE (VAR x:T1) m0(n:INTEGER); BEGIN x.X:=2*n END m0;
. . .
px0.m0(1);    (* вызывается метод, в котором x.X получает значение 1  *)
px0 := px1;   (* изменяется динамический тип px0                      *)
px0.m0(2);    (* вызывается метод, в котором x.X получает значение 2! *)
```

В языке C++ есть такие понятия, как *конструктор* и *деструктор*. Конструктор служит для того, чтобы создать и проинициализировать объект. При его исполнении неявно выделяется память под объект, а затем заполняются поля объекта. В КП нет таких понятий, но для выполнения аналогичных действий необходимо сначала явно выделяется память с помощью NEW, а потом проинициализировать объект. Это можно оформить в процедуру, которая и будет служить

конструктором. В деструкторах вообще нет нужды, так как есть сборка мусора. В C++ также есть сложные правила построения конструкторов дочерних объектов, например, неявный вызов конструкторов базового класса. В КП это тоже реализуется путем явного их вызова.

Важное свойство ООП инкапсуляция реализуется по-разному. Кроме распространенных атрибутов данных и методов public, private, protected, в C# есть пространства имен, в Java – пакеты. В КП основным и единственным средством инкапсуляции является модуль. Если переменная или процедура должны быть видны из другого модуля, после имени ставится знак «*» (аналог атрибута public). Если после имени переменной поставить знак «-», переменная будет доступна только для чтения.

. . .

Профессор Х. Мёссенбок, соавтор Н.Вирта по языку Оберон-2 (КП), так написал в своей статье: «Если программист четко себе представляет силу и предел возможностей объектно-ориентированного программирования и если он использует классы осознанно, то достоинства превзойдут недостатки. Однако негативные моменты могут стремительно нарастать, если классы применять бездумно, особенно в тех ситуациях, когда они не только не уменьшают проблемы, а наоборот, лишь добавляют сложности».[6]


1. Игорь Савчук. Почему объектно-ориентированное программирование провалилось? 23.09.2010. http://citforum.ru/gazeta/165/
2. Ричард П. Гэбриэл Объектная парадигма провалилась. http://bugtraq.ru/library/programming/objectshavefailed.html
3. Гай Л. Стил. Объектная парадигма не провалилась. http://bugtraq.ru/library/programming/objectshavenotfailed.html
4. Т.Бадд. Объектно-ориентированное программирование в действии. – Спб. : Изд- Питер, 1997.
5. Биллиг В.А. Основы программирования на C#. Учебное пособие. – М. : Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2006. – 483. : ил.
6. Х. Мёссенбок (1995). Плюсы и минусы объектно-ориентированного программирования. 1998, ИнфоАрт. http://www.online-ane.ru/oopplus&min.pdf
7.