|  |
| --- |
| EPAM Systems |
| ООП. Принципы и парадигмы |
| [Document subtitle] |

|  |
| --- |
| Oksana Spolnyk  [Date] |

# Определение

**Паради́гма программи́рования** — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

**Основные модели программирования**

* Императивное программирование
* Декларативное программирование
* Структурное программирование
* Функциональное программирование
* Логическое программирование
* Объектно-ориентированное программирование
* Программирование, основанное на классах
* Программирование, основанное на прототипах
* Субъектно-ориентированное программирование

**Подходы и приёмы**

* Структурное программирование
* Процедурное программирование
* Аппликативное программирование
* Обобщённое программирование
* Доказательное программирование
* Порождающее программирование
* Аспектно-ориентированное программирование
* Агентно-ориентированное программирование
* Рекурсия
* Автоматное программирование
* Событийно-ориентированное программирование
* Компонентно-ориентированное программирование
* Грамотное программирование

**Объе́ктно-ориенти́рованное программи́рование (ООП)** — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. В случае языков с прототипированием вместо классов используются объекты-прототипы.

**Класс -** является описываемой на языке терминологии исходного кода моделью ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект — это экземпляр класса. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

**Объект** - сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса или копирования прототипа (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

**Прототип** — это объект-образец, по образу и подобию которого создаются другие объекты. Объекты-копии могут сохранять связь с родительским объектом, автоматически наследуя изменения в прототипе; эта особенность определяется в рамках конкретного языка.

# Основные принципы ООП

Основными принципами объектно-ориентированного программирования являются:

* абстракция данных
* инкапсуляция;
* полиморфизм;
* наследование.

## Абстракция данных

Объекты представляют собою упрощенное, идеализированное описание реальных сущностей предметной области. Если соответствующие модели адекватны решаемой задаче, то работать с ними оказывается намного удобнее, чем с низкоуровневым описанием всех возможных свойств и реакций объекта.

**Абстрагирование** — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик

## Инкапсуляция

**Инкапсуляция** — это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик — пользователь класса должен видеть и использовать только интерфейсную часть класса (т. е. список декларируемых свойств и методов класса) и не вникать в его внутреннюю реализацию. Поэтому данные принято инкапсулировать в классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов. Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между классами и, соответственно, упростить независимую реализацию и модификацию классов.

**Инкапсуляция** — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе, и скрыть детали реализации от пользователя.

## Наследование

Наследованием называется возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка (прародителя, иногда его называют суперклассом) и добавляя, при необходимости, новые свойства и методы. Набор классов, связанных отношением наследования, называют иерархией. Наследование призвано отобразить такое свойство реального мира, как иерархичность.

**Наследование** — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

## Полиморфизм

**Полиморфизмом** называют явление, при котором функции (методу) с одним и тем же именем соответствует разный программный код (полиморфный код) в зависимости от того, объект какого класса используется при вызове данного метода. Полиморфизм обеспечивается тем, что в классе-потомке изменяют реализацию метода класса-предка с обязательным сохранением сигнатуры метода. Это обеспечивает сохранение неизменным интерфейса класса-предка и позволяет осуществить связывание имени метода в коде с разными классами — из объекта какого класса осуществляется вызов, из того класса и берётся метод с данным именем. Такой механизм называется динамическим (или поздним) связыванием — в отличие от статического (раннего) связывания, осуществляемого на этапе компиляции.

Полиморфизм имеет минимум два плюса. Во-первых, он позволяет группировать объекты, имеющие общий базовый класс, и последовательно (например, в цикле) их обрабатывать. Второе достоинство: старый код может использовать новый код.

**Полиморфизм** — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. При использовании термина «полиморфизм» в сообществе ООП подразумевается полиморфизм подтипов; а использование параметрического полиморфизма называют обобщённым программированием.

# Подходы к проектированию программ в целом

ООП ориентировано на разработку крупных программных комплексов, разрабатываемых командой программистов (возможно, достаточно большой). Проектирование системы в целом, создание отдельных компонентов и их объединение в конечный продукт при этом часто выполняется разными людьми, и нет ни одного специалиста, который знал бы о проекте всё.

Объектно-ориентированное проектирование состоит в описании структуры и поведения проектируемой системы, то есть, фактически, в ответе на два основных вопроса:

* Из каких частей состоит система.
* В чём состоит ответственность каждой из частей.

Выделение частей производится таким образом, чтобы каждая имела минимальный по объёму и точно определённый набор выполняемых функций (обязанностей), и при этом взаимодействовала с другими частями как можно меньше.

Дальнейшее уточнение приводит к выделению более мелких фрагментов описания. По мере детализации описания и определения ответственности выявляются данные, которые необходимо хранить, наличие близких по поведению агентов, которые становятся кандидатами на реализацию в виде классов с общими предками. После выделения компонентов и определения интерфейсов между ними реализация каждого компонента может проводиться практически независимо от остальных (разумеется, при соблюдении соответствующей технологической дисциплины).

Большое значение имеет правильное построение иерархии классов. Одна из известных проблем больших систем, построенных по ООП-технологии — так называемая проблема хрупкости базового класса. Она состоит в том, что на поздних этапах разработки, когда иерархия классов построена и на её основе разработано большое количество кода, оказывается трудно или даже невозможно внести какие-либо изменения в код базовых классов иерархии (от которых порождены все или многие работающие в системе классы). Даже если вносимые изменения не затронут интерфейс базового класса, изменение его поведения может непредсказуемым образом отразиться на классах-потомках. В случае крупной системы разработчик базового класса просто не в состоянии предугадать последствия изменений, он даже не знает о том, как именно базовый класс используется и от каких особенностей его поведения зависит корректность работы классов-потомков.

# Производительность объектных программ

Гради Буч указывает на следующие причины, приводящие к снижению производительности программ из-за использования объектно-ориентированных средств:

1. Динамическое связывание методов. Обеспечение полиморфного поведения объектов приводит к необходимости связывать методы, вызываемые программой (то есть определять, какой конкретно метод будет вызываться) не на этапе компиляции, а в процессе исполнения программы, на что тратится дополнительное время. При этом реально динамическое связывание требуется не более чем для 20 % вызовов, но некоторые ООП-языки используют его постоянно.
2. Значительная глубина абстракции. ООП-разработка часто приводит к созданию «многослойных» приложений, где выполнение объектом требуемого действия сводится к множеству обращений к объектам более низкого уровня. В таком приложении происходит очень много вызовов методов и возвратов из методов, что, естественно, сказывается на производительности.
3. Наследование «размывает» код. Код, относящийся к «конечным» классам иерархии наследования, которые обычно и используются программой непосредственно, находится не только в самих этих классах, но и в их классах-предках. Относящиеся к одному классу методы фактически описываются в разных классах. Это приводит к двум неприятным моментам:
   1. Снижается скорость трансляции, так как компоновщику приходится подгружать описания всех классов иерархии.
   2. Снижается производительность программы в системе со страничной памятью — поскольку методы одного класса физически находятся в разных местах кода, далеко друг от друга, при работе фрагментов программы, активно обращающихся к унаследованным методам, система вынуждена производить частые переключения страниц.
4. Инкапсуляция снижает скорость доступа к данным. Запрет на прямой доступ к полям класса извне приводит к необходимости создания и использования методов доступа. И написание, и компиляция, и исполнение методов доступа сопряжены с дополнительными расходами.
5. Динамическое создание и уничтожение объектов. Динамически создаваемые объекты, как правило, размещаются в куче, что менее эффективно, чем размещение их на стеке и, тем более, статическое выделение памяти под них на этапе компиляции.

Несмотря на отмеченные недостатки, Буч утверждает, что выгоды от использования ООП более весомы. Кроме того, повышение производительности за счёт лучшей организации ООП-кода, по его словам, в некоторых случаях компенсирует дополнительные накладные расходы на организацию функционирования программы. Можно также заметить, что многие эффекты снижения производительности могут сглаживаться или даже полностью устраняться за счёт качественной оптимизации кода компилятором. Например, упомянутое выше снижение скорости доступа к полям класса из-за использования методов доступа устраняется, если компилятор вместо вызова метода доступа использует инлайн-подстановку (современные компиляторы делают это вполне уверенно).

# Критика ООП

Несмотря на отдельные критические замечания в адрес ООП, в настоящее время именно эта парадигма используется в подавляющем большинстве промышленных проектов. Однако, нельзя считать, что ООП является наилучшей из методик программирования во всех случаях.

Критические высказывания в адрес ООП:

1. Было показано отсутствие значимой разницы в продуктивности разработки программного обеспечения между ООП и процедурным подходом.
2. Кристофер Дэйт указывает на невозможность сравнения ООП и других технологий во многом из-за отсутствия строгого и общепризнанного определения ООП.
3. Александр Степанов в одном из своих интервью указывал, что ООП «методологически неправильно» и что «…ООП практически такая же мистификация, как и искусственный интеллект…».
4. Фредерик Брукс указывает, что наиболее сложной частью создания программного обеспечения является «…спецификация, дизайн и тестирование концептуальных конструкций, а отнюдь не работа по выражению этих концептуальных конструкций…». ООП (наряду с такими технологиями как искусственный интеллект, верификация программ, автоматическое программирование, графическое программирование, экспертные системы и др.), по его мнению, не является «серебряной пулей», которая могла бы на порядок величины снизить сложность разработки программных систем. Согласно Бруксу, «…ООП позволяет сократить только привнесённую сложность в выражение дизайна. Дизайн остаётся сложным по своей природе…».
5. Эдсгер Дейкстра указывал: «…то, о чём общество в большинстве случаев просит — это змеиное масло. Естественно, „змеиное масло“ имеет очень впечатляющие названия, иначе будет очень трудно что-то продать: „Структурный анализ и Дизайн“, „Программная инженерия“, „Модели зрелости“, „Управляющие информационные системы“ (Management Information Systems), „Интегрированные среды поддержки проектов“, „Объектная ориентированность“, „Реинжиниринг бизнес-процессов“…».
6. Никлаус Вирт считает, что ООП — не более чем тривиальная надстройка над структурным программированием, и преувеличение её значимости, выражающееся, в том числе, во включении в языки программирования всё новых модных «объектно-ориентированных» средств, вредит качеству разрабатываемого программного обеспечения.
7. Патрик Киллелиа в своей книге «Тюнинг веб-сервера» писал: «…ООП предоставляет вам множество способов замедлить работу ваших программ…».

# Источники

<http://megamozg.ru/post/6910/>

<https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>

http://phdblog.org.ua/m/Principy\_obektno-orientirovannogo\_programmirovaniya.html