**ООП — основные принципы**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 1](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=1), Лекция 1

— Привет, Амиго! Сегодня я открою для тебя новый и интересный мир. Этот мир называется **ООП** – объектно-ориентированное программирование. Ты уже познакомился с классами и объектами. Сегодня ты узнаешь про них больше, намного больше.

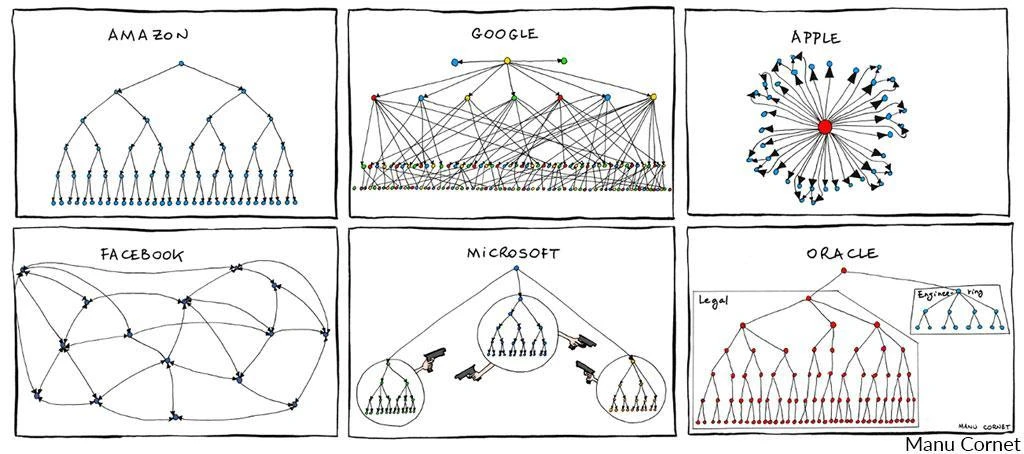
Начнем с четырех принципов ООП. Это: абстракция, инкапсуляция, наследование и полиморфизм. (Раньше их было три, но потом решили добавить абстракцию)

**1) Абстракция.**

Хорошим примером абстракции в реальной жизни является описание должностей в компании или организации. Название должности – это одно, а обязанности каждой конкретной должности – это уже совсем другое.

Представь, что ты проектируешь структуру своей будущей компании. Ты можешь разделить обязанности секретаря: «раскидать» их по нескольким другим должностям. Можешь разбить должность исполнительного директора на несколько независимых должностей: финансовый директор, технический директор, директор по маркетингу, директор по персоналу. Или, например, объединить должности офис-менеджера и рекрутера в одну.

Ты придумываешь названия должностей в своей фирме, а потом «раскидываешь» обязанности по этим должностям. Это и есть **абстракция** – разбиение чего-то большого, монолитного на множество маленьких составных частей.

[](https://cdn.javarush.ru/images/article/0456620d-7e6f-48d8-8d4e-25dc98b88023/original.jpeg)

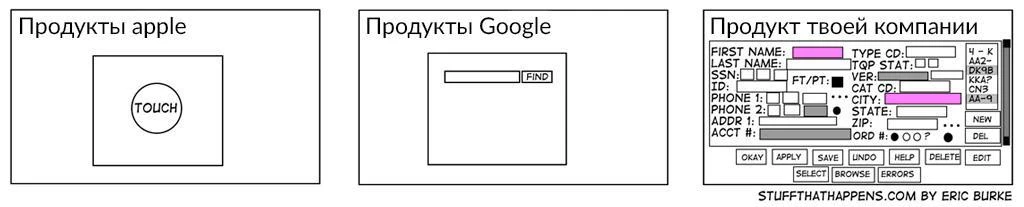
С точки зрения программирования, абстракция — это, скажем так, **правильное разделение программы на объекты.**

Обычно любую большую программу можно десятками способов представить в виде взаимодействующих объектов. Абстракция позволяет отобрать главные характеристики и опустить второстепенные.

Абстракция – это как стратегия в военном деле. Плохая стратегия – и никакой гениальной тактикой ситуацию уже не исправить.

**2) Инкапсуляция.**

Цель инкапсуляции – улучшить качество взаимодействия вещей за счет упрощения их.

[](https://cdn.javarush.ru/images/article/08ef00ec-a6f5-4777-a14b-53bb0948b3c9/original.jpeg)

А лучший способ упростить что-то – это скрыть все сложное от посторонних глаз. Например, если тебя посадят в кабину Боинга, ты не сразу разберешься, как им управлять:

[](https://cdn.javarush.ru/images/article/12f49867-c6dc-498e-8ad1-27119b5a31a1/original.jpeg)

С другой стороны, для пассажиров самолета все выглядит проще: купил билет, сел в самолет, взлетели и приземлились. Ты можешь с легкостью перелететь с континента на континент, обладая только навыками «купить билет» и «сесть на самолет». Все сложности в виде подготовки самолета к полету, взлета, посадки и различных внештатных ситуаций скрыты от нас. Не говоря уже о спутниковой навигации, автопилоте и диспетчерских центрах в аэропортах. И это упрощает нам жизнь.

С точки зрения программирования, инкапсуляция – это «сокрытие реализации». Мне нравится такое определение. Наш класс может содержать сотни методов и реализовывать очень сложное поведение в различных ситуациях. Но мы можем скрыть от посторонних глаз все его методы (пометить модификатором private), а для взаимодействия с другими классами оставить всего пару-тройку методов (пометить их модификатором public). Тогда все остальные классы нашей программы будут видеть в этом классе всего три метода, и будут вызывать именно их. А все сложности будут скрыты внутри класса, как кабина пилотов от счастливых пассажиров.

**3) Наследование.**

У наследования есть две стороны. Сторона программирования и сторона реальной жизни. С точки зрения программирования, наследование – это специальное отношение между двумя классами. Но гораздо интереснее, что же такое наследование с точки зрения реальной жизни.

Если бы нам понадобилось что-то создать в реальной жизни, то у нас есть два решения:

**1)** создать нужную нам вещь с нуля, потратив кучу времени и сил.

**2)** создать нужную нам вещь на основе уже существующей.

Наиболее оптимальная стратегия выглядит так: берем существующее хорошее решение, немного его дорабатываем, подгоняем под свои нужды и используем.

Если мы проследим историю возникновения человека, то окажется, что с момента зарождения жизни на планете прошли миллиарды лет. А если представить, что человек возник из обезьяны (на основе обезьяны), то прошла всего пара миллионов лет. Создание с нуля – дольше. Гораздо дольше.

В программировании тоже есть возможность создавать один класс на основе другого. Новый класс становится потомком (наследником) уже существующего. Это очень выгодно, когда есть класс, который содержит 80%-90% нужных нам данных и методов. Мы просто объявляем подходящий класс родителем нашего нового класса, тогда в новом классе автоматически появляются все данные и методы класса-родителя. Правда, удобно?

**4) Полиморфизм.**

Полиморфизм – это понятие из области программирования. Оно описывает ситуацию, когда за одним интерфейсом скрываются разные реализации. Если постараться поискать его аналоги в реальной жизни, то одним из таких аналогов будет процесс управления машиной.

Если человек может управлять грузовиком, то его можно посадить и за руль скорой, и за руль спорткара. Человек может управлять машиной вне зависимости от того, что это за машина, потому что все они имеют одинаковый интерфейс управления: руль, педали и рычаг коробки передач. Внутреннее устройство машин разное, но все они имеют одинаковый интерфейс управления.

Если вернуться к программированию, то полиморфизм позволяет единообразно обращаться к объектам различных классов (обычно имеющих общего предка) – вещь, которую трудно переоценить. Ценность его тем выше, чем больше программа.

**ООП** – это принципы. Внутренние законы. Каждый из них нас в чем-то ограничивает, давая взамен **большие преимущества**, когда программа вырастает до больших размеров. Четыре [принципа ООП](https://javarush.ru/groups/posts/principy-oop) – это как четыре ножки стула. Убери хотя бы одну, и вся система станет неустойчивой.