**Еще одно объяснение ООП (слабая связность, четкие функции)**

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

[Уровень 1](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=1), Лекция 3

— Привет, Амиго! Хотела тебе рассказать еще об одном преимуществе использования ООП. Видишь ли – программы больше напоминают не строения, а животных. **Их не строят, их выращивают.**Разработка — это постоянные изменения. В строительстве ты можешь иметь хороший план и четко ему следовать. В случае с разработкой программ – это не так.

Очень часто что-то нельзя сделать тем способом, который ты себе наметил, и приходится многое переделывать. Еще чаще меняются требования заказчика.

— А если заказчик проекта дал очень точную его спецификацию?

— Тогда взгляни на ситуацию во времени. Успех продукта приведет к тому, что заказчик захочет выпустить его новую версию, а затем еще и еще. И, конечно, нужно будет всего лишь добавить «**небольшие изменения**» в уже существующий продукт. Поэтому разработка продукта – это последовательность постоянных изменений. Только масштаб времени разный. Каждая новая версия может выходить раз в неделю, раз в месяц или раз в полгода.

— И какой вывод можно сделать из всего этого?

— Внутреннюю структуру продукта нужно поддерживать в таком состоянии, которое позволит внести значительные (и не очень) изменения с минимальными переделками.

— И как такое сделать?

— Мы уже говорили, что программа состоит из объектов, которые взаимодействуют между собой. Давай нанесем на доску все объекты нашей программы, обозначив их жирными точками. И проведем от каждого объекта (точки) стрелочки ко всем объектам (точкам), с которыми он взаимодействует.

Теперь мы будем объединять объекты (точки) в группы. Точки должны быть объединены в группу, если связи между ними гораздо интенсивнее, чем с остальными точками. Если большинство стрелочек от точки идет к точкам ее же группы, тогда разбиение на группы произошло правильно. Точки внутри одной группы мы будем называть сильно связанными, а точки из разных групп – слабо связанными.

Это называется «**принцип слабой связности**». Программа разбивается на несколько частей, часто слоев, логика которых сильно завязана на их внутреннее устройство и очень слабо на другие слои/части. Обычно взаимодействие слоев очень регламентировано. Один слой может обращаться ко второму и использовать только небольшую часть его классов.

— Тот же принцип «разделения на отделы» только в большем масштабе?

— Именно. Это приводит к тому, что мы можем реорганизовать отдел, повысить его эффективность, нанять в него еще больше людей, но если мы не изменим протокол взаимодействия других отделов с нашим, то все сделанные изменения останутся локальными. Никому не придется переучиваться. Не придется переделывать всю систему. Каждый отдел может заниматься такой внутренней оптимизацией, если общие механизмы взаимодействия выбраны удачно.

— Выбраны удачно. А что будет, если они выбраны неудачно?

— Тогда «**запас изменений**» быстро иссякнет и придется переделывать всю систему. Такое приходится делать время от времени. Нельзя предугадать, что будет в будущем, но можно свести количество таких переделок к минимуму.

— Хорошо. Про пользу такого разделения я понял, а ООП тут причем?

— Выбор структуры отделов и способа их взаимодействия – это «принцип Абстракции». В программировании он используется для определения, на какие части лучше разбить программу, и как эти части должны взаимодействовать. Данный принцип также можно применять к разделению полученных частей, пока мы не разобьем программу на отдельные классы.

— А сокрытие внутренней структуры этих частей, и жёсткие ограничения на взаимодействие с другими частями – это Инкапсуляция, да?

— Именно. Инкапсуляция + Абстракция – это краеугольные камни [ООП](https://javarush.ru/groups/posts/principy-oop). Хорошая программа обязана следовать этим двум принципам. В дальнейшем мы рассмотрим остальные принципы и поймем, какие преимущества они дают.

— Очень интересно. Жду с нетерпением.