Лекция 9

Принципы SOLID

Design Patterns - Роберт Мартин

Признаки плохого кода:

- закрепощенность. Система с трудом поддается изменениям. Изменение в одной части затрагивают изменения в других частях.
- неустойчивость. Изменения в одном месте приводят к краху в другом, при том, что оно не связано с первым
- неподвижность. Систему трудно разделить на подсистемы для повторного использования кода
- вязкость. Делать что-то неправильно проще, чем правильно
- неоправданная сложность. Проект использует сложную внутреннюю инфраструктуру, которая не несет выгоды.
- неопределенность. Проект невозможно прочесть, чтобы что-то изменить, необходимо долго разбираться в коде.

Для избавления от признаков плохого кода были сформулированы принципы **solid**

- **S** - single; принцип единственной ответственности. Существует только одна причина, чтобы существовал данный класс.
- **O** - open/close; принцип открытости/закрытости. Программные элементы должны быть открыты для расширения и закрыты для модификации. То, что написано раньше,; то, что мы хотим добавить, должно наследоваться и доопределяться.
- **L** - принцип Барбары Лисков; принцип подстановки. Программные компоненты должны иметь возможность быть замененными на экземпляры их подтипов без изменения основной части.
- **I** - interface segregation; принцип разделения интерфейсов. Лучше много маленьких интерфейсов, чем один большой.
- **D** - dependency inversion; инверсия зависимостей. Все должно зависеть от абстракций, но не от их экземпляров.

Принцип единственной ответственности.

Применяется, когда появляется такая ситуация при изменении кода, отвечающего за одну ответственность для одного класса, происходит изменение кода, отвечающего за другую ответственность.

Когда объекту становится дозволено слишком много.

Когда доменная логика концентрируется в одном классе.

Когда любое изменение логики приводит к изменению в местах, в которых это не подразумевалось.

Приходится тестировать места, которые не связаны с текущим.

Приходится исправлять ошибки в местах, не связанных с текущим.

Когда невозможно использовать написанное в другом месте.

Принцип открытости/закрытости.

Существует принцип открытости/закрытости Мейера и полиморфный.

Будем рассматривать полиморфный принцип.

Программы должня быть открыты для расширения. Это значит, что программа может быть расширена путем добавления новых подтипов. Программа должна быть закрыта для изменения - значит, что при расширении программы не должен изменяться код базовой сущности.

Мейер при написании книги столкнулся с вопросом: как можно разработать проект, устойчивый к изменению, срок жизни которых превышает срок существования первой версии проекта?

Принцип открытости/закрытости поддерживает выделение абстрактных классов.

Принцип подстановки

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового тип, не зная об этом.

Поведение наследуемых классов не должно противоречить поведению базового класса.

Принцип разделения интерфейса

Следование этому принципу помогает проводить рефакторинг кода. Клиенты не должны зависеть от методов, которые они не используют.

CRUD - Create, Read, Update, Delete.