# Принципы и паттерны проектирования

TODO: данная глава может быть рассмотрена в качестве заключения к книге!

Если оглянуться на пройденный нами путь, то будет достаточно легко найти связь между принципами и паттернами проектирования. Принципы проектирования более фундаментальны и лежат в основе практически любого решения, которое принимается во время проектирования. Паттерны проектирования описывают каркас решения определенных проблем и могут меняться от одной предметной области к другой. Эти инструменты находятся "на разных уровнях абстракции", но они преследуют единую цель: получить дизайн, который будет легко понимать, развивать и тестировать.

Те или иные принципы проектирования лежат в основе любого паттерна. Паттерны проектирования предназначены для борьбы со сложностью (SRP), для ограничения распространения изменений и получения расширяемого решения (OCP), в основе многих из них лежит наследование, корректное использование которого невозможно без LSP. Применение паттернов приводит к слабосвязанному дизайну, в основе которого лежит принцип наименьшего знания (ISP), а четкое разграничение ответственности приводит к минимизации связей между слоями приложения (DIP).

**Принцип единственной обязанности** лежит в основе многих паттернов проектирования. Суть большинства паттернов заключается в изоляции сложного аспекта поведения в отдельном классе или группе классов. **Фабрика** позволяет изолировать процесс создания объекта, когда смешивание этой обязанности с логикой самого класса становится неприемлемым. Главный смысл использования **Декораторов** заключается в выделении отдельного аспекта поведения в отдельный класс, что нужно лишь тогда, когда сложность этого аспекта является очень высокой. Паттерн **Итератор** позволяет вынести логику обхода коллекции в отдельный класс, а основная роль **Адаптера** заключается лишь в одном: в адаптации интерфейса класса без изменения поведения.

* **Принцип открыт/закрыт** отражает одну из самых важных задач паттернов: изоляцию изменений. Многие паттерны проектирования представляют собой своеобразные "шлюзы", которые затормозят нахлынувшую волну изменений. **Фасад** позволяет изолировать клиента от сложности внутреннего устройства библиотеки, а **Адаптер** и **Посредник** позволяют ограничить изменения лишь несколькими классами. Очень многие паттерны проектирования отражают второе свойство Принципа открыт/закрыт - расширяемость решения. **Стратегия** позволяет переходить от одной реализации алгоритма к другой без изменения клиентов. **Фабрики** "прячут" процесс создания объектов или целые иерархии наследования, позволяя гибко расширять поведение системы. **Декоратор** дает возможность "нанизывать" поведение объектов, а **Посетитель** позволяет добавлять операции в закрытые иерархии типов.
* **Принцип подстановки Лисков** лежит в основе корректной реализации иерархий наследования и вместе с **Шаблонным методом** является инструментом получения расширяемого и сопровождаемого решения.
* **Принципы разделения интерфейсов** и **Инверсии зависимостей** не лежат в основе паттернов проектирования, тут роль скорее обратная: применение паттернов проектирования позволяет следовать этим принципам. Паттерны **Посредник** и **Фасад** избавляют классы от лишней информации друг о друге, а использование **Стратегии** и **Наблюдателя** позволяет следовать Принципу инверсии зависимостей.

Разумное следование принципам проектирования, и умелое применение паттернов проектирования не являются гарантией успеха проекта. Но они хотя бы подскажут направление, куда нужно двигаться для получения хорошего дизайна.