Структурные паттерны

- **Адаптер**
  - позволяет объектам с несовместимыми интерфейсами работать вместе
  - Проблема:
    - система поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс
  - Решение:
    - создать объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту
  - Пример:
    - вилка-розетка
    - сайт, поддерживающий разные виды оплаты
  - Применимость:
    - Повторное использование чужого кода
    - Адаптивный рефакторинг
- **Мост**
  - разделяет один или несколько классов на две отдельные иерархии — абстракцию и реализацию, позволяя изменять их независимо друг от друга
  - Проблема:
    - при использовании наследования реализация жестко привязывается к абстракции, что затрудняет независимую модификацию (фигуры и цвета)
  - Решение:
    - заменить наследование композицией
  - Пример:
    - Java Abstract Window Toolkit (AWT)
    - иметь несколько видов GUI
    - поддерживать много видов API
    - сайт с разными страницами, и надо разрешить пользователям менять их тему
  - Применимость:
    - когда класс нужно расширять в двух независимых плоскостях
    - разделить монолитный класс на несколько отдельных иерархий
- **Компановщик**
  - позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней так, будто это единичный объект
  - Проблема:
    - предоставить клиенту единообразный доступ к листовым и составным элементам древовидной структуры
  - Решение:
    - вводит абстрактный базовый класс Component с поведением, общим для всех примитивных и составных объектов
  - Пример:
    - Игры
    - программы для вычислений
    - Армия
  - Применимость:
    - когда нужно представить древовидную структуру объектов
    - необходимо объединять группы схожих объектов и управлять ими
    - когда клиенты должны единообразно трактовать простые и составные объекты
- **Декоратор**
  - позволяет динамически добавлять объектам новую функциональность, оборачивая их в полезные «обёртки»
  - Проблема:
    - добавить новые обязанности в поведении или состоянии отдельных объектов во время выполнения программы. Использование наследования не представляется возможным, поскольку это решение статическое и распространяется целиком на весь класс
    - комбинирование поведения объектов обычно приводит к увеличению количества подклассов
  - Решение:
    - поместить целевой объект в другой объект-обёртку, который запускает базовое поведение объекта, а затем добавляет к результату что-то своё
  - Пример:
    - система оповещений
    - для построения графических пользовательских интерфейсов
  - Применимость:
    - Когда вам нужно добавлять обязанности объектам на лету, незаметно для кода, который их использует
    - Когда нельзя расширить обязанности объекта с помощью наследования
- **Фасад**
  - предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку
  - Проблема:
    - большое количество объектов некой сложной библиотеки или фреймворка
    - самостоятельная инициализация объектов
  - Решение:
    - простой интерфейс для работы со сложной подсистемой, содержащей множество классов
  - Пример:
    - сотрудник службы поддержки
    - интегрированная среда разработки
    - система загрузки видео (внутри система сжатия файлов и т.д.)
  - Применение:
    - вы используете какую-то сложную библиотеку со множеством подвижных частей, но вам нужна только часть её возможностей
    - Когда вам нужно представить простой или урезанный интерфейс к сложной подсистеме
- **Легковес/ Приспособленец**
  - позволяет вместить бóльшее количество объектов в отведённую оперативную память
  - экономит память, разделяя общее состояние объектов между собой, вместо хранения одинаковых данных в каждом объекте
  - Проблема:
    - новые объекты уже не вмещаются в оперативную память
  - Решение:
    - не хранить в классе внешнее состояние, а передавать его в те или иные методы через параметры
    - одни и те же объекты можно будет повторно использовать в различных контекстах
  - Пример:
    - игры
    - текст
    - структура города
  - Применение
    - Когда не хватает оперативной памяти для поддержки всех нужных объектов
- **Заместитель/Прокси**
  - позволяет подставлять вместо реальных объектов специальные объекты-заменители
  - эти объекты перехватывают вызовы к оригинальному объекту, позволяя сделать что-то до или после передачи вызова оригиналу
  - Проблема:
    - доступ к ресурсу, создание которого требует больших затрат
  - Решение:
    - создать новый класс-дублёр, имеющий тот же интерфейс, что и оригинальный служебный объект
    - при получении запроса от клиента объект-заместитель сам бы создавал экземпляр служебного объекта и переадресовывал бы ему всю реальную работу
  - Пример:
    - Банковский чек
  - Применение
    - Ленивая инициализация (виртуальный прокси). Когда у вас есть тяжёлый объект, грузящий данные из файловой системы или базы данных
    - Логирование запросов (логирующий прокси). Когда требуется хранить историю обращений к сервисному объекту
    - Защита доступа (защищающий прокси). Когда в программе есть разные типы пользователей, и вам хочется защищать объект от неавторизованного доступа

XMind : ZEN | Trial Mode