Проектирование ПО

Лекция 9: Структурные шаблоны

Тимофей Брыксин
timofey.bryksin@gmail.com

Паттерны проектирования

повторимая архитектурная конструкция, являющаяся решением некоторой типичной технической проблемы

- подходит для класса проблем
- переиспользуемость знаний
- унификация терминологии
- простота в изучении
- BEWARE: карго-культ!

...а ещё есть антипаттерны!

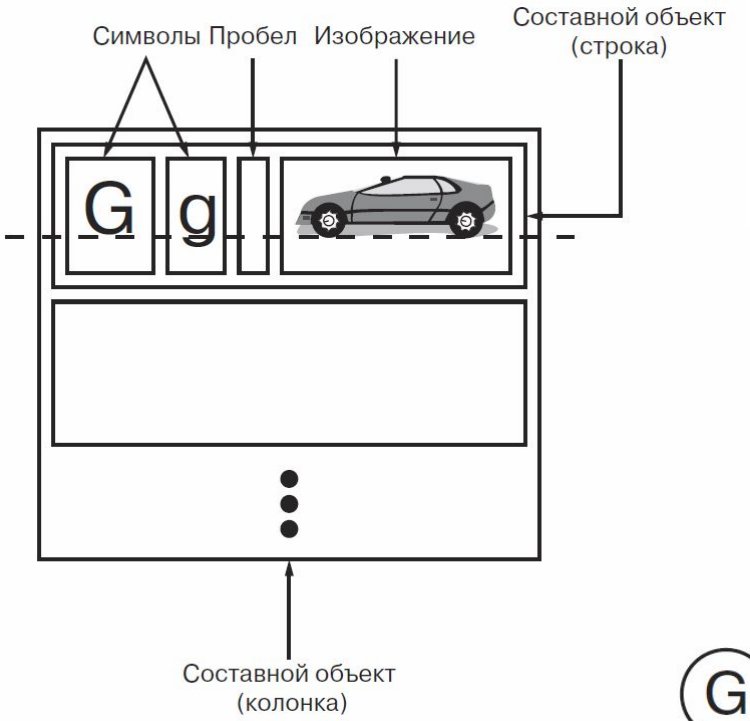
Текстовый редактор

- Структура документа
- Форматирование
- Создание привлекательного интерфейса пользователя
- Поддержка стандартов внешнего облика программы
- Поддержка оконных систем
- Операции пользователя, undo/redo
- Проверка правописания и расстановка переносов

Структура документа

- Документ — множество графических элементов
  - организация в физическую структуру
  - средства UI для манипулирования структурой
- Требования к внутреннему представлению
  - отслеживание внутренней структуры документа
  - генерирование визуального представления
  - отображение позиций экрана на внутренее представление
- Ограничения
  - текст и графика едины
  - простой и составной элементы едины

Рекурсивная композиция



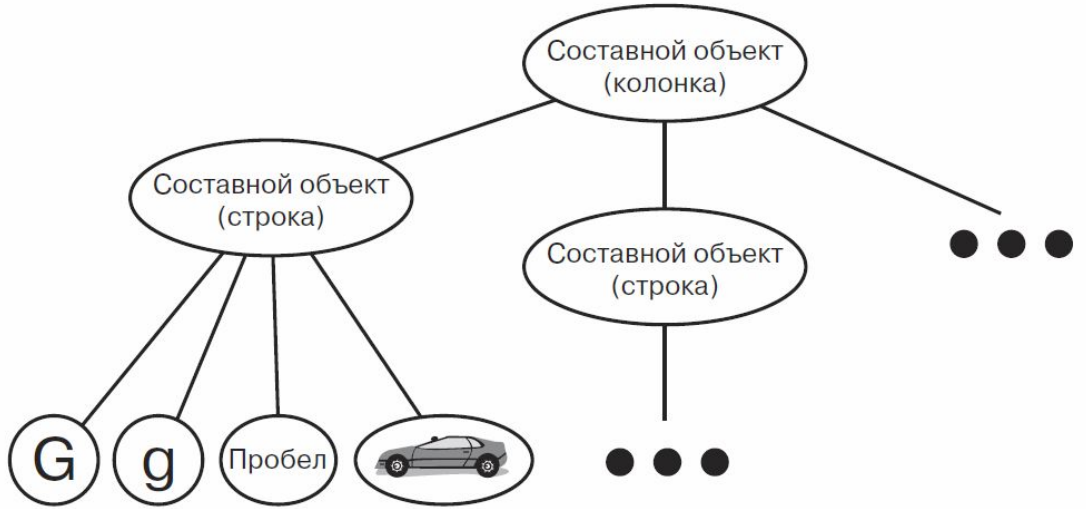
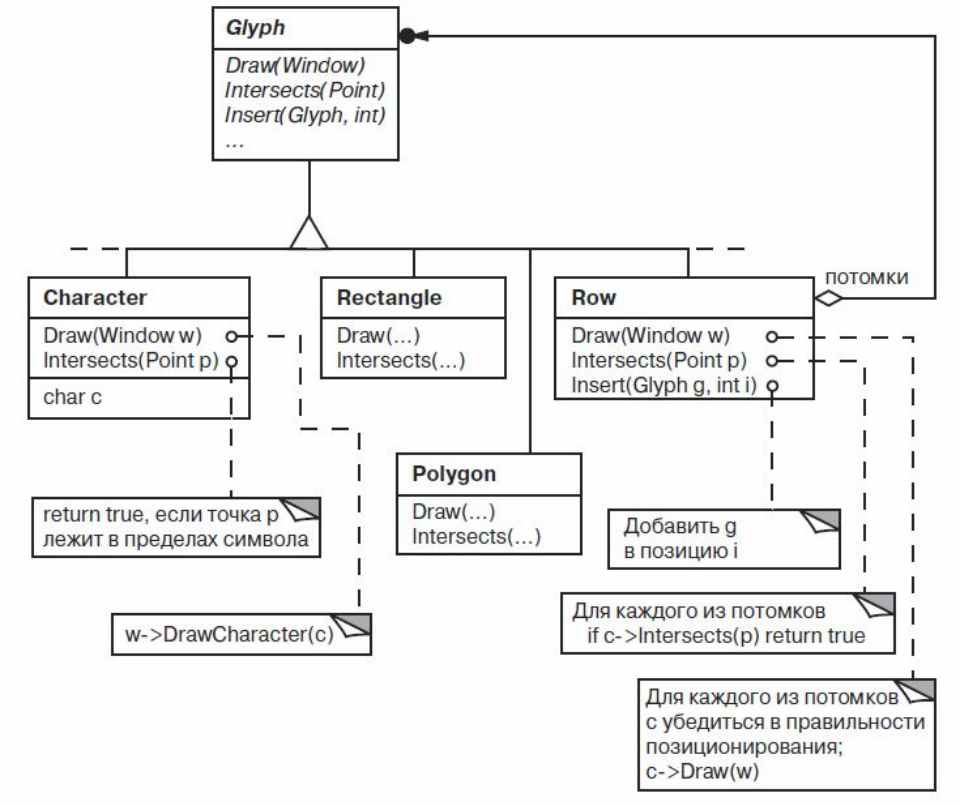
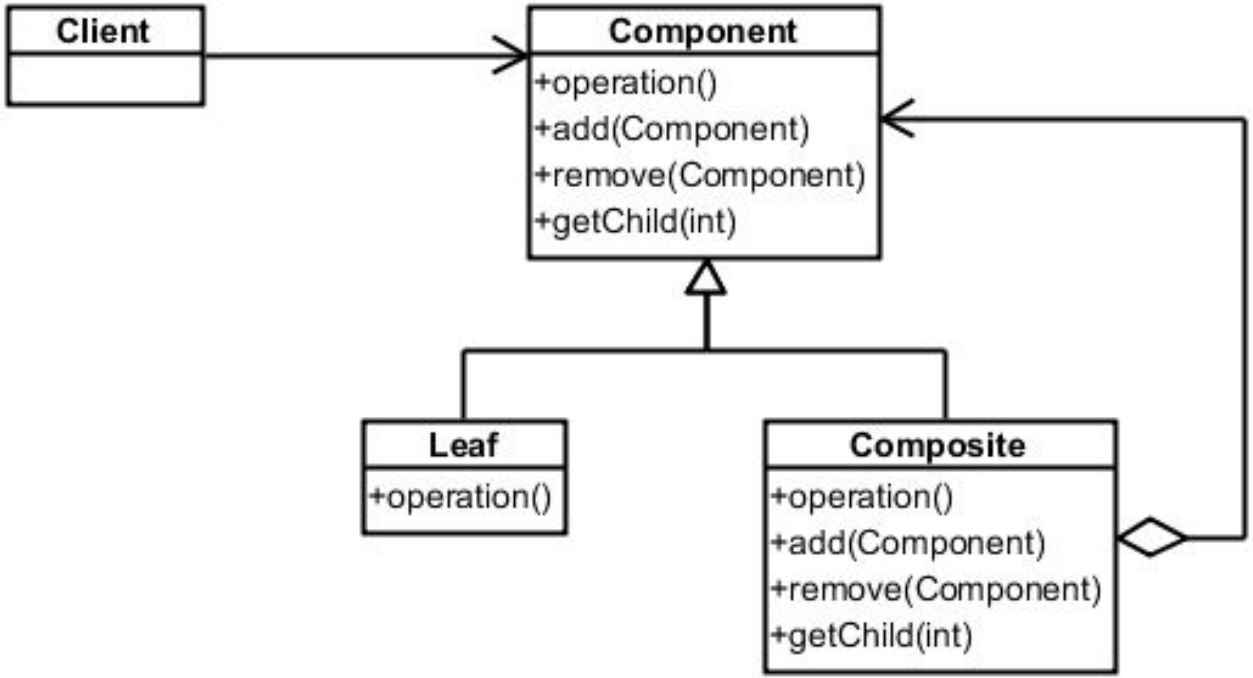


Диаграмма классов: глифы



Паттерн Компоновщик



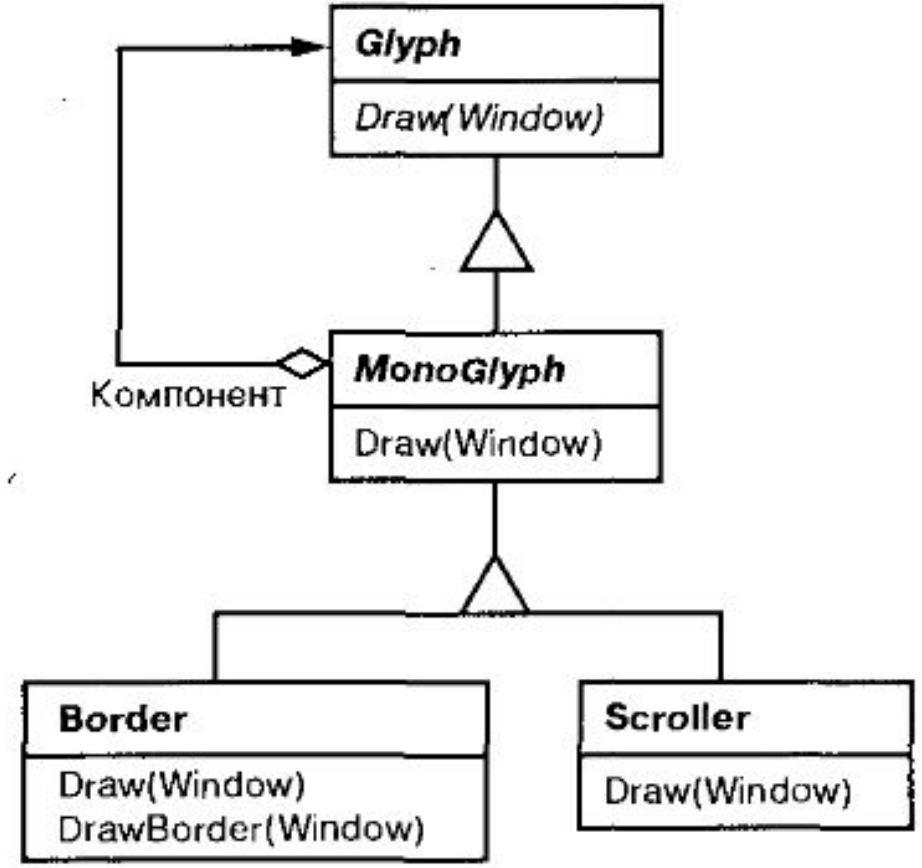
Компоновщик: особенности

- представление иерархии объектов вида часть-целое
- единообразная обработка простых и составных объектов
- простота добавления новых компонентов

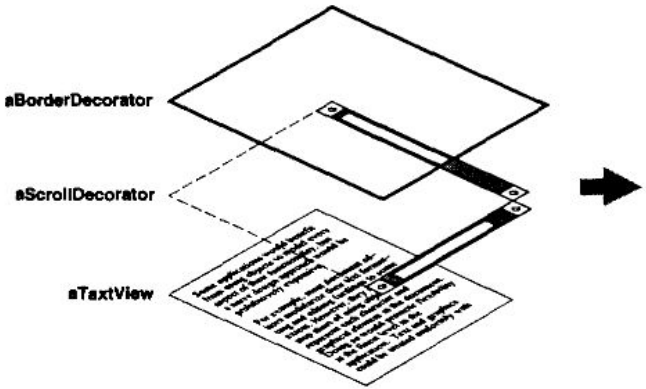
Усовершенствование UI

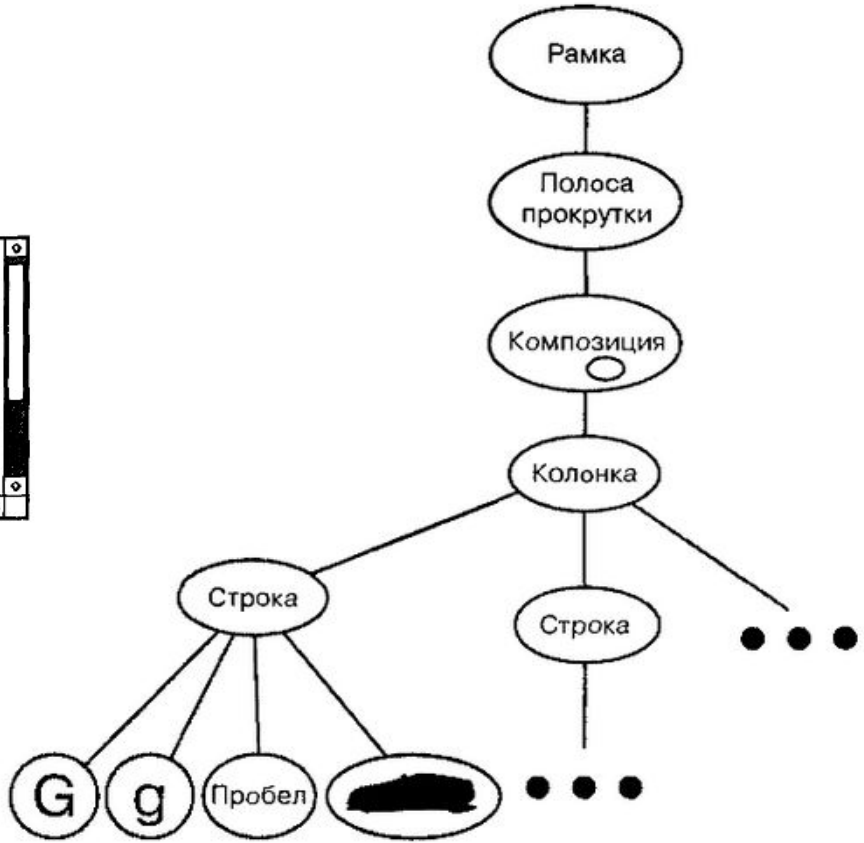
- Хотим сделать рамку вокруг текста и полосы прокрутки, отключаемые по опции
- Желательно убирать и добавлять элементы обрамления так, чтобы другие объекты даже не знали, что они есть
- Хотим менять во время выполнения -- наследование не подойдёт
  - наш выбор --- композиция
  - прозрачное обрамление

Моноглиф

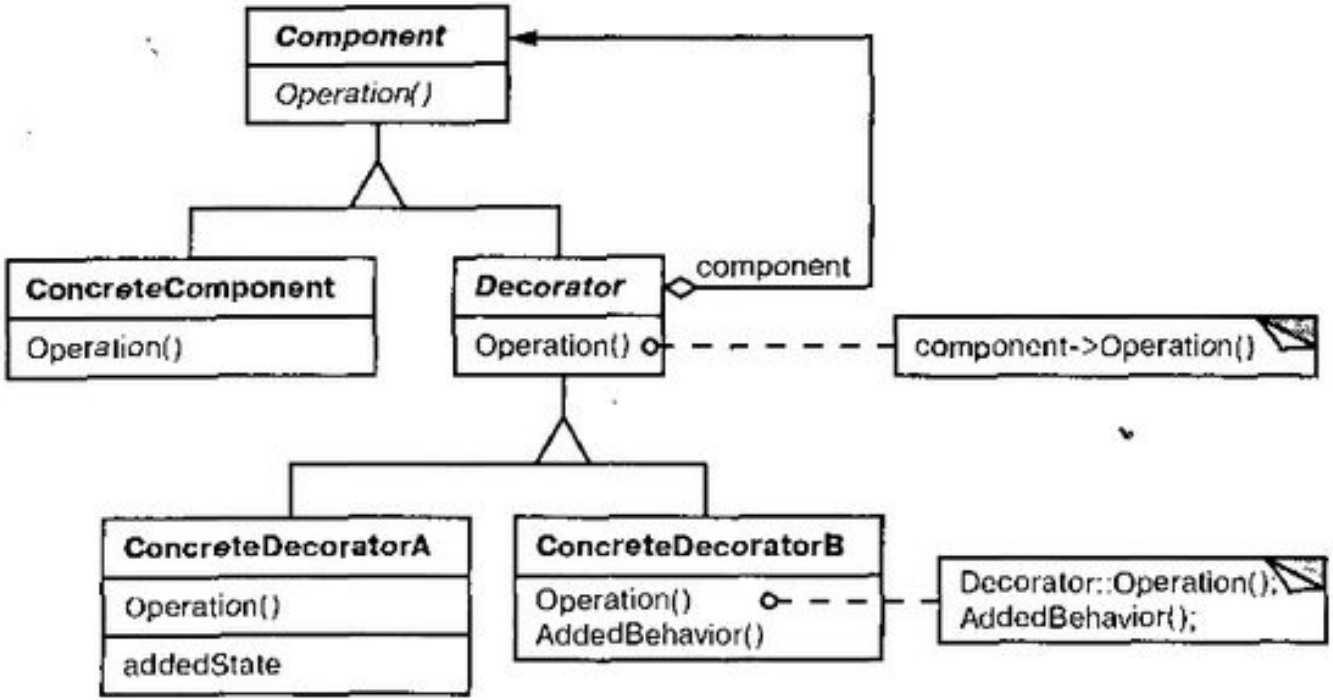

Glyph
Draw(Window)
Компонент
MonoGlyph
Draw(Window)
Border
Draw(Window)
DrawBorder(Window)
Scroller
Draw(Window)

Структура глифов





Паттерн Декоратор



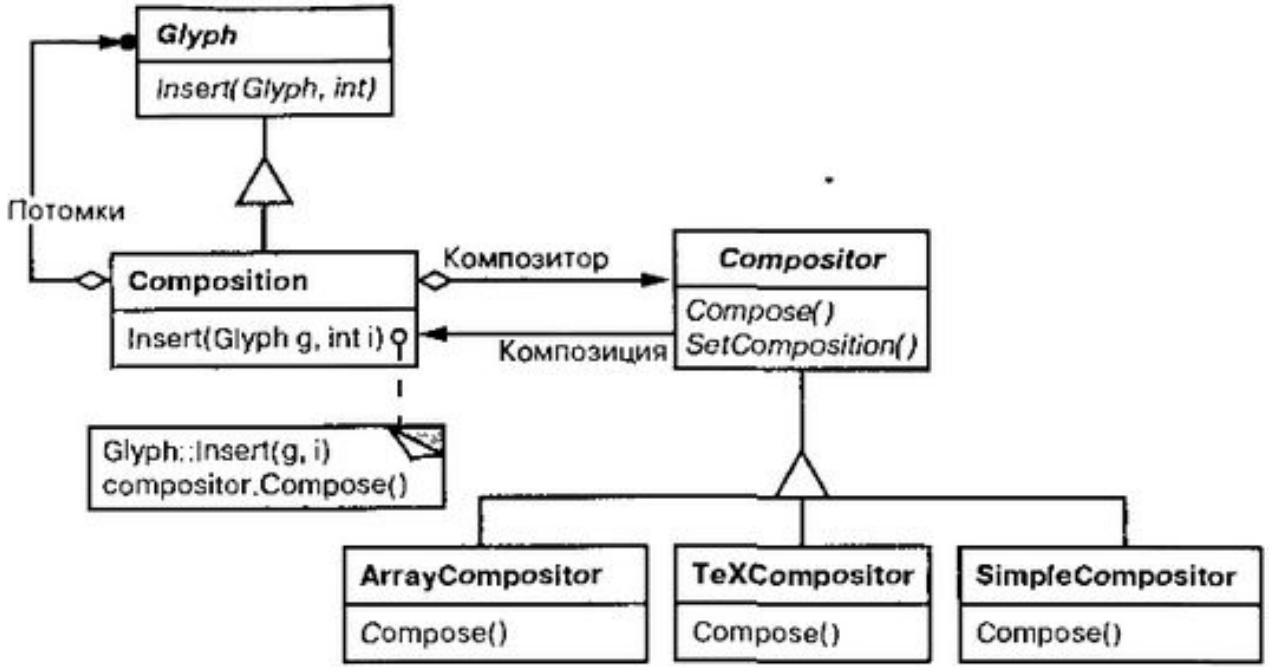
Декоратор: особенности

- динамическое добавление (и удаление) обязанностей объектов
  - большая гибкость, чем у наследования
- позволяет избежать перегруженных функциональностью базовых классов
- декорирующий объект != декорируемый объект
- много мелких объектов

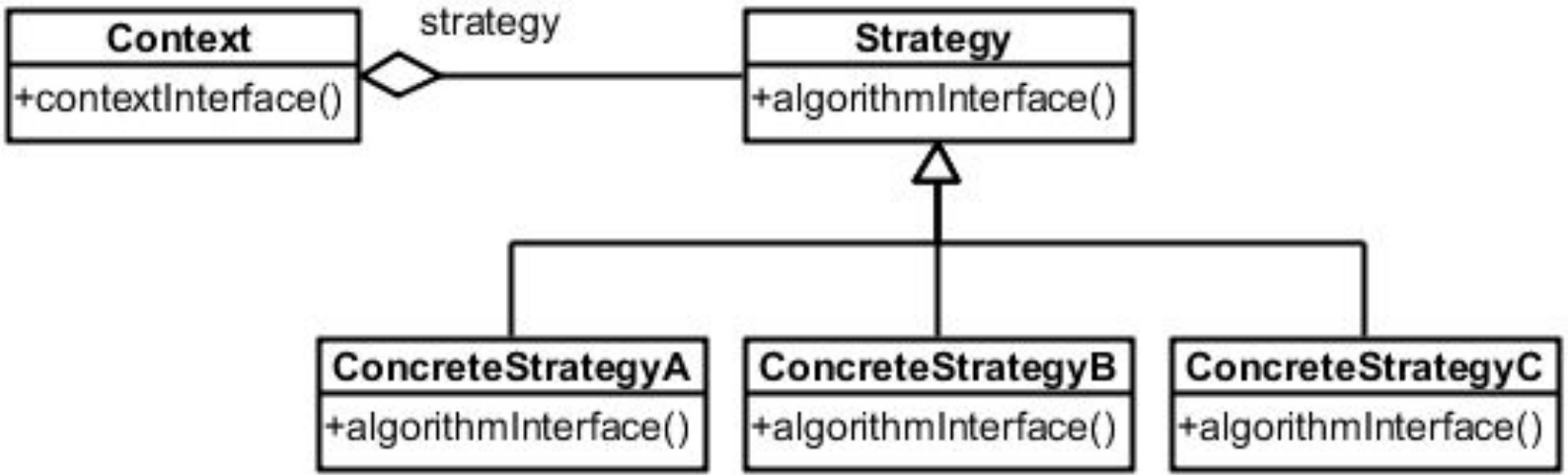
Форматирование текста

- Задача — разбитие текста на строки, колонки и т.д.
- Высокоуровневые параметры форматирования
  - ширина полей, размер отступа, межстрочный интервал и т.д.
- Компромисс между качеством и скоростью работы
- Инкапсуляция алгоритма

Compositor и Composition



Паттерн Стратегия

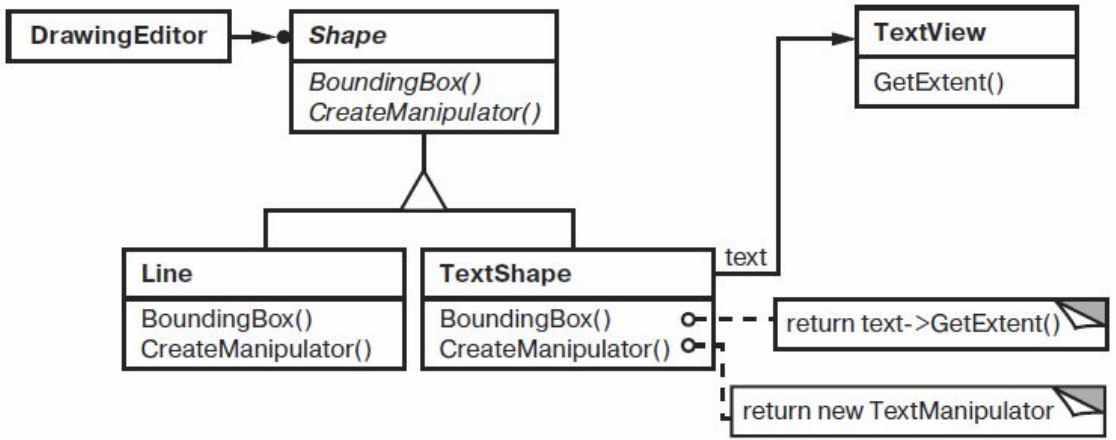


Стратегия: особенности

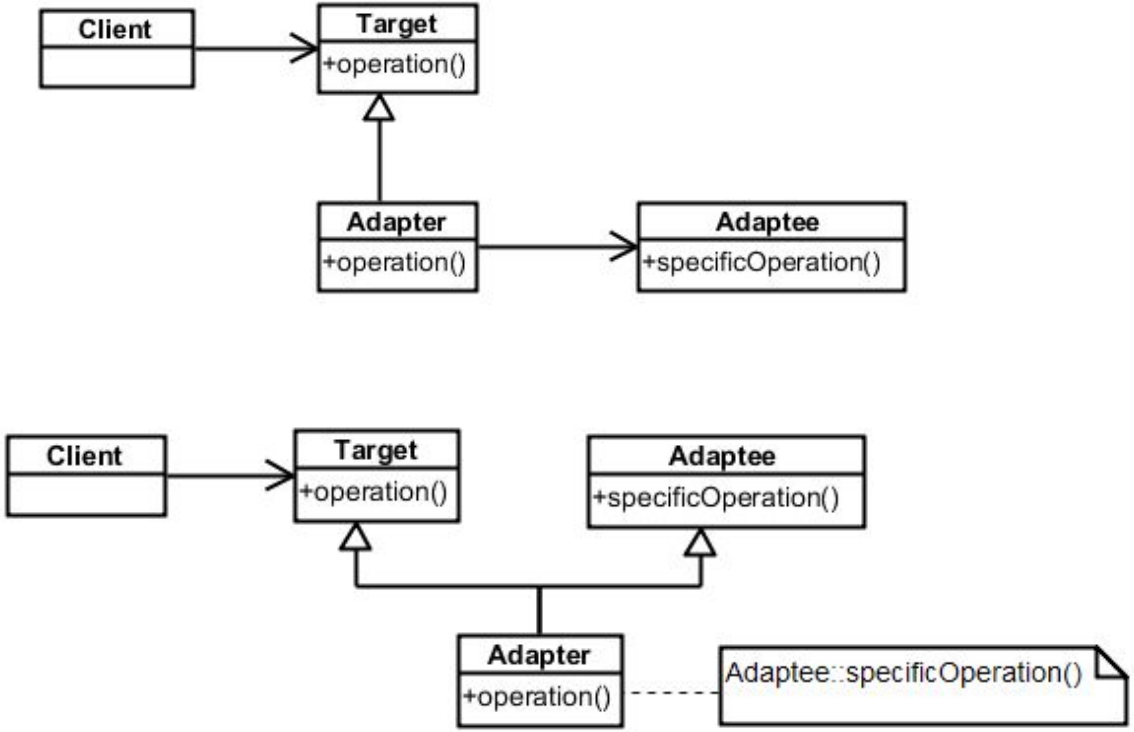
- Назначение — инкапсуляция алгоритма в объект
- Самое важное — спроектировать интерфейсы стратегии и контекста
  - так, чтобы их не менять при каждом чихе
- Применяем, если
  - имеется много родственных классов с разным поведением
  - нужно иметь несколько вариантов алгоритма
  - в алгоритме есть данные, про которые клиенту знать не надо
  - в коде много условных операторов

Проблема неподходящих интерфейсов

- Графический редактор
  - Shape, Line, Polygon, …
- Сторонний класс TextView
  - хотим его реализацию
  - другой интерфейс



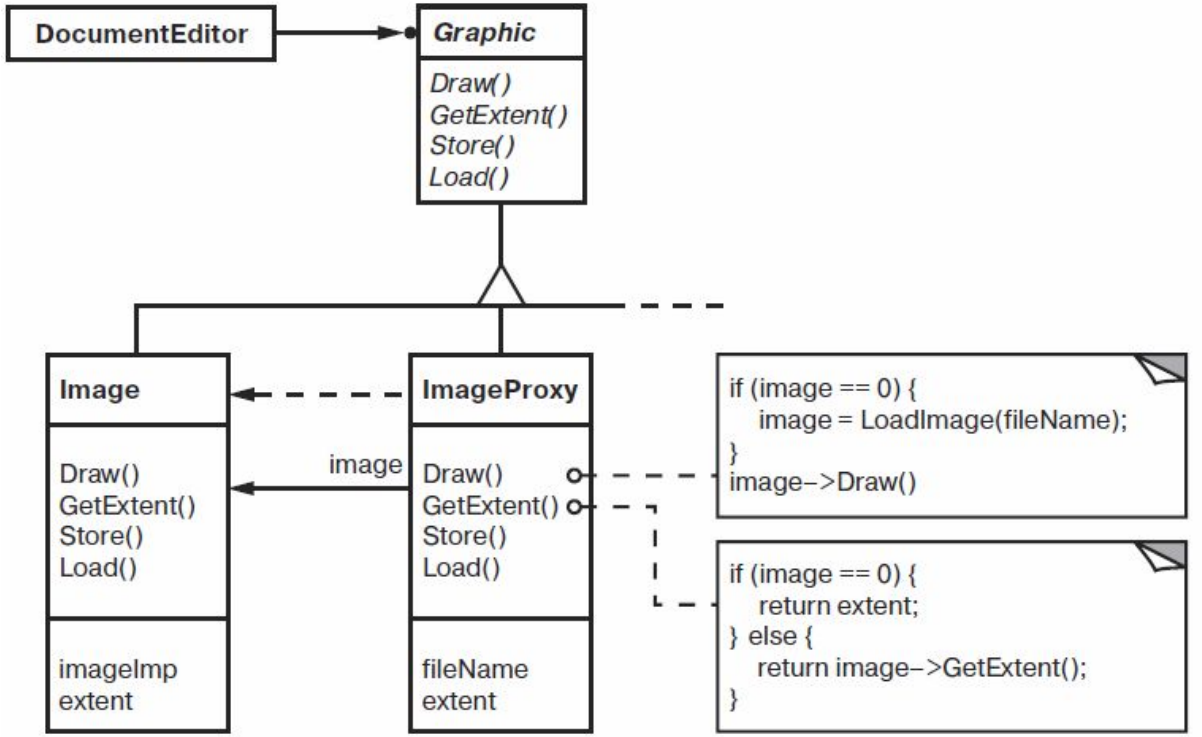
Паттерн Адаптер



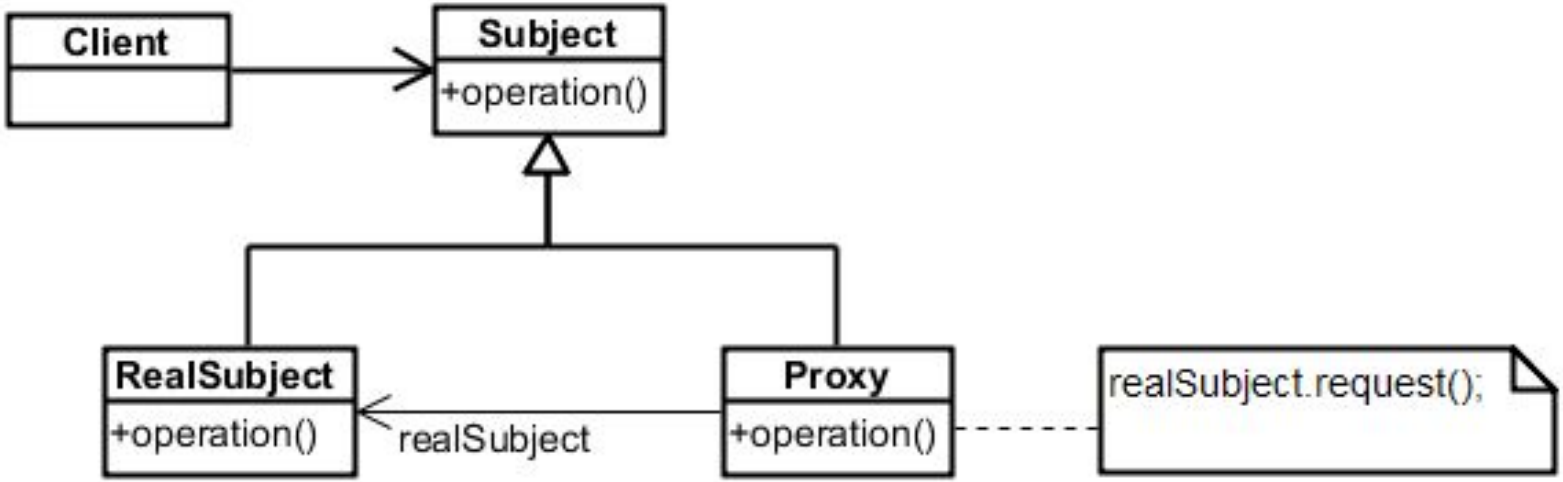
Управление доступом к объектам

- встраивание в документ графических объектов
  - затраты на создание могут быть значительными
  - хотим отложить их на момент использования
- использование заместителей объектов

Отложенная загрузка изображения



Паттерн Прокси



Прокси: особенности

- замещение удалённых объектов
- создание “тяжёлых” объектов по требованию
- контроль доступа
- умные указатели
  - подсчёт ссылок
  - ленивая загрузка/инициализация
  - работа с блокировками
  - копирование при записи

Снижение зависимости между подсистемами

Фасад: особенности

- простой интерфейс к сложной системе
- отделение подсистем от клиента и друг от друга
- многоуровневая архитектура