

Лекция 6: Структурные шаблоны

Юрий Литвинов
yurii.litvinov@gmail.com

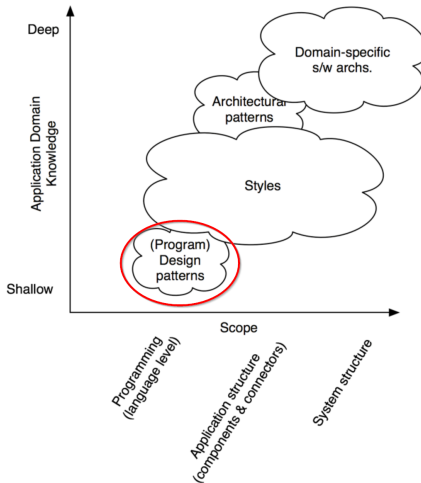
21.04.2022

Паттерны проектирования

Шаблон проектирования — это повторяемая архитектурная конструкция, являющаяся решением некоторой типичной технической проблемы

- ▶ Подходит для класса проблем
- ▶ Обеспечивает переиспользуемость знаний
- ▶ Позволяет унифицировать терминологию
- ▶ В удобной для изучения форме
- ▶ НЕ конкретный рецепт или указания к действию

Паттерны и архитектурные стили



© N. Medvidovic

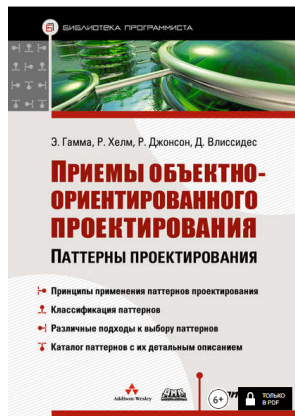
Книжка про паттерны

Must read!

Приемы объектно-ориентированного проектирования. Паттерны проектирования

Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес

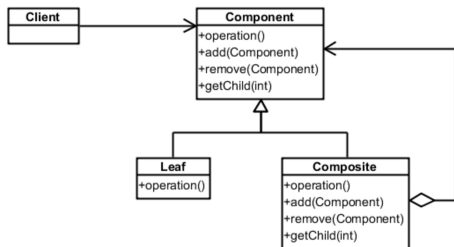
Design Patterns: Elements of Reusable Object-Oriented Software



Паттерн "Компоновщик"

Composite

- ▶ Представление иерархии объектов вида часть-целое
- ▶ Единообразная обработка простых и составных объектов
- ▶ Простота добавления новых компонентов
- ▶ Пример:
 - ▶ Синтаксические деревья



“Компоновщик” (Composite), детали реализации

- ▶ Ссылка на родителя
 - ▶ Может быть полезна для простоты обхода
 - ▶ “Цепочка обязанностей”
 - ▶ Но дополнительный инвариант
 - ▶ Обычно реализуется в Component
- ▶ Разделяемые поддеревья и листья
 - ▶ Позволяют сильно экономить память
 - ▶ Проблемы с навигацией к родителям и разделяемым состоянием
 - ▶ Паттерн “Приспособленец”
- ▶ Идеологические проблемы с операциями для работы с потомками
 - ▶ Не имеют смысла для листа
 - ▶ Можно считать Leaf Composite-ом, у которого всегда 0 потомков
 - ▶ Операции add и remove можно объявить и в Composite, тогда придётся делать cast
 - ▶ Иначе надо бросать исключения в add и remove

“Компоновщик”, детали реализации (2)

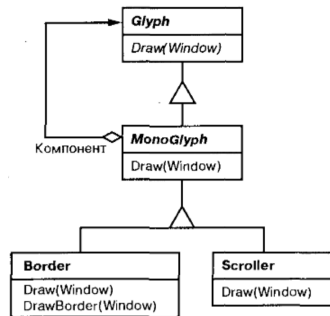
- ▶ Операция `getComposite()` – более аккуратный аналог `cast-a`
- ▶ Порядок потомков может быть важен, может нет
- ▶ Кеширование информации для обхода или поиска
 - ▶ Например, кеширование ограничивающих прямоугольников для фрагментов картинки
 - ▶ Инвалидация кеша
- ▶ Удаление потомков
 - ▶ Если нет сборки мусора, то лучше в `Composite`
 - ▶ Следует опасаться разделяемых листьев/поддеревьев

Усовершенствование UI

- ▶ Хотим сделать рамку вокруг текста и полосы прокрутки, отключаемые по опции
- ▶ Желательно убирать и добавлять элементы оформления так, чтобы другие объекты даже не знали, что они есть
- ▶ Хотим менять во время выполнения — наследование не подойдёт
 - ▶ Наш выбор — композиция
 - ▶ Прозрачное оформление

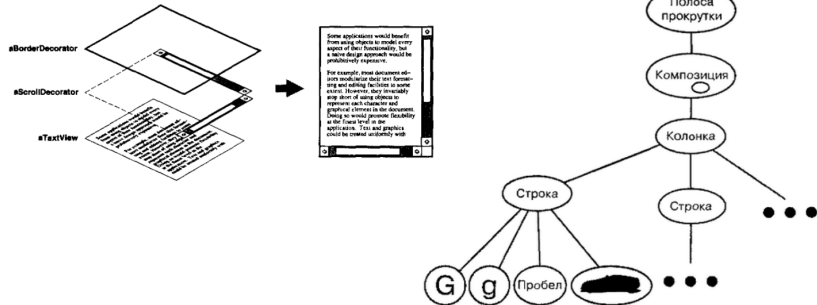
Моноглиф

- ▶ Абстрактный класс с ровно одним сыном
 - ▶ Вырожденный случай компоновщика
- ▶ “Обрамляет” сына, добавляя новую функциональность



© Э. Гамма и др., Приемы
объектно-ориентированного
проектирования

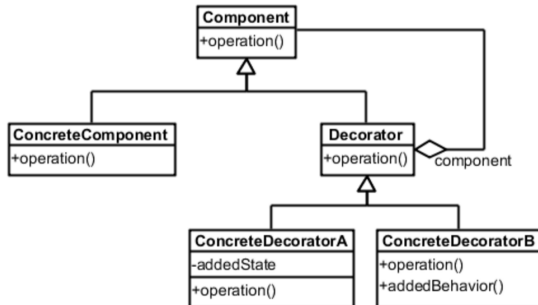
Структура глифов



© Э. Гамма и др., Приемы объектно-ориентированного проектирования

Паттерн “Декоратор”

Decorator

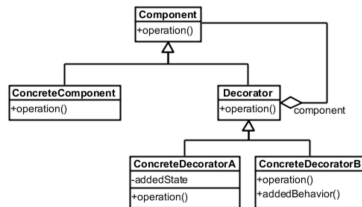


Декоратор, особенности

- ▶ Динамическое добавление (и удаление) обязанностей объектов
 - ▶ Большая гибкость, чем у наследования
- ▶ Позволяет избежать перегруженных функциональностью базовых классов
- ▶ Много мелких объектов

“Декоратор” (Decorator), детали реализации

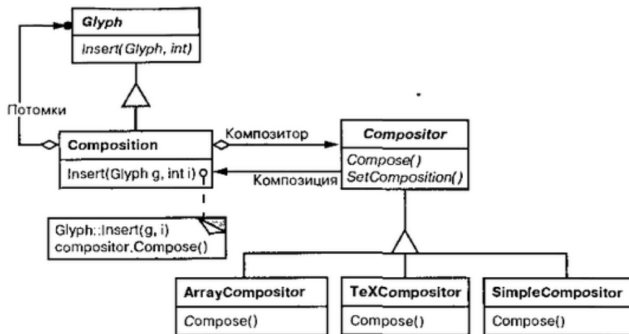
- ▶ Интерфейс декоратора должен соответствовать интерфейсу декорируемого объекта
 - ▶ Иначе получится “Адаптер”
- ▶ Если конкретный декоратор один, абстрактный класс можно не делать
- ▶ Component должен быть по возможности небольшим (в идеале, интерфейсом)
 - ▶ Иначе лучше паттерн “Стратегия”
 - ▶ Или самодельный аналог, например, список “расширений”, которые вызываются декорируемым объектом вручную перед операцией или после неё



Форматирование текста

- ▶ Задача — разбиение 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

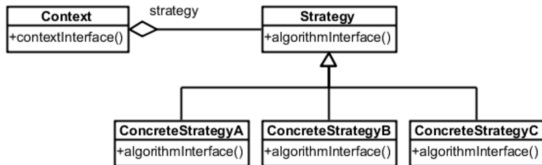


© Э. Гамма и др., Приемы объектно-ориентированного проектирования

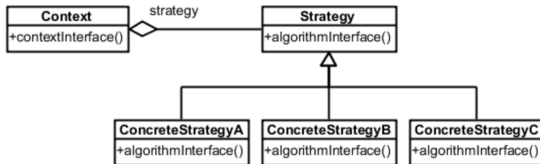
Паттерн “Стратегия”

Strategy

- ▶ Назначение — инкапсуляция алгоритма в объект
- ▶ Самое важное — спроектировать интерфейсы стратегии и контекста
 - ▶ Так, чтобы не менять их для каждой стратегии
- ▶ Применяется, 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



“Стратегия” (Strategy), детали реализации



- ▶ Передача контекста вычислений в стратегию
 - ▶ Как параметры метода — уменьшает связность, но некоторые параметры могут быть стратегии не нужны
 - ▶ Передавать сам контекст в качестве аргумента — в **Context** интерфейс для доступа к данным

“Стратегия” (Strategy), детали реализации (2)

- ▶ Стратегия может быть параметром шаблона
 - ▶ Если не надо её менять на лету
 - ▶ Не надо абстрактного класса и нет оверхеда на вызов виртуальных методов
- ▶ Стратегия по умолчанию
 - ▶ Или просто поведение по умолчанию, если стратегия не установлена
- ▶ Объект-стратегия может быть приспособленцем