Проектирование ПО

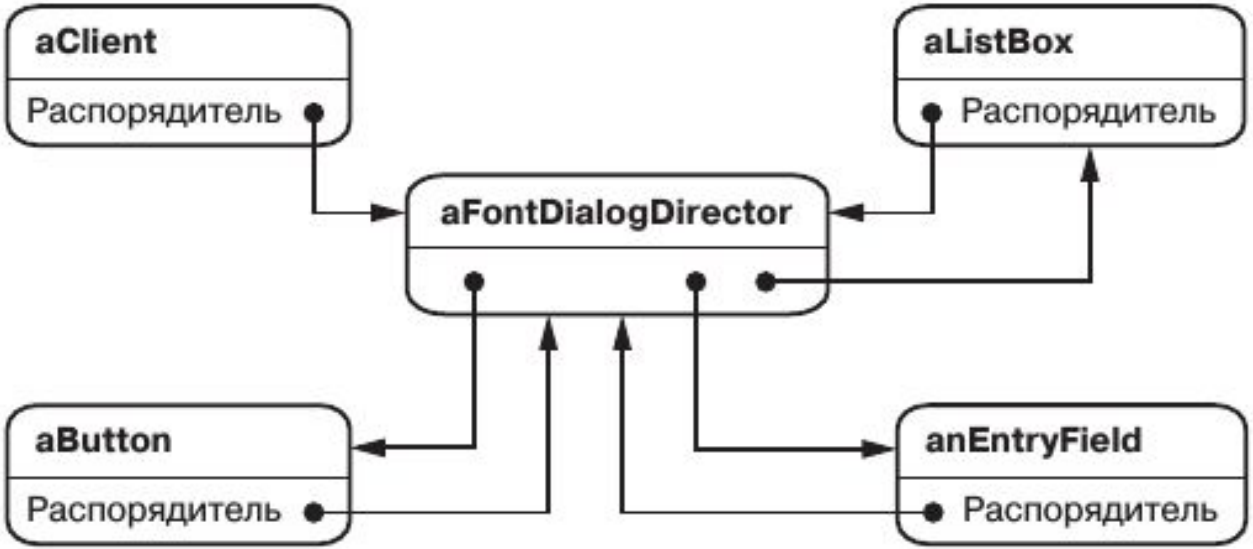
Лекция 11: Поведенческие шаблоны

Тимофей Брыксин
timofey.bryksin@gmail.com

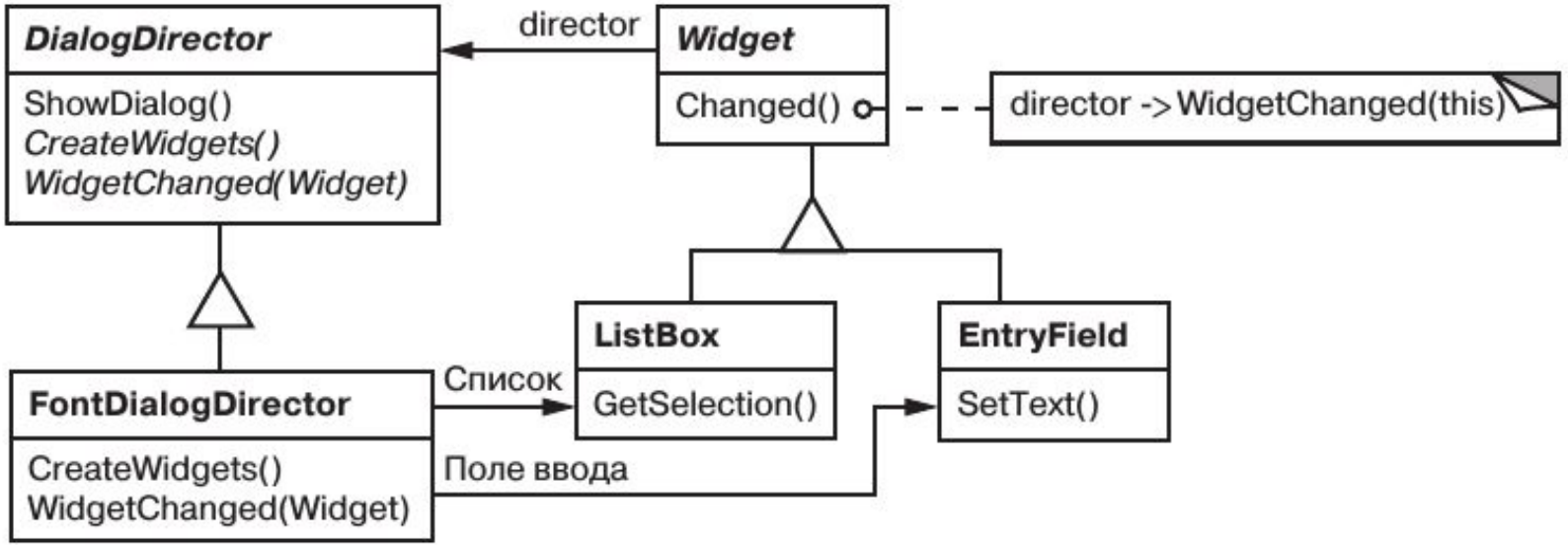
Посредник: мотивация

- большое количество связей между объектами
- объекты знают слишком много
- снижается переиспользуемость

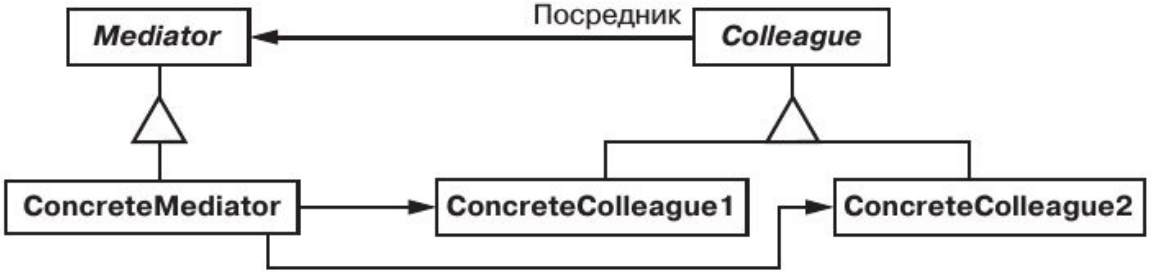
Немного централизации

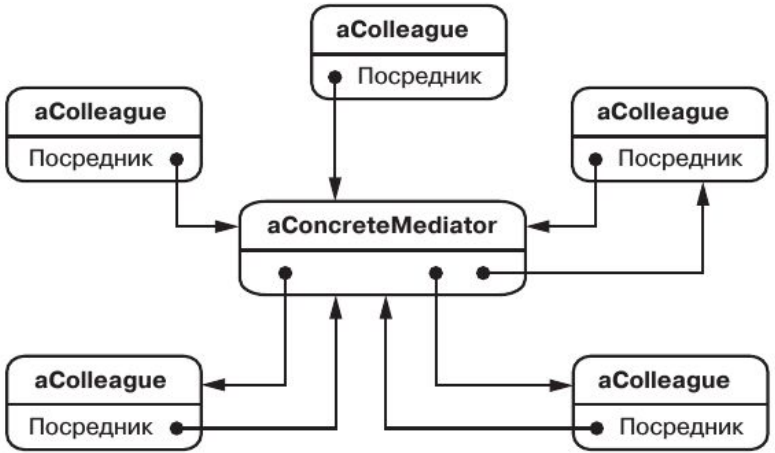


Архитектура решения



Паттерн Посредник





Достоинства и недостатки

- устраняет связанность между классами-коллегами
- повышает переиспользуемость классов-коллег
- упрощает протоколы взаимодействия объектов
- абстрагирует способ кооперирования объектов
- централизует управление (мини-God Object)

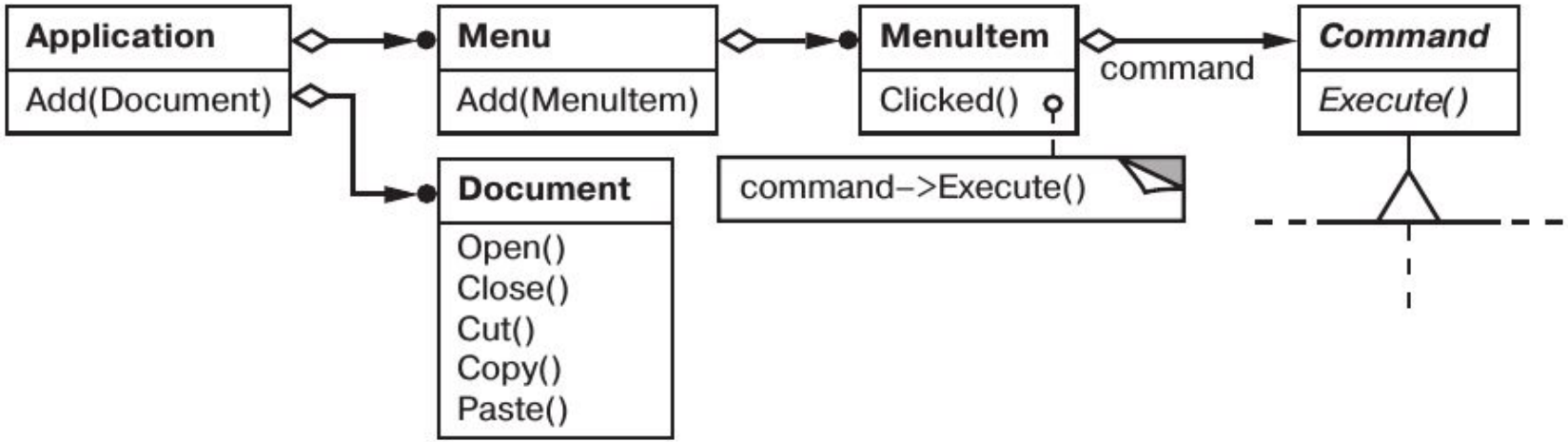
Посредник: когда использовать

- куча объектов со сложными взаимосвязями
- из-за этого их сложно переиспользовать
- поведение хочется уметь настраивать без кучи подклассов

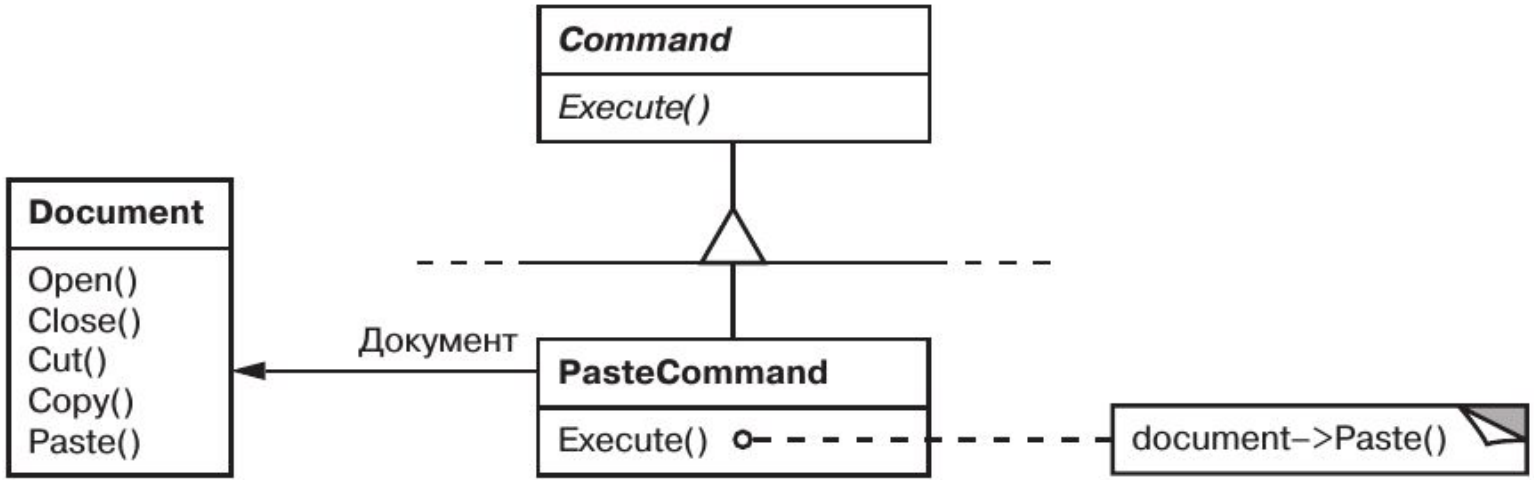
Команда: мотивация

- хотим посылать объектам запросы
- какая операция запрошена, знать не хотим
- и кто выполнит запрос тоже

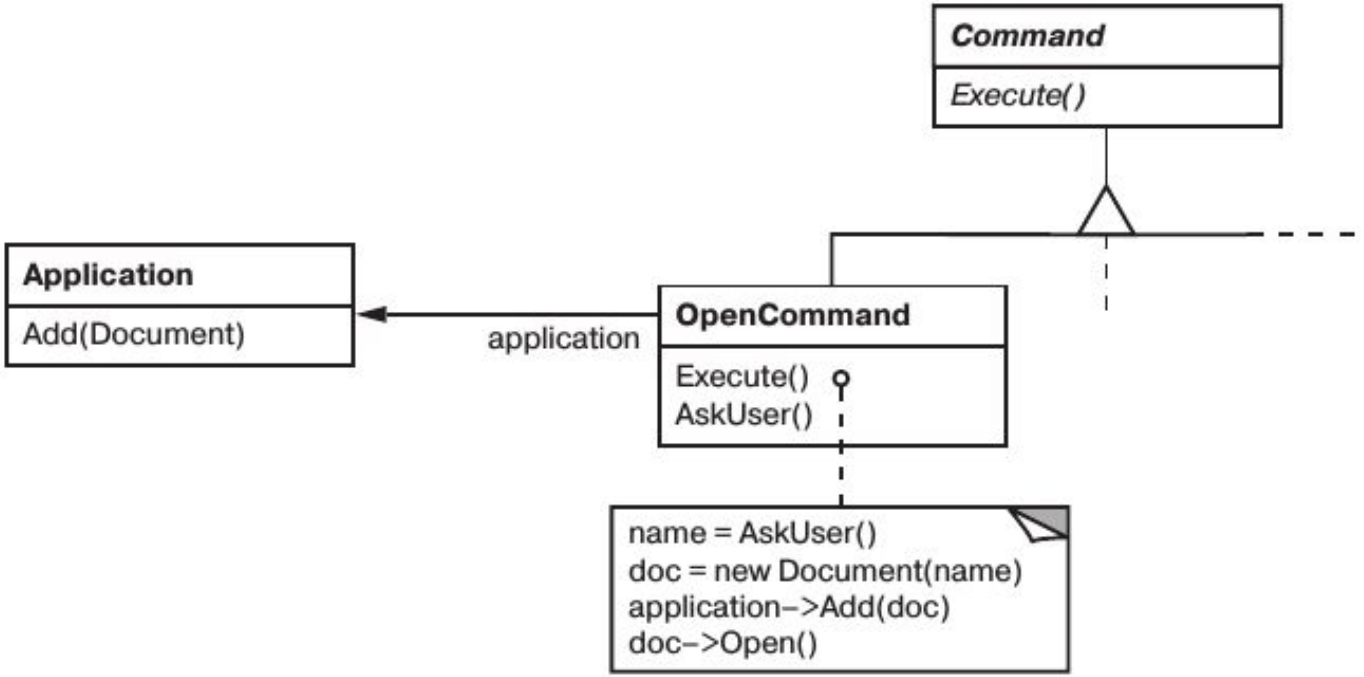
Операцию в объект!



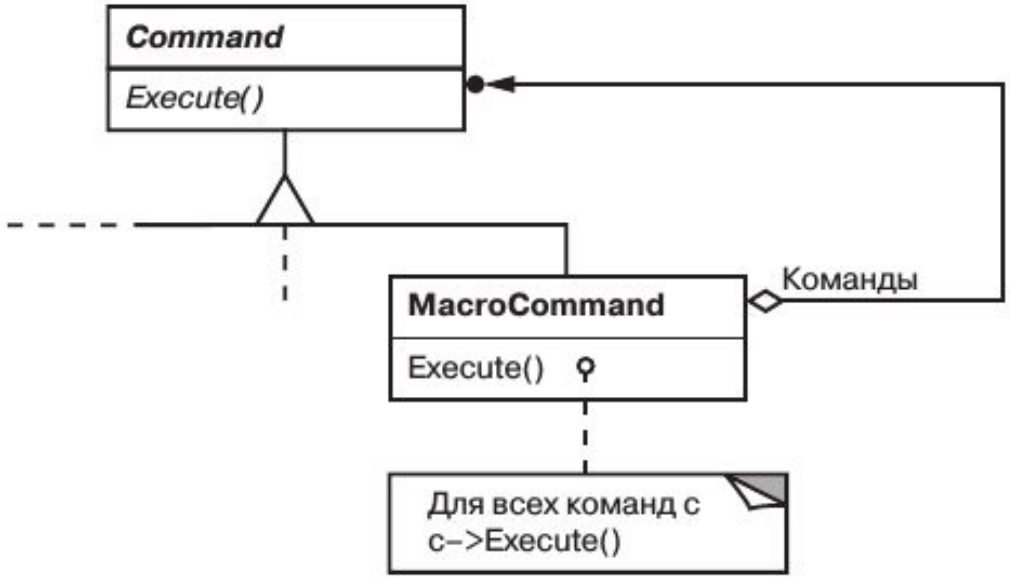
Операция вставки



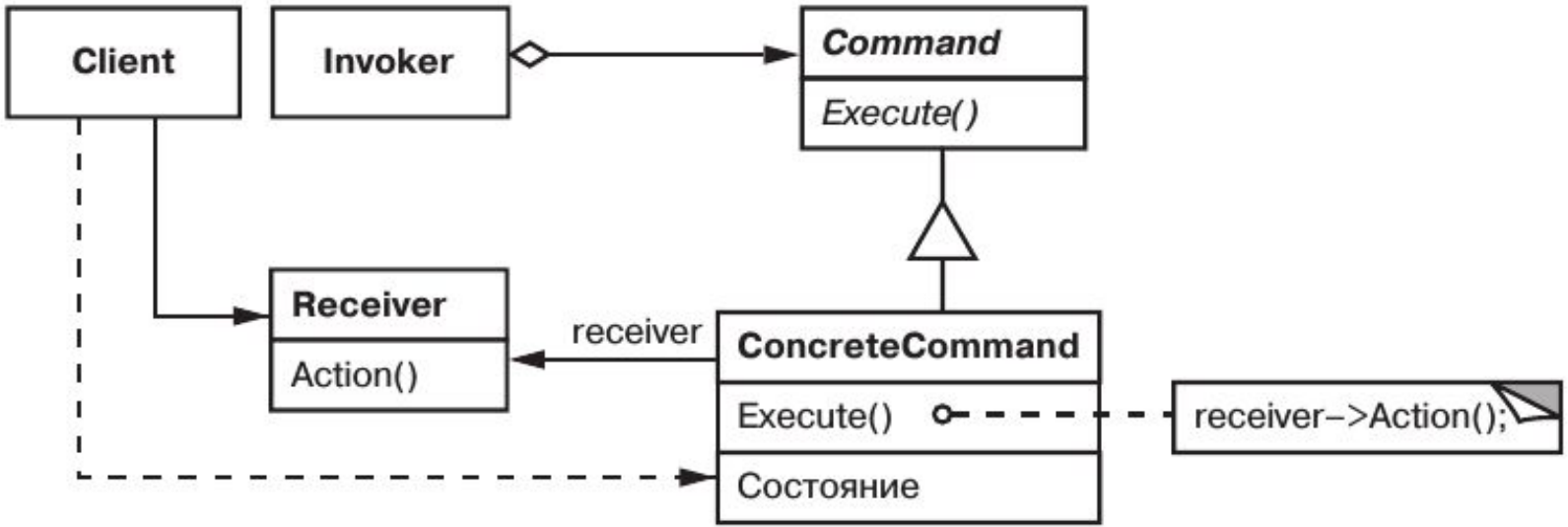
Операция открытия документа



Составная операция



Паттерн Команда

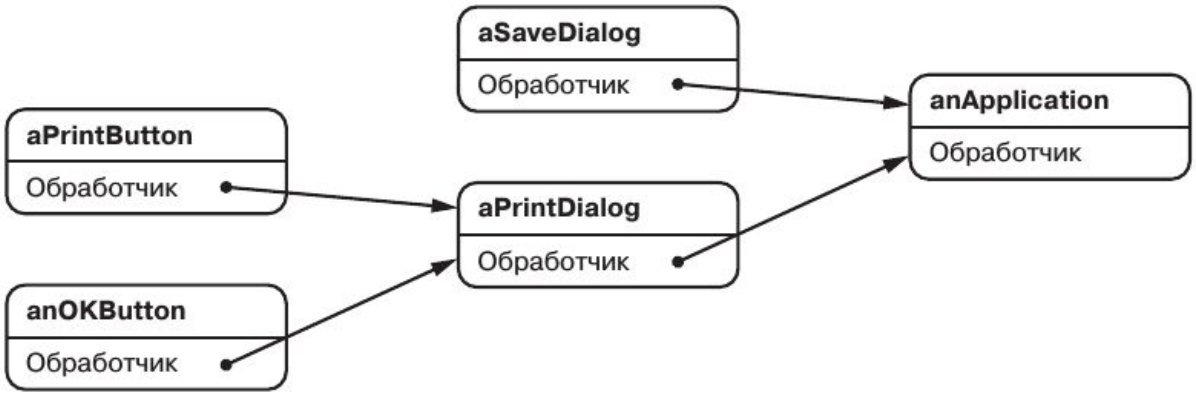


Команда: когда применять

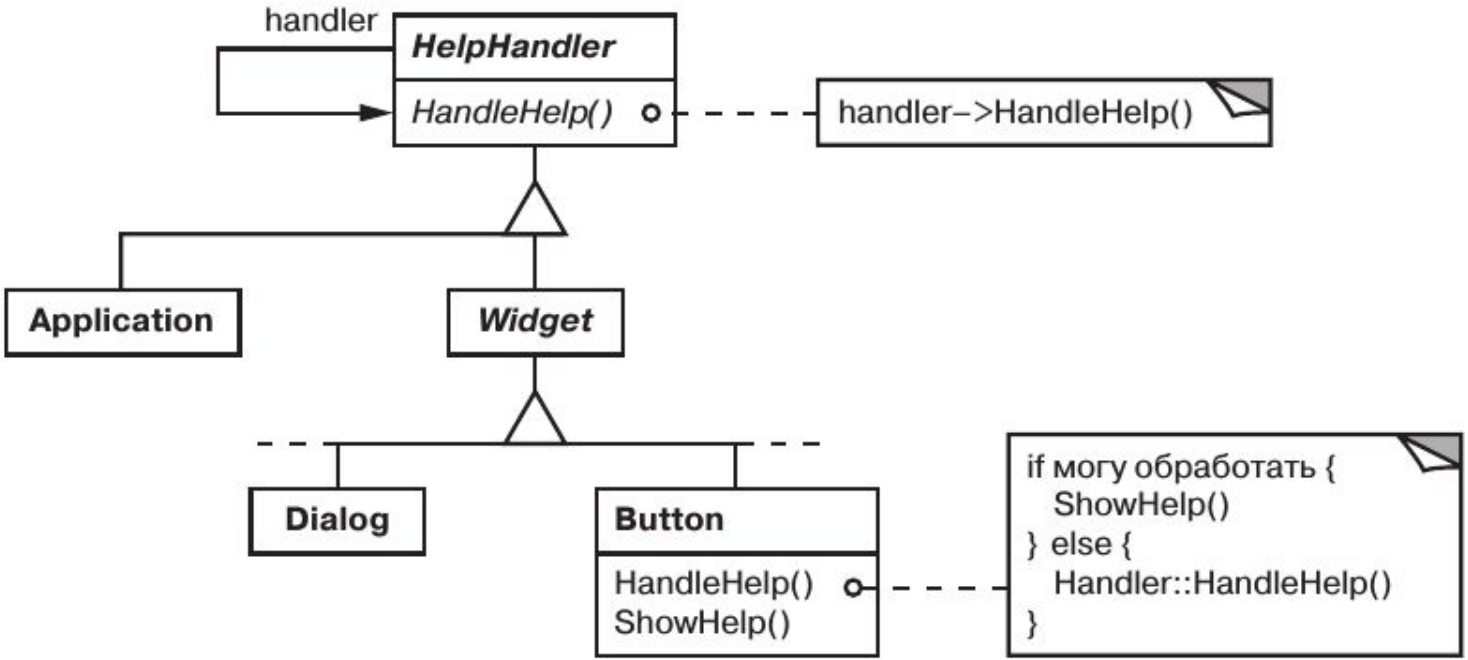
- параметризовать объекты выполняемым действием
- определять, ставить в очередь и выполнять запросы в разное время
- поддержать отмену операций
- структурировать систему на основе высокоуровневых операций, построенных из примитивных
- поддержать протоколирование изменений

Цепочка ответственности: пример

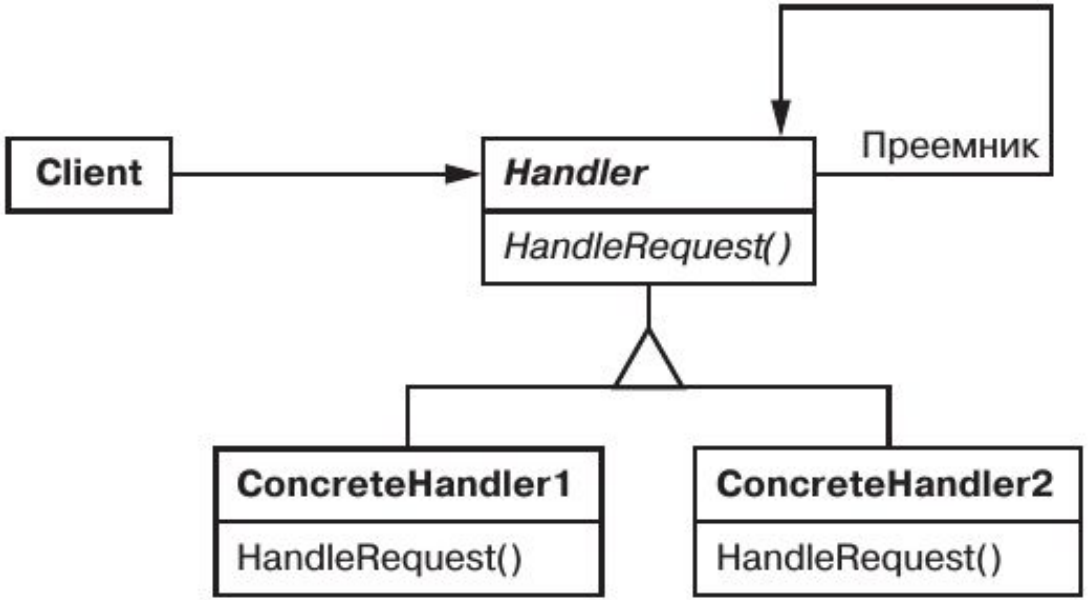
- организация контекстной справки
- заранее неизвестно, кто в итоге обработает запрос
- запрос передаётся по цепочке



Архитектура решения



Паттерн Цепочка ответственности



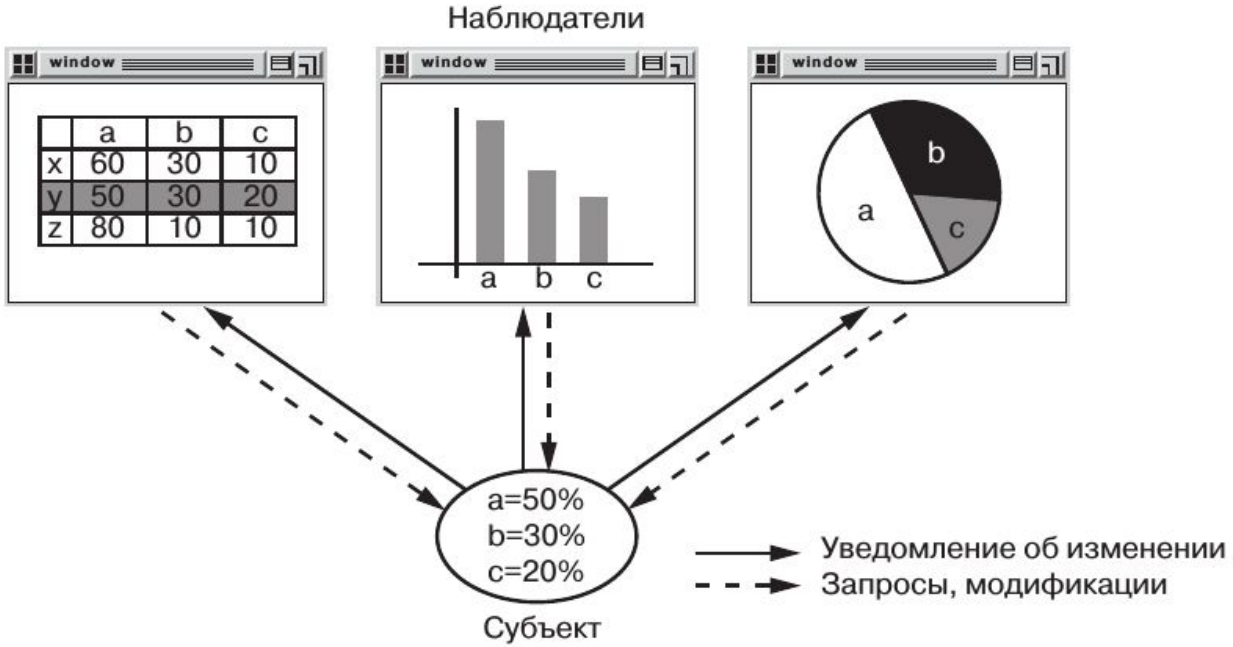
Цепочка ответственности: плюсы и минусы

- ослабление связанности
- дополнительная гибкость при распределении обязанностей
- получение не гарантировано

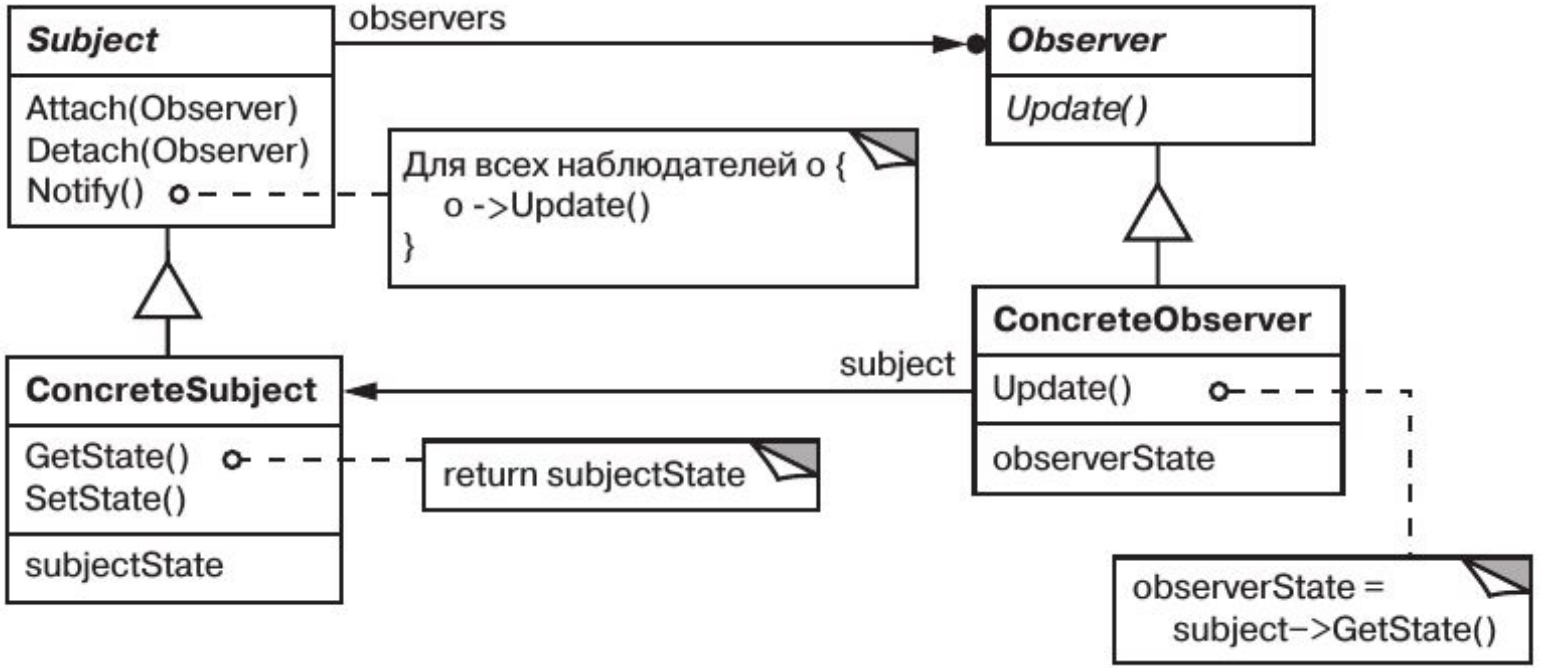
Когда использовать:

- есть более одного объекта-обработчика запросов
- конечный обработчик неизвестен и должен быть найден автоматически
- хотим отправить запрос нескольким объектам
- обработчики могут задаваться динамически

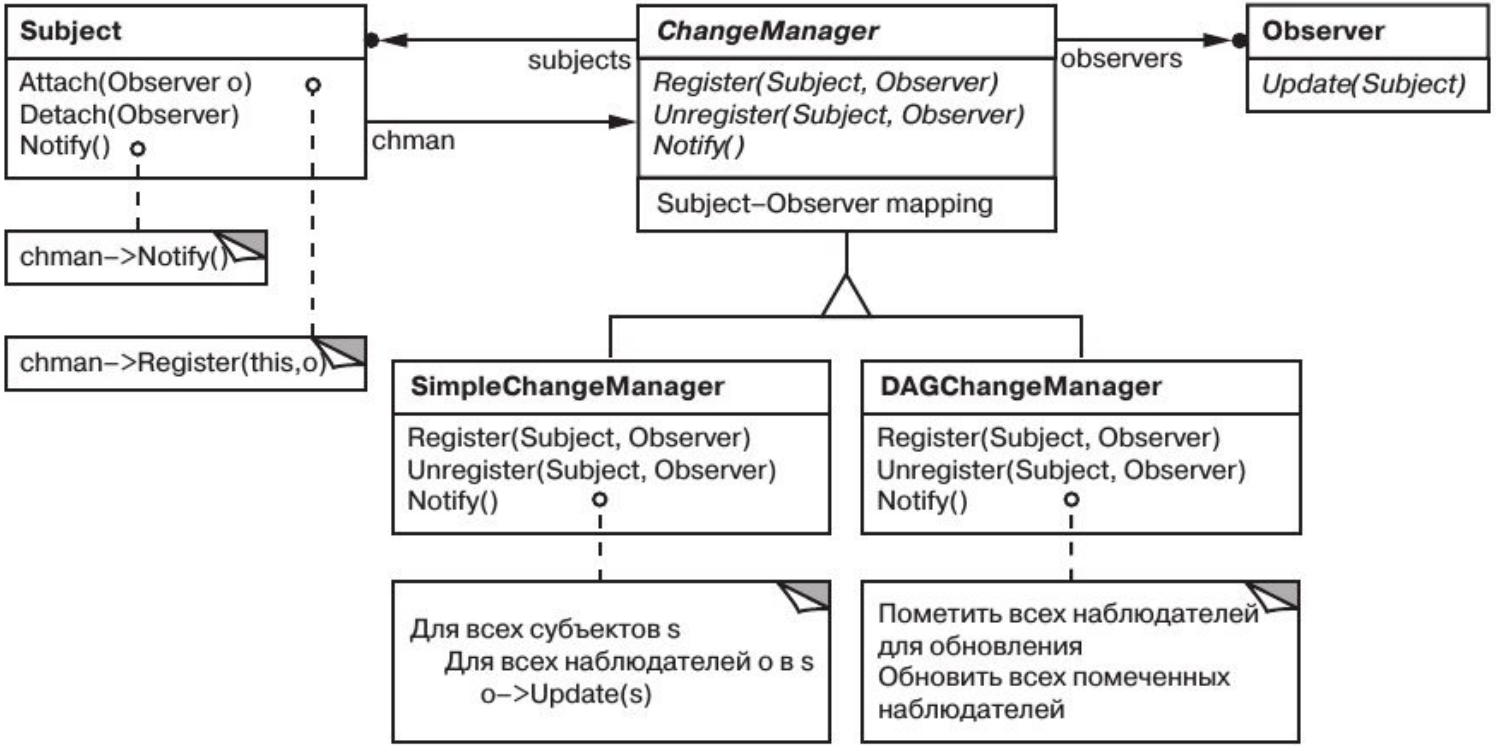
Наблюдатель: пример



Наблюдатель: архитектура



Координация изменений

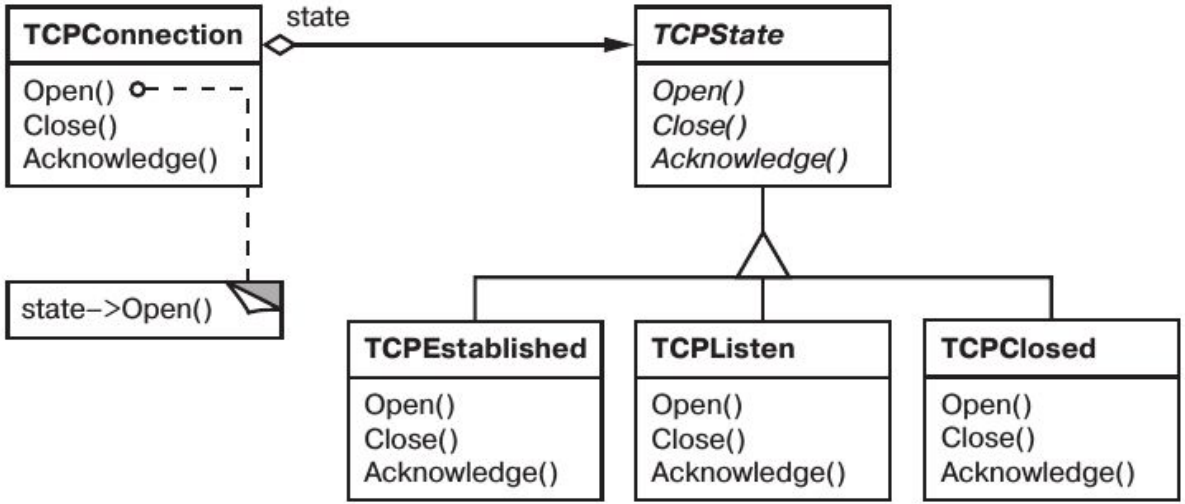


Наблюдатель: плюсы и минусы

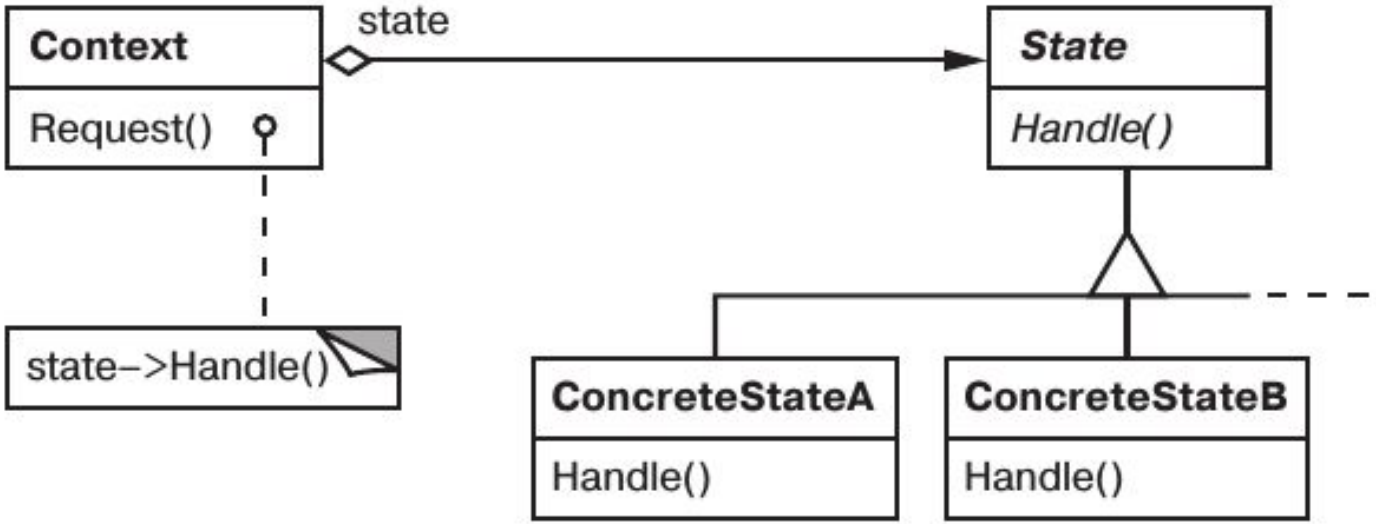
- абстрактная связанность субъекта и наблюдателя
- поддержка широковещательных коммуникаций
- неожиданные обновления

Состояние: пример

- объектно-ориентированный вариант реализации конечных автоматов
  - объект имеет ряд состояний
  - в зависимости от состояний поведение меняется



Состояние: общий вид



Состояние: результаты

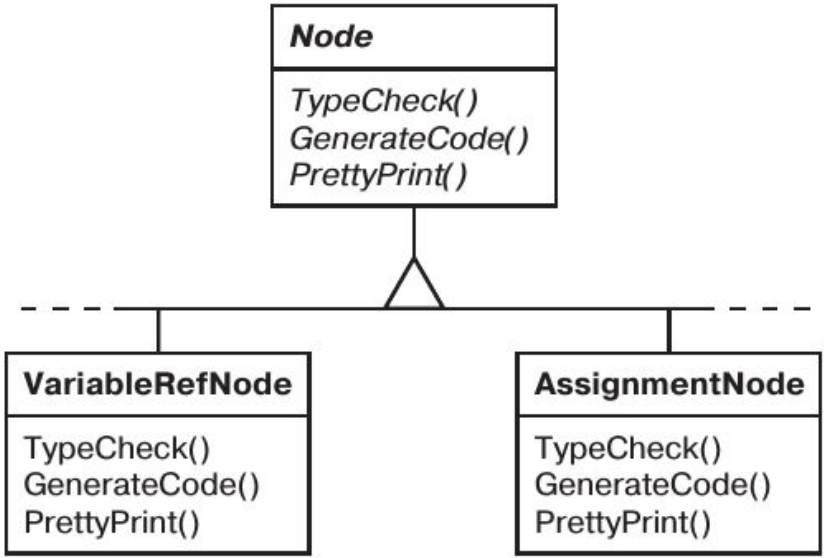
- локализует зависящее от состояния поведение
- делает явными переходы между состояниями
- объекты состояния можно разделять

Когда применять:

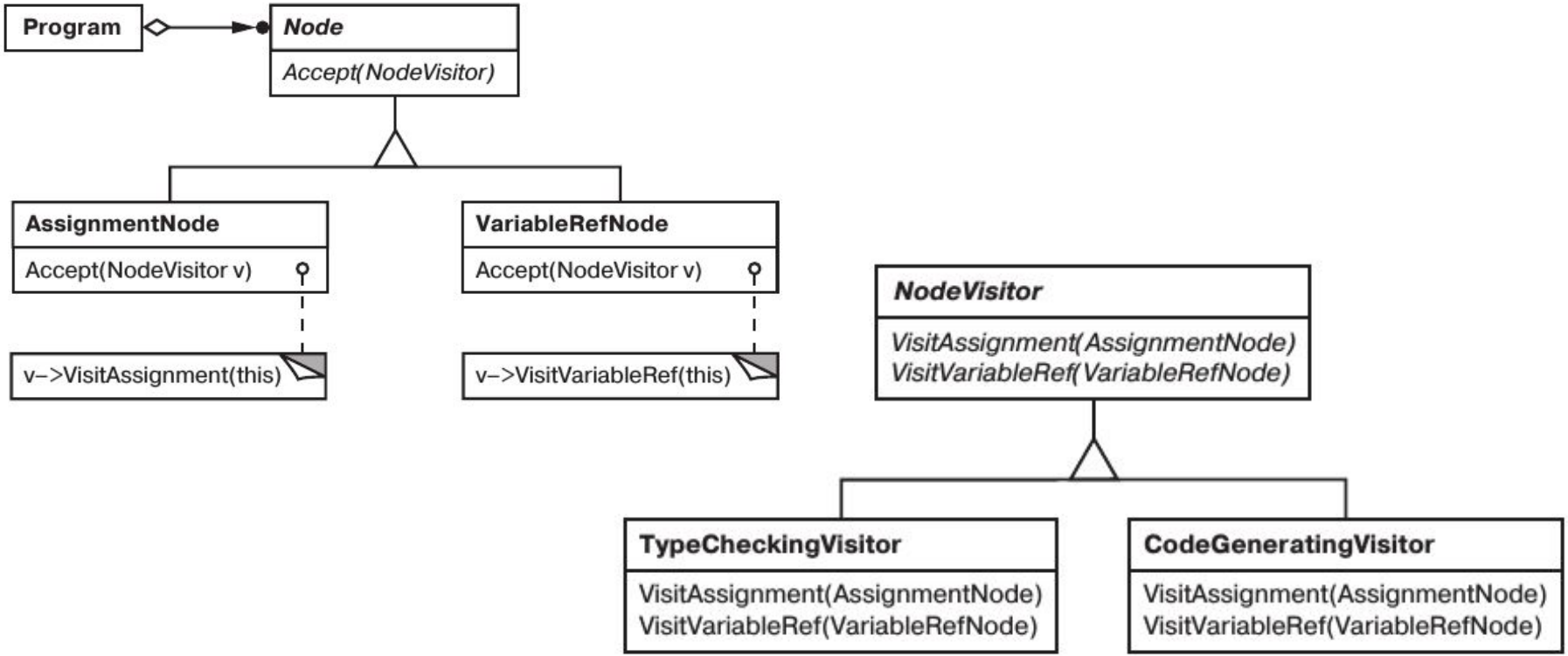
- поведение объекта зависит от его состояния и должно изменяться во время выполнения
- обилие условных операторов, в которых выбор ветви зависит от состояния

Посетитель: пример

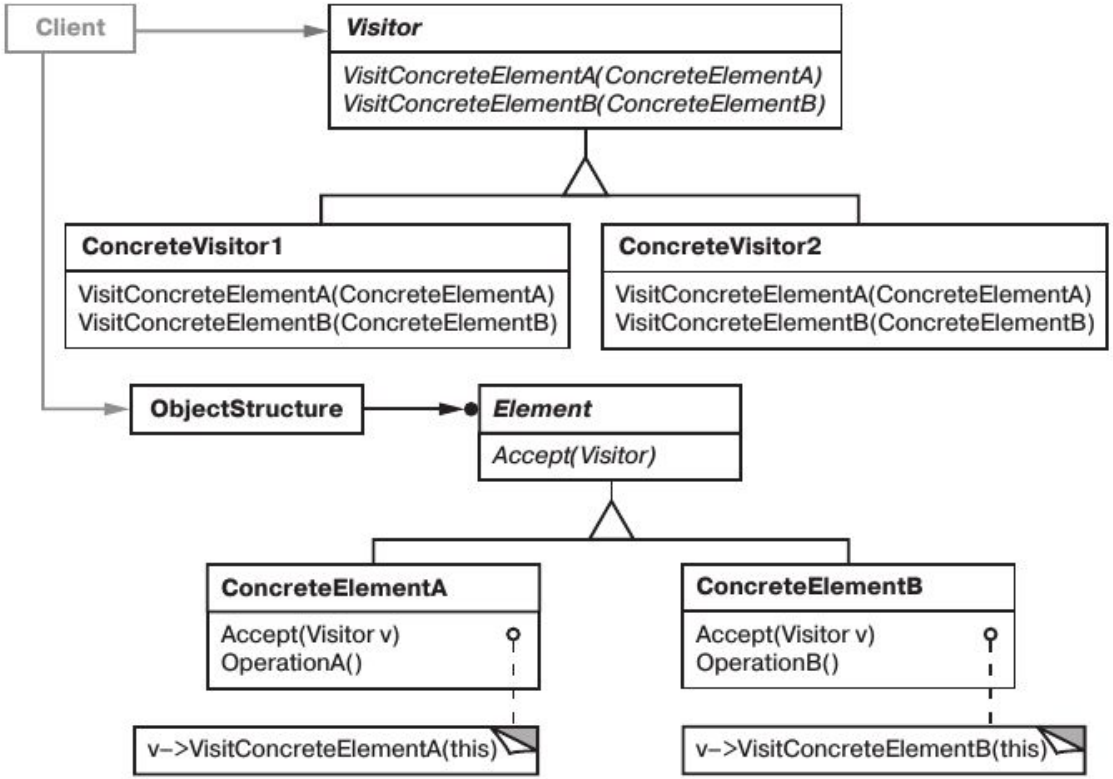
- синтаксическое дерево разбора
- много разных типов узлов
- много алгоритмов их обработки



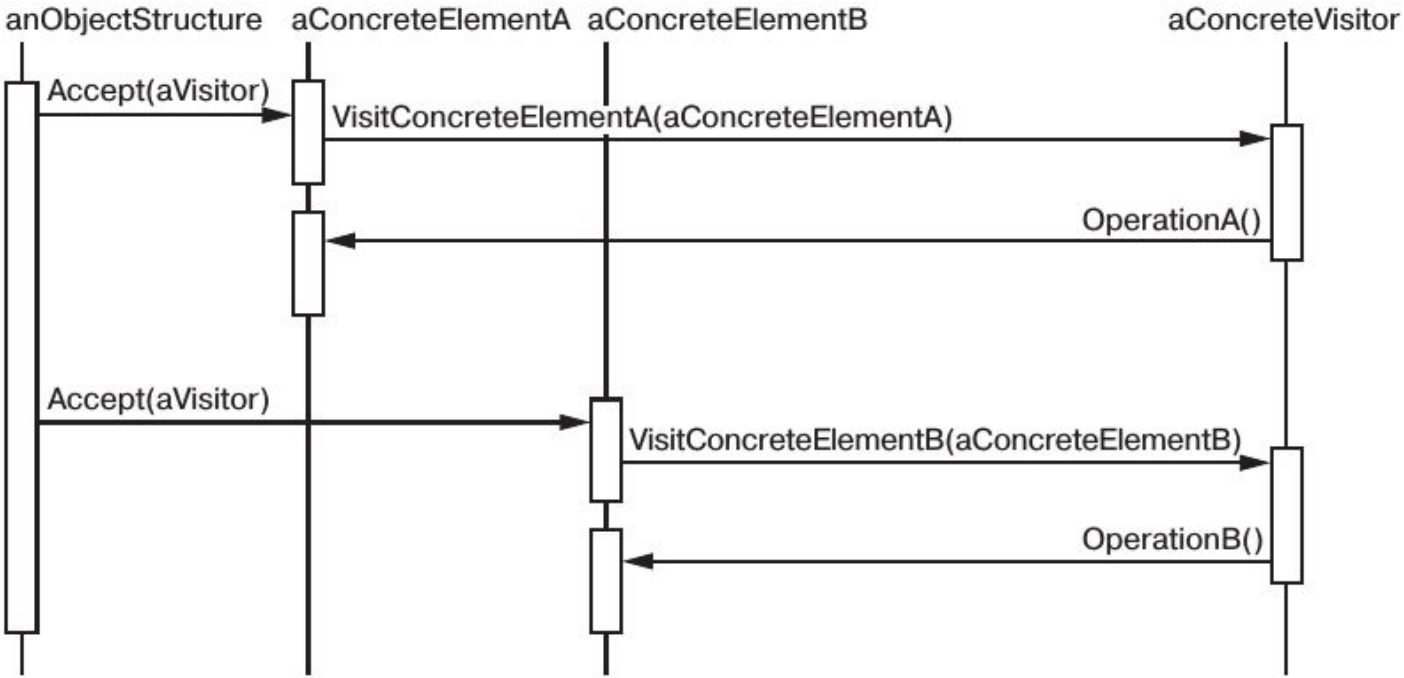
Предлагаемое решение



Паттерн Посетитель



Двойная диспетчеризация



Посетитель: плюсы и минусы

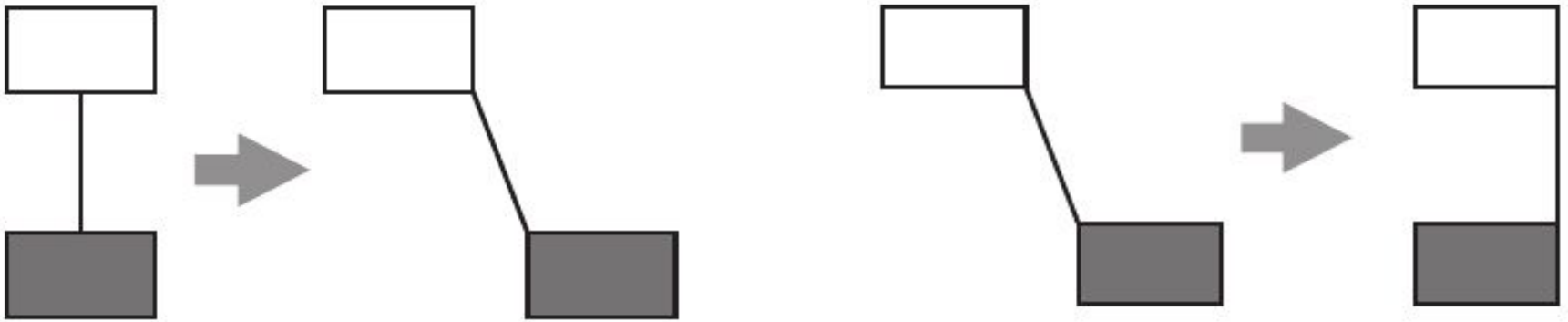
- упрощает добавление новых операций
- объединяет родственные операции
- добавление новых классов ConcreteElement затруднено
- аккумулирование состояния
- нарушение инкапсуляции

Посетитель: когда использовать

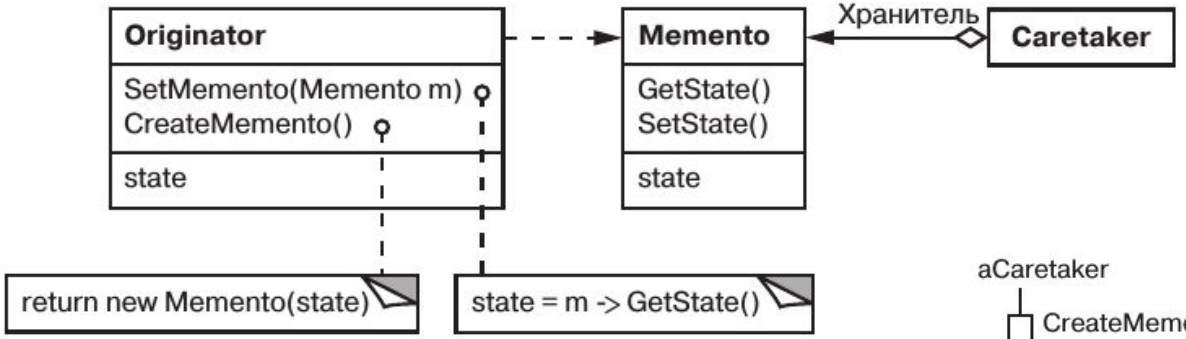
- много классов с разными интерфейсами, хотим выполнять над ними операции в зависимости от типов
- хочется вынести обработку операций из классов
- состав и структура классов меняется редко, а алгоритмы над ними часто

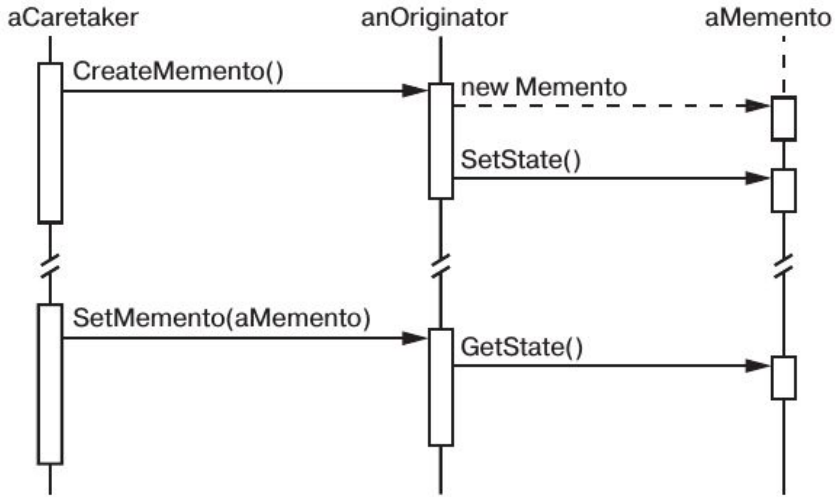
Хранитель: пример

- хотим уметь фиксировать внутреннее состояние объектов



Паттерн Хранитель





Хранитель: особенности

- сохранение границ инкапсуляции
- упрощение структуры сериализуемого объекта
- возможные издержки при использовании хранителей

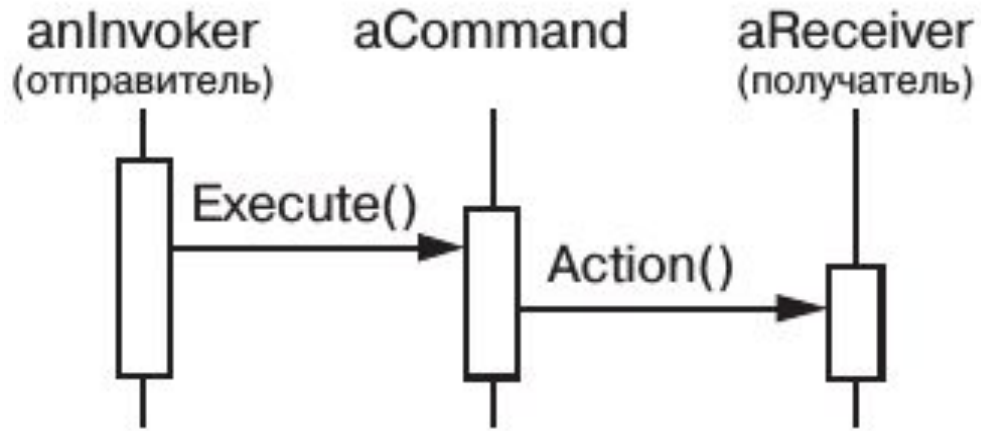
Когда использовать:

- необходимо сохранить снимок состояния объекта с возможностью последующего восстановления
- прямое получение этого состояния нарушает инкапсуляцию объекта

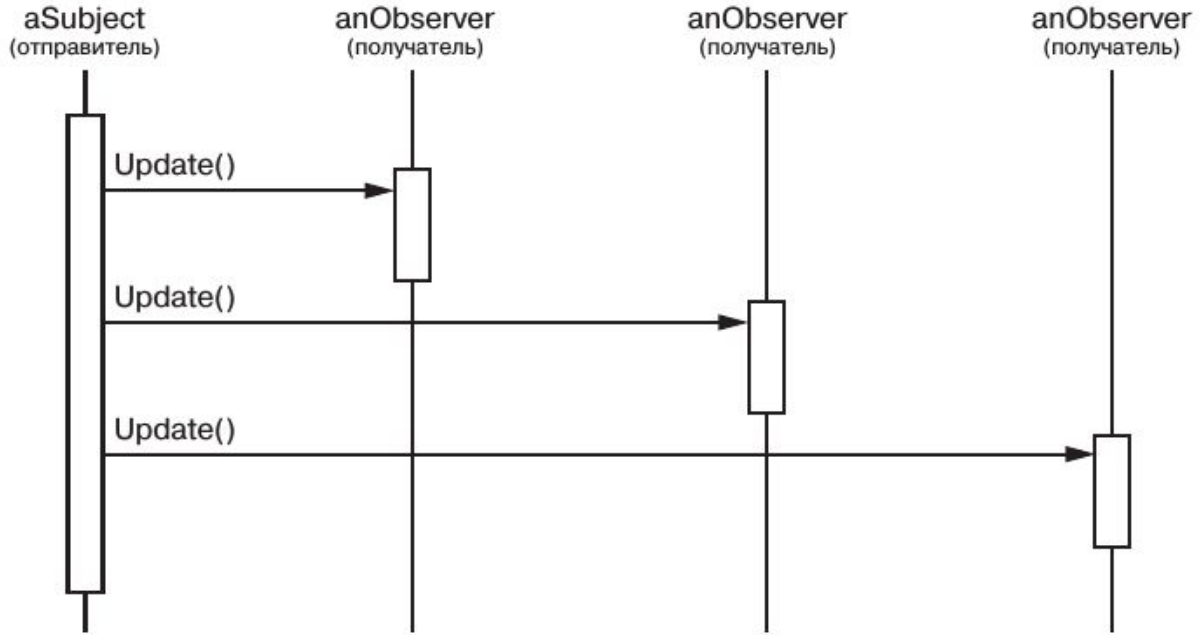
Обсуждение

- организация потока управления, сложно прослеживаемого в run-time
- инкапсуляция причины изменений в объект
  - объект-стратегия
  - объект-состояние
  - объект-посредник
  - объект-итератор
- организация статических и динамических связей между объектами
- объекты как аргументы
- инкапсулированный или распределённый обмен информацией
- паттерны часто хорошо сочетаются друг с другом

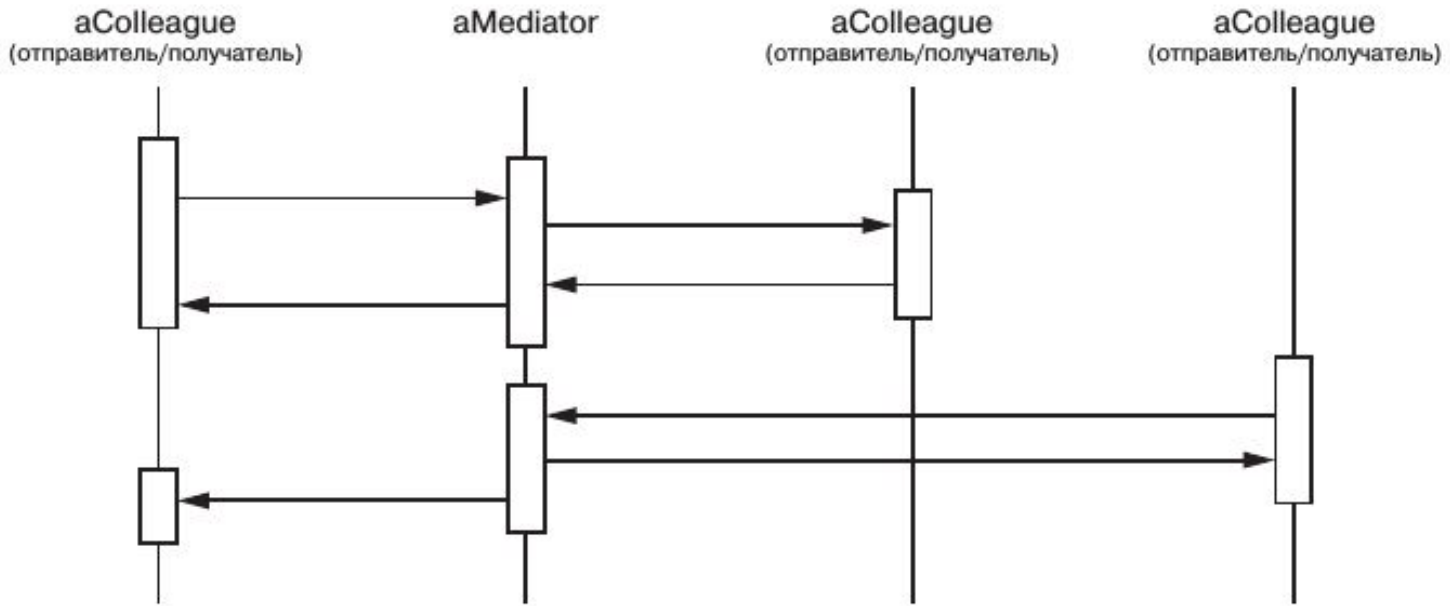
Разделение отправителей и получателей: Команда



Разделение отправителей и получателей: Наблюдатель



Разделение отправителей и получателей: Посредник



Разделение отправителей и получателей: Цепочка обязанностей