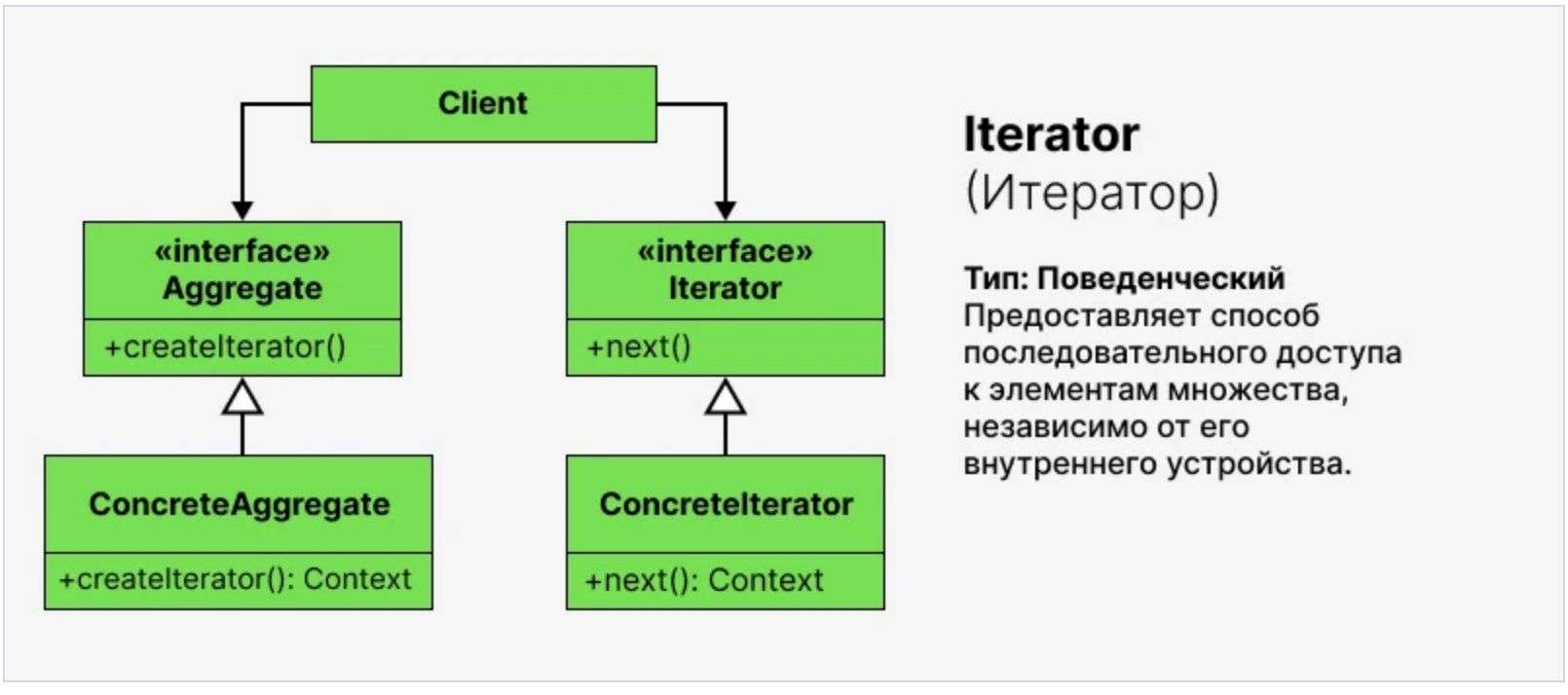
Поведенческие паттерны

JSP & Servlets
17 уровень, 0 лекция

ОТКРЫТА

Iterator

Iterator — поведенческий шаблон проектирования. Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из агрегированных объектов.



Например, такие элементы как дерево, связанный список, хеш-таблица и массив могут быть пролистаны (и модифицированы) с помощью объекта Итератор.

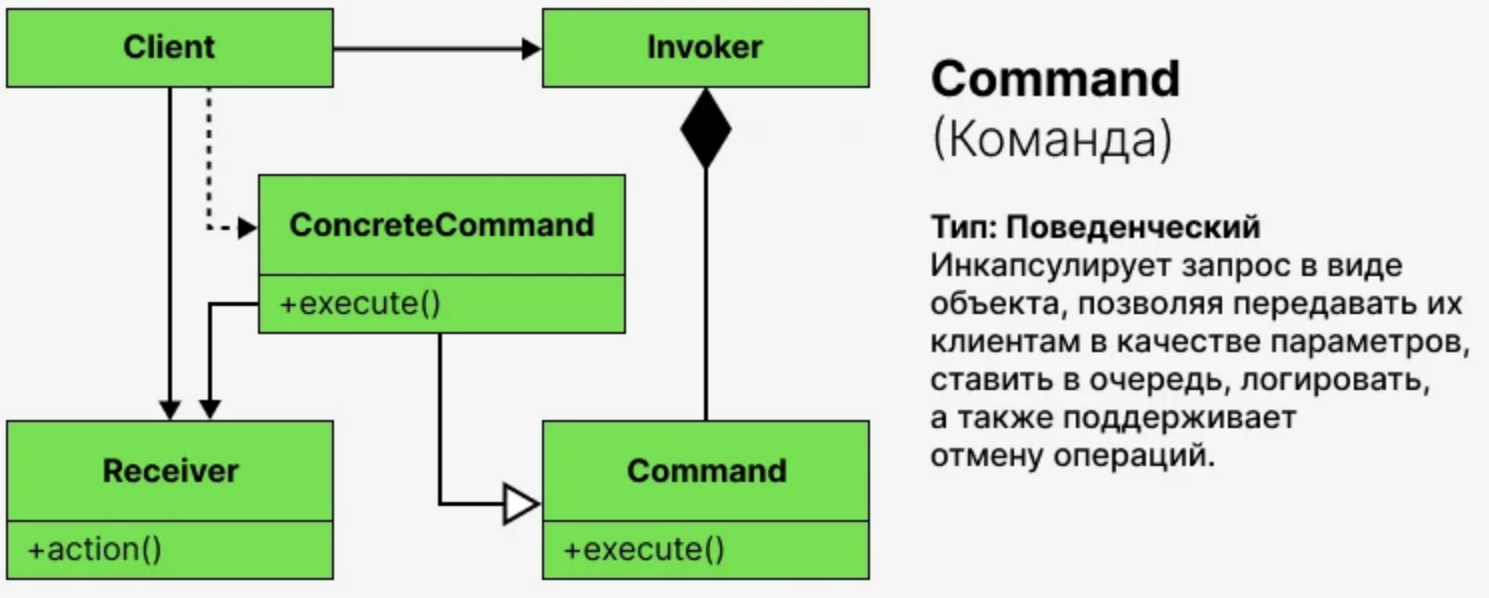
Перебор элементов выполняется объектом итератора, а не самой коллекцией. Это упрощает интерфейс и реализацию коллекции, а также способствует более логичному разделению обязанностей.

Особенностью полноценно реализованного итератора является то, что код, использующий итератор, может ничего не знать о типе итерируемого агрегата.

Такой подход используется очень часто. Например, ты отправляешь базе данных SQL-запрос, а в ответ она возвращает тебе итератор (в терминах SQL его обычно называют курсором). И ты с помощью полученного итератора можешь поочередно брать стоки из SQL-ответа.

Command

Команда (Command) — поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект команды заключает в себе само действие и его параметры.



Чтобы вызывать какой-то метод, обычно нужны:

- ссылка на объект
- имя метода (ссылка на метод)
- значения параметров метода
- ссылка на контекст, который содержит используемые объекты

Все эти данные нужно упаковать в один объект — Команда (**command**).

Но и это еще не все: ведь команду должен кто-то выполнить. Так что в состав этого паттерна входят еще четыре сущности: команды (**command**), приёмник команд (**receiver**), вызывающий команды (**invoker**) и клиент (**client**).

Объект `Command` знает о приёмнике и вызывает метод приемника. Значения параметров приёмника сохраняются в команде. Вызывающий объект (invoker) знает, как выполнить команду и, возможно, делает учёт и запись выполненных команд. Вызывающий объект (invoker) ничего не знает о конкретной команде, он знает только об интерфейсе.

Оба объекта (вызывающий объект и несколько объектов команд) принадлежат объекту клиента (client). Клиент решает, какие команды выполнить и когда. Чтобы выполнить команду, он передает объект команды вызывающему объекту (invoker).

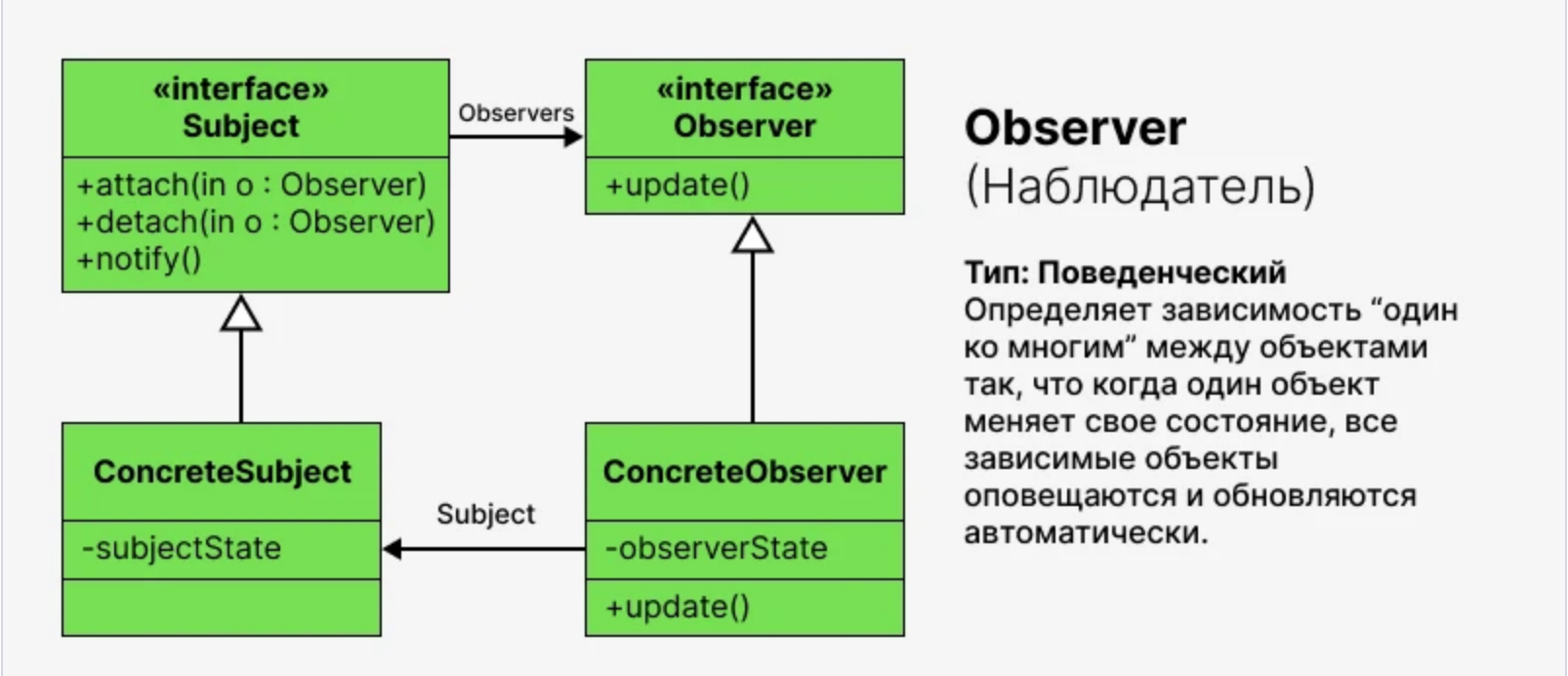
Использование командных объектов упрощает построение общих компонентов, которые необходимо делегировать или выполнять вызовы методов в любое время без необходимости знать методы класса или параметров метода.

Использование вызывающего объекта (invoker) позволяет вести учёт выполненных команд без необходимости знать клиенту об этой модели учёта (такой учёт может пригодиться, например, для реализации отмены и повтора команд).

Например, ты пишишь программу, которая позволяет выполнять различные задания по расписанию. С одной стороны твоя программа ведет учет заданий и управляет их запуском, с другой у нее может быть несколько исполнителей, каждый из которых умеет выполнять команды своего типа. Например, рассылка SMS, рассылка писем, рассылка сообщений в Telegram и т. п.

Observer

Наблюдатель (Observer) — поведенческий шаблон проектирования. Реализует механизм класса, который позволяет объекту этого класса получать оповещения об изменении состояния других объектов и тем самым наблюдать за ними.



Классы, на события которых другие классы подписываются, называются субъектами (**Subjects**), а подписывающиеся классы называются наблюдателями (**Observers**).

При реализации шаблона Наблюдатель обычно используются следующие классы:

- ***Observable*** — интерфейс, определяющий методы для добавления, удаления и оповещения наблюдателей;
- ***Observer*** — интерфейс, с помощью которого наблюдатель получает оповещение;
- ***ConcreteObservable*** — конкретный класс, который реализует интерфейс ***Observable***;
- ***ConcreteObserver*** — конкретный класс, который реализует интерфейс ***Observer***.

Шаблон Наблюдатель применяется в тех случаях, когда в системе:

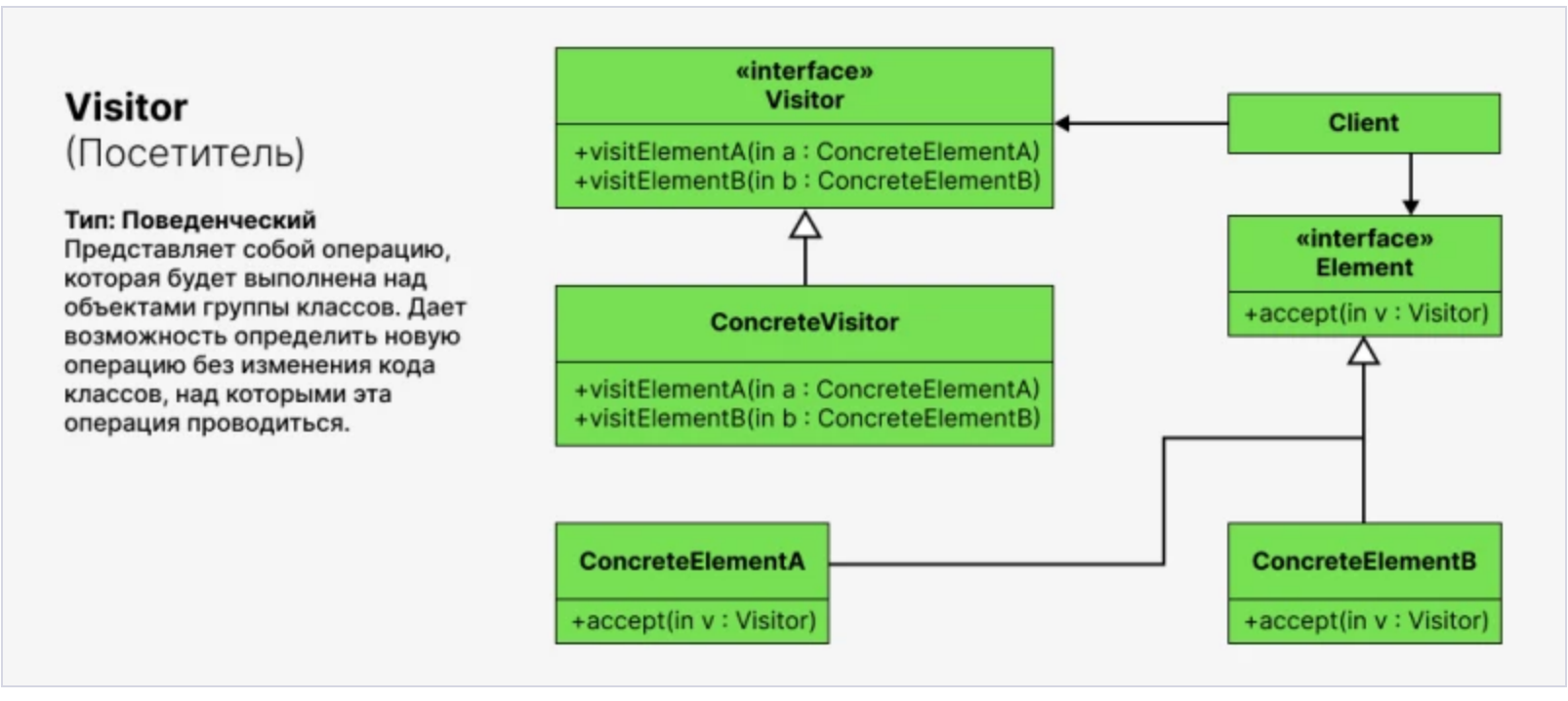
- существует как минимум один объект, рассылающий сообщения;
- имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения;
- позволяет избежать сильного зацепления взаимодействующих классов.

Этот шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

Visitor

Посетитель (Visitor) — поведенческий шаблон проектирования, описывающий операцию, которая выполняется над объектами других классов. При изменении visitor нет необходимости изменять обслуживаемые классы.

Шаблон демонстрирует классический приём восстановления информации о потерянных типах, не прибегая к понижающему приведению типов при помощи двойной диспетчеризации.



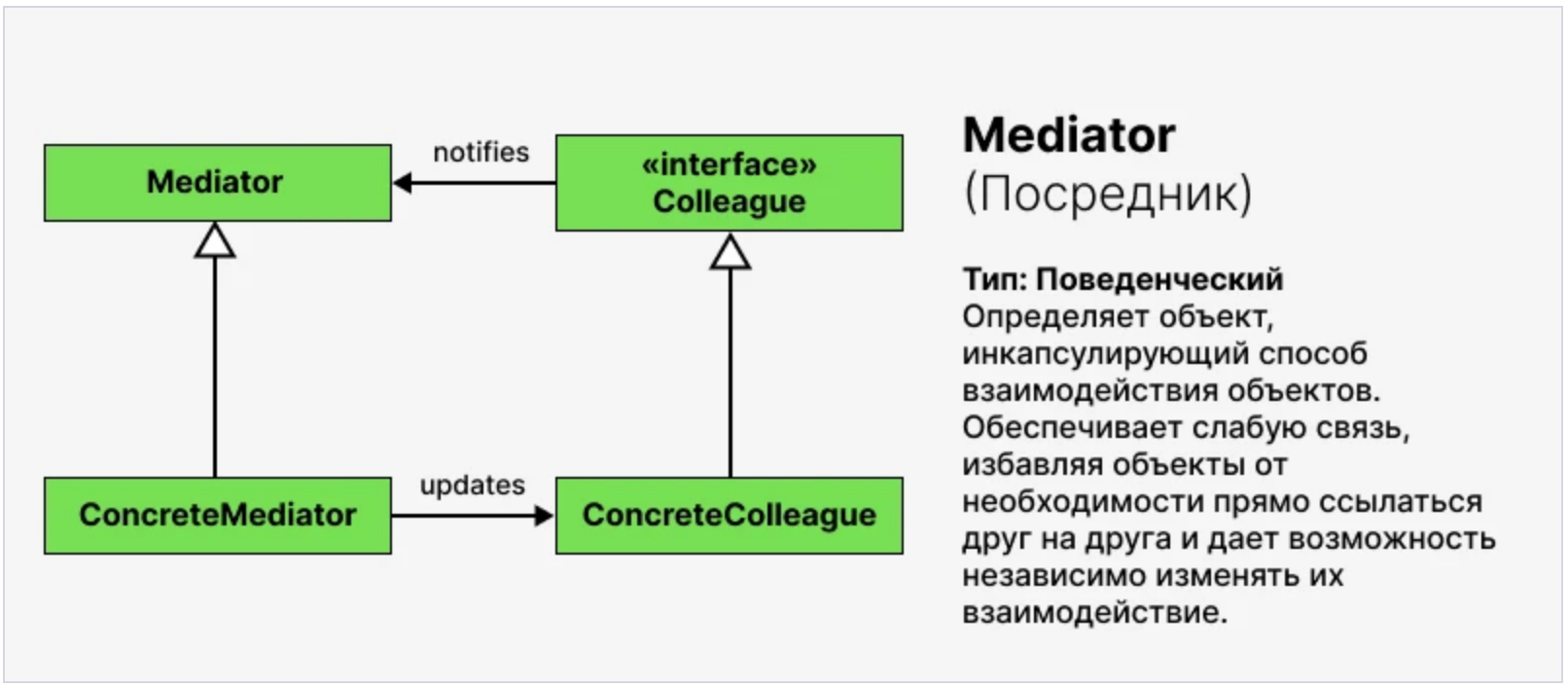
Необходимо сделать какие-то несвязные операции над рядом объектов, но нужно избежать загрязнения их кода. И нет возможности или желания запрашивать тип каждого узла и осуществлять приведение указателя к правильному типу, прежде чем выполнить нужную операцию.

Шаблон следует использовать, если:

- имеются различные объекты разных классов с разными интерфейсами, но над ними нужно совершать операции, зависящие от конкретных классов;
- над структурой необходимо выполнить различные усложняющие структуру операции;
- часто добавляются новые операции над структурой.

Mediator

Посредник (Mediator) — поведенческий шаблон проектирования, обеспечивающий взаимодействие множества объектов, формируя при этом слабое зацепление и избавляя объекты от необходимости явно ссылаться друг на друга.



Паттерн Посредник позволяет обеспечить взаимодействие множества объектов, сформировав при этом слабую связанность и избавив объекты от необходимости явно ссылаться друг на друга.

Посредник определяет интерфейс для обмена информацией с объектами `Коллеги` , **Конкретный посредник** координирует действия объектов `Коллеги` .

Каждый класс `Коллеги` знает о своем объекте `Посредник` , все `Коллеги` обмениваются информацией только с посредником, при его отсутствии им пришлось бы обмениваться информацией напрямую.

`Коллеги` посылают запросы **Посреднику/span> и получают запросы от него. Посредник реализует кооперативное поведение, пересылая каждый запрос одному или нескольким** `Коллегам` **.**

‹ Предыдущая лекция

Следующая лекция ›

не вижу никаких задач в этом квесте

Ответить 0

Илья Кирбижеков Уровень 34 12 октября 2022, 11:19

ничего не понятно, но очень интересно

Ответить +5

Алексей Уровень 89 EXPERT 9 сентября 2022, 13:41

Очень сложным языком написано. Мне как новичку вообще не понятно, например, что это значит: "Шаблон демонстрирует классический приём восстановления информации о потерянных типах, не прибегая к понижающему приведению типов при помощи двойной диспетчеризации."

Ответить +15

Dmytryi Shubchynskyi Уровень 68 4 января, 03:59

Дальше стало понятней?)

Ответить 0

Алексей Уровень 89 EXPERT 4 января, 22:31

Не сильно

Ответить 0