Лабораторная работа по СРВ №4 (Just patterns)

Морозов Андрей 3.3 МО

10 Вариант. Паттерн “**Итератор**”

Итератор – это **поведенческий** паттерн проектирования, который даёт возможность последовательно обходить элементы составных объектов, не раскрывая их внутреннего представления.

**Проблема**

Коллекции элементов могут иметь сложную структуру и для выполнения различных задач последовательный обход её элементов может быть выстроен по-разному.

В качестве примера можно привести обход дерева в глубину или в ширину. Оттого, если помещать реализацию логики обхода (а точнее различные версии реализации), сама сущность коллекции сильно разрастается, загромождается иной логикой помимо той, для чего коллекции и созданы - хранение элементов.

**Решение**

Для решения этой проблемы, внесения гибкости в принципы работы с коллекциями, выделяют итераторы в отдельные классы, сущности от которых имеют представление о текущем элементе, принципе выбора следующего элемента, факте наличия следующих элементов. Коллекция не знает об итераторе, логика обхода отделена от неё.

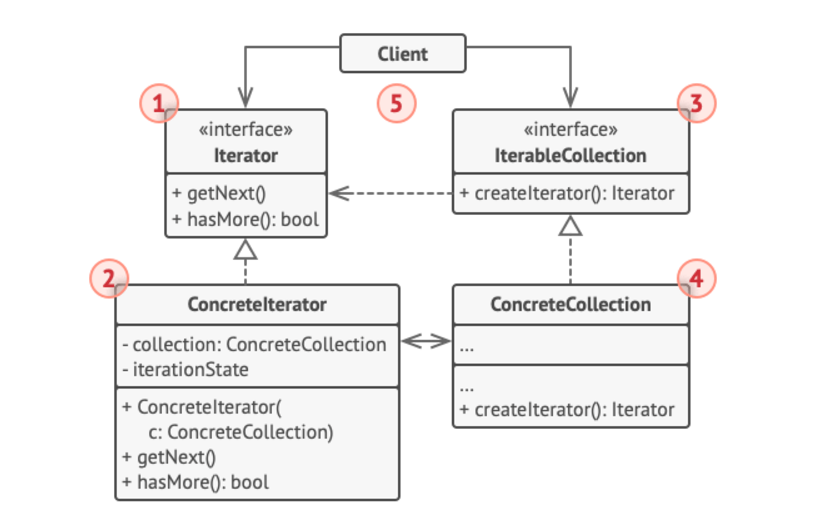
**Аналогия из жизни**

Паттерн можно сравнить с листанием страниц в книге. Представим, что у нас есть книга, и чтобы прочитать её, нам нужно последовательно ознакамливаться со страницами. Мы берём книгу и листаем страницы одну за другой пальцем.

* **Книга** — это коллекция, например, массив или список.
* **Страница** — это отдельный элемент коллекции.
* **Палец** — это итератор, который указывает на текущую страницу.
* **Действие листания** — это метод next(), который перемещает итератор к следующему элементу.
* **Конец книги** — это ситуация, когда метод hasNext() возвращает false, потому что больше страниц нет.

Итератор позволяет абстрагироваться от самой книги и сосредоточиться на последовательном доступе к её страницам без необходимости знать внутреннее устройство книги.

**UML диаграмма**



1. Интерфейс итератора описывает возможность получения следующего элемента коллекции и информации о наличии следующих элементов.

2. Создаётся имплементация интерфейса, принимающая на вход коллекцию и содержащая состояние итерации (отслеживание позиции). Итератор реализует алгоритм обхода некоторой коллекции.

3. Интерфейс коллекции (итерабельной) предписывает возможность создания итератора для данной коллекции.

4. Реализация интерфейса из п.3 при создании объекта коллекции. Реализуется метод создания итератора, возвращающего тип iterator согласно п.1. и связанного с той коллекцией, посредством метода createIterator() которой он был создан.

5. Клиент осуществляет пользование коллекциями и итераторами на уровне интерфейсов, не ссылаясь на конкретные экземпляры классов.

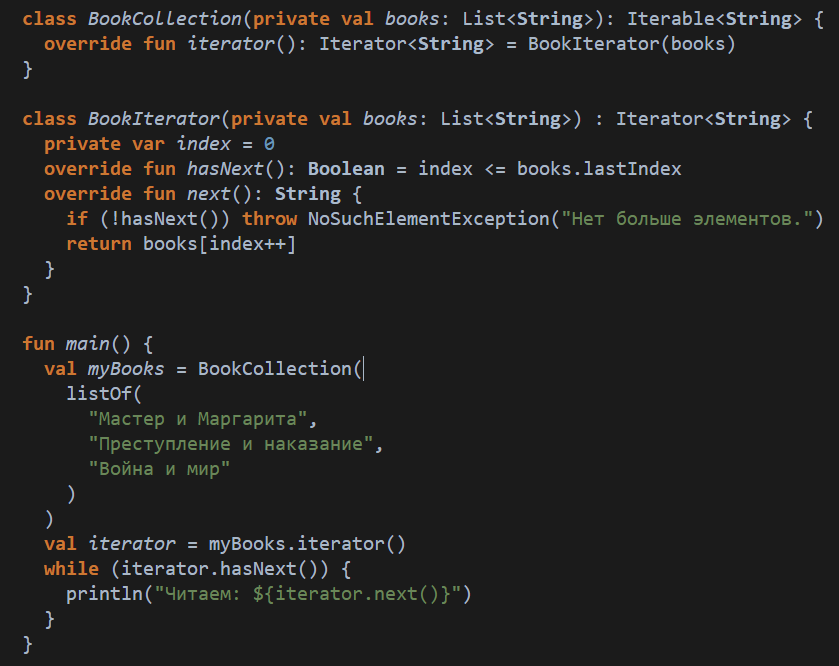
**Преимущества паттерна**

* Упрощает классы хранения данных.
* Скрытие деталей реализации структуры от пользователя.
* Позволяет реализовать различные способы обхода структуры данных.
* Позволяет одновременно перемещаться по структуре данных в разные стороны.
* Единый интерфейс обхода различных структур данных.

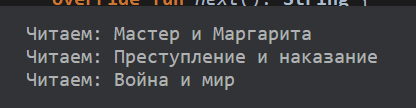
**Недостатки:**

* Не оправдан, если можно обойтись простым циклом.

Пример реализации итератора для коллекции книг на языке Kotlin:



Вывод программы:



Имеем интерфейс Iterator, входящий в стандартную библиотеку Kotlin и предписывающий методы hasNext() и next(), которые реализуются в классе BookIterator, наследуемом от вышеупомянутого интерфейса.

Класс BookCollection, который представляет нашу коллекцию книг. Для простоты примера конструктор принимает простой массив строк. Стоит заметить, что кастомная коллекция наследуется от также стандартного интерфейса Iterable, что предписывает наличие метода iterator() для создания итератора.

В самом классе BookIterator реализована простая логика перебора элементов до тех пор, пока мы не достигнем последнего элемента.