KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

Iterator - 처리를 반복하다

[KB] IT's Your Life

KB국민은행

Iterator - 처리를 반복하다

Iterator 패턴

```
for(int i =0; i < arr.length: i++) {
    System.out.println(arr[i]);
}
```

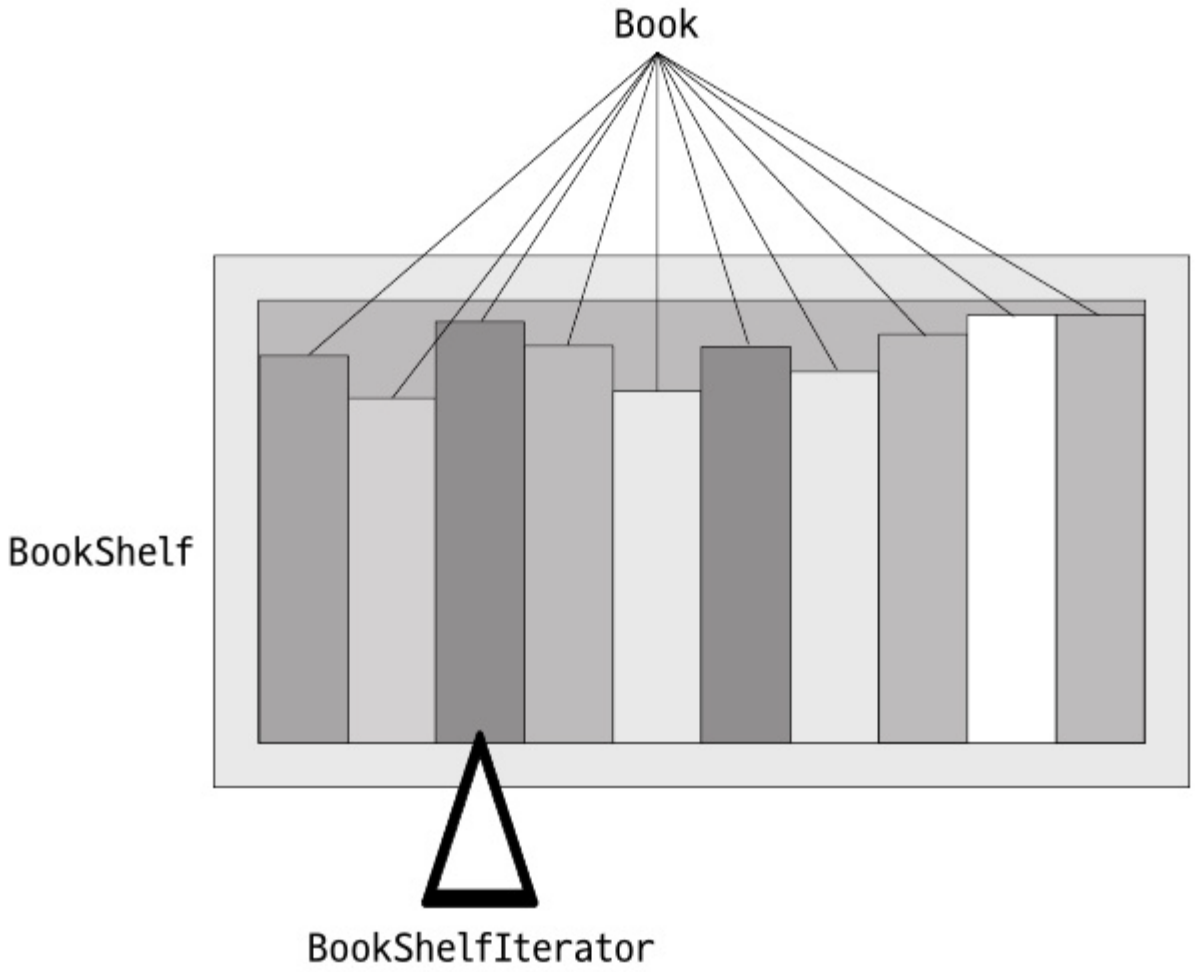
- 변수 i의 기능을 추상화하여 일반화한 것
- 무엇인가 많이 모여 있을 때 이를 순서대로 가리키며 전체를 검색하고 처리를 반복하는 것

```
for(int i  in arr) {
   ...
}
```

→ arr는 Iterator 패턴이 정의된 객체

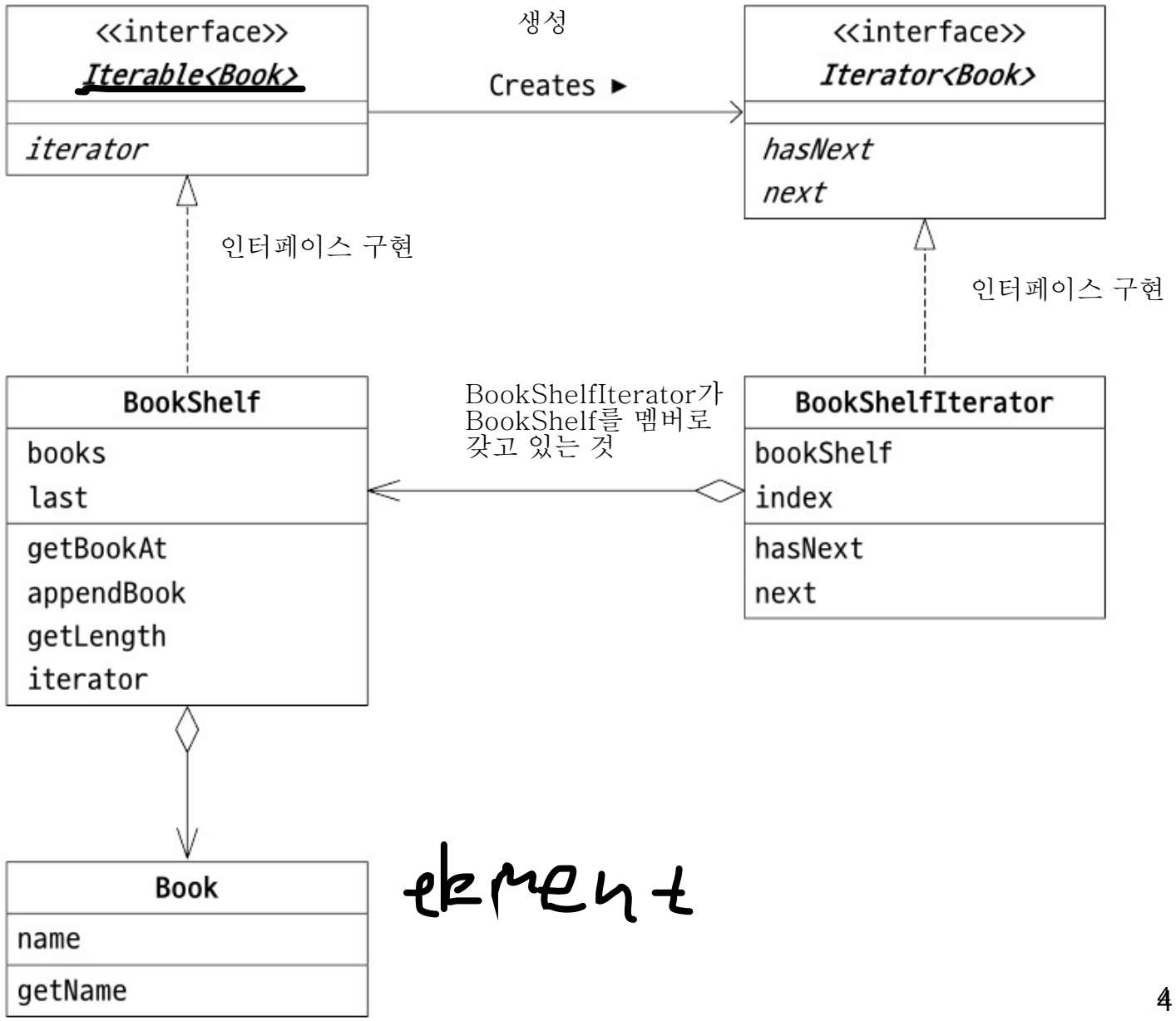
Iterator - 처리를 반복하다

- 예제 프로그램의 이미지 다이어그램



Iterator - 처리를 반복하다

- 예제 프로그램의 클래스 다이어그램



Iterator - 처리를 반복하다

클래스 및 인터페이스 목록

java 인터페이스

이름	설명
Iterable<E>	집합체를 나타내는 인터페이스(java.lang 패키지) 예제 프로그램에서는 Iterable<Book>으로 사용
Iterator<E>	처리를 반복하는 반복자를 나타내는 인터페이스(java.util 패키지) 예제 프로그램에서는 Iterator<Book>으로 사용
Book	책을 나타내는 클래스
BookShelf	책장을 나타내는 클래스
BookShelfIterator	책장을 검색하는 클래스
Main	동작 테스트용 클래스

Iterator - 처리를 반복하다

Iterable<E> 인터페이스

- 처리를 반복할 대상을 나타내는 것
- java.lang.Iterable

```
public interface Iterable<E> {
    public abstract Iterator<E> iterator();
}
```

Iterator - 처리를 반복하다

Iterator<E> 인터페이스

- 하나 하나의 요소 처리를 반복하기 위한 것
- 루프 변수와 같은 역할
- java.lang.Iterator

```
public interface Iterator<E> {
    public abstract boolean hasNext();
    public abstract E next();
}
```

Iterator - 처리를 반복하다

Book.java

```
public class Book {
    private String name;

    public Book(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

Iterator - 처리를 반복하다

BookShelf.java

우리가 정의하는 우리만의 컬렉션

```
public class BookShelf implements Iterable<Book>{
    private Book[] books;
    private int last = 0;

    public BookShelf(int maxsize) {
        this.books = new Book[maxsize];
    }

    public Book getBookAt(int index) {
        return books[index];
    }

    public void appendBook(Book book) {
        books[last] = book;
        last++;
    }

    public int getLength() {
        return last;
    }

    @Override
    public Iterator<Book> iterator() {
        return new BookShelfIterator(this);
    }
}
```

구현

모던 for문을 돌때

for(: AAA)
에서
AAA가 iterator() 함수를 호출함

Iterator - 처리를 반복하다

BookShelfIterator.java

```
public class BookShelfIterator {
    private BookShelf bookShelf;
    private int index;

    public BookShelfIterator(BookShelf bookShelf) {
        this.bookShelf = bookShelf;
        this.index = 0;
    }
}
```

Iterator - 처리를 반복하다

BookShelfIterator.java

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class BookShelfIterator implements Iterator<Book> {
    …
    @Override
    public boolean hasNext() {
        if(index < bookShelf.getLength()) {
            return true;
        } else {
            return false;
        }
    }

    @Override
    public Book next() {
        if(!hasNext()) {
            throw new NoSuchElementException();
        }

        Book book = bookShelf.getBookAt(index);
        index++;
        return book;
    }
}
```

nosuch element exception은 런타임 예외기 때문에 throw 머시기 머시기 필요 없음!

Iterator - 처리를 반복하다

Main.java

```
import java.util.Iterator;

public class Main {
    public static void main(String[] args) {
        BookShelf bookShelf = new BookShelf(4);
        bookShelf.appendBook(new Book("Around the world in 80 Days"));
        bookShelf.appendBook(new Book("Bible"));
        bookShelf.appendBook(new Book("Cinderella"));
        bookShelf.appendBook(new Book("Daddy-Long-Legs"));

        // 명시적으로 Iterator를 사용하는 방법
        Iterator<Book> it = bookShelf.iterator();
        while(it.hasNext()) {
            Book book = it.next();
            System.out.println(book.getName());
        }
        System.out.println();
```

Iterator - 처리를 반복하다

BookShelf.java

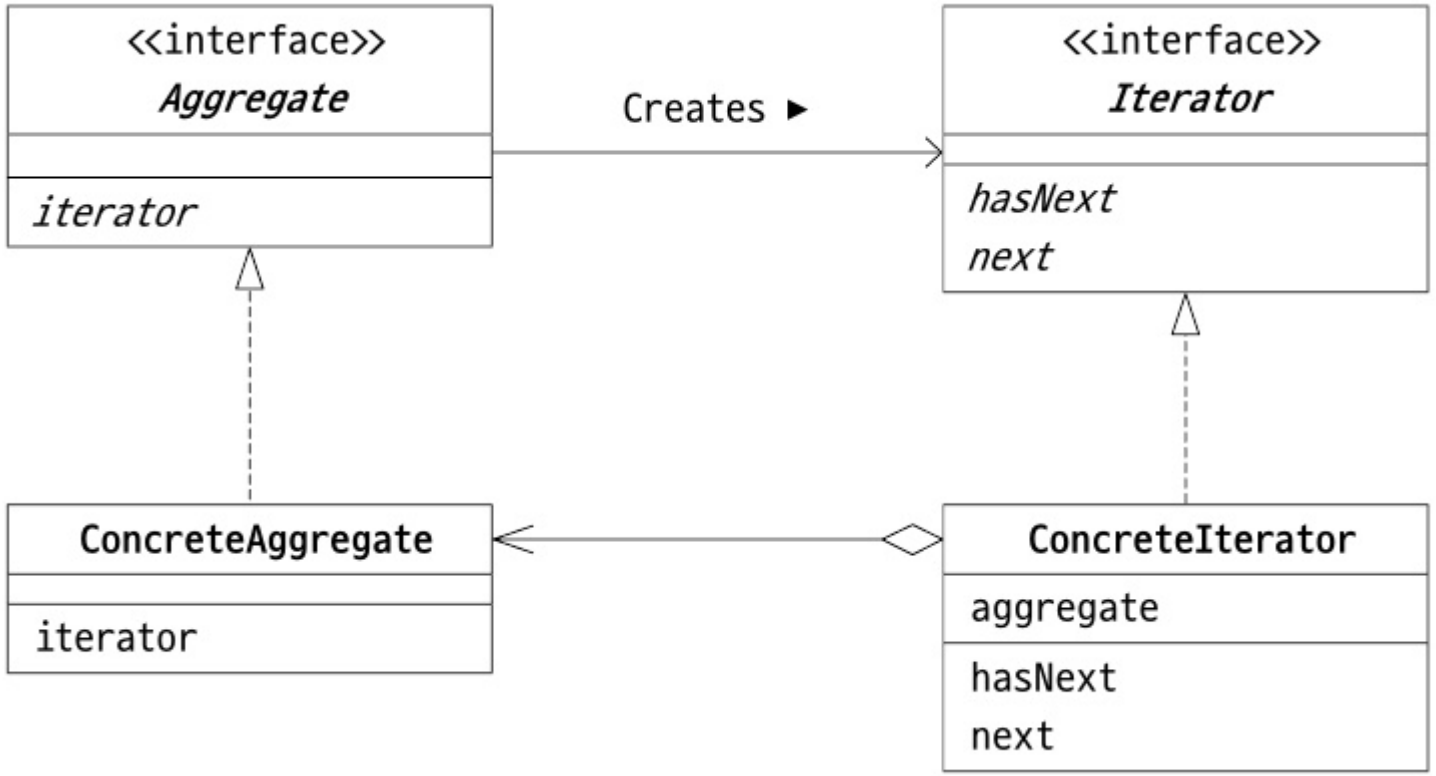
```
        // 확장 for문을 사용하는 방법
        for(Book book: bookShelf) {
            System.out.println(book.getName());
        }
        System.out.println();
    }
}
```

```
Around the world in 80 Days
Bible
Cinderella
Daddy-Long-Legs

Around the world in 80 Days
Bible
Cinderella
Daddy-Long-Legs
```

Iterator - 처리를 반복하다

- Iterator 패턴의 클래스 다이어그램



Iterator - 처리를 반복하다

Iterator 패턴을 사용하는 이유

- 구현과 분리하여 반복할 수 있음

→ 컬렉션이 무엇이든, 반복문에 변화는 없음

```
for(T x  in collection) {
   ...
}
```