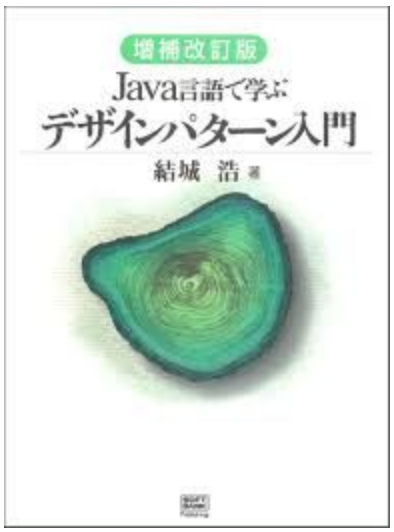
Java言語で学ぶデザインパターン入門

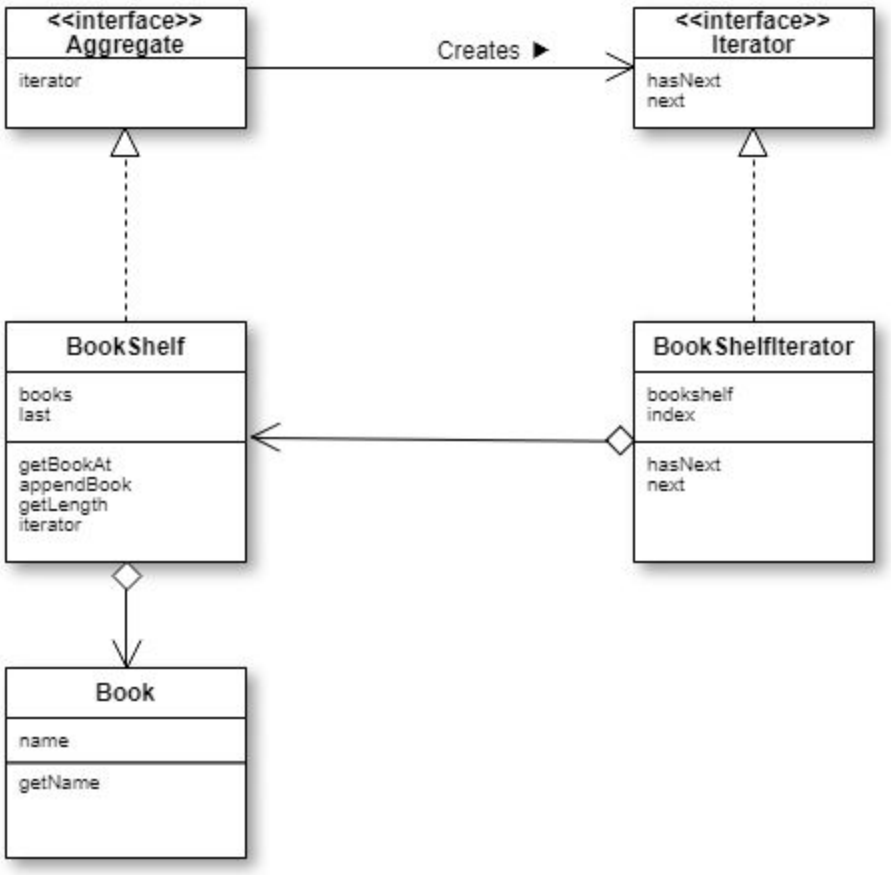
Iterator
１つ１つ数え上げる



題材

本棚（BookShelf）の中に本（Book）を入れ、その本の名前を順番に表示する。

クラス図



クラスとインタフェース一覧

名前	解説
Aggregate	集合体を表すインタフェース
Iterator	数え上げ、スキャンを行うインタフェース
Book	本を表すクラス
BookShelf	本棚を表すクラス
BookShelfIterator	本棚をスキャンするクラス
Main	動作テスト用のクラス

```java
import java.util.Iterator;

public interface Aggregate {
	public abstract Iterator iterator();
}
```

Aggregateインタフェース

iteratorメソッドは、集合体に対するIteratorを１個作成するためのもの。集合体を数え上げたい、スキャンしたい、１つずつ調べていきたいというときには、このiteratorメソッドをつかってIteratorインタフェースを実装したクラスのインスタンスを１個つくる。

```java
public interface Iterator {
	// 次の要素が存在するかを返す
	public abstract boolean hasNext();
	// 次の要素を返す
	public abstract Object next();
}
```

Iteratorインタフェース

要素の数え上げを行うもの、ループ変数のような役割を果たすもの。

```java
public class Book{
	private String name;
	public Book(String name) {
		this.name = name;
	}
	public String getName() {
		return name;
	}
}
```

Bookクラス

```
public class BookShelf implements Aggregate{
	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) {
		this.books[last] = book;
		last++;
	}
	public int getLength() {
		return last;
	}
	public Iterator iterator() {
		return new BookShelfIterator(this);
	}
}
```

Bookshelfクラス

```
public class BookShelfIterator implements Iterator{
	private BookShelf bookShelf;
	private int index;
	public BookShelfIterator(BookShelf bookShelf) {
		this.bookShelf = bookShelf;
		this.index = 0;
	}

	public boolean hasNext() {
		if(index < bookShelf.getLength()) {
			return true;
		} else {
			return false;
		}
	}
	public Object next() {
		Book book = bookShelf.getBookAt(index);
```

```
            index++;
            return book;
      }

}
```

BookShelfIteratorクラス

```
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"));
            Iterator it = bookShelf.iterator();
            while (it.hasNext()) {
                  Book book = (Book)it.next();
                  System.out.println(book.getName());
            }
      }
}
```

Mainクラス

なぜItaratorパターンにする必要があるのか

```
while (it.hasNext()) {
      Book book = (Book)it.next();
      System.out.println(book.getName());
}
```

Whileループではbookshelfの実装には依存していないことがわかる。つまり、BookShelfを実装した人が配列を使って本を管理することをやめて、Vectorを使うようにプログラムを修正したとしても上記のWhileループは全く変更しなくても動作する。実装とは切り離して数え上げを行うことができるのである。メソッドiteratorの戻り値をBookShelfIteratorではなく、Iteratorにしているのもポイント。再利用化の促進。