Iterator Pattern

집합체의 요소들을 순서대로 처리

Architecture, Modeling, DesignPattern, Behavioral

반복자(Iterator) API

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

```
public class ItemSequenceIterator<T> implements Iterator<T> {
    private ItemSquence<T> concreteAggregate;
    private int index = 0;

    public ItemSequenceIterator(ItemSquence<T> concreteAggregate) {
        this.concreteAggregate = concreteAggregate;
        this.index = 0;
    }

    @Override
    public boolean hasNext() {
        return index < concreteAggregate.length();
    }

    @Override
    public T next() {
        T item = concreteAggregate.get(index);
        index++;
        return item;
    }
}
```

반복할 집합체(Aggregate)

```
public interface Aggregate<T> {
    Iterator<T> iterator();
}
```

```
/* 간단한 ArrayList를 구현한 형태 */
public class ItemSquence<T> implements Aggregate<T> {
    private T[] items;
    private int last = 0;

    public ItemSquence(int capacity) {
        this.items = (T[]) new Object[capacity];
    }

    public T get(int index) {
        return items[index];
    }

    public void add(T item) {
        this.items[last] = item;
        last++;
    }

    public int length() {
        return last;
    }

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

실행

```
public class Main {
    public static void main(String[] args) {
        ItemSquence<Item> itemContainer = new ItemSquence<>(2);
        itemContainer.add(new Item("Item1"));
        itemContainer.add(new Item("Item2"));
        Iterator<Item> it = itemContainer.iterator();
        while(it.hasNext()) {
            Item item = it.next();
            System.out.println(item.getStr());
        }
    }
}

/* 출력 */
//Item1
//Item2
```