Java20-2\_Lab8

32191826 박주희

<어뎁터 패턴(Adapter Pattern)>

한 클래스의 인터페이스를 클라이언트에서 사용하고자 하는 다른 인터페이스로 변환한다. 이 패턴을 사용하면 다른 인터페이스와의 호환성 문제를 해결할 수 있다.

어뎁터가 특정 규격의 플러그를 다른 플러그와 호환이 가능하게 하듯이, 객체 지향 프로그래밍에서도 비슷하게 사용할 수 있습니다.

🡪 어뎁터 패턴은 B를 A처럼 포장하여 A로 사용할 수 있게 하는 패턴이다.

Collection에 있는 일련의 항목들에 접근할 수 있게 해 주면서 항목을 제거할수도 있게 해 주는 iterator 인터페이스

<수업시간에 배운 TurkeyAdaper>

이번 과제 이해를 위해 TurkeyAdaper class부터 이해해보았다.

클라이언트 -> request() -> 어댑터 - translatedRequest() -> 어댑티

Target= Duck Adaptee = Turkey

TurkeyAdaper implements Duck.

main클래스에서 Duck turkeyAdapter = new TurkeyAdapter(turkey);로 생성

< 전체적인 구조>

Target= DataList<E> Adaptee = List<E>

DataListAdapter implements DataList<E>

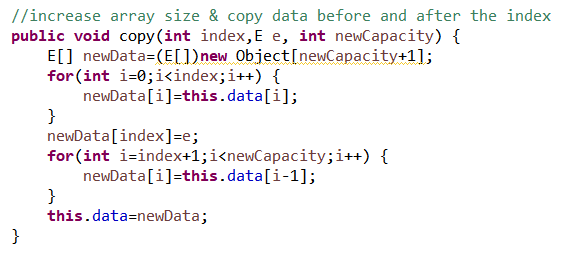
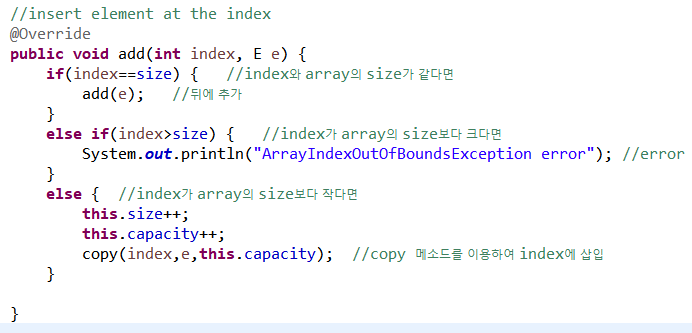
main 클래스에서 DataList<String> arr2=new DataListAdapter<>(new ArrayList<>());

DataList<String> arr3=new DataListAdapter<>(new LinkedList<>());로 생성

Collection<E>⊃List<E>⊃ArrayList<E>, Vector<E>, LinkedList<E>

<주요 구현 과정>

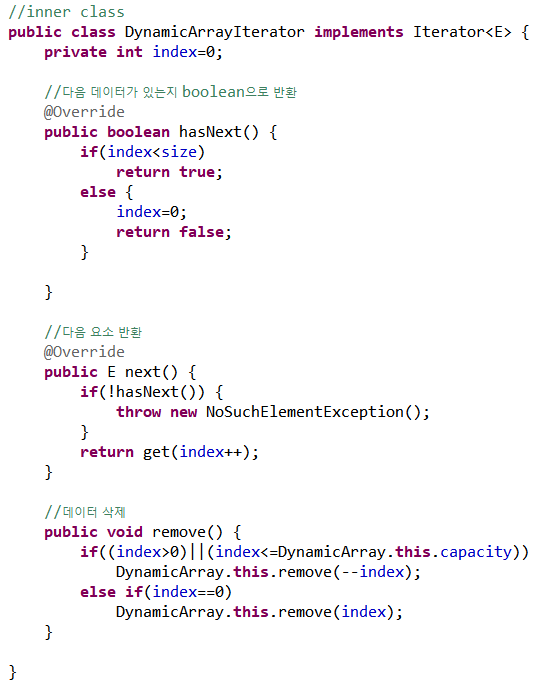
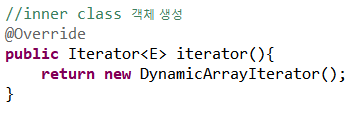
DynamicArray implements DataList<E>이므로 DataList<E>의 메소드를 모두 override해야 한다.

수업시간에 교수님께서 많이 도와주셔서 DynamicArray의 모든 메소드를 구현할 때 어려움은 없었다. index, e, newCapacity를 받아 index에 e를 넣는 메소드인 copy 메소드를 미리 구현해두어서 add(int index, E e)를 구현할 때 조건문과 copy를 이용하여 어렵지 않게 구현할 수 있었다.

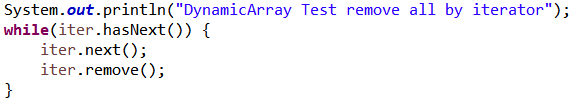
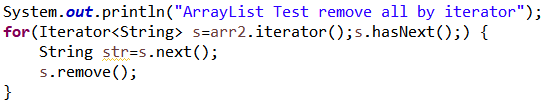
E[] new Data=(E[]) new Obejct[newCapacity+1]

E[] new Data=new E[newCapacity+1]는 오류가 발생한다고 수업시간에 배워서 바로 구현할 수 있었다.



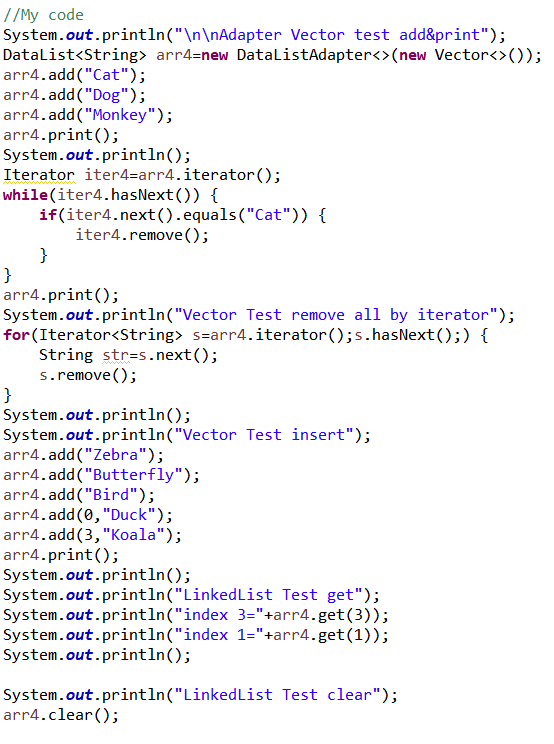
아직 innerclass의 사용이 익숙하지 않지만 이 또한 수업시간에 많이 도와주셔서 어려움은 없었다.

iterator 메소드가 있어서 main class에서 outerclass.innerclass를 하지 않아도 iterator를 부르면 innerclass가 불러진다.

iterator를 이용하여 선택적으로 삭제하는 것은 잘 돌아갔으나 전체 삭제하는 부분은 좀 어려웠다. iter.next()를 해야 다음 원소가 넘어가기 때문에 iter.next를 해주어야한다. 이 부분이 빠지면 오류가 뜬다.

<My code>



Collection<E>⊃List<E>⊃ArrayList<E>, Vector<E>, LinkedList<E> 중에서 과제로 내주신 ArrayList<E>와 LinkedList<E>를 제외한 Vector<E>도 test 해보았다. 모두 다 잘 돌아감을 알 수 있다.

<결과화면>

결과 화면이 너무 길어서 txt파일로 따로 첨부해두었습니다.