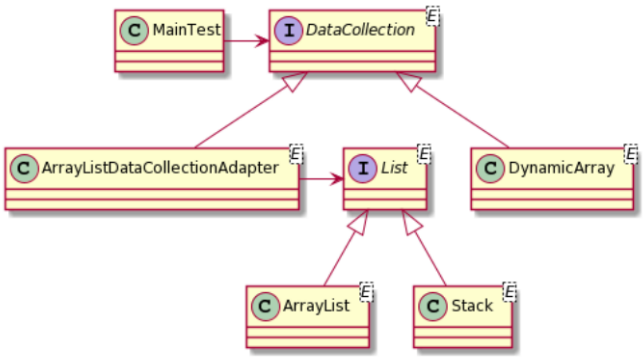
# Java Programming II Lab 8


교 과 명 : 자바프로그래밍 2
담당교수명 : 박경신 교수님
학 과 : 컴퓨터공학과
학 번 : 32185010
성 명 : 홍찬희
제 출 일 : 2021. 11. 16

# Lab 8

어댑터패턴은 한 클래스의 인터페이스를 사용하고자 하는 다른 인터페이스로 변환할때 사용한다. 이번 Lab8에서는 List 인터페이스를 사용하는 Stack, ArrayList 객체를 직접만든 DataCollection 인터페이스로 변환하였다.




구조는 위 그림과 같다. List 인터페이스를 가지고 있는 DataCollection을 통해 DataCollection의 동작을 DynamicArray의 메소드명과 동일하게 맞춰, DataCollection을 어댑터로 구현했다.

```
// Iterable 객체의 상속을 받는 interface
public interface DataCollection<E> extends Iterable<E> {
    void put(E e);
    void insert(int index, E e);
    void remove(int index);
    E elemAt(int index);
    int length();
    void clear();
}
```

인터페이스를 통해 DataCollection의 기능을 정의했다. 기능은 6가지이고, 추가로 Iterable을 extends하고 있으므로, Data
Collection을 implements하게 된다면 추가로 Iterator를 정의해줘야한다.

```
// DataCollection을 implements하는 DynamicArray
public class DynamicArray<E> implements DataCollection<E>{
    private int length; // null이 아닌 배열의 길이
    private int capacity; // 배열의 용량
    private E[] data; // 제네릭의 배열형태

    // constructor
    public DynamicArray(){
        this.length = 0;
        this.capacity = 5;
        this.data = (E[]) new Object[capacity]; // DynamicArray객체가 생성될때 기본으로 capacity길이 만큼의 제네릭 배열을 생성함.
    }
```

DynamicArray의 data에 들어오는 객체를 유연하게 하기위해 제네릭을 사용하였다. DynamicArray 객체를 생성하게되면, length는 0, capacity는 5로 초기화되고, data는 capacity만큼 배열이 생성된다. 여기서 직접 제네릭의 배열을 생성할 수 없으므로 Object로 배열을 생성한뒤, 제네릭의 배열로 다운캐스팅을 시켰다.

```java
// 배열을 맨뒤에 추가하는 메소드
@Override
public void put(E e) {
    if(length >= capacity){
        System.out.println("동적으로 용량 증가");
        length++; capacity++;
        copy(e, capacity);
    }else{
        data[length] = e;
        length += 1;
    }
}

// 배열의 index에 해당 객체를 추가하는 메소드
@Override
public void insert(int index, E e) {
    // 객체를 추가했을때 배열의 길이가 용량보다 크면 동적으로 용량을 증가시킨다.
    if(length+1 >= capacity){
        System.out.println("동적으로 용량 증가");
        length++; capacity++;
        copy(index, e, capacity);
    }else{
        for(int i=length;i>index;i--){
            this.data[i] = this.data[i-1];
        }
        this.data[index] = e;
        length++;
    }
}
```

```java
// 용량이 꽉찼을때, 데이터를 받아 추가해주는 코드
private void copy(E data, int newCapacity){
    E[] new_data = (E[]) new Object[newCapacity];
    for(int i=0;i<newCapacity-1;i++){
        new_data[i] = this.data[i];
    }
    new_data[newCapacity-1] = data;
    this.data = new_data;
}

// 용량이 꽉찼을때, 데이터를 중간에 삽입해주는 코드
private void copy(int index, E data, int newCapacity){
    E[] new_data = (E[]) new Object[newCapacity];
    for(int i = 0; i < index; i++){
        new_data[i] = this.data[i];
    }
    new_data[index] = data;
    for(int i = newCapacity - 1; i > index; i--){
        new_data[i] = this.data[i-1];
    }
    this.data = new_data;
}
```

DynamicArray의 put과 insert구현이다. put은 객체를 현재 배열의 가장뒤에 넣었고, insert는 지정 인덱스부터 끝까지 있던 배열들을 한칸씩 뒤로 밀고, 지정 인덱스에 해당 객체를 넣는 방식으로 구현했다.
length와 capacity를 이용해 해당 배열의 용량이 꽉찼을때 위의 오른쪽그림의 copy를 이용해 동적으로 배열의 용량을 증가시켜주도록했다. 해당 메소드 역시 현재 용량의 +1 한만큼의 용량을 Object 배열로 생성한뒤 기존의 값을 복사한 뒤, 각 메소드에 맞는 실행을 한 뒤, this.data에 새로운 배열을 넣어주는 식으로 구현했다.
나머지 기능들도 위와 비슷한 방법들로 구현했다.

```java
// DataCollection을 implements해서 List의 사용을 DataCollection과 같은 방법으로
// 사용하게 해주는 어댑터 클래스
public class ListDataCollection<E> implements DataCollection<E>{
    private List<E> list; // ArrayList와 Stack의 상위객체인 List로 객체를 관리한다.

    // constructor
    public ListDataCollection(List<E> list) { this.list = list; }

    // DataCollection의 put은 List에서 add와 같다.
    @Override
    public void put(E e) { list.add(e); }

    // List에서 객체.add(index, 객체)를 통해 insert를 구현할 수 있다.
    @Override
    public void insert(int index, E e) { list.add(index, e); }

    // List의 객체 삭제는 DataCollection의 메소드명과 동일
    @Override
    public void remove(int index) { list.remove(index); }
```

다음은 List 자료구조를 DynamicArray와 동일한 메소드로 실행시키기 위해서 만든 ListDataCollection 클래스 즉 어댑터이다. DataColleciton 인터페이스를 implements 해서 해당 메소드에 맞는 List 객체의 실행을 정의해주면 된다.
예를 들어, DataCollection에서 put은 객체를 배열의 맨뒤에 추가해주는 명령이므로, List에서 add(객체)를 통해 동일한 동작을 구현했다.

**[실행결과]**

```
System.out.println("\n\nDynamicArray add & print");
DataCollection<City> arr = new DynamicArray<>();
arr.put(new City( city: "서울", country: "한국")); // put 8~10 elements
arr.put(new City( city: "베이징", country: "중국"));
arr.put(new City( city: "도쿄", country: "일본"));
arr.put(new City( city: "마닐라", country: "필리핀"));
arr.put(new City( city: "방콕", country: "타이"));
arr.put(new City( city: "하노이", country: "베트남"));
arr.put(new City( city: "몰레", country: "몰디브"));
arr.put(new City( city: "카불", country: "아프카니스탄"));
arr.put(new City( city: "리야드", country: "사우디아라비아"));
arr.put(new City( city: "예루살렘", country: "이스라엘"));

arr.forEach(System.out::println); // test Iterable<E>
System.out.println();
// remove & insert & elemAt & clear & remove all using iterator &
// print using for/while/foreach
arr.remove( index: 1);
arr.remove( index: 2);
arr.remove( index: 3);
arr.insert( index: 0,new City( city: "앙카라", country: "터키"));
System.out.println("index 5: "+arr.elemAt( index: 5));
arr.forEach(System.out::println);
System.out.println();
```

```
DynamicArray add & print
동적으로 용량 증가
동적으로 용량 증가
동적으로 용량 증가
동적으로 용량 증가
동적으로 용량 증가
City{city='서울', country='한국'}
City{city='베이징', country='중국'}
City{city='도쿄', country='일본'}
City{city='마닐라', country='필리핀'}
City{city='방콕', country='타이'}
City{city='하노이', country='베트남'}
City{city='몰레', country='몰디브'}
City{city='카불', country='아프카니스탄'}
City{city='리야드', country='사우디아라비아'}
City{city='예루살렘', country='이스라엘'}
```

DynamicArray를 출력해보면, 동적으로 용량이 증가할때마다 해당 메시지가 출력된다. 또한 Iterator를 구현했기때문에 람다식도 정상적으로 출력된다.

arr는 DataCollection 인터페이스이기 때문에 ListDataCollection 어댑터를 통해 List객체를 넣어 업캐스팅을 시켜주면 위와 비슷하게 구현해도 정상적으로 작동한다.

```
ArrayList<City> list = new ArrayList<>();
list.add(new City( city: "서울", country: "한국")); // put 8~10 elements
list.add(new City( city: "베이징", country: "중국"));
list.add(new City( city: "도쿄", country: "일본"));
list.add(new City( city: "마닐라", country: "필리핀"));
list.add(new City( city: "방콕", country: "타이"));
list.add(new City( city: "하노이", country: "베트남"));
list.add(new City( city: "몰레", country: "몰디브"));
list.add(new City( city: "카불", country: "아프카니스탄"));
list.add(new City( city: "리야드", country: "사우디아라비아"));
list.add(new City( city: "예루살렘", country: "이스라엘"));
arr = new ListDataCollection<>(list);
arr.put(new City( city: "시카고", country: "미국")); // put 2~3 elements
arr.put(new City( city: "브라질리아", country: "브라질"));
arr.forEach(System.out::println);
System.out.println();
```

```
index 5: City{city='카불', country='아프카니스탄'}
City{city='앙카라', country='터키'}
City{city='서울', country='한국'}
City{city='도쿄', country='일본'}
City{city='방콕', country='타이'}
City{city='몰레', country='몰디브'}
City{city='카불', country='아프카니스탄'}
City{city='리야드', country='사우디아라비아'}
City{city='예루살렘', country='이스라엘'}

City{city='서울', country='한국'}
City{city='베이징', country='중국'}
City{city='도쿄', country='일본'}
City{city='마닐라', country='필리핀'}
City{city='방콕', country='타이'}
City{city='하노이', country='베트남'}
City{city='몰레', country='몰디브'}
City{city='카불', country='아프카니스탄'}
City{city='리야드', country='사우디아라비아'}
City{city='예루살렘', country='이스라엘'}
City{city='시카고', country='미국'}
City{city='브라질리아', country='브라질'}
```

ArrayList이지만, arr에 ListDataCollection에 ArrayList를 넣어 객체를 생성해서 DynamicArray와 같은 메소드를 사용해도 동일하게 동작하는것을 볼 수 있다. Stack 또한, 동일하게 동작하는 것을 알 수 있다.

**[추가코드]**

List의 다른 하위객체인 LinkedList도 넣어봤다. 동일하게 동작하는 것을 확인할 수 있다.

```
// [추가코드] LinkedList
LinkedList<City> linkedList = new LinkedList<>();
linkedList.add(new City( city: "서울", country: "한국")); // put 8~10 elements
linkedList.add(new City( city: "베이징", country: "중국"));
linkedList.add(new City( city: "도쿄", country: "일본"));
linkedList.add(new City( city: "마닐라", country: "필리핀"));
linkedList.add(new City( city: "방콕", country: "타이"));
linkedList.add(new City( city: "하노이", country: "베트남"));
linkedList.add(new City( city: "몰레", country: "몰디브"));
linkedList.add(new City( city: "카불", country: "아프카니스탄"));
linkedList.add(new City( city: "리야드", country: "사우디아라비아"));
linkedList.add(new City( city: "예루살렘", country: "이스라엘"));
arr = new ListDataCollection<>(linkedList);
arr.put(new City( city: "시카고", country: "미국")); // put 2~3 elements
arr.put(new City( city: "브라질리아", country: "브라질"));
// remove & insert & elemAt & clear & remove all using iterator &
arr.remove( index: 0); arr.remove( index: 0); arr.remove( index: 0);
arr.insert( index: 0,new City( city: "시드니", country: "호주"));
System.out.println("index0 : "+arr.elemAt( index: 0));
// print using for/while/foreach
arr.forEach(System.out::println);
System.out.println();
arr.clear();
arr.forEach(System.out::println);
System.out.println();
```

```
index 5: City{city='카불', country='아프카니스탄'}
City{city='앙카라', country='터키'}
City{city='서울', country='한국'}
City{city='도쿄', country='일본'}
City{city='방콕', country='타이'}
City{city='몰레', country='몰디브'}
City{city='카불', country='아프카니스탄'}
City{city='리야드', country='사우디아라비아'}
City{city='예루살렘', country='이스라엘'}

City{city='서울', country='한국'}
City{city='베이징', country='중국'}
City{city='도쿄', country='일본'}
City{city='마닐라', country='필리핀'}
City{city='방콕', country='타이'}
City{city='하노이', country='베트남'}
City{city='몰레', country='몰디브'}
City{city='카불', country='아프카니스탄'}
City{city='리야드', country='사우디아라비아'}
City{city='예루살렘', country='이스라엘'}
City{city='시카고', country='미국'}
City{city='브라질리아', country='브라질'}
```