# REPORT

자바 프로그래밍2 1분반 LAB7


**제출일** 2023.11.20
**소속** 소프트웨어학과
**학번** 32201817
**이름** 박정운

[1] DataCollection<E>

```
public interface DataCollection <E> extends Iterable<E>{
    void put(E e);// put element to the dynamic array
    void insert(int index, E e);// insert element at the index
    void remove(int index);// remove element at the index
    E elemAt(int index);// get element at the index
    int length(); // get the length of dynamic array (# of elements)
    void clear();// remove all elements and rese
}
```

DataCollection 인터페이스는 Iterable<E>를 상속하며  데이터를 관리하는데 필요한 추상 메소드를 가진다. 메소드에는 배열의 끝에 데이터를 추가하는 put, 원하는 인덱스에 데이터를 삽입하는 insert, 지정된 인덱스의 데이터를 삭제하는 remove, 해당 index의 값을 가져오는 elemAt, 배열의 길이를 반환하는 length, 배열을 모두 삭제하는 clear가 있다.

[2] DynamicArray<E>

```
public class DynamicArray<E> implements DataCollection<E>{
    private Object[] data = null; //객체가 저장될 배열
    private int length = 0; //길이
    private int capacity = 0; //수용 가능 길이

    //생성자
    public DynamicArray(int capacity) {
        this.capacity = capacity;
        this.data = new Object[this.capacity];
    }

    //getter
    public Object[] getData() { return this.data;}
    public int getCapacity() { return this.capacity; }
    @Override
    public int length() { return this.length;}
```

DynamicArray는 DataCollection을 상속받아 추상 클래스를 구현한다. member로는 주 배열인 Object[] data, 길이를 나타내는 length, 수용 가능 최대 길이를 나타내는 capacity가 있다.

```
//배열 뒤에 객체 넣기
@Override
public void put(E e) {
    if (this.length < this.capacity) {
        this.data[length++] = e; //객체 입력 후 length++
    }
    else {
        System.out.println("dynamic array increase size=" + length);
        this.length++;
        this.capacity++;
        copy(e, this.capacity);
    }
}
```

put() 메소드는 만약 capacity보다 length가 작은 경우 인자를 추가하고 length를 1 증가시킨다. 그렇지 않은 경우 length와 capacity를 증가시키고 객체를 추가한다.

```
@Override
public void insert(int index, E e) {
    //새 객체가 배열에 들어갈 수 있으면(capacity로 비교)
    if (this.length + 1 < this.capacity) {
        this.length++;
        for (int i = this.length - 1; i > index - 1; i--) {
            //index 이후 객체들 한 칸씩 뒤로 이동
            this.data[i+1] = this.data[i];
        }
        // insert new e
        this.data[index] = e;
    }
    else {
        System.out.println("insert: dynamic array increase capacity=" + capacity);
        this.length++;
        this.capacity++;
        copy(index, e, this.capacity);
    }
}
```

Insert() 메소드는 새 객체가 배열에 들어갈 수 있는지 확인하고 들어갈 수 있으면 index 이후 객체를 한 칸씩 뒤로 이동시키고 해당 index에 인자를 저장한다. 그렇지 않은 경우 length와 capacity를 증가시키고 객체를 추가한다.

```
@Override
public void remove(int index) {
    if (index < 0 || index > length) {
        System.out.println(x:"Error ArrayIndexOutOfBounds");
    }
    else {
        //삭제 index 이후 객체들 앞으로 한 칸씩 이동
        for (int i = index; i < length-1; i++) {
            this.data[i] = this.data[i+1];
        }
        this.length--;
        this.data[length] = null; //마지막 칸 null
    }
}
```

remove() 함수는 해당 인덱스에 있는 객체를 삭제한 후 index 이후 객체들을 한 칸씩 앞으로 이동한다. 그리고 length를 -1 한다.

```
@Override
public E elemAt(int index) {
    // if index in the array bounds
    if (index >= 0 && index < this.length) {
        //해당 index의 객체 return
        return (E)this.data[index];
    }
    return null;
}
```

elemAt() 함수는 해당 인덱스에 존재하는 객체를 리턴해준다.

```
@Override
public void clear() {
    //data[] 재생성, length 0으로 초기화
    this.data = new Object[this.capacity];
    this.length = 0;
}
```

clear() 함수는 원래 배열을 빈 data[]로 다시 생성하고, length를 0으로 초기화한다.

```
public class DynamicArrayIterator implements Iterator<E> {
    private int index = 0;
    //다음 값이 존재하는가
    @Override
    public boolean hasNext() {
        return index < length;
    }
    //다음 객체를 return
    @Override
    public E next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        return elemAt(index++);
    }
    //index-1 객체를 삭제
    @Override
    public void remove() {
        DynamicArray.this.remove(--index);
    }
}

@Override //iterator return
public Iterator<E> iterator() {
    return new DynamicArrayIterator();
}
```

DataArrayIterator 클래스는 Inner class로, hasNext(), next(), remove() 함수를 제공한다.
DynamicArray 클래스는 iterator()를 통해 Iterator를 return 한다.

[3] ListDataCollectionAdapter<E>

```
public class ListDataCollectionAdapter<E> implements DataCollection<E> {
    private List list; //DataCollection으로 변환될 List

    //생성자
    public ListDataCollectionAdapter(List list){
        this.list = list;
    }
```

ListDataCollectionAdapter는 List interface를 입력받고 이를 DataCollection처럼 사용할 수 있도록 하는 Adapter 클래스이다.

```java
@Override
public Iterator iterator() {
    //DataCollection.iterator -> list.iterator
    return list.iterator();
}

@Override
public void put(Object e) {
    //DataCollection.put -> list.add
    list.add(e);
}

@Override
public void insert(int index, Object e) {
    //DataCollection.insert -> list.add
    list.add(index, e);
}

@Override
public void remove(int index) {
    //DataCollection.remove -> list.remove
    list.remove(index);
}

@Override
public E elemAt(int index) {
    //DataCollection.elemAt -> list.get
    return (E)list.get(index);
}

@Override
public int length() {
    //DataCollection.length -> list.size
    return list.size();
}

@Override
public void clear() {
    //DataCollection.clear -> list.removeAll
    list.removeAll(list);
}
```

List의 메소드를 동일한 기능을 수행하는 DataCollection의 추상 메소드에 매칭시켜 List가 DataCollection 처럼 동작할 수 있도록 돕는다.

[4] FileImporter

```java
public interface FileImporter<E> {
    List<Element> importFile(String filePath); //파일 읽어오기
    void exportFile(String filePath, List<Element> list); //파일 생성
}
```

FileImporter는 인터페이스로서, 파일을 읽어오는 메소드와 파일을 생성하는 메소드를 가진다.

[5] ElementCSVImporter

```java
public class ElementCSVImporter implements FileImporter<Element> {
    //csv 파일 import
    @Override
    public List<Element> importFile(String filePath) {
        List<Element> list = new ArrayList<Element>();
        String line = "";

        // load data
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("file import: " + filePath);
            String delimiter = ",";
            while ((line = br.readLine()) != null) {
                if (line.contains(s:"#")) {
                    System.out.println("line contains #: " + line);
                    continue;
                }
                // use comma as separator
                String[] items = line.split(delimiter);
                /*for (String i : items) {
                    System.out.println(i);
                }*/
                Element pe = parse(items); // String[] items -> Element
                list.add(pe);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        //System.out.println("load successfully");
        return new ArrayList(list);
    }
```

ElementCSVImporter는 FileImporter를 import, export 하는 클래스로 CSV 파일만을 읽어오고 생성한다.

[6] FileLoader

```
public interface FileLoader<E> {
    List<E> load(String filePath); //파일 로드
}
```

FileLoader 인터페이스는 파일을 로드하는 추상 메소드를 가진다.

[7] ElementJSONLoader

```
public class ElementJSONLoader implements FileLoader<Element>{
    @Override
    public List<Element> load(String filePath) {
        System.out.println("file import: "+filePath);
        try (Reader reader = new FileReader(filePath)) {
            //Gson 라이브러리를 이용해 JSON -> List<Element> 변환
            List<Element> elementList
                = new Gson().fromJson(reader, new TypeToken<List<Element>>(){}.getType());
            return elementList;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

}
```

ElementJSONLoader는 FileLoader를 상속하며 com.google.gson.Gson을 이용해 JSON 파일을 Load 해온다.

[8] ElementXMLLoader

```
public class ElementXMLLoader implements FileLoader<solution.Element>{

    @Override
    public List<solution.Element> load(String filePath) {
        System.out.println("file import: "+filePath);
        List<solution.Element> elementList = new ArrayList<>();

        try {
            // org.w3c.dom.Element와 solution.Element의 혼동 방지를 위해 solution. 표기
            File xmlFile = new File(filePath);
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document document = dBuilder.parse(xmlFile);
            document.getDocumentElement().normalize();

            NodeList elementNodeList = document.getElementsByTagName(tagname:"Element");
```

ElementXMLLoader는 FileLoader를 상속하며 DocumentBuilder를 이용하여 XML 파일을 Load 해온다.

[9] FileLoaderImporterAdapter

```java
public class FileLoaderImporterAdapter<E> implements FileImporter<Element> {
    private FileLoader fileLoader; //FileImporter로 변환할 FileLoader

    //생성자
    public FileLoaderImporterAdapter(FileLoader fileLoader){
        this.fileLoader = fileLoader;
    }
    @Override
    public List<Element> importFile(String filePath) {
        //FileImporter.importFile -> fileLoader.load
        return fileLoader.load(filePath);
    }
    @Override
    public void exportFile(String filePath, List<Element> list) {
        // 매칭되는 메소드가 없어 구현하지 않음.
        // TODO Auto-generated method stub
        throw new UnsupportedOperationException(message:"Unimplemented method
    }
}
```

FileLoaderImporterAdapter는 FileLoader를 FileImporter 처럼 동작할 수 있도록 연결한다. FileLoader.load 와 FileImporter.importFile 이 매칭되고, exportFile은 매칭되는 메소드가 없어 구현하지 않았다.

[10] MainTest

```java
// Jeong woon Park (Lab7, Dankook University, Department of Software) 2023/11/19
public class MainTest {
    public static void Main(){
        //csv파일에서 PeriodicElement list load
        List<PeriodicElement> list = PeriodicElementImporter.loadCSV(filename:"PeriodicElements.csv");
        System.out.println(x:"\n\nDynamicArray add & print");
        //DynamicArray<Element> 선언
        DataCollection<Element> arr = new DynamicArray<>(list.size());
        //PeriodicElement -> Element 변환 후 arr 저장
        for (PeriodicElement e : list) {
            arr.put(new Element(e.getNumber(), e.getName(), e.getSymbol(), e.getWeight()));
        }
        //DynamicArray 동작 Test
        test(testName:"DynamicArray Test",list,arr);
```

[10-1] DynamicArray를 생성해 배열을 Element 객체로 채운 후, Test() 메소드를 통해 배열이 올바르게 동작하는 지 확인한다.

```java
//DataCollection이 올바르게 동작하는지 확인하는 Test Method
public static void test(String testName, List<PeriodicElement> list, DataCollection<Element> arr){
    System.out.println("\n**"+testName+"**");
    //[1] elemAt 동작 확인
    System.out.println(x:"[1] elemAt index 100");
    System.out.println(arr.elemAt(index:100));
    //[2] clear 동작 확인
    System.out.println(x:"[2] clear all ");
    arr.clear();
    System.out.println("length: " + arr.length());
    //[3] put 동작 확인
    System.out.println(x:"[3] put PeriodicElement 1~5 ");
    for(int i=0; i<5; i++){
        PeriodicElement e = list.get(i);
        arr.put(new Element(e.getNumber(), e.getName(), e.getSymbol(), e.getWeight()));
    }
    arr.forEach(System.out::println);
    //[4] insert 동작 확인
    System.out.println(x:"[4] insert PeriodicElement 6 into index 3");
    PeriodicElement e = list.get(index:5);
    arr.insert(index:3, new Element(e.getNumber(), e.getName(), e.getSymbol(), e.getWeight()));
    arr.forEach(System.out::println);
    //[5] remove 동작 확인
    System.out.println(x:"[5] remove index 3");
    arr.remove(index:3);
    arr.forEach(System.out::println);
    //[6] remove all using iterator 동작 확인
    System.out.println(x:"[6] remove all using iterator");
    Iterator<Element> it = arr.iterator();
    while(it.hasNext()){
        it.next();
        it.remove();
    }
    arr.forEach(System.out::println);
    System.out.println(x:"**End**\n\n");
}
```

[10-2] Test 메소드는 다음과 같이 구성된다. elemAt -> clear -> put -> insert -> remove -> remove all using iterator 순서로 올바른 결과가 나타나는지 확인한다.

```java
//list<Element> 생성 후 원소 추가
List<Element> elements = new ArrayList<>();
list.forEach(pe -> elements.add(new Element(pe.getNumber(), pe.getName(), pe.getSymbol(),pe.getWeight())));
//list -> DataCollection Adapter 전환
DataCollection<Element> arr2 = new ListDataCollectionAdapter<>(elements);
//List가 DataCollection처럼 동작하는지 확인 Test
test(testName:"List to DataCollection Test",list,arr2);
//Stack<Element> 생성 후 원소 추가
Stack<Element> stack = new Stack<>();
list.forEach(pe->stack.add(new Element(pe.getNumber(), pe.getName(), pe.getSymbol(),pe.getWeight())));
//Stack -> DataCollection Adapter 전환
DataCollection<Element> arr3 = new ListDataCollectionAdapter<>(stack);
//Stack이 DataCollection처럼 동작하는지 확인 Test
test(testName:"Stack to DataCollection Test", list, arr3);
```

[10-3] ListDataCollectionAdapter 가 잘 동작하는지 확인하는 코드이다. List와 Stack 객체를 만든 후 이를 Adapter에 넣고 DataCollection을 생성해 Test() 메소드를 통해 DataCollection으로서 동작이 올바른지를 확인한다.

```
//ElementJSONLoader -> FileImporter
FileImporter<Element> importer = new FileLoaderImporterAdapter<>(new ElementJSONLoader());
//FileImporter로 변환한 ElementJSONLoader로 json 파일 import
List<Element> elements2 = importer.importFile(filePath:"Elements.json");
//elements2.forEach(System.out::println);
// ElementXMLLoader -> FileImporter
importer = new FileLoaderImporterAdapter<>(new ElementXMLLoader());
//FileImporter로 변환한 ElementXMLLoader로 xml 파일 import
elements2 = importer.importFile(filePath:"Elements.xml");
//elements2.forEach(System.out::println);
// ElementCSVImporter
importer = new ElementCSVImporter();
//ElementCSVImporter로 파일 import, export
elements2 = importer.importFile(filePath:"Elements.csv");
//elements2.forEach(System.out::println);
importer.exportFile(filePath:"Elements1.csv", elements2);
```

[10-4] FileLoaderImporterAdapter와 importer, loader가 잘 동작하는지 확인하는 코드이다. ElementJSONLoader와 ElementXMLLoader를 FileImporter로 변환한 후 잘 import 되는지 확인한다. 그리고 ElementCSVImporter 가 잘 동작하는지 확인한다.

## [RESULT]

```
DynamicArray add & print

**DynamicArray Test**
[1] elemAt index 100
{ number='101', name='Mendelevium', symbol='Md', weight='258.0'}
[2] clear all
length: 0
[3] put PeriodicElement 1~5
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[4] insert PeriodicElement 6 into index 3
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='6', name='Carbon', symbol='C', weight='12.011'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[5] remove index 3
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[6] remove all using iterator
**End**
```

```
**List to DataCollection Test**
[1] elemAt index 100
{ number='101', name='Mendelevium', symbol='Md', weight='258.0'}
[2] clear all
length: 0
[3] put PeriodicElement 1~5
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[4] insert PeriodicElement 6 into index 3
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='6', name='Carbon', symbol='C', weight='12.011'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[5] remove index 3
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[6] remove all using iterator
**End**

**Stack to DataCollection Test**
[1] elemAt index 100
{ number='101', name='Mendelevium', symbol='Md', weight='258.0'}
[2] clear all
length: 0
[3] put PeriodicElement 1~5
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[4] insert PeriodicElement 6 into index 3
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='6', name='Carbon', symbol='C', weight='12.011'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[5] remove index 3
{ number='1', name='Hydrogen', symbol='H', weight='1.007'}
{ number='2', name='Helium', symbol='He', weight='4.002'}
{ number='3', name='Lithium', symbol='Li', weight='6.941'}
{ number='4', name='Beryllium', symbol='Be', weight='9.012'}
{ number='5', name='Boron', symbol='B', weight='10.811'}
[6] remove all using iterator
**End**

 file import: Elements.json
 file import: Elements.xml
 file import: Elements.csv
 file save: Elements1.csv
 jeongwoon@jeongwoonui-MacBookPro Lab7 %
```