REPORT

HW7

자바프로그래밍2

제출일 2023. 11.20

소속 컴퓨터공학과

학번 32183520

이름 이 주성

Adapter Pattern

- 서로 다른 인터페이스를 구현한 두 클래스를 함께 동작할 수 있도록 해주는 패턴
- 기존의 클래스를 수정하지 않고도 인터페이스를 변환해 클래스 간의 상호 작용을 가능하게 한다.
- 보통은 먼저 서로 인터페이스를 통일시켜 개발을 시작하지만 그러지 못했을 경우 어댑터 패턴으로 맞춰준다.

목표

1. List<E>를 지원하는 DataCollection<E> 만들기
2. FileImporter<E>를 지원하는 FileLoader<E> 만들기

Target 인터페이스

DataCollection<E>

- Iterable<E>를 확장
- put(), insert(), remove(), clear(), elemAt(), length()

```
// Iterable<E>를 확장하며 요소의 컬렉션을 나타내는 커스텀 컬렉션
public interface DataCollection<E> extends Iterable<E> {
   // 요소 추가
   void put(E e);

   // 인덱스에 요소 삽입
   void insert(int index, E e);

   // 주어진 인덱스의 요소 삭제
   void remove(int index);

   // 주어진 인덱스의 요소 반환
   E elemAt(int index);
```

```
    // 요소의 개수 반환
    int length();

    // 모든 요소 제거 후 초기화
    void clear();
}
```

DynamicArray<E>

- DataCollection<E> 인터페이스 구현
- DynamicArrayIterator<E> inner class 보유
    - `java.util.Iterator<E>` 구현
        - hasNext(), next(), remove()

```
// DataCollection<E> 인터페이스 구현
public class DynamicArray<E> implements DataCollection<E> {
    private static final int SIZE = 3;      // default capacity SIZE
    int length;         // 요소의 개수
    int capacity;       // 용량 (할당된 배열의 크기)
    E[] data;           // E 배열 -> Type Parameter로 인스턴스 생성 불가

    // 기본 생성자: 용량을 정해주지 않으면 default로 SIZE
    public DynamicArray() {
        this(SIZE);
    }

    // 생성자: capacity, length 설정 + capacity 크기의 data 배열 생성
    public DynamicArray(int capacity) {
        this.capacity = capacity;
        this.length = 0;
        this.data = (E[]) new Object[capacity];
    }

    // DataCollection<E> 인터페이스 메서드 구현..
    // put(), insert(), remove(), clear(), elemAt(), length()
    // ...

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

```
    }

    // DynamicArray<E>의 inner 클래스
    public class DynamicArrayIterator<E> implements Iterator<E> {
        // 다음 원소 index
        private int index = 0;

        // 다음 요소가 있는지 확인
        @Override
        public boolean hasNext() {
            // 다음 요소가 length보다 작으면 다음 원소가 있다는 뜻이므로 true
            return index < length;
        }

        // 다음 요소 반환
        @Override
        public E next() {
            // next()로 가져올 다음 원소가 없는 경우
            if (!hasNext()) {
                throw new NoSuchElementException();
            }

            // 다음 원소 조회 후 index 증가
            return (E) elemAt(index++);
        }

        // 현재 요소 제거
        @Override
        public void remove() {
            // index가 0인 경우 next()를 호출하지 않은 것이므로 index-1의 값을 삭제 할
수 없으니까 조건 처리
            if (index > 0) {
                // 다음 원소가 아닌 현재 원소를 삭제해야 하므로 --index 해준 후 삭제
                DynamicArray.this.remove(--index);
            } else {
                throw new IllegalStateException("next()가 먼저 호출된 후에
호출되어야 합니다.");
            }
        }
    }
}
```

Adaptee 인터페이스

List<E> 클래스

- ArrayList, LinkedList, Vector, Stack 등

Adapter 클래스

ListDataCollectionAdapter<E>

- List<E>를 구현한 클래스를 DataCollection<E> 인터페이스에 맞추는 역할
- 이 어댑터를 이용해 List<E>를 지원하는 DataCollection<E>를 만드는 것이 목표
    - List<E>의 메서드를 DataCollection<E>의 메서드에 연결

```
// List<E>를 구현한 클래스(Adaptee)를 DataCollection<E> 인터페이스(Target)에 맞추는 역할
public class ListDataCollectionAdapter<E> implements DataCollection<E> { // Target 인터페이스 구현
    // Adaptee
    private List<E> list;

    public ListDataCollectionAdapter(List<E> list) {
        this.list = list;
    }

    // 동일한 동작을 하는 Adaptee의 메서드 연결
    @Override
    public void put(E e) {
        list.add(e);
    }

    @Override
    public void insert(int index, E e) {
        list.add(index, e);
    }

    @Override
    public void remove(int index) {
        list.remove(index);
```

```
    }

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

    @Override
    public int length() {
        return list.size();
    }

    @Override
    public void clear() {
        list.clear();
    }

    // Iterator 반환 메서드 오버라이딩
    @Override
    public Iterator<E> iterator() {
        return list.iterator();
    }
}
```

Target 인터페이스

FileImporter

- 파일을 가져오기 위한 메서드: importFile
- 파일을 내보내기 위한 메서드: exportFile

```
public interface FileImporter<E> {
    // 파일 가져오기
    List<E> importFile(String filepath);

    // 파일 내보내기
    void exportFile(String filepath, List<E> list);
}
```

ElementCSVImporter<E>

- FileLoader<E> 인터페이스 구현
- CSV 형식의 파일에서 데이터를 가져옴

```
public class ElementCSVImporter implements FileImporter<Element> {
    // csv 파일에서 데이터를 읽고 그 데이터를 Element 객체 리스트로 만들어 반환
    @Override
    public List<Element> importFile(String filepath) {
        // 반환할 Element 객체 리스트
        List<Element> elist = new ArrayList<Element>();
        String line = "";

        // load data
        try (BufferedReader br = new BufferedReader(new FileReader(filepath))) {
            // 구분자
            String delimiter = ",";

            // 파일의 끝까지 읽기
            while ((line = br.readLine()) != null) {
                // "#" 이 포함된 line은 continue
                if (line.contains("#")) {
                    continue;
                }

                // 구분자를 기준으로 스플릿
                String[] items = line.split(delimiter);

                // 데이터를 Element 객체로 파싱
                Element e = parse(items);

                // 리스트에 추가
                elist.add(e);
            }

        // 예외 처리
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        return elist;
    }

    // filepath 위치에 Element 리스트를 csv 파일로 내보내기
    @Override
    public void exportFile(String filepath, List<Element> list) {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filepath))){

            // Element 리스트를 돌면서 파일에 한줄씩 이어쓰기
            for (Element e : list) {
                bw.append(e.getDescription()).append("\n");
            }

        // 예외 처리
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // String 배열을 파싱해 Element 객체로 만들어 반환하는 메소드
    public Element parse(String[] items) {
        try {
            // PeriodicElement 필드값으로 채울 변수 저장
            int number = Integer.parseInt(items[0]);
            String name = items[1];
            String symbol = items[2];
            double weight = Double.parseDouble(items[3]);

            // 파싱한 변수들로 Element 생성 후 반환
            return new Element(number, name, symbol, weight);

            // 예외처리
        } catch (Exception e) {
            System.out.println("\n[ Error 발생!! ]:  " + e.getMessage() + "\n\n");
        }

        return null;
    }
}
```

Adaptee 인터페이스

FileLoader<E>

- 파일에서 데이터 로드하는 메서드: load
- 파일을 저장하는 메서드: save

```
public interface FileLoader<E> {
   // 파일 가져오기
   List<E> load(String filepath);

   // 파일 내보내기
   void save(String filepath, List<E> list);
}
```

ElementJSONLoader

- JSON 파일에서 데이터를 읽고 그 데이터를 Element 객체로 변환
- `com.google.gson.Gson` 라이브러리 사용

```
// JSON 파일에서 데이터를 읽고 그 데이터를 Element 객체로 변환
public class ElementJSONLoader implements FileLoader<Element> {

   @Override
   public List<Element> load(String filepath) {
       // 반환할 Element 객체 리스트
       List<Element> elist = new ArrayList<Element>();
       String line = "";

       // Gson 라이브러리 사용
       Gson gson = new Gson();

       // load data
       try (BufferedReader br = new BufferedReader(new FileReader(filepath))) {
           StringBuilder jsonString = new StringBuilder();

           // 파일의 끝까지 읽기
           while ((line = br.readLine()) != null) {
               // JSON 파일을 읽어와서 문자열로 저장
               jsonString.append(line);
```

```
            }

            // JSON 문자열을 List<Element> 객체로 변환
            Type elementType = new TypeToken<List<Element>>(){}.getType();
            elist = gson.fromJson(jsonString.toString(), elementType);

        // 예외 처리
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        return elist;
    }

    @Override
    public void save(String filepath, List<Element> list) {
        // Gson 라이브러리 사용
        Gson gson = new GsonBuilder().setPrettyPrinting().create();

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filepath))) {
            // List<Element> 객체를 JSON 문자열로 변환 후 파일에 쓰기
            gson.toJson(list, bw);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ElementXMLLoader

- XML 파일에서 데이터를 읽고 그 데이터를 Element 객체로 변환
- `org.w3c.dom.Element`, `Document`, `NodeList` 등 사용

```
// XML 파일에서 데이터를 읽고 그 데이터를 Element 객체로 변환
public class ElementXMLLoader implements FileLoader<Element> {

    @Override
```

```
    public List<Element> load(String filepath) {
        // 반환할 Element 객체 리스트
        List<Element> elist = new ArrayList<Element>();

        // load data
        try {
            // XML 파일을 파싱해 Document 객체 생성
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document document = dBuilder.parse(filepath);

            // <Elements> 엘리먼트 얻기 + normalize()로 정규화
            document.getDocumentElement().normalize();

            // <Element> 리스트 얻기
            NodeList nodeList = document.getElementsByTagName("Element");

            // nodeList 순회하면서 XML 데이터를 추출해 Element 객체로 만들기
            for (int i = 0; i < nodeList.getLength(); i++) {
                org.w3c.dom.Element node = (org.w3c.dom.Element) nodeList.item(i);

                // XML에서 데이터 추출해 Element 객체 생성
                int number =
Integer.parseInt(node.getElementsByTagName("Number").item(0).getTextContent());
                String name =
node.getElementsByTagName("Name").item(0).getTextContent();
                String symbol =
node.getElementsByTagName("Symbol").item(0).getTextContent();
                double weight =
Double.parseDouble(node.getElementsByTagName("Weight").item(0).getTextContent());

                Element element = new Element(number, name, symbol, weight);

                // 리스트에 추가
                elist.add(element);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return elist;
    }

    // XML 파일로 저장
```

```
    @Override
    public void save(String filepath, List<Element> list) {
        try {
            // DocumentBuilder 및 Document 객체 생성
            DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document document = dBuilder.newDocument();

            // 최상위 엘리먼트 <Elements> 생성
            org.w3c.dom.Element elements = document.createElement("Elements");
            document.appendChild(elements);

            // Element 객체 리스트 순회하면서 XML 엘리먼트 생성
            for (Element e : list) {
                // <Element> 생성
                org.w3c.dom.Element element = document.createElement("Element");

                // <Number> 생성 후 값 설정
                Node number = document.createElement("Number");

number.appendChild(document.createTextNode(String.valueOf(e.getNumber())));
                element.appendChild(number);

                // <Name> 생성 후 값 설정
                Node name = document.createElement("Name");

name.appendChild(document.createTextNode(String.valueOf(e.getName())));
                element.appendChild(name);

                // <Symbol> 생성 후 값 설정
                Node symbol = document.createElement("Symbol");

symbol.appendChild(document.createTextNode(String.valueOf(e.getSymbol())));
                element.appendChild(symbol);

                // <Weight> 생성 후 값 설정
                Node weight = document.createElement("Weight");

weight.appendChild(document.createTextNode(String.valueOf(e.getWeight())));
                element.appendChild(weight);

                // <Element> 엘리먼트를 <Elements> 엘리먼트에 추가
                elements.appendChild(element);
            }
```

```
            // filepath에 XML 파일로 저장
            TransformerFactory.newInstance().newTransformer().transform(new
DOMSource(document), new StreamResult(filepath));
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Adapter 클래스

- FileLoader<E>를 받아 FileImporter<E>로 바꿔주는 어댑터 클래스
- Adaptee: FileLoader<E>
- Target: FileImporter<E>

```
// FileLoader<E>(Adaptee)를 받아 FileImporter<E>(Target)로 바꿔주는 어댑터 클래스
public class FileLoaderImporterAdapter<E> implements FileImporter<E> {
    // Adaptee(기존 클래스): FileLoader<E>
    FileLoader<E> adaptee;

    public FileLoaderImporterAdapter(FileLoader<E> loader) {
        this.adaptee = loader;
    }

    // adaptee의 메서드로 연결
    @Override
    public List<E> importFile(String filepath) {
        return adaptee.load(filepath);
    }

    @Override
    public void exportFile(String filepath, List<E> list) {
        adaptee.save(filepath, list);
    }
}
```

MainTest

- ArrayList를 DataCollection으로 변환해서 사용하기

```
// ArrayList 생성
List<Element> arrayList = new ArrayList<>();

// ArrayList(Adaptee)를 ListDataCollectionAdapter(Adapter)를 이용해
DataCollection(Target)으로 사용하기
DataCollection<Element> arrayDataList = new
ListDataCollectionAdapter<>(arrayList);

// 원소 put
peList.forEach(e -> arrayDataList.put(new Element(e.getNumber(), e.getName(),
       e.getSymbol(), e.getWeight())));

// remove()
arrayDataList.remove(0);

// insert()
arrayDataList.insert(5, new Element(0, "No", "N", 0L));

// elemAt()
Element e2 = arrayDataList.elemAt(5);

// clear()
arrayDataList.clear();
```

- Stack을 DataCollection으로 변환해서 사용하기
    - ArrayList와 동일한 방법

- FileLoader를 FileImporter로 변환해 사용하기
    - load
    - save

```
// ElementJSONLoader - Import

FileImporter<Element>   importer   =   new   FileLoaderImporterAdapter<>(new
ElementJSONLoader());

List<Element>                              elements                              =
```

```
importer.importFile("ElementsCSVJSONXML/Elements.json");

// ElementJSONLoader - Export
importer.exportFile("ElementsCSVJSONXML/myJSON.json", elements);

// ElementXMLLoader - Import
importer = new FileLoaderImporterAdapter<>(new ElementXMLLoader());
elements = importer.importFile("ElementsCSVJSONXML/Elements.xml");

// ElementXMLLoader - Export
importer.exportFile("ElementsCSVJSONXML/myXML.xml", elements);
```

결과

```
ElementJSONLoader Load Elements Size: 118

[ JSON File Export: ElementsCSVJSONXML/myJSON.json ]

ElementXMLLoader Load Elements Size: 118

[ XML File Export: ElementsCSVJSONXML/myXML.xml ]

ElementCSVImporter Load Elements Size: 118

[ CSV File Export: ElementsCSVJSONXML/myCSV.csv ]
```