

자료구조 실습 보고서

[제05주] 리스트 기본 기능

2021년 04월 04일

201702039 오명주

1. 프로그램 설명서

(1) 프로그램의 전체 설계 구조

➔ MVC (Model – View – Controller) 구조

Model : 프로그램이 "무엇"을 할 것인지 정의. 사용자의 요청에 맞는 알고리즘을 처리하고 DB와 상호작용하여 결과물을 산출하고 Controller에게 전달.

View : 화면에 무엇인가를 "보여주기 위한" 역할. 최종 사용자에게 "무엇"을 화면으로 보여줌.

Controller : 모델이 "어떻게" 처리할 지 알려주는 역할. 사용자로부터 입력을 받고 중개인 역할. Model과 View는 서로 직접 주고받을 수 없음. Controller을 통해 이야기함.

➔ LinkedList 학생 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

- LinkedList<T>: 학생 리스트의 기능을 배열로 구성한다.
- Student : 학생의 변수, 속성을 구성한다.
- Iterator : 반복자 인터페이스로, 리스트 출력 시 사용한다.
- LinedNode<T> : 노드를 구성한다.

View :

- AppView : 프로그램의 입/출력을 담당한다.

Controller :

- ApplicationController : AppView를 통해 수행할 메뉴 번호를 입력 받아 Model에 해당하는 클래스들에 전달하고 결과물을 AppView를 통해 출력한다.

➔ List로 구현(Bag, Set과의 차이)

- List는 순서가 있음. 중복을 허용함.
- Bag은 순서가 없음. 중복을 허용함.
- Set은 순서가 없음. 중복을 허용하지 않음.

(2) 함수 설명서

➔ 주요 알고리즘

```
// 주어진 원소가 포함되어있는지 확인
public boolean doesContain(T anElement) {
    return (orderOf(anElement) != -1); // orderOf : 존재하지 않으면 -1을 반환
}
```

▶ doesContain : 리스트에서 주어진 원소가 포함되어 있는지 여부를 반환하는 함수. orderOf 함수를 이용하여 확인한다. (orderOf 함수는 아래에서 설명)

```
// 주어진 순서의 원소를 반환
public T elementAt(int order) {
    if (this.anElementDoesExistAt(order)) { // 순서가 유효한지 확인
        ListNode<T> currentNode = this._head; // head를 current로 설정
        int nodeCount = 0;
        while (nodeCount < order) { // 순서만큼 반복
            currentNode = currentNode.next(); // 걸어나감
            nodeCount++;
        }
        return currentNode.element(); // 해당 순서의 원소 반환
    } else {
        return null; // 순서가 유효하지 않다면 null을 반환
    }
}
```

▶ elementAt 함수: 주어진 순서의 원소를 반환하는 함수. 주어진 순서가 유효한지 확인하여 유효하다면 순서만큼 next() 함수를 통해 걸어 나가서 해당 순서의 원소를 반환한다.

```
// 리스트 첫번째 원소를 반환
public T first() {
    if (this.isEmpty()) {
        return null; // 마지막 원소가 존재할 수 없으므로
    } else {
        return elementAt(0);
        // 또는 이렇게 : return this._head.element()
    }
}

// 리스트 마지막 원소를 반환
public T last() {
    if (this.isEmpty()) {
        return null; // 마지막 원소가 존재할 수 없으므로
    } else {
        return elementAt(this.size() - 1);
    }
}
```

- ▶ First, Last 함수 : 리스트의 처음과 끝 원소를 반환한다. 리스트가 비어 있다면 null을 반환하고, 원소가 있다면, elementAt함수를 이용하여 반환한다.

```
// 주어진 원소의 순서를 반환
public int orderOf(T anElement) { // 순차 검색
    int order = 0;
    ListNode<T> currentNode = this._head; // head를 current로 설정
    // 리스트 크기만큼 돌거나 원소를 찾을 때까지 반복
    while (currentNode != null && (!currentNode.element().equals(anElement))) {
        order++;
        currentNode = currentNode.next();
    }
    if (currentNode == null) { // Not Found
        return -1; // 존재하지 않으면 -1을 돌려주기로 한다
    } else {
        return order;
    }
}
```

- ▶ orderOf 함수 : 주어진 원소가 리스트 안에 존재하면 해당 원소의 인덱스를 반환, 존재하지 않으면 -1을 반환하는 함수. 리스트 전체 다 돌거나 주어진 원소와 동일한 원소를 발견하면 해당 순서를 변수에 저장하고 반복문을 나온다. 객체 비교에는 equals 함수를 활용한다.

```
// 주어진 원소를 주어진 순서에 삽입
public boolean addTo(T anElement, int order) {
    if ((order < 0) || (order > this.size())) { // order가 유효한지 검사
        return false;
    } else if (this.isFull()) { // 가득 차 있는지 확인 -> LinkedList에서 무의미
        return false;
    } else {
        ListNode<T> nodeForAdd = new ListNode<T>(anElement, null);
        if (order == 0) { // 맨 앞 순서에 삽입. 앞 (previous) 노드가 존재하지 않는다
            nodeForAdd.setNext(this._head);
            this._head = nodeForAdd;
        } else { // 순서가 맨 앞이 아니므로, 반드시 앞 (previous) 노드가 존재한다
            ListNode<T> previousNode = this._head;
            for (int i = 1; i < order; i++) {
                previousNode = previousNode.next(); // 삽입할 위치의 앞 노드를 찾는다
            }
            nodeForAdd.setNext(previousNode.next());
            previousNode.setNext(nodeForAdd);
        }
        this._size++;
        return true;
    }
}
```

- ▶ addTo 함수 : 주어진 원소를 주어진 순서에 삽입하는 함수. LinkedList에는 최대 개수가 존재하지 않으므로 먼저 주어진 순서가 유효한지 (리스트에 존재하는 순서의 숫자인지) 확인한다. 유효하지 않으면 false를 반환, 유효한 숫자라면 삽입할 순서가 0인 경우와 아닌 경우로 나눈다. 0이라면 head로 설정해주고 맨 앞이 아니라면 previous 노드 위치를 찾아 삽입을 진행한다.
addFirst, addLast, add 함수 모두 addTo 함수를 이용하여 구현 가능하다. (삽입할 위치를 매개변수로

지정하는 형태)

```
// 주어진 순서의 원소를 삭제
public T removeFrom(int order) {
    if (!this.anElementDoesExistAt(order)) { // 삭제할 원소가 없거나, 잘못된 위치
        return null;
    } else {
        // 리스트는 비어 있지 않으며, 삭제할 원소가 있음
        LinkedNode<T> removedNode = null;
        if (order == 0) { // 삭제할 원소가 맨 앞 원소
            removedNode = this._head;
            this._head = this._head.next();
        } else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
            LinkedNode<T> previousNode = this._head;
            for (int i = 1; i < order; i++) {
                previousNode = previousNode.next(); // 삭제할 위치의 앞 노드를 찾는다
            }
            removedNode = previousNode.next();
            previousNode.setNext(removedNode.next());
        }
        this._size--; // 사이즈 감소
        return removedNode.element();
    }
}
```

- ▶ removeFrom 함수 : 주어진 순서의 원소를 삭제하는 함수. anElementDoesExistAt 함수로 숫자가 유효한지 확인한다. 만약 anElementDoesExistAt에서 Size가 0이라면 false를 반환한다. 주어진 숫자가 유효하다면 삭제할 원소가 맨 앞인지 아닌지 나누어 구현한다. 맨 앞 원소를 삭제하는 경우 head 다음의 원소를 head로 설정한다. 맨 앞 원소가 아니라면 반복문을 이용하여 걸어 나가서 previous 노드를 찾아 삭제를 진행한다.

removeFirst, removeLast, removeAny 함수 모두 removeFrom 함수를 이용하여 구현 가능하다. (삭제할 위치를 매개변수로 지정하는 형태)

```
// 주어진 순서의 원소를 주어진 원소로 교체
public boolean replaceAt(T anElement, int order) {
    if (!this.anElementDoesExistAt(order)) {
        // 대체할 노드가 없거나, 잘못된 위치
        return false;
    } else {
        LinkedNode<T> currentNode = this._head; // head로 지정
        for (int i = 0; i < order; i++) { // 순서만큼 반복하여 위치 찾는다
            currentNode = currentNode.next();
            // 원소를 대체할 노드를 찾는다
        }
        currentNode.setElement(anElement);
        return true;
    }
}
```

- ▶ replaceAt 함수 : 주어진 순서의 원소를 주어진 원소로 교체하는 함수. 마찬가지로 anElementDoesExistAt 함수를 이용하여 순서가 유효한지 확인한다. 유효하다면 주어진 순서만큼 반복문을 이용하여 위치를 찾아 교체를 진행한다.

(3) 종합 설명서

➔ 프로그램 실행 순서대로 설명해보자.

```
public class _DS05_201702039_오명주L {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        ApplicationController appController = new ApplicationController() ;  
        // ApplicationController가 실질적인 main class 이다  
        appController.run() ;  
        //여기 main() 예서는 앱 실행이 시작되도록 해주는 일이 전부이다  
    }  
}
```

Main 클래스에서는 ApplicationController 객체 생성 후 run함수만 호출한다. ArrayList와 헷갈리지 않기 위해 LinkedList를 의미하는 L를 붙여서 구별하였다.

```
public enum MainMenu {  
    Error,  
  
    DoesContain, ElementAt, First, Last, OrderOf,  
  
    AddTo, AddToFirst, AddToLast, Add,  
  
    RemoveFrom, RemoveFirst, RemoveLast, RemoveAny,  
  
    ReplaceAt,  
  
    EndOfRun;
```

Enum class MainMenu에는 각각의 함수를 호출할 메뉴를 미리 선언하였다. Ordinal 함수를 이용하여 각각의 메뉴를 차례로 Error==1, DoesContain==2 .. 와 같은 형태로 숫자로 사용한다. Enum 클래스에서는 각각의 메뉴 번호를 관리하는 value 함수를 선언한다.

```
// Public Methods  
public void run() {  
    AppView.outputLine("<<< 리스트 기능 확인 프로그램을 시작합니다 >>>");  
  
    MainMenu selectedMenuValue = this.selectMenu(); // enum을 하나의 type으로 사용하여 입력값 관리  
    while (selectedMenuValue != MainMenu.EndOfRun) { // 프로그램 종료되는 '99'가 들어오지 않은 동안 반복  
        switch (selectedMenuValue) {  
            case DoesContain:  
                this.doesContain();  
                break;
```

Main에서 호출된 ApplicationController의 run 함수에서는 리스트 기능 확인 프로그램을 시작하는 출력문과 각각의 메뉴를 호출하고 수행할 입력 값을 받아 switch문으로 관리하는 것을 확인할 수 있다.


```
// Enum class를 이용하여 입력값을 관리 - enum을 하나의 type처럼 사용
private MainMenu selectMenu() {
    AppView.outputLine("");
    this.showList();
    this.showMenu();
    // 정수가 아닌 입력값을 관리하는 try-catch문
    try {
        int selectedMenuNumber = AppView.inputInteger(); // 사용자로부터 수행할 메뉴 번호를 입력받음
        // "NumberFormatException" can occur. It will be caught.
        MainMenu selectedMenuValue = MainMenu.value(selectedMenuNumber);
        if (selectedMenuValue == MainMenu.Error) { // Enum class에 없는 메뉴 번호를 입력했을 경우
            AppView.outputLine("!오류: 작업 선택이 잘못되었습니다. (잘못된 메뉴 번호: " + selectedMenuNumber + ")"); // 오류 출력문 출력
        }
        return selectedMenuValue;
    } catch (NumberFormatException e) { // 정수가 아닌 잘못된 포맷을 입력했을 경우
        AppView.outputLine("!오류: 입력된 메뉴 번호가 정수 숫자가 아닙니다. "); // 오류 출력
        return MainMenu.Error;
    }
}
```

run에서 호출하는 selecMenu 함수이다. 현재 리스트 원소들을 보여주는 showList 함수와 메뉴를 출력해주는 showMenu 함수를 호출하고있다. Try-catch문을 이용하여 정수가 아닌 값에 대한 에러처리를 하였고 메뉴에 없는 숫자(ReplaceAt보다 높은 숫자, 99도 아닌 경우)도 오류로 처리하였다.

```
// 현재 리스트를 보여주는 함수
private void showList() {
    AppView.output("! 현재의 리스트 원소들: [");
    Student student = null;
    Iterator<Student> iterator = this.list().iterator(); // iterator 객체 사용
    while (iterator.hasNext()) { // 다음 원소가 존재하는 동안 반복
        student = iterator.next(); // 걸어 나가며 객체 저장
        AppView.output(" " + student.score()); // 저장한 객체 값을 출력
    }
    AppView.outputLine(" ]");
}
```

리스트를 출력하는 showList 함수이다. Iterator을 이용하여 구현하였다.

- Student 타입을 갖는 iterator을 선언한다
- Iterator의 다음 원소가 있는 경우 반복하는 while문 사용. Student 객체에 iterator의 학생 데이터가 들어가게 되어 score을 출력하는 형태이다.

➔ Iterator

```
public interface Iterator<T> {
    public boolean hasNext(); // 다음 원소가 존재하는지 여부 반환
    public T next(); // 다음 원소를 얻어냄. 없으면 null반환
}
```

Interface로 선언된 Iterator이다. Generic type이고 인터페이스에 선언된 함수는 반드시 구현해야 하는 약속을 의미한다.

```

// Inner Class "ListIterator"의 선언
private class ListIterator implements Iterator<T> {

    private int _nextPosition; // 배열에서의 다음 원소 위치

    // 다음 위치 반환하는 getter
    private int nextPosition() {
        return this._nextPosition;
    }

    // 다음 위치 설정하는 setter
    private void setNextPosition(int newNextPosition) {
        this._nextPosition = newNextPosition;
    }

    // 생성자
    private ListIterator() {
        this.setNextPosition(0);
    }

    // 다음 원소가 존재하는지를 알아낸다
    @Override
    public boolean hasNext() {
        return (this.nextPosition() < ArrayList.this.size());
    }

    // 다음 원소를 얻어낸다. 없으면 null을 얻는다
    @Override
    public T next() {
        T nextElement = null;
        if (this.hasNext()) { // 다음 원소가 존재하는 동안 반복
            nextElement = ArrayList.this.elements()[this.nextPosition()];
            this.setNextPosition(this.nextPosition() + 1); // 건너나간다
        }
        return nextElement;
    }
}

```

- Iterator에 대한 구현 클래스는 LinkedList 내부 클래스로 한다. Iterator는 Appcontroller을 통해 List에 접근해야만 하는데 iterator의 역할 자체가 List에 접근하여 원소를 반복해서 출력하는 역할. 따라서 내부 클래스로 선언하여 자유롭게 직접적으로 접근할 수 있도록 한다.
- 생성자, 다음 위치에 대한 반환과 설정을 하는 getter/setter, 인터페이스에 선언한 hasNext와 next 함수가 구현되어 있음을 확인할 수 있다.

2. 프로그램 장단점 / 특이점 분석

➔ 장점

- MVC 모델을 이용하여 가독성과 생산성이 뛰어나다. 각 클래스, 함수의 역할이 분명해서 코드와 프로그램을 잘 이해할 수 있다.
- 유사한 기능을 하는 다른 프로그램에도 재사용할 수 있다. 객체 지향 프로그램의 가장 큰 장점이라고 할 수 있다.
- 수정이 편리하다. 데이터나 기능을 수정하려고 하면 해당 메소드만 수정하면 되기 때문에 편리하다.
- Generic 타입을 활용하여 프로그램 성능저하를 유발하는 Type Casting을 제거한다.
코드 절약 및 코드 재사용성을 증진시켜 유지보수가 편하다
엄격한 데이터 타입 체크를 가능하게 한다.
- ArrayList의 설계가 잘 되어있어서 LinkedList로 바꾸는 것이 용이했다. Node 클래스를 생성하고 List 클래스의 함수를 바꾸고 ApplicationController의 ArrayList만 LinkedList로 바꾸니 잘 실행되었다.
- 하나의 함수로 다른 함수 구현에 사용하였다. (addTo->add, addFirst, addLast / removeFrom->removeFirst, removeLast, removeAny) 매개변수를 주는 것으로 구현할 수 있다.

➔ 단점

- 보기 좋고 기능을 확연하게 나누려다 보니 설계는 잘 되어있지만 메모리를 많이 차지한다.
Enum을 설정하고, iterator을 사용하여 구현하니 코드가 길어졌다.

3. 실행 결과 분석

(1) 입력과 출력 (화면 capture하여 제출)

_DS05_201702039_오명주L (2) [Java Application] C:\Program Files\Java\jdk-12.0.1\bin\javaw.exe (2021. 4. 5. 오후 8:18:59)

<<< 리스트 기능 확인 프로그램을 시작합니다 >>>

! 현재의 리스트 원소들: []

> 해야 할 작업의 번호를 선택해야 합니다:

DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,

AddTo=6, AddToFirst=7, AddToLast=8, Add=9,

RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99

? 작업 번호를 입력하시오: 7

!AddToFirst 작업을 실행합니다 :

? 점수를 입력하시오: 11

! 입력된 점수 (11)의 학생을 [맨 앞]에 삽입하는 작업을 성공하였습니다.

! 현재의 리스트 원소들: [11]

> 해야 할 작업의 번호를 선택해야 합니다:

DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,

AddTo=6, AddToFirst=7, AddToLast=8, Add=9,

RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99

? 작업 번호를 입력하시오: 8

!AddToLast 작업을 실행합니다 :

? 점수를 입력하시오: 22

! 입력된 점수 (22)의 학생을 [맨 뒤]에 삽입하는 작업을 성공하였습니다.

! 현재의 리스트 원소들: [11 22]

> 해야 할 작업의 번호를 선택해야 합니다:

DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,

AddTo=6, AddToFirst=7, AddToLast=8, Add=9,

RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99

? 작업 번호를 입력하시오: 6

!AddTo 작업을 실행합니다 :

? 리스트에서의 순서 번호를 입력하시오: 1

? 점수를 입력하시오: 33

! 입력된 순서 [1]에 입력된 점수 (33)의 학생을 삽입하는 작업을 성공하였습니다.

! 현재의 리스트 원소들: [11 33 22]

> 해야 할 작업의 번호를 선택해야 합니다:

DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,

AddTo=6, AddToFirst=7, AddToLast=8, Add=9,

RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99

? 작업 번호를 입력하시오: 9

!Add 작업을 실행합니다 :

? 점수를 입력하시오: 44

! 입력된 점수 (44)의 학생을 [임의의 순서]에 삽입하는 작업을 성공하였습니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 1

! DoesContain 작업을 실행합니다 :
? 점수를 입력하시오: 55
! 입력된 점수 (55)의 학생이 리스트에 존재하지 않습니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 1

! DoesContain 작업을 실행합니다 :
? 점수를 입력하시오: 33
! 입력된 점수 (33)의 학생이 리스트에 존재합니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 2

!ElementAt 작업을 실행합니다 :
? 리스트에서의 순서 번호를 입력하시오: 1
! 입력된 순서 [1]의 학생의 점수는 (11) 입니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 3

! FirstElement 작업을 실행합니다 :
! [맨 앞] 학생의 점수는 (44) 입니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 4

!LastElement 작업을 실행합니다 :
! [맨 뒤] 학생의 점수는 (22) 입니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 5

!OrderOf 작업을 실행합니다 :
? 점수를 입력하시오: 55
! 입력된 점수 (55)의 학생이 리스트에 존재하지 않습니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 5

!OrderOf 작업을 실행합니다 :
? 점수를 입력하시오: 22
! 입력된 점수 (22)의 학생의 순서는 [3] 입니다.

! 현재의 리스트 원소들: [44 11 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 10

!RemoveFrom 작업을 실행합니다 :
? 리스트에서의 순서 번호를 입력하시오: 1
! 입력된 순서 [1]에서 삭제된 학생의 성적은 (11) 입니다.

! 현재의 리스트 원소들: [44 33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 11

!RemoveFirst 작업을 실행합니다 :
! 삭제된 [맨 앞] 학생의 성적은 (44) 입니다.

! 현재의 리스트 원소들: [33 22]
> 해야 할 작업의 번호를 선택해야 합니다:
DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 12

!RemoveLast 작업을 실행합니다 :
! 삭제된 [맨 뒤] 학생의 성적은 (22) 입니다.

```

! 현재의 리스트 원소들: [ 33 ]
> 해야 할 작업의 번호를 선택해야 합니다:
  DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
  AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
  RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 14

!ReplaceAt 작업을 실행합니다 :
? 리스트에서의 순서 번호를 입력하시오: 0
? 점수를 입력하시오: 22
! 주어진 순서 [0]의 학생의 점수가 (22)로 바뀌었습니다.

! 현재의 리스트 원소들: [ 22 ]
> 해야 할 작업의 번호를 선택해야 합니다:
  DoesContain=1, ElementAt=2, First=3, Last=4, OrderOf=5,
  AddTo=6, AddToFirst=7, AddToLast=8, Add=9,
  RemoveFrom=10, RemoveFirst=11, RemoveLast=12, RemoveAny=13, ReplaceAt=14, EndOfRun=99
? 작업 번호를 입력하시오: 99

> 리스트 정보입니다:
! 학생 수: 1
! 현재의 리스트 원소들: [ 22 ]

<<< 리스트 기능 확인 프로그램을 종료합니다 >>>

```

(2) 결과 분석 (자신의 논리적 평가, 기타 느낀 점)

- ⇒ 지난번 과제에 이어 Generic class를 이용하여 <student> 클래스 타입을 지정하여 코드가 깔끔하였다. 이해하기 쉽지 않았는데 틀을 잡아주고 함수 구현을 하니 이해 하는데 많이 도움이 된 것 같다.
- ⇒ 함수들이 서로 연관되어 있고 한 함수만 구현하여도 그 함수를 이용하여 다른 함수를 구현할 수 있는게 편리했다. 하지만 ArrayList에서 addTo 할 때, makeRoomAt과 같은 삽입할 공간을 확보하는 등의 함수를 분리하였는데, 이 함수들 없이 코딩이 가능함에도 분리하여 코드가 길어졌다고 생각한다. 프로그램 크기가 커지면 유용할 것이라고 생각한다.
- ⇒ 설계가 잘 되어있어서 구현이 다름에도 불구하고 ArrayList에서 LinkedList로의 변경이 용이했다.
- ⇒ Add, Remove 에서 임의의 원소를 삽입, 삭제할 때 효율적인 위치?
 - addToFirst의 경우 headNode에 삽입하면 되지만 addToLast의 경우 반복문을 통해 size만큼 걸어 나가서 삽입해야 한다. 그렇기 때문에 임의의 원소를 넣기에 더 적절한 함수는 addToFirst 라고 할 수 있다.
 - removeFirst의 경우 첫번째 원소 삭제시 head를 삭제하면 되지만 removeLast의 경우 반복문을 통해 size만큼 걸어 나가서 삭제해야 한다. 그러므로 임의의 원소를 삭제하기에 더 적절한 함수는 removeFirst 라고 할 수 있다.

4. 소스코드

```
public class _DS05_201702039_오명주L {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        AppController appController = new AppController() ;  
        // AppController가 실질적인 main class 이다  
        appController.run() ;  
        //여기 main() 에서는 앱 실행이 시작되도록 해주는 일이 전부이다  
    }  
}  
  
1 import java.util.Scanner;  
2  
3 public class AppView {  
4  
5     private static Scanner scanner = new Scanner(System.in);  
6  
7     // 생성자는 private 이어야함 - 생성자 불필요  
8     private AppView() {  
9  
10    }  
11  
12    // public OUTPUT methods  
13    public static void outputDebugMessage(String message) {  
14  
15    }  
16  
17    // 출력 관련 함수  
18    // 한줄을 출력하는 함수 (한줄이 띄워지지않는다)  
19    public static void output(String message) {  
20        System.out.print(message); // 입력받은 message를 출력한다  
21    }  
22  
23    // 한줄을 출력하는 함수 (한줄이 띄워진다)  
24    public static void outputLine(String message) {  
25        System.out.println(message); // 입력받은 message를 출력한다  
26    }  
27  
28    // 정수가 아닌 경우의 예러 처리를 보완할 것 : exception throws  
29    public static int inputInteger() throws NumberFormatException {  
30        return Integer.parseInt(AppView.scanner.next());  
31    }  
32 }  
  
public interface Iterator<T> {  
    public boolean hasNext(); // 다음 원소가 존재하는지 여부 반환  
    public T next(); // 다음 원소를 얻어냄. 없으면 null반환  
}
```



```

public enum MainMenu {
    Error,

    DoesContain, ElementAt, First, Last, OrderOf,

    AddTo, AddToFirst, AddToLast, Add,

    RemoveFrom, RemoveFirst, RemoveLast, RemoveAny,

    ReplaceAt,

    EndOfRun;

    public static final int END_OF_RUN = 99; // 프로그램 종료

    public static MainMenu value(int menuNumber) {
        if (menuNumber == END_OF_RUN) { // 99가 들어오면
            return MainMenu.EndOfRun; // 프로그램 종료
        } else if (menuNumber < DoesContain.ordinal() || menuNumber > ReplaceAt.ordinal()) {
            // ordinal() 함수는 Enum 값이 선언된 순서를 얻는다
            return MainMenu.Error; // Enum에 선언되지 않은 값이면 Error처리
        } else {
            // values() 함수는 모든 enum 값들을 그 순서대로 배열로 만들어준다
            return MainMenu.values()[menuNumber];
        }
    }
}

```

```

public class Student {
    // Constants
    private static final int DEFAULT_SCORE = 0;
    // Private Instance Variables
    private int _score;

    // Getters/Setters
    public int score() {
        return this._score;
    }

    public void setScore(int newScore) {
        this._score = newScore;
    }

    // Constructor
    public Student() {
        this.setScore(Student.DEFAULT_SCORE);
    }

    public Student(int givenScore) {
        this.setScore(givenScore);
    }

    @Override
    public boolean equals(Object aStudent) {
        if (aStudent.getClass() != Student.class) {
            return false;
        } else {
            return (this.score() == ((Student) aStudent).score());
        }
    }
}

```