자료구조 실습 보고서

[제 03 주] LinkedBag(연결체인) 을 이용하는 동전가방

제출일	2017/03/26
학 번	201000287
소 속	일어일문학과
이 름	유다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 입력
    - 처음 가방에 들어갈 최대 가방 사이즈를 입력한다.
    - 1 이 입력되면 코인 액수를 입력 받아 가방에 넣는다
    - 2 가 입력되면 코인의 액수를 입력 받아 가방에서 해당 코인을 삭제한다.
    - 3 이 입력되면 총 코인의 개수와 가장 큰 코인의 값, 현재 있는 코인의 총 금액을 출력한다.
    - 4 가 입력되면 코인의 액수를 입력 받아 해당 코인을 검색하여 개수를 출력한다.
    - 5 가 입력되면 임의의 코인을 삭제하고 삭제된 코인의 값을 출력한다.
    - 9 가 입력되면 프로그램이 종료되고 총 코인의 개수, 가장 큰 코인의 값, 가방에 있는 코인의 합을 출력한다.
  - 출력
    - 3 이 입력되면 총 코인의 개수와 가장 큰 코인의 값, 현재 있는 코인의 총 금액을 출력한다.
    - 4 가 입력되면 다시 코인의 값을 입력받아 해당 값을 가진 코인을 검색하여 개수를 출력한다.
    - 5 가 입력되면 임의의 코인을 삭제하고, 삭제된 코인의 값을 출력한다.
    - 9 가 입력되면 프로그램이 종료되고, 총 코인의 개수, 가장 큰 코인의 값, 가방에 있는 코인의 합을 출력한다.
- 자료구조
  - 코인을 담는 가방 역할을 하는 Node
  - Node 가 다른 노드를 연결하며 노드가 추가 될 때마다 크기의 제한없이 계속 증가하는 연결된 체인 형태의 구조(Linked Chain)
  - 메뉴, 메뉴 종료, 프로그램 종료 시에 각각 쓰이는 enum 형태의 자료구조

2 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	run()	없음	없음	동전 가방 프로그램을 실행시키는 메소드
	showMessage(MessageID aMessageID)	에러메세지 혹은 알림	없음	에러메세지 혹은 알림에 따라 메세지를 출력하는 메소드

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 코인값을 입력받는 스캐너 생성
	int inputInt()	없음	입력받은 코인값	사용자로부터 코인값을 입력받아 리턴
	void outputResult(int aTotalCoinSize, int aMaxCoinValue, int aSumOfCoinValue)	총 코인갯수, 제일 큰 코인값, 총 코인값의 합	없음	총 코인의 갯수, 가장 큰 코인의 값 총 코인값의 합을 출력
	void outputMessage(String aMessageString)	출력하고 싶은 문자열	없음	메세지 출력 메소드.
	void outputSearch(int aSearchValue, int aSearchedSize)	찾으려는 코인값 찾으려는 코인값의 갯수	없음	찾으려는 코인의 값과 해당 값을 가진 코인이 몇 개 있는지 출력.

	void outputRemove(int removedCoin)	삭제된 코인값	없음	삭제된 코인을 출력하는 메소드

Class	Coin			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Coin(int givenValue)	코인값	없음	주어지는 값으로 코인의 값을 정하는 생성자 메소드
	int value()	없음	코인값 리턴	코인 값을 반환
	void setValue(int newValue)	저장하고싶은 새로운 코인값	없음	코인값을 새롭게 저장
	boolean equals(Coin aCoin)	코인	입력된 코인의 값과 현재 코인이 값이 같으면 참, 아니면 거짓.	입력받은 코인값과 가방안에 있는 코인들 중 같은 값이 있는지 확인하려할 때 사용.

Class	LinkedBag			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	LinkedBag()	없음	없음	가방을 초기화 시키는 생성자 메소드
	int size()	없음	가방 내에 있는 노드의 갯수 리턴	가방 안에 있는 노드의 갯수를 리턴
	boolean isEmpty()	없음	가방이 비어있으면 참, 아니면 거짓	가방에 노드가 있는지 없는지 검사
	boolean doesContain(Coin anCoin)	코인	가방 안에 있는 노드들 중에 코인 값을 가진 노드가 있으면 참, 없으면 거짓	가방 안에 들어있는 노드들 중 특정 코인 값을 가진 노드가 있는지 검사
	int frequencyOf(Coin anCoin)	코인	특정 값을 가진 코인의 갯수	가방 안의 노드에 특정 값을 가진 코인이 있는지 확인하여 개수를 체크
	int maxElementValue()	없음	가방 안에 들어있는 노드들의 코인값 중 최대값	가방 안에 모든 노드의 코인값들을 조사해서 제일 큰 코인 값을 리턴
	int sumElementValues()	없음	가방 안에 들어있는 노드들의 코인 값의 합	가방 안의 모든 노드의 코인의 값을 더해서 리턴
	boolean add(Coin anCoin)	코인	생성되었음을 알리는 참값	주어진 코인값을 가지는 노드를 새롭게 생성한 후, 체인형태의 자료구조에서 맨 앞 공간에 노드를 채워 넣음. 가방의 맨 앞 쪽.
	boolean remove(Coin anCoin)	코인	주어진 코인을 가방의 노드 안에서 못찾으면 null, 찾으면 삭제된 코인값	가방 안에 주어진 코인값을 가진 노드를 검색한다. 더이상 노드가 없을 때까지 반복하여 검색한다. 만일 값을 가진 노드를 찾았을 때, 해당 노드가 맨 앞의 노드라면, 그것을 제거하고 삭제된 노드의 다음 노드를 앞으로 끌어온다. 맨 앞의 노드가 아니라면 현재 노드를 삭제하고 이전

				노드의 다음 노드를 현재 노드의 다음노드로 설정한다.
	Coin removeAny()	없음	삭제된 코인	코인을 아무거나 삭제한다. 굳이 아무거나 삭제할 것이라면 맨 앞에 있는 노드를 삭제한다.
	void clear()	없음	없음	가방 안의 모든 코인을 비운다.

Class	Node			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Node()	없음	없음	노드를 초기화 시키는 생성자 메소드
	Node(Coin givenCoin)	저장하려는 코인	없음	주어진 코인값을 가지는 노드를 생성하는 생성자 메소드
	Node(Coin givenCoin, Node givenNext)	저장하려는 코인, 다음 노드	없음	주어진 코인을 가지고, 다음 노드를 알고 있는 노드를 생성하는 생성자 메소드
	Coin coin()	없음	코인	노드에 담긴 코인 리턴
	Node next()	없음	노드	현재 노드의 다음 노드값을 리턴
	void setCoin()	없음	없음	코인값을 새롭게 설정
	void setNext()	없음	없음	다음 노드값을 새롭게 설정

3 종합 설명서

- 프로그램 시작 후 가방에 들어갈 총 코인의 개수를 설정한다.
- 프로그램이 종료될 때까지 수행하고자하는 메뉴를 입력한다.
- 1 번을 입력하면 가방에 넣고자 하는 코인 값을 입력한다.
  - i 코인 값을 입력하게 되면 해당 코인을 담는 노드가 생성되며, 새로운 노드는 _head 로 설정된다. 원래 있던 노드값은 새로운 노드의 다음 노드값으로 설정되어 뒤로 밀리게 된다.
- 2 번을 입력하면 가방에서 지우고자하는 코인 값을 입력한다.
  - i 코인 값을 입력하면 해당 코인 값을 가진 노드를 찾기 위해 검색을 한다.
  - ii 만일 찾는다면 해당 코인 값을 가진 노드의 이전 노드와, 코인 값을 가진 노드의 다음 노드를 연결하여 코인 값을 가진 노드를 삭제한다.
- 3 번을 입력하면 현재 가방 안에 있는 코인의 총 개수, 가장 값이 큰 코인, 총 코인 값의 합을 출력한다.
  - i 현재 연결되어 있는 노드들을 검색하여 총 개수, 가장 값이 큰 코인, 총 코인의 합을 출력한다.
- 4 번을 입력하면 찾고자 하는 코인의 값을 입력하고, 해당 값을 가진 코인의 개수를 출력한다.
  - i 찾고자 하는 코인 값을 가진 노드를 찾고, 찾게 되면 카운터를 증가시켜

개수를 출력한다.

- 5 번을 입력하면 임의의 코인을 삭제한다.
  - i 임의의 코인을 삭제한다. 이 때 임의의 코인은 제일 앞의 노드를 삭제한다.
- 9 번을 입력하면 가방 안에 있는 코인의 총 개수, 가장 값이 큰 코인, 총 코인 값의 합을 출력 한 후 프로그램을 종료한다.

2 프로그램 장단점 분석

- 장점
  - 클래스를 이용하여 LinkedBag 이라는 자료구조 형태를 구현해볼 수 있다.
  - 자료의 개수를 제한하지 않고 계속해서 자료를 생성해서 저장할 수 있다.
  - Bag 에서 내가 원하는 값을 찾아서 삭제, 조회를 해볼 수 있다.
  - 입력되는 코인 값은 맨 앞에서 뒤로 밀리는 LinkedChain 자료구조의 특성상 순서대로 LinkedBag 에 들어간다.
  - 메뉴에 표시된 번호 이외의 번호를 입력했을 때 오류 메세지를 출력한다.
  - 삭제된 노드는 따로 처리하지 않아도 자바의 Garbage Collection 에 의해서 자동적으로 삭제된다.
- 단점
  - 가방 내부에 아이템을 검색할 때 x 번째에 어떤 아이템이 들어 있는지 알 수 없다
  - 가방 내부에 특정 아이템이 몇 번째 순서에 들어있는지 알 수 없다.
  - 아이템을 중간에서부터 넣을 수는 없다.

2 실행 결과 분석

1 입력과 출력

입력 - 가방에 코인을 넣고자 할 때

```
<<동전 가방 프로그램을 시작합니다>>
수행하려고하는 메뉴를 선택하세요
(add: 1, remove: 2, print: 3, search: 4, removeAny: 5, exit: 9): 1
코인의 액수를 입력하세요: 5
```

입력 - 가방의 특정 코인 값을 삭제하고자 할 때

```
수행하려고하는 메뉴를 선택하세요
(add: 1, remove: 2, print: 3, search: 4, removeAny: 5, exit: 9): 2
코인의 액수를 입력하세요: 5
```

입력 - 가방 내부 총 코인, 가장 큰 코인, 코인의 합을 알고자 할 때

```
수행하려고하는 메뉴를 선택하세요
(add: 1, remove: 2, print: 3, search: 4, removeAny: 5, exit: 9): 3
총 코인 : 5
가장 큰 코인 : 30
코인의 합 : 70
```

입력 - 가방에서 특정 값을 가진 코인이 몇 개 있는지 알고자 할 때

```
수행하려고하는 메뉴를 선택하세요
(add: 1, remove: 2, print: 3, search: 4, removeAny: 5, exit: 9): 4
코인의 액수를 입력하세요: 5
5코인은 2개 존재합니다.
```

입력 - 가방 내부에서 임의의 코인 삭제

```
수행하려고하는 메뉴를 선택하세요
(add: 1, remove: 2, print: 3, search: 4, removeAny: 5, exit: 9): 5
삭제된 코인 : 30
```

입력 - 프로그램을 종료하고자 할 때

```
수행하려고하는 메뉴를 선택하세요
(add: 1, remove: 2, print: 3, search: 4, removeAny: 5, exit: 9): 9
9가 입력되어 종료합니다.
총 코인 : 4
가장 큰 코인 : 20
코인의 합 : 40
<<동전 가방 프로그램을 종료합니다>>
```

2 결과 분석

- 1 번을 입력 했을 때 : 가방 안에 특정 값의 코인을 넣음
- 2 번을 입력 했을 때 : 가방 안에 특정 값의 코인을 삭제
- 3 번을 입력 했을 때 : 가방 안에 들어 있는 코인의 총 개수, 가장 큰 코인의 값, 모든 코인 값의 합을 출력
- 4 번을 입력 했을 때 : 가방 안에 들어 있는 특정 값을 가진 코인을 입력하고, 그 값을 가진 코인이 몇 개가 들어 있는지 출력.
- 5 번을 입력 했을 때 : 가방 안에 들어 있는 임의의 코인을 삭제한다. 이 때 삭제되는 코인 값은 가방 안에 들어가는 맨 앞의 값(마지막 값)이 삭제된다. 위 스크린샷에서 마지막에 들어간 코인 값이 30 이므로 30 코인이 삭제된 것을 확인할 수 있다.
- 9 번을 입력 했을 때 : 가방 안에 들어 있는 코인의 총 개수, 가장 큰 코인의 값, 모든 코인의 값의 합을 출력하고 프로그램을 종료한다.
  ⇒ 프로그램이 정상적으로 작동되는 것을 확인할 수 있다.

3 소스 코드

Class	AppController

```
public class AppController {
	private AppView _appView;
	private LinkedBag _coinCollector; //코인 컨트롤러

	public AppController() {
		this._appView = new AppView();
	}



	public void run() {
		// TODO Auto-generated method stub


		int input = 0; //넣으려는 코인
		int order = 0; //메뉴 번호

		this.showMessage(MessageID.Notice_StartProgram); //프로그램 시작
		this._coinCollector = new LinkedBag();

		while (order != 9) { //메뉴 번호가 9 가 아니라면 계속 실행
			this.showMessage(MessageID.Notice_Menu); //메뉴 출력
			order = this._appView.inputInt(); //수행하려고 하는 메뉴 입력받기

			if(order == 1) { //1 번메뉴
				this.showMessage(MessageID.Notice_InputCoin);  //넣으려는 코인의
액수
				input = this._appView.inputInt(); //액수 입력
				Coin anCoin = new Coin(input); //코인값을 저장하는 코인객체 생성
				this._coinCollector.add(anCoin); //코인값을 가방에 저장
			}
			else if (order == 2) { //2 번메뉴
				this.showMessage(MessageID.Notice_InputCoin); //코인액수 입력
				input = this._appView.inputInt();
				Coin givenCoin = new Coin(input); //입력받은 코인액수를 이용하여
코인 객체 생성
				this._coinCollector.remove(givenCoin); //입력받은 코인액수를 삭제
			} else if (order == 3) { //3 번메뉴
				this._appView.outputResult(this._coinCollector.size(),
this._coinCollector.maxElementValue(), this._coinCollector.sumElementValues());
				//가방에 들어있는 코인의 갯수 , 가장 큰 코인값, 모든 코인의 합
			} else if (order == 4) { //4 번 메뉴
				this.showMessage(MessageID.Notice_InputCoin); //코인값 입력
				input = this._appView.inputInt();
				Coin givenCoin = new Coin(input);
				this._appView.outputSearch(input,
this._coinCollector.frequencyOf(givenCoin));
				//입력받은 코인값이 몇개가 존재하는지 검색
			} else if (order == 5){

				this._appView.outputRemove(_coinCollector.removeAny().value());
```

```
                                        //LinkedBag 의 removeAny()를 실행하여 리턴받는 코인의 값을 삭제된
값으로 출력.


                                } else if (order == 9) { //9 번메뉴
                                        this.showMessage(MessageID.Notice_EndMenu); //메뉴를 종료
                                        this._appView.outputResult(this._coinCollector.size(),
this._coinCollector.maxElementValue(), this._coinCollector.sumElementValues());
                                        //결과출력. 총 코인갯수, 가장 큰 코인, 코인들의 합.
                                        this.showMessage(MessageID.Notice_EndProgram);  //프로그램 종료
메세지 출력
                                        System.exit(0); //프로그램 종료
                                } else {
                                        this.showMessage(MessageID.Error_WrongMenu);    //메뉴의    값이
아닌 값을 입력하면 오류메세지 출력
                                }


                        }

                }
                private void showMessage(MessageID aMessageID) { //메세지 출력메소드
                        // TODO Auto-generated method stub
                        switch(aMessageID) {
                        case Notice_StartProgram : //프로그램 시작시
                                this._appView.outputMessage("<<동전 가방 프로그램을 시작합니다>>" + "₩n");
                                break;

                        case Notice_Menu : // 메뉴알림 및 선택
                                this._appView.outputMessage("수행하려고하는 메뉴를 선택하세요" + "₩n");
                                this._appView.outputMessage("(add:  1,  remove:  2,  print:  3,  search:  4,
removeAny: 5, exit: 9): ");
                                break;
                        case Notice_InputCoin: // 코인의 액수
                                this._appView.outputMessage("코인의 액수를 입력하세요: ");
                                break;
                        case Notice_EndMenu: // 종료합니다
                                this._appView.outputMessage("9 가 입력되어 종료합니다." + "₩n");
                                break;
                        case Notice_EndProgram: // 프로그램 종료
                                this._appView.outputMessage("<<동전 가방 프로그램을 종료합니다>>₩n");
                                break;
                        case Error_WrongMenu: //오류메세지
                                this._appView.outputMessage("<<ERROR : 잘못된 메뉴입니다.>>₩n");
                        default:
                                break;

                        }




                }

}
```

Class	AppView

```
import java.util.Scanner;

public class AppView {

	private Scanner _scanner;

	public AppView() {
		this._scanner = new Scanner(System.in);
	}

	public int inputInt() {

		return _scanner.nextInt(); //코인을 입력받음.
	}

	public void outputResult(int aTotalCoinSize, int aMaxCoinValue, int aSumOfCoinValue) {
		System.out.println("총 코인 : " +aTotalCoinSize); // 총 코인 출력
		System.out.println("가장 큰 코인 : " +aMaxCoinValue); //가장 큰 코인
		System.out.println("코인의 합 : " + aSumOfCoinValue); //총 코인의 합

	}

	public void outputMessage(String aMessageString) {
		System.out.print(aMessageString); //메세지 출력

	}
	public void outputSearch(int aSearchValue, int aSearchedSize) {
		System.out.println(aSearchValue+"코인은 " + aSearchedSize + "개 존재합니다."); //검색하고
싶은 코인이 몇개 있는지

	}
	public void outputRemove(int removedCoin) {
		System.out.println("삭제된 코인 : " +removedCoin);
	}

}
```

Class	LinkedBag

```
public class LinkedBag {

	private int _size; //여기서 사이즈는 총 노드의 갯수
	private Node _head;

	public LinkedBag() {
		this._head = null; //초기화실행
		this._size = 0;

	}
	public int size() {
```

```
                return this._size; //노드갯수 리턴.
        }
        public boolean isEmpty() {
                return (this._size == 0); //비었는지 안비었는지 체크
        }
        public boolean doesContain(Coin anCoin) {
                boolean found = false;

                Node searchNode = this._head; //제일 앞부터 검색해나가기
                while(searchNode != null && !found) { //검색할 노드가 없고, 못찾을때까지 검색
                        if(searchNode.coin().equals(anCoin)){ //노드의 코인값이 입력된 코인값과 같다면
                                found = true; //빙고
                        }
                        searchNode = searchNode.next(); //아니라면 다음 노드로 이동
                }
                return found;
        }
        public int frequencyOf(Coin anCoin) {
                Node searchNode = this._head; //처음 노드부터 검색
                int count = 0; //카운터 증가
                while(searchNode != null) { //찾을 노드가 없을때까지 검색
                if (searchNode.coin().equals(anCoin)){ //현재 찾고있는 노드의 코인값과 입력된 코인값이
같다면
                        count++; //갯수 증가
                }
                searchNode = searchNode.next(); //다음 노드로 이동
                }
                return count;

        }
        public int maxElementValue() { //최대값 찾기
                Node searchNode = this._head; //현재 노드는 맨 앞부터
                int maxValue = 0; //최대값을 저장할 변수

                while(searchNode != null) { //검색할 노드가 없을때까지. 노드가 존재하는한 계속검색
                if (maxValue < searchNode.coin().value()) { //최대값과 현재 찾고있는 노드의 코인값과
비교하여 노드값이 크면
                        maxValue = searchNode.coin().value(); //맥스값을 현재 코인값으로 교체
                }
                searchNode = searchNode.next(); //다음 노드로 이동
                }
                return maxValue;
        }
        public int sumElementValues() {
                Node searchNode = this._head; //현재 노드 맨 앞부터
                int sumValue = 0; //값을 저장할 노드
                while(searchNode != null) { //노드가 존재하는한 계속

                sumValue += searchNode.coin().value(); //찾고있는 코인의 값을 계속 더해서 저장

                searchNode = searchNode.next(); //다음값으로 이동
                }
                return sumValue;
        }
        public boolean add(Coin anCoin) { //코인 저장하기

                Node newNode = new Node(anCoin);
                //주어진 코인값의 새로운 노드를 생성
```

```
		newNode.setNext(this._head); // 현재 노드의 다음값을 현재 헤드로 지정. 지금 헤드노드는 뒤로 밀려남.
		this._head = newNode; //현재헤드를 지금 헤드로 새로 지정. 노드를 추가할때는 맨 앞에서 추가하므로 원래 있던 헤드는 뒤로 밀려나가는 형식
		this._size++;
		return true;
	}
	public Coin remove(Coin anCoin) {
		if(this.isEmpty()){
			return null;
		} else {
			Node previousNode = null; //이전노드
			Node currentNode = this._head; //현재노드는 머리부터
			boolean found = false;

			/*입력된 코인값을 가진 노드를 찾기*/
			while(currentNode != null && !found) { //지우고싶은 코인값의 노드 찾기 현재노드가 존재하고있으면서 못찾을때까지 계속
				if(currentNode.coin().equals(anCoin)) { //입력한 값의 코인값과 현재 코인값이 같다면
					found = true; //빙고
				} else {
					previousNode = currentNode; //아니라면 현재값을 이전값으로 하고
					currentNode = currentNode.next(); //현재값을 현재의 다음 값으로하여 다음 노드로 진행
				}
			}

			/*해당 노드를 찾아 반복문을 탈출했다면 노드를 삭제하는 방법*/
			if(!found) {
				return null; //패스
			} else {
				if(currentNode == this._head) { //찾았을 때, 현재 노드가 헤드라면
					this._head = this._head.next(); //헤드를 헤드노드의 다음 노드로 설정
				} else {
					previousNode.setNext(currentNode.next());	//아니라면 이전노드의 다음 노드를 현재 노드의 다음 노드로 설정
					//이전노드와 다음노드의 사이에 있는 현재 노드를 삭제
				}
				this._size--; //여기서 없어진 노드는 garbage collection 으로 처리.
				return anCoin;
			}
		}

	}
	public Coin removeAny () {
		if (this.isEmpty()) {
			return null ;
		} else {
			Coin removedCoin = this._head.coin(); //삭제될 코인을 저장. 삭제되는 코인값이 무엇인지 확인해야하므로
			this._head = this._head.next();
			this._size--;
			return removedCoin; //아무거나 없애는 것이면 맨 앞에 있는 노드를 없애는 것.
		}
	}
```

```
	public void clear() {
		this._size = 0;
		this._head = null; //초기화. 헤드도 없앰.

	}
}
```

Class	Coin

```
public class Coin {
	 /*코인 클래스*/
	private int _value;

	public Coin() {
		//기본생성자
	}
	public Coin(int givenValue) {
		this._value   = givenValue; //코인값을 저장
	}

	public int value() {
		return this._value; //코인값 리턴
	}


	public void setValue(int newValue) {
		this._value = newValue; //코인값을 새로저장
	}

	public boolean equals(Coin aCoin) {
		return (this._value == aCoin.value());
	}

}
```

Class	Node

```
public class Node {

	private Coin _coin; //노드에 담기는 코인
	private Node _next; //다음 노드


	public Node(){ //노드 초기화
		this._coin = null;
		this._next = null;
	}

	public Node(Coin givenCoin) { //노드의 코인값을 설정
		this._coin = givenCoin;
		this._next = null;
	}
```

```
	public Node(Coin givenCoin, Node givenNext) { //노드의 코인과 다음노드값을 설정
		this._coin = givenCoin;
		this._next = givenNext;
	}

	public Coin coin() {


		return this._coin; //코인값 리턴
	}

	public Node next() {
		return this._next; //다음노드값 리턴

	}
	public void setCoin (Coin newCoin) {
		this._coin = newCoin; //코인값 새로 설정

	}
	public void setNext (Node newNext) {
		this._next = newNext; //다음 노드값 새로 설정
	}


}
```