자료구조 보고서

[제 02주]

2018.3.25.

201502273/김현종

1. 코드

```
public class LinkedBag {

	private int maxSize;
	private int curSize;
	private int totalValue;
	private Node<Coin> next;


	/*
	 * LinkedBag의 생성자
	 * 인수인 maxSize를 LinkedBag의 최대 크기로 정한다.
	 * init() 메소드를 통해 필요한 값들을 초기화한다.
	 */
	public LinkedBag(int maxSize) {
		this.maxSize = maxSize;
		init();
	}

	/*
	 * 현재 담긴 코인의 개수와 총 가격, next포인터를 초기화한다.
	 */
	public void init() {
		this.curSize = 0;
		this.totalValue = 0;
		this.next = null;
	}

	/*
	 * 현재 사이즈와 최대 사이즈를 비교하여 둘이 같으면 꽉 찼다고 알려준다.
	 */
	public boolean isFull() {
		return (this.maxSize == this.curSize);
	}

	/*
	 * 현재 사이즈가 0이면 LinkedBag이 비었다고 알려준다.
	 */
	public boolean isEmpty() {
		return (this.curSize == 0);
	}

	/*
	 * 특정 위치에 value라는 값을 갖는 코인을 추가하는 함수이다.
	 * 먼저 꽉 찼는지 확인하고 꽉 차지 않았으면 현재 넣으려고 하는 인덱스가 음수이거나
혹은 현재 들어있는 개수보다 뒤인지 확인한다.
	 * 둘 다 아닐경우 이전 노드와 현재 노드, 두 개의 노드를 가지고 리스트를 탐색하며
index가 가리키는 자리까지 이동한다.
	 * 다 이동하면 새로 넣을 값인 newNode를 리스트 사이에 끼어 넣는다.
	 * prev.setNext(newNode);
	 * if(curr != null)
	 *		newNode.setNext(curr);
	 * 현재 사이즈를 1 올리고 총 가격을 value만큼 올린다.
	 */
	public boolean addAt(int value, int index) {
		if(this.isFull()) {
			System.out.print("\n실패\n");
			return false;
		}else if(index > this.curSize || index < 0){
			System.out.print("\n실패\n");
			return false;
		} else {
			Node<Coin> prev = null;
			Node<Coin> curr = this.next;
			Node<Coin> newNode = new Node<Coin>(new Coin(value));
			for(int i = 0; i < index; i++) {
				prev = curr;
				curr = curr.getNext();
			}
			if(prev == null) {
```

```
                                this.next = newNode;
                        }else {
                                prev.setNext(newNode);
                                if(curr != null)
                                        newNode.setNext(curr);
                        }
                        this.curSize++;
                        this.totalValue += value;

                        return true;
                }
        }

        /*
         * value라는 값을 갖는 새로운 코인을 넣는 함수이다.
         * 먼저 LinkedBag이 꽉 찼는지 확인해보고
         * 꽉 차지 않았으면 addAt함수를 통해 현재 맨 뒷자리에 코인을 추가한다.
         */
        public boolean add(int value) {
                if(this.isFull()) {
                        System.out.print("\n실패\n");
                }else {
                        if(this.addAt(value, this.curSize))
                                return true;
                }
                return false;
        }

        /*
         * 현재 사이즈를 반환해주는 함수이다.
         */
        public int getSize() {
                return this.curSize;
        }

        /*
         * 총 가격을 반환해주는 함수이다.
         */
        public int getTotalValue() {
                return this.totalValue;
        }

        /*
         * value라는 값을 갖는 코인을 리스트에서 제거해주는 함수이다.
         * pre와 curr 두 개의 노드를 가지고 리스트를 탐색하고 현재 노드가 가지고 있는
value가 찾던것과 같으면 이전 노드에 next에 현재 노드의 next를 연결해준다.
         * 그리고 현재 사이즈를 1 줄이고 총 가격도 value만큼 줄인다.
         */
        public  boolean remove(int value) {
                if(!this.isEmpty()) {
                        Node<Coin> pre = null;
                        Node<Coin> curr = this.next;
                        while(curr != null) {
                                if(curr.getValue().getValue() == value) {
                                        pre.setNext(curr.getNext());
                                        this.curSize--;
                                        this.totalValue -= value;
                                        return true;
                                }
                                pre = curr;
                                curr = curr.getNext();
                        }
                        return false;
                }
                System.out.print("\nFull\n");
                return false;
        }

        /*
         * 모든 코인을 삭제하는 함수이다.
         * 먼저 코인을 담을 coins라는 Coin 어레이를 만들고 현재 들어있는 사이즈만큼의
크기를 준다.
```

```
	 * temp라는 노드로 리스트 전체를 탐색하면서 나오는 코인들을 coins에 순서대로 저장한다.
	 * 그리고 작업이 모두 끝나면 리스트를 다시 초기화하기 위해 init() 메소드를 호출한다.
	 */
	public Coin[] removeAll() {
		Coin[] coins = new Coin[this.curSize];
		Node<Coin> temp = this.next;

		for(int i = 0; i < this.curSize; i++) {
			coins[i] = temp.getValue();
			temp = temp.getNext();
		}

		init();

		return coins;
	}

	/*
	 * 리스트에 있는 가장 큰 값을 삭제한다.
	 */
	public boolean removeMax() {
		return this.remove(this.max());
	}

	/*
	 * 입력된 값의 코인이 LinkedBag에 몇개나 들어있는지를 알려주는 메소드이다.
	 * frequency라는 변수를 만들고 temp라는 노드를 통해 리스트 전체를 탐색하며 value와 값이 같은 코인이 있을때마다 frequency를 1 올린다.
	 * 그리고 탐색이 끝나면 frequency를 반환한다.
	 */
	public int frequentCoin(int value) {
		int frequency = 0;
		Node<Coin> temp = this.next;
		while(temp != null) {
			if(temp.getValue().getValue() == value) {
				frequency++;
			}
		}
		return frequency;
	}

	/*
	 * temp노드를 사용해 리스트 전체를 탐색하다가 value와 같은 값이 나오면 true를 return한다.
	 * 찾지 못하면 false를 return한다.
	 */
	public boolean doesContain(int value) {
		Node<Coin> temp = this.next;
		while(temp != null) {
			if(temp.getValue().getValue() == value) {
				return true;
			}
		}
		return false;
	}

	/*
	 * next라는 노드가 null이 아니면 해당 노드의 print() 메소드를 호출한다.
	 * null이면 EMPTY를 출력한다.
	 */
	public void print() {
		if(this.next != null)
			this.next.print();
		else
			System.out.println("EMPTY");
	}

	/*
	 * 현재 가방에 들었는 코인들 중 가장 큰 값을 반환한다.
	 * temp라는 노드를 통해 리스트를 탐색하고 현재값(value)보다 더 큰값이 나올때마다 value값을 그 값으로 갱신한다.
```

```
	 * 탐색이 다 끝나면 value값을 반환한다.
	 */
	public int max() {
		int value = -1;
		Node<Coin> temp = this.next;
		while(temp != null) {
			if(temp.getValue().getValue() > value) {
				value = temp.getValue().getValue();
			}
			temp = temp.getNext();
		}
		return value;
	}

	/*
	 * 가방의 크기를 재 조정하는 함수이다.
	 * 새로 정하는 크기를 인수로 받아오며 새로 정하는 사이즈가 현재 사이즈보다
클경우에는 그냥 최댓값을 키운다.
	 * 그렇지 못할경우에는 새로 정하는 사이즈에 있는 노드까지 찾아가 리스트를
끊어버린다.
	 * 그리고 총 가격을 다시 계산하기 위해 totalValue를 0으로 초기화하고 리스트 전체를
탐색해서 모든 코인의 값을 totalValue에 더한다.
	 */
	public boolean resize(int size) {

		if(this.curSize > size) {
			Node<Coin> temp = this.next;
			for(int i = 1; i < size; i++) {
				temp = temp.getNext();
			}
			temp.setNext(null);
		}

		this.curSize = size;
		this.totalValue = 0;

		Node<Coin> temp = this.next;
		while(temp != null) {
			this.totalValue += temp.getValue().getValue();
			temp = temp.getNext();
		}

		return false;
	}

	/*
	 * 현재 가방에 들어있는 코인의 총 값을 반환한다.
	 */
	private int sum() {
		return this.totalValue;
	}

	/*
	 * 노드 클래스이다.
	 */
	private class Node<T> {

		private Node<T> next; //해당 노드 다음으로 오는 노드이다. 없을경우 null;
		private T value; //해당 노드가 가지고 있는 value이다.

		/*
		 * value의 setter이다.
		 */
		public void setValue(T value) {
			this.value = value;
		}

		/*
		 * 기본 생성자로 다음 노드와 값 모두 null로 초기화한다.
		 */
		public Node() {
			this.next = null;
```

```
                        this.value = null;
                }

                /*
                 * 생성자이다. 받은 인수로 노드의 값을 초기화한다.
                 */
                public Node(T value) {
                        this.next = null;
                        this.value = value;
                }

                /*
                 * 노드의 값을 반환하는 메소드이다.
                 */
                public T getValue() {
                        return this.value;
                }

                /*
                 * 해당 노드 다음에 오는 노드를 설정하는 메소드이다.
                 * 다음에 오는 노드가 null일경우 null을 반환하고 그렇지 않으면 값을
반환한다.
                 */
                public T setNext(Node<T> node) {

                        this.next = node;
                        if(node == null) {
                                return null;
                        }
                        return node.getValue();
                }

                /*
                 * 해당 노드 다음에 오는 노드를 반환하는 메소드이다.
                 */
                public Node<T> getNext() {
                        return next;
                }

                /*
                 * 노드의 관계를 보여주는 메소드이다. 다음 노드가 null이 아니면 value의
값을 출력해준 뒤 다음 노드의 print() 메소드를 호출한다.
                 * 다음 노드가 null이면 현재 노드의 값을 출력한다.
                 */
                public void print() {
                        if( this.next != null) {
                                if(this.value.getClass() == Coin.class) {
                                        System.out.print( ((Coin)this.value).getValue() +
" -> ");
                                        next.print();
                                }
                        } else {
                                if(this.value.getClass() == Coin.class) {
                                        System.out.println(
((Coin)this.value).getValue());
                                }
                        }
                }


        }
}
```

LinkedBag.java

```java
public class Coin {

	private static final int DEFAULT_VALUE = 0;
	private int value;

	/*
	 * 코인의 기본 생성자
	 * 코인의 값을 기본값으로 설정한다.
	 */
	public Coin() {
		this.value = Coin.DEFAULT_VALUE;
	}

	/*
	 * 코인의 생성자
	 * 코인의 값을 입력된 값으로 설정한다.
	 */
	public Coin(int value) {
		this.value = value;
	}

	/*
	 * 코인의 값을 반환해주는 메소드
	 */
	public int getValue() {
		return this.value;
	}
}
```

Coin.java

```java
import java.util.Scanner;


public class MainClass_02_201502273 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub


		Scanner scanner = new Scanner(System.in);
		int menu = 0, bagSize;
		System.out.println("<< 프로그램을 시작합니다 >>");
		System.out.print("+ 가방에 들어갈 총 코인의 개수를 입력하시오 : ");
		bagSize = scanner.nextInt();
		while(bagSize < 0) {
			System.out.print("<< 가방의 크기가 음수가 될 수 없습니다 >>");
			bagSize = scanner.nextInt();
		}
		LinkedBag bag = new LinkedBag(bagSize);

		while(menu != 9) {


			System.out.println("+ 메뉴를 선택하세요 \t\t 1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit");
			menu = scanner.nextInt();

			int value;

			switch(menu) {
			case 1:
				System.out.print("코인의 액수를 입력하세요 : ");
				value = scanner.nextInt();
				if(bag.add(value)) {
					System.out.println(value + "코인을 넣었습니다.");
				}else {
					System.out.println(value + "코인을 넣을 수 없습니다.");
				}
```

```
                                        break;
                                case 2:
                                        System.out.print("코인의 액수를 입력하세요 : ");
                                        value = scanner.nextInt();
                                        if(bag.remove(value)) {
                                                System.out.println(value + "코인이
제거되었습니다.");
                                        }else {
                                                System.out.println(value + "코인이 없습니다.");
                                        }
                                        break;
                                case 3:
                                        bag.print();
                                        break;
                                case 4:
                                        System.out.print("코인의 액수를 입력하세요 : ");
                                        value = scanner.nextInt();
                                        System.out.println(value + "코인은 " +
bag.frequentCoin(value) + "개 존재합니다.");
                                        break;
                                case 5:
                                        Coin[] coins = bag.removeAll();
                                        System.out.print("[ ");
                                        for (Coin coin : coins) {
                                                System.out.print(coin.getValue() + " ");
                                        }
                                        System.out.println("]");
                                        System.out.println("코인들이 제거되었습니다.");
                                        break;
                                case 6:
                                        System.out.print("코인의 액수를 입력하세요 : ");
                                        value = scanner.nextInt();
                                        System.out.print("넣을 인덱스를 입력하세요 : ");
                                        if(bag.addAt(value, scanner.nextInt())) {
                                                System.out.println(value + "코인을
넣었습니다.");
                                        }else {
                                                System.out.println(value + "코인을 넣을 수
없습니다.");
                                        }
                                        break;
                                case 7:
                                        System.out.print("가장 큰 값을 제거합니다 : ");
                                        System.out.println(bag.removeMax());
                                        break;
                                case 8:
                                        System.out.print("크기를 입력해주세요");
                                        bag.resize(scanner.nextInt());
                                        System.out.println("크기를 " + bag.getSize() + "로
변경합니다.");
                                        break;
                                case 9:
                                        System.out.print("<9가 입력되어 종료합니다>\n");
                                        bag.print();
                                        System.out.print("<<프로그램을 종료합니다>>");
                                        break;

                                default:

                                }


                        }


                }

}
```

MainClass_02_201502273

2. 결과

```
<< 프로그램을 시작합니다 >>
+ 가방에 들어갈 총 코인의 개수를 입력하시오 : 20
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
1
코인의 액수를 입력하세요 : 1
1코인을 넣었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
1
코인의 액수를 입력하세요 : 2
2코인을 넣었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
1
코인의 액수를 입력하세요 : 3
3코인을 넣었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
1
코인의 액수를 입력하세요 : 4
4코인을 넣었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
3
1 -> 2 -> 3 -> 4
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
2
코인의 액수를 입력하세요 : 2
2코인이 제거되었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
3
1 -> 3 -> 4
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
6
코인의 액수를 입력하세요 : 7
넣을 인덱스를 입력하세요 : 1
7코인을 넣었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
3
1 -> 7 -> 3 -> 4
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
7
가장 큰 값을 제거합니다 : true
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
3
1 -> 3 -> 4
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
8
크기를 입력해주세요2
크기를 2로 변경합니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
3
1 -> 3
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
5
[ 1 3 ]
코인들이 제거되었습니다.
+ 메뉴를 선택하세요            1:add | 2:remove | 3:print | 4:search | 5:removeAll | 6:add index | 7:removeMax | 8:resize | 9:exit
9
<9가 입력되어 종료합니다>
EMPTY
<<프로그램을 종료합니다>>
```

처음에 가방에 들어갈 총 코인 개수를 입력하고 메뉴를 선택한다. 먼저 각각 1, 2, 3, 4의 값을 갖는 코인을 가방에 넣었다. 그리고 print를 시키자 1 -> 2 -> 3 -> 4 라는 글자가 나왔고 이는 가방 내부의 LinkedList의 형태를 보여주는 것이다. 그리고 2의 값을 갖는 코인을 제거하고 다시 print를 시키자 2가 제거된 1 -> 3 -> 4가 출력되었다. 그리고 다음으로 7의 값을 갖는 코인을 1자리에 삽입하고 출력하자 1 -> 7 -> 3 -> 4가 출력되었고 최댓값을 제거하니 다시 1 -> 3 -> 4가 출력되는 것을 볼 수 있다. 그리고 현재 세 개의 코인이 들어있는 가방의 크기를 2로 줄이자 1 -> 3이라고 나오는 것을 볼 수 있고 모두 제거를 실행하자 [1 3]이 제거되는 것을 볼 수 있다.

깨달은 점 및 결론

노드를 템플릿을 사용하여 만들었는데 이 때문에 print함수를 구현할 때 까다로운 점이 있었으며 별로 좋아 보이지 않는 방식으로 해결을 했다. 또한 resize 메소드의 경우 사이즈에 잘려 뒤쪽 요소들이 잘릴 때 총 가격을 다시 계산하기 위해 모든 요소를 다시 접근하여

계산하는데 지워지는 양이 남는 양보다 많을 경우에는 하나하나 지우며 총 값을 계산하는 것이 더 효율적일 것으로 생각된다.