자료구조 실습 보고서

[제04주] 동전가방(LinkedBag)

2021년 3월 29일

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.

➔ 동전가방(LinkedBag) 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

- LinkedBag<E> : Generic Type으로 정의하여 가방의 구조와 무관하게 가방에 넣을 원소는 필요에 따라 결정되도록 한다. 동전 가방의 기능을 구성한다.
- Coin : 가방에 넣을 '동전'을 구체화하는 클래스. 동전의 변수, 속성으로 이루어져 있다.
- LinkedNode : 연결 노드의 멤버 변수를 구성한다. 노드를 구체화한다.

View :

- AppView : 프로그램의 입/출력을 담당한다.

Controller :

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

➔ Generic Class?

- Class "LinkedList" 를 정의한다고 하면 설계할 때 미리 type을 결정하는 것이 아니라 LinkedList를 사용하는 시점에 결정하는 것.
- 만약 정의하는 시점에 자료형을 고정해야 한다면 원소 class 마다 별도로 만들어 사용해야 하므로 Generic class는 코드의 생산성을 높인다.

(2) 함수 설명서

➔ 주요 알고리즘

```
while (menuNumber != MENU_END_OF_RUN) {
    switch (menuNumber) {
    case MENU_ADD: // 1을 입력받으면
        this.addCoin(); // addCoin함수 호출
        break;
    case MENU_REMOVE: // 2를 입력받으면
        this.removeCoin(); // removeCoin함수 호출
        break;
    case MENU_SEARCH: // 3을 입력받으면
        this.searchForCoin(); // searchForCoin함수 호출
        break;
    case MENU_FREQUENCY: // 4를 입력받으면
        this.frequencyOfCoin(); // freQuencyOfCoin함수 호출
        break;

    default:
        this.undefinedMenuNumber(menuNumber); // 잘못된 번호를 입력받은 경우
    }
```

사용자로부터 수행할 메뉴 번호를 입력 받는다. 가방에 동전을 넣는 MENU_ADD, 가방에서 해당 동전을 제거하는 MENU_REMOVE, 해당 동전이 있는지 없는지 확인하는 MENU_SEARCH, 해당 동전에 가방에 몇 개 있는지 확인하는 MENU_FREQUENCY로 구성되어 있다. 해당 상수들은 Enum 클래스로 정의해도 되지만 switch문을 실행하는 AppController클래스에 정의하였다.

```
// 정수를 입력받아 Bag에 넣는 함수
private void addCoin() {
    if (this.coinBag().isFull()) { // 만약 Bag이 가득찼다면
        AppView.outputLine("- 동전 가방이 꽉 차서 동전을 넣을 수 없습니다 ."); // 동전을 넣지못한다는 출력문 출력
    } else { // Bag이 비었다면
        AppView.output("? 동전 값을 입력하시오: ");
        int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
        if (this.coinBag().add(new Coin(coinValue))) {
            AppView.outputLine("- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .");
        } else {
            AppView.outputLine("- 주어진 값을 갖는 동전을 가방에 넣는데 실패하였습니다 .");
        }
    }
}
```

'1'을 입력 받으면 호출되는 addCoin 함수이다. 가방이 모두 찼는지 확인하는 isFull 함수 호출 후 비어 있다면 동전 값을 입력 받아 동전 객체 생성 후, 성공적으로 넣었다는 출력문을 출력한다. 이때, 출력문 역시 AppView를 활용한다. 실제 넣는 함수 행위는 ArrayBag 클래스의 add 함수에서 행한다.

```java
// 정수를 입력받아 해당 동전이 있으면 Bag에서 삭제하는 함수
private void removeCoin() {
    AppView.output("? 동전 값을 입력하시오: ");
    int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
    if (!this.coinBag().remove(new Coin(coinValue))) { // 정상적으로 삭제되는지 확인
        // 정상적으로 삭제되지 않는다면
        // 주어진 값의 동전이 Bag에 없으므로 존재하지 않는다는 출력문 출력
        AppView.outputLine("- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다 .");
    } else {
        // 정상적으로 삭제된 경우
        AppView.outputLine("- 주어진 값을 갖는 동전 하나가 가방에서 정상적으로 삭제되었습니다 .");
    }
}
```

'2'를 입력 받으면 호출되는 removeCoin 함수이다. 삭제할 동전 값을 입력 받고 해당 값의 동전 객체 생성 후 ArrayBag 클래스의 remove 함수를 이용하여 삭제가 되는지 확인한다. 정상적으로 삭제되지 않는다면 주어진 값의 동전에 Bag에 없으므로 존재하지 않는다는 출력문을 출력한다. 정상적으로 삭제된 경우 삭제되었다는 출력문을 출력한다.

```java
// 정수를 입력받아 해당 숫자의 동전이 Bag에 있는지 확인하는 함수
private void searchForCoin() {
    AppView.output("? 동전 값을 입력하시오: ");
    int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
    if (this.coinBag().doesContain(new Coin(coinValue))) { // 해당 동전이 Bag에 있는지 확인
        AppView.outputLine("- 주어진 값을 갖는 동전이 가방 안에 존재합니다 .");
    } else {
        AppView.outputLine("- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다 .");
    }
}
```

'3'을 입력 받으면 호출되는 searchForCoin 함수이다. 찾을 동전 값을 입력 받고 해당 동전이 Bag에 있는지 확인한다. 이때 역시 실제 찾는 행위는 ArrayBag 클래스의 doesContain 함수에서 행한다.

```java
// 정수를 입력받아 해당 숫자의 동전이 Bag에 몇개 있는지 확인하는 함수
private void frequencyOfCoin() {
    AppView.output("? 동전 값을 입력하시오: ");
    int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
    int frequency = this.coinBag().frequencyOf(new Coin(coinValue)); // 해당 숫자의 동전이 존재하는 개수의 정수를 입력받는다
    AppView.outputLine("- 주어진 값을 갖는 동전의 개수는 " + frequency + " 개 입니다 .");
}
```

'4'를 입력 받으면 호출되는 frequencyOfCoin 함수이다. 찾을 동전 값을 입력 받고 해당 동전에 Bag에 몇 개 있는지 알아본다. 실제 찾는 행위는 ArrayBag 클래스의 frequencyOf 함수를 이용한다.

```
// 메뉴에 없는 정수를 입력했을 경우 호출하는 함수
private void undefinedMenuNumber(int menuNumber) {
    AppView.outputLine("- 선택된 메뉴 번호 " + menuNumber + " 는 잘못된 번호입니다 .");
}
```

잘못 된 번호를 입력한 경우 호출되는 함수.

```
// 동전의 개수, 가장큰 값, 동전들의 합을 출력하는 함수
private void showStatistics() {
    AppView.outputLine(" 가방에 들어 있는 동전의 개수 : " + this.coinBag().size()); // Bag의 크기 출력
    AppView.outputLine("동전 중 가장 큰 값 : " + this.maxCoinValue()); // Bag 속의 가장 큰 동전 값 출력
    AppView.outputLine("모든 동전 값의 합 : " + this.sumOfCoinValues()); // Bag 속의 동전들의 합 출력
}
```

'9'를 이용하여 프로그램이 종료될 때 출력되는 통계를 나타내는 함수.

⇨ maxCoinValue 함수에서는 Bag 속의 가장 큰 동전 값을 찾기 위해 for문을 Bag의 Size만큼 반복하여 원소 값을 모두 비교하여 max 값을 저장하여 반환한다.

⇨ sumOfCoinValues 함수에서는 마찬가지로 for문을 Bag의 Size만큼 반복하여 원소 값을 모두 더하여 반환한다.

(3) 종합 설명서

➔ 프로그램 실행 순서대로 설명해보자.

```
public class _DS02_Main_201702039_오명주 {
    public static void main(String[] args) {
        AppController appController = new AppController();
        // AppController가 실질적인 main class
        appController.run();
        // 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다
    }
}
```

Main 클래스에서는 AppController 객체 생성 후 run함수만 호출한다.

```
private static final int MENU_ADD = 1;
private static final int MENU_REMOVE = 2;
private static final int MENU_SEARCH = 3;
private static final int MENU_FREQUENCY = 4;
private static final int MENU_END_OF_RUN = 9;
```

정의 해 놓은 상수를 이용하여 번호를 입력 받고 해당 메뉴를 수행한다.

➔ LinkedBag 클래스

1) add 함수

```
// Bag에 원소를 추가한다
public boolean add(E anElement) {
    if (this.isFull()) {
        return false;
    } else {
        LinkedNode<E> newNode = new LinkedNode<E>();
        newNode.setElement(anElement);
        newNode.setNext(this.head());
        this.setHead(newNode);
        this.setSize(this.size() + 1);
        return true;
    }
}
```

⇨ isFull 함수로 가방이 가득 찼는지 확인하고 가득 찼다면 false를 반환.(사실 LinkedNode 에서는 'full'이라는 개념이 없기 때문에 의미없는 확인임)

⇨ 가득 차 있지 않다면 새로운 노드를 만들어 입력 받은 값을 넣는다.

⇨ 새로운 head로 설정하여 앞부분에 넣어준다.

⇨ Size를 1 증가 시켜 다시 설정해준다. 그리고 true를 반환한다.

```
public boolean isFull() {
    return false;
    // 원소 저장 개수에 영향 받지 않으므로
    // 시스템 메모리 부족 오류는 없다고 가정한다
}
```

⇨ Bag이 가득 차 있는지 확인하는 isFull 함수. Size가 Bag의 크기와 일치하는지 확인.

2) remove 함수

```
// Bag에서 원소를 삭제한다
public boolean remove(E anElement) {
    if (this.isEmpty()) { // 비어있는지 확인한다
        return false; // 비어있다면 false 반환
    } else {
        LinkedNode<E> previousNode = null; // 이전노드를 나타낼 변수
        LinkedNode<E> currentNode = this._head; // 현재 노드를 head로 지정한다
        boolean found = false; // 찾았는지 여부를 저장할 변수

        // 첫 번째 단계 : 삭제할 위치 찾기
        while (currentNode != null && !found) {
            if (currentNode.element().equals(anElement)) {
                found = true;
            } else {
                previousNode = currentNode;
                currentNode = currentNode.next();
            }
        }
```

⇨ 주어진 원소의 위치를 while문을 통해 찾는다. 이때, Coin 클래스의 equals 함수를 활용하는데 해당 객체를 찾은 경우 인덱스와 찾았는지 여부를 변수에 저장한다. 삭제할 노드를 찾았다면 currentNode에 저장된 채로 단계2로 넘어간다.

```
// 두 번째 단계 : 삭제하기
if (!found) { // 찾지 못한 경우는 false를 반환
    return false;
} else { // 삭제할 대상을 찾은 경우,
    if (currentNode == this.head()) { // 삭제할 대상이 head인 경우
        this.setHead(this.head().next()); // head를 next로 바꾸어 head를 없앤다
    }
    else {
        previousNode.setNext(currentNode.next()); // 이전노드의 다음을 현재노드의 다음으로 저장하여 현재노드를 삭제한다.
    }
    this.setSize(this.size() - 1); // size는 하나 줄여 저장한다
    return true;
}
```

⇨ 찾지 못한 경우 false를 반환

⇨ 찾은 경우 찾은 대상이 head인 경우는 head.next를 새로운 head로 저장하여 삭제한다.

⇨ 찾은 대상이 head가 아니라 중간 node인 경우는 이전 노드의 다음을 현재 노드의 다음으로 설정하여 currentNode를 삭제한다.

⇨ Size는 1 감소하여 저장. True 반환

3) doesContain 함수

```
// Bag 안에 주어진 원소가 존재하는지 확인한다
public boolean doesContain(E anElement) {
    LinkedNode<E> currentNode = this.head(); // head를 currentNode로 지정
    while (currentNode != null) { // currentNode가 null이 아닌동안 반복
        if (currentNode.element().equals(anElement)) {
            // 만약 찾는 원소가 있다면 true 반환
            return true;
        }
        currentNode = currentNode.next(); // 다음 노드 반복
    }
    return false;
}
```

LinkedNode 객체를 생성하여 currentNode로 head를 지정한다. currentNode가 null이 아닌동안 반복하여 확인한다. 만약 찾는 원소가 있다면 true를, 그렇지 않다면 false를 반환한다. 비교할 때는 Coin 객체의 equals 함수를 활용한다.

4) frequencyOf 함수

```
// Bag 안에 주어진 원소가 몇 개 있는지 확인한다
public int frequencyOf(E anElement) {
    int frequencyCount = 0; // 원소를 count할 변수 초기화
    LinkedNode<E> currentNode = this._head; // head를 currentNode로 지정
    while (currentNode != null) { // currentNode가 null이 아닌동안 반복
        if (currentNode.element().equals(anElement)) {
            // 주어진 원소가 있다면 count증가
            frequencyCount++;
        }
        currentNode = currentNode.next(); // 다음 노드 반복
    }
    return frequencyCount;
}
```

주어진 원소가 몇 개 있는지 저장할 frequencyCount 변수를 선언하고 0으로 초기화한다. LinkedNode 객체를 생성하고 head를 currentNode로 지정한다. currentNode가 null이 아닌동안 반복하여 주어진 원소가 몇 개 있는지 확인하여 frequencyCount를 1 증가시킨다. 이때도 Coin 클래스의 equals 함수를 활용하였다.

5) removeAny 함수

```java
// Bag에서 원소 하나를 무작위로 삭제한다-> head 노드 삭제
public E removeAny() {
    if (this.isEmpty()) { // 비어있는 경우 null 반환
        return null;
    } else {
        E removedElement = this.head().element(); // head에 있는 element를 저장
        this.setHead(this.head().next()); // head 다음 노드를 head로 지정
        this.setSize(this.size() - 1); // size를 하나 감소한다
        return removedElement; // 저장할 element를 반환
    }
}
```

아무 원소나 삭제하는 함수이다. Head 노드를 삭제하도록 구현하였다. Head에 있는 element 값을 변수에 저장하고 head를 삭제한다. 그리고 저장된 element값을 반환한다.

➔ Coin 클래스

```java
// 객체 값을 비교하는 함수
@Override
public boolean equals(Object otherCoin) {
    if (otherCoin.getClass() != Coin.class) { // 주어진 객체의 클래스 정보를 받아와 Coin 클래스와 동일한지 확인
        return false;
    } else { // Coin의 class를 안전하게 Coin class로 형변환 가능
        return (this.value() == ((Coin) otherCoin).value()); // class가 동일하기 때문에 해당 객체의 값을 비교
    }
}
```

객체의 값을 비교하는 equals 함수이다. getClass 함수를 이용하여 Class 정보를 받아와 먼저 비교하고 Coin 클래스와 동일하지 않다면 false를 반환한다. 만약 동일하다면 해당 객체의 값을 비교하는데 값을 비교하여 일치하면 true를, 일치하지않으면 false를 반환하도록 구현하였다.

➔ LinkedNode 클래스

```java
// 생성자
// element와 next를 null로 초기화한다
public LinkedNode() {
    this.setElement(null);
    this.setNext(null);
}

// element를 givenElement로, next를 null로 초기화한다
public LinkedNode(E givenElement) {
    this.setElement(givenElement);
    this.setNext(null);
}

// element를 givenElement로, next를 givenNext로 초기화한다
public LinkedNode(E givenElement, LinkedNode<E> givenNext) {
    this.setElement(givenElement);
    this.setNext(givenNext);
}
```

LinkedNode 생성자. 주어진 값이 없으면 null로 초기화하고, 만약 값이 주어지면 주어진 값으로 초기화한다. 그 외 LinkedNode 클래스에는 현재 코인에 대한 값과 다음 노드의 getter/setter 함수들이 공개함수로 존재한다.

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.
- ArrayBag과 비교했을 때, Array와 달리 크기에 제한이 없다. Array는 한번 지정한 크기를 변경하려면 프로그램을 다시 실행하거나 clear 함수를 이용해야 하는 반면 LinkedNode를 이용한 Bag은 계속 추가가 가능하다.
- Array보다 원소 추가와 삭제가 쉽다. 배열을 이용했을 경우 가장 마지막에 추가해야 하지만 LinkedNode는 원하는 위치에 삽입이 가능하다. 그리고 삭제의 경우 배열은 삭제한 원소 이후의 원소를 모두 옮겨주어야 하지만 LinkedNode는 연결을 끊으면 되기 때문에 삭제가 편리하다.

➔ 단점

- 처음에 클래스와 함수 역할을 뚜렷하게 나누는 것이 쉽지않다. 객체 지향 프로그램 설계할 때 시간이 오래 걸린다.
- 입/출력까지 모두 분리 하다 보니 코드양이 많아지고 시간이 오래 걸린다.
- Array를 이용한 구현보다 메모리 사용량이 많다. 배열은 노드 간의 연결 정보가 필요가 없이 때문이다.

3. 실행 결과 분석

(1) 입력과 출력 (화면 capture하여 제출)

```
<terminated> _DS03_Main_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 3. 27. 오전 12:25:11)
<<< 동전 가방 프로그램을 시작합니다 >>>

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 10
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 20
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 30
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 1
? 동전 값을 입력하시오: 100
- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .
```

```
? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 2
? 동전 값을 입력하시오: 50
- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 2
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전 하나가 가방에서 정상적으로 삭제되었습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 3
? 동전 값을 입력하시오: 20
- 주어진 값을 갖는 동전이 가방 안에 존재합니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 3
? 동전 값을 입력하시오: 70
- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 4
? 동전 값을 입력하시오: 5
- 주어진 값을 갖는 동전의 개수는 2 개 입니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 4
? 동전 값을 입력하시오: 80
- 주어진 값을 갖는 동전의 개수는 0 개 입니다 .

? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): 9
 가방에 들어 있는 동전의 개수 : 6
동전 중 가장 큰 값 : 100
모든 동전 값의 합 : 170
<<< 동전 가방 프로그램을 종료합니다 >>>
```

(2) 결과 분석 (자신의 논리적 평가, 기타 느낀 점)

⇨ Generic Type을 이용한 구현이 쉽지는 않았고 다시 처음부터 구현하라고 한다면 다시 할 수 있을까라는 생각이 들 정도로 익숙하지 않은 구현이었다. 그래도 Generic Type을 이용하니 코드가 유연해지고 깔끔하다는 생각이 들었다. 이 과제에서는 Coin 클래스를 정의하여 이용했지만 이를 바꾸어 구현한다면 재사용성 면에서도 꽤 유용하다고 생각된다.

⇨ 자료형만 바꾸었는데 ArrayBag보다 더 깔끔하다는 생각이 들었다. 일단 동전을 받는 최대 크기를 지정하지 않는다는 점, remove시 모든 원소에 대한 접근이 필요 없다는 점이 가장 큰 차이점인 것 같았다. 이 과제에서는 노드 위치를 지정하여 삽입하지는 않았지만 만약 add를 원하는 위치에 넣는다면 그 또한 배열과는 다른 차이점이 될 것 같다.

⇨ Sequential Search?

- doesContain() 의 ver1과 ver2차이 : ver1은 찾고자 하는 element가 있으면 for문이 종료되고 ver2는 element를 찾아도 노드의 끝까지 실행한다.
- 시간적 차이는 ver1이 처리속도 빠르지만 변수를 하나 더 선언하기 때문에 메모리가 더 크다. Ver2는 속도는 느리지만 메모리가 더 작다. 하지만 이 차이는 매우 근소할 것으로 생각된다.

4. 소스코드

[main class]

```
public class _DS03_Main_201702039_오명주 {
    public static void main(String[] args) {
        AppController appController = new AppController();
        // AppController가 실질적인 main class
        appController.run();
        // 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다
    }
}
```

```java
public class Coin {
    // 상수
    private static final int DEFAULT_VALUE = 0;

    // private instance variables
    private int _value; // 동전의 금액

    // 생성자
    // 객체 생성시 주어진 값이 없는 경우
    public Coin() {
        // default 값은 0으로 설정
        this._value = DEFAULT_VALUE;
    }

    // 객체 생성시 주어진 값이 있는 경우
    public Coin(int givenValue) {
        this._value = givenValue;
    }

    // 공개 함수
    // getter / setter
    public int value() {
        return this._value; // 동전 값 반환
    }

    public void setValue(int newValue) {
        this._value = newValue; // 입력받은 값으로 값 설정
    }

    // 객체 값을 비교하는 함수
    @Override
    public boolean equals(Object otherCoin) {
        if (otherCoin.getClass() != Coin.class) { // 주어진 객체의 클래스 정보를 받아와 Coin 클래스와 동일한지 확인
            return false;
        } else { // Coin의 class를 안전하게 Coin class로 형변환 가능
            return (this.value() == ((Coin) otherCoin).value()); // class가 동일하기 때문에 해당 객체의 값을 비교
        }
    }
}
```

```java
import java.util.Scanner;

public class AppView {
    // 비공개 상수, 변수
    private static Scanner scanner = new Scanner(System.in); // scanner import하여 입력받음

    // 생성자 : 객체 생성할 일 없음
    private AppView() {

    }

    // 입력 관련 함수
    // 수행할 메뉴의 값을 입력받는 함수
    public static int inputMenuNumber() {
        AppView.outputLine("");
        AppView.output("? 수행하려고 하는 메뉴 번호를 선택하시오 (add:1, remove:2, search:3, frequency:4, exit:9): ");
        int inputValue = scanner.nextInt(); // scanner를 이용하여 정수를 입력받는다
        return inputValue;                  // 입력받은 정수를 return한다
    }

    // 최대 Bag의 크기를 입력받는 함수
    public static int inputCapacityOfCoinBag() {
        AppView.outputLine("");
        AppView.output("? 동전 가방의 크기, 즉 가방에 들어갈 동전의 최대 개수를 입력하시오: ");
        int inputValue = scanner.nextInt(); // scanner를 이용하여 정수를 입력받는다
        return inputValue;                  // 입력받은 정수를 return한다
    }

    // 동전의 값을 입력받는 함수
    public static int inputCoinValue() {
        int inputValue = scanner.nextInt(); // scanner를 이용하여 정수를 입력받는다
        return inputValue;                  // 입력받은 정수를 return한다
    }

    // 출력 관련 함수
    // 한줄을 출력하는 함수 (한줄이 띄워지지않는다)
    public static void output(String message) {
        System.out.print(message);          // 입력받은 message를 출력한다
    }

    // 한줄을 출력하는 함수 (한줄이 띄워진다)
    public static void outputLine(String message) {
        System.out.println(message);        // 입력받은 message를 출력한다
    }

}
```

```
    private static final int MENU_ADD = 1;
    private static final int MENU_REMOVE = 2;
    private static final int MENU_SEARCH = 3;
    private static final int MENU_FREQUENCY = 4;
    private static final int MENU_END_OF_RUN = 9;
    // static은 클래스 하나에 선언하지만 모든 클래스가 공유
    // final은 수정할 수 없는 상수를 의미

    // 비공개 인스턴스 변수들
    private LinkedBag<Coin> _coinBag;

    // getter/setter
    private LinkedBag<Coin> coinBag() {
        return this._coinBag;
    }

    private void setCoinBag(LinkedBag<Coin> newCoinBag) {
        this._coinBag = newCoinBag;
    }

    // 공개함수
    public void run() {
        // 프로그램을 시작하는 출력문
        AppView.outputLine("<<< 동전 가방 프로그램을 시작합니다 >>>");


        this.setCoinBag(new LinkedBag<Coin>()); //  객체 생성


        int menuNumber = AppView.inputMenuNumber(); // 수행할 메뉴 번호를 입력받는 입력문 호출
        while (menuNumber != MENU_END_OF_RUN) {
            switch (menuNumber) {
            case MENU_ADD: // 1을 입력받으면
                this.addCoin(); // addCoin함수 호출
                break;
            case MENU_REMOVE: // 2를 입력받으면
                this.removeCoin(); // removeCoin함수 호출
                break;
            case MENU_SEARCH: // 3을 입력받으면
                this.searchForCoin(); // searchForCoin함수 호출
                break;
            case MENU_FREQUENCY: // 4를 입력받으면
                this.frequencyOfCoin(); // freQuencyOfCoin함수 호출
                break;

            default:
                this.undefinedMenuNumber(menuNumber); // 잘못된 번호를 입력받은 경우
            }

            // 프로그램을 반복하여 수정
            menuNumber = AppView.inputMenuNumber();
        }
        this.showStatistics(); // 프로그램이 끝나면 통계 출력
        AppView.outputLine("<<< 동전 가방 프로그램을 종료합니다 >>>");
    }

    // 비공개함수
    // 정수를 입력받아 Bag에 넣는 함수
    private void addCoin() {
        if (this.coinBag().isFull()) { // 만약 Bag의 가득찼다면
            AppView.outputLine("- 동전 가방이 꽉 차서 동전을 넣을 수 없습니다 ."); // 동전을 넣지못한다는 출력문 출력
        } else { // Bag이 비었다면
            AppView.output("? 동전 값을 입력하시오: ");
            int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
            if (this.coinBag().add(new Coin(coinValue))) {
                AppView.outputLine("- 주어진 값을 갖는 동전을 가방에 성공적으로 넣었습니다 .");
            } else {
                AppView.outputLine("- 주어진 값을 갖는 동전을 가방에 넣는데 실패하였습니다 .");
            }
        }
    }

    // 정수를 입력받아 해당 동전이 있으면 Bag에서 삭제하는 함수
    private void removeCoin() {
        AppView.output("? 동전 값을 입력하시오: ");
        int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
        if (!this.coinBag().remove(new Coin(coinValue))) { // 정상적으로 삭제되는지 확인
            // 정상적으로 삭제되지 않는다면
            // 주어진 값의 동전이 Bag에 없으므로 존재하지 않는다는 출력문 출력
            AppView.outputLine("- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다 .");
        } else {
            // 정상적으로 삭제된 경우
            AppView.outputLine("- 주어진 값을 갖는 동전 하나가 가방에서 정상적으로 삭제되었습니다 .");
        }
    }

    // 정수를 입력받아 해당 숫자의 동전이 Bag에 있는지 확인하는 함수
    private void searchForCoin() {
        AppView.output("? 동전 값을 입력하시오: ");
        int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
        if (this.coinBag().doesContain(new Coin(coinValue))) { // 해당 동전이 Bag에 있는지 확인
            AppView.outputLine("- 주어진 값을 갖는 동전이 가방 안에 존재합니다 .");
        } else {
            AppView.outputLine("- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다 .");
        }
    }

    // 정수를 입력받아 해당 숫자의 동전이 Bag에 몇개 있는지 확인하는 함수
    private void frequencyOfCoin() {
        AppView.output("? 동전 값을 입력하시오: ");
        int coinValue = AppView.inputCoinValue(); // 동전 값을 입력받는다
        int frequency = this.coinBag().frequencyOf(new Coin(coinValue)); // 해당 숫자의 동전이 존재하는 개수의 정수를 입력받는다
        AppView.outputLine("- 주어진 값을 갖는 동전의 개수는 " + frequency + " 개 입니다 .");
    }

    // 메뉴에 없는 정수를 입력했을 경우 호출하는 함수
    private void undefinedMenuNumber(int menuNumber) {
        AppView.outputLine("- 선택된 메뉴 번호 " + menuNumber + " 는 잘못된 번호입니다 .");
    }

    // Bag에 있는 동전의 값의 합을 return하는 함수
    private int sumOfCoinValues() {
        int sum = 0;
        for (int i = 0; i < this.coinBag().size(); i++) { // Bag의 크기만큼 반복하여
            sum += this.coinBag().elementAt(i).value(); // 동전의 값을 순서대로 반환하여 sum에 합하여 저장
        }
        return sum;
    }

    // Bag에 있는 동전 중 가장 큰 값을 return하는 함수
    private int maxCoinValue() {
        int maxValue = 0;
        for (int i = 0; i < this.coinBag().size(); i++) { // Bag의 크기만큼 반복하여
            if (maxValue < this.coinBag().elementAt(i).value()) { // maxValue의 값을 동전의 값과 비교
                maxValue = this.coinBag().elementAt(i).value(); // maxValue보다 큰 값이면 다시 maxValue에 저장
            }
        }
        return maxValue;
    }

    // 동전의 개수, 가장큰 값, 동전들의 합을 출력하는 함수
    private void showStatistics() {
        AppView.outputLine(" 가방에 들어 있는 동전의 개수 : " + this.coinBag().size()); // Bag의 크기 출력
        AppView.outputLine("동전 중 가장 큰 값 : " + this.maxCoinValue()); // Bag 속의 가장 큰 동전 값 출력
        AppView.outputLine("모든 동전 값의 합 : " + this.sumOfCoinValues()); // Bag 속의 동전들의 합 출력
    }

}
```

```java
    // 비공개 인스턴스 변수
    private int _size; // 가방이 가지고 있는 원소의 갯수(연결된 노드의 갯수)
    private LinkedNode<E> _head; // 연결 체인의 맨 앞 노드를 소유한다

    // 생성자
    public LinkedBag() {
        setSize(0);
        setHead(null);
    }

    // getter / setter
    // Bag에 들어있는 개수를 확인한다
    public int size() {
        return this._size;
    }

    private void setSize(int newSize) {
        this._size = newSize;
    }

    private LinkedNode<E> head() {
        return this._head;
    }

    private void setHead(LinkedNode<E> newHead) {
        this._head = newHead;
    }

    // Bag이 비어있는지 확인한다
    public boolean isEmpty() {
        return (this.size() == 0);
    }

    public boolean isFull() {
        return false;
        // 원소 저장 개수에 영향 받지 않으므로
        // 시스템 메모리 부족 오류는 없다고 가정한다
    }

    // Bag 안에 주어진 원소가 존재하는지 확인한다
    public boolean doesContain(E anElement) {
        LinkedNode<E> currentNode = this.head(); // head를 currentNode로 지정
        while (currentNode != null) { // currentNode가 null이 아닐동안 반복
            if (currentNode.element().equals(anElement)) {
                // 만약 찾는 원소가 있다면 true 반환
                return true;
            }
            currentNode = currentNode.next(); // 다음 노드 반복
        }
        return false;
    }

    // Bag 안에 주어진 원소가 몇 개 있는지 확인한다
    public int frequencyOf(E anElement) {
        int frequencyCount = 0; // 원소를 count할 변수 초기화
        LinkedNode<E> currentNode = this._head; // head를 currentNode로 지정
        while (currentNode != null) { // currentNode가 null이 아닐동안 반복
            if (currentNode.element().equals(anElement)) {
                // 주어진 원소가 있다면 count증가
                frequencyCount++;
            }
            currentNode = currentNode.next(); // 다음 노드 반복
        }
        return frequencyCount;
    }

    // Bag 안에 주어진 순서의 원소를 얻는다
    // 맨 앞의 order 0, 맨 뒤가 order size()-1
    // 주어진 순서가 적정 범위를 벗어나 있으면 null을 얻는다
    public E elementAt(int anOrder) {
        if ((anOrder < 0) || (anOrder >= this.size())) {
            return null; // anOrder 가 적정 범위를 벗어나 있다
        } else { // anOrder가 적정 범위 안에 있다
            LinkedNode<E> currentNode = this.head();
            for (int i = 0; i < anOrder; i++) {
                currentNode = currentNode.next();
            }
            return currentNode.element();
        }
    }

    // Bag에 원소를 추가한다
    public boolean add(E anElement) {
        // LinkedList에서는 가득 차는 개념이 없으므로 무의미한 확인.
        if (this.isFull()) {
            return false;
        } else {
            LinkedNode<E> newNode = new LinkedNode<E>(); // 새로운 노드 생성
            newNode.setElement(anElement); // 새로운 노드에 입력받은 값을 넣는다
            newNode.setNext(this.head()); // 새로운 head로 지정한다
            this.setHead(newNode); // head 저장
            this.setSize(this.size() + 1); // size 증가했음을 저장
            return true;
        }
    }

    // Bag에서 원소를 삭제한다
    public boolean remove(E anElement) {
        if (this.isEmpty()) { // 비어있는지 확인한다
            return false; // 비어있다면 false 반환
        } else {
            LinkedNode<E> previousNode = null; // 이전노드를 나타낼 변수
            LinkedNode<E> currentNode = this._head; // 현재 노드를 head로 지정한다
            boolean found = false; // 찾았는지 여부를 저장할 변수

            // 첫 번째 단계 : 삭제할 위치 찾기
            while (currentNode != null && !found) { // 노드가 끝날때까지 반복
                if (currentNode.element().equals(anElement)) { // 찾고자 하는 값의 노드를 발견하면
                    found = true; // 발견여부를 저장한다
                } else {
                    // 찾지 못한경우 다음 노드로 이동
                    previousNode = currentNode;
                    currentNode = currentNode.next();
                }
            }

            // 두 번째 단계 : 삭제하기
            if (!found) { // 찾지 못한 경우는 false를 반환
                return false;
            } else { // 삭제할 대상을 찾은 경우,
                if (currentNode == this.head()) { // 삭제할 대상이 head인 경우
                    this.setHead(this.head().next()); // head를 next로 따꾸어 head를 없앤다
                }
                else {
                    previousNode.setNext(currentNode.next()); // 이전노드의 다음을 현재노드의 다음으로 저장하여 현재노드를 삭제한다.
                }
                this.setSize(this.size() - 1); // size는 하나 줄여 저장한다
                return true;
            }
        }
    }

    // Bag에서 원소 하나를 무작위로 삭제한다-> head 노드 삭제
    public E removeAny() {
        if (this.isEmpty()) { // 비어있는 경우 null 반환
            return null;
        } else {
            E removedElement = this.head().element(); // head에 있는 element를 저장
            this.setHead(this.head().next()); // head 다음 노드를 head로 지정
            this.setSize(this.size() - 1); // size를 하나 감소한다
            return removedElement; // 저장할 element를 반환
        }
    }

    // Bag를 초기화한다. 모든 원소 삭제한다
    public void clear() {
        this.setSize(0);
        this.setHead(null);
    }
}
```

```java
    // 비공개 인스턴스 변수
    private E _element; // 현재 노드에 있는 코인
    private LinkedNode<E> _next; // 다음노드

    // 생성자
    // element와 next를 null로 초기화한다
    public LinkedNode() {
        this.setElement(null);
        this.setNext(null);
    }

    // element를 givenElement로, next를 null로 초기화한다
    public LinkedNode(E givenElement) {
        this.setElement(givenElement);
        this.setNext(null);
    }

    // element를 givenElement로, next를 givenNext로 초기화한다
    public LinkedNode(E givenElement, LinkedNode<E> givenNext) {
        this.setElement(givenElement);
        this.setNext(givenNext);
    }

    // LinkedNode 객체에 있는 element를 얻는다
    public E element() {
        return this._element;
    }

    // LinkedNode 객체의 다음 LinkedNode 객체를 얻는다
    public LinkedNode<E> next() {
        return this._next;
    }

    // LinkedNode에 있는 element를 newElement로 변경한다
    public void setElement(E newElement) {
        this._element = newElement;
    }

    // LinkedNode 객체의 next를 newNext로 변경한다
    public void setNext(LinkedNode<E> newNext) {
        this._next = newNext;
    }

}
```