자료구조 실습 보고서

[제03주] 동전가방(ArrayBag)

2021년 3월 22일

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(ArrayBag) 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

- ArrayBag<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.

View :

- AppView : 프로그램의 입/출력을 담당한다.

Controller :

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

(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.

```
// 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.

```
// 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.

```
// 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.

➔ ArrayBag 클래스

1) add 함수

```
// Bag에 주어진 원소를 넣는다
public boolean add(E anElement) {
    if (this.isFull()) { // 가방이 꽉 찼으므로 넣을 수 없다
        return false;
    } else {
        // 빈 여유 공간이 있으므로 넣는다
        // 원소의 순서가 중요하지 않으므로 아무곳에 넣어도 된다
        // 단, 맨 앞부터 차있는 상태는 유지해야한다
        // 가장 편한곳은 배열 맨 마지막 원소 다음칸
        this.elements()[this.size()] = anElement;
        this.setSize(this.size() + 1); // 실제 사용하는 크기 +1
        return true;
    }
}
```

⇨ isFull 함수로 가방이 가득 찼는지 확인하고 가득 찼다면 false를 반환

⇨ 가득 차 있지 않다면 마지막 원소 다음 칸에 입력 받은 값(객체)를 넣는다.

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

```
// Bag이 가득 차 있는지 알려준다
public boolean isFull() {
    return (this.size() == this.capacity());
    // 실제 들어있는 동전의 개수(size)와 Bag의 크기(capacity)가 동일하면 true를 반환
    // 같지않으면 false를 반환
}
```

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

2) remove 함수

```
int foundIndex = -1; // 음수로 설정
boolean found = false;
// 단계1 : 주어진 원소의 위치를 찾는다
for (int i = 0; i < this.size() && !found; i++) { // Bag의 실제 사용하는 크기만큼 반복
    if (this.elements()[i].equals(anElement)) { // 만약 같은 값을 찾은 경우,
        foundIndex = i; // 인덱스 저장
        found = true; // 찾았음을 저장
    }
}
```

⇨ 주어진 원소의 위치를 for문을 통해 찾는다. 이때, Coin 클래스의 equals 함수를 활용하는데 해당 객체를 찾은 경우 인덱스와 찾았는지 여부를 변수에 저장한다.

```
// 단계2 : 삭제된 원소 이후의 모든 원소를 앞쪽으로 한 칸씩 이동시킨다
if (!found) { // 찾지 못했다면 found는 false이므로 not을 하면 true가 된다
    return false;
} else { // 찾은경우
    for (int i = foundIndex; i < this.size() - 1; i++) { // 찾은 원소 이후의 모든 원소 반복
        this.elements()[i] = this.elements()[i + 1]; // 한칸 앞으로 저장
    }
    this.elements()[this.size() - 1] = null; // 더이상 의미 없는 소유권은 null로!
    this.setSize(this.size() - 1); // 실제 사용하는 크기 -1
    return true;
}
```

⇨ 찾지 못한 경우 false를 반환

⇨ 찾은 경우 찾은 원소 이후 인덱스부터 모든 원소를 한 칸 앞쪽으로 저장한다. For문 활용.

⇨ Size는 1 감소하여 저장. 삭제한 부분은 null 처리. True 반환

3) doesContain 함수

```
// 주어진 원소가 Bag에 있는지 알려준다
public boolean doesContain(E anElement) {
    return (this.indexOf(anElement) >= 0); // 원소가 없다면 음수가 반환되어 false를 반환하게 된다
}
```

해당 함수에서는 indexOf 라는 private 함수를 호출하여 활용하였다.

```
// 해당 동전의 index를 반환한다
// 해당 동전이 Bag에 없다면 음수를 반환
private int indexOf(E anElement) {
    int foundIndex = -1; // 음수를 설정
    for (int i = 0; i < this.size() && foundIndex < 0; i++) {
        if (this.elements()[i].equals(anElement)) { // 만약 찾는 원소가 Bag 안에 존재한다면
            foundIndex = i; // 해당 원소의 index를 foundIndex에 저장
        }
    }
    return foundIndex;
}
```

⇨ 인덱스를 저장할 foundIndex 변수를 생성 후 음수로 초기화한다.

⇨ size만큼 반복하는 for문을 이용하여 찾는 원소가 있는지 확인한다.

⇨ 만약 찾는다면 해당 원소의 index를 저장한다. return했을 때 찾은 경우 양수가 되고 못 찾은 경우 음수가 된다.

4) frequencyOf 함수

```
// 주어진 원소가 Bag에 몇개 있는지 알려준다
public int frequencyOf(E anElement) {
    int frequencyCount = 0;
    for (int i = 0; i < this._size; i++) { // Bag의 실제 사용하는 크기만큼 반복
        if (this.elements()[i].equals(anElement)) { // 찾는 원소가 Bag의 원소라면
            frequencyCount++; // 개수 증가
        }
    }
    return frequencyCount;
}
```

주어진 원소가 몇 개 있는지 저장할 frequencyCount 변수를 선언하고 0으로 초기화한다. Bag의 size만큼 반복하여 해당 원소를 찾는다. 찾는다면 frequencyCount 변수를 개수 증가한다. 하나도 없다면 그대로 0을 반환한다. 이때도 Coin 클래스의 equals 함수를 활용하였다.

➔ Coin 클래스

```
// 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.

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.

➔ 단점

- 처음에 클래스와 함수 역할을 뚜렷하게 나누는 것이 쉽지않다. 객체 지향 프로그램 설계할 때 시간이 오래 걸린다.
- 순서가 상관이 있는 ArrayBag이었다면 함수를 수정해야 하고 추가해야 할 것이다. 재사용성이 상황에 따라 감소한다.
- 입/출력까지 모두 분리 하다 보니 코드양이 많아지고 시간이 오래 걸린다.

3. 실행 결과 분석

(1) 입력과 출력 (화면 capture하여 제출)

```
_DS02_Main_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 3. 22.
<<< 동전 가방 프로그램을 시작합니다 >>>

? 동전 가방의 크기, 즉 가방에 들어갈 동전의 최대 개수를 입력하시오: 6

? 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 가방이 꽉 차서 동전을 넣을 수 없습니다 .
```

```
? 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 : 5
동전 중 가장 큰 값 : 30
모든 동전 값의 합 : 70
<<< 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.

⇨ 지난 주 과제와 동일하게 객체 지향적인 과제를 수행했는데 AppView를 재사용하면서 다시한번 객체 지향 설계의 장점을 되새긴 것 같았다. 출력문을 출력하는 함수는 따로 구현없이 재사용하였다.

4. 소스코드

[main class]

```
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
    }
}
```

```
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(); // scanner을 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(); // scanner을 이용하여 정수를 입력받는다
        return inputValue;                  // 입력받은 정수를 return한다
    }

    // 동전의 값을 입력받는 함수
    public static int inputCoinValue() {
        int inputValue = scanner.nextInt(); // scanner을 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
    }

}
```

```java
public class ArrayBag<E> {

    // 비공개 인스턴스 변수
    private static final int DEFAULT_CAPACITY = 100;
    private int _capacity; // Bag의 최대크기
    private int _size; // Bag의 동전이 실제 들어있는 크기
    private E _elements[]; // ArrayBag의 원소들을 담을 java 배열

    // 생성자
    // 객체 생성시 Bag의 값을 지정하지 않는경우
    @SuppressWarnings("unchecked") // 형변환에 대한 경고를 제외
    public ArrayBag() {
        this.setCapacity(ArrayBag.DEFAULT_CAPACITY); // Bag의 크기는 default 100으로 설정
        this.setElements((E[]) new Object[this.capacity()]);
        this.setSize(0);
    }

    // 객체 생성시 Bag의 값을 지정하는 경우
    @SuppressWarnings("unchecked") // 형변환에 대한 경고를 제외
    public ArrayBag(int givenCapacity) {
        this.setCapacity(givenCapacity); // Bag의 크기는 입력받은 정수의 값으로 설정
        this.setElements((E[]) new Object[this.capacity()]);
        this.setSize(0);
    }

    // 비공개 함수 -> class 내부에서만 사용
    // getter / setter
    private int capacity() {
        return this._capacity;
    }

    private void setCapacity(int newCapacity) {
        this._capacity = newCapacity;
    }

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

    private E[] elements() {
        return this._elements;
    }

    private void setElements(E[] newElements) {
        this._elements = newElements;
    }

    // 해당 동전의 index를 반환한다
    // 해당 동전이 Bag에 없다면 음수를 반환
    private int indexOf(E anElement) {
        int foundIndex = -1; // 음수를 설정
        for (int i = 0; i < this.size() && foundIndex < 0; i++) {
            if (this.elements()[i].equals(anElement)) { // 만약 찾는 원소가 Bag 안에 존재한다면
                foundIndex = i; // 해당 원소의 index를 foundIndex에 저장
            }
        }
        return foundIndex;
    }

    // 공개함수
    // Bag에 들어있는 원소의 개수를 알려준다
    public int size() {
        return this._size;
    }

    // Bag이 비어있는지 알려준다
    public boolean isEmpty() {
        return (this.size() == 0);
        // 실제 들어있는 동전의 개수, size가 0이면 true를 반환
        // 0이 아니면 false를 반환
    }

    // Bag이 가득 차 있는지 알려준다
    public boolean isFull() {
        return (this.size() == this.capacity());
        // 실제 들어있는 동전의 개수(size)와 Bag의 크기(capacity)가 동일하면 true를 반환
        // 같지않으면 false를 반환
    }

    // 주어진 원소가 Bag에 있는지 알려준다
    public boolean doesContain(E anElement) {
        return (this.indexOf(anElement) >= 0); // 원소가 없다면 음수가 반환되어 false를 반환하게 된다
    }

    // 주어진 원소가 Bag에 몇개 있는지 알려준다
    public int frequencyOf(E anElement) {
        int frequencyCount = 0;
        for (int i = 0; i < this._size; i++) { // Bag의 실제 사용하는 크기만큼 반복
            if (this.elements()[i].equals(anElement)) { // 찾는 원소가 Bag의 원소라면
                frequencyCount++; // 개수 증가
            }
        }
        return frequencyCount;
    }

    // Bag에 주어진 원소를 넣는다
    public boolean add(E anElement) {
        if (this.isFull()) { // 가방이 꽉 찼으므로 넣을 수 없다
            return false;
        } else {
            // 빈 여유 공간이 있으므로 넣는다
            // 원소의 순서가 중요하지 않으므로 아무곳에 넣어도 된다
            // 단, 한 번부터 차있는 상태는 유지해야한다
            // 가장 편한곳은 배열 맨 마지막 원소 다음칸
            this.elements()[this.size()] = anElement;
            this.setSize(this.size() + 1); // 실제 사용하는 크기 +1
            return true;
        }
    }

    // Bag에서 지정된 원소를 찾아서 있으면 제거한다
    public boolean remove(E anElement) {
        int foundIndex = -1; // 음수로 설정
        boolean found = false;
        // 단계1 : 주어진 원소의 위치를 찾는다
        for (int i = 0; i < this.size() && !found; i++) { // Bag의 실제 사용하는 크기만큼 반복
            if (this.elements()[i].equals(anElement)) { // 만약 같은 값을 찾은 경우,
                foundIndex = i; // 인덱스 저장
                found = true; // 찾았음을 저장
            }
        }
        // 단계2 : 삭제된 원소 이후의 모든 원소를 앞쪽으로 한 칸씩 이동시킨다
        if (!found) { // 찾지 못했다면 found는 false이므로 not을 하면 true가 된다
            return false;
        } else { // 찾은경우
            for (int i = foundIndex; i < this.size() - 1; i++) { // 찾은 원소 이후의 모든 원소 반복
                this.elements()[i] = this.elements()[i + 1]; // 한칸 앞으로 저장
            }
            this.elements()[this.size() - 1] = null; // 마이상 의미 없는 쓰레기존 null로!
            this.setSize(this.size() - 1); // 실제 사용하는 크기 -1
            return true;
        }
    }

    // Bag을 비운다
    public void clear() {
        // 모든 원소에 대해 null을 진행
        for (int i = 0; i < this.size(); i++) {
            this.elements()[i] = null;
        }
        this.setSize(0); // 실제 사용하는 크기도 0으로 설정
    }

    // 주어진 순서 anOrder에 있는 원소를 돌려준다
    public E elementAt(int anOrder) {
        if ((0 <= anOrder) && (anOrder < this.size())) { // 주어진 순서에 대한 유효성 확인
            return this.elements()[anOrder];
        } else {
            return null;
        }

    }

}
```

```java
public class AppController {
    // 상수
    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 선언되지만 모든 클래스가 공유
    // final은 수정할 수 없는 상수를 의미

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

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

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

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

        int coinBagSize = AppView.inputCapacityOfCoinBag(); // 가방의 최대 개수를 입력받는 입력문 호출
        this.setCoinBag(new ArrayBag<Coin>(coinBagSize)); // 입력받은 크기를 가진 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: // 4을 입력받으면
                this.frequencyOfCoin(); // freQuencyOfCoin함수 호출
                break;

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

            // 프로그램을 반복하여 수행
            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 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 .");
            }
        }
    }

    // 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("- 선택한 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
    }
}
```