자료구조: 2022년 1학기 [실습]

제 3 주:

동전 가방

배열로 구현하는 가방

강 지 훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

실습 목표

실습 목표

- 이론적 관점
  - Bag 의 개념을 이해한다
  - 캡슐화 (Encapsulation): 사용자적 관점에서 객체의 공개 함수를 정의하는 행위의 중요성
  - Generic Type 의 개념과 그 필요성을 이해한다

- 구현적 관점
  - Java 언어로 Class 표현
  - Class 선언에 generic Type 의 적용
  - Class “ArrayBag”: 배열로 Bag 을 구현하는 방법
  - 인스턴스 변수에 대해 getter/setter 를 선언하여 사용하는 이유
  - Model-View-Controller 설계 방식의 적용
  - “equals()” 함수의 의미와 적용

과제에서 해야 할 일

ArrayBag 과 관련된 필요한 기능

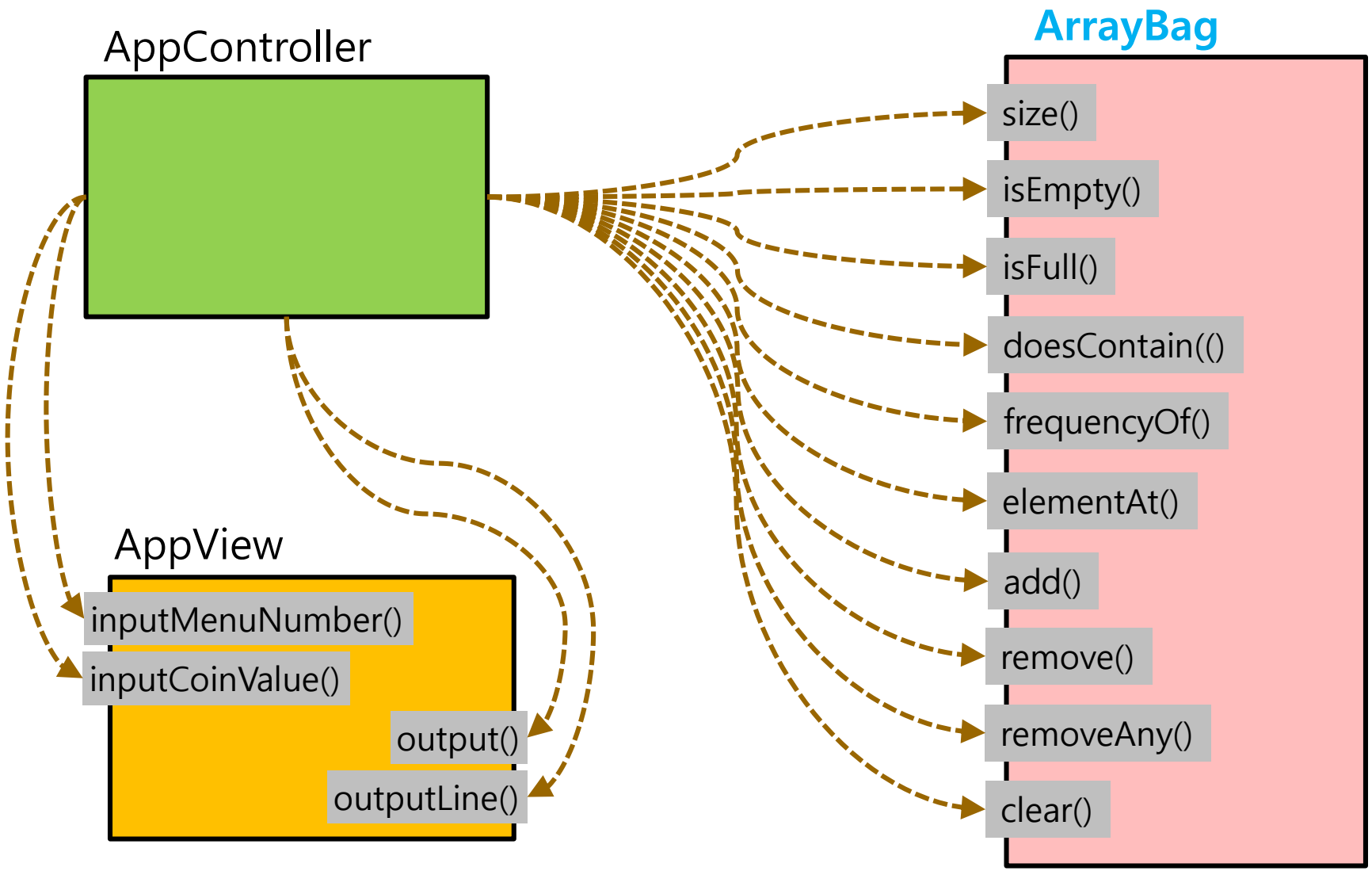
- 초기화
  - 코인 값들의 초기화
- 코인 입력
- 코인 수 세기
  - 코인을 정상적으로 저장할 때마다 +1
- 코인 삭제
- 전체 코인 수
- 코인이 가방에 있는지 확인
- 가방 안에 있는 특정 코인의 개수

이 과제에서 필요한 객체는?

- class AppController
- enum MainMenu
- class AppView
- Model
  - class ArrayBag<E>
  - class Coin

Generic Type:
가방의 구조와 무관하게, 가방에 넣을 원소는 필요에 따라 결정될 것이다.
Class 를 정의할 때, 원소의 type, 즉 class 를 확정하지 않은 채로 원소의 type으로 사용하는 type 을 "generic type" 이라 한다.

MVC 구조


AppController
ArrayBag
size()
isEmpty()
isFull()
doesContain(()
frequencyOf()
elementAt()
add()
remove()
removeAny()
clear()
AppView
inputMenuNumber()
inputCoinValue()
output()
outputLine()

입출력

- 메뉴 입력에 따른 할 일
  - 처음 가방에 들어갈 최대 가방 사이즈를 입력한다.
  - 메뉴화면을 보여준다.
  - 9 가 입력 되면 "<9 가 입력되어 종료합니다>" 라는 메시지를 내보내고 프로그램을 종료한다
  - 1 (add) 이 입력되면, 동전의 액수를 입력 받아 동전 가방에 동전을 넣는 일을 실행한다
  - 2 (remove) 가 입력되면, 동전의 액수를 입력 받아 동전 가방에서 동일한 값을 갖는 동전 하나를 삭제한다.
  - 3 (search) 이 입력 되면, 동전의 액수를 입력 받아 동전 가방에 동일한 값을 갖는 동전이 존재하는지 검색하여 그 여부를 출력한다.
  - 4 (frequency) 가 입력되면, 코인의 액수를 입력 받아 동전 가방에 동일한 값을 갖는 동전의 개수를 세어 출력한다.

- 통계 출력
  - 프로그램 종료 시점에, 가방에 들어 있는 동전의 개수, 동전 중에서 가장 큰 값, 모든 동전들의 값의 합을 출력한다.

❑ 출력의 예 [1]

```
<< 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 가방에 넣을 수 없습니다.
```

출력의 예 [2]

```
? 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 값의 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 대한 수행을 종료합니다.

가방에 들어 있는 동전의 개수: 5
동전 중에서 가장 큰 값: 30
모든 동전들의 값의 합: 70

<< 동전 가방 프로그램을 종료합니다 >>
```

<단계 1>

main()

main()을 위한 class

```
public class _DS03_Main_학번_이름 {
    public static void main (String[] args)
    {
        AppController  appController = new AppController() ;
            // AppController 가 실질적인 main class 이다.
        appController.run() ;
            // 여기 main()에서는 앱 실행이 시작되도록 해주는 일이 전부이다.
    }
}
```

Class “AppController”

AppController:

```
public class AppController {
  // 상수
  ……
  // 비공개 인스턴스 변수들
  ……

  // 생성자
  public AppController()
  {
  }

  // 비공개함수의 구현
  ……

  // 공개함수의 구현
  ……
} // End of class “AppController”
```

AppController: 상수

```
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 ;
```

AppController: run()

```
public class AppController {
    // 상수
    ……
    // 비공개 인스턴스 변수들
    ……
    // 생성자
    ……
    // 공개함수
    public void run() {
        AppView.outputLine ("<<< 동전 가방 프로그램을 시작합니다 >>>") ;

        // 다음 단계에서 채워야 할 곳
        int  menuNumber = AppView.inputMenuNumber() ;
        while ( menuNumber != MENU_END_OF_RUN ) {
            switch ( menuNumber ) {
            case MENU_ADD:
                this.addCoin() ;
                break;
            case  MENU_REMOVE:
                this.removeCoin() ;
                break;
            case  MENU_SEARCH:
                this.searchForCoin() ;
                break;
            case MENU_FREQUENCY:
                this.frequencyOfCoin() ;
                break;
            default:
                this.undefinedMenuNumber(menuNumber) ;
            }
            menuNumber = AppView.inputMenuNumber() ;
        }
        this.showStatistics() ;
        AppView.outputLine ("<<< 동전 가방 프로그램을 종료합니다 >>>") ;
    }
```

AppController: 비공개 함수

```
public class AppController {
    // 상수
    ……

    // 비공개 인스턴스 변수
    ……

    // 생성자
    ……

    // 비공개 함수의 구현
    private void addCoin () {
        // 다음 단계에서 채워야 할 곳
    }

    private void removeCoin () {
        // 다음 단계에서 채워야 할 곳
    }

    private void searchForCoin () {
        // 다음 단계에서 채워야 할 곳
    }
    private void frequencyOfCoin () {
        // 다음 단계에서 채워야 할 곳
    }

    private void undefinedMenuNumber (int menuNumber) {
        // 다음 단계에서 채워야 할 곳
    }

    private void showStatistics () {
        // 다음 단계에서 채워야 할 곳
    }
```

Class “AppView”

AppView:

- 지난 과제에 만든 것을 가져와서, 일단 다음의 함수만 남겨서 사용한다.

```
public class AppView {
  // 비공개 상수/변수들
  private static Scanner       scanner = new Scanner(System.in);

  // 생성자
  private  AppView ()   { }

  // 출력 관련 공개 함수들
  public static void  output (String message) {...}
  public static void  outputLine (String message) {...}
}
```

AppView: inputMenuNumber()

- inputMenuNumber() 추가하여 완성한다.

```
public class AppView {
   // 비공개 상수/변수들
   private static Scanner        scanner = new Scanner(System.in);

   // 생성자
   private AppView ()   { }

   // 출력 관련 공개함수
   public static void  output (String message) {...}
   public static void  outputLine (String message) {...}

   // 입력 관련 공개함수
   public static int inputMenuNumber() {
         ..... // 여기를 채우시오
   }

}
```

<단계 1> 점검

❑ 실행해 보자

- 현재까지 완성된 상태로 실행해 보자.
  - 메뉴 선택이 정상적으로 작동하는지 확인하자.
  - run() 의 while 구조를 확실하게 이해하자.

<단계 2>

Class “AppView”

AppView: 입력 함수 추가

- "inputCapacityOfCoinBag()" 과 "inputCoinValue()" 를 추가하여 완성하자.
  - 입출력 예제를 보고, 그에 맞게 작성한다.
  - 잘못 입력된 경우, 오류 처리하고 다시 입력 받는다.

```
public class AppView {
   // 비공개 상수/변수들
   .....
   // 생성자
   ......

   public static int  inputCapacityOfCoinBag() {
         ..... // 여기를 채우시오
   }

   public static int inputCoinValue() {
         ..... // 여기를 채우시오
   }

   public static int  inputMenuNumber () { ... }

   public static void  output (String aString) { .... }
   public static void  outputLine (String aString) { .... }
```

Class “Coin”

Coin: 생성자, 공개함수

생성자

- public Coin ( ) ;
  - Coin 객체를 생성한다. 동전의 값은 모르는 상태이다.
- public Coin (int givenValue) ;
  - 전달받은 값을 갖는 새로운 Coin 객체를 생성한다.

공개함수

- public int value ( ) ;
  - 코인의 금액을 얻는다.
- public void setValue (int newValue) ;
  - 전달받은 newValue를 현재 코인의 금액으로 설정한다.
- public boolean equals (**Object** otherCoin) ;
  - 현재 코인의 금액이 otherCoin 의 금액과 같은지 확인한다

Class “Coin” 의 구현

Coin: 인스턴스 변수

```
public class Coin {
        // Constants
        private static final int DEFAUT_VALUE = 0 ;

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

Coin: 생성자, 공개함수

- 생성자
  - public Coin ( )
    - this._value 를 DEFALUT_VALUE 로 설정한다.
  - public Coin (int givenValue)
    - this._value 를 givenValue 값으로 설정한다.

- 공개 함수
  - public int value ( )
    - this._value 값을 돌려준다.
  - public void setValue (int newValue)
    - this._value 를 newValue 값으로 설정한다.
  - public boolean equals (Object otherCoin)
    - otherCoin 의 실제 class 가 Coin 인지 확인한다. Coin 이 아니면 false.
    - otherCoin 의 class 가 Coin 이면, this.value() 와 otherCoin 의 value() 를 비교한다.
      - 동일한 값을 가지고 있으면 true, 아니면 false 를 얻는다.

```
@Override
public boolean equals (Object otherCoin) {
      if ( otherCoin.getClass() != Coin.class ) {
            return  false ;
      }
      else { // aCoin 의 class 를 안전하게 Coin class 로 형 변환 가능.
            return  ( this.value() == ((Coin) otherCoin).value() ) ;
      }
}
```

Class “ArrayBag<E>”

ArrayBag⟨E⟩: Generic Type

```
public class ArrayBag<E> {



}
```

Generic Type의 선언:
가방의 구조와 무관하게, 가방에 넣을 원소로서의 객체의 type 은 이 class 를 만드는 시점에는 구체적으로 무엇인지를 알 수 없지만, 그 무엇이 있는 것은 안다.
Class 를 만들 때, 아직 확정되지는 않았으나, 그렇지만 존재하여 class 안에서 사용될 어떤 객체의 type, 즉 객체의 class 를 확정하지 않은 채로 객체의 type 으로 사용하는, 그러한 type 을 "generic type" 이라 한다.

ArrayBag⟨E⟩: 생성자, 공개함수[1]

- 생성자
  - public ArrayBag () ;
  - public ArrayBag (int givenCapacity) ;

- 공개함수[1]
  - public int size () ;
  - public boolean isEmpty () ;
    - 배열이 비어 있는지 확인한다.
  - public boolean isFull () ;
    - 배열이 가득 차 있는지 확인한다.
  - public boolean doesContain(E anElement) ;
    - 가방 안에 주어진 anElement 와 동일한 원소가 존재하는지 확인한다.
  - public int frequencyOf (E anElement) ;
    - 가방 안에 주어진 원소 anElement 가 몇 개 있는지 돌려준다.

ArrayBag: 공개함수[2]

- 공개함수[2]
  - public boolean add (E anElement) ;
    - 가방에 anElement 를 추가한다.
  - public boolean remove (E anElement) ;
    - 가방에서 주어진 anElement 를 찾아 삭제한다.
  - public void clear() ;
    - 가방을 초기화 한다.
  - E elementAt (int anOrder)
    - 주어진 순서 anOrder 에 있는 원소를 돌려준다.
    - 동전의 합이나 최대 값이 필요할 경우, 이 함수를 사용하여 모든 동전 객체를 얻어내어 처리한다.
    - (생각해 볼 할 점):
      Bag 에서는 개념적으로 원소의 순서 (order) 가 존재하지 않는다. 그러므로 "elementAt()" 은 존재하지 않는 것이 맞다고 할 수 있다. 다만 실용적 관점에서, 이번 과제와 같이 응용의 필요에 따라 order 를 부여하여 사용할 수도 있다. 사용자가 Bag 의 원소를 하나씩 차례로 얻어 처리할 필요가 있어서 추가한 것이다. 추후, 반복자 (iterator) 개념을 적용하게 되면, 이번 과제는 "elementAt()" 공개함수 없이도 구현이 가능하다.

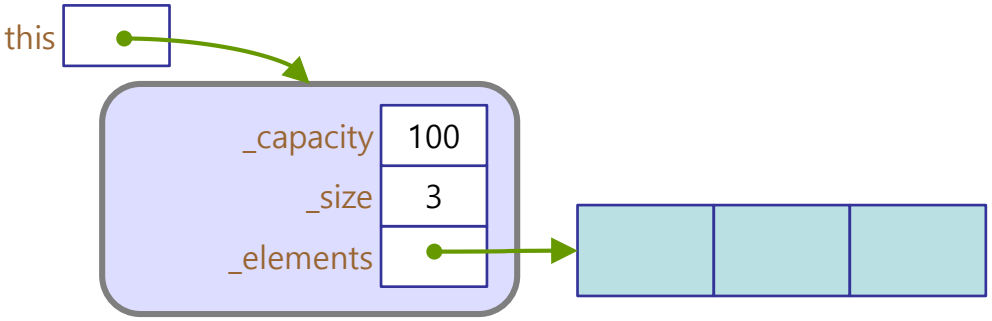
Class “ArrayBag” 의 구현

ArrayBag⟨E⟩:

- 강의 슬라이드의 내용과, 다음의 슬라이드를 참고하여 class "ArrayBag<E>" 을 완성하시오.

ArrayBag 객체의 구조 – 비공개 인스턴스 변수

```
public class ArrayBag<E> {
    // 비공개 인스턴스 변수
    private static final int DEFAULT_CAPACITY = 100 ;
    private int  _capacity ;
    private int  _size ;
    private E    _elements[] ; // ArrayBag의 원소들을 담을 java 배열
```



ArrayBag⟨E⟩: 생성자, 공개함수[1]

- 생성자
  - public ArrayBag ( )
    - _capacity 를 DEFAULT_CAPACITY 로 초기화
    - _elements 를 _capacity 만큼 생성
    - _size를 0 으로 초기화
  - public ArrayBag (int givenCapacity)
    - _capacity 를 givenCapacity 로 초기화
    - _elements 를 _capacity 만큼 생성
    - _size 를 0 으로 초기화

- 공개함수[1]
  - public int size ( )
    - _size 를 반환
  - public boolean isEmpty ()
    - _size 가 0 인지 확인한다.
  - public boolean isFull ()
    - _size 가 _capacity 와 같은지 확인한다.

모든 인스턴스 변수에는 Getter/Setter를 두자!

```
public class  ArrayBag
{
      ......
      // 비공개 인스턴스 변수
      private int          _capacity ;
      private int          _size ;
      private E[]          _elements ; // 원소들을 저장할 배열

      private int capacity() { // Class 내부에서만 사용
            return  this._capacity ;
      }
      private void setCapacity (int newCapacity) { // Class 내부에서만 사용
            this._capacity = newCapacity ;
      }

      pubic int size() { // 공개 함수
            return  this._size ;
      }
      private void setSize (int newSize) { // Class 내부에서만 사용
            this._size = newSize ;
      }

      private E[] elements () {// Class 내부에서만 사용
            return  this._elements ;
      }
      private void setElements (E[] newElements) {// Class 내부에서만 사용
            this._elements = newElements ;
      }
```

ArrayBag: 공개함수[2]

- 공개함수[2]
  - public boolean doesContain (E anElement)
    - 찾으면 확인 할 수 있는 found 변수를 선언 후 false 로 초기화
    - _size 만큼 각 _elements[] 를 확인
    - 만약 같은 값(equals) 이면 found 를 true 로 변경
    - _elements[] 의 값을 모두 확인 하였거나 found 가 true 이면 찾는 것을 종료
    - found 값을 반환
  - public int frequencyOf (E anElement)
    - 개수 (frequency)를 저장할 frequencyCount 를 선언하여 0 으로 초기화
    - _elements[] 의 모든 원소를 하나씩 anElement 의 value 와 비교하여 얻는다
    - 값이 value 와 같으면 frequencyCount 를 1 증가
    - frequencyCount 를 반환

ArrayBag: 공개함수[3]

- 공개함수[3]
  - public boolean add (E anElement)
    - 가득 차 있는 경우(isFull()) false 를 반환
    - _elements[] 배열에 anElement 값을 저장
    - _size 를 증가
  - public boolean remove (E anElement)
    - _elements[] 가 비어있는 경우 false 를 돌려준다
    - _elements[] 를 확인하여 anElement 와 같은 값이 있는 위치를 찾음
    - 위치를 찾지 못하면 false 를 반환
    - 원소의 위치를 찾으면 원소를 삭제 하고, 삭제 된 원소 이후의 모든 배열의 원소들을 앞으로 이동
    - 마지막 위치의 값을 null 로 초기화 한다.
    - 이동이 완료 되면 true 를 돌려준다.
  - Public void clear()
    - _size 를 초기화한다.

ArrayBag: 공개함수[4]

공개함수[4]

- E elementAt (int order)
  - 주어진 순서 order 에 있는 원소를 돌려준다.

```
public E  elementAt (int order) {
    if ( (0 <= order) && (order < this.size()) ) {
        return  this.elements() [order] ;
    }
    else {
        return  null ;
    }
}
```

Class “AppController”

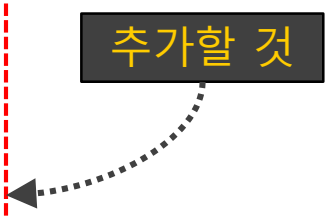
AppController: run() 에 기능 추가 [1]

```
public class AppController {
    // 상수
    ......

    // 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
    ......
```



AppController: run() 에 기능 추가 [2]

```
// Public Method
    public void run() {
        AppView.outputLine("<<< 동전 가방 프로그램을 시작합니다 >>>") ;
        AppView.outputLine("") ;

        int coinBagSize = AppView.inputCapacityOfCoinBag ();
        this.setCoinBag (new ArrayBag<Coin> (coinBagSize) ) ;

        int  menuNumber = AppView.inputMenuNumber() ;
        while ( menuNumber != MENU_END_OF_RUN ) {
            switch (menuNumber) {
            case MENU_ADD:
                this.addCoin () ;
                break ;
            case  MENU_REMOVE:
                this.removeCoin () ;
                break ;
            case MENU_SEARCH:
                this.searchForCoin () ;
                break ;
            case  MENU_FREQUENCY:
                 this.frequencyOfCoin () ;
                break ;
            default:
                this.undefinedMenuNumber (menuNumber) ;
            }
            menuNumber = AppView.inputMenuNumber() ;
        } // End while
        this.showStatistics();
        AppView.outputLine("<<< 동전 가방 프로그램을 종료합니다 >>>") ;
    }
} // End of class  "AppController"
```

추가할 것

AppController: addCoin(), removeCoin()

```
public class AppController {

    ..... // 생략

    // 비공개 함수의 구현
    private void  addCoin () {
        if ( this.coinBag().isFull() ) {
            AppView.outputLine("- 동전 가방이 꽉 차서 동전을 넣을 수 없습니다.") ;
        }
        else {
            int  coinValue = AppView.inputCoinValue () ;
            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.") ;
            }
        }
    }

    private void  removeCoin () {
        int  coinValue = AppView.inputCoinValue () ;
        if ( ! this.coinBag().remove (new Coin(coinValue)) ) {
            AppView.outputLine("- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다.") ;
        }
        else {
            AppView.outputLine ("- 주어진 값을 갖는 동전 하나가 가방에서 정상적으로 삭제되었습니다.") ;
        }
    }
```

AppController: 단계 4 [2]

```
public class AppController {


    private void  searchForCoin () {
        int  coinValue = AppView.inputCoinValue () ;
        if ( this.coinBag().doesContain (new Coin(coinValue)) ) {
            AppView.outputLine ("- 주어진 값을 갖는 동전이 가방 안에 존재합니다.") ;
        }
        else {
            AppView.outputLine("- 주어진 값을 갖는 동전은 가방 안에 존재하지 않습니다.") ;
        }
    }

    private void frequencyOfCoin () {
        int  coinValue = AppView.inputCoinValue () ;
        int  frequency = this.coinBag().frequencyOf (new Coin(coinValue)) ;
        AppView.outputLine ("- 주어진 값을 갖는 동전의 개수는 " + frequency + " 개 입니다.") ;
    }

    private void  undefinedMenuNumber (int menuNumber) {
        AppView.outputLine ("- 선택된 메뉴 번호 " + menuNumber + " 는 잘못된 번호입니다." ) ;
    }
```

AppController: 단계 4 [3]

```
public class AppController {
    ..... // 생략

    // 비공개 함수의 구현
    private void  addCoin () {.....}
    private void  removeCoin () {.....}
    private void  searchForCoin () {.....}
    private void  frequencyOfCoin () {.....}
    private void  undefinedMenuNumber (int aMenuNumber) {.....}

    private int  sumOfCoinValues() {
        int  sum = 0 ;
        for ( int i = 0 ; i < this.coinBag().size() ;  i++ ) {
            sum += this.coinBag().elementAt(i).value() ;
        }
        return  sum ;
    }
    private int  maxCoinValue() {
        int maxValue = 0 ;
        for ( int i = 0 ; i < this.coinBag().size() ;  i++ ) {
            if ( maxValue < this.coinBag().elementAt(i).value() ) {
                maxValue = this.coinBag().elementAt(i).value() ;
            }
        }
        return  maxValue ;
    }

    private void  showStatistics() {
        AppView.outputLine ("가방에 들어 있는 동전의 개수: " + this.coinBag().size()) ;
        AppView.outputLine ("동전 중 가장 큰 값: " + this.maxCoinValue()) ;
        AppView.outputLine ("모든 동전 값의 합: " + this.sumOfCoinValues()) ;
    }
```

AppController: 단계 4 [4]

```
// 공개함수의 구현
public void run() {
        AppView.outputLine ("<<< 동전 가방 프로그램을 시작합니다 >>>") ;

        int  capacityOfCoinBag = AppView.inputCapacityOfCoinBag () ;
        this.setCoinBag ( new ArrayBag<Coin> (capacityOfCoinBag) ) ;

        int  menuNumber = AppView.inputMenuNumber() ;
        while ( menuNumber != MENU_END_OF_RUN ) {
                switch (menuNumber) {
                case MENU_ADD:
                        this.addCoin () ;
                        break ;
                case  MENU_REMOVE:
                        this.removeCoin () ;
                        break ;
                case  MENU_SEARCH:
                        this.searchForCoin () ;
                        break ;
                case MENU_FREQUENCY:
                        this.frequencyOfCoin () ;
                        break ;
                default:
                        this.undefinedMenuNumber (menuNumber) ;
                        break ;
                }
                menuNumber = AppView.inputMenuNumber() ;
        }
        this.showStatistics() ;
        AppView.outputLine ("<<< 동전 가방 프로그램을 종료합니다 >>>") ;
    }
} // End of class  "AppController"
```

<단계 1> 점검

실행해 보자

- 현재까지 완성된 상태로 실행해 보자.
  - 메뉴 선택이 정상적으로 작동하는지 확인하자.
  - run()의 while 구조를 확실하게 이해하자.

<단계 2> 점검

최종적으로 실행해 보자

- ArrayBag 의 각 기능이 정상적으로 작동하는지 확인하자.
- 입력에서 오류 처리가 잘 되는지 확인하자

요약

확인하자

- 다음의 내용을 잘 이해했는지 확인하자.
  - Model-View-Controller
  - ArrayBag 의 구현
  - Getter/Setter의 필요성
  - Getter/Setter 중의 어떤 것은 “private” 이고, 어떤 것은 “public” 인 이유
  - “equals()” 의 개념과 사용법

❑ 생각해 볼 점

- Class 의 인스턴스 변수의 getter/setter 가 , “private” 함에도 불구하고 만들어 사용하는 이유는?
  - 객체의 “캡슐화 (encapsulation)” 와 관련하여 생각해 볼 것.
- Scanner 는 왜 class “AppController” 가 아닌 class “AppView” 에서 선언하는가?
- Generic Type 이 무엇이고, 어떤 장점이 있는가?

⇒ 보고서에 자신의 의견을 작성하시오.

[실습 끝]