가방, 집합

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

Bag / Set / List

비슷하면서도 다른 세 가지

- 비슷한 점
  - 원소를 모아 둔다
  - 자주 사용된다
  - 구현할 때 유사한 내부 구조를 사용한다
    - 배열, 연결 체인
- 다른 점
  - 원소 중복이 가능한지
  - 원소가 순서가 필요한지
  - 그에 따라서, 사용하는 방법이 조금씩 다르다
- 생각할 점
  - 이런 차이점을 어떻게 추상화시켜 표현하면 좋을까?
  - 구현자가 구현은 어떻게 해야 할까?
    - 각각에 대한 효율적인 방법은?

Bag이란?

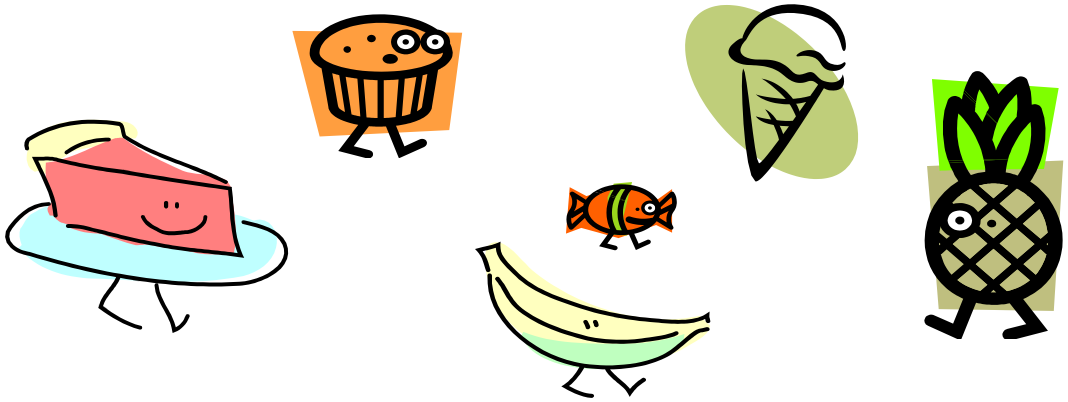
- 원소들을 단순히 모아 놓은 것
  - 원소들 간에 아무 순서가 없다.
  - 중복된 원소들도 있을 수 있다.

- 예: 내가 구매한 기호 식품들

Set 이란?

- 원소들의 집합
  - 원소들 사이에 아무 순서가 없다.
    - 원소들의 순서에 의미를 둘 필요가 없을 경우
  - 중복된 원소는 존재하지 않는다.

- 예: 기호 식품 종류



List 란?

- 원소들이 순서 있게 나열되어 있다
  - 원소들의 순서가 중요한 경우
  - 원소가 중복될 수도 있다

- 예: 내가 좋아하는 기호 식품 구매 순서

가방 (Bag)

Class “Bag”

Bag 객체 추상화: 사용법

- Bag 객체 생성과 소멸
- Bag 상태 알아보기
- Bag 내용 알아보기
- Bag 내용 바꾸기

객체를 정의할 때
맨 먼저 생각할 것은 언제나

"객체의 사용법"

Bag 객체 추상화: 사용법

- Bag 객체 생성과 소멸
  - Bag 객체 생성
- Bag 상태 알아보기
  - Bag 에 들어있는 원소의 개수를 알려주시오
  - Bag 이 비어 있는지 알려주시오
  - Bag 이 가득 찾는지 알려주시오
  - 주어진 원소가 Bag 에 있는지 알려주시오
  - 주어진 원소가 Bag 에 몇 개 있는지 알려주시오
- Bag 내용 알아보기
  - Bag 에서 아무 원소 하나를 얻어내시오
- Bag 내용 바꾸기
  - Bag 에 주어진 원소를 넣으시오
  - Bag 에서 아무 원소 하나를 제거하여 얻어내시오
  - Bag 에서 지정된 원소를 찾아서 있으면 제거하시오
  - Bag 을 비우시오

Bag의 사용법은 공개함수로

- Bag 객체 사용법을 Java로 구체적으로 표현해보자
  - public Bag() {...}
  - public int size () {...}
  - public boolean isEmpty () {...}
  - public boolean isFull () {...}
  - public boolean doesContain (Element anElement) {...}
  - public int frequencyOf (Element anElement) {...}
  - public Element any() {...}
  - public boolean add (Element anElement) {...}
  - public Element removeAny () {...}
  - public boolean remove (Element anElement) {...}
  - public void clear () {...}

Class "Bag"의 초기 형태는 이렇게!

```
public class  Bag
{
      public Bag ( )
      {
      }

      public int size ()
      {
            return 0 ; // 수정해야 함
      }
      public boolean       isEmpty ()
      {
            return true ; // 수정해야 함
      }
      public boolean       isFull ()
      {
            return true ; // 수정해야 함
      }
      public boolean       doesContain (Element anElement)
      {
            return true ; // 수정해야 함
      }
      public int           frequencyOf (Element anElement)
      {
            return 0 ; // 수정해야 함
      }

      public Element any()
      {
            return  null ; // 수정해야 함
      }

      public boolean       add (Element anElement) { return true ; }
      public Element       removeAny () { return null ; }
      public boolean       remove (Element anElement) { return true ; }
      public void          clear () { }
} // End of Class "Bag"
```

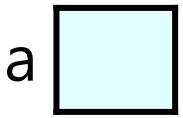
이렇게만 정의해 두어도 사용하는 곳에서 프로그래밍 하는 데는 전혀 지장이 없다. 즉 컴파일 오류가 발생하지 않는다.

Array in Java

- Class “Bag” 구현에 사용 -

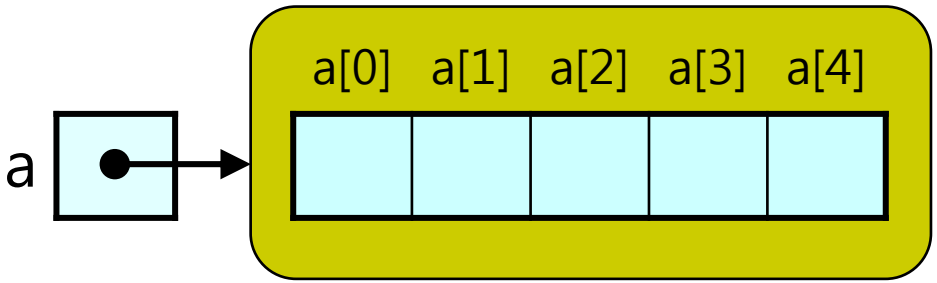
Java 에서의 배열

- 배열은 객체
  - int[] a ;
    - 선언 직후 배열 객체 변수 a는 아무 내용도 가지고 있지 않다



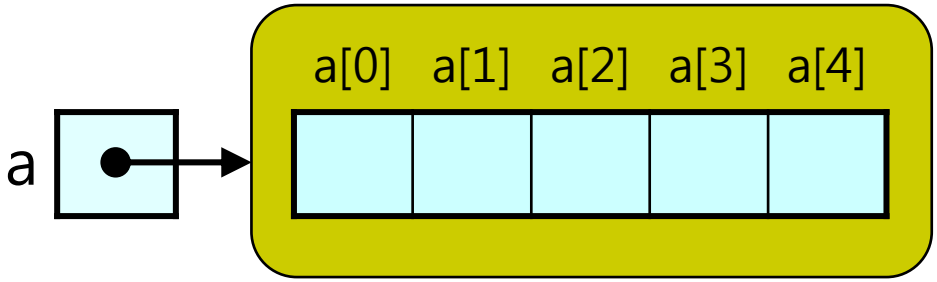
Java 에서의 배열

- 배열은 객체
  - int[] a ;
    - 선언 직후 배열 객체 변수 a는 아무 내용도 가지고 있지 않다
  - a = new int[5] ;



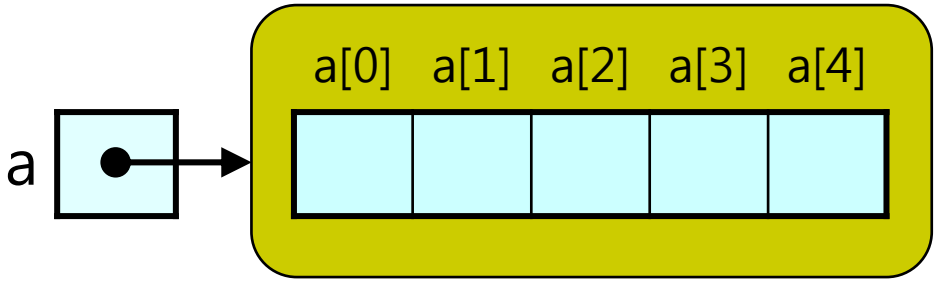
    - new 를 실행한 후에야 배열 객체 변수 a는 배열 객체를 소유하게 된다

Java 에서의 배열 객체 사용법



- 배열의 크기 알아보기
  - int numOfElements = a.**length** ;
- 특정 위치의 값 얻기 (getter)
  - int value = a[2] ;
  - int currentLocation = 2 ;
    int value = a[currentLocation] ;
- 특정 위치의 값 설정하기 (setter)
  - a[3] = 20 ;
  - int thirdValue = 20 ;
    a[3] = thirdValue ;

Java 에서의 배열: 인덱스



- 배열의 유효한 인덱스 범위를 벗어나면?
  - int value = a[6] ;
  - a[-1] = 10 ;
  - int lastValue = a[a.length] ;
- 예외처리
  - ArrayIndexOutOfBoundsException

Class "ArrayBag"

"ArrayBag" as a Bag

- ArrayBag
  - 추상적인 Bag에 대해, 구현하는 방법을 반영하여 지은 이름
  - Java Array를 이용하여 구현

- "Bag"과 "ArrayBag"
  - 이 두 class의 관계는?
  - 사용자는 Bag을 사용할 때, 그 구현에 Array가 사용되었는지 아니면 다른 구현 방법이 사용되었는지 알 필요가 있을까?

ArrayBag의 공개함수

- ArrayBag 객체 사용법을 Java로 구체적으로 표현
  - public ArrayBag () { }
  - public ArrayBag (int givenMaxSize) { }
  - public int size () { }
  - public boolean isEmpty () { }
  - public boolean isFull () { }
  - public boolean doesContain (Element anElement) { }
  - public int frequencyOf (Element anElement) { }
  - public Element any() { }
  - public boolean add (Element anElement) { }
  - public Element removeAny () { }
  - public boolean remove (Element anElement) { }
  - public void clear () { }

Class "ArrayBag"의 초기 형태는 이렇게!

```
public calss  ArrayBag
{
      public  ArrayBag ( )
      {
      }
      public  ArrayBag (int givenMaxSize)
      {
      }

      public int  size ()
      {
            return 0 ; // 수정해야 함
      }
      public boolean  isEmpty ()
      {
            return true ; // 수정해야 함
      }
      public boolean  isFull ()
      {
            return true ; // 수정해야 함
      }
      public boolean  doesContain (Element anElement)
      {
            return true ; // 수정해야 함
      }
      public int          frequencyOf (Element anElement)
      {
            return 0 ; // 수정해야 함
      }

      public Element  any()
      {
            return  null ; // 수정해야 함
      }

      public boolean  add (Element anElement) { return true ; }
      public Element  removeAny () { return null ; }
      public boolean  remove (Element anElement) { return true ; }
      public void       clear () { }
} // End of Class "ArrayBag"
```

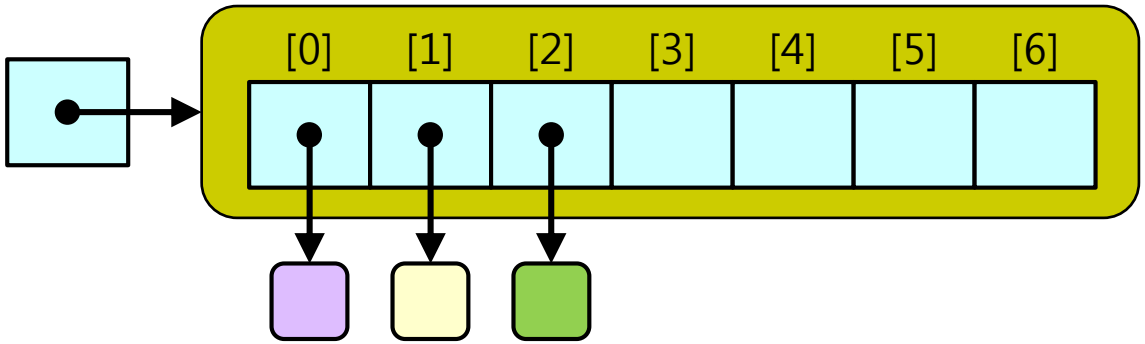
이렇게만 정의해 두어도 사용하는 곳에서 프로그래밍 하는 데는 전혀 지장이 없다. 즉 컴파일 오류가 발생하지 않는다.

Class "ArrayBag"의 구현

ArrayBag 객체에 필요한 속성은?

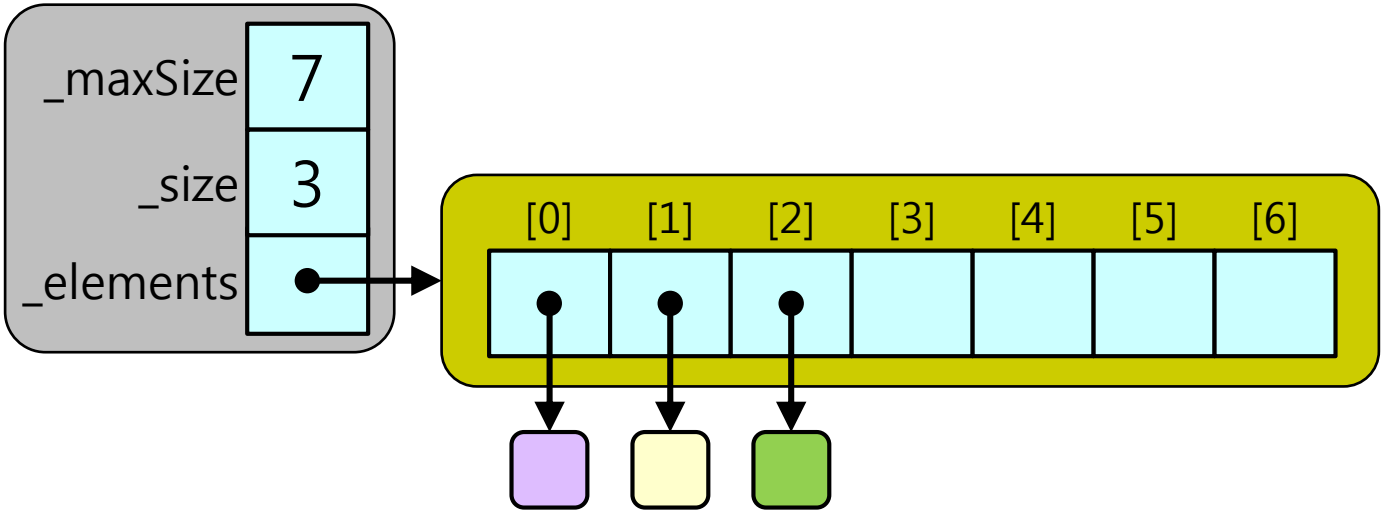
- 객체의 속성은 객체를 구성하는 부품
  - 객체 내부를 설계해야 부품이 결정된다.

- 먼저 객체 내부 구현 방법을 결정하자.
  - Java 배열에 원소를 모아 놓기로 하자.



ArrayBag의 속성

- Bag의 기능을 하려면 어떤 것들이 필요할까?
  - 원소는 배열의 앞 쪽에 빈 칸이 없도록 맨 앞부터 차례로 저장
  - 배열의 최대 크기 (예: 7)
  - 배열이 현재 가지고 있는 원소의 개수 (예: 3)



- int _maxSize ; // 배열의 최대 크기
- int _size ; // 배열이 현재 가지고 있는 원소의 개수
- Element[] _elements ; // 배열 객체(의 소유권)

ArrayBag에서 객체의 속성 표현

```
public class  ArrayBag
{
    // 모든 ArrayBag 객체에서 공통으로 사용
    private static  final  int DEFAULT_MAX_SIZE = 100 ;

    // 비공개 인스턴스 변수
    private int            _maxSize ;
    private int            _size ;
    private Element[]  _elements ; // 원소들을 저장할 배열
```

속성은 객체를 구성하는 부품과 같은 것이다.
부품을 사용자에게 공개할 필요는 없으며, 해서도 안 된다.
그러므로, 반드시 "private"

ArrayBag의 구현

```
public class  ArrayBag
{
    // 모든 ArrayBag 객체에서 공통으로 사용
    private static final int DEFAULT_MAX_SIZE = 100 ;

    // 비공개 인스턴스 변수
    private int _maxSize ;
```

"static" 이 붙어있는 변수는 각각의 객체에 존재하지 않고, class에 대해 단 하나만 존재한다. 그러나 모든 객체에서 인식할 수 있다.

ArrayBag::DEFAULT_MAX_SIZE

즉, "static" 변수는 그 앞에 class 이름을 붙여 사용한다.
(일반 변수는 그 앞에 "this"를 붙인다)

"final" 이 붙어있는 변수는 값을 변경할 수 없다. 결국 상수를 선언하는 것이다.

ArrayBag의 구현: 객체의 생성자

```
public calss  ArrayBag
{
    // 비
    ……
```

생성자는 하나만 존재해야 할까?

```
    // 생성자 1
    public ArrayBag ( )
    {
        this._maxSize = ArrayBag::DEFAULT_MAX_SIZE ;
        this._elements = new  Element[ArrayBag::DEFAULT_MAX_SIZE] ;
        this._size = 0 ;
    }

    // 생성자 2
    public ArrayBag (int givenMaxSize)
    {
        this._maxSize = givenMaxSize ;
        this._elements = new  Element[givenMaxSize] ;
        this._size = 0 ;
    }
```

ArrayBag의 구현: 여러 개의 생성자

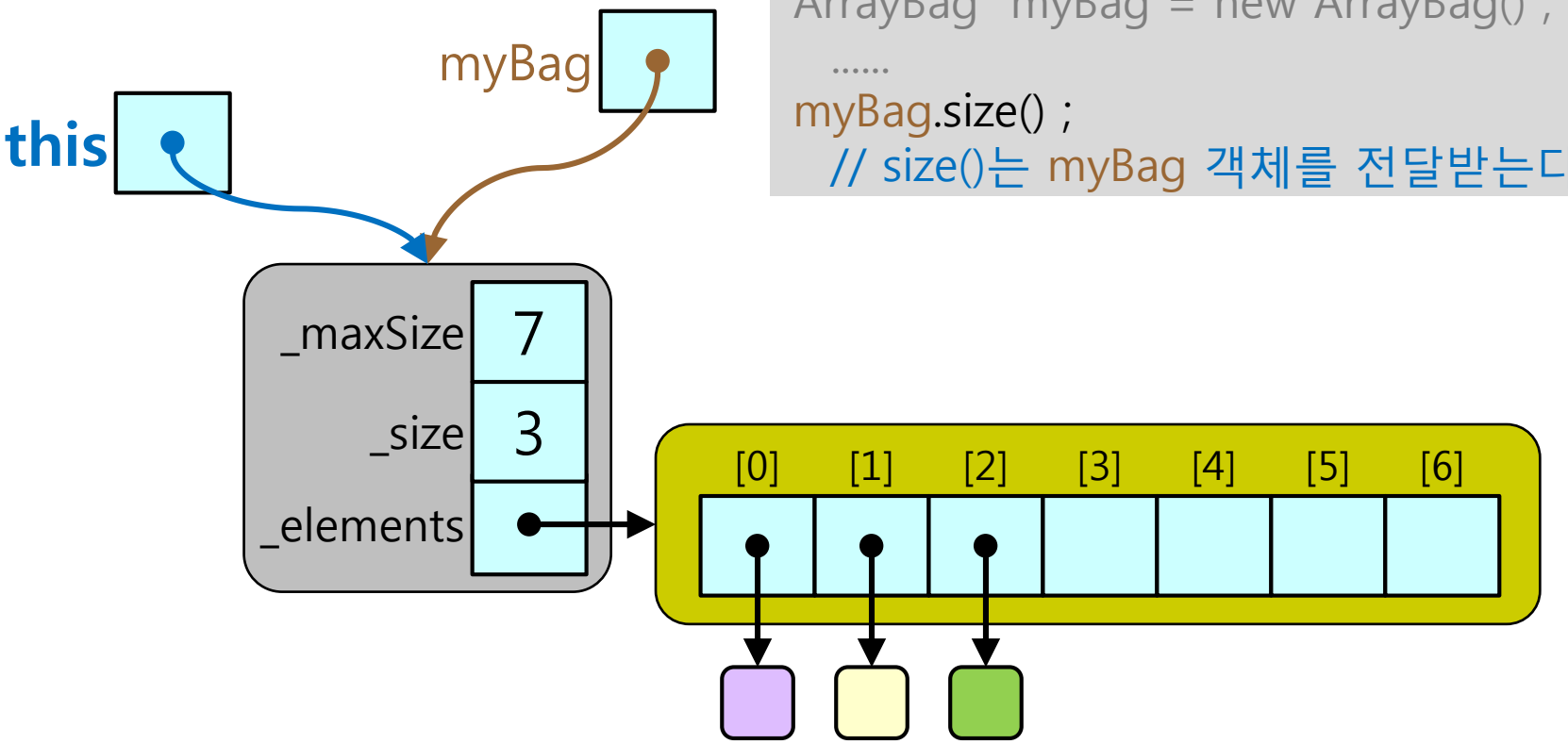
```
public calss  ArrayBag
{
    // 비공개 인스턴스 변수
    ......

    // 생성자 1
    public ArrayBag ( )
    {
        this._maxSize = ArrayBag::DEFAULT_MAX_SIZE ;
        this._elements = new  Element[ArrayBag::DEFAULT_MAX_SIZE] ;
        this._size = 0 ;
    }

    // 생성자 2
    public ArrayBag (int givenMaxSize)
    {
        this._maxSize = givenMaxSize ;
        this._elements = new  Element[givenMaxSize] ;
        this._size = 0 ;
    }
```

여러 개의 생성자들:

객체 생성은 여러가지 방법으로 할 수 있다.
생성자의 이름은 모두 같다.

멤버 함수에서의 "this" 는?

- Class의 멤버 함수가 전달 받은, 그래서 일을 해주어야 할 객체를 가리킨다.
  - 생성자 안에서는 생성된 객체를 소유




```
ArrayBag  myBag = new ArrayBag() ;
   ......
myBag.size() ;
  // size()는 myBag 객체를 전달받는다
```

ArrayBag의 구현: 상태 알아보기

```
public calss  ArrayBag
{
    // 비공개 인스턴스 변수
    ……

    // 생성자
    ……

    // 상태 알아보기
    public int  size ()    // Bag 에 들어있는 원소의 개수를 알려준다
    {
        return  this._size ;
    }

    public boolean  isEmpty ()     // Bag 이 비어 있는지 알려준다
    {
        return  (this._size == 0) ;
    }

    public boolean  isFull ()        // Bag 이 가득 차 있는지 알려준다
    {
        return  (this._size == this._maxSize) ;
    }
```

ArrayBag: 상태 알아보기

```
// 상태 알아보기
.....
public boolean  doesContain (Element anElement)
{   // 주어진 원소가 Bag 에 있는지 알려준다
    boolean  found = false ;
    for ( int i = 0 ;  i < this._size  && ! found ; i++ ) {
        if ( this._elements[i]. equals (anElement) ) {
            found = true ;
        }
    }
    return found ;
}

public int  frequencyOf (Element anElement)
{   // 주어진 원소가 Bag 에 몇 개 있는지 알려준다
    int  frequencyCount = 0 ;
    for ( int i = 0 ;  i < this._size ; i++ ) {
        if ( (this._elements[i].equals(anElement) ) {
            frequencyCount ++;
        }
    }
    return  frequencyCount ;
}
```

"equals()"에 대해

```
// 상태 알아보기
.....
public boolean  doesContain (Element anElement)
{   // 주어진 원소가 Bag 에 있는지 알려준다
    boolean  found = false ;
    for ( int i = 0 ;  i < this. size  && ! found ; i++ ) {
        if ( this._elements[i]. equals (anElement) ) {
            found = true ;
        }
    }
    return fou
}

public int  fre
{   // 주어진
    int  freque
    for ( int i =
        if ( (this
            freq
        }
    }
    return  fre
}
```

<두 객체가 "같다"는 의미는?>

Java에서 "equals()" 는 모든 객체에 대해 사용가능.
(예) x.equals(y) // 객체 x에게 x가 y와 같은지를 검사시킨다.

단순한 일반 변수에서의 "같다(==)" 와는 달리, 단순치 않은 객체에서의 "equals()" 는 class 마다 서로 다른 의미가 될 수 밖에 없다.

각 class는 자신의 "equals()" 의 의미를 가지고 있어야 한다.
즉 class 안에 자신만의 "equals()" 를 구현해 놓아야 한다.

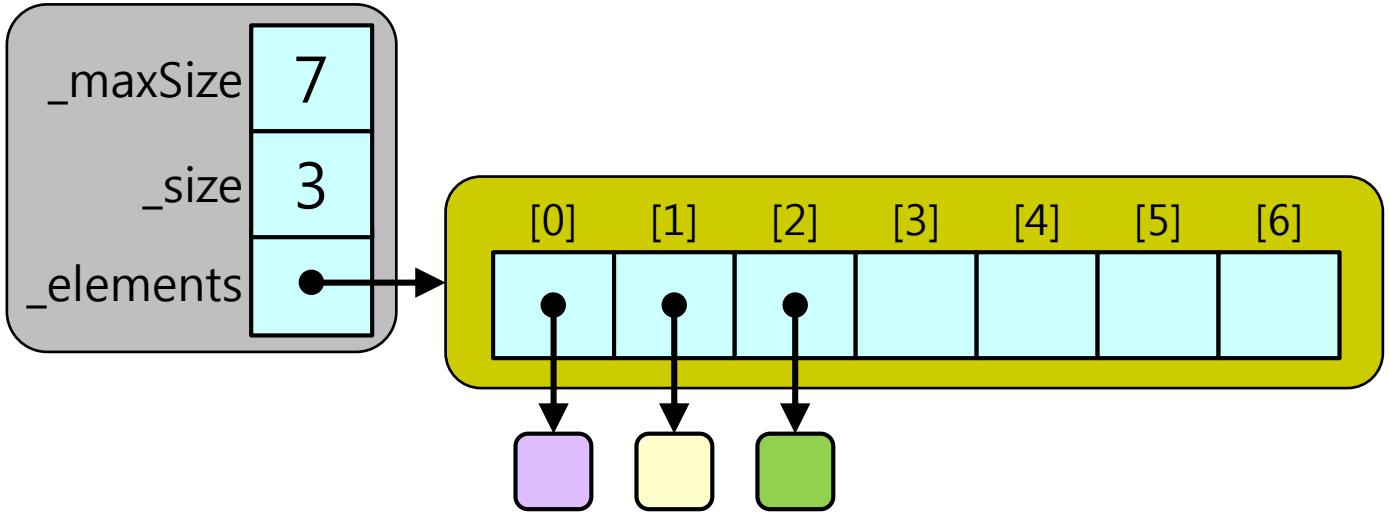
ArrayBag: 내용 알아보기

```
// 내용 알아보기
public Element any ()     // Bag 에서 아무 원소 하나를 얻어낸다.
{
    if ( this.isEmpty() ) {
        return  null ;
    }
    else {
        // 아무 원소나 가능하다
        return  this._elements[0] ; // 맨 앞 원소를 돌려준다
    }
}
```

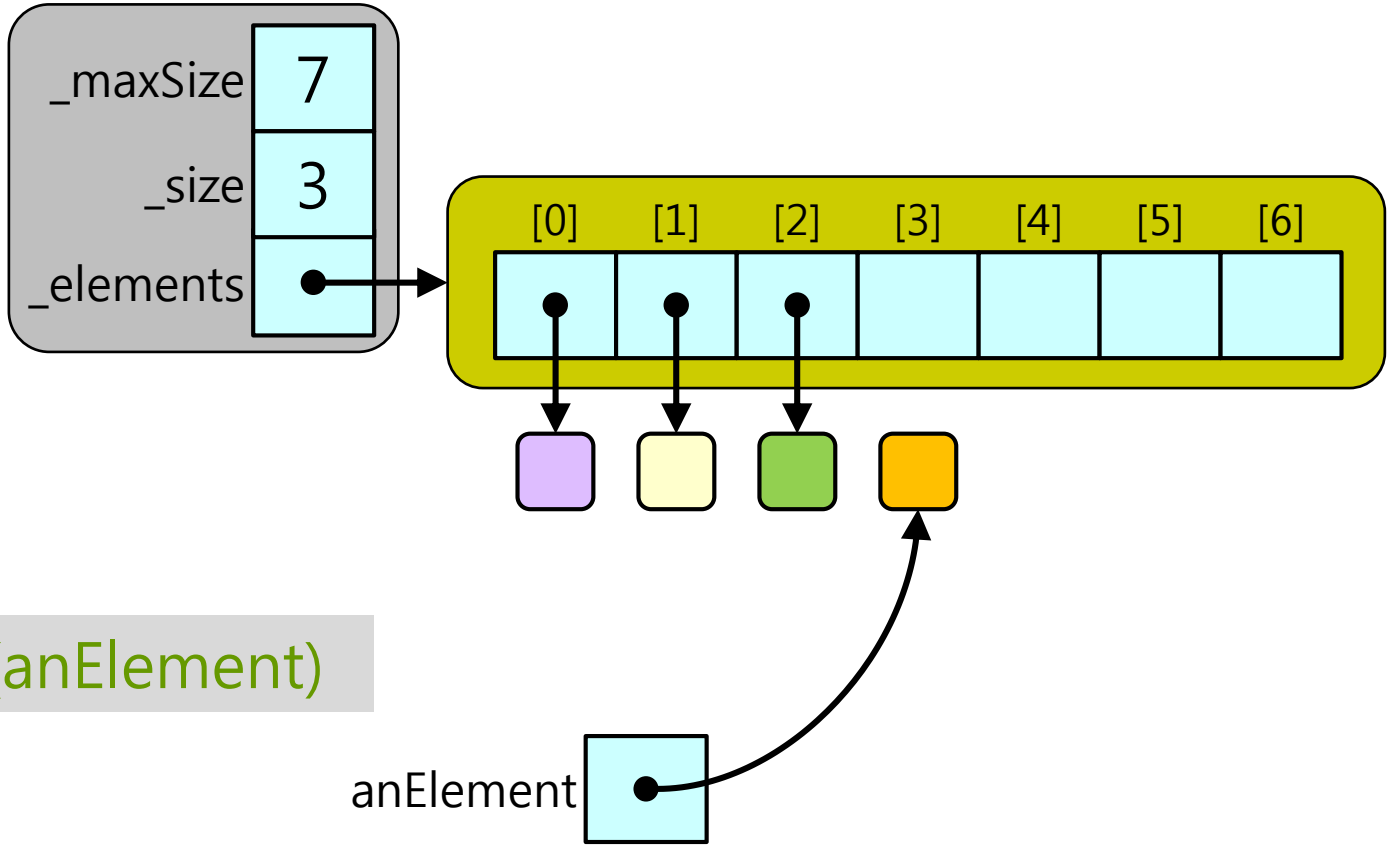
ArrayBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{   // Bag 에 주어진 원소를 넣는다

    if ( this.isFull() ) {
        // 가방이 가득 찼으므로 넣을 수가 없다
        return  false ;
    }
    else {
        // 빈 여유 공간이 있으므로 넣는다
        // 원소의 순서가 중요하지 않으므로 아무 곳에 넣어도 된다
        // 단, 맨 앞부터 꽉 차 있는 상태는 유지해야 한다
        // 가장 편한 곳은 배열의 맨 마지막 원소의 다음 칸
        this,_elements[this._size] = anElement ;
        this._size++ ;
        return  true ;
    }
}
```

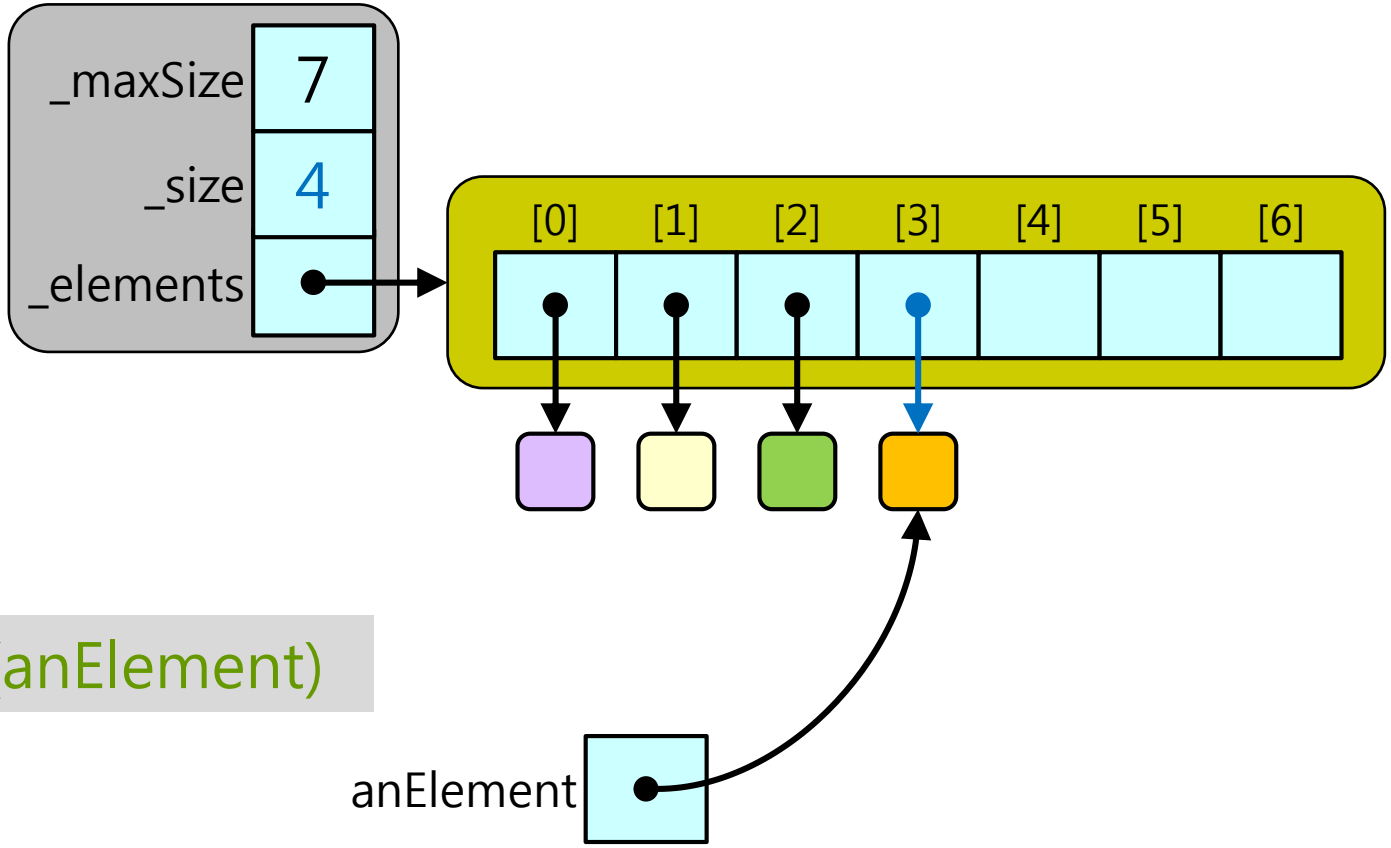
ArrayBag: add()



ArrayBag: add()



ArrayBag: add()

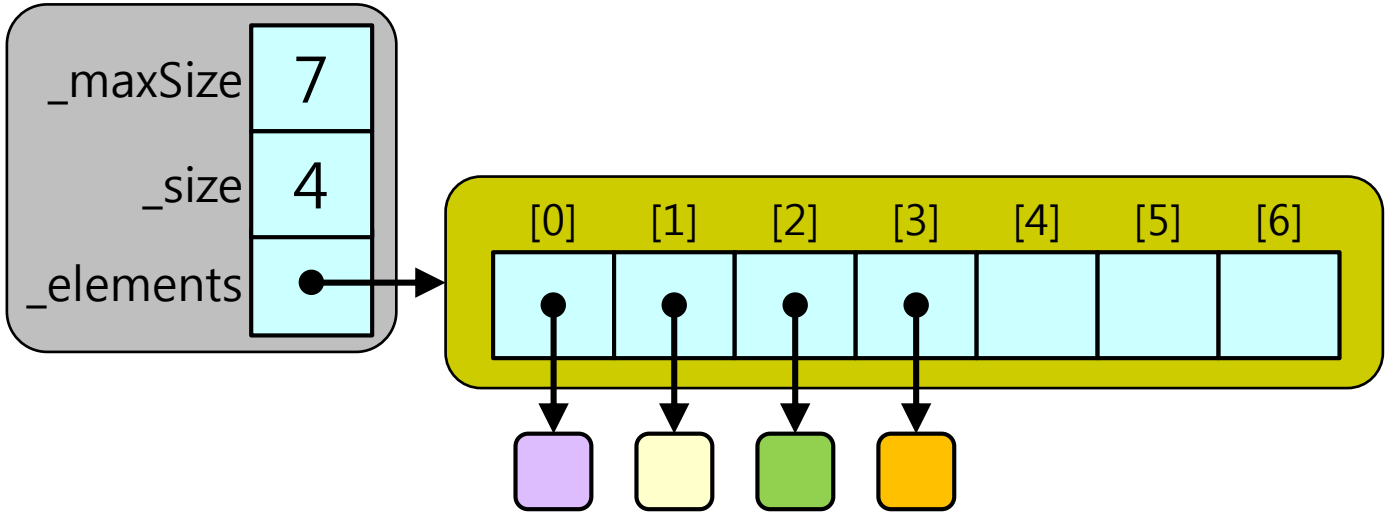


ArrayBag: removeAny()

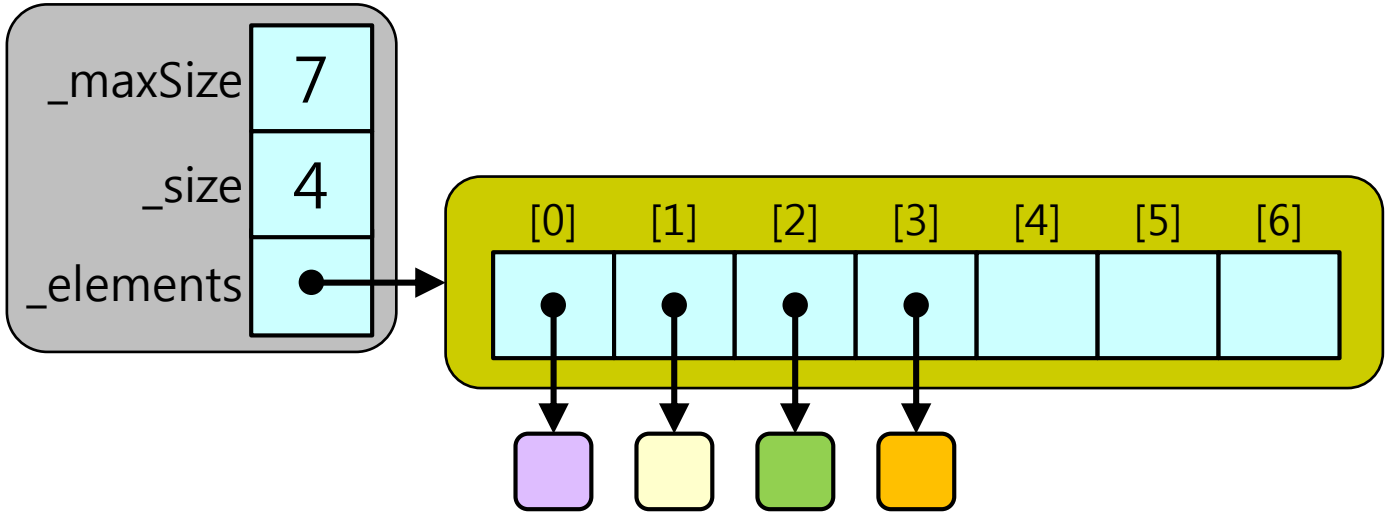
```
// 내용 바꾸기
.....
public Element removeAny ()
{   // Bag 에서 아무 원소 하나를 제거하여 얻어낸다
    if ( this.isEmpty() ) {
        // 가방이 비어 있으므로 돌려 줄 원소가 없다
        return  null ;
    }
    else {
        // 원소가 존재하므로 아무거나 하나 제거하여 얻는다
        // 맨 마지막 원소를 제거하여 얻기로 한다
        Element anyElement = this._elements[this._size-1] ;
        this._elements[this._size-1] = null ; // 필요한 이유는?
        this._size-- ;
        return  anyElement ;
    }
}
```

맨 앞 원소를 제거한다면 어떻게 해야 할까?
어느 편이 효율적일까?

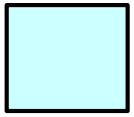
ArrayBag: removeAny()



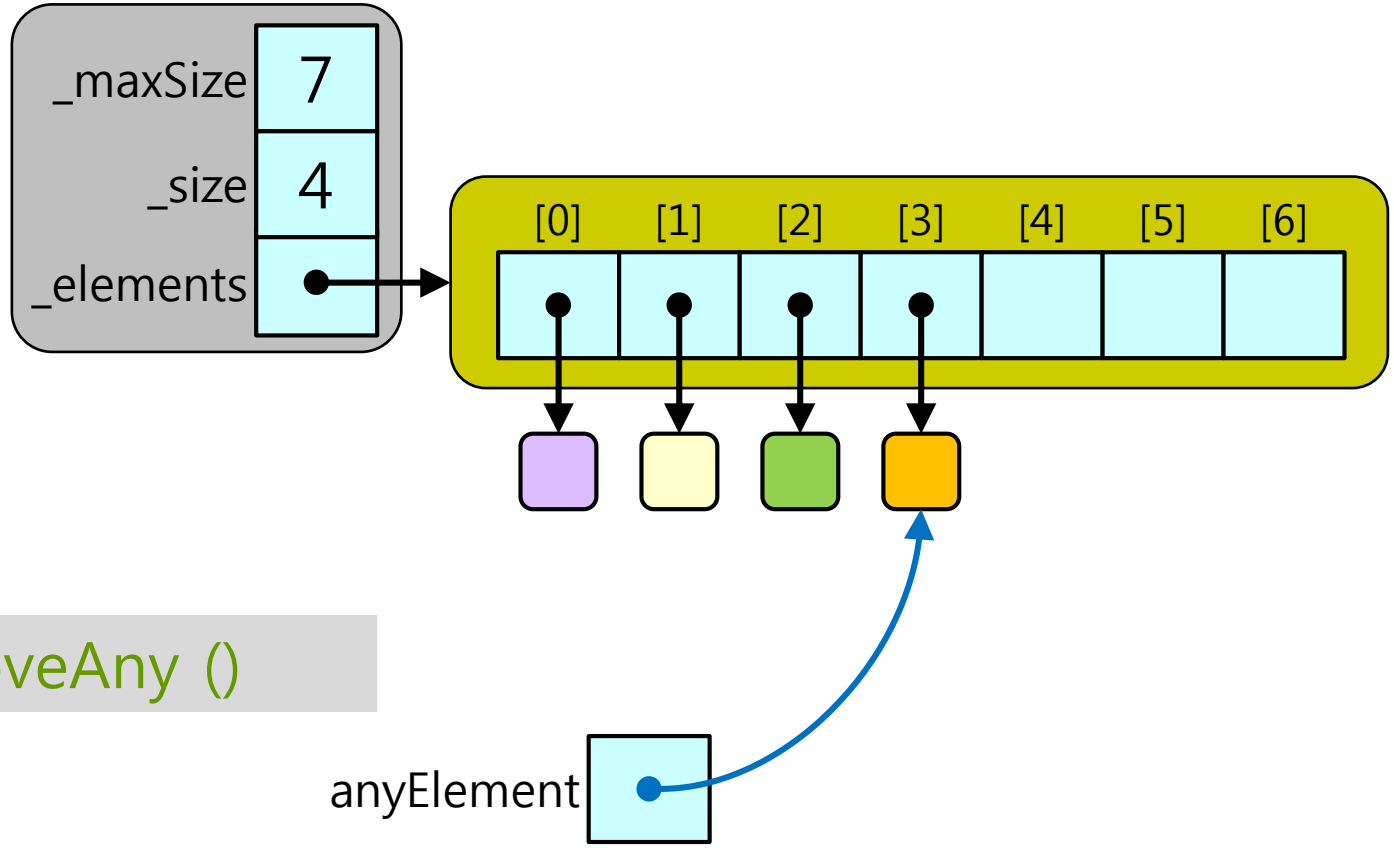
ArrayBag: removeAny()



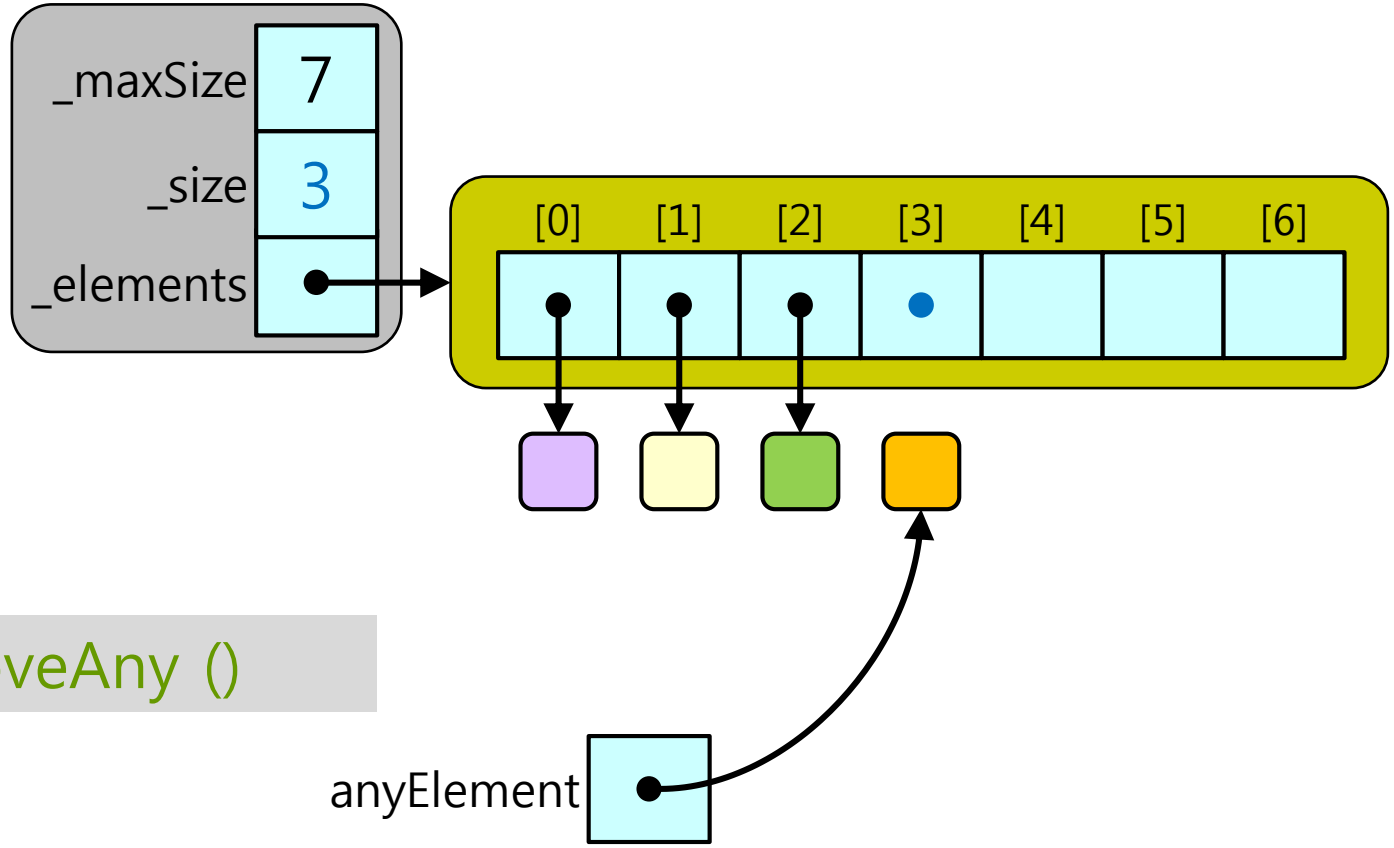

```
removeAny ()
```

anyElement 

ArrayBag: removeAny()



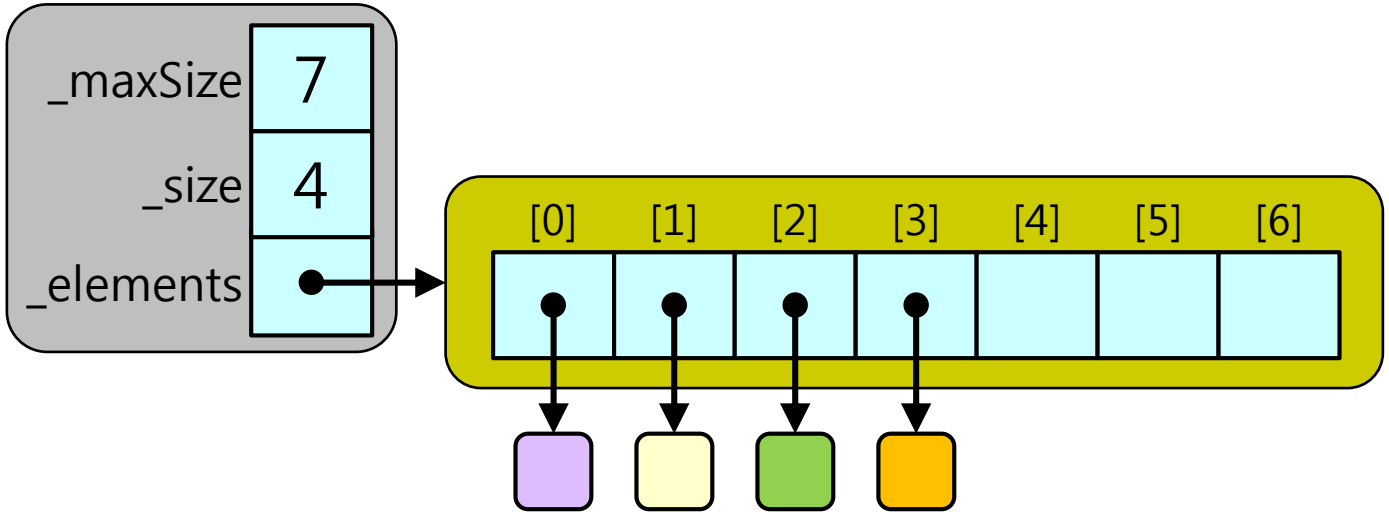
ArrayBag: removeAny()



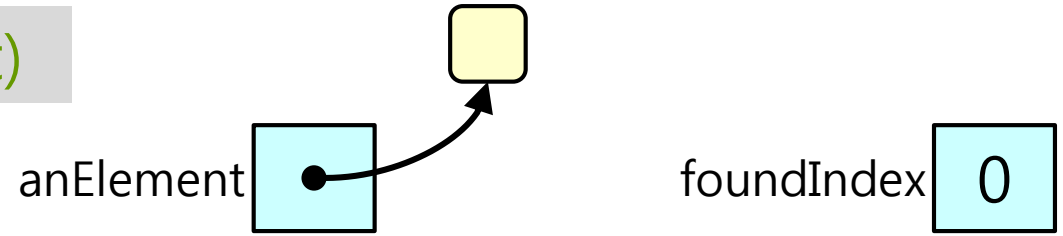
ArrayBag: remove()

```
.....
public boolean  remove (Element anElement)
{    // Bag 에서 지정된 원소를 찾아서 있으면 제거한다

     int  foundIndex ;
     boolean  found = false ;
     // 단계 1: 주어진 원소의 위치를 찾는다
     for ( foundIndex = 0 ; foundIndex < this._size && ! found ; foundIndex++ ) {
          if (this._elements[foundIndex].equals(anElement) {
               found = true ;
          }
     }
     // 단계 2: 삭제된 원소 이후의 모든 원소를 앞쪽으로 한 칸씩 이동시킨다.
     if ( ! found ) {
          return false ;
     }
     else {
          for ( int i = foundIndex ; i < this._size-1 ; i++ ) {
               this._elements[i] = this._elements[i+1] ;
          }
          this._elements[this._size-1] = null ; // 더 이상 의미가 없는 소유권은 null로!
          this._size-- ;
          return true ;
     }
}
```

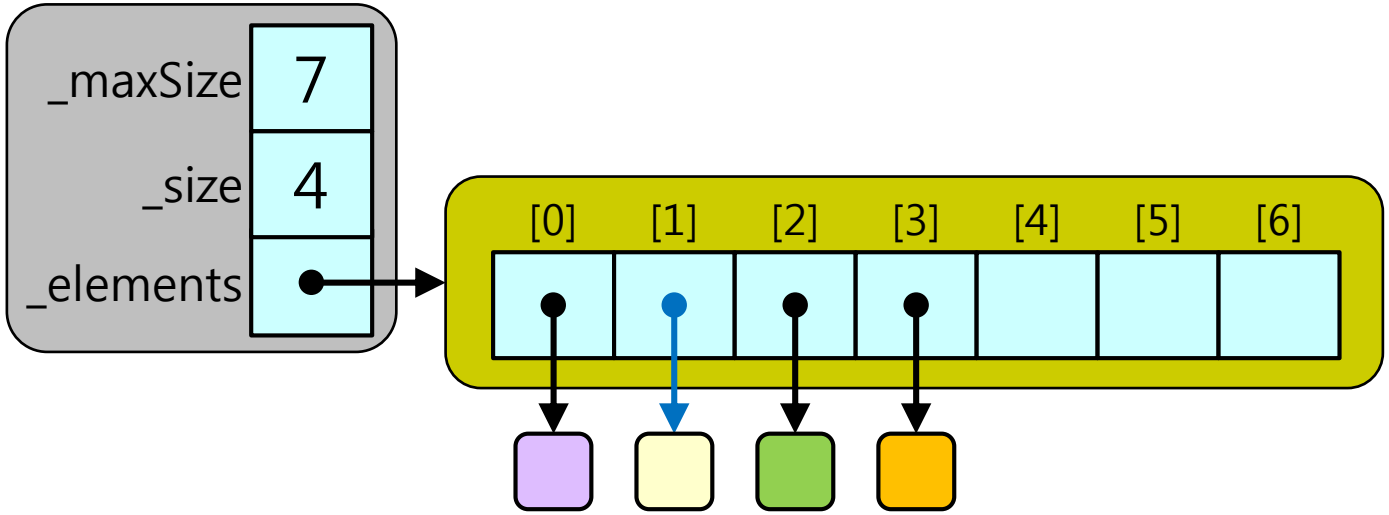
ArrayBag: remove()



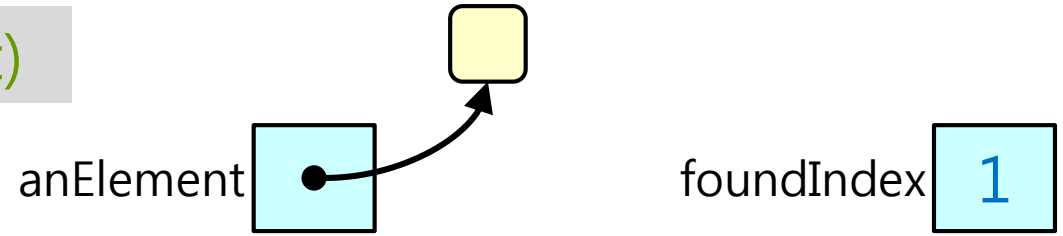
remove (anElement)



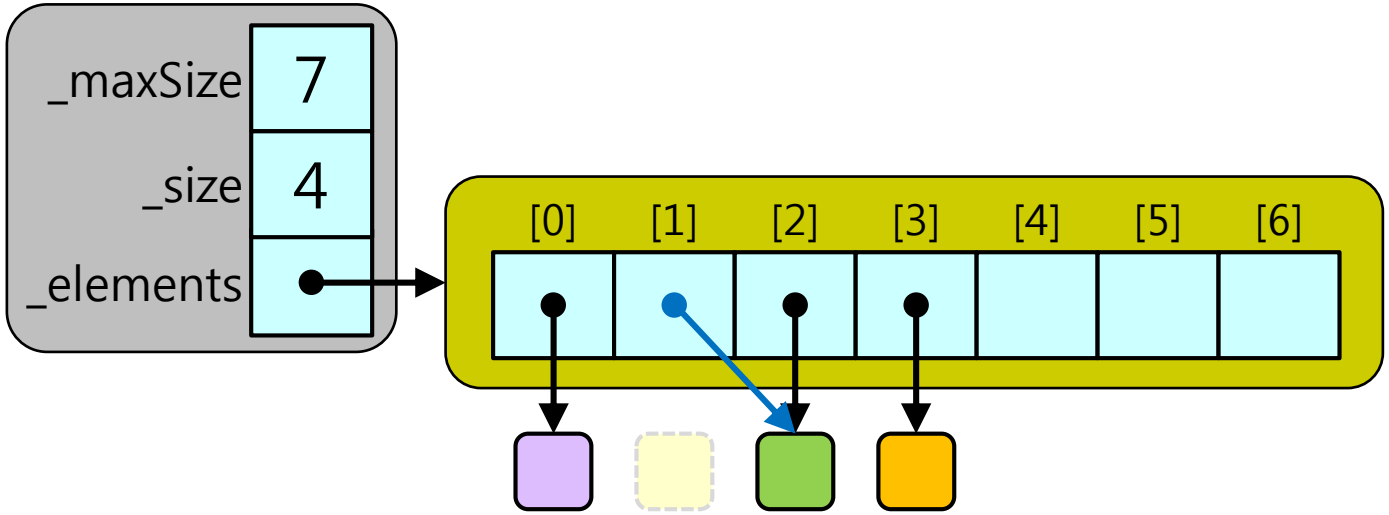
ArrayBag: remove()



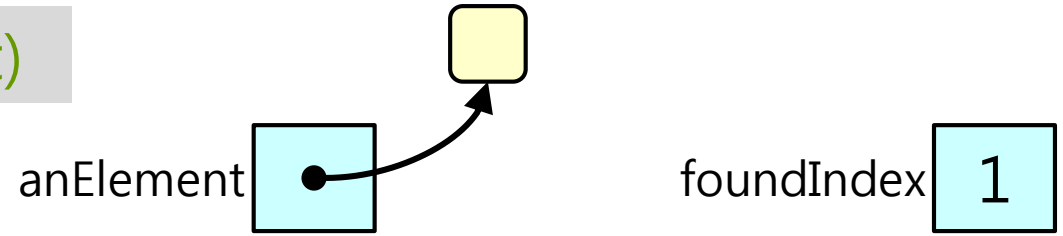
remove (anElement)



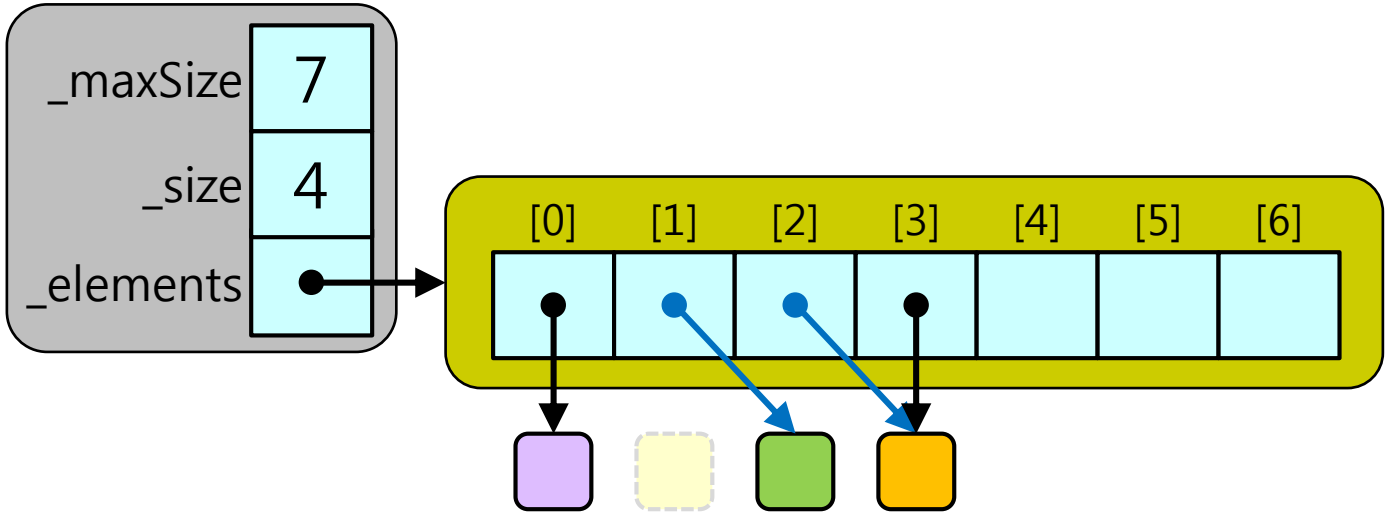
ArrayBag: remove()



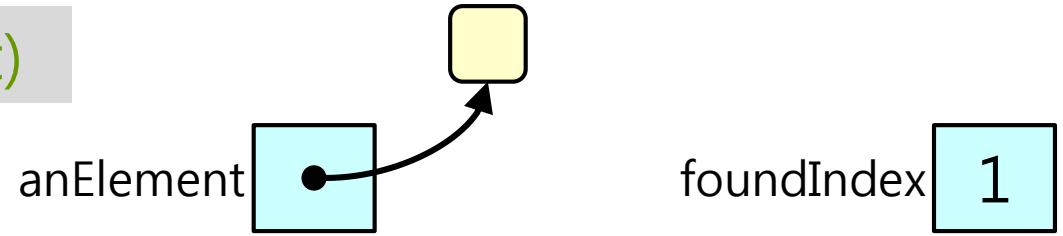
remove (anElement)



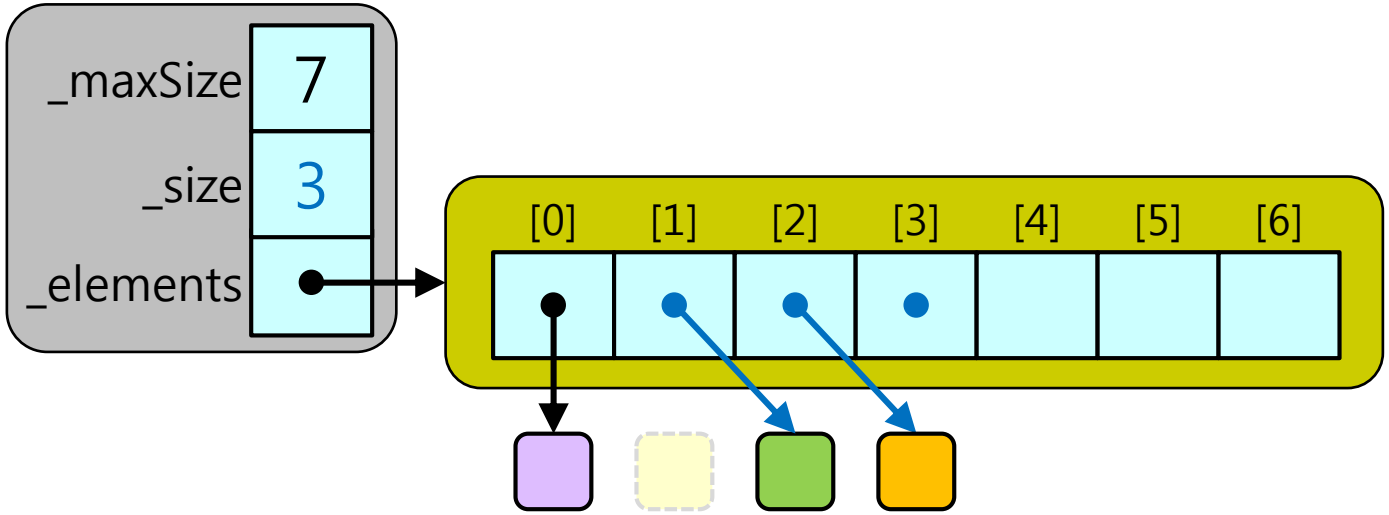
ArrayBag: remove()



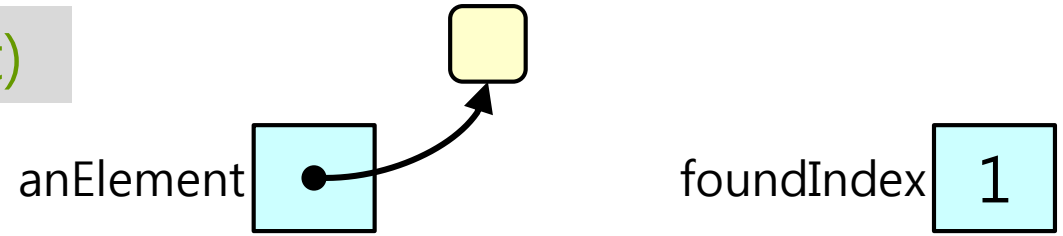
remove (anElement)



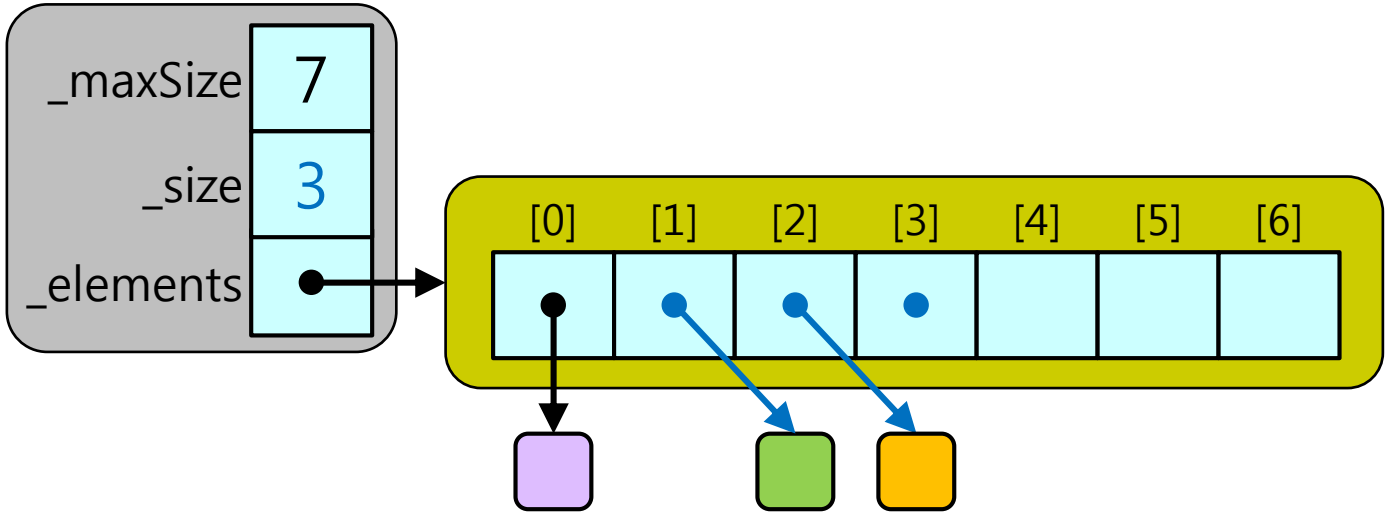
ArrayBag: remove()



remove (anElement)



ArrayBag: remove()



ArrayBag: clear()

```
// 내용 바꾸기
......

public void  clear()     // Bag 을 비운다
{
    for ( int i = 0 ; i < this._size ; i++ ) {
    {
        this._elements[i] = null ;
    }
    this._size = 0 ;
}
```

Class "Element" for coins

Class "Element"

```
public class  Element
{
    private static final int     DEFAULT_VALUE = 0 ;

    // 비공개 인스턴스 변수
    private int             _value ;

    // 생성자
    public  Element()
    {
        this._value = Element::DEFAULT_VALUE ;
    }

    public  Element (int givenValue)
    {
        this._value = givenValue ;
    }
```

Class "Element"에서의 "equals()"

```
public class  Element
{
    ......
    public void  setValue (int aValue)
    {
        this._value = aValue ;
    }
    public int    value ()
    {
        return  this._value ;
    }

    public boolean  equals (Element anElement)
    {
        return  ( this._value == anElement.value() ) ;
    }

} // End of class "Element"
```

순차 검색 (Sequential Search)

Sequential Search: Version 1

```
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;
    for ( int i = 0 ;  i < this._size  &&  ! found ; i++ ) {
        if ( this._elements[i].equals(anElement) ) {
            found = true ;
        }
    }
    return found ;
}
```

- 시간은 얼마나 걸릴까?
  - this._size의 값을 n 이라고 하면…

Sequential Search: Version 2

```
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;
    for ( int i = 0 ;  i < this._size  && ! found ; i++ ) {
        if ( this._elements[i].equals(anElement) ) {
            return true ; // found = true ;
        }
    }
    return false ; // return found ;
}
```

Sequential Search: Version 2

```
public boolean  doesContain (Element anElement)
{

    for ( int i = 0 ;  i < this._size ; i++ ) {
        if ( this._elements[i].equals(anElement) ) {
            return true ;
        }
    }
    return false ;
}
```

- Version 1과 Version 2
  - 시간적 성능의 차이는?
  - 각각의 장단점은?
  - 어느 코드가 더 이해하기 좋을까?
  - 결과가 true이든 false이든 (이 예제의 경우 return 하기 전에) 공통적으로 해야 할 일이 더 있다면?

Fast Enumeration for Array in Java

```
public boolean  doesContain (Element anElement)
{

    for ( Element currentElement in this._elements ) {
        if ( currentElement.equals(anElement) ) {
            return true ;
        }
    }
    return false ;
}
```

- 어떤 경우에 Fast Enumeration을 사용하면 좋을까?
  - 위의 코드는 doesContain()을 정상적으로 실행하는가?
  - 시간적 성능의 차이는?
  - 장단점은?
  - 어느 코드가 더 이해하기 좋을까?

실습: Coin Collection

실습: Coin Collection

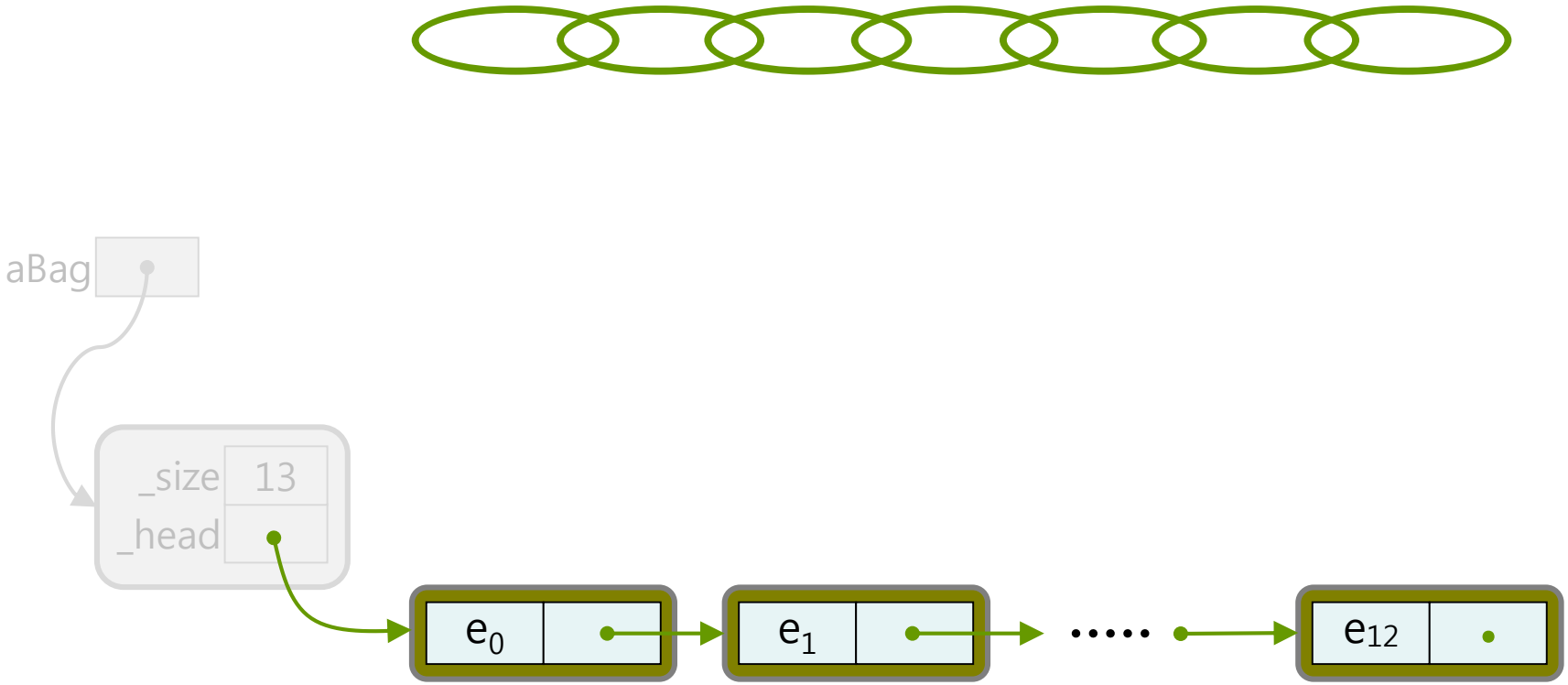
- 입력 메뉴에서 선택된 일을 한다
  - coin 삽입
  - 임의의 coin 삭제
  - 주어진 coin 삭제
  - 주어진 coin이 Collector 안에 있는지?
  - 주어진 coin이 Collector 안에 몇 개 있는지?
  - 코인의 종류별 개수
  - coin collector 비우기
- ArrayBag coinCollector ;

End of "ArrayBag"

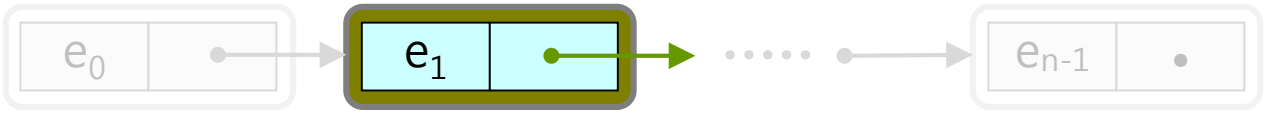
연결 체인 (Linked Chain)

- Class “Bag” 구현에 사용 -

Linked Chain



Class “Node”



Node 의 공개함수

- Node 객체 사용법을 Java로 구체적으로 표현
  - public Node () { }
  - public Element element () { }
  - public Node next () { }
  - public void setElement (Element anElement) { }
  - public void setNext (Node aNode) { }

Class "Node"의 구현: 멤버 변수

```
public class   Node
{
    // 비공개 인스턴스 변수
    private    Element  _element ;
    private    Node     _next ;
```

Class "Node"의 구현: 생성자

```
public class  Node
{
    // 비공개 멤버 변수
    ......

    // 생성자
    public Node ( )
    {
        this._element = null ;
        this._next = null ;
    }
```

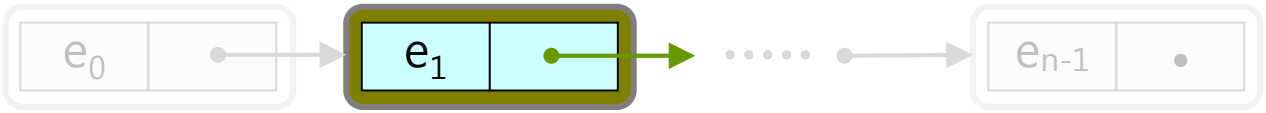
Class "Node"의 구현: Getters

```
public class  Node
{
    // 비공개 멤버 변수
    ……

    // Getters
    public Element  element ( )
    {
        return  this._element ;
    }

    public Node  next ( )
    {
        return  this._next ;
    }
```

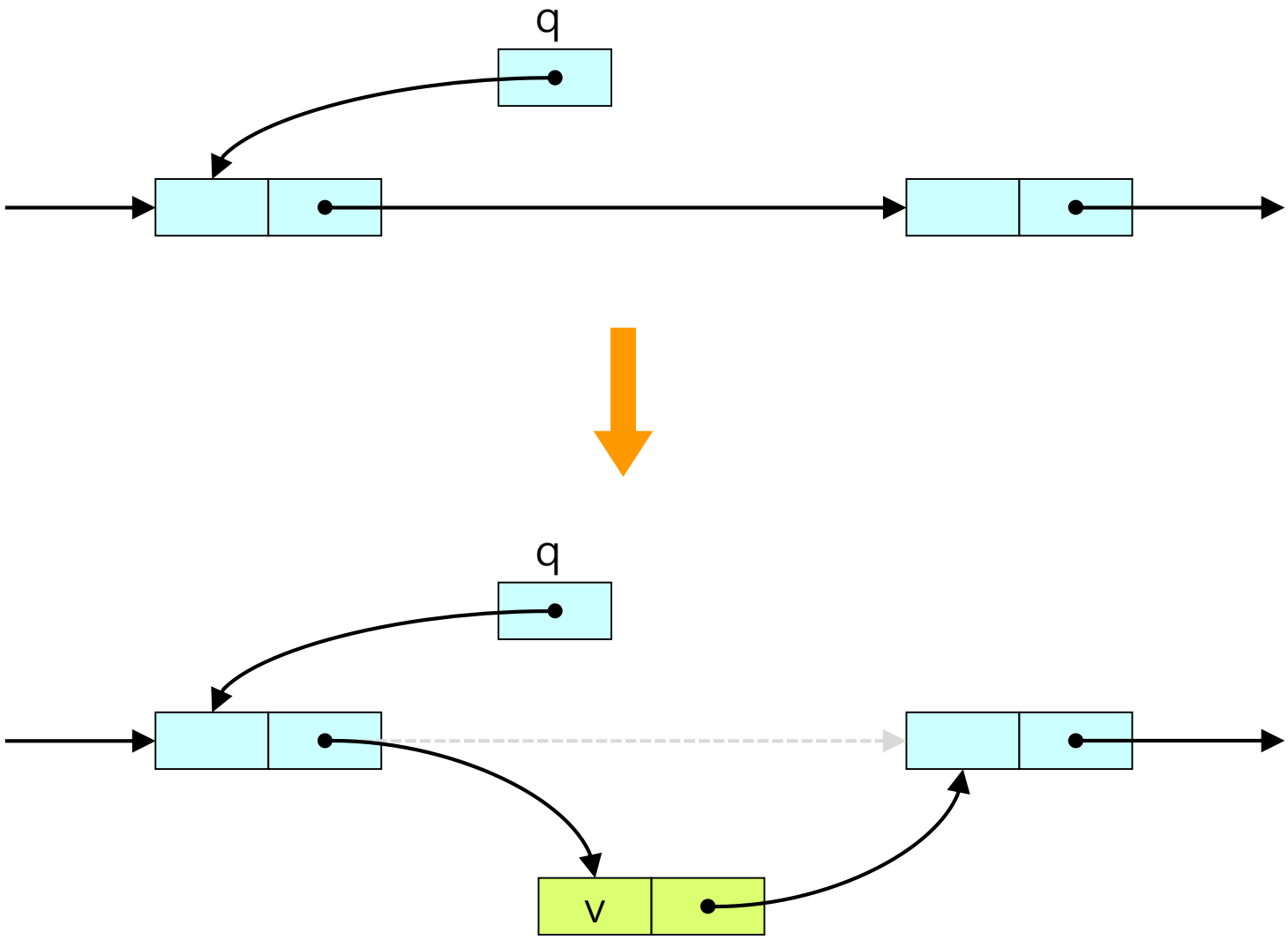
Class "Node"의 구현: Setters

```
public calss  Node
{
    // 비공개 멤버 변수
    ......

    // Getters
    ......

    // Setters
    public void  setElement (Element anElement)
    {
        this._element = anElement ;
    }

    public void  setNext (Node aNode)
    {
        this._next =  aNode ;
    }
```

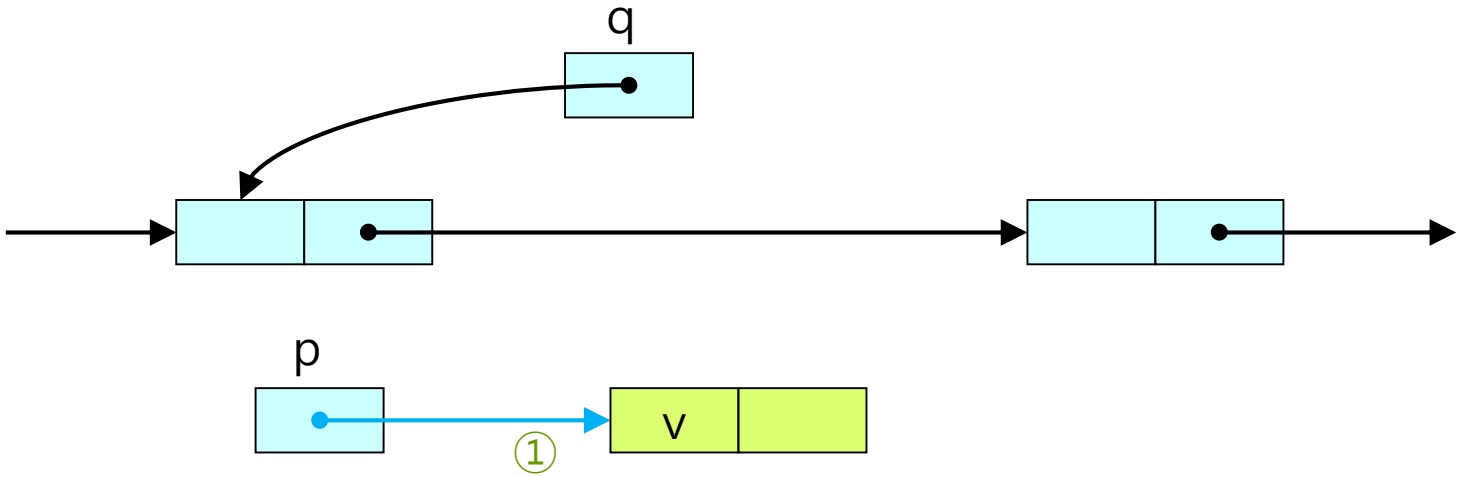
연결 체인에서의 삽입과 삭제



노드의 삽입 [1]



노드의 삽입 [2]



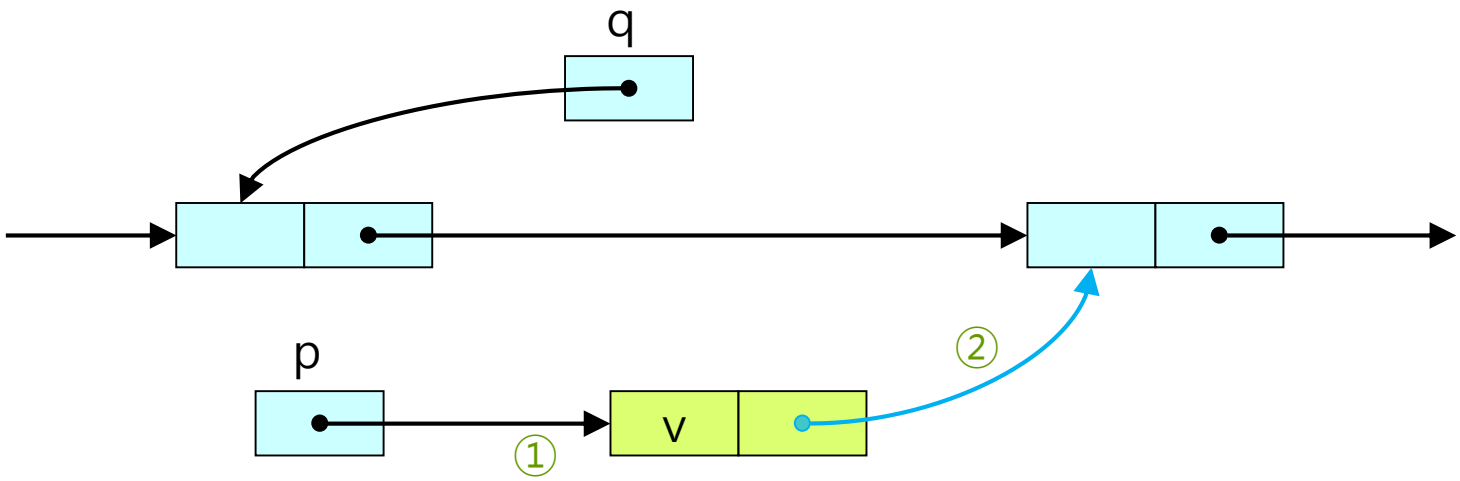
① p = new Node() ;
p.setElement(v) ;

② p.setNext(q.next()) ;

③ q.setNext(p) ;

```
public class   Node
{
    // 비공개 멤버 변수
    private   Element   _element ;
    private   Node      _next ;
```

노드의 삽입 [3]



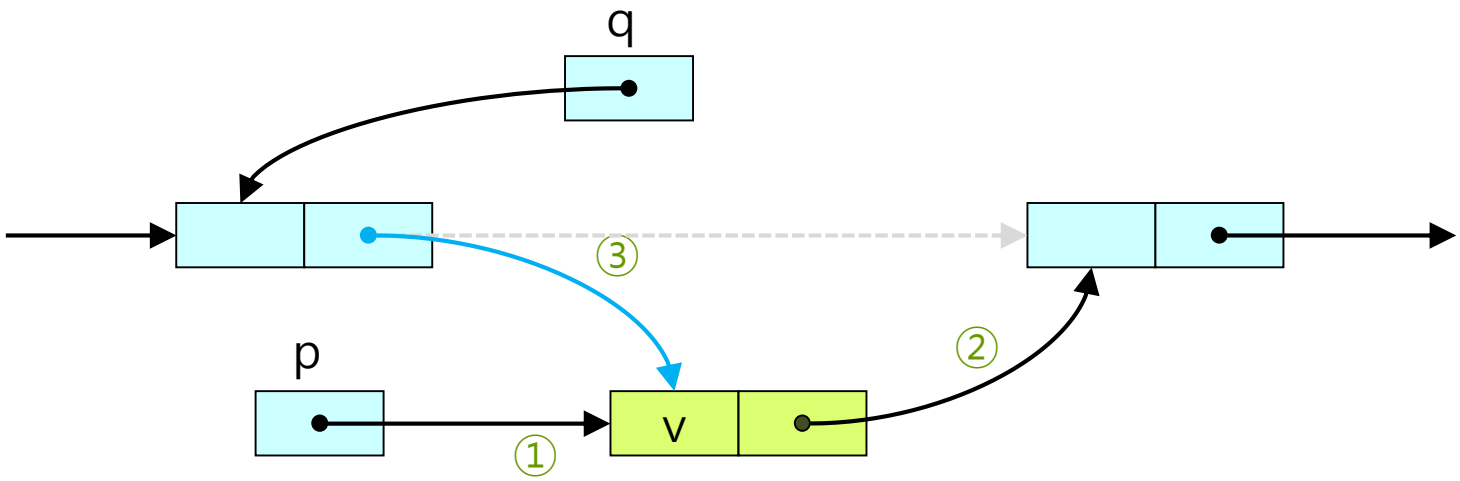
① p = new Node() ;
p.setElement(v) ;

② p.setNext(q.next()) ;

③ q.setNext(p) ;

```
public class   Node
{
    // 비공개 멤버 변수
    private   Element   _element ;
    private   Node      _next ;
```

노드의 삽입 [4]



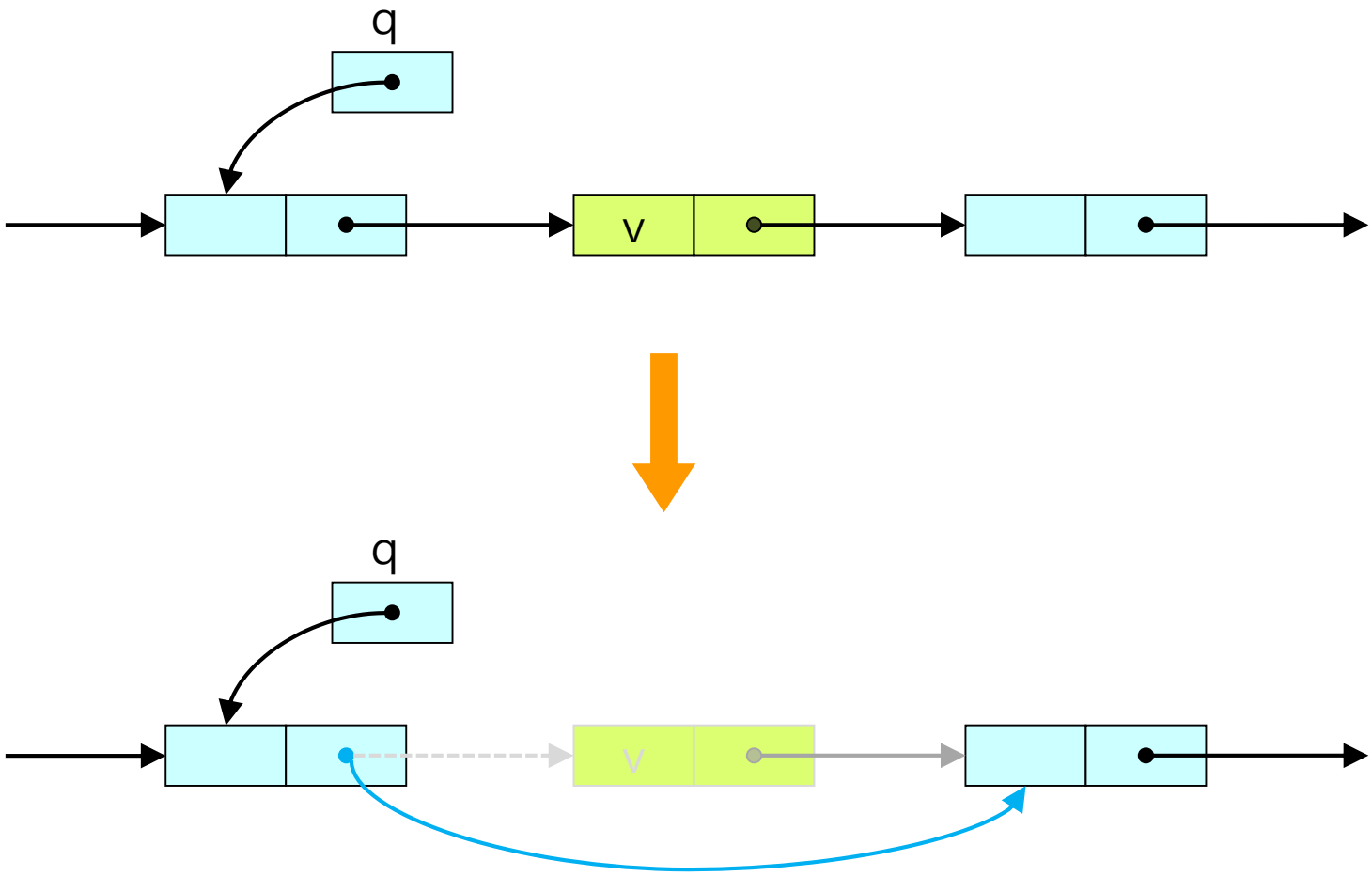
① p = new Node ;
p.setElement(v) ;

② p.setNext(q.next()) ;

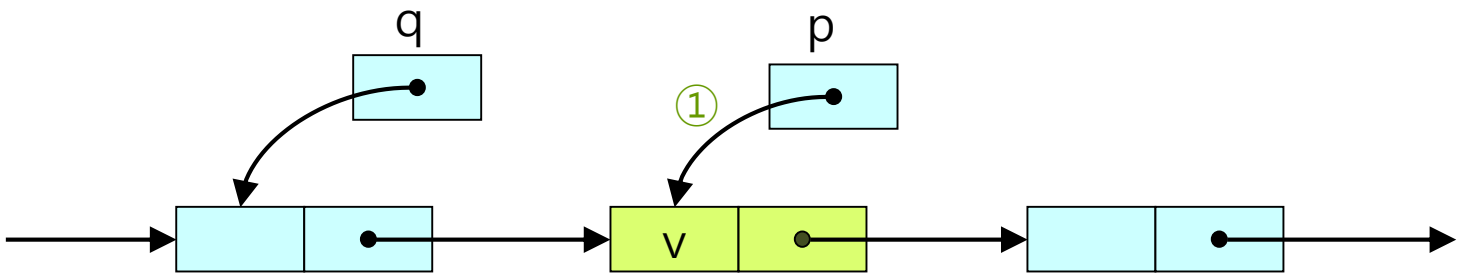
③ q.setNext(p) ;

```
public class Node
{
    // 비공개 멤버 변수
    private Element _element ;
    private Node    _next ;
```

노드의 삭제 [1]



노드의 삭제 [2]

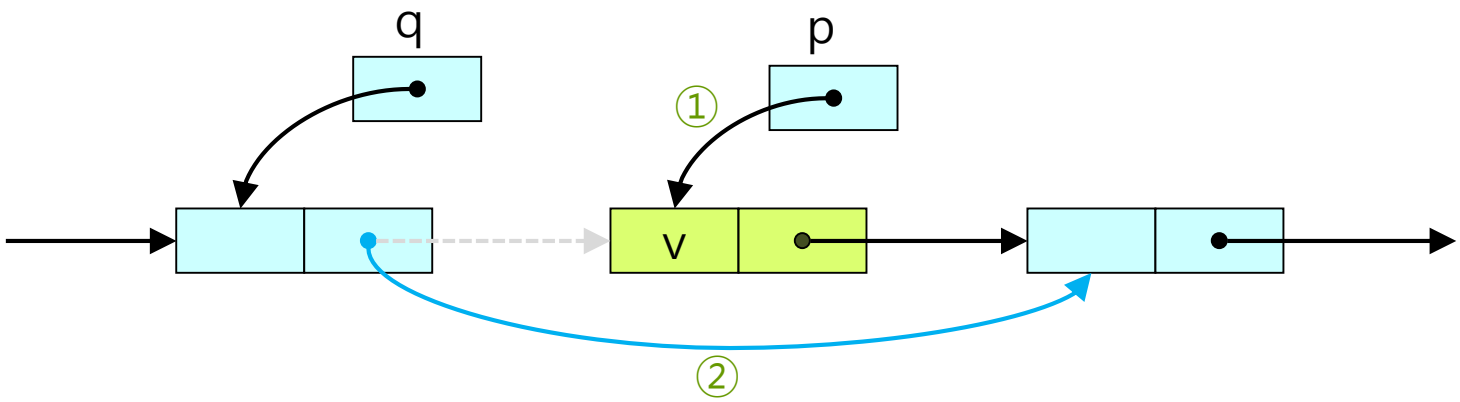


① p = q.next() ;

② q.setNext(p.next()) ;

// (또는) q.setNext((q.next()).next()) ;

노드의 삭제 [3]



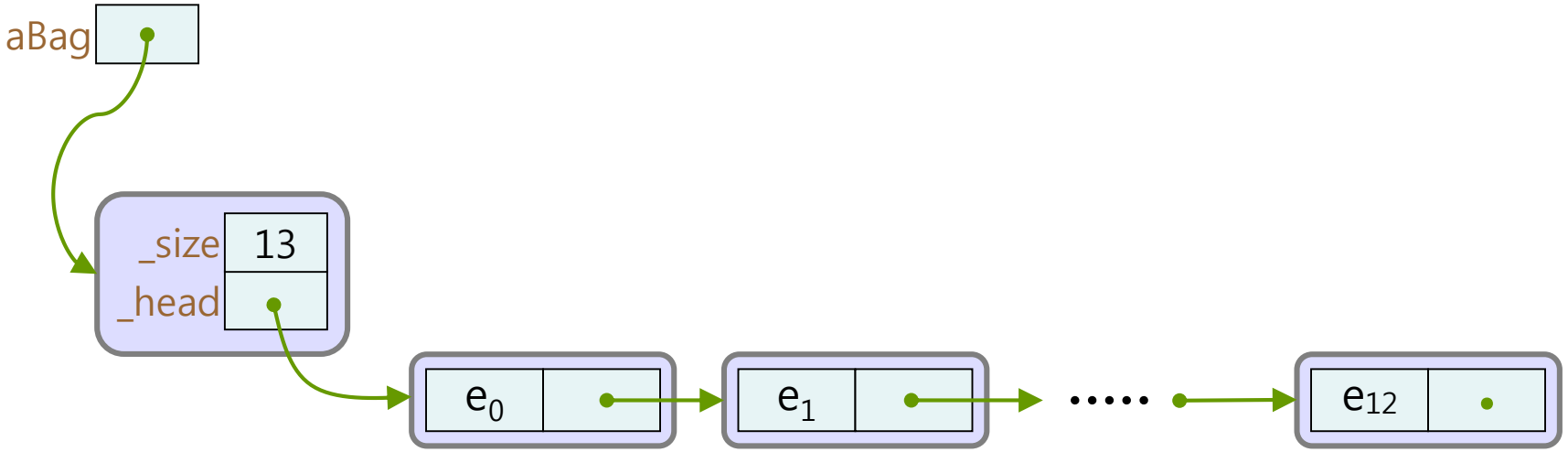
① p = q.next() ;

② q.setNext(p.next()) ;
// (또는) q.setNext((q.next()).next()) ;

Class “LinkedBag”

"LinkedBag" as a Bag

- 추상적인 Bag을 Linked Chain를 이용하여 구현
  - LinkedBag aBag = new LinkedBag() ;
    …… // doing some operations for aBag



LinkedBag의 공개함수

- LinkedBag 객체 사용법을 Java로 구체적으로 표현
  - public LinkedBag() { }
  - public int size () { }
  - public boolean isEmpty () { }
  - public boolean isFull () { }
  - public boolean doesContain (Element anElement) { }
  - public int frequencyOf (Element anElement) { }
  - public Element any () { }
  - public boolean add (Element anElement) { }
  - public Element removeAny () { }
  - public boolean remove (Element anElement) { }
  - public void clear () { }

LinkedBag의 구현

```
public class  LinkedBag
{
    // 비공개 인스턴스 변수
    private int        _size ;
    private Node    _head ; // LinkedBag의 원소들을 담을 Linked Chain
```

LinkedBag의 구현

```
public class  LinkedBag
{
    // 비공개 인스턴스 변수
    ......

    // 생성자
    public LinkedBag ( )
    {
        this._size = 0 ;
        this._head = null ;
    }
```

LinkedBag: 상태 알아보기

```
public class  LinkedBag
{
    // 비공개 인스턴스 변수
    ……

    // 생성자
    ……

    // 상태 알아보기
    public int  size ()
    {
        return  this._size ;
    }

    public boolean  isEmpty ()
    {
        return  (this._size == 0) ; // 또는 return (this._head == null) ;
    }
```

LinkedBag: 상태 알아보기

```
public calss  LinkedBag
{
    // 비공개 인스턴스 변수
    ……

    // 생성자
    ……

    // 상태 알아보기
    ……

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

LinkedBag: doesContain()

```
// 상태 알아보기
……
public boolean  doesContain (Element anElement)
{

    Node currentNode = this._head ;
    while ( currentNode != null ) {
        if ( currentNode.element().equals(anElement) ) {
            return true ;
        }
        currentNode = currentNode.next() ;
    }
    return false ;
}
```

LinkedBag: frequencyOf()

```
// 상태 알아보기
……
public int  frequencyOf (Element anElement)
{
    int  frequencyCount = 0 ;
    Node currentNode = this._head ;
    while ( currentNode != null ) {
        if ( currentNode.element().equals(anElement) ) {
            frequencyCount ++;
        }
        currentNode = currentNode.next() ;
    }
    return  frequencyCount ;
}
```

LinkedBag: any()

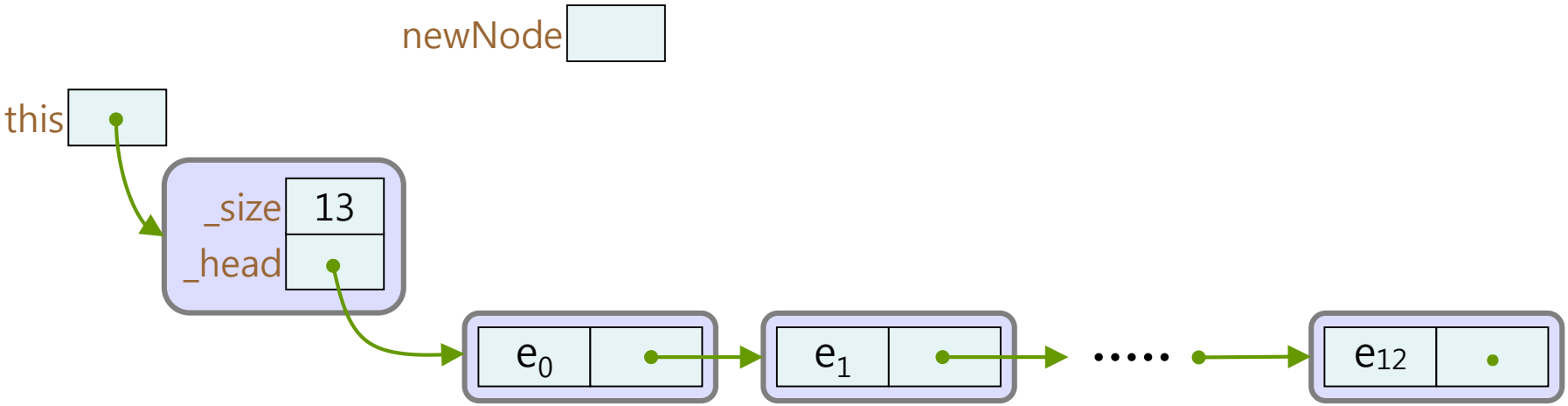
```
// 내용 알아보기
public Element any ()
{
    if ( this.isEmpty() ) {
        return   null ;
    }
    else {
        return   this._head.element() ;
    }
}
```

LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```

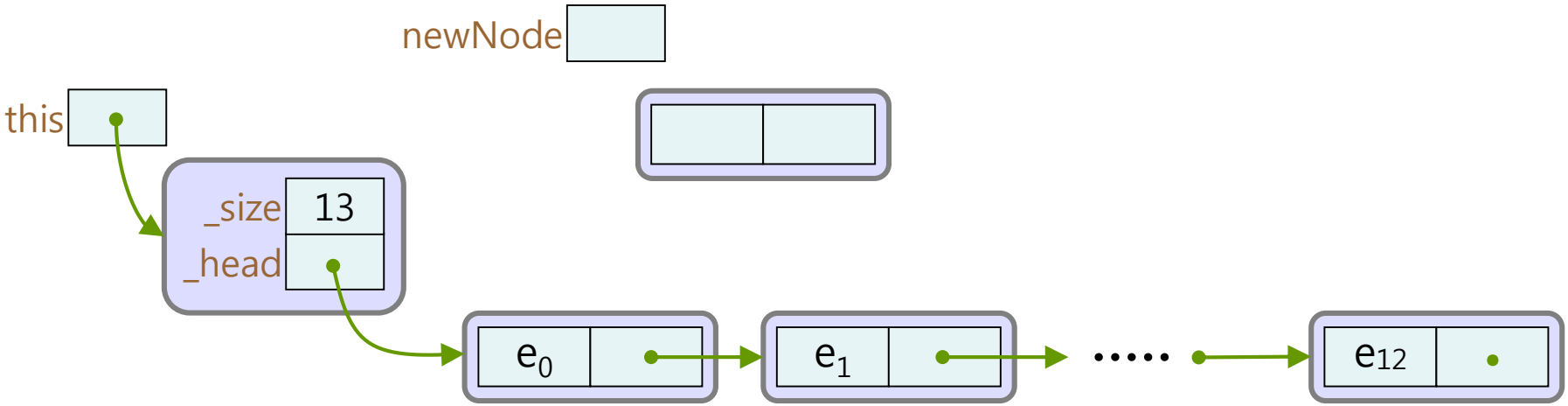
LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```



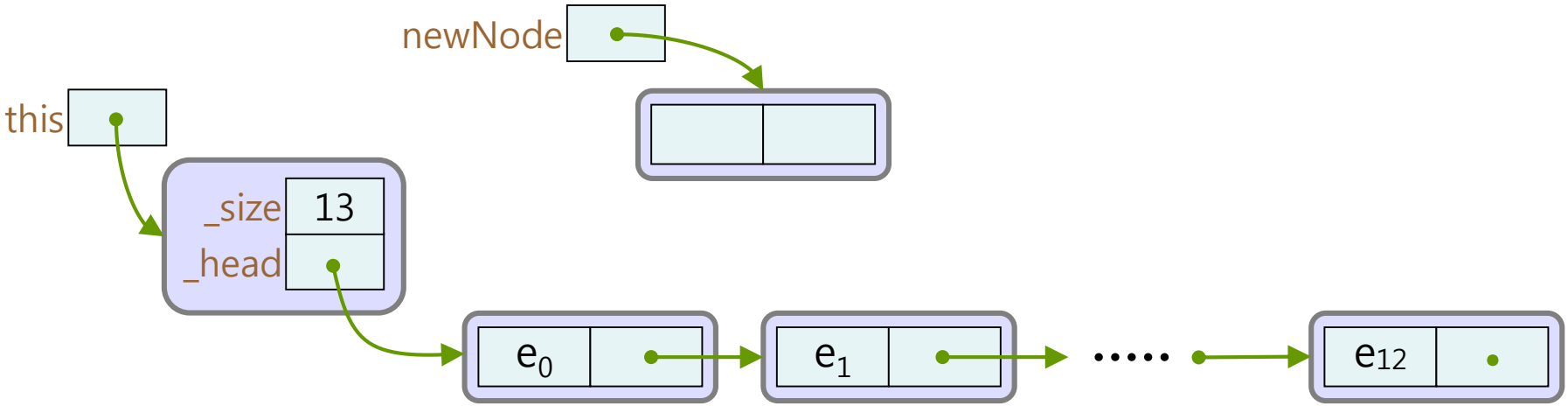
LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```



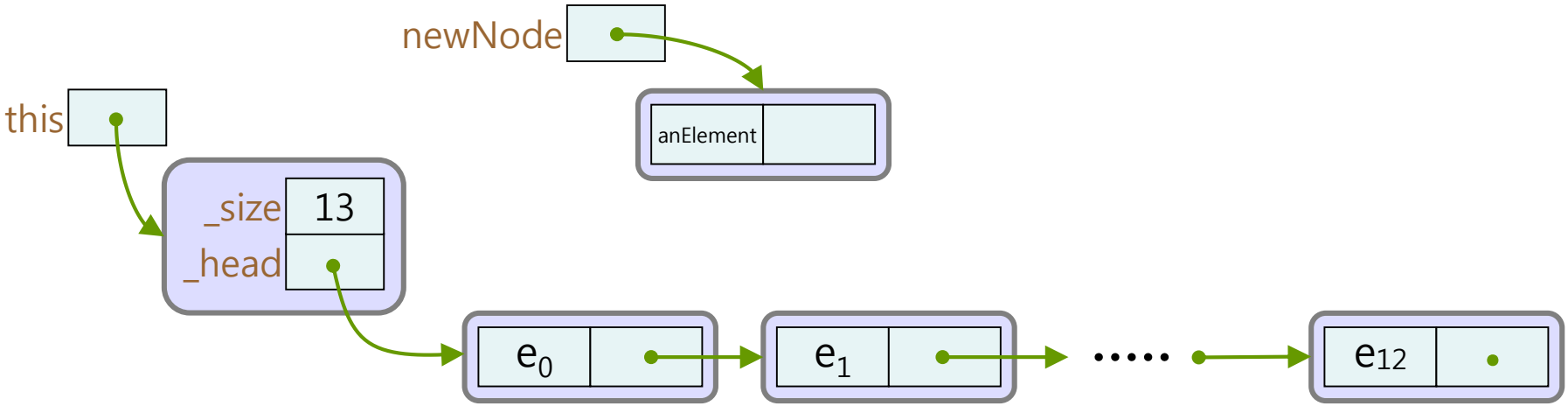
LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```



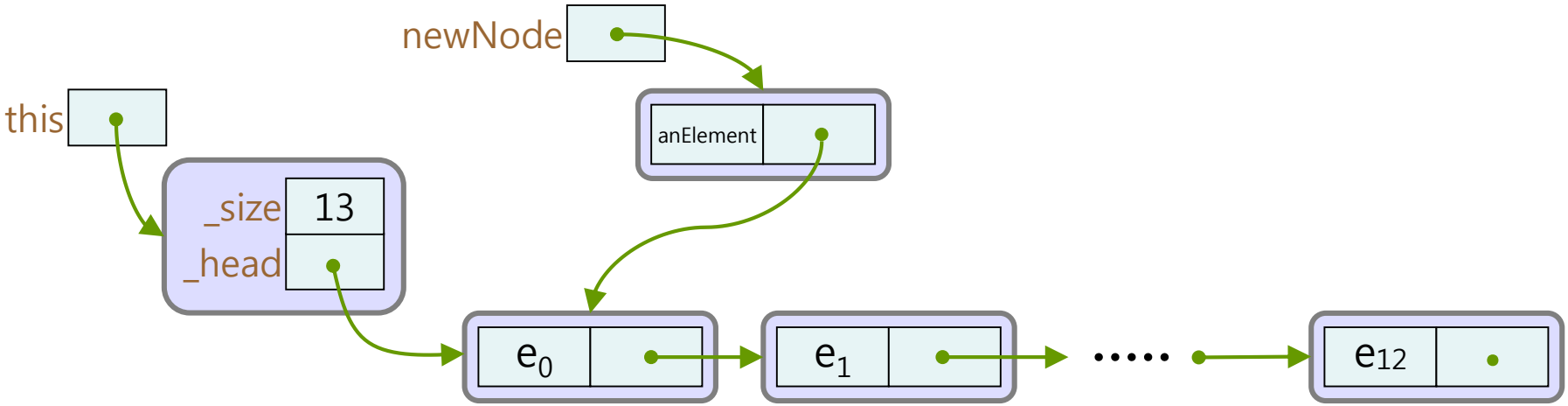
LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```



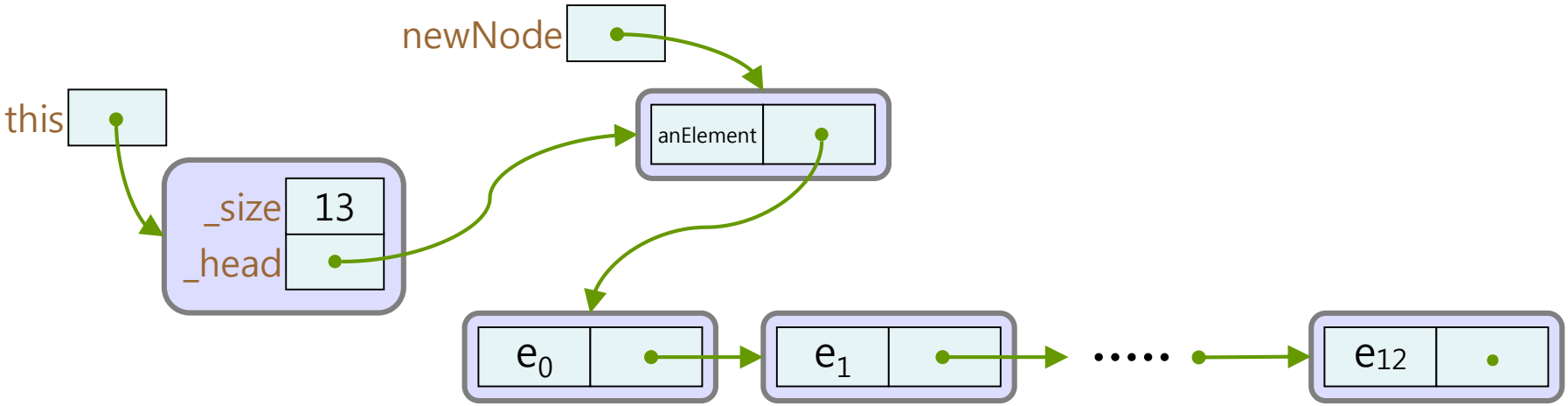
LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```



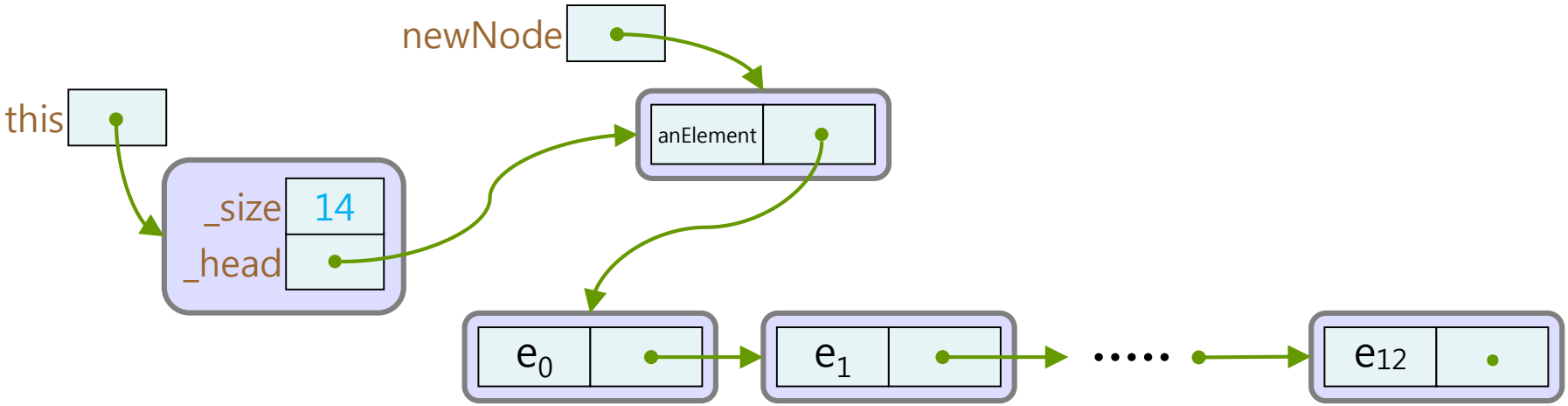
LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```



LinkedBag: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        Node  newNode = new Node() ;
        newNode.setElement(anElement) ;
        newNode.setNext(this._head) ;
        this._head = newNode ;
        this._size++ ;
        return  true ;
    }
}
```

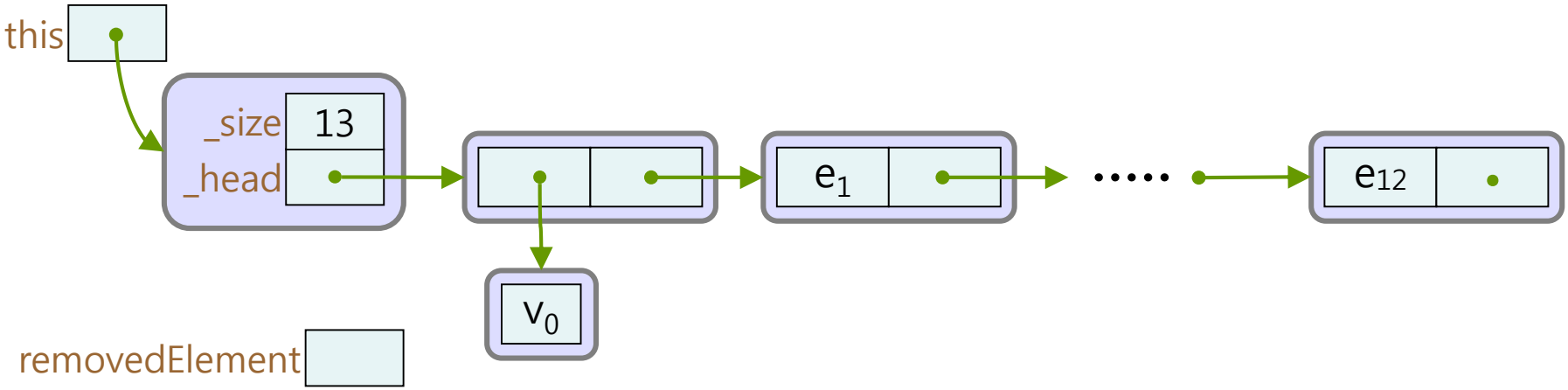


LinkedBag: removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement = this._head.element() ;
        this._head = this._head.next() ;
        this._size-- ;
        return removedElement ;
    }
}
```

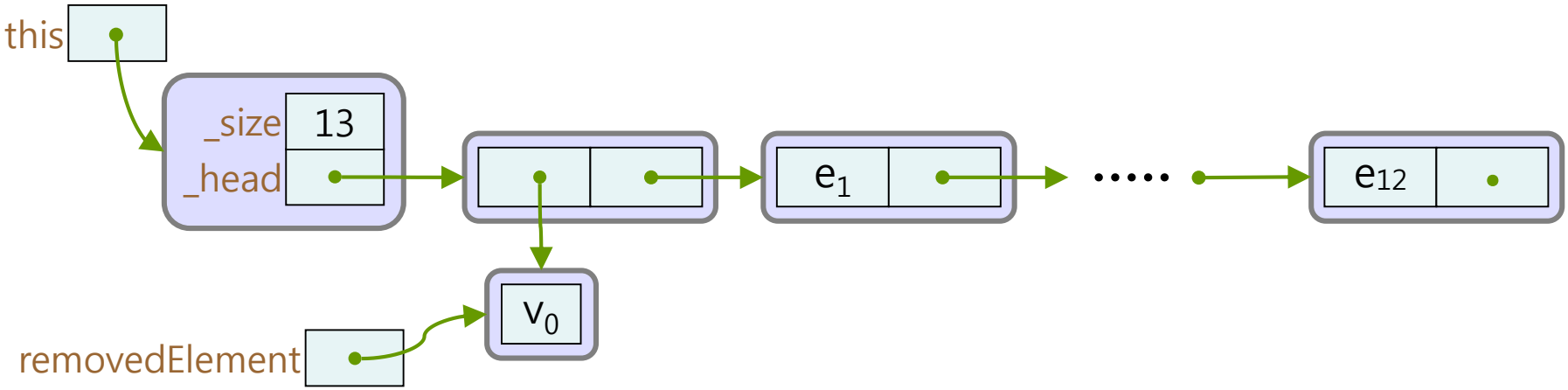
LinkedBag: removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement = this._head.element() ;
        this._head = this._head.next() ;
        this._size-- ;
        return removedElement ;
    }
}
```



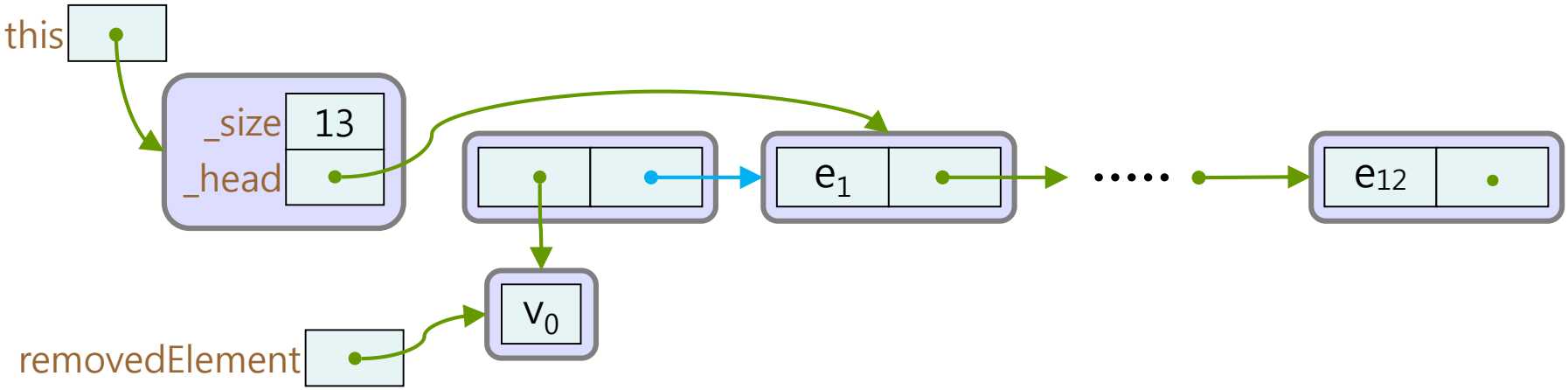
LinkedBag: removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement = this._head.element() ;
        this._head = this._head.next() ;
        this._size-- ;
        return removedElement ;
    }
}
```



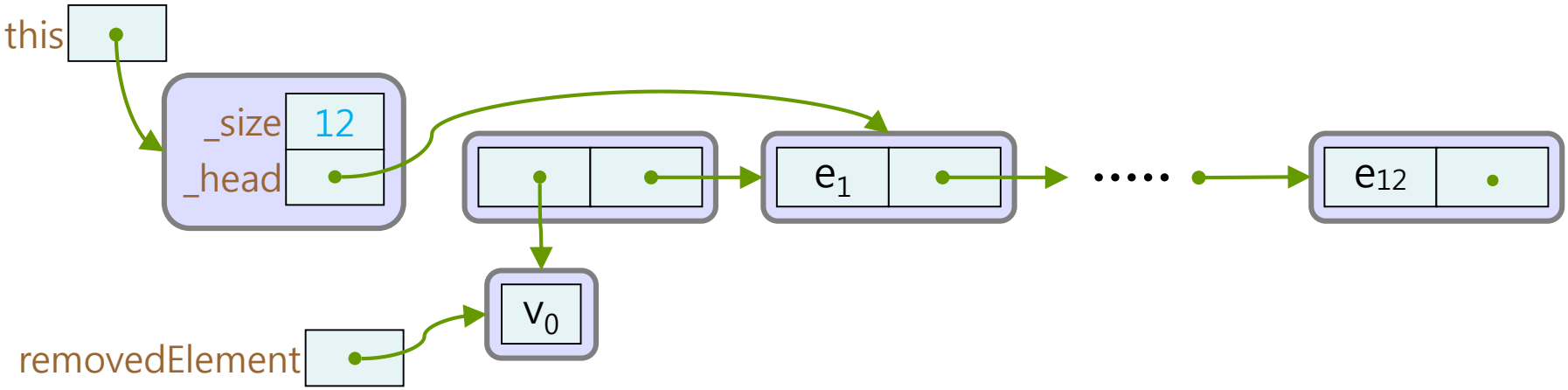
LinkedBag: removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement = this._head.element() ;
        this._head = this._head.next() ;
        this._size-- ;
        return removedElement ;
    }
}
```



LinkedBag: removeAny()

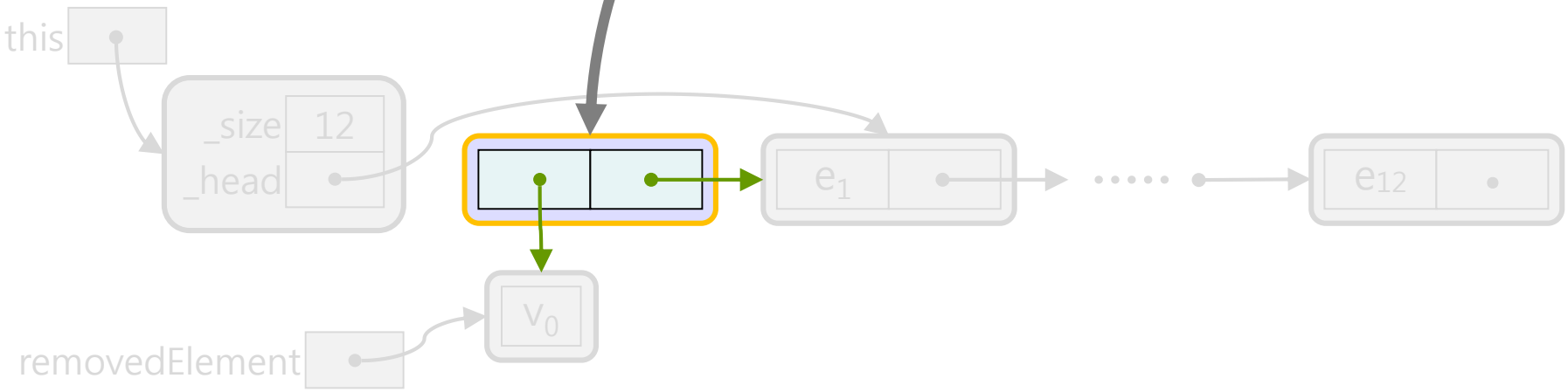
```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement = this._head.element() ;
        this._head = this._head.next() ;
        this._size-- ;
        return removedElement ;
    }
}
```



LinkedBag: removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement
        this._head = this._head.ne
        this._size-- ;
        return removedElement ;
    }
}
```

함수 종료 후에
이 노드는 어떻게 될까?
아무 곳에서도
이 노드를 가지고 있지 않음!!!

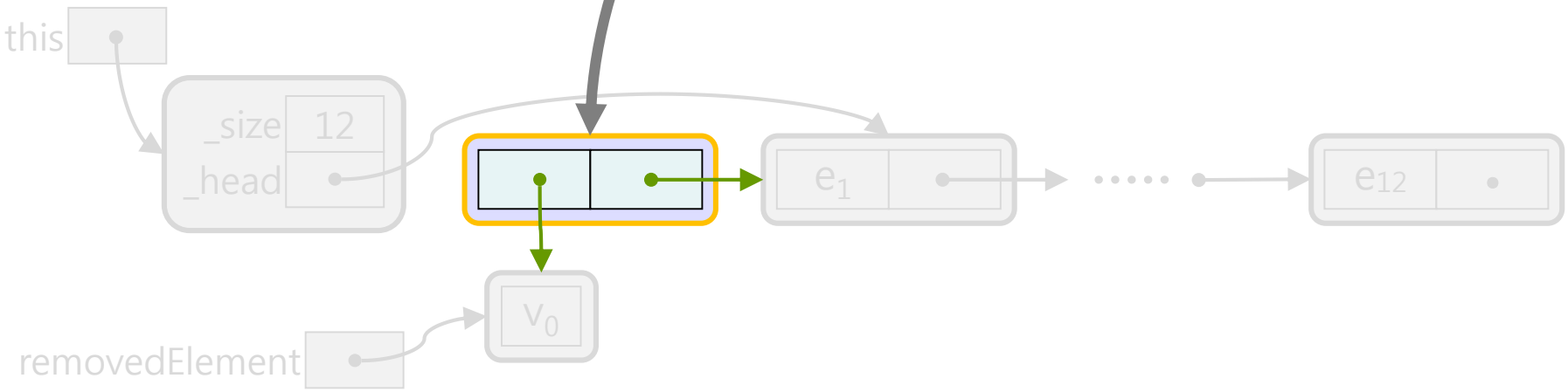


LinkedBag: removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return null ;
    }
    else {
        Element removedElement
        this._head = this._head.ne
        this._size-- ;
        return removedElement ;
    }
}
```

Java 시스템은 이러한 메모리 조각들을
주기적으로 찾아 모아서
다시 사용할 수 있게 한다!

➡ "Garbage Collection"

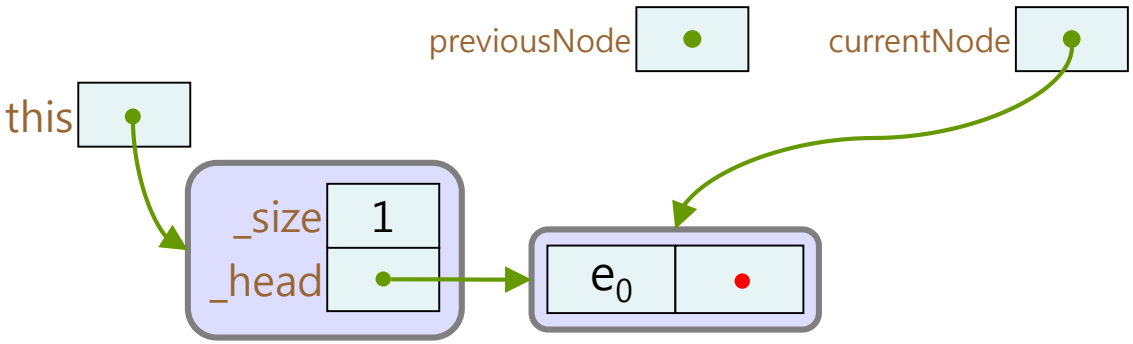


LinkedBag: remove()

```
public boolean  remove (Element anElement)
{
        if ( this.isEmpty() ) {
                return false ;
        }
        else {
                Node  previousNode = null ;
                Node  currentNode = _head ;
                boolean  found = false ;
                // 첫번째 단계: 삭제할 위치 찾기
                while ( currentNode != null && !found ) {
                        if ( currentNode.element().equals(anElement) ) {
                                found = true ;
                        }
                        else {
                                previousNode = currentNode  ;
                                currentNode = currentNode.next() ;
                        }
                }

                // 두번째 단계: 삭제하기
                if ( ! found ) {
                        return false ;
                }
                else {
                        if ( currentNode == this._head ) {
                                this._head = this._head.next() ;
                        }
                        else {
                                previousNode.setNext(currentNode.next()) ;

                        this._size-- ;
                        return true ;
                }
        }
}
```

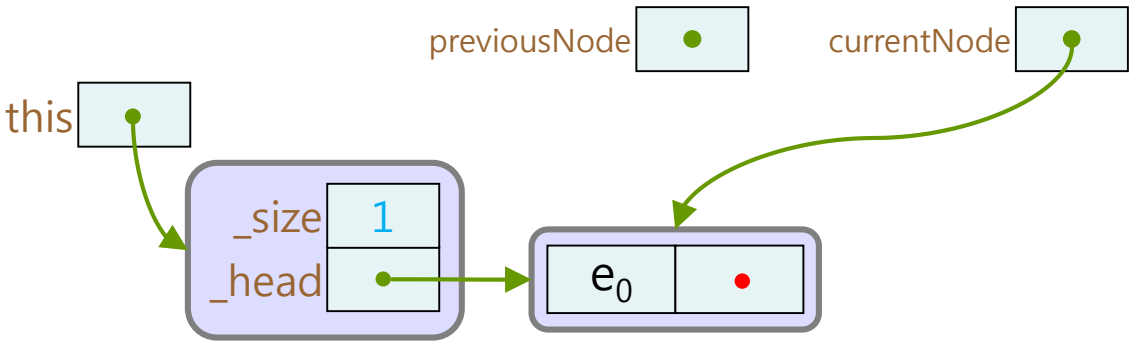
LinkedBag: remove() [single node]

```
public boolean  remove (Element anElement)
{
    ......
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
         // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found ) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode  ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ......
```



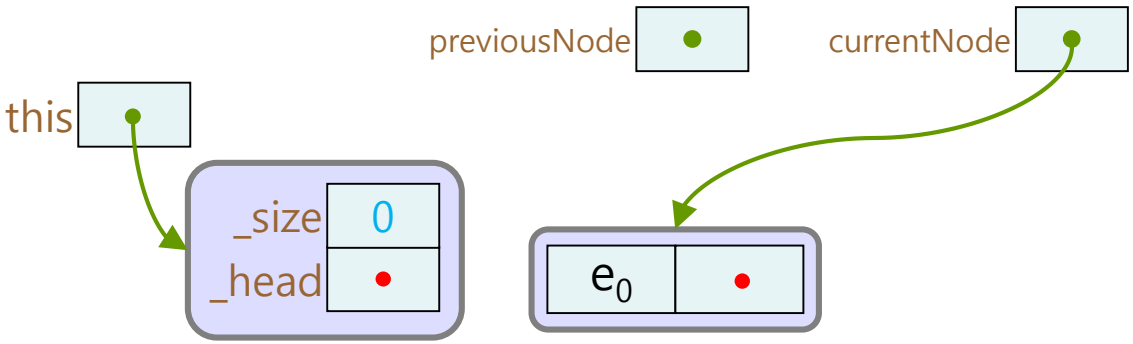
LinkedBag: remove() [single node]

```
public boolean  remove (Element anElement)
{
    ......
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
        // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode  ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ......
```



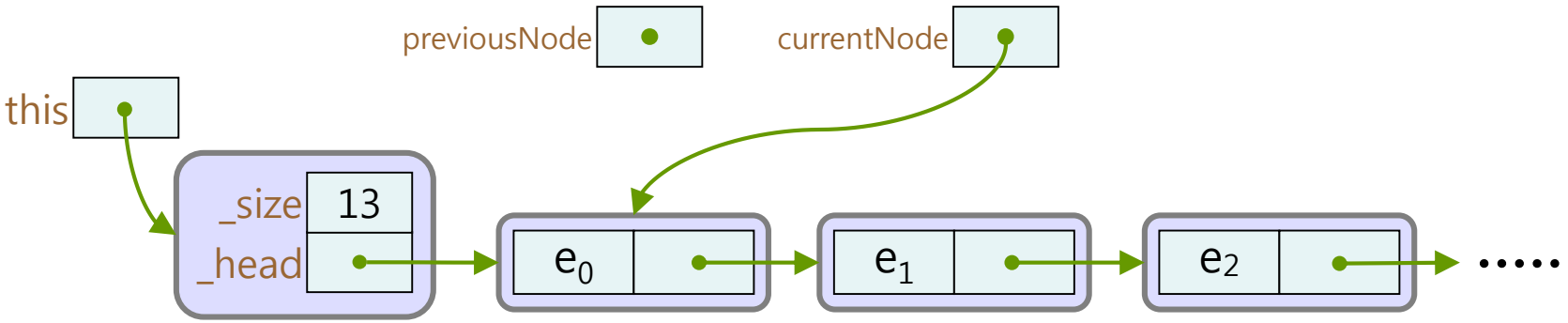
LinkedBag: remove() [single node]

```
public boolean  remove (Element anElement)
{
    ......
        // 두번째 단계: 삭제하기
        if (! found ) {
            return false ;
        }
        else {
            if ( currentNode == this._head ) {
                this._head = this._head.next() ;
            }
            else {
                previousNode.setNext(current.next()) ;
            }
            this._size-- ;
    ......
}
```



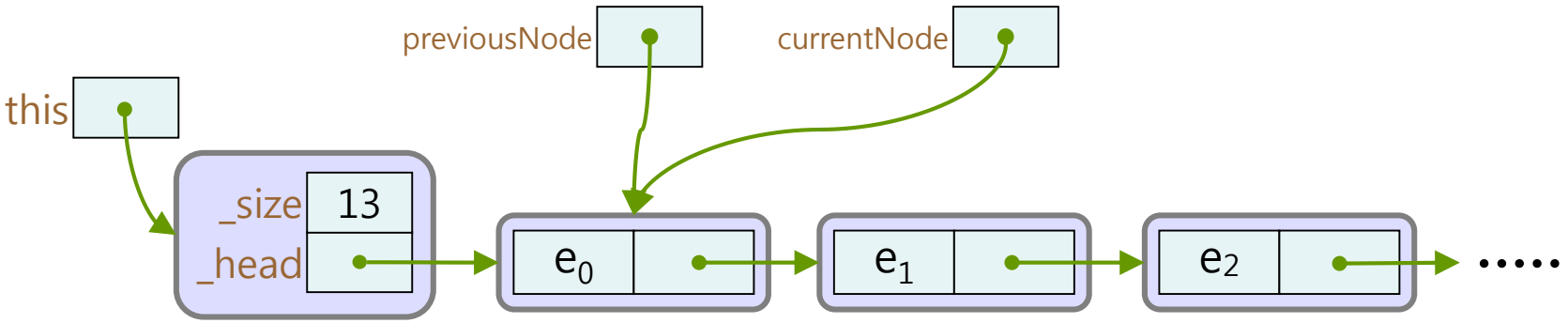
LinkedBag: remove() [multiple nodes]

```
public boolean  remove (Element anElement)
{
    ……
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
        // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found ) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode  ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ……
```



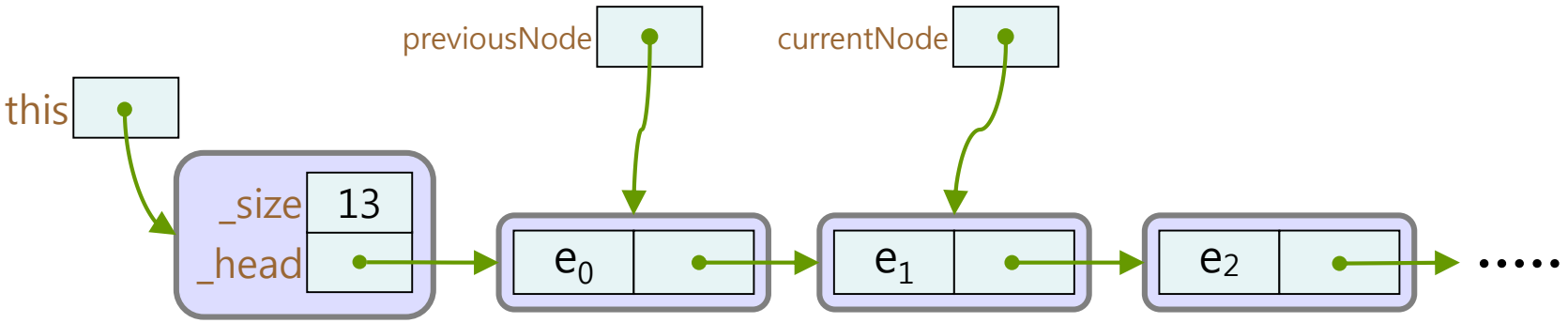
LinkedBag: remove() [multiple nodes]

```
public boolean  remove (Element anElement)
{
    ……
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
        // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found ) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode  ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ……
```



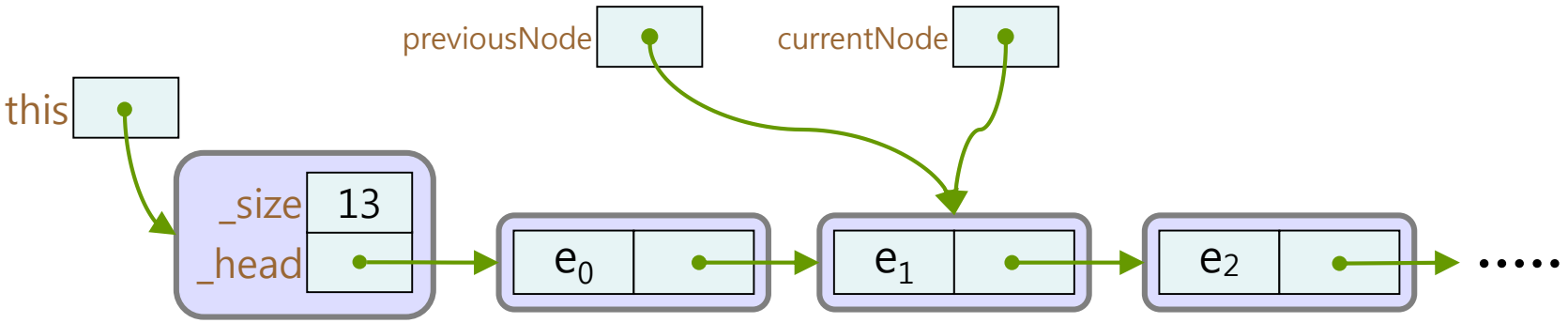
LinkedBag: remove() [multiple nodes]

```
public boolean  remove (Element anElement)
{
    ……
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
        // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found ) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ……
```



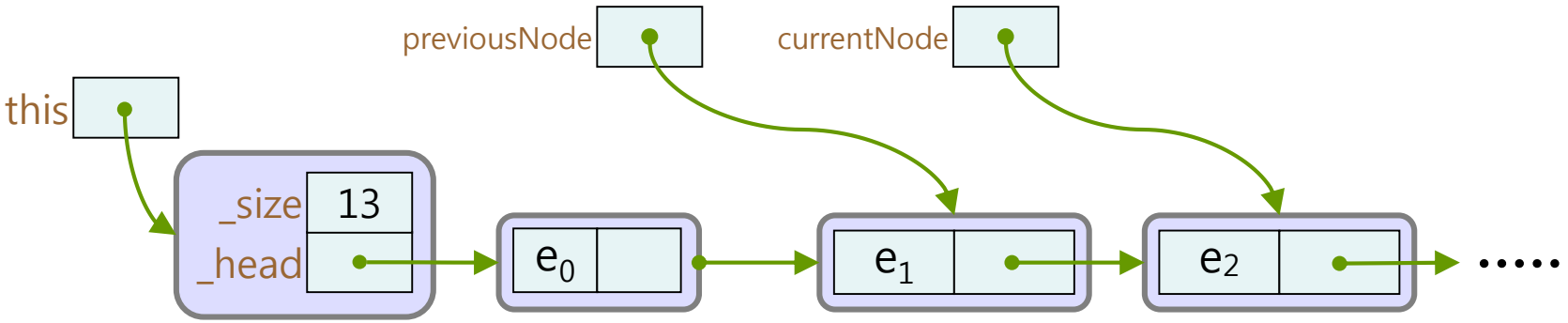
LinkedBag: remove() [multiple nodes]

```
public boolean  remove (Element anElement)
{
    ……
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
        // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ……
```



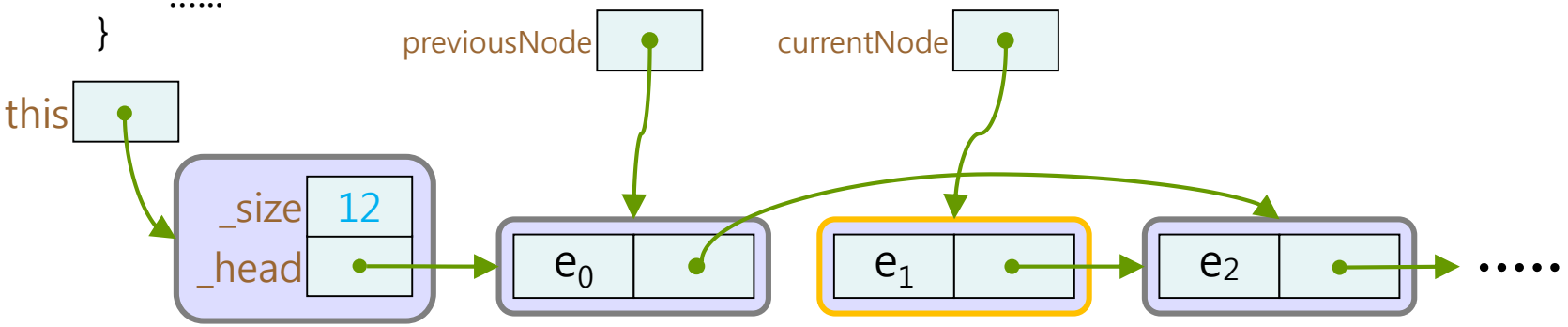
LinkedBag: remove() [multiple nodes]

```
public boolean  remove (Element anElement)
{
    ……
        Node  previousNode = null ;
        Node  currentNode = this._head ;
        boolean  found = false ;
        // 첫번째 단계: 삭제할 위치 찾기
        while ( currentNode != null && !found ) {
            if ( currentNode.element().equals(anElement) ) {
                found = true ;
            }
            else {
                previousNode = currentNode ;
                currentNode = currentNode.next() ;
            }
        }
        // 두번째 단계: 삭제하기
    ……
```



LinkedBag: remove() [multiple nodes]

```
public boolean  remove (Element anElement)
{
      ……
          // 두번째 단계: 삭제하기
          if ( ! found ) {
                    return false ;
          }
          else {
               if (currentNode = this._head) {
               {
                    this._head = this._head.next() ;
               }
               else {
                    previous.setNext(current.next()) ;
               }
               this._size-- ;
                return true ;
          }
      ……
}
```



LinkedBag: clear()

```
// 내용 바꾸기
.....

public void  clear()
{
    this._size = 0 ;
    this._head = null ;
}
```

연결 체인에서의 순차검색 (Sequential Search)

순차 검색: Version 1

```
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;
    Node  currentNode = this._head ;
    while ( currentNode != null && ! found ) {
        if ( currentNode.element().equals(anElement) ) {
            found = true ;
        }
        currentNode = currentNode.next() ;
    }
    return found ;
}
```

- 시간은 얼마나 걸릴까?
  - this._size 의 값을 n 이라고 하면…

순차 검색: Version 2

```
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;
    Node  currentNode = this._head ;
    while ( currentNode != null && ! found )  {
        if ( currentNode.element().equals(anElement) ) {
            found = true ;
            return  true ;
        }
        currentNode = currentNode.next() ;
    }
    return  found ;
    return  false ;
}
```

- Version 1과 Version 2
  - 시간적 성능의 차이는?
  - 각각의 장단점은?
  - 어느 코드가 더 이해하기 좋을까?
  - 결과가 true이든 false이든 (이 예제의 경우 return 하기 전에) 공통적으로 해야 할 일이 더 있다면?

순차 검색: Version 3

```
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;

    for (   Node currentNode = _this.head ;
            currentNode != null && ! found ;
            currentNode = currentNode.next() )
    {
        if ( currentNode.element().equals(anElement) ) {
            found = true ;
        }
    }
    return found ;
}
```

순차 검색: remove()

```
public boolean  remove (Element anElement)
{
        if ( this.isEmpty() ) {
                return false ;
        }
        else if {
                Node  previousNode = null ;
                Node  currentNode = _head ;
                boolean  found = false ;

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

                // 두번째 단계: 삭제하기
                if ( ! found ) {
                        return false ;
                }
                else {
                        if ( currentNode = this._head ) {
                                this._head = this._head.next() ;
                        }
                        else {
                                previousNode.setNext(currentNode.next()) ;
                        }
                        this._size-- ;
                        return true ;
                }
        }
}
```

배열과 연결체인

배열과 연결 체인

- 배열에서의 문제점:
  - 연속된 (contiguous) 메모리 공간에 원소들을 저장
  - 임의의 원소를 삽입하거나 삭제하는데 많은 시간이 걸린다.
  - 최대 크기를 미리 지정

- 연결 체인의 장점:
  - 배열과 달리 원소들이 메모리에 연속적으로 모여 있지 않아도 된다. 아무 곳이나 가능하다.
  - 저장할 원소의 개수에 제한을 받지 않는다.
  - 원소를 삽입/삭제 하는데 배열보다 유리하다.

ArrayBag 구현의 문제점 [1]

- Bag을 Java 언어로 구현하는 손쉬운 방법은 배열(array)을 사용하는 것이다.
  - 예: 최대 100개 동전 가방

```
public class ArrayBag
{
    // 비공개 인스턴스 변수
    private int                 _size ;
    private Element[100]  _coins ;
```

ArrayBag 구현의 문제점 [2]

- 어떤 점이 불편한가?
  - 처음에는 동전 수가 최대 100 개면 충분했는데, 나중에 120개로 늘어난다면?
    - 실행 중에는 대처 불능
      - 현재의 실행 프로그램으로는 120 개의 상황이 발생하면 실행이 불가능하다.
    - 사후 대처 방법
      - 프로그램 원시 코드에서 배열의 크기를 100에서 120으로 바꾸어주고, 다시 컴파일 하여, 새로운 실행 프로그램을 얻는다.
    - 동적 대응 방법
      - 실행 중에 배열의 크기를 확장
      - 크기에 비례한 시간이 걸림

ArrayBag 구현의 문제점 [2]

- 어떤 점이 불편한가? [계속]
  - 새로운 원소를 배열의 맨 앞에 삽입해야 한다면?
    - 즉 0 번째 위치에 삽입해야 한다면, 기존의 모든 원소를 맨 뒤부터 하나씩 뒤로 밀고 0 번째 위치를 비운 다음 삽입할 수 있다.

```
for ( i = this._size ; i > 0 ; i-- ) {
    this._coins[i] = this._coins[i-1] ;
}
this._coins[0] = addedCoin ;
this._size ++ ;
```

    - 리스트에 원소의 개수가 많아지면 그 수에 비례해 시간이 많이 걸리게 된다.
  - 배열의 맨 앞의 원소를 삭제하려면?
    - 즉 0 번째 원소를 삭제해야 한다면, 기존의 모든 원소를 맨 앞에서부터 하나씩 앞으로 당겨야 한다.

```
removedCoin = this._coins[0] ;
for ( i = 1 ; i < this._size ; i++ ) {
    this._coins[i-1] = this._coins[i] ;
}
this._size -- ;
```

    - 배열에 원소의 개수가 많아지면 그 수에 비례해 시간이 많이 걸리게 된다.

ArrayBag 구현의 문제점 [3]

어떤 점이 불편한가? [계속]

- 새로운 원소를 k 번째 위치에 삽입하려면?

```
for ( i = this._size ; i > k ; i-- ) {
    this._coins[i] = this._coins[i-1] ;
}
this._coins[k] = addedCoin ;
```

  - (n-k) 개의 원소를 한 칸씩 뒤로 이동시켜야 한다.
- k 번째 위치의 원소를 삭제하려면?

```
removedCoin = this._coin[k] ;
for ( i = k+1 ; i < this._size ; i++ ) {
    this._coins[i-1] = this._coins[i] ;
}
```

  - (n-k-1) 개의 원소를 한 칸씩 앞으로 이동시켜야 한다.
- 결국, 삽입이나 삭제의 경우 전체 리스트의 길이가 길어지면 그만큼 해야 할 일이 늘어난다.

ArrayBag 구현의 문제점 [4]

- 장점은?
  - 프로그램에서 다루기가 간편한 편이다.
    - 프로그램 코드 작성이 상대적으로 단순하며, 이해하기도 쉬운 편이다.
    - 이는 연결 체인을 이용한 구현과 비교된다.
  - 메모리 활용이 효율적일 수 있다.
    - 동일한 양의 자료를 저장할 경우, 메모리의 사용량이 연결 체인을 이용한 구현에 비해 적다.

실습: Coin Collection

실습: Coin Collection

- 입력 메뉴에서 선택된 일을 한다
  - coin 삽입
  - 임의의 coin 삭제
  - 주어진 coin 삭제
  - 주어진 coin이 Collector 안에 있는지?
  - 주어진 coin이 Collector 안에 몇 개 있는지?
  - 코인의 종류별 개수
  - coin collector 비우기
- LinkedBag coinCollector ;

End of "LinkedBag"

집합 (Set)

Set 객체 사용법

- Set 객체 생성과 소멸
  - Set 객체 생성
- Set 상태 알아보기
  - Set 에 들어있는 원소의 개수를 알려주시오
  - Set 이 비어 있는지 알려주시오
  - Set 이 가득 찾는지 알려주시오
  - 주어진 원소가 Set 에 있는지 알려주시오
- Set 내용 알아보기
  - Set 에서 아무 원소 하나를 얻어내시오
- Set 내용 바꾸기
  - Set 에 주어진 원소를 넣으시오
  - Set 에서 아무 원소 하나를 제거하여 얻어내시오
  - Set 에서 지정된 원소를 찾아서 있으면 제거하시오
  - Set 을 비우시오

Class "Set"

Set 의 공개함수

- Set 객체 사용법을 Java로 구체적으로 표현
  - public ArraySet () { }
  - public int size () { }
  - public boolean isEmpty () { }
  - public boolean isFull () { }
  - public boolean doesContain (Element anElement) { }
  - public Element any() { }
  - public boolean add (Element anElement) { }
  - public Element removeAny () { }
  - public boolean remove (Element anElement) { }
  - public void clear () { }

Class "ArraySet"

"ArraySet" as a Set

- ArraySet
  - 추상적인 Set에 대해, 구현하는 방법을 반영하여 지은 이름
  - 집합에는 동일한 원소가 들어 있을 수 없다
  - Java Array를 이용하여 구현

ArraySet의 공개함수

- ArraySet 객체 사용법을 Java로 구체적으로 표현
  - public ArraySet() { }
  - public ArraySet(int givenMaxSize) { }
  - public int size () { }
  - public boolean isEmpty () { }
  - public boolean isFull () { }
  - public boolean doesContain (Element anElement) { }
  - public Element any() { }
  - public boolean add (Element anElement) { }
  - public Element removeAny( ) { }
  - public boolean remove (Element anElement) { }
  - public void clear () { }

ArraySet의 구현

```
public class  ArraySet
{
    // 비공개 인스턴스 변수
    private static final int  DEFAULT_MAX_SIZE = 100 ;
    private int               _maxSize ;
    private int               _size ;
    private Element[]  _elements ; // ArraySet의 원소들을 담을 java 배열
```

ArraySet의 구현: 객체의 생성

```
public calss  ArraySet
{
    // 비공개 인스턴스 변수
    ......

    // 생성자
    public ArraySet  ()
    {
        this._maxSize = DEFAULT_MAX_SIZE ;
        this._elements = new  Element[DEFAULT_MAX_SIZE] ;
        this._size = 0 ;
    }

    public ArraySet  (int givenMaxSize)
    {
        this._maxSize = givenMaxSize ;
        this._elements = new  Element[givenMaxSize] ;
        this._size = 0 ;
    }
```

ArraySet의 구현 : 상태 알아보기

```
public calss ArraySet
{
    // 비공개 인스턴스 변수
    ……

    // 생성자
    ……

    // 상태 알아보기
    public int  size ()
    {
        return  this._size ;
    }

    public boolean  isEmpty ()
    {
        return  (this._size == 0) ;
    }

    public boolean  isFull ()
    {
        return  (this._size == this._maxSize) ;
    }
```

ArraySet : 상태 알아보기

```
// 상태 알아보기
.....
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;
    int  i ;
    for ( i = 0 ;  i < this._size  && ! found ; i++ ) {
        if ( this._elements[i].equals(anElement) ) {
            found = true ;
        }
    }
    return found ;
}
```

ArraySet: 내용 알아보기

```
// 내용 알아보기
public Element  any ()
{
    if ( this.isEmpty() ) {
        return  null ;
    }
    else {
        return  this._elements[0] ;
    }
}
```

ArraySet: add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return  false ;
    }
    else {
        if ( ! doesContain(anElement) ) {
            this._elements[this._size] = anElement ;
            this._size++ ;
            return  true ;
        }
        else {
            return false ;
        }
    }
}
```

ArraySet : removeAny()

```
// 내용 바꾸기
.....
public Element  removeAny ()
{
    if ( this.isEmpty() ) {
        return  null ;
    }
    else {
        this._size -- ;
        Element  removedElement = this._elements[this._size] ;
        this._elements[this._size] = null ;
        return  removedElement ;
    }
}
```

ArraySet : remove()

```
// 내용 바꾸기
.....
public boolean  remove (Element anElement)
{
    int  positionForRemove = 0 ;

    // 첫번째 단계: 삭제할 원소의 위치 찾기
    while ( (positionForRemove < this._size) &&
            ! (this._elements[positionForRemove].equals(anElement) )
    {
        positionForRemove ++ ;
    }

    // 두번째 단계: 삭제하기
    if ( positionForRemove < this._size ) {
        for ( int j = positionForRemove ; j < this._size-1 ; j++ ) {
            this._elements[j] = this._elements[j+1] ;
        }
        this._size-- ;
        this._elements[this._size] = null ;
        return true ;
    }
    else {
        return false ;
    }
}
```

Class "ArraySet"

```
// 내용 바꾸기
......

public void  clear ()
{
    for ( int i = 0 ; i < this._size ; i++ ) {
        this._elements[i] = null ;
    }
    this._size = 0 ;
}
```

Class "LinkedSet"

LinkedSet의 공개함수

- LinkedSet 객체 사용법을 Java로 구체적으로 표현
  - public LinkedSet () { }
  - public LinkedSet (int givenMaxSize) { }
  - public int size () { }
  - public boolean isEmpty () { }
  - public boolean isFull () { }
  - public boolean doesContain (Element anElement) { }
  - public Element any () { }
  - public boolean add (Element anElement) { }
  - public Element removeAny () { }
  - public boolean remove (Element anElement) { }
  - public void clear () { }

LinkedSet 의 구현

```
public class  LinkedSet
{
    // 비공개 인스턴스 변수
    private int        _size ;
    private Node    _head ;
```

LinkedSet 의 구현: 객체의 생성

```
public calss  LinkedSet
{
    // 비공개 인스턴스 변수
    ……

    // 생성자
    public  LinkedSet ()
    {
        this._size = 0 ;
        this._head= null ;
    }
```

LinkedSet 의 구현 : 상태 알아보기

```
public calss  LinkedSet
{
    // 비공개 인스턴스 변수
    ……

    // 생성자
    ……

    // 상태 알아보기
    public int  size ()
    {
        return  this._size ;
    }

    public boolean  isEmpty ()
    {
        return  (this._size == 0) ; // 또는 return (this._head == null) ;
    }

    public boolean  isFull ()
    {
        return  false ;
    }
```

LinkedSet : 상태 알아보기

```
// 상태 알아보기
.....
// 순차 검색
public boolean  doesContain (Element anElement)
{
    boolean  found = false ;
    Node  currentNode = _head ;
    while ( currentNode != null && ! found ) {
        if ( currentNode.element().equals(anElement) ) {
            found = true ;
        }
        currentNode = currentNode.next() ;
    }
    return found ;
}
```

LinkedSet : 내용 알아보기

```
// 내용 알아보기
public  Element  any ()
{
    if ( this.isEmpty() ) {
        return  null ;
    }
    else {
        return  this._head.element() ;
    }
}
```

LinkedSet : add()

```
// 내용 바꾸기
public boolean  add (Element anElement)
{
    if ( this.isFull() ) {
        return  false ;
    }
    else {
        if ( ! this.doesContain(anElement) ) {
            Node newNode = new Node(anElement, this._head) ;
                // newNode.setElement(anElement) ;
                // newNode.setNext(this._head) ;
            this._head = newNode ;
            return  true ;
        }
        else {
            return  false ;
        }
    }
}
```

LinkedSet : removeAny()

```
// 내용 바꾸기
.....
public Element   removeAny()
{
    if ( this.isEmpty() ) {
        return   false ;
    }
    else {
        Node removedNode = this._head ;
        this._head = removedNode.next() ;
        this._size -- ;
        return   removedNode.element() ;
    }
}
```

LinkedSet : remove()

```
// 내용 바꾸기
.....
public boolean  remove (Element anElement)
{
        if ( this.isEmpty() ) {
                return false ;
        }
        else {
                // 첫번째 단계: 삭제할 위치 찾기
                Node  previousNode = null ;
                Node  currentNode = _head ;
                boolean  found = false ;
                while ( currentNode != null && !found ) {
                        if ( currentNode.element().equals(anElement) ) {
                                found = true ;
                        }
                        else {
                                previousNode = currentNode  ;
                                currentNode = currentNode.next() ;
                        }
                }
                // 두번째 단계: 삭제하기
                if ( ! found ) {
                        return false ;
                }
                else {
                        if ( currentNode = this._head ) {
                                this._head = this._head.next() ;
                        }
                        else {
                                previousNode.setNext (currentNode.next()) ;
                        }
                        this._size-- ;
                        return true ;
                }
        }
}
```

LinkedSet : clear()

```
// 내용 바꾸기
.....

public void  clear()
{
    this._size = 0 ;
    this._head = null ;
}
```

실습: Constellation

실습: Constellation

- 입력 메뉴에서 선택된 일을 한다
  - 별의 위치와 별 이름 삽입 (xxx,yyy,name)
  - 주어진 별 삭제
  - 주어진 별이 Collector 안에 있는지?
  - 별의 총 개수
  - coin collector 비우기
- ArraySet StarCollector ;

End of "Sets"