자료구조: 2022년 1학기 [강의]

Class “ArrayBag”

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터융합학부

© J.-H. Kang, CNU

Class “ArrayBag”

"ArrayBag" as a Bag

- ArrayBag
  - 추상적인 Bag 에 대해, 구현하는 방법을 반영하여 지은 이름
  - Java Array 를 이용하여 구현

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

ArrayBag 의 사용법 (공개함수)

- ArrayBag 객체 사용법을 Java 로 구체적으로 표현

ArrayBag 의 사용법 (공개함수)

- ArrayBag 객체 사용법을 Java 로 구체적으로 표현
  - public ArrayBag () { }
  - public ArrayBag (int givenCapacity) { }

  - public int size () { }
  - public boolean isEmpty () { }
  - public boolean isFull () { }
  - public boolean doesContain (E anElement) { }
  - public int frequencyOf (E anElement) { }

  - public E any() { }

  - public boolean add (E anElement) { }
  - public E removeAny () { }
  - public boolean remove (E anElement) { }
  - public void clear () { }

Class "ArrayBag"의 초기 형태는 이렇게!

```
public class  ArrayBag<E>
{
        public  ArrayBag ( )
        {       // 수정해야 함
        }
        public  ArrayBag (int givenCapacity)
        {       // 수정해야 함
        }

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

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

        public boolean      add (E anElement) { return true ; /* 수정해야 함 */ }
        public E            removeAny () { return null ; /* 수정해야 함 */ }
        public boolean      remove (E anElement) { return true ; /* 수정해야 함 */ }
        public void         clear () { /* 수정해야 함 */ }

} // End of Class "ArrayBag"
```

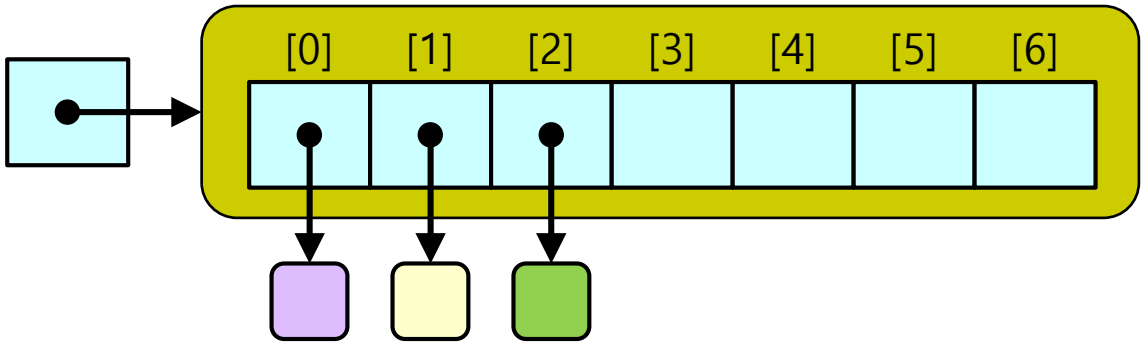
이렇게만 정의해 두어도 사용하는 곳에서 프로그래밍 하는 데는 전혀 지장이 없다. 즉 컴파일 오류가 발생하지 않는다.

Class "ArrayBag" 의 구현

ArrayBag 객체에 필요한 속성은?

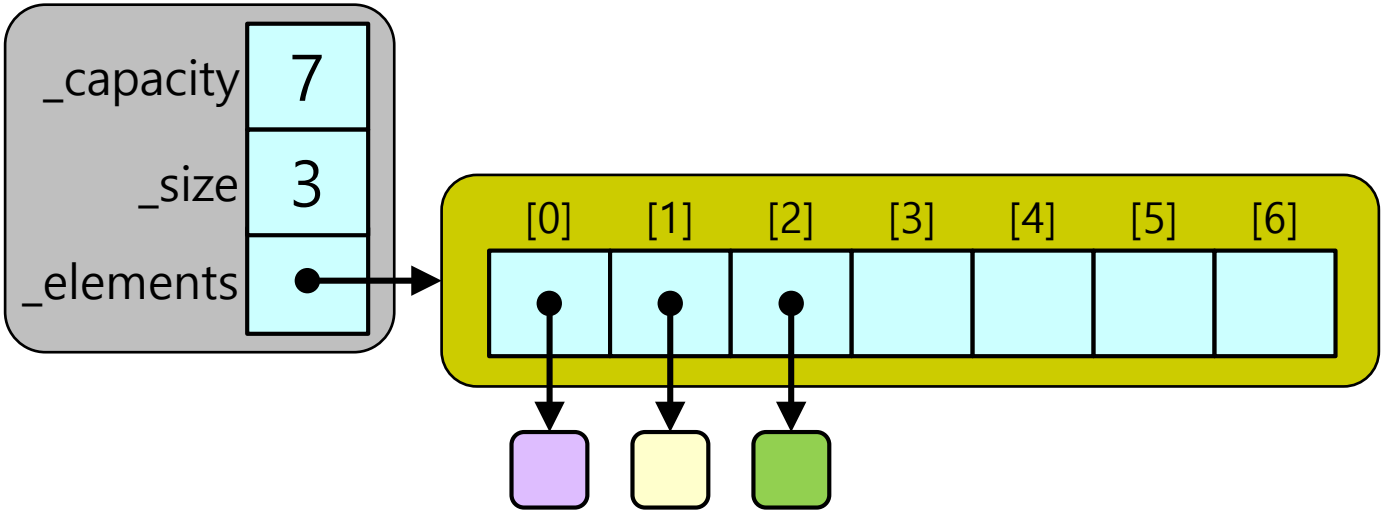
- 객체의 속성은 객체를 구성하는 부품
  - 객체 내부 구조에 따라, 부품이 결정된다.

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



ArrayBag 의 속성

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

ArrayBag 에서 객체의 속성 표현

```
public class  ArrayBag<E>
{
    // 모든 ArrayBag 객체에서 공통으로 사용
    private static  final  int DEFAULT_CAPACITY = 100 ;

    // 비공개 인스턴스 변수 (Private Instance Variables)
    private int          _capacity ;
    private int          _size ;
    private E[]          _elements ; // 원소들을 저장할 배열
```

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

❑ 모든 인스턴스 변수에는 Getter/Setter 를 두자!

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

      // Getters / Setters
      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 의 구현

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

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

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

ArrayBag.DEFAULT_CAPACITY

즉, "static" 상수/변수는 그 앞에 class 이름을 붙여 사용한다.
(인스턴스 변수는 그 앞에 "this"를 붙여 사용한다.)

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

ArrayBag 의 구현: 객체의 생성자

```
public class  ArrayBag<E>
{
    // 비
    .....

    // 생성자 1
    @SuppressWarnings ("unchecked")
    public ArrayBag ( )
    {
        this.setCapacity (ArrayBag.DEFAULT_CAPACITY) ;
        this.setElements ( (E[]) new Object [this.capacity()] ) ;
        this.setSize (0) ;
    }

    // 생성자 2
    @SuppressWarnings ("unchecked")
    public ArrayBag (int givenCapacity)
    {
        this.setCapacity (givenCapacity) ;
        this.setElements ( (E[]) new Object[this.capacity()] ) ;
        this.setSize (0) ;
    }
```

생성자는 하나만 존재해야 할까?

ArrayBag 의 구현: 여러 개의 생성자

```
public class  ArrayBag<E>
{
    // 비공개 인스턴스 변수
    ......

    // 생성자 1
    @SuppressWarnings ("unchecked")
    public ArrayBag ( )
    {
        this.setCapacity (ArrayBag.DEFAULT_CAPACITY) ;
        this.setElements ( (E[]) new Object [this.capacity()] ) ;
        this.setSize (0) ;
    }

    // 생성자 2
    @SuppressWarnings ("unchecked")
    public ArrayBag (int givenCapacity)
    {
        this.setCapacity (givenCapacity) ;
        this.setElements ( (E[]) new Object[this.capacity()] ) ;
        this.setSize (0) ;
    }
```

여러 개의 생성자들:

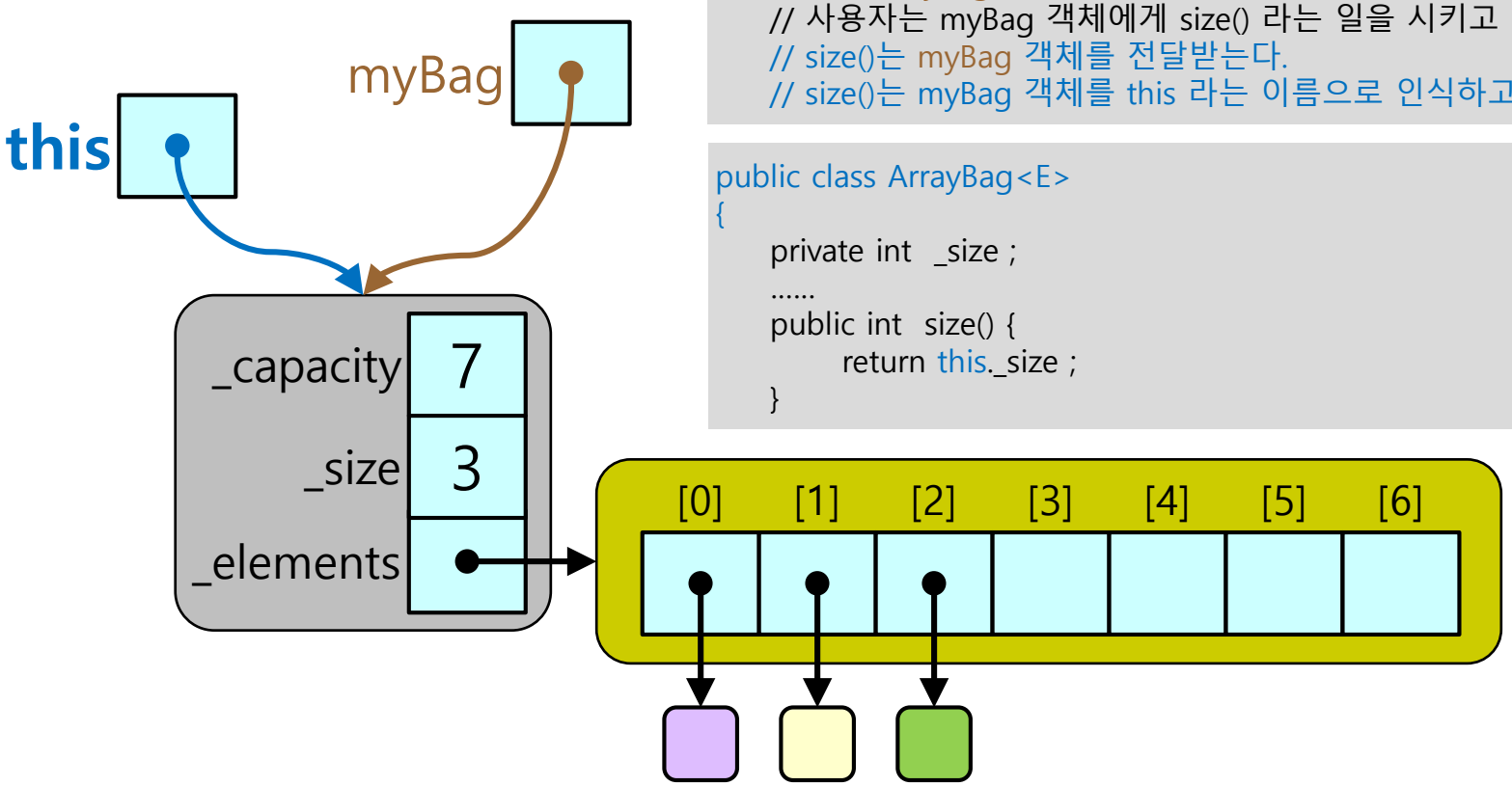
객체 생성은 여러가지 방법으로 할 수 있다.
생성자의 이름은 모두 같다.

멤버 함수에서의 "this" 는?

- 사용자가 class 의 멤버 함수에게 전달한, 그래서 멤버함수가 일을 해 주어야 할 객체를 가리킨다.
  - 생성자 안에서는 생성된 객체를 소유

```
ArrayBag<Coin>   myBag = new ArrayBag<Coin>() ;
    ......
int count = myBag.size() ;
    // 사용자는 myBag 객체에게 size() 라는 일을 시키고 있다.
    // size()는 myBag 객체를 전달받는다.
    // size()는 myBag 객체를 this 라는 이름으로 인식하고 일을 한다.
```

```
public class ArrayBag<E>
{
    private int  _size ;
    ......
    public int  size() {
        return this._size ;
    }
```



ArrayBag: 상태 알아보기

```
public class  ArrayBag<E>
{
    // 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.capacity() ) ;
    }
```

ArrayBag: 상태 알아보기

```
// 상태 알아보기
.....
private int indexOf (E anElement)
{
    int  foundIndex = -1 ;
    for ( int i = 0 ;  i < this.size() && foundIndex < 0 ; i++ ) {
        if ( this.elements()[i].equals(anElement) ) {
            foundIndex = i ;
        }
    }
    return  foundIndex ;
}

public boolean  doesContain (E anElement)
{   // 주어진 원소가 Bag 에 있는지 알려준다

    return  ( this.indexOf(anElement) >= 0 ) ;
}
```

ArrayBag: 상태 알아보기

```
// 상태 알아보기
.....

// indexOf() 를 사용하지 않고 구현할 수도 있다.
public boolean  doesContain (E anElement)
{
    int  foundIndex = -1 ;
    for ( int i = 0 ;  i < this.size() && foundIndex < 0 ; i++ ) {
        if ( this.elements()[i].equals(anElement) ) {
            foundIndex = i ;
        }
    }
    return  (foundIndex >= 0) ;
}
```

ArrayBag: 상태 알아보기

```
// 상태 알아보기
.....

public int  frequencyOf (E 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 int  frequencyOf (E 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  frequenc
}
```

<두 객체가 "같다"는 의미는?>

Java에서 "equals()" 는 모든 객체에 대해 사용가능.
(예) x.equals(y) // 객체 x 에게 x 가 y 와 같은지를 검사시킨다.

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

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

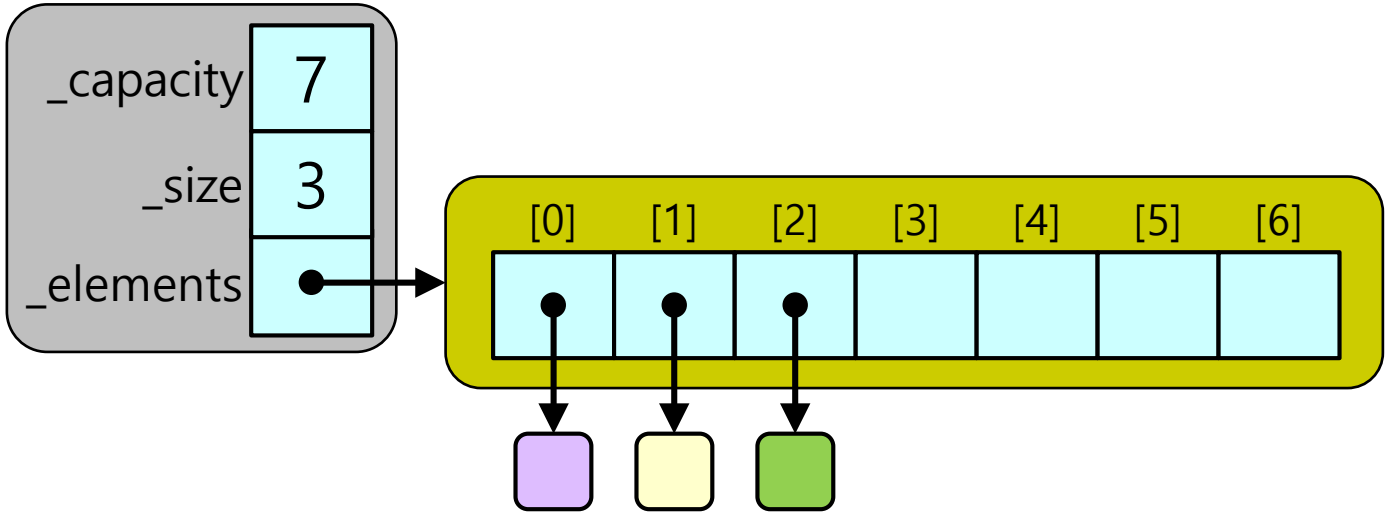
ArrayBag: 내용 알아보기

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

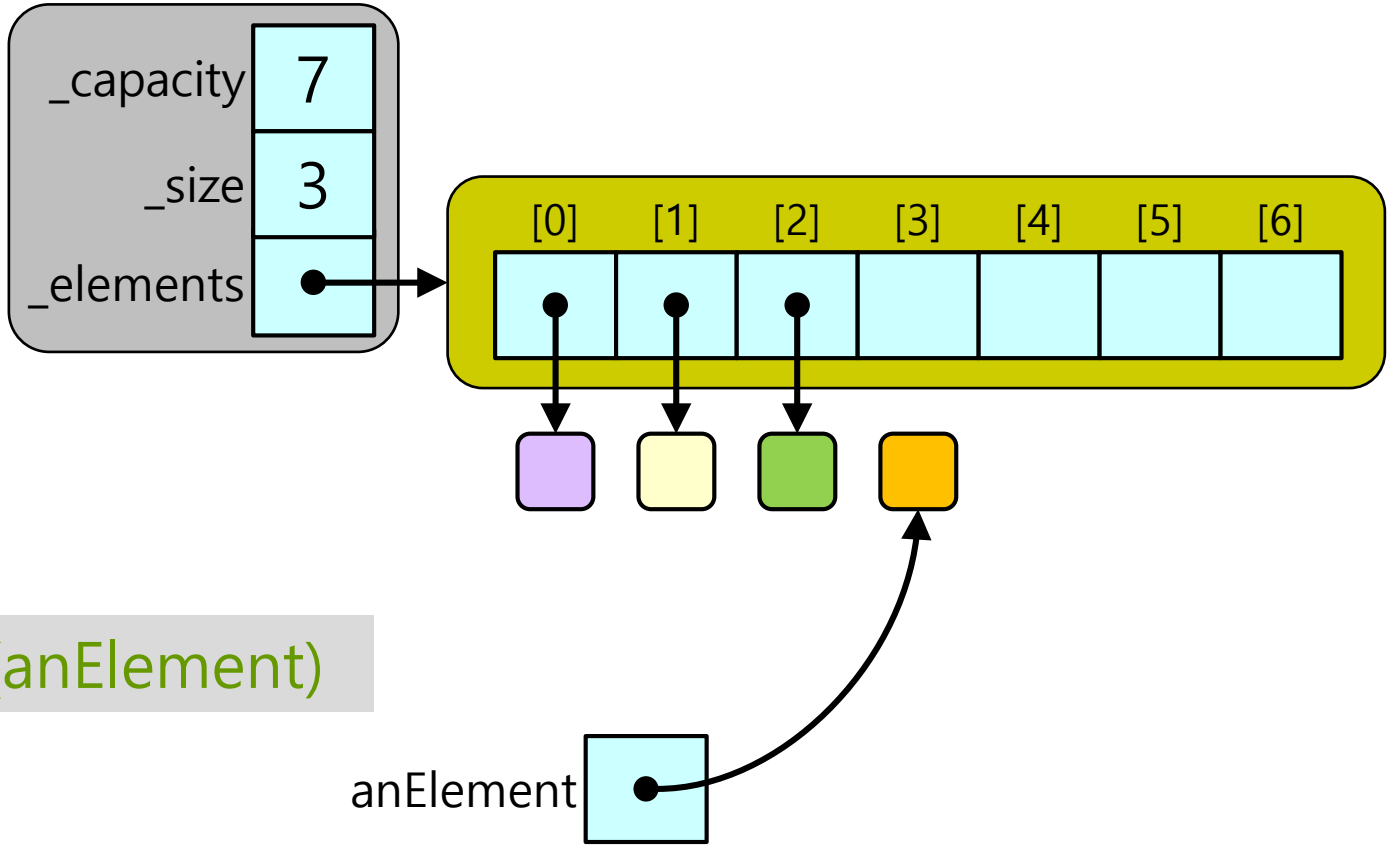
ArrayBag: add()

```
// 내용 바꾸기
public boolean  add (E 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.setSize (this.size()+1) ;
        return  true ;
    }
}
```

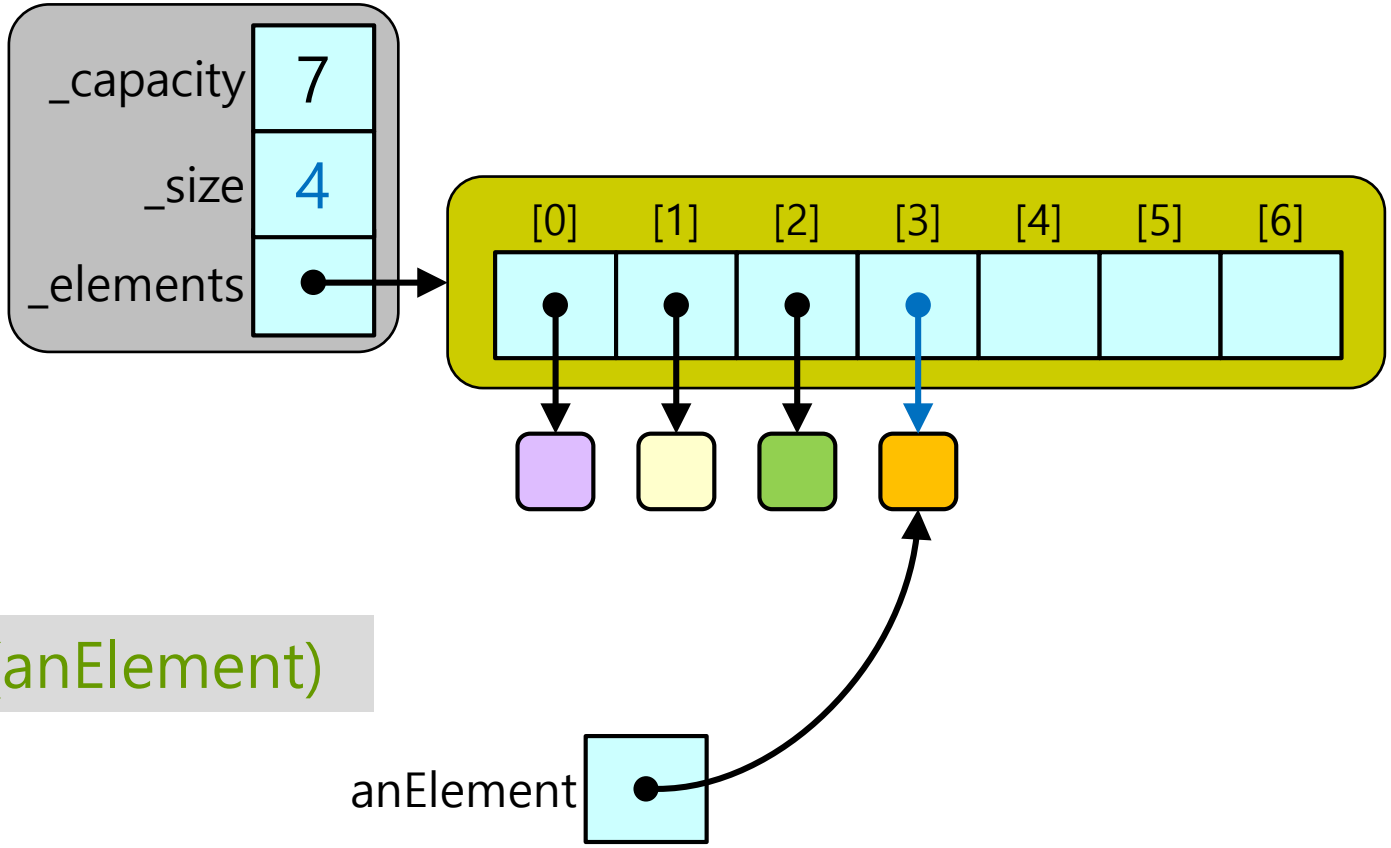
ArrayBag: add()



ArrayBag: add()



ArrayBag: add()

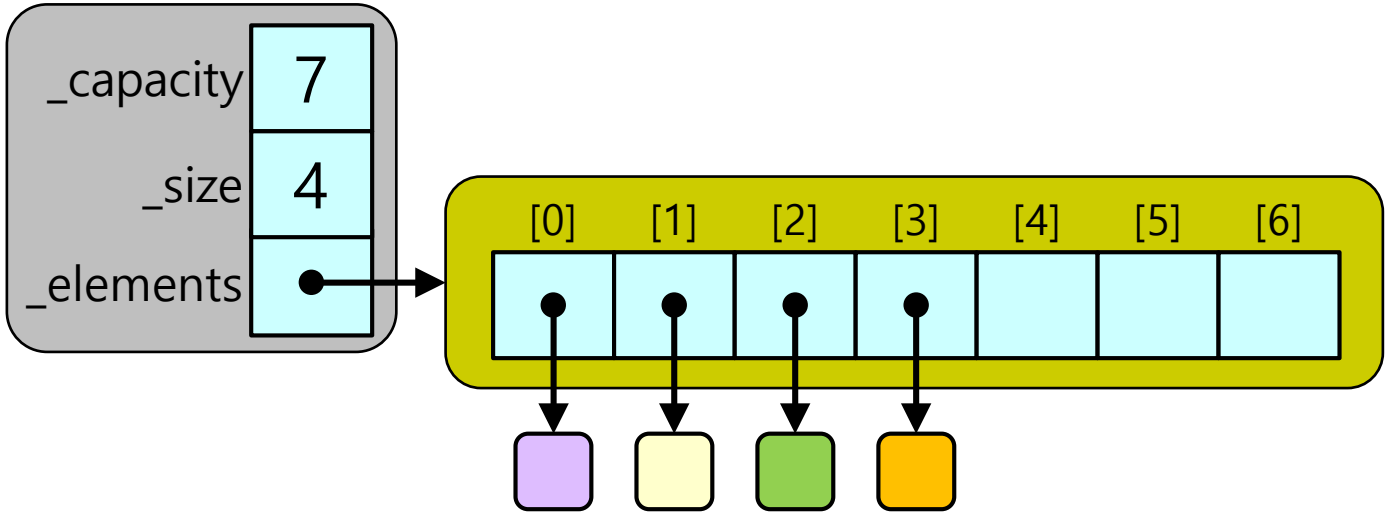


ArrayBag: removeAny()

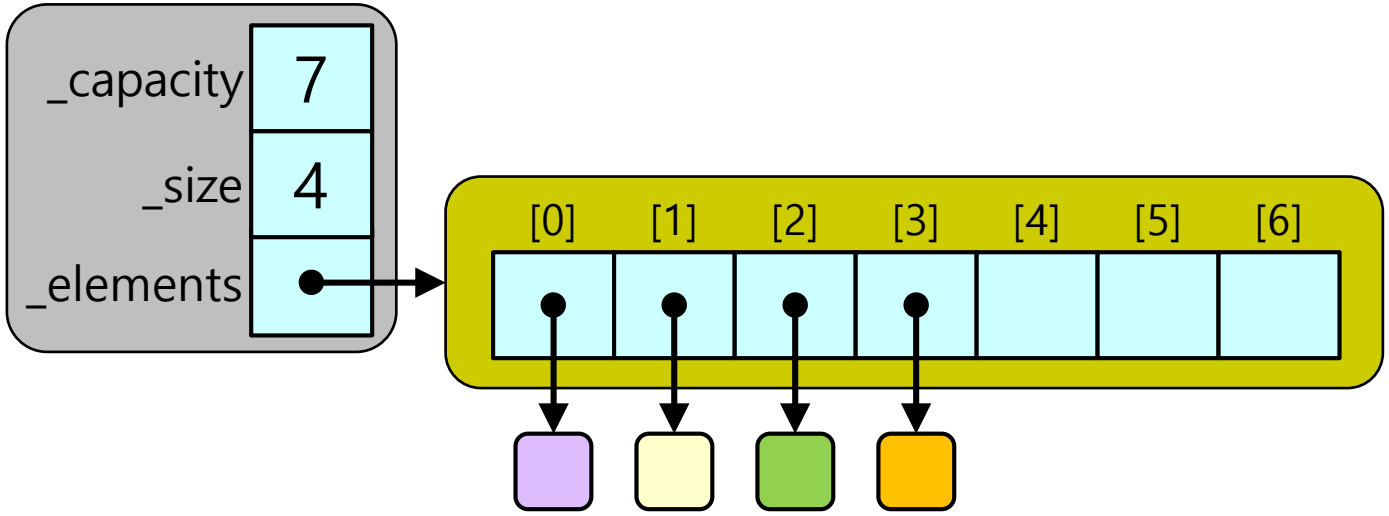
```
// 내용 바꾸기
.....
public E  removeAny ()
{   // Bag 에서 아무 원소 하나를 제거하여 얻어낸다
    if ( this.isEmpty() ) {
        // 가방이 비어 있으므로 제거 할 원소가 없다
        return  null ;
    }
    else {
        // 원소가 존재하므로 아무거나 하나 제거하여 얻는다
        // 맨 마지막 원소를 제거하여 얻기로 한다
        E removedElement = this.elements()[this.size()-1] ;
        this.elements()[this.size()-1] = null ; // 필요한 이유는?
        this.setSize (this.size()-1) ;
        return  removedElement ;
    }
}
```

맨 앞 원소를 제거한다면 어떻게 해야 할까?
어느 편이 효율적일까?

ArrayBag: removeAny()



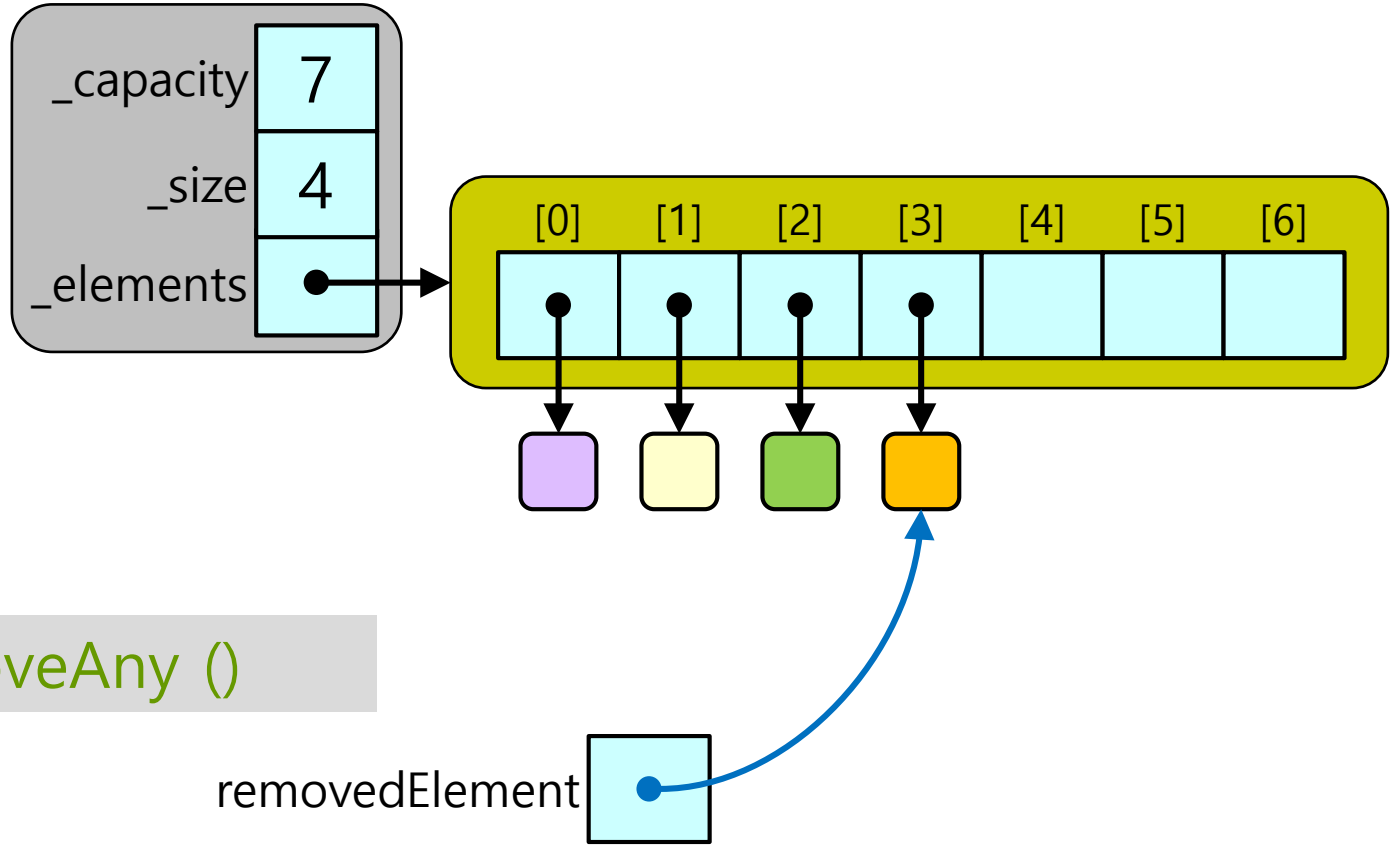
ArrayBag: removeAny()



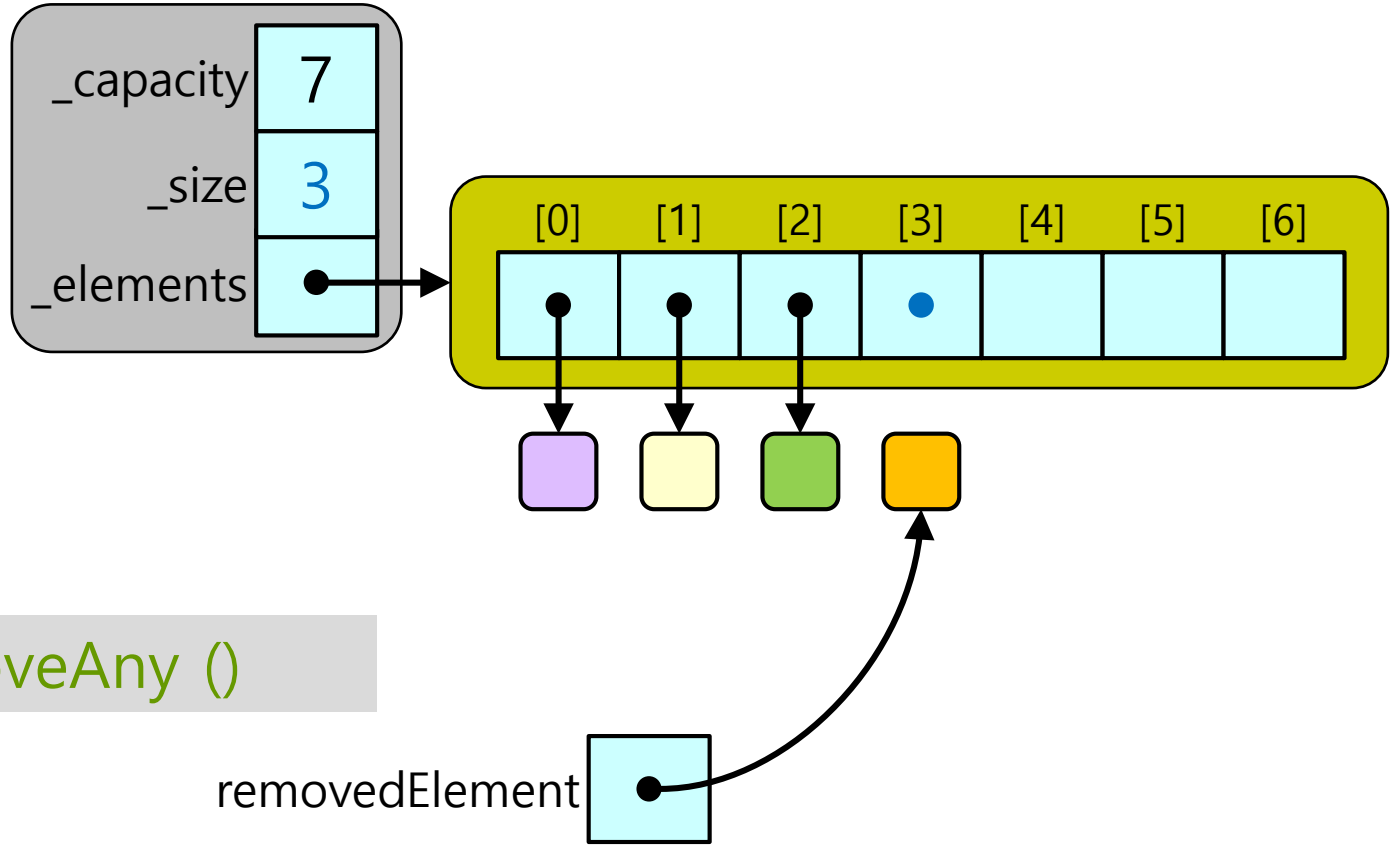
removeAny ()

removedElement

ArrayBag: removeAny()



ArrayBag: removeAny()



ArrayBag: remove()

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

     int  foundIndex = -1 ;
     boolean  found = false ;
     // 단계 1: 주어진 원소의 위치를 찾는다
     for ( int i= 0 ; i < this.size() && ! found ; i++ ) {
          if ( this.elements()[i].equals(anElement) ) {
               foundIndex = i ;
               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 null 로!
          this.setSize (this.size()-1) ;
          return  true ;
     }
}
```

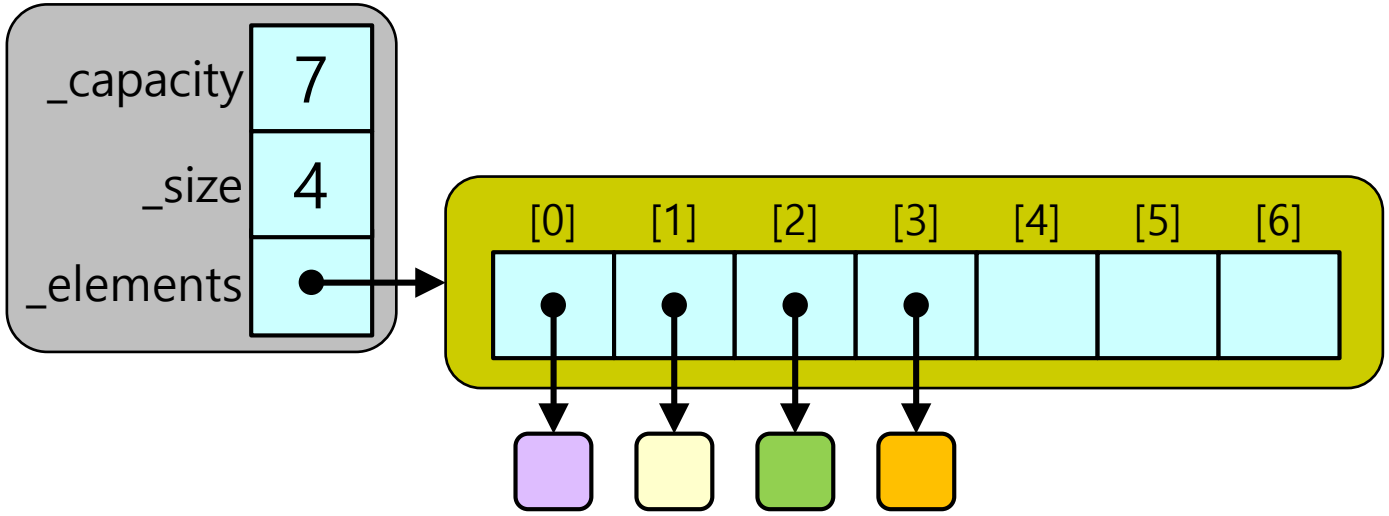
ArrayBag: remove()

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

     int  foundIndex = -1 ;
     boolean  found = false ;
     // 단계 1: 주어진 원소의 위치를 찾는다
     for ( int i= 0 ; i < this.size() && foundIndex < 0 ; i++ ) {
          if ( this.elements()[i].equals(anElement) ) {
               foundIndex = i ;
               found = true ;
          }
     }

     // 단계 2: 삭제된 원소 이후의 모든 원소를 앞쪽으로 한 칸씩 이동시킨다.
     if ( foundIndex < 0 ) {
          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 null 로!
          this.setSize (this.size()-1) ;
          return  true ;
     }
}
```

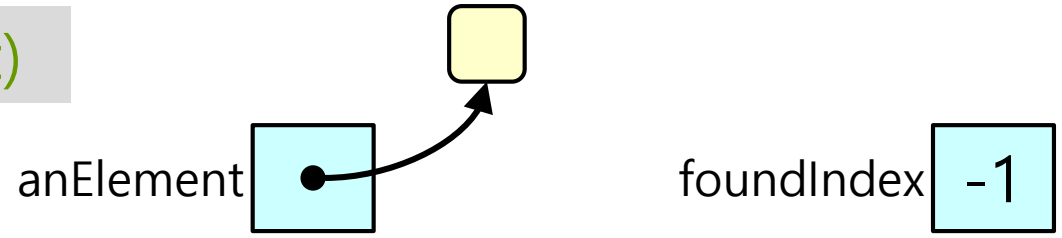
ArrayBag: remove() (Using indexOf())

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

     // 단계 1: 주어진 원소의 위치를 찾는다
     int  foundIndex = this.indexOf (anElement) ;

     // 단계 2: 삭제된 원소 이후의 모든 원소를 앞쪽으로 한 칸씩 이동시킨다.
     if ( foundIndex < 0 ) {
          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 null 로!
          this.setSize (this.size()-1) ;
          return  true ;
     }
}
```

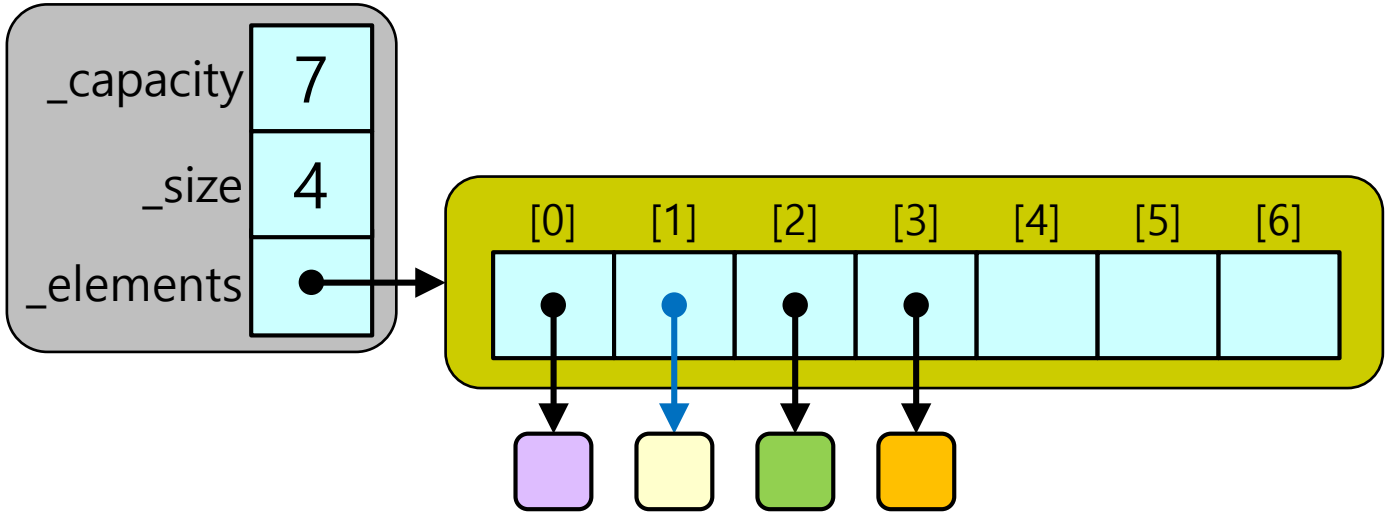
ArrayBag: remove()



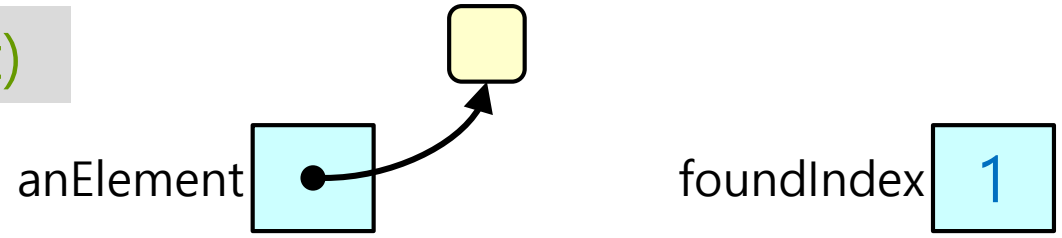
remove (anElement)



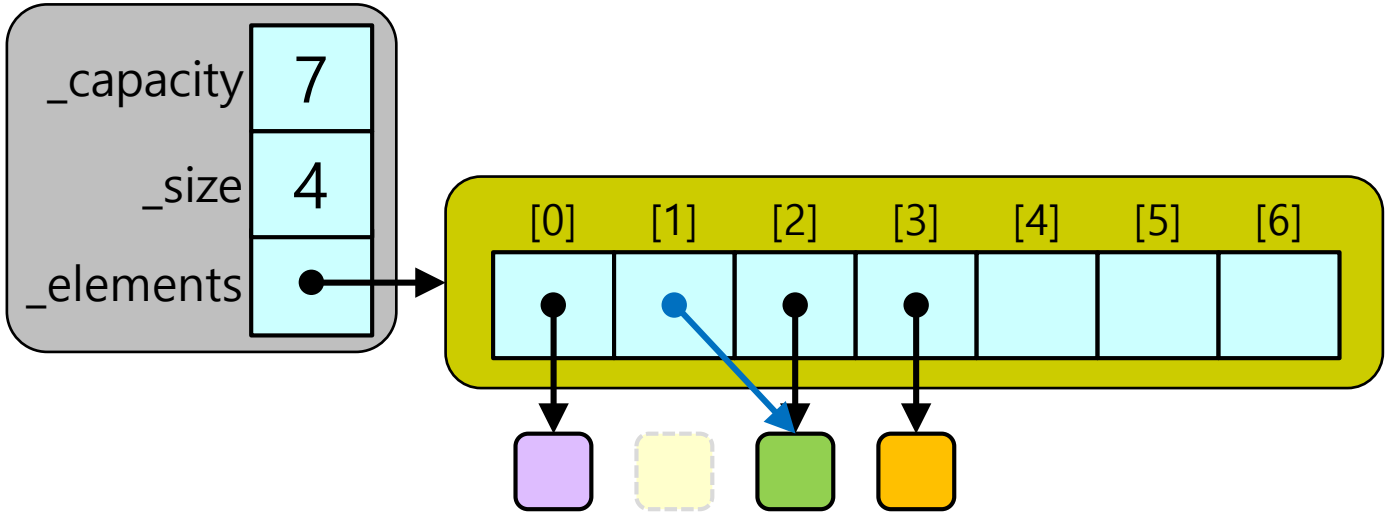
ArrayBag: remove()



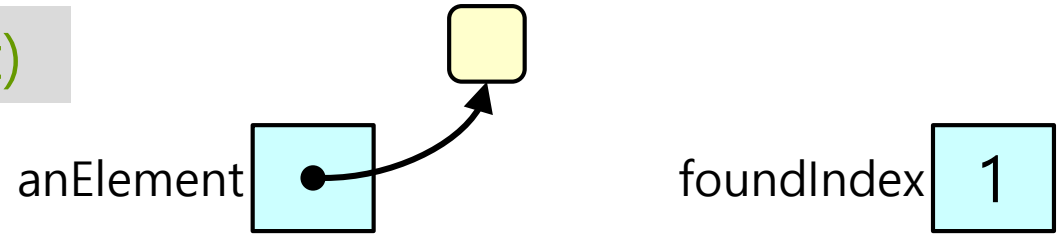
remove (anElement)



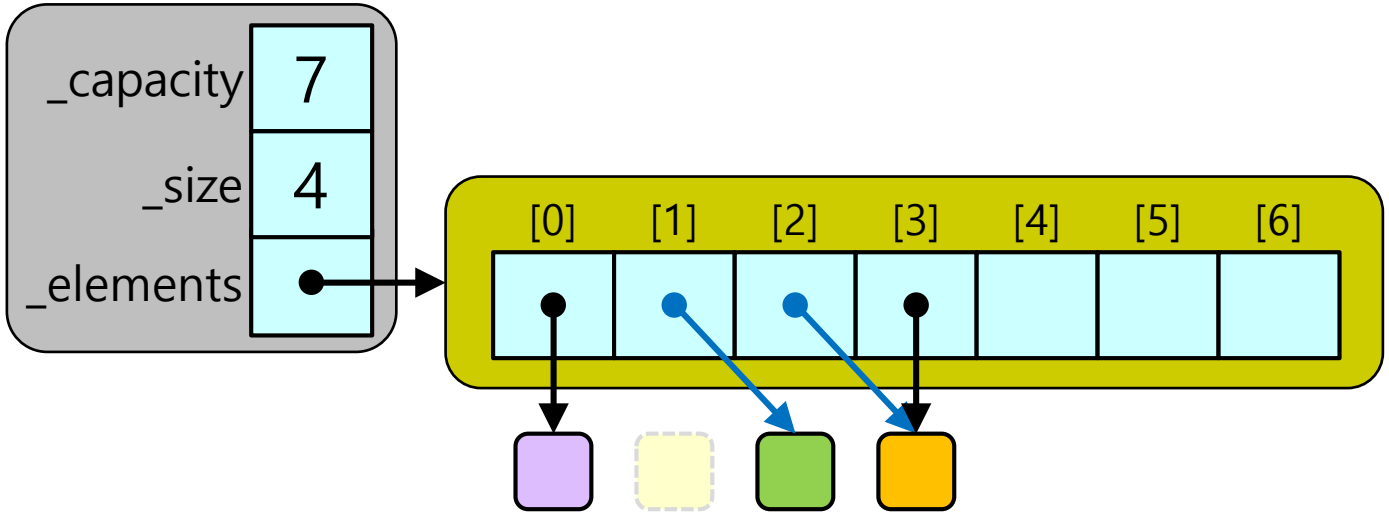
ArrayBag: remove()



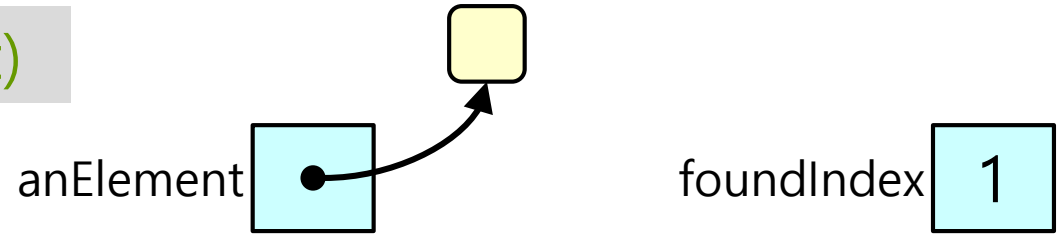
remove (anElement)



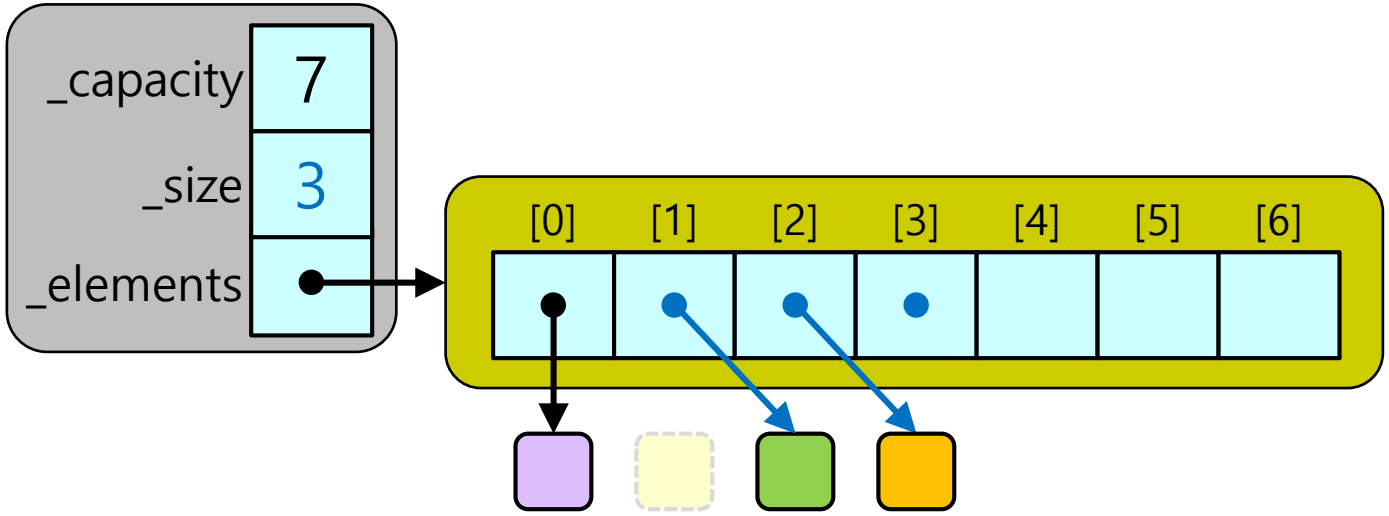
ArrayBag: remove()



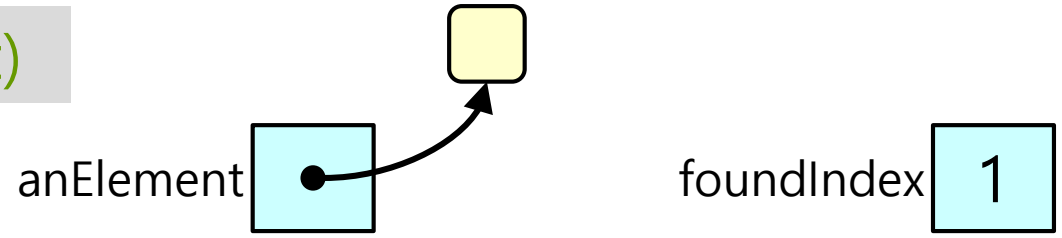
remove (anElement)



ArrayBag: remove()



remove (anElement)



ArrayBag: clear()

```
// 내용 바꾸기
.....

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

Class "Coin" for coins

Class "Coin"

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

    // 비공개 인스턴스 변수
    private int              _value ;

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

Class "Coin"에서의 "equals()"

```
public class  Coin
{
    ……

    // 생성자
    public   Coin ()
    {
        this.setValue (Coin.DEFAULT_VALUE) ;
    }

    public   Coin (int givenValue)
    {
        this.setValue (givenValue) ;
    }

    public boolean  equals (Coin aCoin)
    {
        return  ( this.value() == aCoin.value() ) ;
    }

} // End of class "Coin"
```

순차 검색
(Sequential Search)

Sequential Search: Version 1

```
public boolean  doesContain (E 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.size() 의 값을 n 이라고 하면…

Sequential Search: Version 2

```
public boolean  doesContain (E 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 ;
}
```

- 이렇게 약간은 간결하게도...

Sequential Search: Version 2

```
public boolean  doesContain (E anElement)
{

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

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

Fast Enumeration for Array in Java

```
public boolean  doesContain (Element anElement)
{

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

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

실습: 동전 가방

실습: Coin Bag

- 입력 메뉴에서 선택된 일을 한다
  - Coin 삽입
  - 임의의 coin 삭제
  - 주어진 coin 삭제
  - 주어진 coin 이 가방 안에 있는지?
  - 주어진 coin 이 가방 안에 몇 개 있는지?
  - 코인의 종류별 개수
  - Coin 가방 비우기
- ArrayBag<Coin> coinBag ;

End of

"ArrayBag"

© J.-H. Kang, CNU

© J.-H. Kang, CNU