리스트 (List)

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

Class “List”

리스트

- 원소들이 순서 있게 나열
  - Bag과 Set의 원소는 순서가 없다.
  - 필요에 따라 정렬되어 있을 수 있다.
- 원소가 중복될 수 있다.
  - Set에서는 원소가 중복될 수 없다.

- 예
  - 학번 순으로 나열되어 있는 우리 학과 학생들

List의 공개함수

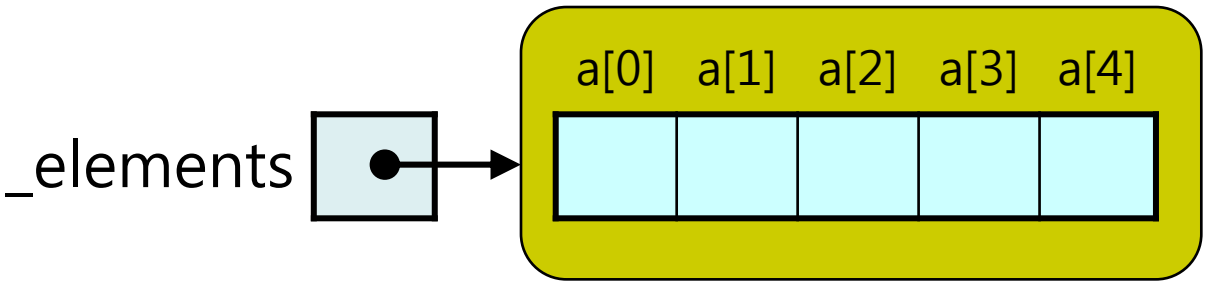
- List 객체 사용법

```
public          List () { }

public boolean  isEmpty () { }
public boolean  isFull () { }
public boolean  size () { }

public Element  elementAt (int aPosition) { }
public Element  last () { }
public int      positionOf (Element anElement) { }
public boolean  doesContain (Element anElement) { }

public boolean  addTo (Element anElement, int aPosition) { }
public boolean  addToLast (Element anElement) { }
public Element  removeFrom (int aPosition) { }
public Element  removeLast () { }
public boolean  replaceAt (Element anElement, int aPosition) { }
public void     clear() { }
```

Class “ArrayList”

리스트로서의 "ArrayList"

- 추상적인 List를 Array를 이용하여 구현
  - ArrayList aList = new ArrayList() ;
    ..... // aList를 이용하여 일을 한다



ArrayList의 공개함수

- ArrayList 객체 사용법:

```
public          ArrayList () { }
public          ArrayList (int initialGivenCapacity) { }

public boolean  isEmpty () { }
public boolean  isFull () { }
public boolean  size () { }

public T        elementAt (int aPosition) { }
public T        last () { }
public int      positionOf (T anElement) { }
public boolean  doesContain (T anElement) { }

public boolean  addTo (T anElement, int aPosition) { }
public boolean  addToLast (T anElement) { }
public T        removeFrom (int aPosition) { }
public T        removeLast () { }
public boolean  replaceAt (T anElement, int aPosition) { }
public void     clear () { }
```

Class “ArrayList”의 구현

ArrayList의 비공개 인스턴스 변수

```
public class ArrayList<T>
{
    // 비공개 인스턴스 변수
    private static final int DEFAULT_INITIAL_CAPACITY = 25;
    private int              _maxSize ;
    private int              _size ;
    private T[]              _elements ;
```

ArrayList의 비공개함수

- 여러 함수에서 공통으로 자주 사용
  - private boolean anElementDoesExistAt (int aPosition) { }
  - private void makeRoomAt (int aPosition) { }
  - private void removeGapAt (int aPosition) { }

ArrayList의 생성자

```
public class ArrayList < T>
{
    // 비공개 인스턴스 변수
    ......

    // 생성자
    public ArrayList ( )
    {
        this ( ArrayList.DEFAULT_INITIAL_CAPACITY ) ;
    }

    public ArrayList ( int initialGivenCapacity )
    {
        @SuppressWarnings("Unchecked") ;
        this._elements = (T[ ]) new Object[initialGivenCapacity] ;
        this._maxSize = initialGivenCapacity ;
        this._size = 0 ;
    }
```

ArrayList의 생성자

```
public class ArrayList < T>
{
    // 비공개 인스턴스 변수
    ......

    // 생성자
    public ArrayList ( )
    {
        this ( ArrayList.DEFAULT_INITIAL_CAPACITY ) ;
    }

    public ArrayList ( int initialGivenCapacity )
    {
        @SuppressWarnings("Unchecked") ;
        this._elements = (T[ ]) new Object[initialGivenCapacity] ;
        this._maxSize = initialGivenCapacity ;
        this._size = 0 ;
    }
```

여기서의 "**this**"는 객체 생성자

ArrayList: 상태 알아보기

```
public calss ArrayList<T>
{
    // 비공개 인스턴스 변수
    ......


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

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

    public int  size()
    {
        return  this._size ;
    }
```

ArrayList: 내용 알아보기

```
public T  elementAt (int aPosition)
{
    if ( this.anElementDoesExistAt (aPosition) ) {
        return  this._elements[aPosition] ;
    }
    else {
        return  null ;
    }
}

private boolean  anElementDoesExistAt (int aPosition)
{
    return  ((aPosition >= 0) && (aPosition < this._size)) ;
}
```

ArrayList: 내용 알아보기

```
// Version 1
public T  last ()
{
    return this.elementAt (this._size-1) ;
    // 아래와 같이 하면 안되는 이유는?
    // return this._elements[this._size-1] ; // 리스트가 empty라면?
}


// version 2
public T  last ()
{
    if ( this.isEmpty() ) {
        return   null ;
    }
    else {
        return   this._elements[this._size-1];
}
```

ArrayList: 내용 알아보기

```
public int  positionOf (T anElement)
{
    // 원소 anElement 가 리스트 안에 존재하면 해당 위치를 돌려준다
    // 존재하지 않으면 -1을 돌려준다
    for ( int position = 0 ; position < this._size ; position ++) {
        if ( this._elements[position].equals(anElement) ) {
            return position ;
        }
    }
    return -1 ; // 주어진 원소 anElement가 리스트 안에 없다
}

public boolean  doesContain (T anElement)
{
    return  (this.positionOf(anElement) != -1) ;
}
```

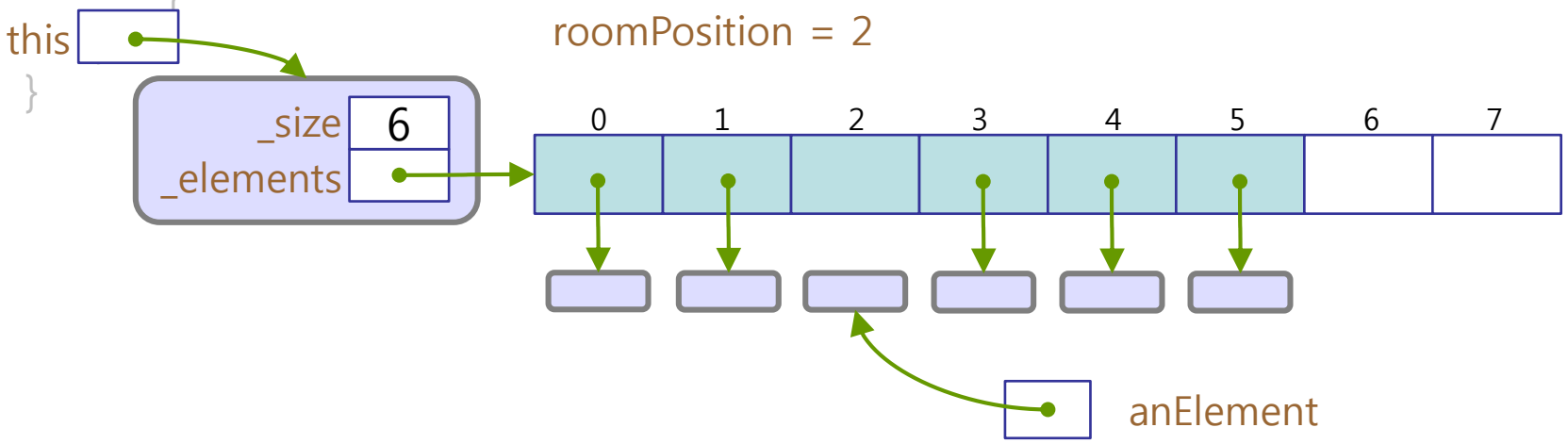
ArrayList: 원소 삽입하기

```
// 원소 삽입
public boolean  addTo (T anElement, int aPosition)
{
    if ( this.isFull() ) {
        return false ;
    }
    else {
        if ( (aPosition >= 0) && (aPosition <= this._size ) ) {
            this.makeRoomAt (aPosition) ;
            this._elements[aPosition] = anElement ;
            this._size++;
            return  true ;
        }
        else {
            return  false ; // 잘못된 삽입 위치
        }
    }
}

public boolean  addToLast (T anElement)
{
    return  this.addTo(anElement, this._size) ;
}
```

"<" 가 아닌 "<="을 사용한 이유는?

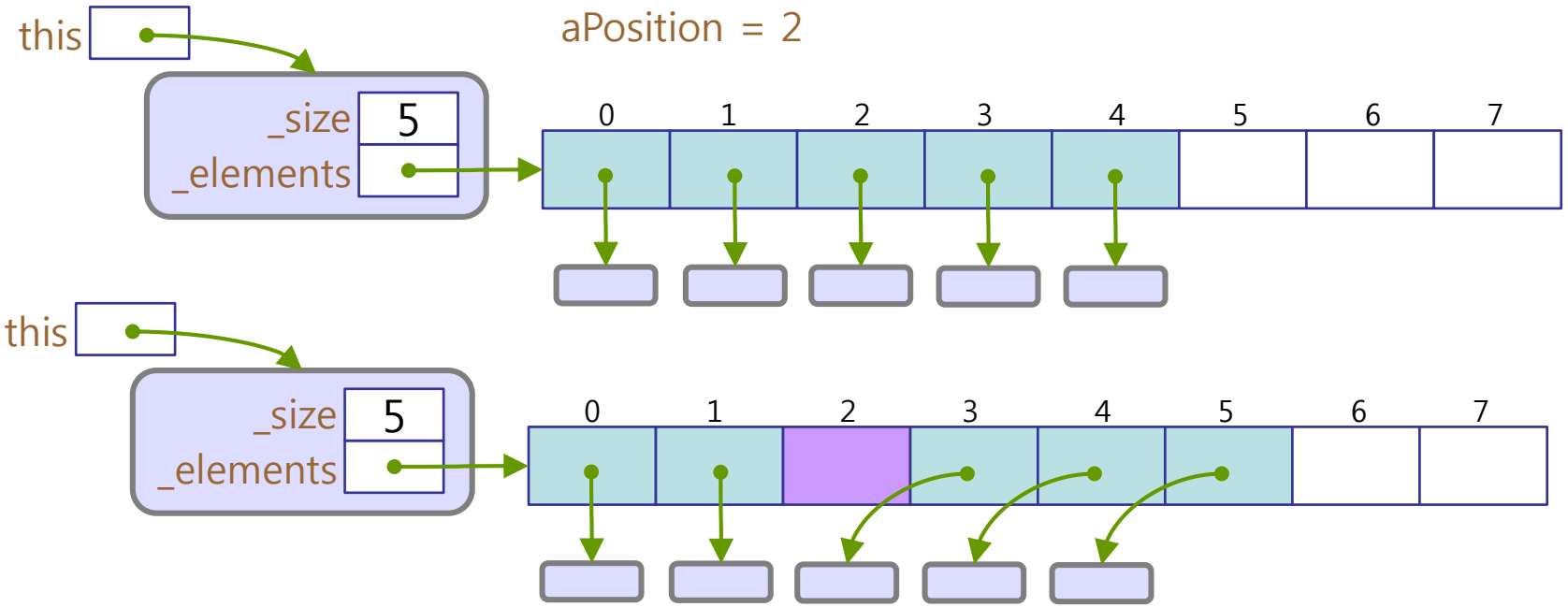
ArrayList: 원소 삽입하기

```
public boolean   addTo (T anElement, int aPosition)
{
    ...
    else {
        if ( (aPosition >= 0) && (aPosition <= this._size ) ) {
            this.makeRoomAt (aPosition) ;
            this._elements[aPosition] = anElement ;
            this._size++;
            return  true ;
        }
        else {
            return  false ; // 잘못된 삽입 위치
        }
}
```



ArrayList: 삽입 공간 만들기

```
private void makeRoomAt ( int aPosition )
{
    for ( int i= this._size ; i > aPosition; i-- ) {
        this._elements[i] = this._elements[i-1] ;
    }
}
```



ArrayList: 원소 삭제하기 (Version 1)

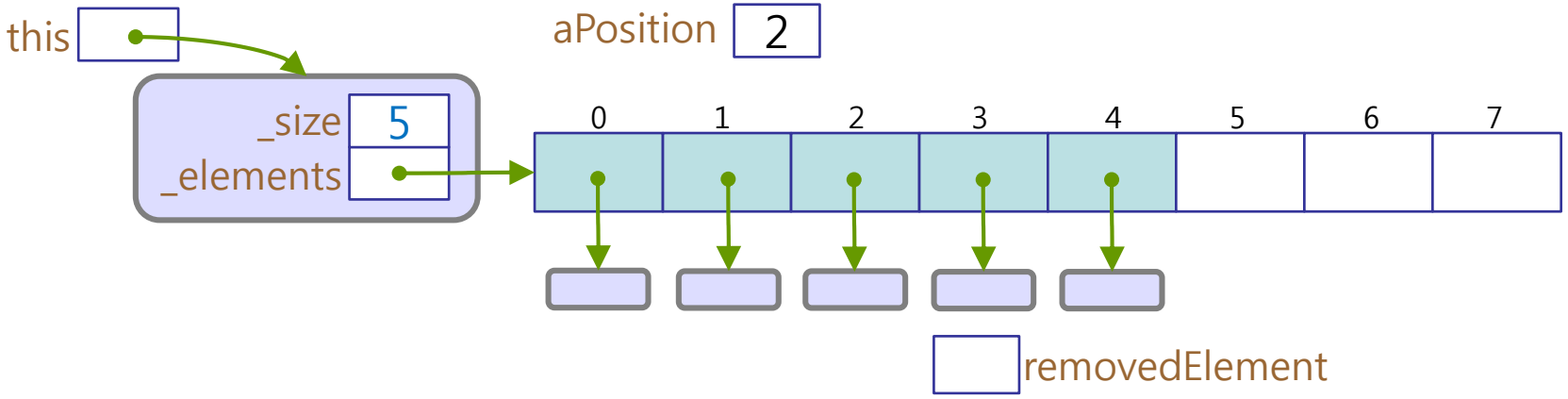
```
// Version 1:
public T  removeFrom ( int aPosition )
{
    // 주어진 위치 aPosition에 원소가 없으면 null을 return한다
    // 원소가 있으면 리스트에서 제거하여 return 한다.
    if ( this.isEmpty() ) { // 이 검사가 꼭 필요한가?
        return null ;
    else {
        T  removedElement = null ;
        if ( this.anElementDoesExistAt (aPositon) ) {
            removedElement = this._elements[aPosition] ;
            this.removeGapAt (aPosition) ;
            this._size-- ;
        }
        return  removedElement ;
    }
}
```

ArrayList: 원소 삭제하기 (Version 2)

```
// Version 2:
public T  removeFrom ( int aPosition )
{
    // 주어진 위치 aPosition에 원소가 없으면 null을 return한다
    // 원소가 있으면 리스트에서 제거하여 return 한다.
    T  removedElement = null ;
    if ( this.anElementDoesExistAt (aPositon) ) {
        removedElement = this._elements[aPosition] ;
        this.removeGapAt (aPosition) ;
        this._size-- ;
    }
    return  removedElement ;
}
```

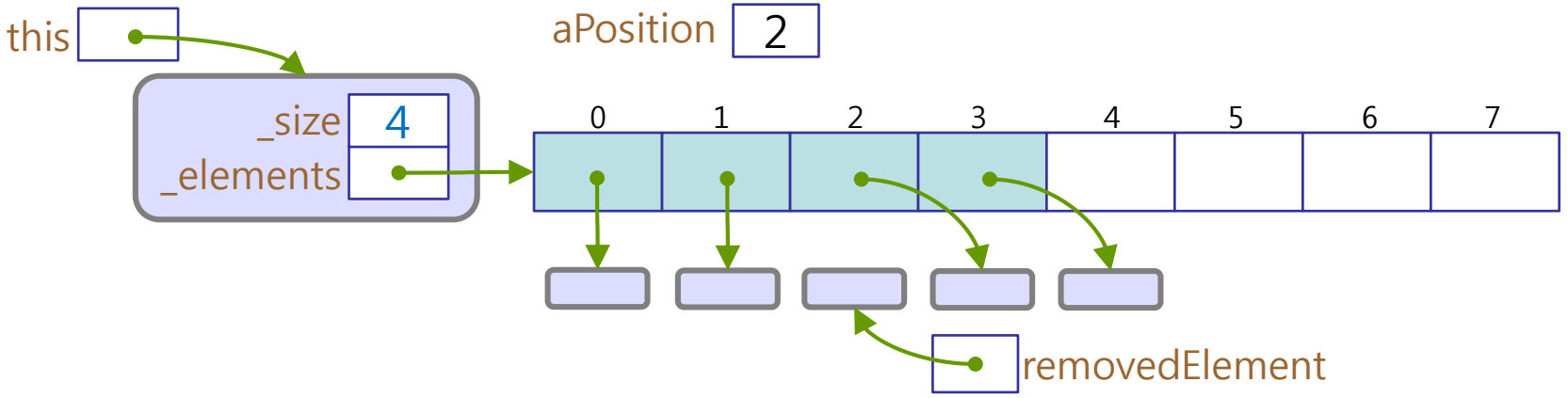
ArrayList: 원소 삭제하기 [전]

```
public T  removeFrom ( int aPosition )
{
      T  removedElement = null ;
      if (  this.anElementDoesExistAt (aPositon) ) {
            T  removedElement = this._elements[aPosition] ;
            this.removeGapAt (aPosition) ;
            this._size-- ;
      }
      return  removedElement ;
}
```



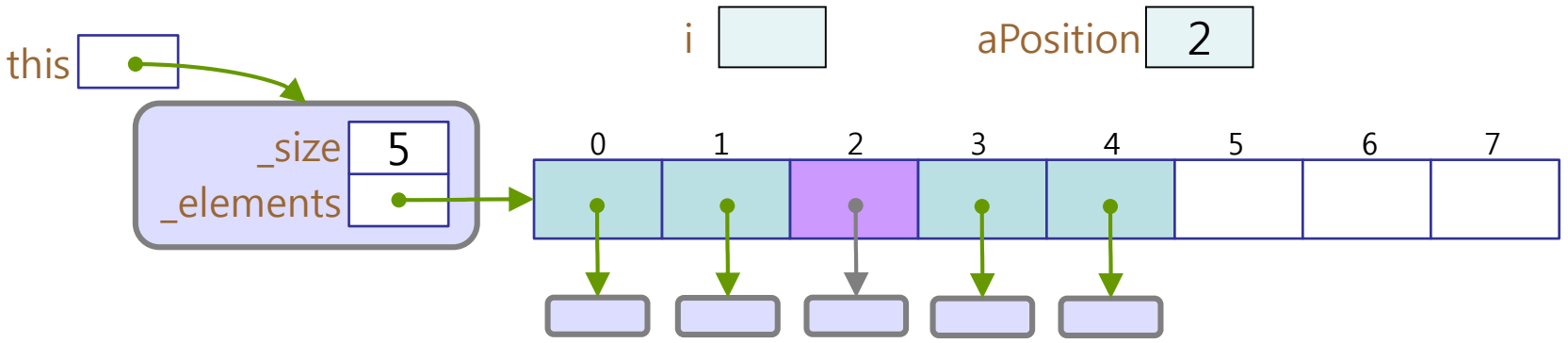
ArrayList: 원소 삭제하기 [후]

```
public T  removeFrom ( int aPosition )
{
      T  removedElement = null ;
      if (  this.anElementDoesExistAt (aPositon) ) {
            T  removedElement = this._elements[aPosition] ;
            this.removeGapAt (aPosition) ;
            this._size-- ;
      }
      return  removedElement ;
}
```



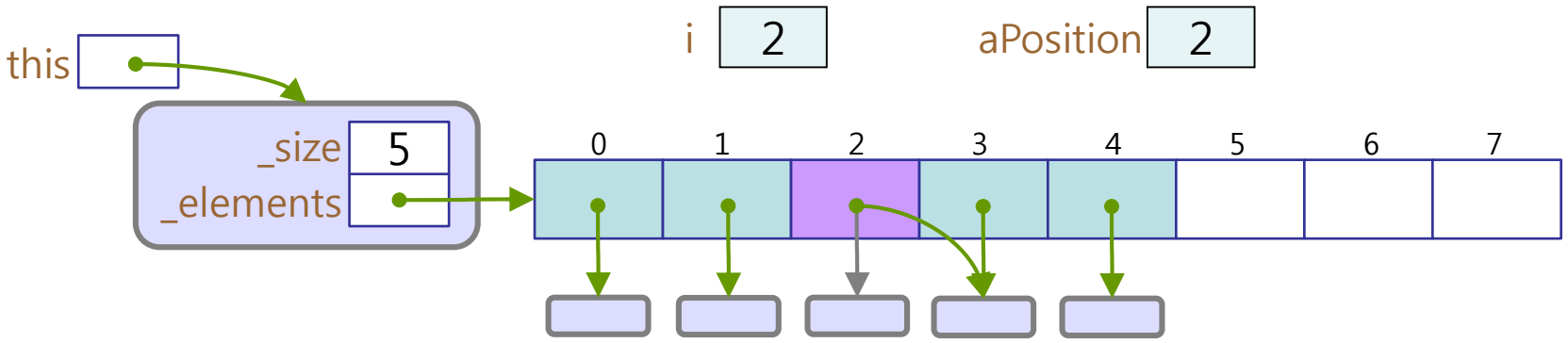
ArrayList: 삭제된 빈 공간 제거하기[1]

```
private void removeGapAt ( int aPosition )
{
    // 리스트는 empty가 아님: 언제나 (this._size > 0)
    // aPosition은 valid: 언제나 (0 <= aPosition < this._size)

    for ( int i = aPosition ; i < this._size-1 ; i++ ) {
        this._elements [i] = this._elements [i+1] ;
    }
    this._elements[this._size-1] = null ;
}
```



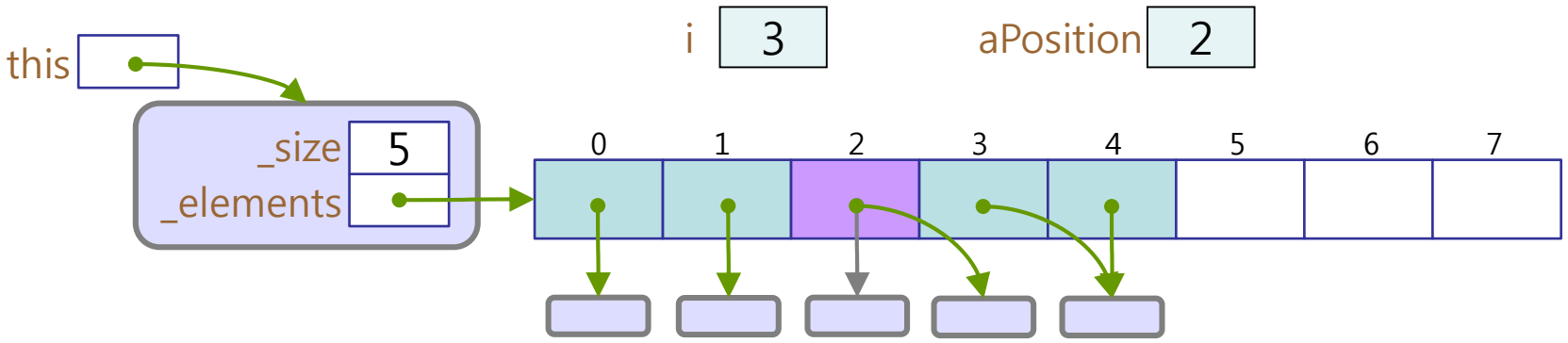
ArrayList: 삭제된 빈 공간 제거하기[2]

```
private void removeGapAt ( int aPosition )
{
    // 리스트는 empty가 아님: 언제나 (this._size > 0)
    // aPosition은 valid: 언제나 (0 <= aPosition < this._size)

    for ( int i = aPosition ; i < this._size-1 ; i++ ) {
        this._elements [i] = this._elements [i+1] ;
    }
    this._elements[this._size-1] = null ;
}
```



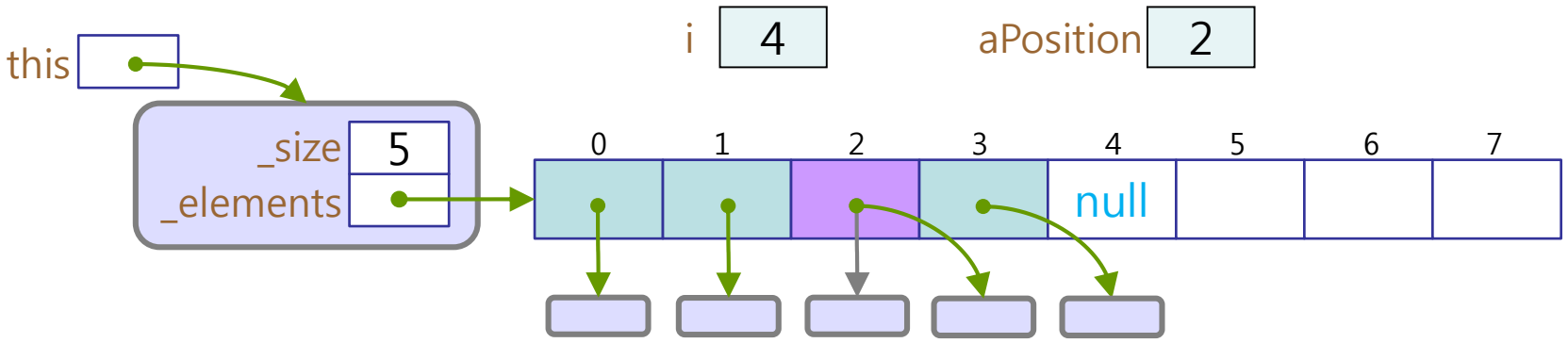
ArrayList: 삭제된 빈 공간 제거하기[3]

```
private void removeGapAt ( int aPosition )
{
    // 리스트는 empty가 아님: 언제나 (this._size > 0)
    // aPosition은 valid: 언제나 (0 <= aPosition < this._size)

    for ( int i = aPosition ; i < this._size-1 ; i++ ) {
        this._elements [i] = this._elements [i+1] ;
    }
    this._elements[this._size-1] = null ;
}
```



ArrayList: 삭제된 빈 공간 제거하기[4]

```
private void removeGapAt ( int aPosition )
{
    // 리스트는 empty가 아님: 언제나 (this._size > 0)
    // aPosition은 valid: 언제나 (0 <= aPosition < this._size)

    for ( int i = aPosition ; i < this._size-1 ; i++ ) {
        this._elements [i] = this._elements [i+1] ;
    }
    this._elements[this._size-1] = null ;
}
```



ArrayList: 마지막 원소 삭제하기

```
public T  removeLast ()
{
    return  removeFrom (this._size-1) ;
}
```

ArrayList: 원소 대체하기

```
// Version 1:
public boolean  replace (T anElement, int aPosition)
{
     if ( this.isEmpty() ) {
          return  false ;
     }
     else {
          if ( this.anElementDoesExistAt (aPosition) ) {
               this._elements[aPosition] = anElement ;
               return  true ;
          }
          else {
               return  false ;
          }
     }
}

// Version 2:
public boolean  replace (T anElement, int aPosition)
{
     if ( this.anElementDoesExistAt (aPosition) ) {
          this._elements[aPosition] = anElement ;
          return  true ;
     }
     else {
          return  false ;
     }
}
```

ArrayList: 리스트 비우기

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

Class “LinkedList”

리스트로서의 "LinkedList"

- 추상적인 List를 연결 체인을 이용하여 구현
  - LinkedList aList = new LinkedList() ;
    ..... // aList 를 이용하여 일을 한다

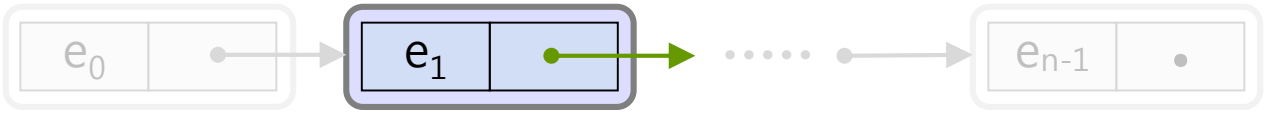
LinkedList의 공개함수

- LinkedList 객체 사용법

```
public              ArrayList () { }

public boolean      isEmpty () { }
public boolean      isFull () { }
public boolean      size () { }

public T            elementAt (int aPosition) { }
public T            last () ;
public int          positionOf (T anElement) { }
public boolean      doesContain (T anElement) { }

public boolean      addTo (T anElement, int aPosition) { }
public boolean      addToLast (T anElement) { }
public T            removeFrom (int aPosition) { }
public T            removeLast () { }
public boolean      replaceAt (T anElement, int aPosition) { }
public void         clear () { }
```

Linked List 구현에 필요한

Class "Node"



Node 의 공개함수

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

Class Node: 비공개 인스턴스 변수

```
public class Node<T>
{
    // 비공개 인스턴스 변수
    private  T          _element ;
    private  Node       _next ;
```

Class "Node"의 구현: 생성자

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

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

    public Node (T anElement, Node aNextNode)
    {
        this._element = anElement ;
        this._next = aNextNode ;
    }
```

Class "Node"의 구현: Getters

```
public class  Node<T>
{
    // 비공개 멤버 변수
    ……

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

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

Class "Node"의 구현: Setters

```
public calss Node<T>
{
    // 비공개 멤버 변수
    ......

    // Getters
    ......

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

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

Class “LinkedList”의 구현

LinkedList: 비공개 인스턴스 변수

```
public class LinkedList<T>
{
    // 비공개 인스턴스 변수
    private int        _size ; // 리스트가 가지고 있는 원소의 개수
    private Node    _head ; // LinkedChain의 맨 앞 노드
```

ArrayList의 비공개함수

- 여러 함수에서 공통으로 자주 사용
  - private boolean anElementDoesExistAt (int aPosition)

LinkedList의 생성자

```
public class LinkedList<T>
{
    // 비공개 인스턴스 변수
    ......

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

LinkedList: 상태 알아보기

```
public calss LinkedList<T>
{
    ......

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

    public boolean  isFull ()
    {
        // 시스템 메모리가 모자라는 경우는 없다고 가정
        return  false ; // 언제나 full 이 아니다
    }

    public int  size  ()
    {
        return  this._size ;
    }
```

LinkedList: 내용 알아보기

```
public T  elementFrom (int aPosition)
{
    if ( this.anElementDoesExistAt (aPosition) ) {
        Node  currentNode = this._head ;
        int  nodeCount = 0 ;
        while (nodeCount < aPosition) {
            currentNode = currentNode.next() ;
            nodeCount++ ;
        }
        return currentNode.element() ;
    }
    else {
        return null ;
    }
}
```

LinkedList: 내용 알아보기

```
public T  last () ;
{
    if  ( this.isEmpty() ) {
        return  null ; // 마지막 원소가 존재할 수 없으므로
    }
    else {
        return  elementFrom(this._size-1) ;
    }
}
```

LinkedList: 내용 알아보기

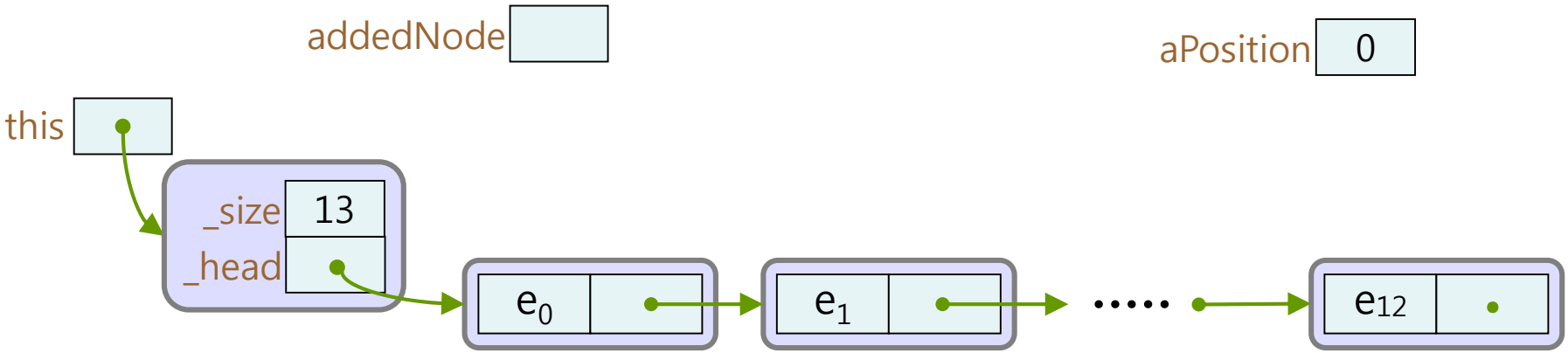
```
public int   positionOf (T anElement)
{ // 순차 검색
    int          position = 0 ;
    Node         currentNode = this._head ;
    while ( currentNode != null &&
            (! currentNode.element().equals(anElement)) )
    {
        position++ ;
        currentNode = currentNode.next() ;
    }
    if ( currentNode == null ) { // Not Found
        position = -1 ; // 존재하지 않으면 -1 을 돌려주기로 한다
    }
    return  position ;
}

public boolean    contains (T anElement) ;
{
    return  (positionOf(anElement)  != -1) ;
}
```

LinkedList: 원소 삽입하기

```
public boolean  addTo (T anElement, int aPosition)
{
    if ( (aPosition >= 0 && (aPosition <= this._size) ) {
        Node    addedNode = new Node (anElement, null) ;
        if ( aPosition == 0 ) {
            addedNode.setNext (this._head) ;
            this._head = addedNode ;
        }
        else {
            Node  previousNode = this._head ;
            for (int i = 1 ; i < givenPosition ; i++) {
                previousNode = previousNode.next() ; // 삽입할 위치의 앞 노드를 찾는다
            }
            addedNode.setNext (previousNode.next()) ;
            previousNode.setNext (addedNode) ;
        }
        this._size++ ;
        return true ;
    }
    else {
        return false ;
    }
}

public boolean  addToLast (T anElement)
{
    return  addTo (anElement, this._size) ;
}
```

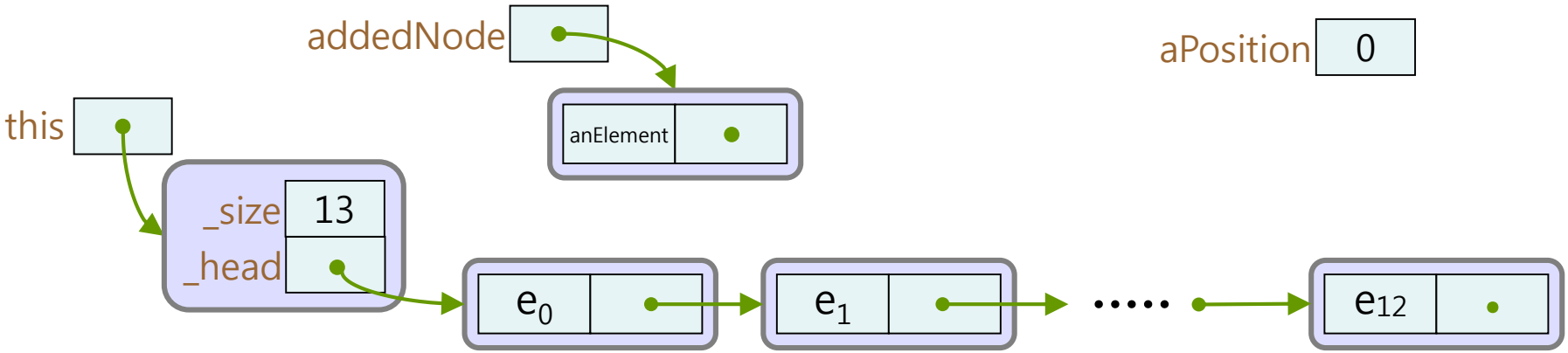
LinkedList: addTo() [1]

```
public boolean    addTo (T anElement, int aPosition)
{
    if ( (aPosition >= 0 && (aPosition <= this._size) ) {
        Node  addedNode = new Node (anElement, null) ;
        if ( aPosition == 0 ) {
            addedNode.setNext (this._head) ;
            this._head = addedNode ;
        }
        else {
            Node  previousNode = this._head ;
```



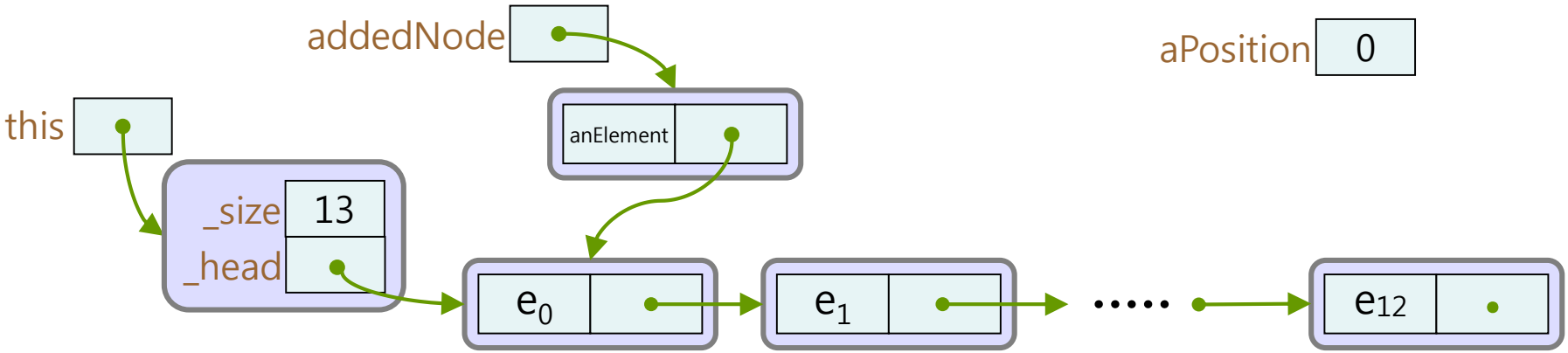
LinkedList: addTo() [2]

```
public boolean    addTo (T anElement, int aPosition)
{
    if ( (aPosition >= 0 && (aPosition <= this._size) ) {
        Node  addedNode = new Node (anElement, null) ;
        if ( aPosition == 0 ) {
            addedNode.setNext (this._head) ;
            this._head = addedNode ;
        }
        else {
            Node  previousNode = this._head ;
```



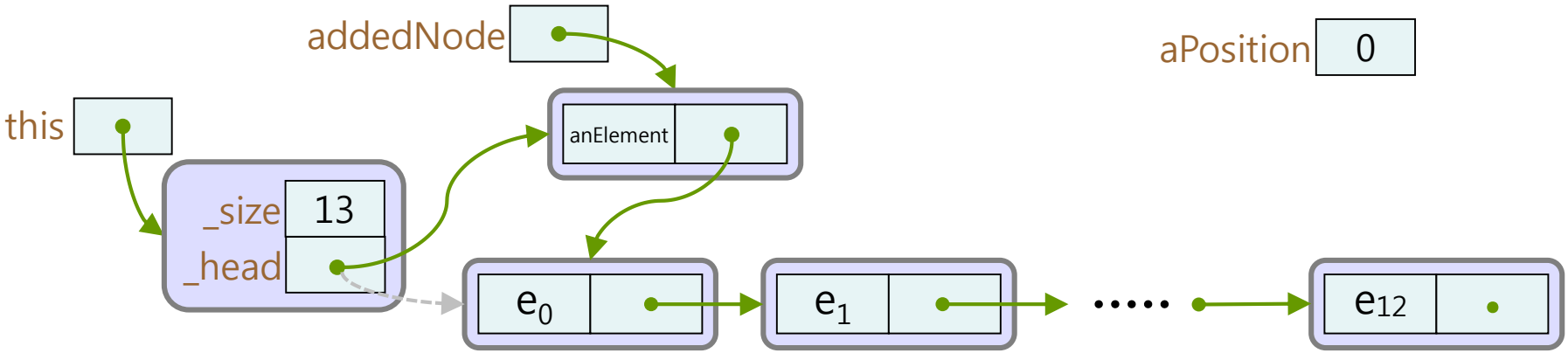
LinkedList: addTo() [3]

```
public boolean    addTo (T anElement, int aPosition)
{
    if ( (aPosition >= 0 && (aPosition <= this._size) ) {
        Node  addedNode = new Node (anElement, null) ;
        if ( aPosition == 0 ) {
            addedNode.setNext (this._head) ;
            this._head = addedNode ;
        }
        else {
            Node  previousNode = this._head ;
```



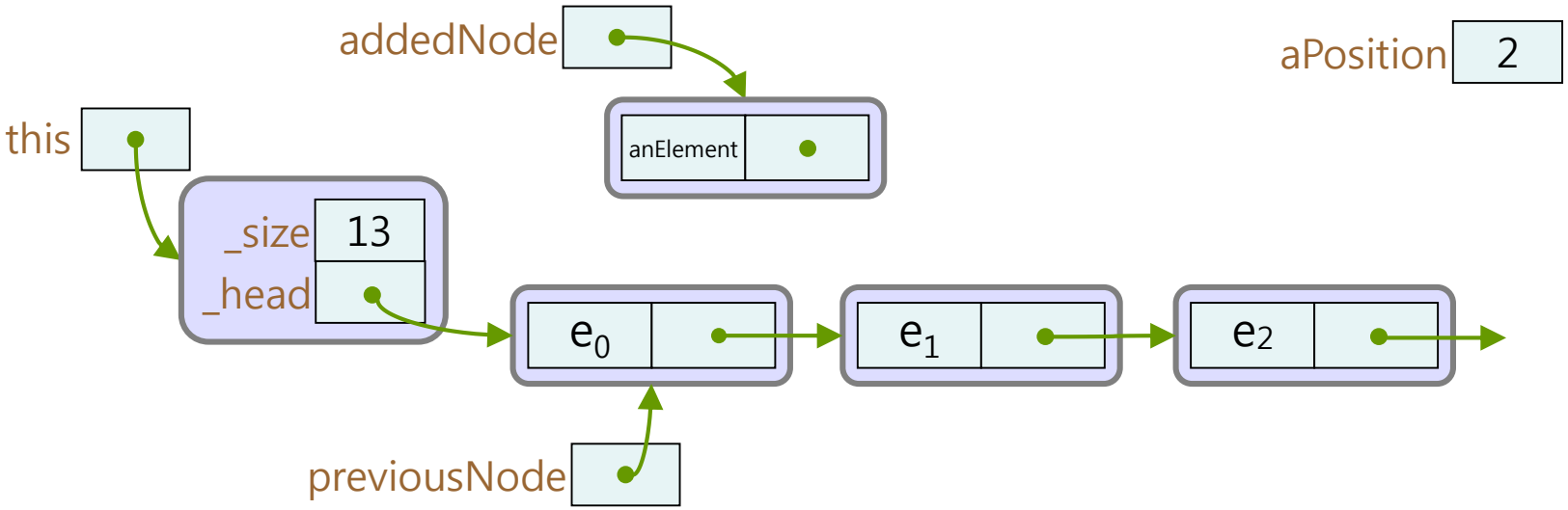
LinkedList: addTo() [4]

```
public boolean    addTo (T anElement, int aPosition)
{
    if ( (aPosition >= 0 && (aPosition <= this._size) ) {
        Node  addedNode = new Node (anElement, null) ;
        if ( aPosition == 0 ) {
            addedNode.setNext (this._head) ;
            this._head = addedNode ;
        }
        else {
            Node  previous = this._head  ;
```



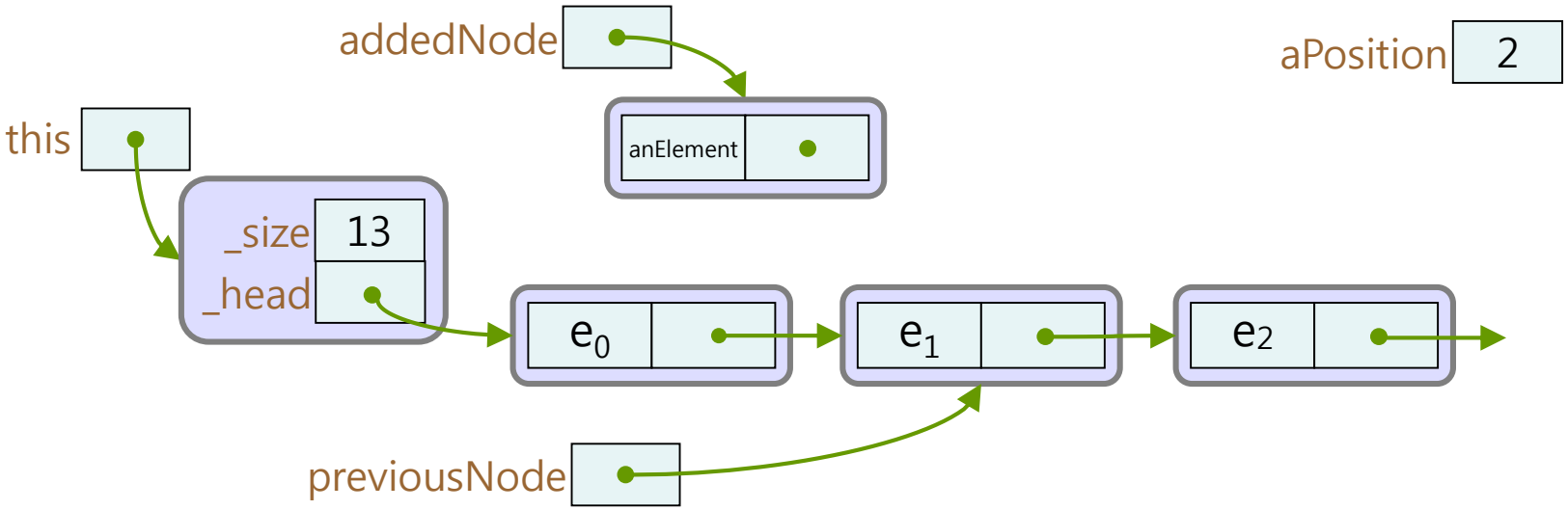
LinkedList: addTo() [5]

```
public boolean    addTo (T anElement, int aPosition)
{
        ......
        else {
            Node  previousNode = this._head ;
            for (int i = 1 ; i < givenPosition ; i++) {
                previousNode = previousNode.next() ;
            }
            addedNode.setNext (previousNode.next()) ;
            previousNode.setNext (addedNode) ;
        }
```



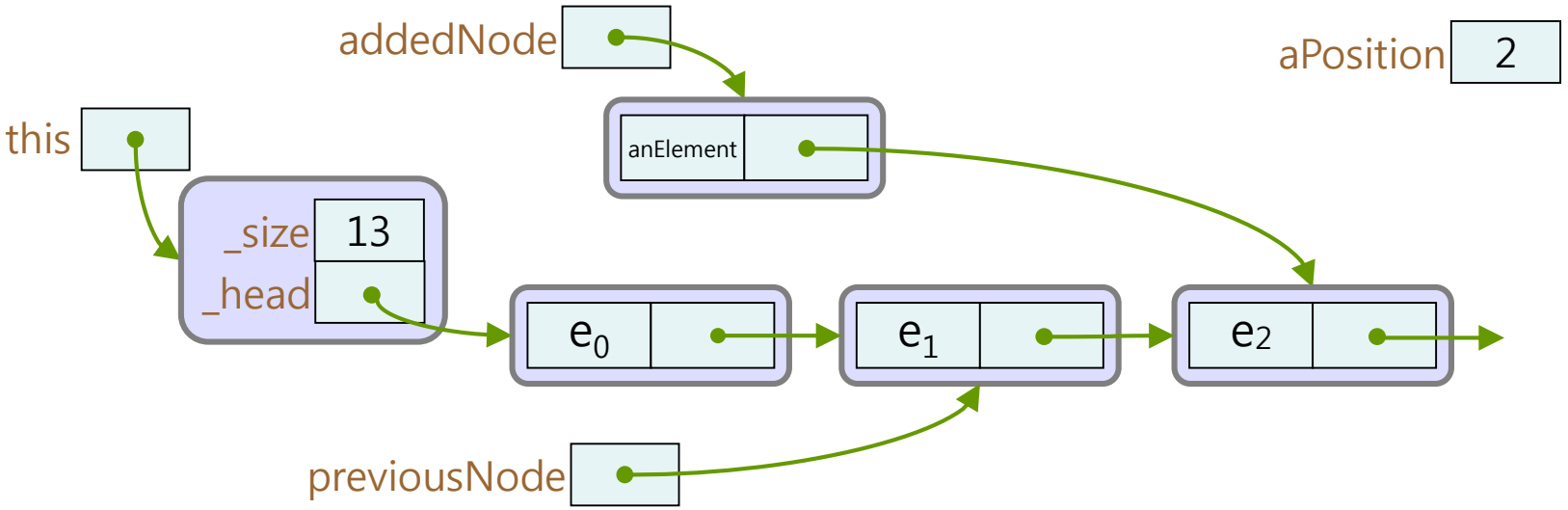
LinkedList: addTo() [6]

```
public boolean    addTo (T anElement, int aPosition)
{
        ......
        else {
            Node   previousNode = this._head ;
            for (int i = 1 ; i < givenPosition ; i++) {
                previousNode = previousNode.next() ;
            }
            addedNode.setNext (previousNode.next()) ;
            previousNode.setNext (addedNode) ;
        }
```



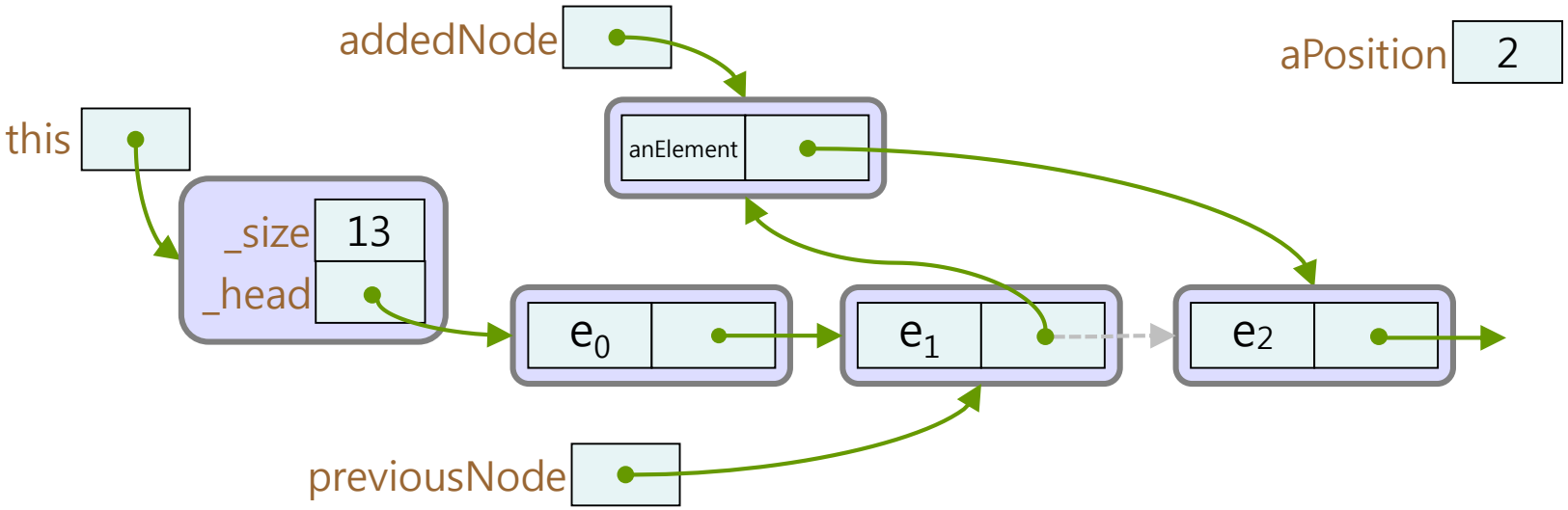
LinkedList: addTo() [7]

```
public boolean    addTo (T anElement, int aPosition)
{
        ......
        else {
            Node  previousNode = this._head ;
            for (int i = 1 ; i < aPosition ; i++) {
                previousNode = previousNode.next() ;
            }
            addedNode.setNext (previousNode.next()) ;
            previousNode.setNext (addedNode) ;
        }
```



LinkedList: addTo() [8]

```
public boolean    addTo (T anElement, int aPosition)
{
        .....
        else {
            Node  previousNode = this._head ;
            for (int i = 1 ; i < aPosition ; i++) {
                previousNode = previousNode.next() ;
            }
            addedNode.setNext (previousNode.next()) ;
            previousNode.setNext (addedNode) ;
        }
```

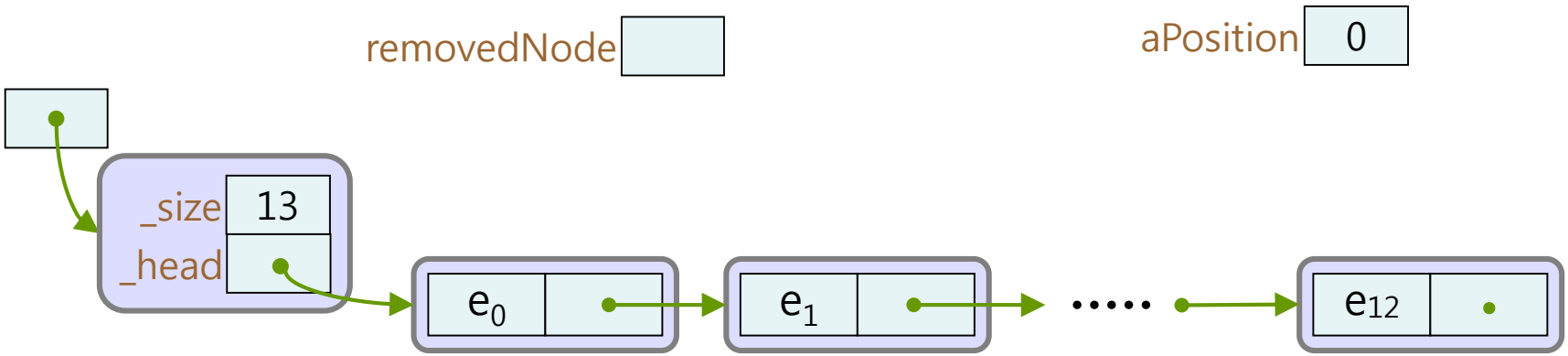


LinkedList: 원소 삭제하기

```
public Element  removeFrom (int aPosition)
{
    if ( ! this.anElementDoesExistAt (aPosition) ) { // 삭제할 원소가 없거나, 잘못된 위치
        return null ;
    }
    else {
        // 리스트는 비어 있지 않으며, 삭제할 원소가 있음
        T  removedNode= null ;
        if ( aPosition == 0) { // 삭제할 원소가 맨 앞 원소
            removedNode = this._head ;
            this._head = this._head.next() ;
        }
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
            Node  previousNode = this._head ;
            for (int i = 1 ; i < aPosition ; i++) {
                previousNode = previousNode.next() ; // 삭제할 위치의 앞 노드를 찾는다
            }
            Node  removedNode = previousNode.next() ;
            previousNode.setNext (removedNode().next()) ;
        }
        this._size-- ;
        return  removedNode.element() ;
    }
}

public Element  removeLast ()
{
    return  (removeFrom(this._size-1));
}
```

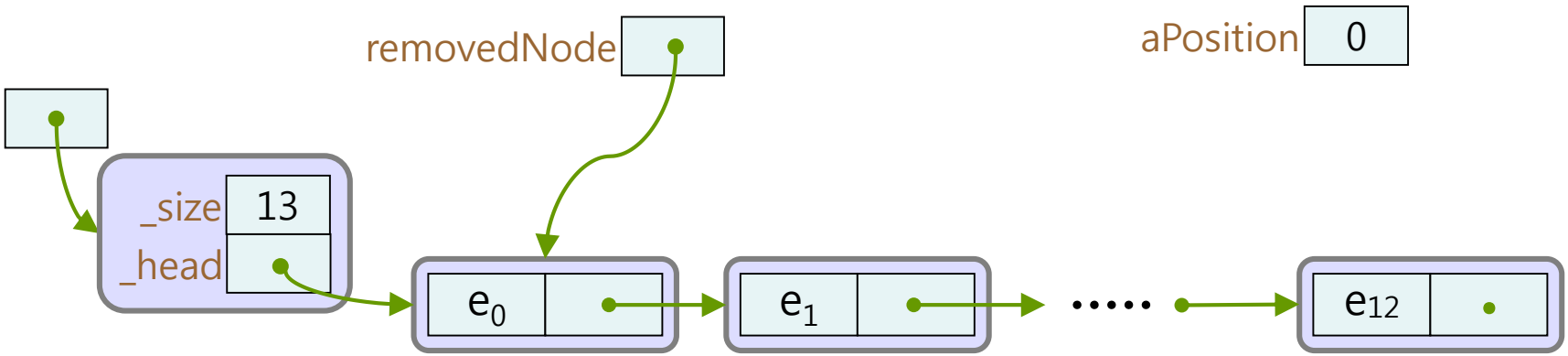
LinkedList: removeFrom() [1]

```
public Element  removeFrom (int aPosition)
{
      if ( ! this.anElementDoesExistAt (aPosition) ) { // 삭제할 원소가 없거나, 잘못된 위치
            return null ;
      }
      else {
            // 리스트는 비어 있지 않으며, 삭제할 원소가 있음
            T  removedNode= null ;
            if ( aPosition == 0 ) { // 삭제할 원소가 맨 앞 원소
                  removedNode = this._head ;
                  this._head = this._head.next() ;
            }
            else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
```



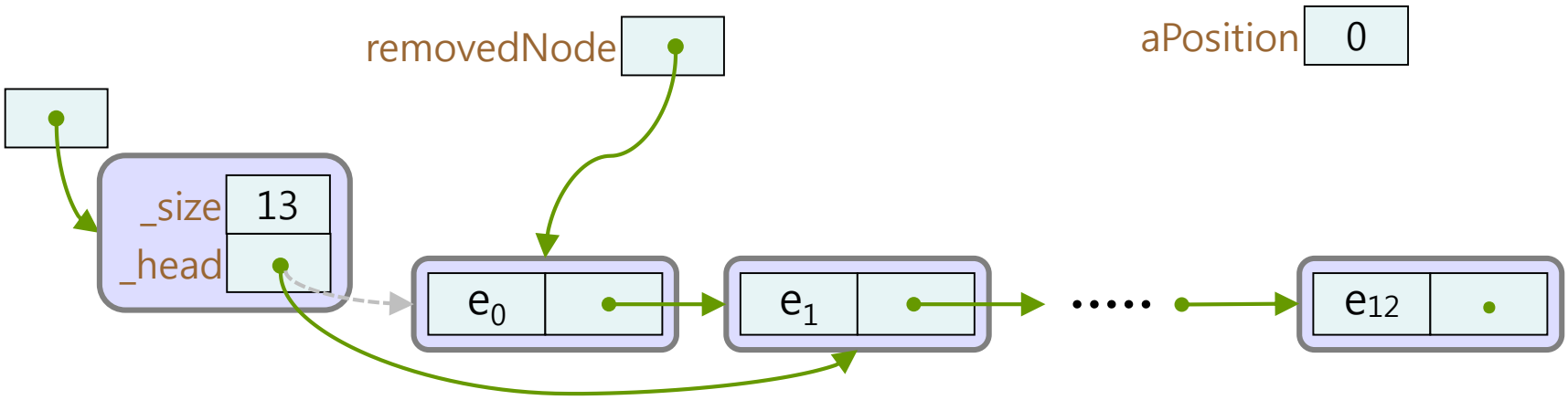
LinkedList: removeFrom() [2]

```
public Element  removeFrom (int aPosition)
{
      if ( ! this.anElementDoesExistAt (aPosition) ) { // 삭제할 원소가 없거나, 잘못된 위치
            return null ;
      }
      else {
            // 리스트는 비어 있지 않으며, 삭제할 원소가 있음
            T  removedNode= null ;
            if ( aPosition == 0 ) { // 삭제할 원소가 맨 앞 원소
                  removedNode = this._head ;
                  this._head = this._head.next() ;
            }
            else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
```



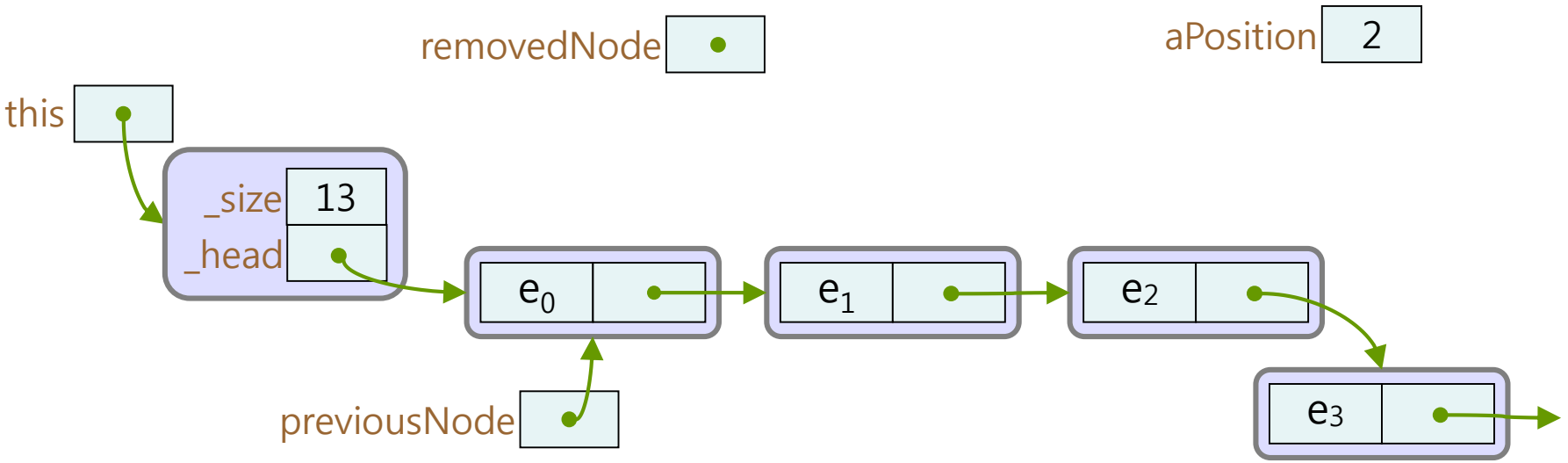
LinkedList: removeFrom() [3]

```
public Element  removeFrom (int aPosition)
{
      if ( ! this.anElementDoesExistAt (aPosition) ) { // 삭제할 원소가 없거나, 잘못된 위치
            return null ;
      }
      else {
            // 리스트는 비어 있지 않으며, 삭제할 원소가 있음
            T  removedNode= null ;
            if ( aPosition == 0 ) { // 삭제할 원소가 맨 앞 원소
                  removedNode = this._head ;
                  this._head = this._head.next() ;
            }
            else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
```



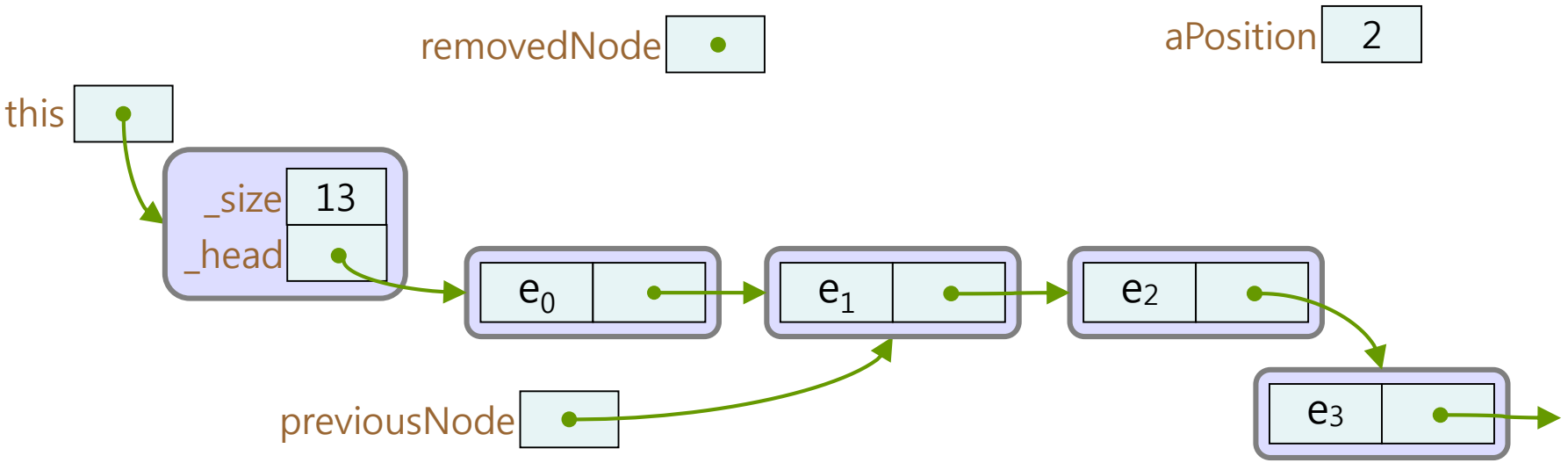
LinkedList: removeFrom() [4]

```
public Element  removeFrom (int aPosition)
{
        ......
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
              Node  previousNode = this._head ;
              for (int i = 1 ; i < aPosition ; i++) {
                    previousNode = previousNode.next(); // 삭제할 위치의 앞 노드를 찾는다
              }
              Node  removedNode = previousNode.next() ;
              previous.setNext (removedNode().next()) ;
        }
        this._size-- ;
        return  removedNode.element() ;
```



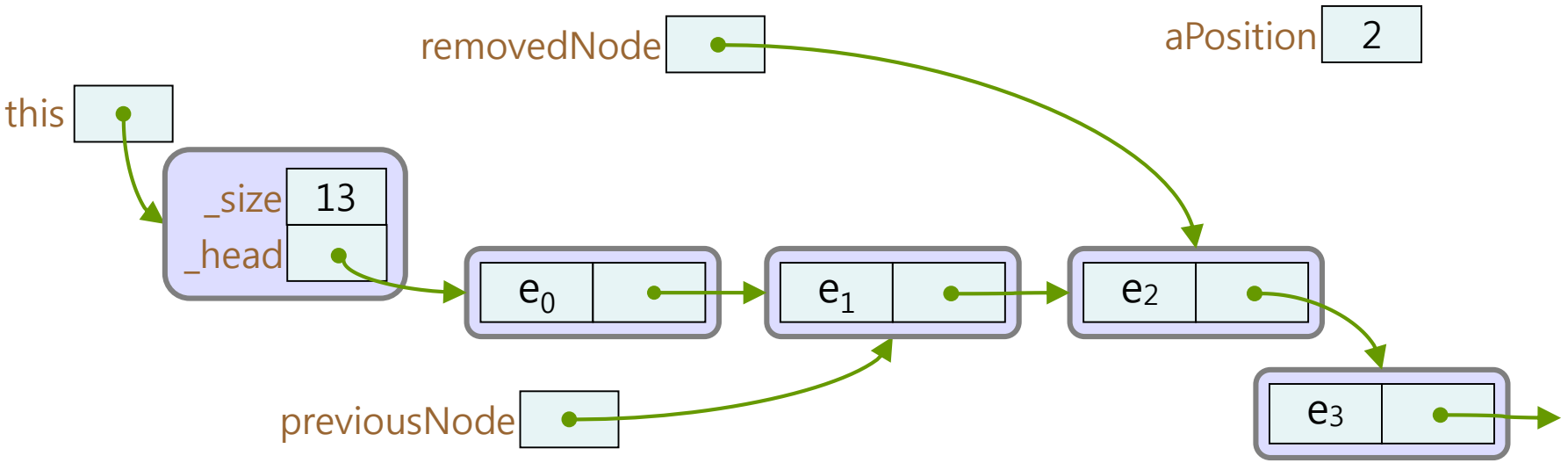
LinkedList: removeFrom() [5]

```
public Element  removeFrom (int aPosition)
{
        ......
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
              Node  previousNode = this._head ;
              for (int i = 1 ; i < aPosition ; i++) {
                    previousNode = previousNode.next(); // 삭제할 위치의 앞 노드를 찾는다
              }
              Node  removedNode = previousNode.next() ;
              previousNode.setNext (removedNode().next()) ;
        }
        this._size-- ;
        return  removedNode.element() ;
```



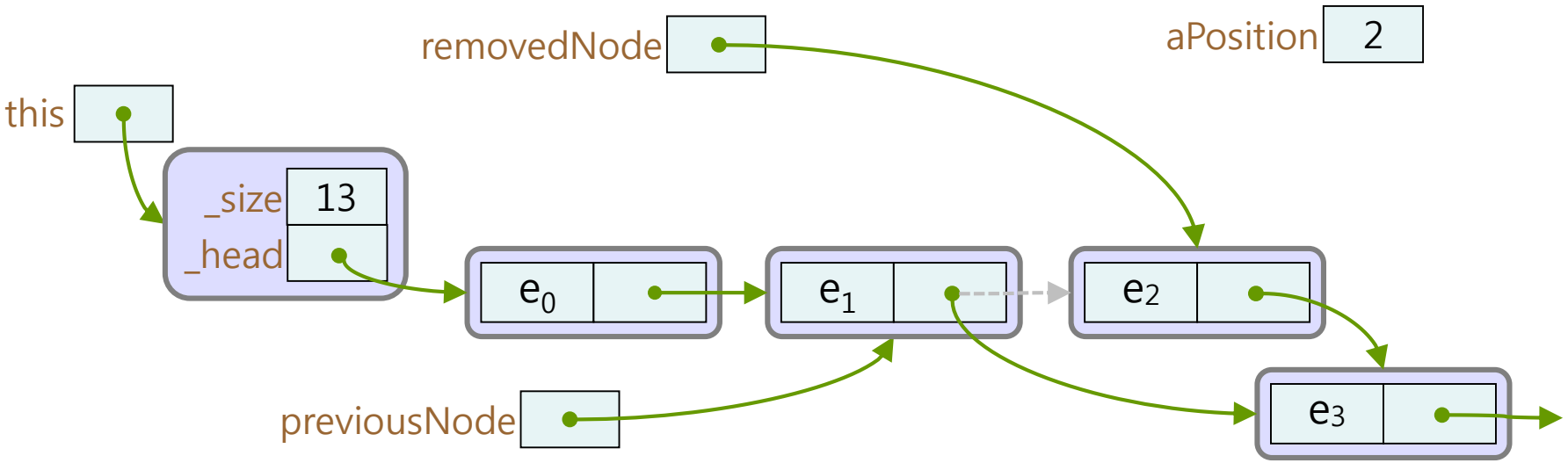
LinkedList: removeFrom() [6]

```
public Element  removeFrom (int aPosition)
{
        ......
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
              Node  previousNode = this._head ;
              for (int i = 1 ; i < aPosition ; i++) {
                    previousNode = previousNode.next(); // 삭제할 위치의 앞 노드를 찾는다
              }
              Node  removedNode = previousNode.next() ;
              previous.setNext (removedNode().next()) ;
        }
        this._size-- ;
        return  removedNode.element() ;
```



LinkedList: removeFrom() [7]

```
public Element  removeFrom (int aPosition)
{
        ......
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
              Node  previousNode = this._head ;
              for (int i = 1 ; i < aPosition ; i++) {
                    previousNode = previousNode.next(); // 삭제할 위치의 앞 노드를 찾는다
              }
              Node  removedNode = previousNode.next() ;
              previousNode.setNext (removedNode().next()) ;
        }
        this._size-- ;
        return  removedNode.element() ;
```



LinkedList: 원소 바꾸기

```
public boolean  replaceAt (T anElement, int aPosition)
{
    if ( ! this.anElementDoesExistAt (aPosition) ) {
        // 대체할 노드가 없거나, 잘못된 위치
        return false ;
    }
    else {
        Node  currentNode = this._head ;
        for (int i = 0 ; i < aPosition ; i++) {
              currentNode = currentNode.next() ;
                 // 원소를 대체할 노드를 찾는다
        }
        currentNode.setElement (anElement) ;
        return  true ;
    }
}
```

LinkedList: clear()

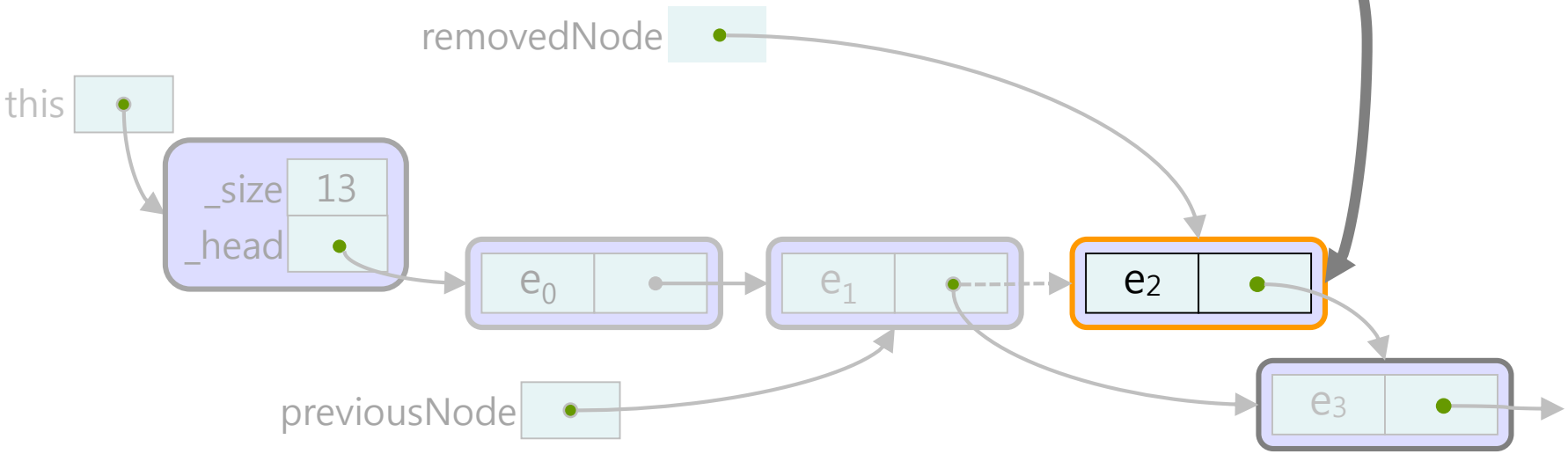
```
// 내용 바꾸기
.....

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

쓰레기 줍기
(Garbage Collection)

쓰레기 줍기

```
public Element  removeFrom (int aPosition)
{
        ......
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
            Node  previousNode = this._head ;
            for (int i = 1 ; i < aPosition ; i++) {
                p                                          앞 노드를 찾는다
            }
            Node
            previo
        }
        this._size--
        return  re
```

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

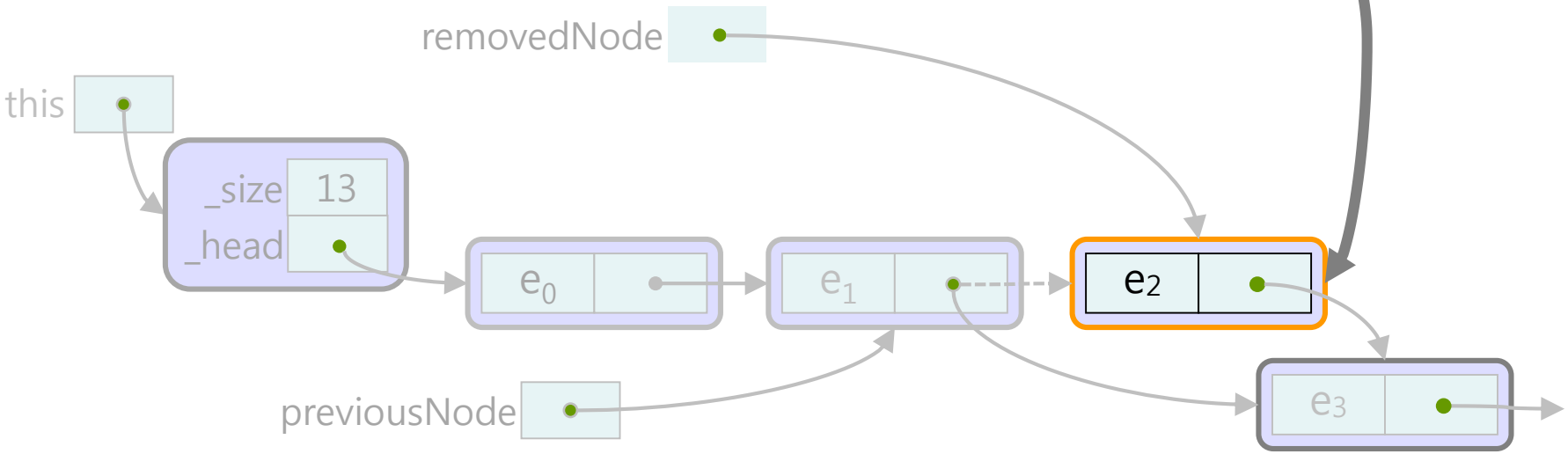


쓰레기 줍기

```
public Element  removeFrom (int aPosition)
{
        ......
        else { // 삭제할 원소의 위치는 맨 앞이 아니며, 따라서 원소가 두 개 이상
            Node  previousNode = this._head ;
            for (int i = 1 ; i < aPosition ; i++) {
                p                                          앞 노드를 찾는다
            }
            Node
            previo
        }
        this._size--
        return  re
```

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

➡ "Garbage Collection"



모델-뷰-컨트롤러

입출력을 리스트 클래스 안에서?

```
public void  showAll()
{
    Node  current = this._head;
    while (current != null)
    {
        System.out.println (current.element()) ; //  Why BAD??
        current = current.next() ;
    }
}
```

- 클래스의 역할의 구분:
  - 모델 (Model): 입출력과 무관한 순수한 알고리즘
  - 뷰 (View): 입출력만 담당
  - 컨트롤러 (Controller): 모델 객체와 뷰 객체를 소유하고 제어

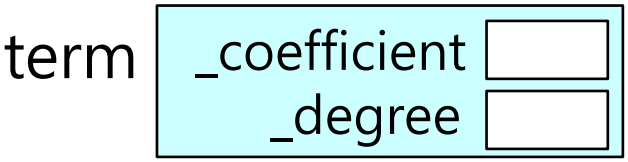
실습: 다항식

실습: 다항식 연산

- 다항식의 덧셈을 구현한다.

$$P(x) + Q(x) = \sum_{i=0}^{n} c_i x^i + \sum_{i=0}^{n} d_i x^i = \sum_{i=0}^{n} (c_i + d_i) x^i$$

  - 항(term)은 계수 (coefficient)와 차수(degree)로 이루어진다.



  - 다항식은 항들(의 합) 으로 이루어진다.

- 입력
  - 다항식을 구성하는 항들을 입력 받는다

필요한 Class

- Term
  - 다항식의 구성 요소
- Node<Term>
  - 연결 체인에 사용할 노드
  - Term을 원소로 한다
- Polynomial: List<Term>
  - Term들의 연결 리스트

“리스트” [끝]