Ch. 3 Data Abstraction

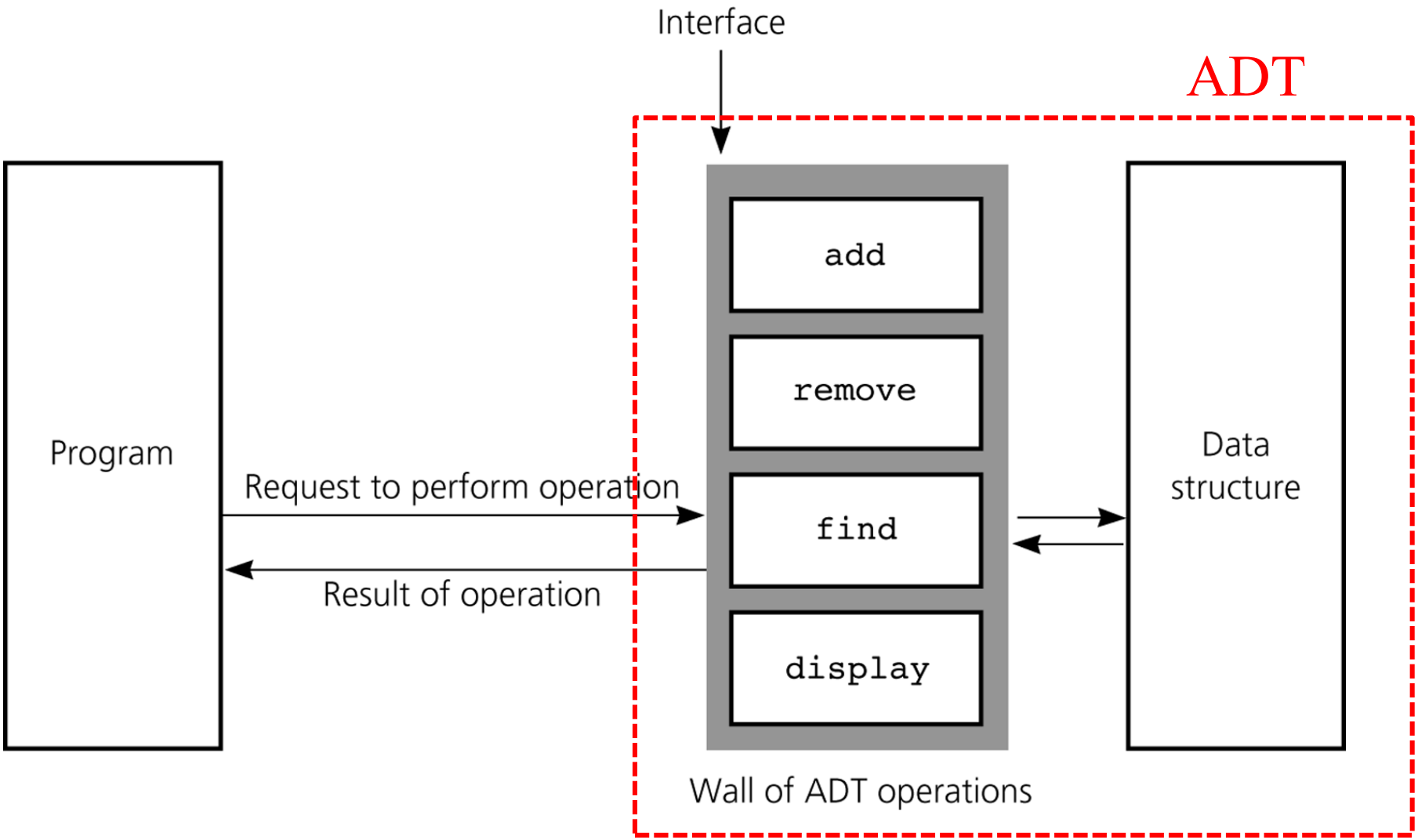
- Modular programming
- Procedural abstraction
  - Know what each method does but not how it does
- Data abstraction
  - Focus what you can do to a collection of data but not how you do it
- ADT (Abstract Data Type)
  - A collection of data

    together with a set of operations on the data

Examples of ADT

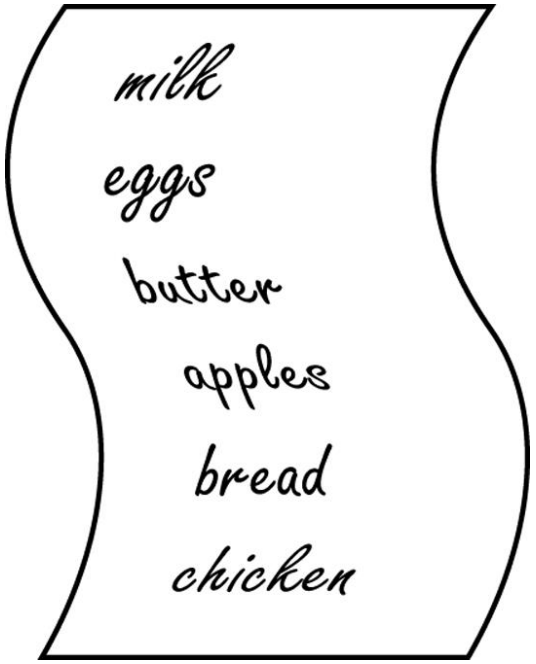
- A sorted array
  together w/ the binary search algorithm
- A set of students
  together w/ student-affairs operations
- Grocery items
  together w/ operations on them

ADT Operations Isolates a Data Structure from the Program



An Example Design of ADT

- ADT Grocery_List
- ADT consists of
  - Items (data)
  - Operations that you can do on the items

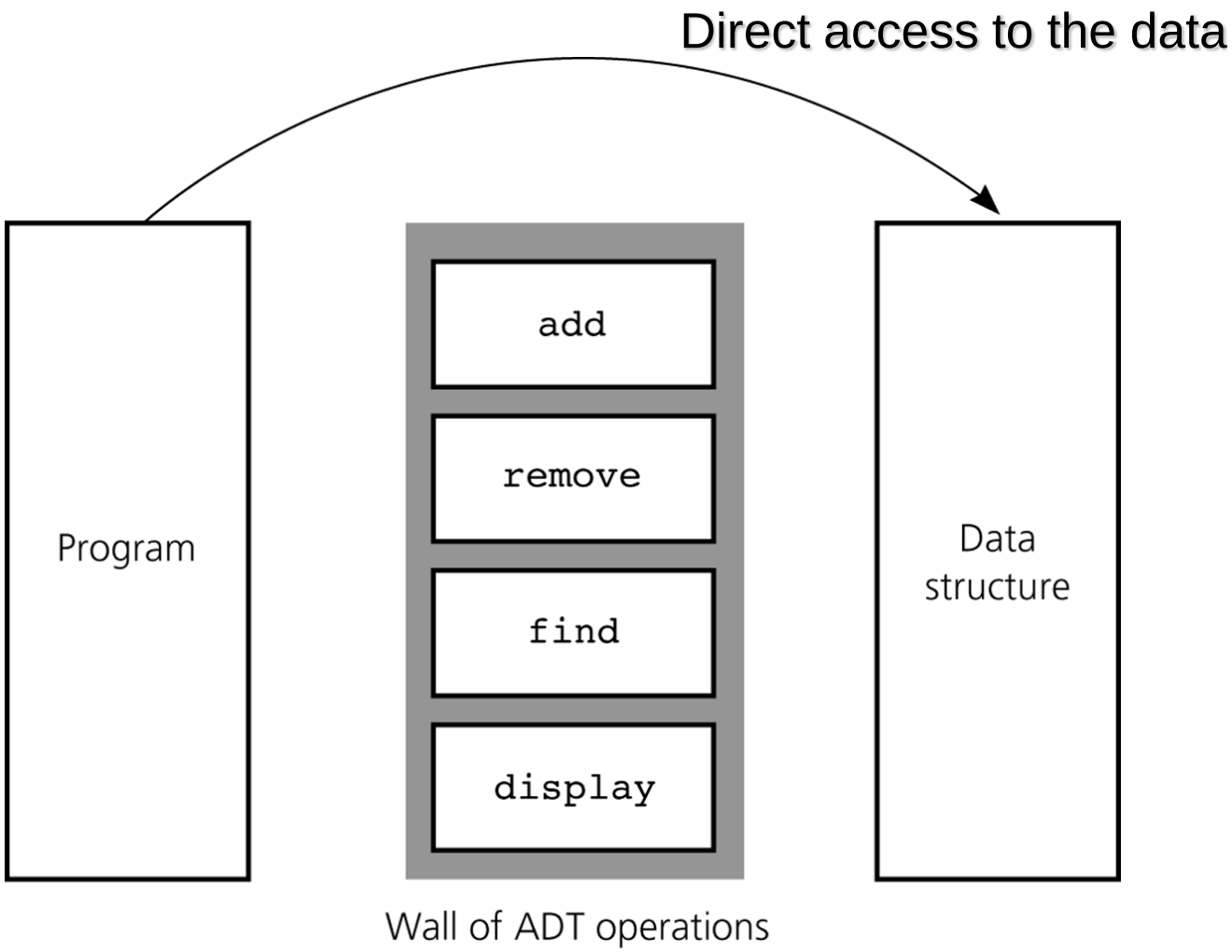


Operations on the Items

- Create an empty list
- Determine whether a list is empty
- Determine the # of items on a list
- Add an item in the list
- Remove an item from the list
- Retrieve an item from the list
- Get the price of an item
- Get the price sum of the items in the list
- Print the items on the list

✓ “How to implement” is another issue

A Bad Design


Direct access to the data
Program
add
remove
find
display
Data structure
Wall of ADT operations

Abstraction Example

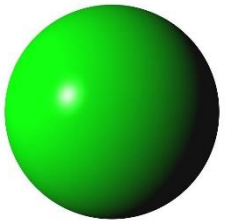
```
firstItem = aList.item[0];  vs.  firstItem = aList.get(1);
```

- Get the 1st item of the list
- Get the item at slot 0 in the array item[ ] for list

What if the array changes to another data structure later?

JAVA Classes

- Constructors
- Inheritance
- Class Object



constructors

```
public class Sphere {
        private double theRadius;
        public Sphere( ) {
                setRadius(1.0);
        } // default constructor
        public Sphere(double initialRadius) {
                setRadius(initialRadius);
        }
        public void setRadius(double newRadius) {
                if (newRadius >= 0.0) {
                        theRadius = newRadius;
                }
        }
        public double radius( ) {
                return theRadius;
        }
        public double volume( ) {
                return (4.0 * Math.PI * Math.pow(theRadius, 3.0)) / 3.0;
        }
        public void displayStatistics( ) {
                System.out.println("\nRadius = " + radius( ) + "\nDiameter = "
                    + diameter( ) + "\nCircumference = " + circumference( )
                    + "\nArea" + area( ) + "\nVolume" + volume( ));
        }
} // end Sphere
```

Inheritance

```
public class Ball extends Sphere {
        private String theName;
        public Ball( ) { // at first create a sphere w/ radius 1.0
                setName("unknown");
        } // default constructor
        public Ball(double initialRadius, String initialName) {
                super(initialRadius);
                setName(initialName);
        }
        public String name( ) {
                return theName;
        }
        public void setName(String newName) {
                theName = newName;
        }
        public void displayStatistics( ) {
                System.out.print("\nStatistics for a "+ name( ));
                super.displayStatistics( );
        }
} // end Ball
```

new methods

overwriting (overriding)

Class Object

- Every class is a subclass of the class *Object*

```
public class Sphere {
        …
}
```

implicitly means

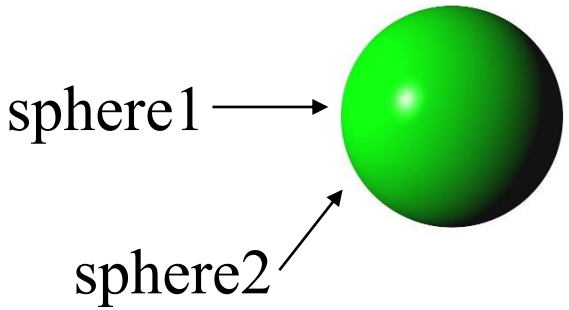
```
public class Sphere extends Object {
        …
}
```

Object Equality (Overriding을 설명하기 위한 예)

- Class Object provides the method equals( )
- Method equals( ) basically checks to see if two references are the same

```
Sphere sphere1 = new Sphere(2.0);
Sphere sphere2 = sphere1;
If (sphere1.equals(sphere2)) {
        System.out.println("The same!");
} else {
        System.out.println("Different!");

}
```

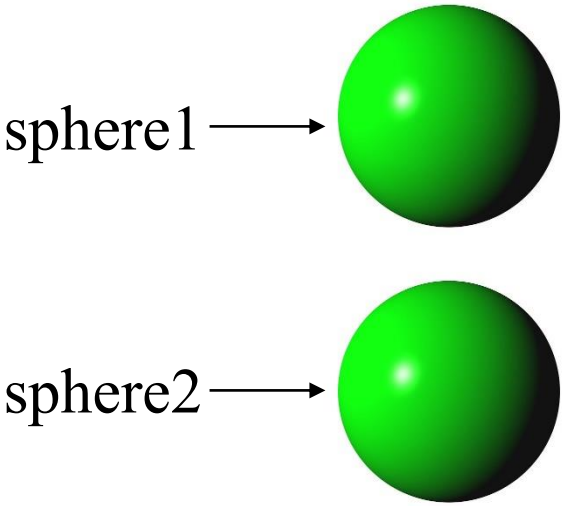


The same!

Object Equality

```
Sphere sphere1 = new Sphere(2.0);
Sphere sphere2 = new Sphere(2.0);
If (sphere1.equals(sphere2)) {
        System.out.println("The same!");
} else {
        System.out.println("Different!");

}
```

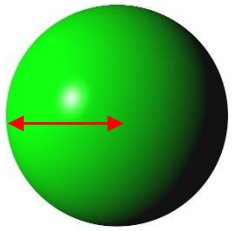


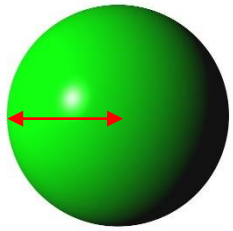
Different!

Overriding

equals( )가 다시 정의된다

```
public class Sphere {
        …
        public boolean equals(Object sp) {
                return ((sp instanceof Sphere)
                        && theRadius == ((Sphere)sp).radius( ));
        }
        …
}
```





cf: Overloading

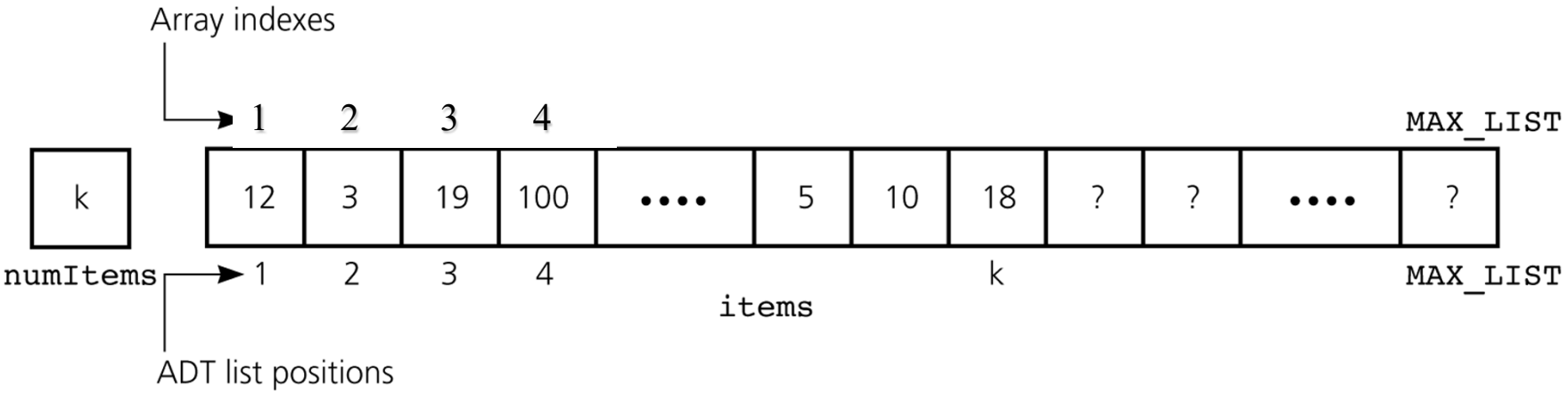
```
public class Sphere {
        …
        public boolean equals(Sphere sp) {
                return (theRadius == sp.radius( ));
        }
        …
}
```

type이 다름

equals( )란 이름으로 또하나 정의된다

- ✓ Overriding: the same name, the same number & types of parameters
  - ➢ Has only one method accessible w/ the name
- ✓ Overloading: the same name, different number or types of parameters
  - ➢ Has two different methods accessible w/ the name

ADT List Implementation w/ Array



- Operations
  - createList( )
  - isEmpty( )
  - size( )
  - add(newPosition, newItem)
  - remove(index)
  - removeAll( )
  - get(index)

Interface Design

```
public interface ListInterface {
        public boolean isEmpty( );
        public int size( );
        public void add(int index, Object item)
                throws ListIndexOutOfBoundsException,
                        ListException;
        public Object get(int index)
                throws ListIndexOutOfBoundsException;
        public void remove(int index)
                throws ListIndexOutOfBoundsException;
        public void removeAll( );
}
```

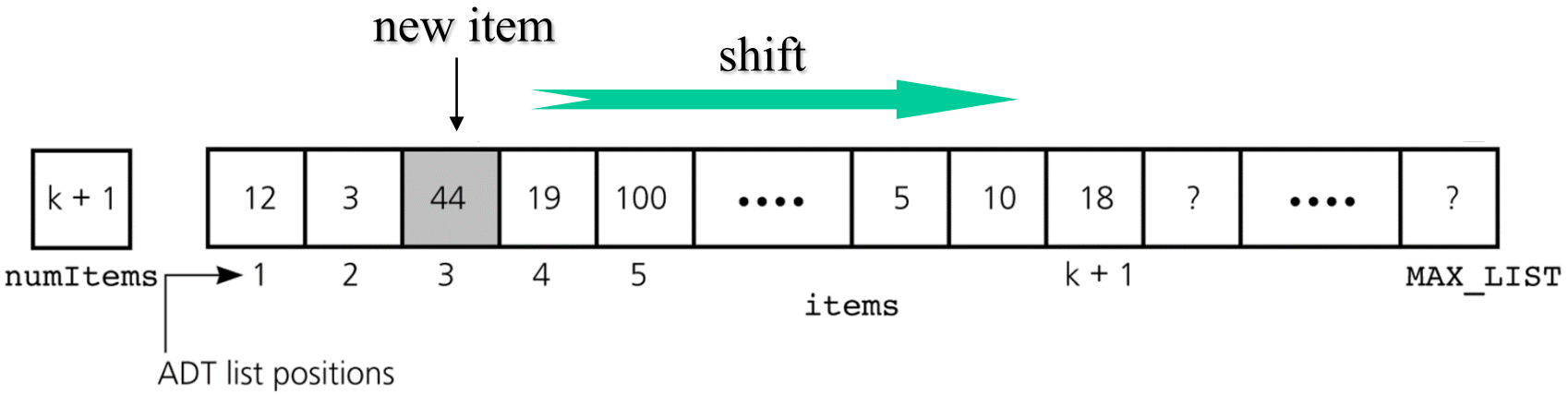
Implementation

```
public class ListArrayBased implements ListInterface {
    private final int MAX_LIST = 50;
    private Object items[ ];   // array of list items[1…MAX_LIST]
    private int numItems;      // # of items in the list

    public ListArrayBased( ) {
        items = new Object[MAX_LIST+1];
        numItems = 0;
    }

    public boolean isEmpty( ) {
            return (numItems == 0);
    }
    public int size( ) {
            return numItems;
    }
```

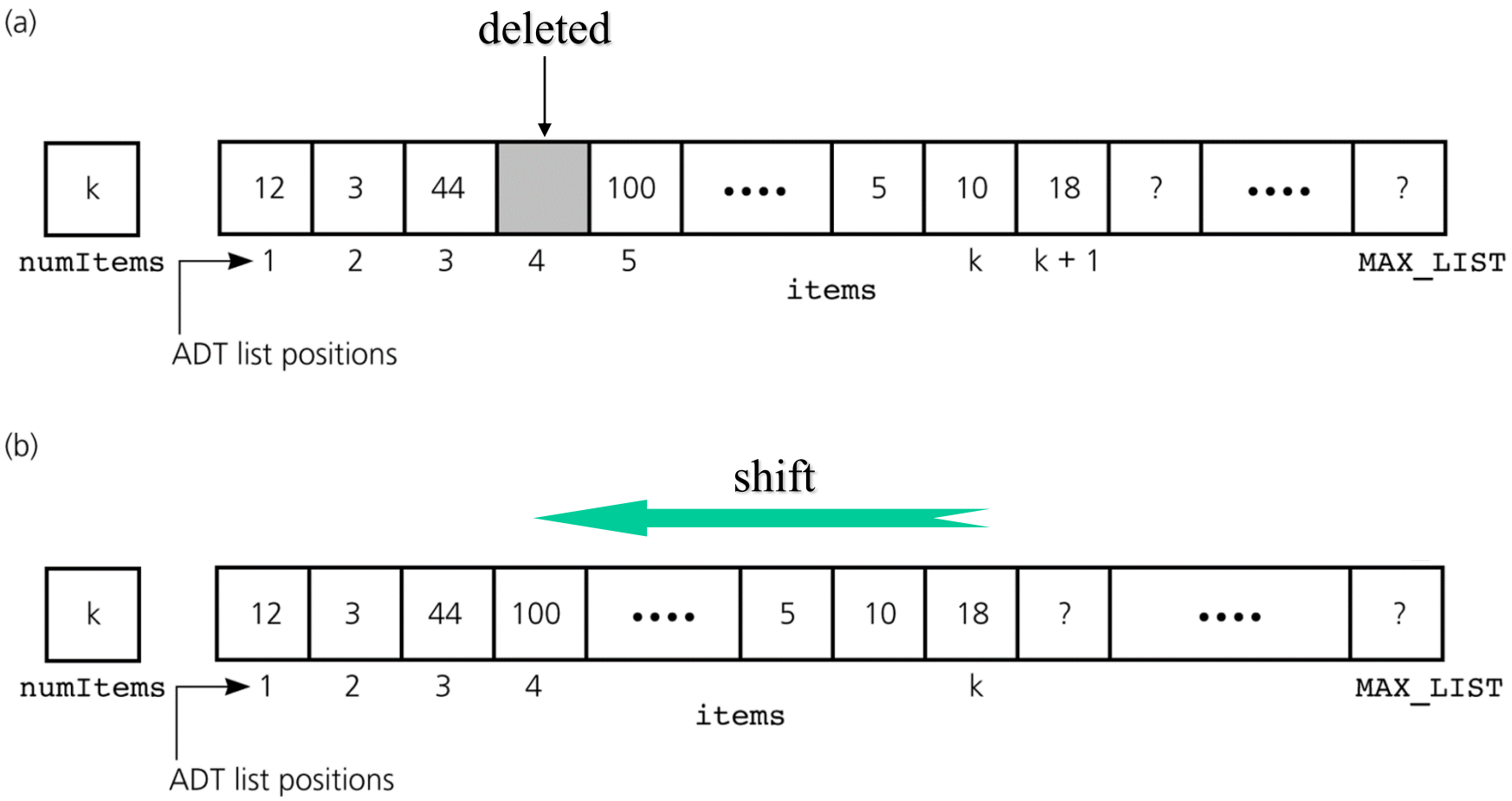
예: Shifting Items Before Insertion at Position 3



Continued…

```
public void removeAll( ) {
        items = new Object[MAX_LIST+1];
        numItems = 0;
}
public void add(int index, Object item)
                throws ListIndexOutOfBoundsException {
        if (numItems > MAX_LIST) {
                Exception 처리;
        }
        if (index >= 1 && index <= numItems+1) { // shift right
                for (int pos = numItems; pos >= index; pos--) {
                        items[pos+1] = items[pos];
                }
                items[index] = item;
                numItems++;
        } else {
                Exception 처리;
        }
}
```

예: Shifting Items After Deletion at Position 4



```
    public Object get(int index)
                    throws ListIndexOutOfBoundsException {
        if (index >= 1 && index <= numItems) {
                    return items[index];
        } else { // index out of range
                    Exception handling;
        }
    }
    public void remove(int index)
                     throws ListIndexOutOfBoundsException {
        if (index >= 1 && index <= numItems) { // shift left
                    for (int pos = index+1; pos <= size( ); pos++) {
                                items[pos-1] = items[pos];
                    }
                    numItems--;
        } else { // index out of range
                    Exception handling;
        }
    }
} // end class ListArrayBased
```