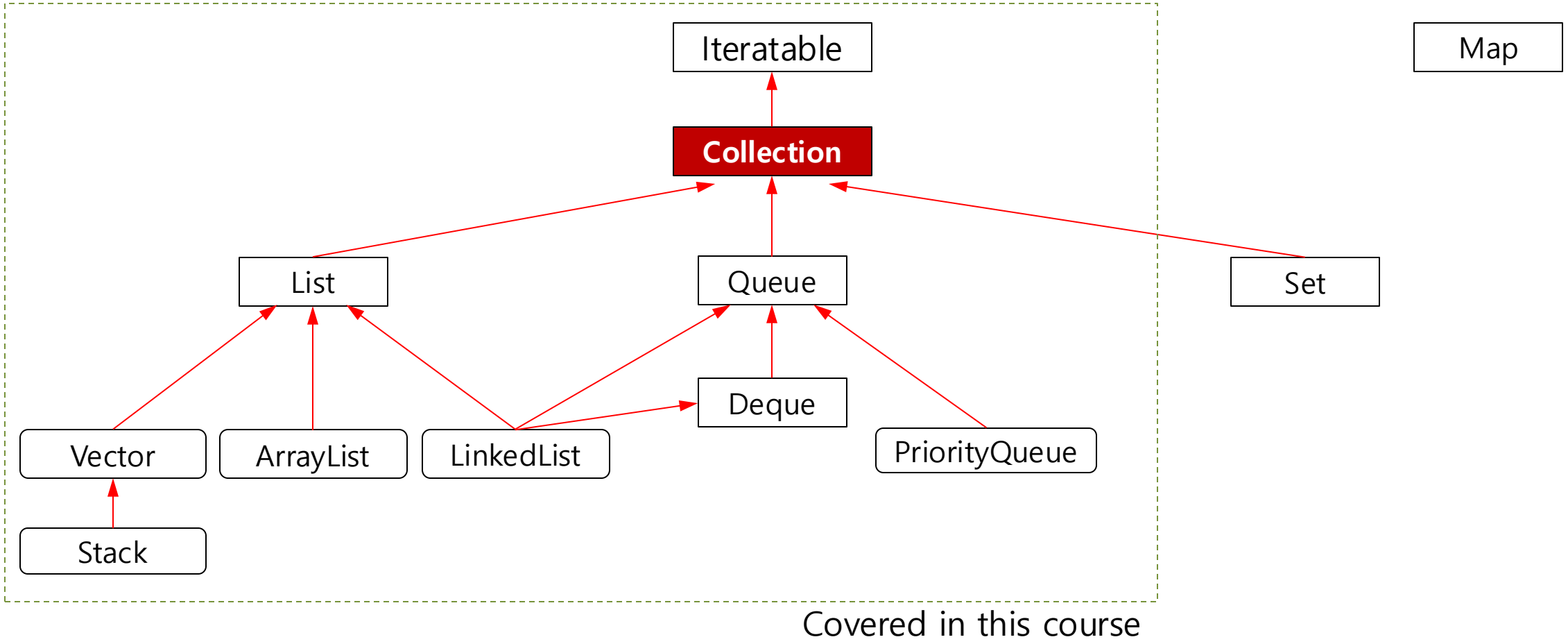
10 Collection Framework

Collection Framework

- Standardized design of classes that store data groups
- Consists of classes that provide an easy way to process multiple data



Core Interfaces of Collection Framework

Interface	Features
List	Ordered set of data. Allow duplicate data. ex) Waiting List (Data with the same name can be duplicated)
	Class: ArrayList, LinkedList, Stack, Vector, ...
Set	A set of data that is out of order. Do not allow duplicate data ex) Set of four-legged animals = {dog, cat, bear, lion, ...}
	Class: HashSet, TreeSet, ...
Map	Set of data consisting of the tuple (key, value) Order not maintained, no duplicated key, value allows duplicates ex) Postal code (area, number), Local phone number (area, number)... (seoul, 02)
	Class: HashMap, TreeMap, HashTable, Properties, ...

Methods in Interface Collection<E> (1/2)

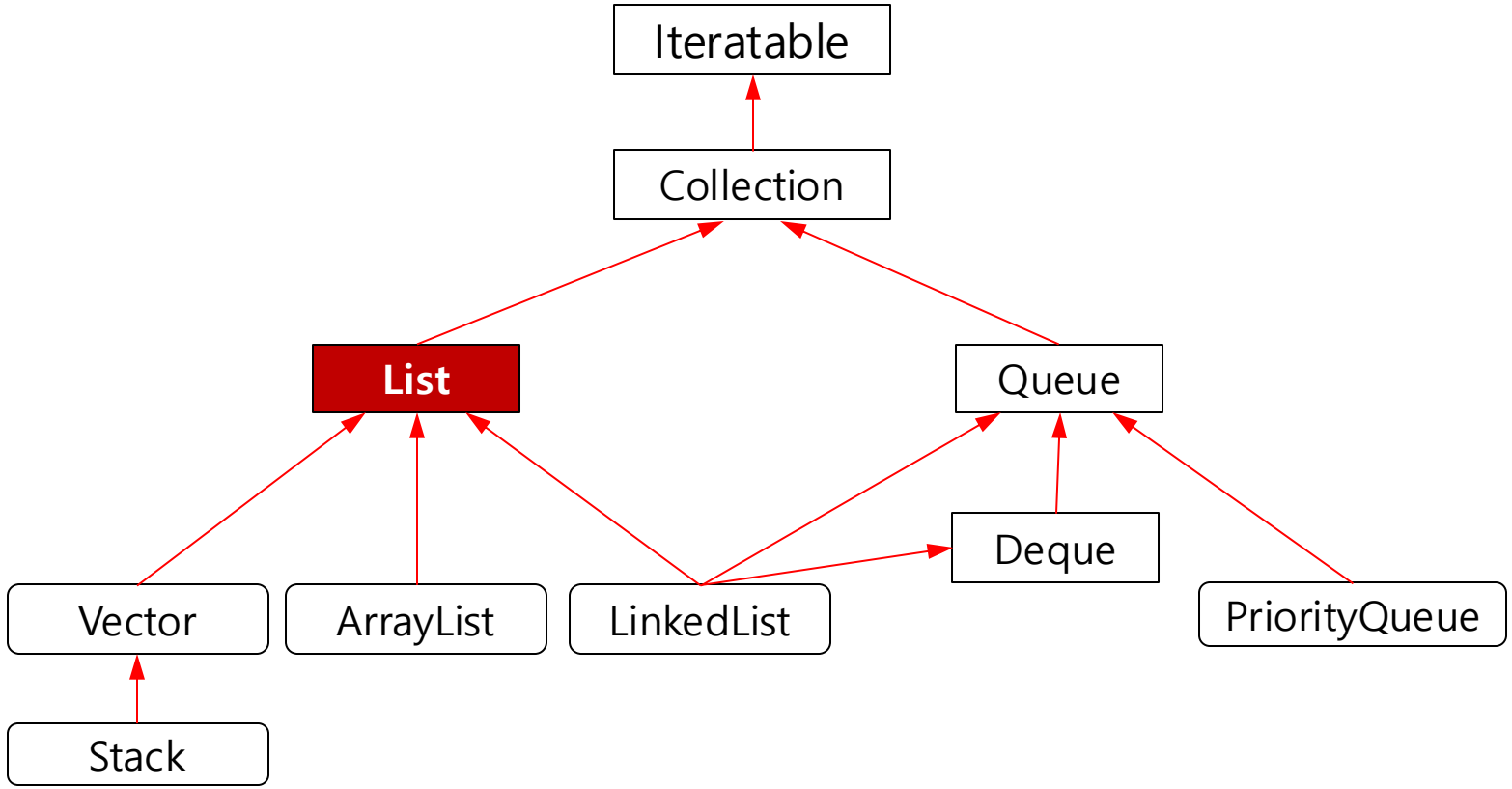
Type	Method	Description
boolean	**add**(**E** e)	Ensures that this collection contains the specified element
boolean	**addAll**(**Collection**<? extends **E**> c)	Adds all of the elements in the specified collection to this collection
void	**clear**()	Removes all of the elements from this collection
boolean	**contains**(**Object** o)	Returns true if this collection contains the specified element.
boolean	**containsAll**(**Collection**<?> c)	Returns true if this collection contains all of the elements in the specified collection.
boolean	**equals**(**Object** o)	Compares the specified object with this collection for equality.
int	**hashCode**()	Returns the hash code value for this collection.
boolean	**isEmpty**()	Returns true if this collection contains no elements.
Iterator<**E**>	**iterator**()	Returns an iterator over the elements in this collection.
boolean	**remove**(**Object** o)	Removes a single instance of the specified element from this collection, if it is present (optional operation), return true when changed
boolean	**removeAll**(**Collection**<> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation), return true when changed

Methods in Interface Collection<E> (2/2)

Type	Method	Description
boolean	**retainAll**(**Collection**<?> c)	Retains only the elements in this collection that are contained in the specified collection. Return true if any change by this call
int	**size**()	Returns the number of elements in this collection.
Object[]	**toArray**()	Returns an array containing all of the elements in this collection.

Interface List<E>

- Order (O): preserving order in which data came into the list
- Duplication (O): allow duplication of the same data in the list



Methods in Interface List<E> (1/3)

Type	Method	Description
void	**add**(int index, **E** element)	Inserts the specified element at the specified position in this list
boolean	**add**(**E** e)	Appends the specified element to the end of this list
boolean	**addAll**(int index, **Collection**<? extends **E**> c)	Inserts all of the elements in the specified collection into this list at the specified position
boolean	**addAll**(**Collection**<? extends **E**> c)	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator
void	**clear**()	Removes all of the elements from this list
boolean	**contains**(**Object** o)	Returns true if this list contains the specified element.
boolean	**containsAll**(**Collection**<?> c)	Returns true if this list contains all of the elements of the specified collection.
static <E> **List**<E>	**copyOf**(**Collection**<? extends E> coll)	Returns an unmodifiable List containing the elements of the given Collection, in its iteration order.

Methods in Interface List<E> (2/3)

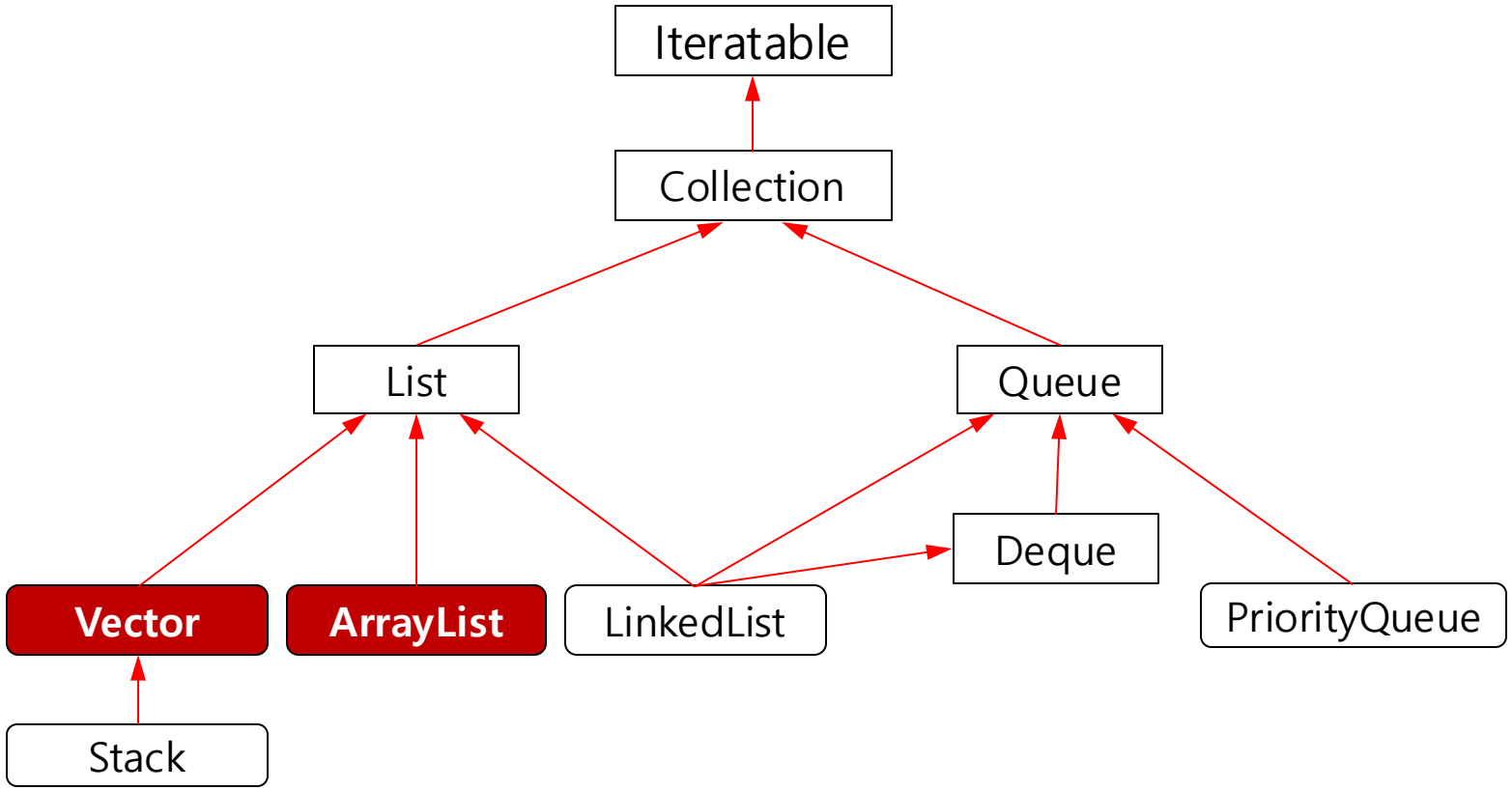
Type	Method	Description
boolean	**equals**(**Object** o)	Compares the specified object with this list for equality.
E	**get**(int index)	Returns the element at the specified position in this list.
int	**hashCode**()	Returns the hash code value for this list.
int	**indexOf**(**Object** o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**()	Returns true if this list contains no elements.
Iterator<E>	**iterator**()	Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(**Object** o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.

Methods in Interface List<E> (3/3)

Type	Method	Description
E	remove(int index)	Removes the element at the specified position in this list
boolean	remove(Object o)	Removes the first occurrence of the specified element from this list, if it is present
boolean	removeAll(Collection<?> c)	Removes from this list all of its elements that are contained in the specified collection
boolean	retainAll(Collection<?> c)	Retains only the elements in this list that are contained in the specified collection
E	set(int index, E element)	Replaces the element at the specified position in this list with the specified element
int	size()	Returns the number of elements in this list.
default void	sort(Comparator<? super E> c)	Sorts this list according to the order induced by the specified Comparator.
List<E>	subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	toArray()	Returns an array containing all of the elements in this list in proper sequence (from first to last element).

ArrayList and Vector

- Order (0), Duplication (0)
- Use array as storage space
- Details of ArrayList in Lecture Note 14-1, too



Using ArrayList (1/2)

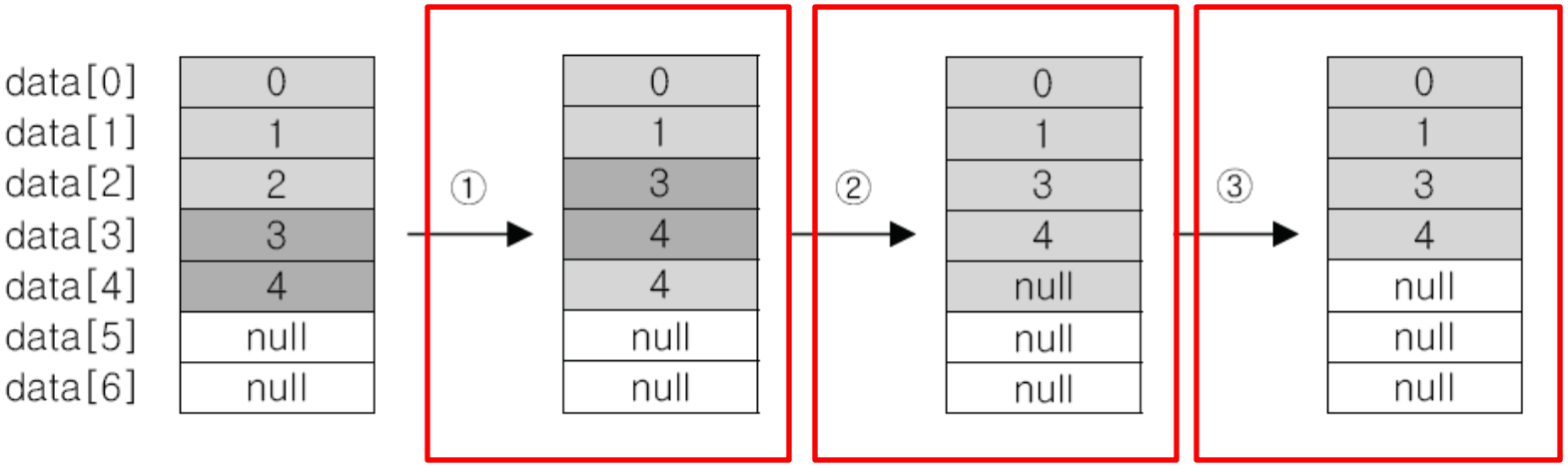
```
import java.util.ArrayList;
import java.util.Collections;    // utility class for Collection (ex. sort())

public class ArrayListEx {

   public static void main(String[] args) {

      ArrayList<Integer> list1 = new ArrayList<Integer>(10);

      list1.add(5);
      list1.add(4);
      list1.add(2);
      list1.add(0);
      list1.add(1);
      list1.add(3);

      ArrayList<Integer> list2 = new ArrayList<Integer>(list1.subList(1, 4));

      System.out.println("list1:" + list1);  // list1:[5, 4, 2, 0, 1, 3]
      System.out.println("list2:" + list2);  // list2:[4, 2, 0]
```

Using ArrayList (2/2)

```
        Collections.sort(list1);
        Collections.sort(list2);
        System.out.println("list1:" + list1);  // list1:[0,1,2,3,4,5]
        System.out.println("list2:" + list2);  // list2:[0,2,4]

        System.out.println("list1.containsall(list2):" +
                        list1.containsAll(list2)); // list1.containsall(list2):true

    }
}
```

Removing Sequence in ArrayList

- Sequence of processing: v.remove(2);



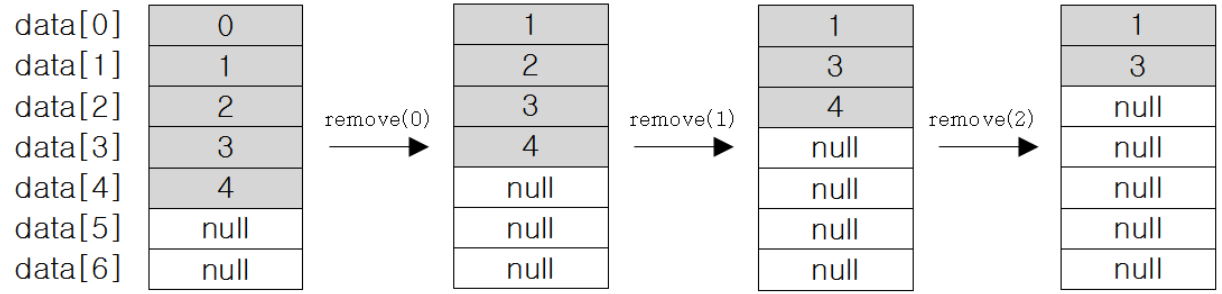
1) The data under the element to be deleted are copied one space upward one by one, overwriting the element to be deleted
2) The last data moved is changed to be null
3) Decrease the size by one

- Note that in the worst case (the case when the first element is removed)
  - (size – 1) data should be moved … O(n) time for n data

Forward or Backward

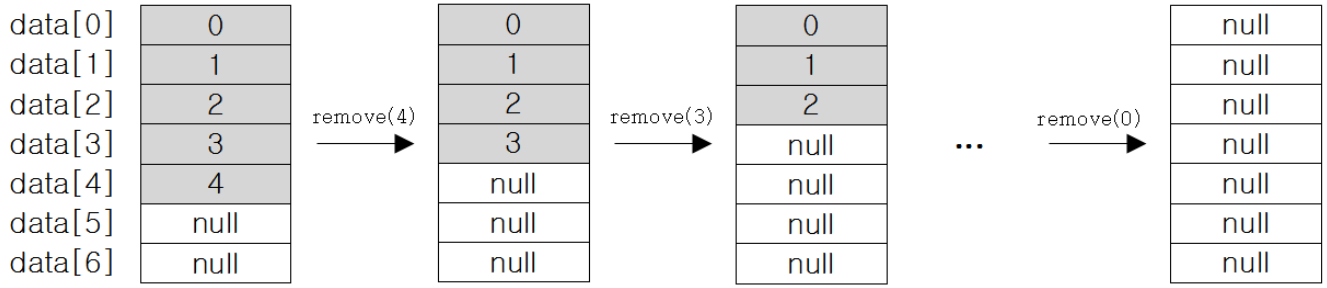
- Removing from the first object to the last (array copy occurs)

```
for(int i=0;i<list.size();i++)
    list.remove(i);
```



- Removing from the last object to the first (no array copy)

```
for(int i=list.size()-1;i>=0;i--)
    list.remove(i);
```

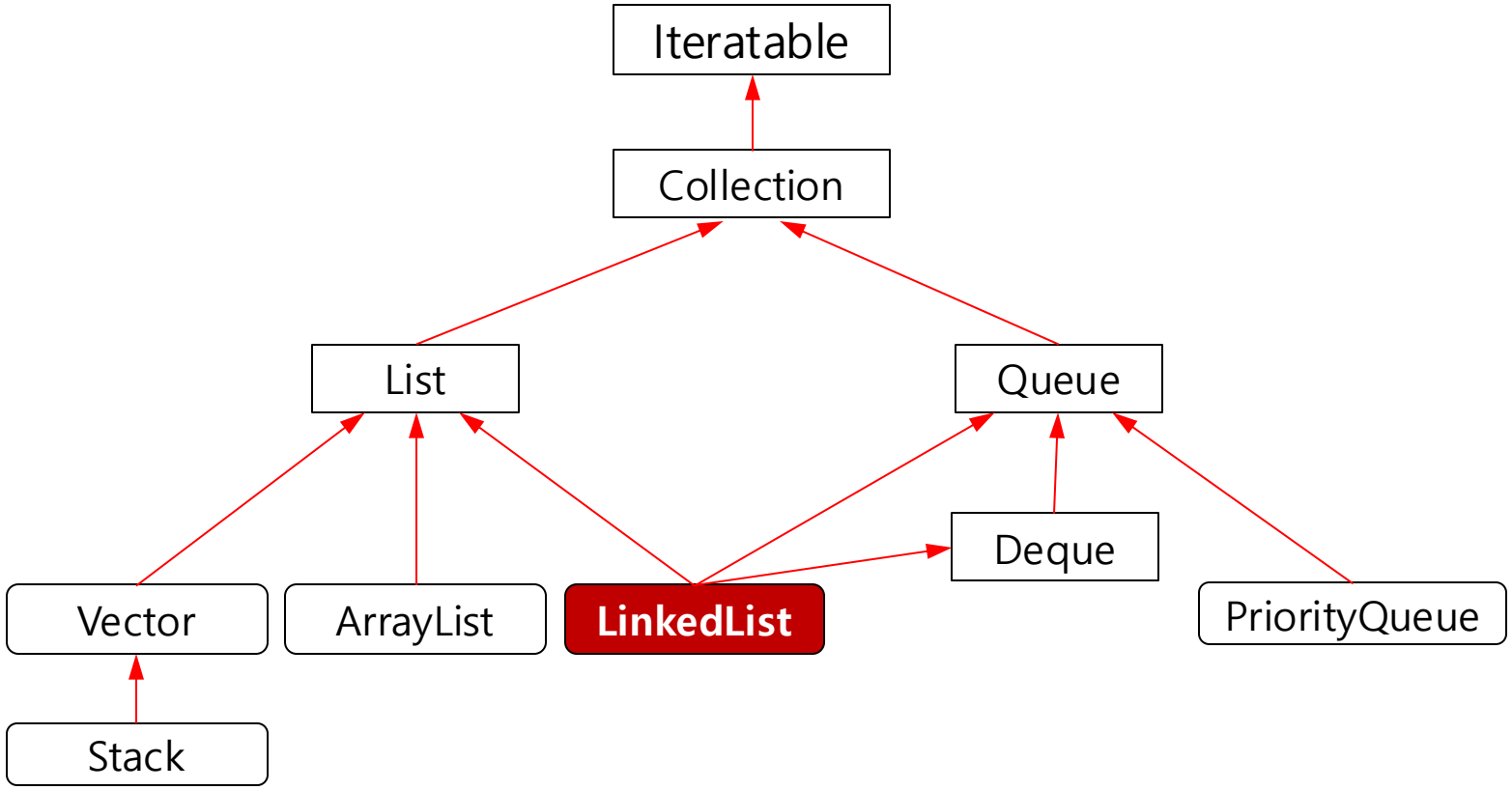


Pros and Cons of ArrayList

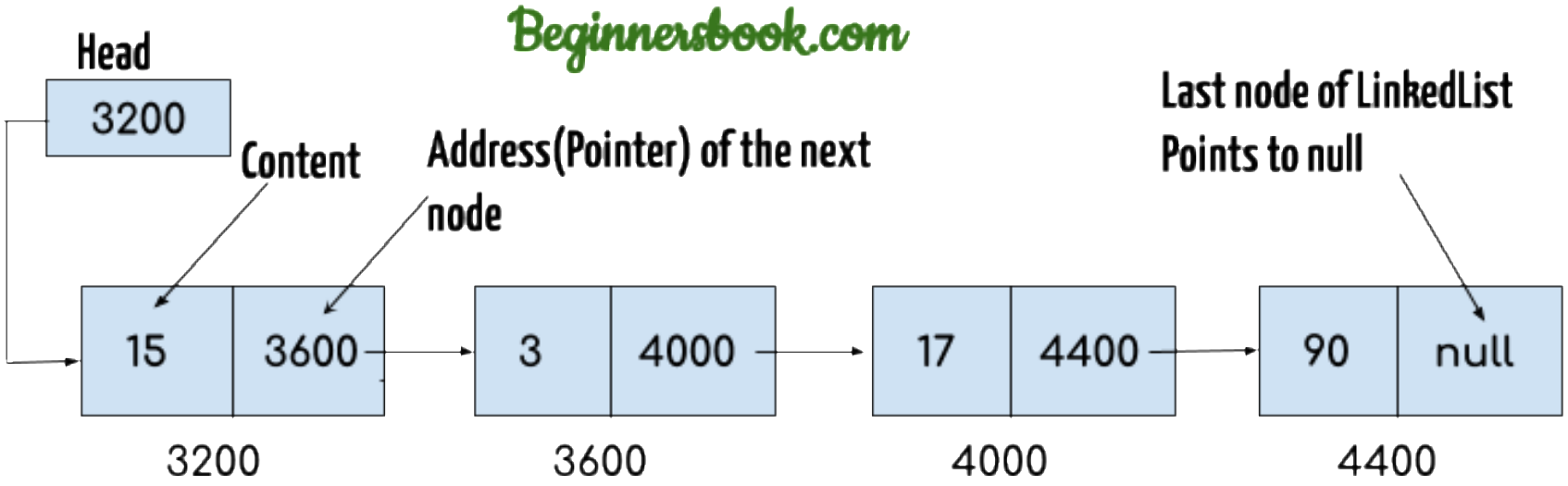
- Advantages
  - Simple structure
  - Fast (direct) access time
    - The memory address can be calculated simply from the index
    - ex) list.get(index) ... address of list[index] = address of list[0] + sizeof(data) * index;
- Disadvantages
  - To change the capacity, we should create a new memory space and copy the data from old space to the new one.
    - If we create an array large enough to avoid resizing, the memory is wasted.
  - It takes a lot of time to add and delete out of sequence data.
    - To add or delete data, a lot of data must be moved.
    - However, sequential data addition (adding to the last) and data deletion sequentially (deleting from the last) are fast.

LinkedList

- Order (0), Duplication (0)

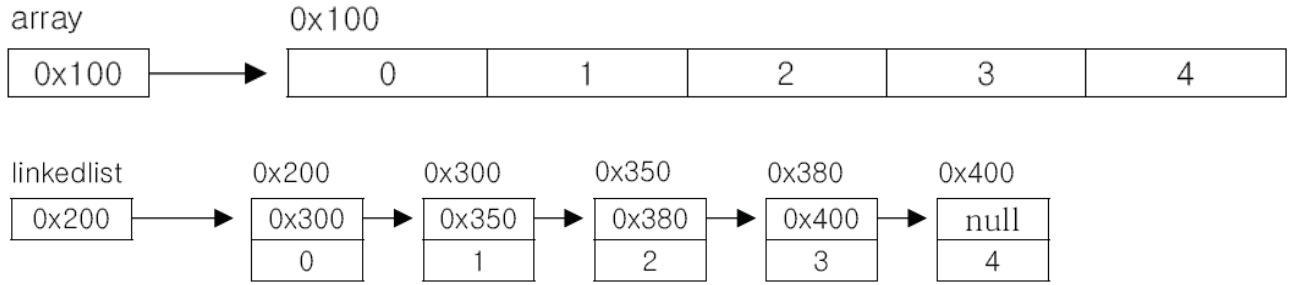


Linked List

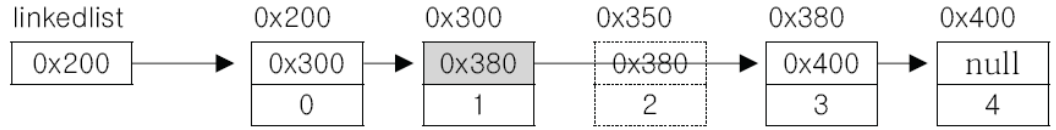

Beginnersbook.com
Head
3200
Content
Address(Pointer) of the next node
Last node of LinkedList
Points to null
15
3600
3
4000
17
4400
90
null
3200
3600
4000
4400

Features of Linked List

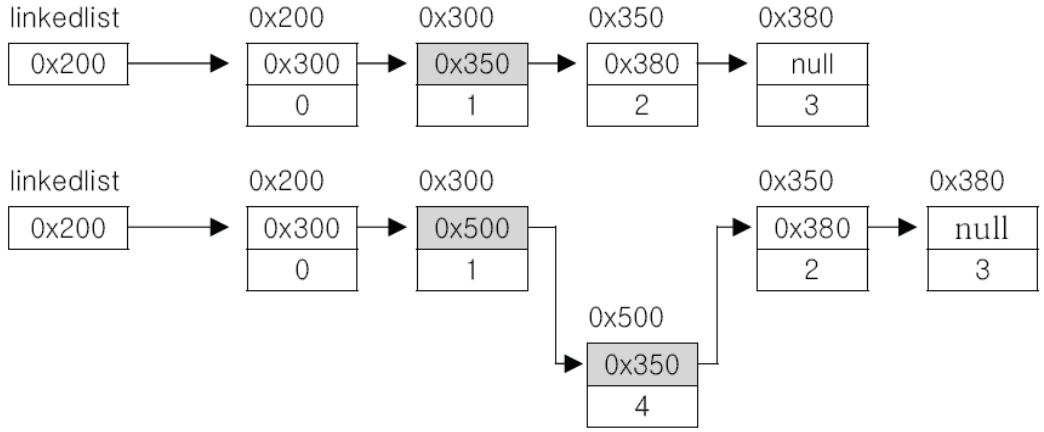
- Unlike arrays, linked lists link data that exists discontinuously.



- Delete Data: by Single change of reference

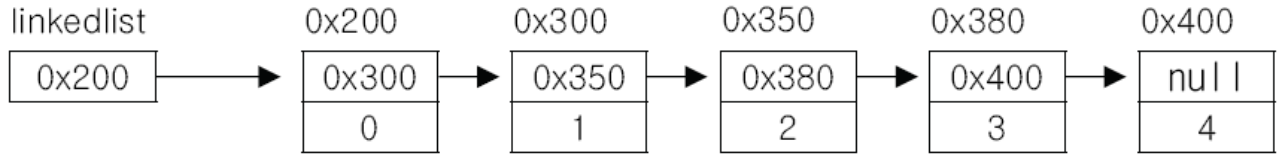


- Insert Data: One node creation + Several changes of references



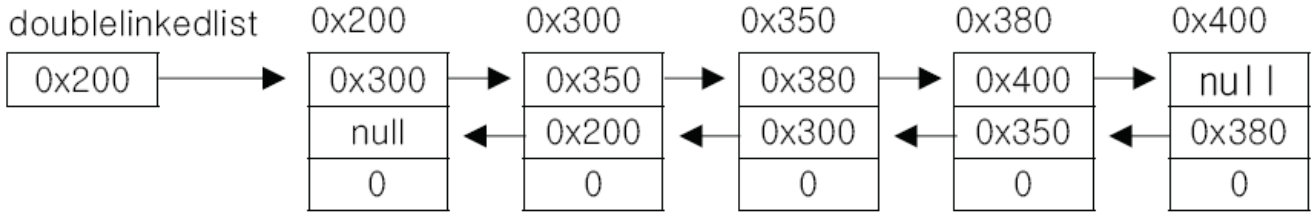
Various LinkedList Types

- Linked List – Hard to access



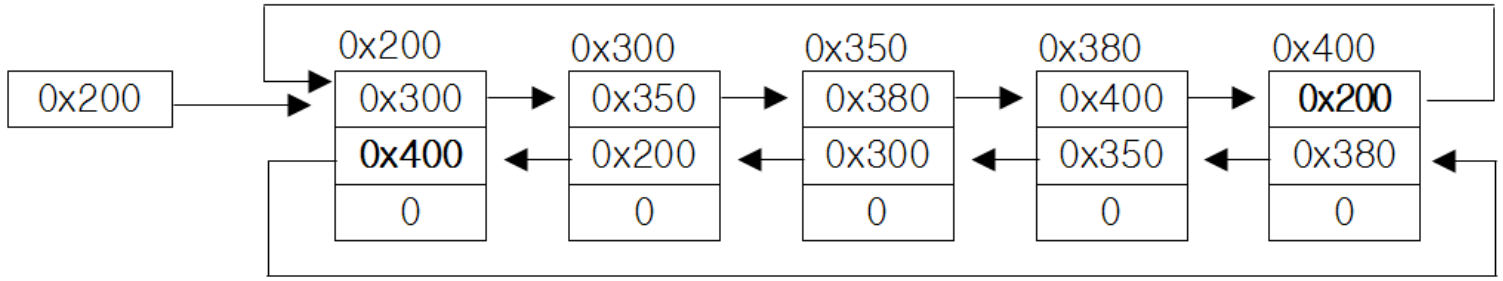

```
class Node {
    Node next;
    Object obj;
}
```

- Doubly linked list – Better accessibility




```
class Node {
    Node next;
    Node previous;
    Object obj;
}
```

- Doubly circular linked list



Example: LinkedList - 1

```
import java.util.LinkedList;

public class CollectionLinkedList1 {
   public static void main(String args[]) {

      LinkedList<String> ll = new LinkedList<String>();

      ll.add("A");       // [A]
      ll.add("B");       // [A, B]
      ll.addLast("C");   // [A, B, C]
      ll.addFirst("D");  // [D, A, B, C]
      ll.add(2, "E");    // [D, A, E, B, C]
      System.out.println(ll);

      ll.remove("B");    // [D, A, E, C]
      ll.remove(3);      // [D, A, E]
      ll.removeFirst();  // [A, E]
      ll.removeLast();   // [A]
      System.out.println(ll);
   }
}
```

Example: LinkedList - 2

```
import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");    // [Geeks]
        ll.add("Heeks");    // [Geeks, Heeks]
        ll.add(1, "Jeeks"); // [Geeks, Jeeks, Heeks]

        System.out.println("Initial LinkedList " + ll);

        ll.set(1, "For");   // [Geeks, For, Heeks]

        System.out.println("Updated LinkedList " + ll);
    }
}
```

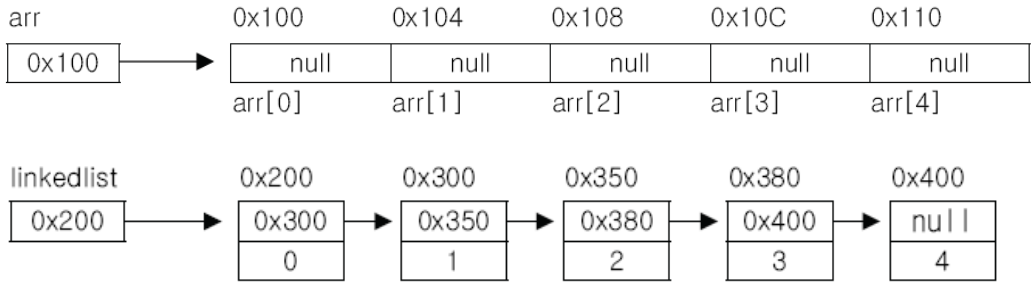
Example: LinkedList - 3

```
import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");
        ll.add("Geeks");
        ll.add(1, "For");

        for (int i = 0; i < ll.size(); i++) {
            System.out.print(ll.get(i) + " "); //Geeks For Geeks
        }

        System.out.println();

        for (String str : ll)
            System.out.print(str + " ");     // Geeks For Geeks
    }
}
```

(More) Methods in class LinkedList

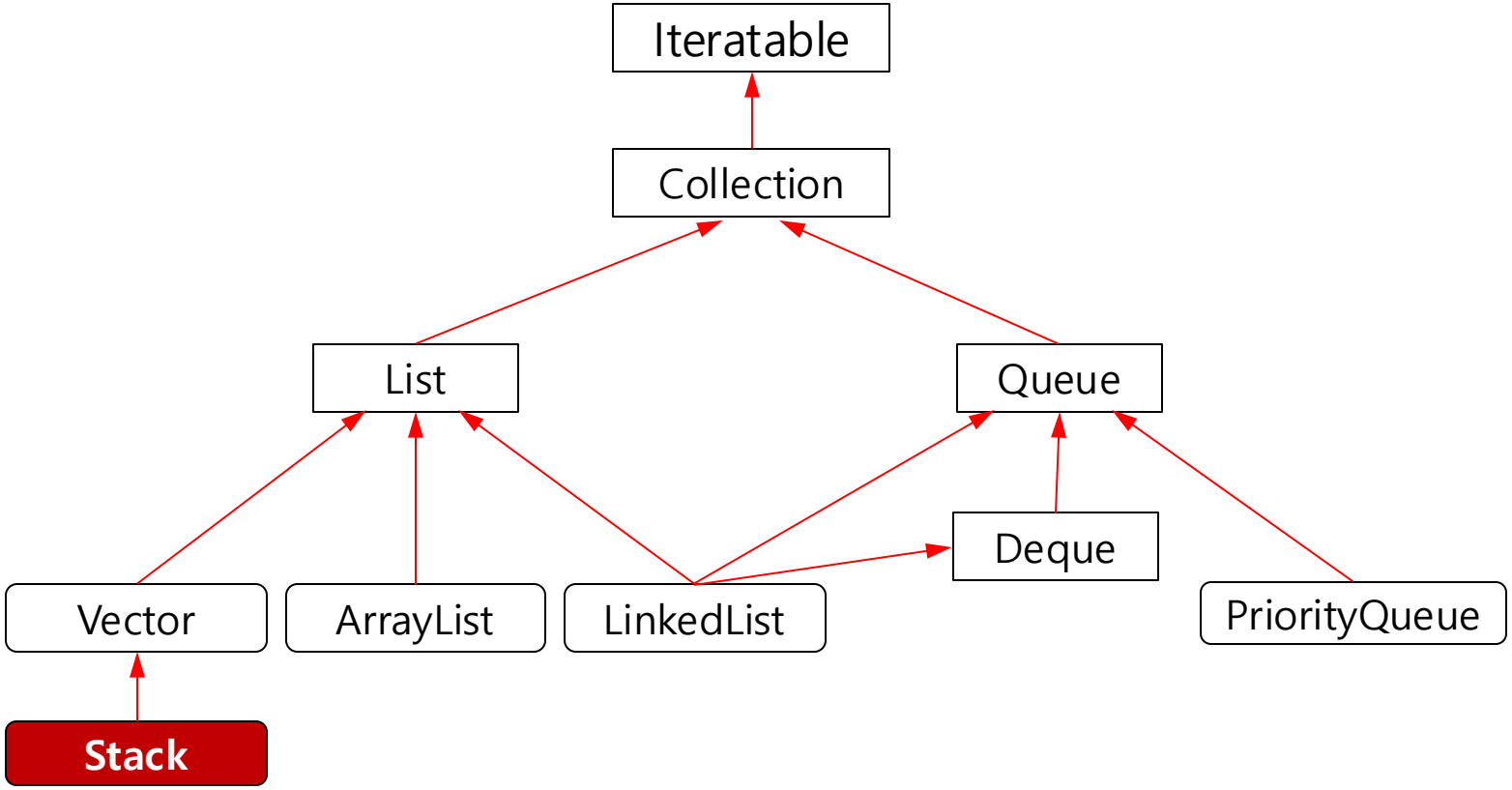
Type	Method	Description
void	addFirst(E e)	Inserts the specified element at the beginning of this list.
void	addLast(E e)	Appends the specified element to the end of this list.
void	clear()	Removes all of the elements from this list.
Object	clone()	Returns a shallow copy of this LinkedList.
E	element()	Retrieves, but does not remove, the head (first element) of this list.
E	getFirst()	Returns the first element in this list.
E	getLast()	Returns the last element in this list.
int	lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	offer(E e)	Adds the specified element as the tail (last element) of this list.
boolean	offerFirst(E e)	Inserts the specified element at the front of this list.
boolean	offerLast(E e)	Inserts the specified element at the end of this list.
E	peek()	Retrieves, but does not remove, the head (first element) of this list.
E	peekFirst()	Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E	peekLast()	Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
E	poll()	Retrieves and removes the head (first element) of this list.

ArrayList vs. LinkedList

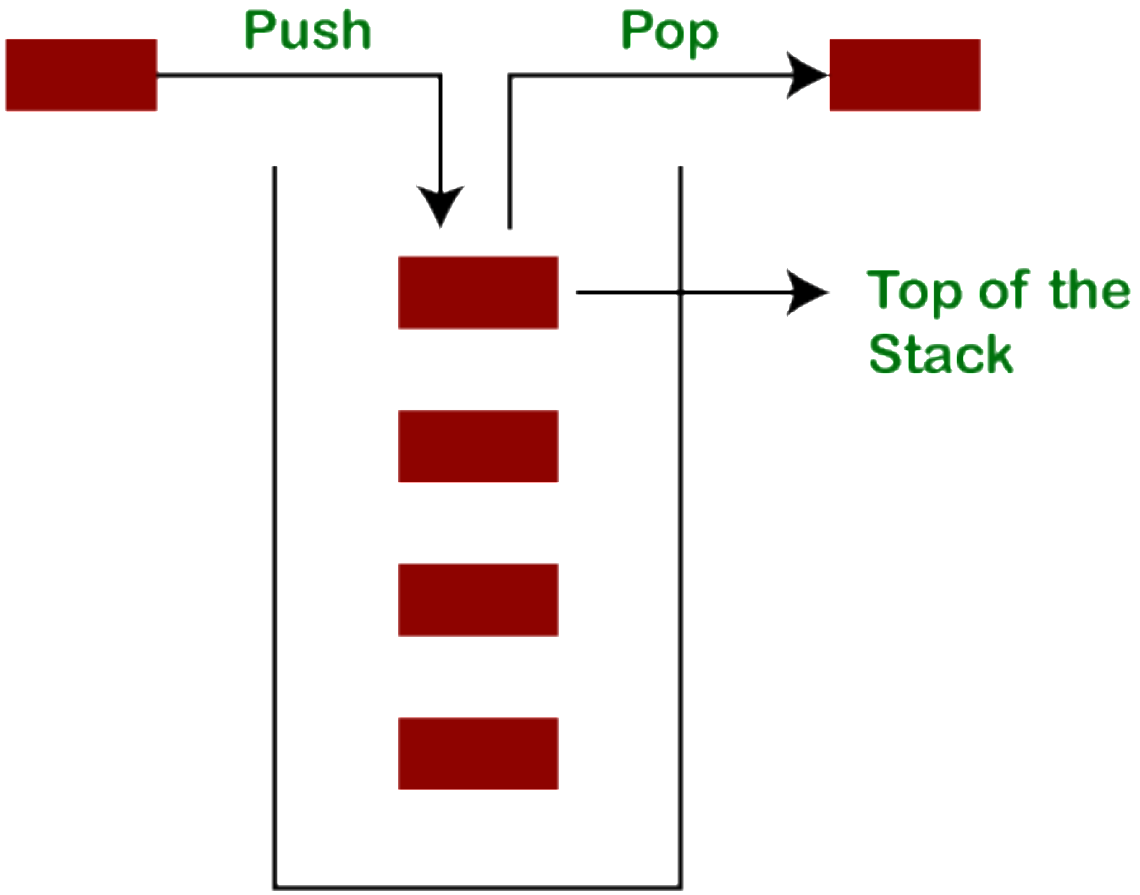


Operation	Worst case time (ArrayList)	Worst case time (LinkedList)
Finding an element	n – 1	n – 1
Add an element at tail	C	n – 1
Add an element at head	n – 1	C
Find and Remove an element	n – 1	(n – 1) + C
Remove an element at tail	C	(n – 1) + C
Remove an element at head	n – 1	C
Access i'th element	C	n – 1

Stack (Class)


Iteratable
Collection
List
Queue
Deque
Vector
ArrayList
LinkedList
PriorityQueue
Stack

Stack (Class)

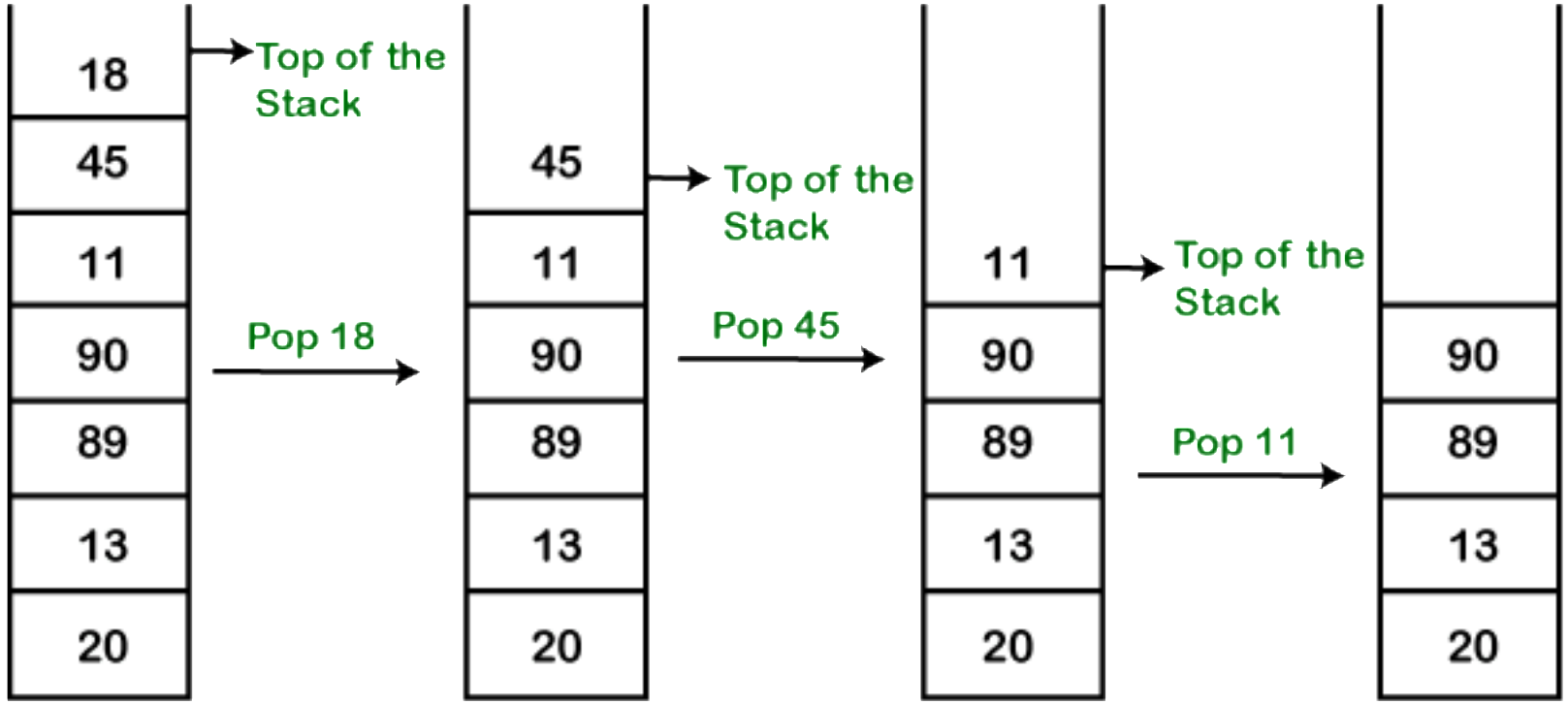


Stack (Class): Push

- Push 20, 13, 89, 90, 11, 45, 18

Stack (Class): Pop

- Pop 18, 45, 11 from the stack



Pop operation

(More) Methods in class Stack

- Stack is implemented on Vector class, so it has all the methods of Vector class
- More methods in Stack:

Method	Type	Description
empty()	boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the removed element
peek()	E	The method looks at the top element of the stack without removing it
search(Object o)	int	The method searches the specified object and returns the position of the object.

Example: Stack - 1

```
import java.util.Stack;

public class StackEmptyMethodExample {
   public static void main(String[] args) {
      //creating an instance of Stack class
      Stack<Integer> stk= new Stack<Integer>();

      // checking stack is empty or not
      System.out.println("Is the stack empty? " + stk.empty());

      // pushing elements into stack
      stk.push(78);
      stk.push(113);

      //prints elements of the stack
      System.out.println("Elements in Stack: " + stk);
      System.out.println("Is the stack empty? " + stk.empty());
   }
}
```

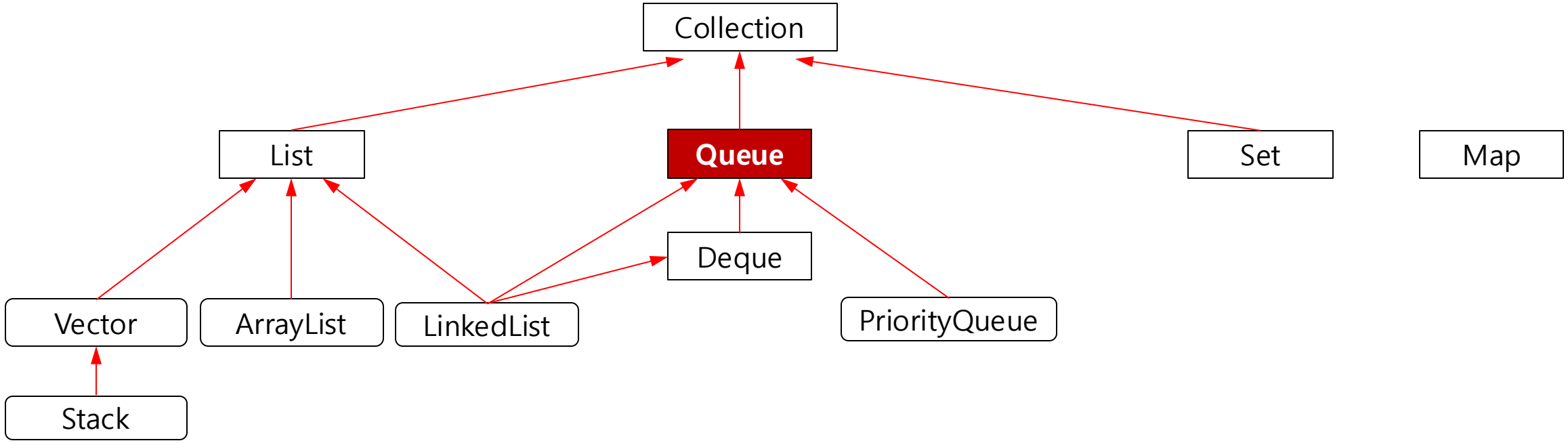
```
Is the stack empty? true
Elements in Stack: [78, 113]
Is the stack empty? false
```

Example: Stack - 2

```
import java.util.*;

public class StackPushPopExample {

   public static void main(String args[]) {

      Stack <Integer> stk = new Stack<Integer>();
      System.out.println("stack: " + stk);

      stk.push(20);
      stk.push(13);
      stk.push(89);
      stk.push(90);
      System.out.println("stack: " + stk);

      System.out.println(stk.pop() + " popped");
      System.out.println(stk.pop() + " popped");
      System.out.println("stack: " + stk);
   }
}
```

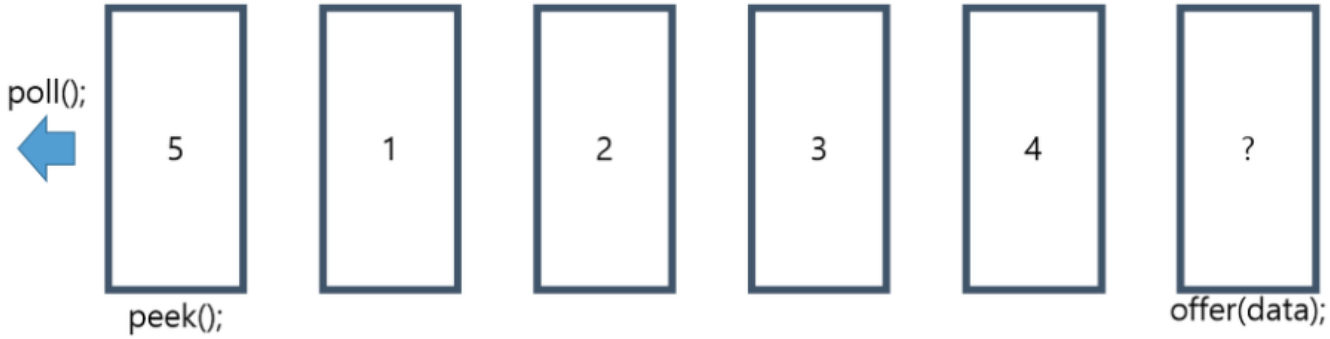
```
stack: []
stack: [20, 13, 89, 90]
90 popped
89 popped
stack: [20, 13]
```

Interface Queue<E>



Interface Queue

- First In First Out (FIFO)
- Add data at one end (offer)
- Remove data at the other end (poll)
- Observe data without removal (peek)



- LinkedList is used in many cases for implementing Queue

Operation	Worst case time (ArrayList)	Worst case time (LinkedList)
Add an element at tail (offer)	C	n – 1
Remove an element at head (poll)	n – 1	C

Methods in Interface Queue<E>

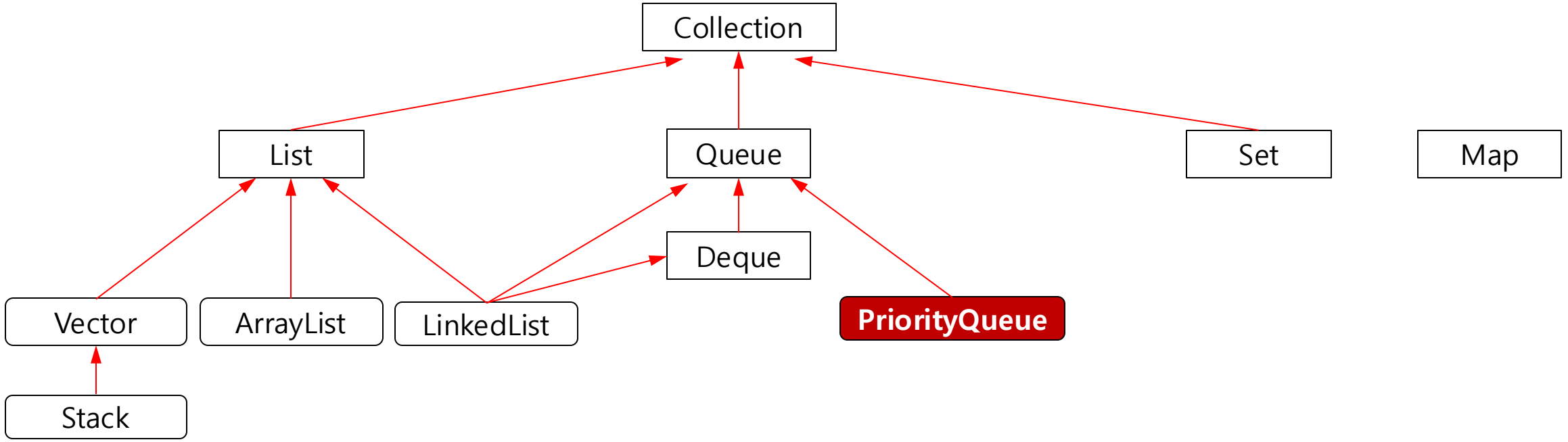
Type	Method	Description
boolean	add(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
E	element()	Retrieves, but does not remove, the head of this queue.
boolean	offer(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	peek()	Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	poll()	Retrieves and removes the head of this queue, or returns null if this queue is empty.
E	remove()	Retrieves and removes the head of this queue.

Queue by LinkedList

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueLinkedList {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();  // creating queue
        queue.offer(5);  queue.offer(1);  queue.offer(2);  queue.offer(3);  queue.offer(4); // pushing 5 data
        System.out.println(queue);
        System.out.print("poll: " + queue.poll() + " "); System.out.println(queue); //poll
        System.out.print("poll: " + queue.poll() + " "); System.out.println(queue); //poll
        System.out.print("poll: " + queue.poll() + " "); System.out.println(queue); //poll
        System.out.print("peek: " + queue.peek() + " "); System.out.println(queue); //peek
        System.out.print("peek: " + queue.peek() + " "); System.out.println(queue); //peek
    }
}
```

```
[5, 1, 2, 3, 4]
poll: 5 [1, 2, 3, 4]
poll: 1 [2, 3, 4]
poll: 2 [3, 4]
peek: 3 [3, 4]
peek: 3 [3, 4]
```

class PriorityQueue



PriorityQueue

- (Minimum) Priority Queue: default
  - Minimum data always at the head (NOTE: not completely sorted)
  - After removing head (remove(), poll()), minimum of remaining data at the head
  - ex) PriorityQueue<Integer> pq = **new** PriorityQueue<Integer>();

- (Maximum) Priority Queue
  - Maximum data always at the head (NOTE: not completely sorted)
  - ex) PriorityQueue<Integer> pqr = **new** PriorityQueue<Integer>(Collections.*reverseOrder*());

Methods in class PriorityQueue<E>

Modifier and Type	Method	Description
boolean	add(E e)	Inserts the specified element into this priority queue.
void	clear()	Removes all of the elements from this priority queue.
boolean	contains(Object o)	Returns true if this queue contains the specified element.
Iterator<E>	iterator()	Returns an iterator over the elements in this queue.
boolean	offer(E e)	Inserts the specified element into this priority queue.
boolean	remove(Object o)	Removes a single instance of the specified element from this queue, if it is present.
Object[]	toArray()	Returns an array containing all of the elements in this queue.

PriorityQueue

```
import java.util.*;

public class PriorityQueueTest {
	public static void main(String args[]){
		PriorityQueue<Integer> queue=new PriorityQueue<Integer>();
		queue.add(4);
		System.out.println("Adding 4: " + queue);
		queue.add(7);
		System.out.println("Adding 7: " + queue);
		queue.add(2);
		System.out.println("Adding 2: " + queue);
		queue.add(5);
		System.out.println("Adding 5: " + queue);
		queue.add(9);
		System.out.println("Adding 9: " + queue);
		System.out.print("Removing " + queue.peek() + ": ");
		queue.remove();
		System.out.println(queue);
		System.out.println("Removing " + queue.poll() + ": " + queue);
	}
}
```

```
Adding 4: [4]

Adding 7: [4, 7]

Adding 2: [2, 7, 4]

Adding 5: [2, 5, 4, 7]

Adding 9: [2, 5, 4, 7, 9]

Removing 2: [4, 5, 9, 7]

Removing 4: [5, 7, 9]
```

Interface Iterator

- Interface used to access (in order if exists) the data stored in the collection
- Generating an iterator:
  - ex) // let al be any collection (list, queue, ...)

    Iterator itr = al.iterator();

- Methods in Iterator:

메서드	설 명
boolean hasNext( )	읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next( )	다음 요소를 읽어 온다. next( )를 호출하기 전에 hasNext( )를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
void remove( )	next( )로 읽어 온 요소를 삭제한다. next( )를 호출한 다음에 remove( )를 호출해야한다.(선택적 기능)

Example: Using Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorTest {

  public static void main(String args[]) {
    ArrayList al = new ArrayList();
    al.add("C");
    al.add("A");
    al.add("E");
    al.add("B");
    al.add("D");
    al.add("F");

    System.out.print("Original contents of al: ");
    Iterator itr = al.iterator();

    while(itr.hasNext()) {
      Object element = itr.next();
      System.out.print(element + " ");
    }
  }
}
```

```
Original contents of al: C A E B D F
```

ListIterator

- Improves the accessibility of Iterator (unidirectional → bidirectional)

메서드	설 명
boolean hasNext( )	읽어 올 다음 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
boolean hasPrevious( )	읽어 올 이전 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next( )	다음 요소를 읽어 온다. next( )를 호출하기 전에 hasNext( )를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
Object previous( )	이전 요소를 읽어 온다. previous( )를 호출하기 전에 hasPrevious( )를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
int nextIndex( )	다음 요소의 index를 반환한다.
int previousIndex( )	이전 요소의 index를 반환한다.
void add(Object o)	컬렉션에 새로운 객체(o)를 추가한다.(선택적 기능)
void remove( )	next( ) 또는 previous( )로 읽어 온 요소를 삭제한다. 반드시 next( )나 previous( )를 먼저 호출한 다음에 이 메서드를 호출해야한다.(선택적 기능)
void set(Object o)	next( ) 또는 previous( )로 읽어 온 요소를 지정된 객체(o)로 변경한다. 반드시 next( )나 previous( )를 먼저 호출한 다음에 이 메서드를 호출해야한다.(선택적 기능)

Example: Using ListIterator

```
import java.util.*;

public class ListIteratorTest {
  public static void main(String args[]) {
    ArrayList al = new ArrayList();
    al.add("C");
    al.add("A");
    al.add("E");

    ListIterator litr = al.listIterator();
    while(litr.hasNext()) {
      Object element = litr.next();
      litr.set(element + "+");
    }

    System.out.print("Modified List backwards: ");
    while(litr.hasPrevious()) {
      Object element = litr.previous();
      System.out.print(element + " ");
    }
    System.out.println();
  }
}
```

```
Modified List backwards: E+ A+ C+
```

java.util.Arrays Class (1/3)

- Offer the static methods to manipulate arrays easily

```
import java.util.Arrays;
...

double[] values = {1.0, 1.1, 1.2};
int[][] arr = {{1,2,3,4,5},{5,4,3,2,1}};

System.out.println(values.toString());              // [D@46a49e6...
System.out.println(Arrays.toString(values));        // [1.0, 1.1, 1.2]
System.out.println(Arrays.deepToString(arr));       // [[1,2,3,4,5],[5,4,3,2,1]]

int[] arr1 = { 1, 2 };
int[] arr2 = { 1, 2 };
int[][] arr3 = {{1,2,3,4,5},{5,4,3,2,1}};

System.out.println(Arrays.equals(arr1, arr2));      // true
System.out.println(Arrays.deepEquals(arr, arr3));   // true
```

java.util.Arrays Class (2/3)

- Copy: copyOf(), copyOfRange()

```
int[] arr = {0,1,2,3,4};
int[] arr2 = Arrays.copyOf(arr, arr.length);      // [0,1,2,3,4]
int[] arr3 = Arrays.copyOf(arr, 3);               // [0,1,2]
int[] arr4 = Arrays.copyOf(arr, 7);               // [0,1,2,3,4,0,0]
int[] arr5 = Arrays.copyOfRange(arr, 2, 4);       // [2,3]
int[] arr6 = Arrays.copyOfRange(arr, 0, 7);       // [0,1,2,3,4,0,0]
```

- Fill: fill(), setAll()

```
int[] arr = new int[5];
Arrays.fill(arr, 9);                                          // [9,9,9,9,9]
Arrays.setAll(arr, () -> (int) (Math.random() * 5) + 1);      // [1,5,2,1,1]
```

java.util.Arrays Class (3/3)

- Convert to List: asList(Object... a)

```
ArrayList<Integer> list = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));
```

- Sort and Search: sort(), binarySearch()

```
int[] arr = {3, 2, 0, 1, 4};

Arrays.sort(arr);                          // sorting the array arr
System.out.println(Arrays.toString(arr));  // [0, 1, 2, 3, 4]
int index = Arrays.binarySearch(arr, 2);   // index = 2
```