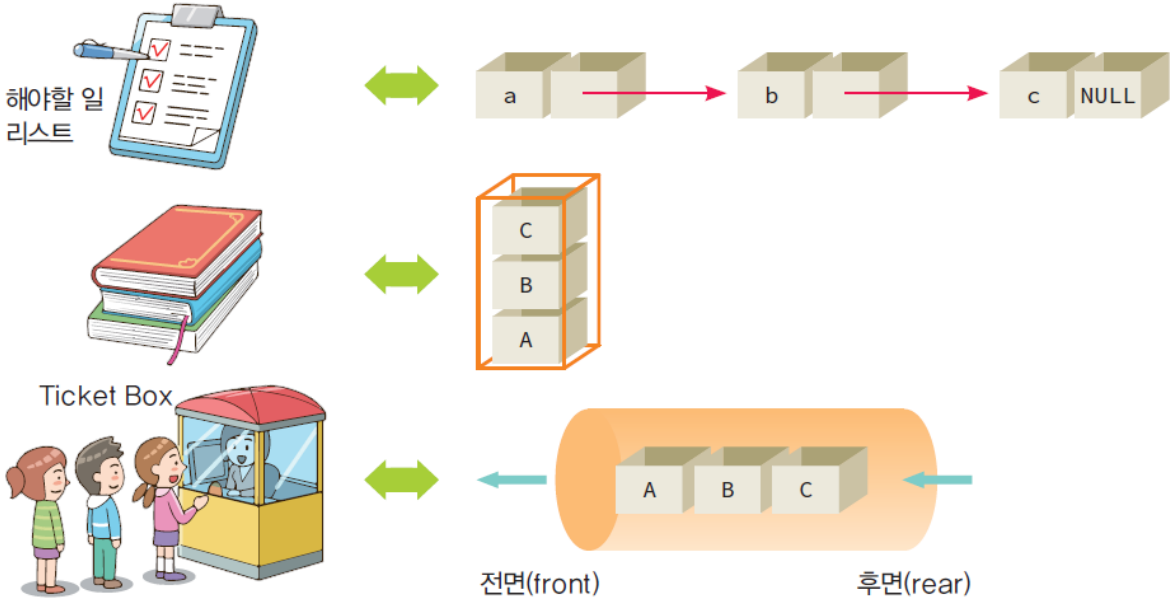
어서와 Java는 처음이지!

제15장 컬렉션

컬렉션

- 컬렉션(collection)은 자바에서 자료 구조를 구현한 클래스
- 자료 구조로는 리스트(list), 스택(stack), 큐(queue), 집합(set), 해쉬 테이블(hash table) 등이 있다.



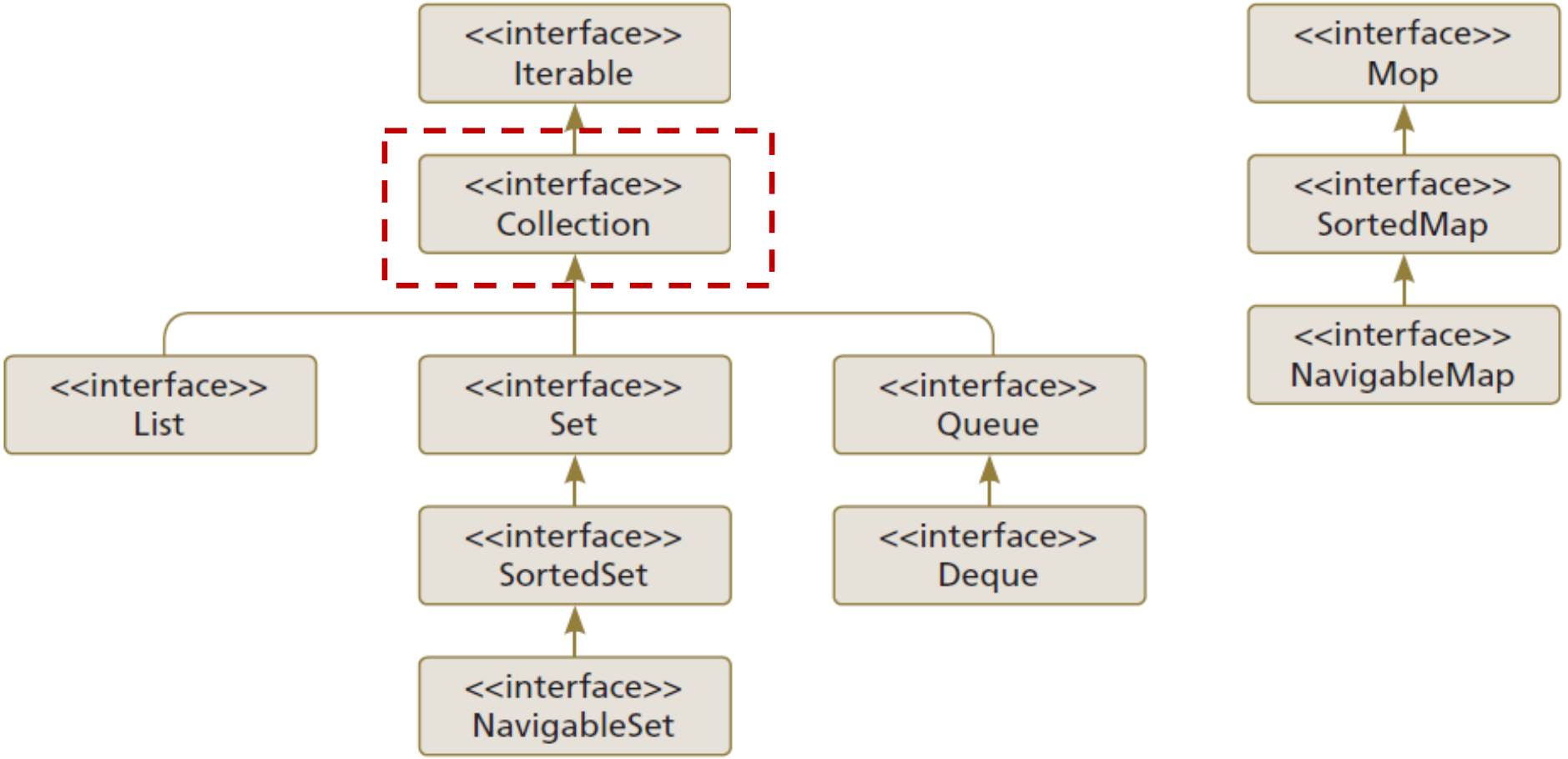
컬렉션 인터페이스와 컬렉션 클래스

- 자바는 컬렉션 인터페이스와 컬렉션 클래스로 나누어서 제공한다. 자바에서는 컬렉션 인터페이스를 구현한 클래스도 함께 제공하므로 이것을 간단하게 사용할 수도 있고 아니면 각자 필요에 맞추어 인터페이스를 자신의 클래스로 구현할 수도 있다.

표 14.1 • 컬렉션 인터페이스

인터페이스	설명
Collection	모든 자료 구조의 부모 인터페이스로서 객체의 모임을 나타낸다.
Set	집합(중복된 원소를 가지지 않는)을 나타내는 자료 구조
List	순서가 있는 자료 구조로 중복된 원소를 가질 수 있다.
Map	키와 값들이 연관되어 있는 사전과 같은 자료 구조
Queue	극장에서의 대기줄과 같이 들어온 순서대로 나가는 자료구조

Collection 인터페이스



Collection 인터페이스

표 14.1 • Collection 인터페이스의 메소드

메소드	설명

`boolean isEmpty()`
`boolean contains(Object obj)`
`boolean containsAll(Collection<?> c)`
	공백 상태이면 true 반환
	obj를 포함하고 있으면 true 반환

`boolean add(E element)` `boolean addAll(Collection<? extends E> from)`	원소를 추가한다.

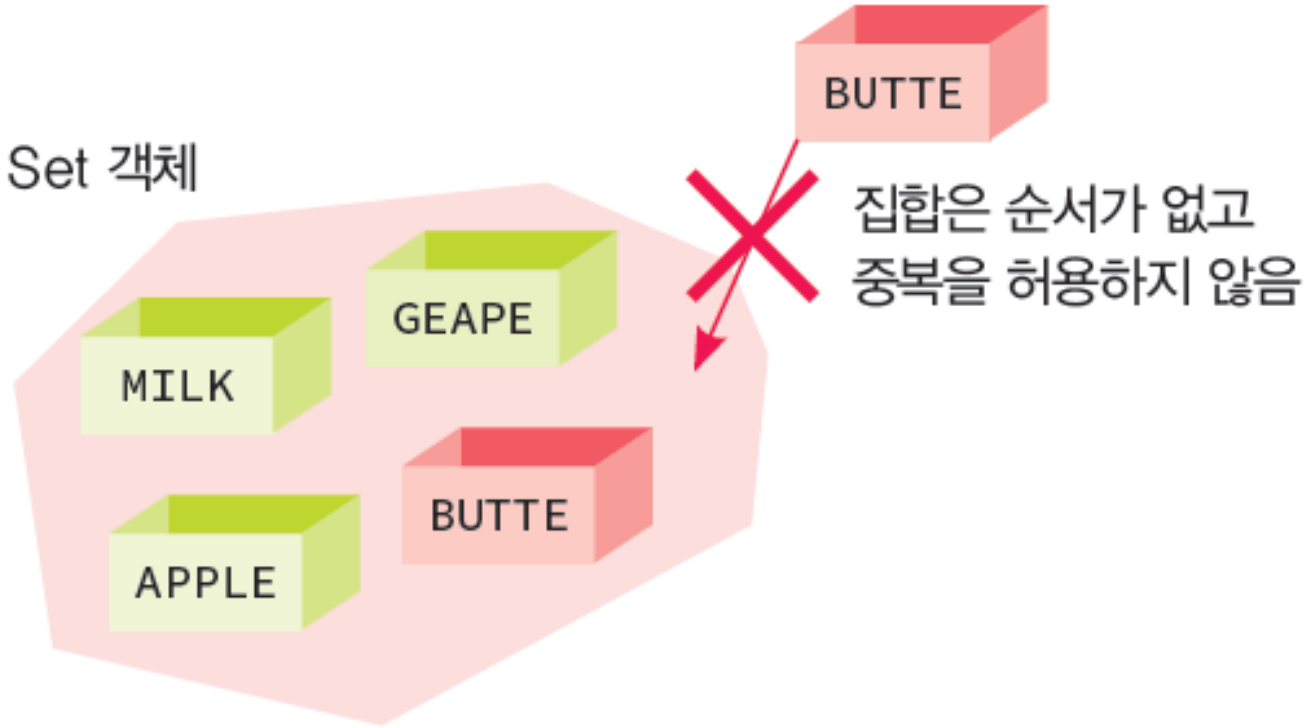
`boolean remove(Object obj)`
`boolean removeAll(Collection<?> c)`
`boolean retainAll(Collection<?> c)`
`void clear()`
	원소를 삭제한다.

`Iterator<E> iterator()`
`Stream<E> stream()`
`Stream<E> parallelStream()`
	원소 방문

`int size()`	원소의 개수 반환
`Object[] toArray()` `<T> T[] toArray(T[] a)`	컬렉션을 배열로 변환

Set

- 집합(Set)은 원소의 중복을 허용하지 않는다.



Set 인터페이스를 구현하는 방법

- HashSet
  - HashSet은 해쉬 테이블에 원소를 저장하기 때문에 성능면에서 가장 우수하다. 하지만 원소들의 순서가 일정하지 않은 단점이 있다.
- TreeSet
  - 레드-블랙 트리(red-black tree)에 원소를 저장한다. 따라서 값에 따라서 순서가 결정되며 하지만 HashSet보다는 느리다.
- LinkedHashSet
  - 해쉬 테이블과 연결 리스트를 결합한 것으로 원소들의 순서는 삽입되었던 순서와 같다.

예제

```
import java.util.*;
public class SetTest {
        public static void main(String args[]) {
                HashSet<String> set = new HashSet<String>();
                set.add("Milk");
                set.add("Bread");
                set.add("Butter");
                set.add("Cheese");
                set.add("Ham");
                set.add("Ham");
                System.out.println(set);
        }
}x
```


```
[Bread, Milk, Butter, Ham, Cheese]
```

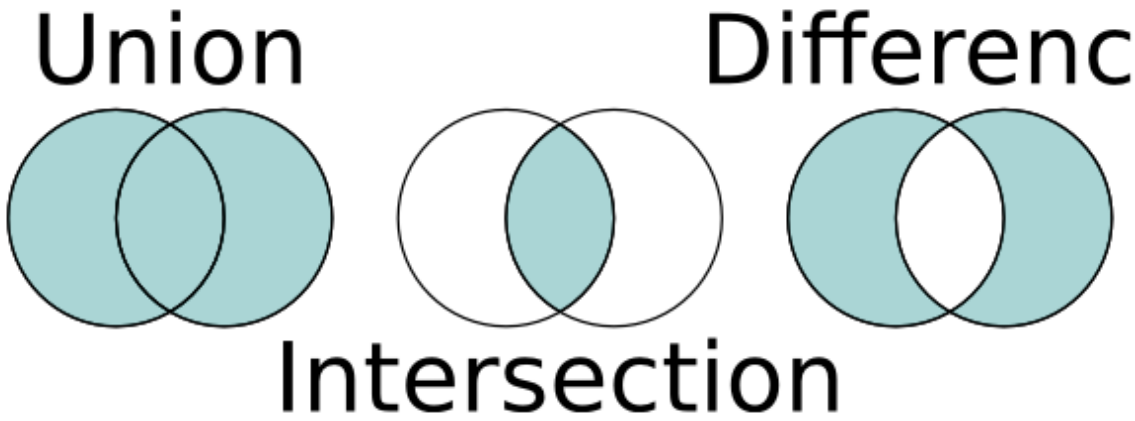
예제

```
import java.util.*;
public class FindDupplication {
        public static void main(String[] args) {
                Set<String> s = new HashSet<String>();
                String[] sample = { "단어", "중복", "구절", "중복" };
                for (String a : sample)
                        if (!s.add(a))
                                System.out.println("중복된 단어 " + a);
                System.out.println(s.size() + " 중복되지 않은 단어: " + s);
        }
}
```


```
중복된 단어 중복
3 중복되지 않은 단어: [중복, 구절, 단어]
```

대량 연산 메소드

- s1.containsAll(s2) - 만약 s2가 s1의 부분 집합이면 참이다.
- s1.addAll(s2) - s1을 s1과 s2의 합집합으로 만든다.
- s1.retainAll(s2) - s1을 s1과 s2의 교집합으로 만든다.
- s1.removeAll(s2) - s1을 s1과 s2의 차집합으로 만든다.



예제

```
public class SetTest1 {
        public static void main(String[] args) {
                Set<String> s1 = new HashSet<String>();
                Set<String> s2 = new HashSet<String>();
                s1.add("A");
                s1.add("B");
                s1.add("C");
                s2.add("A");
                s2.add("D");
                Set<String> union = new HashSet<String>(s1);
                union.addAll(s2);
                Set<String> intersection = new HashSet<String>(s1);
                intersection.retainAll(s2);
                System.out.println("합집합 " + union);
                System.out.println("교집합 " + intersection);
        }
}
```


```
합집합 [D, A, B, C]
교집합 [A]
```

List 인터페이스

- 리스트(List)는 순서를 가지는 요소들의 모임으로 중복된 요소를 가질 수 있다.

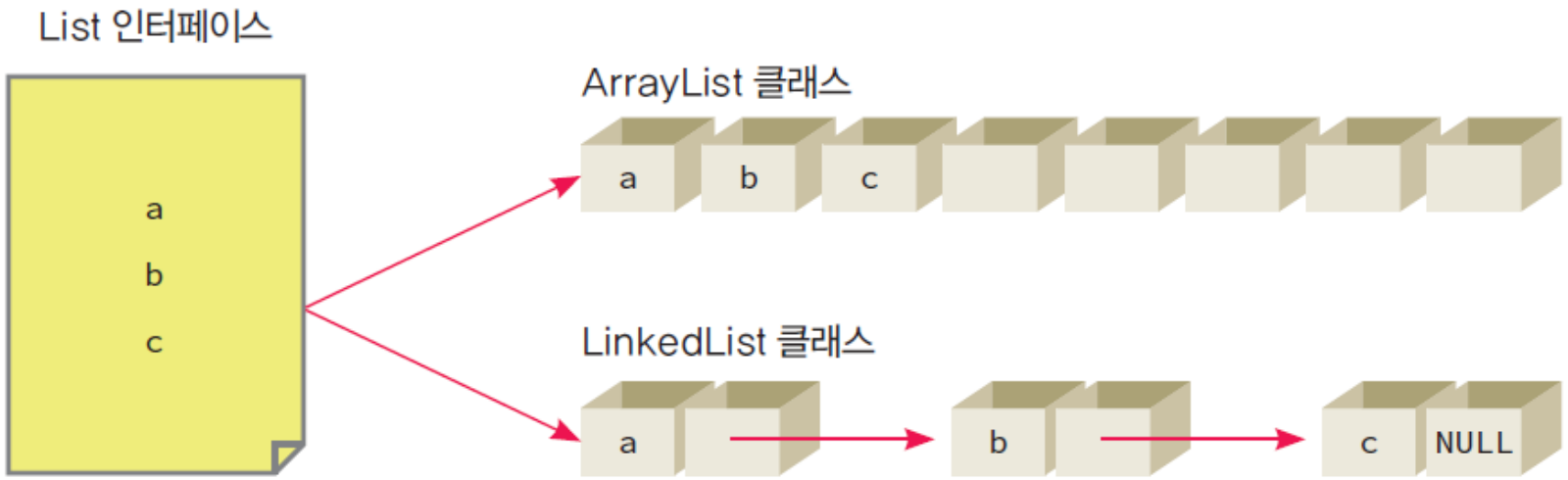
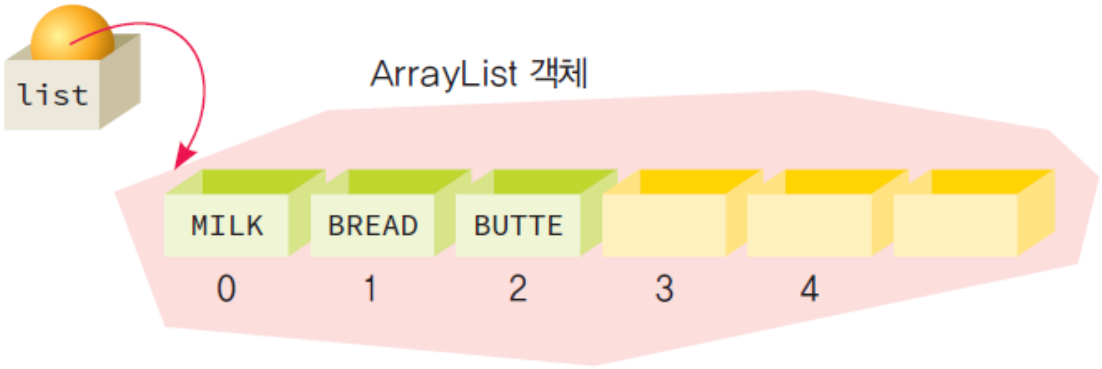


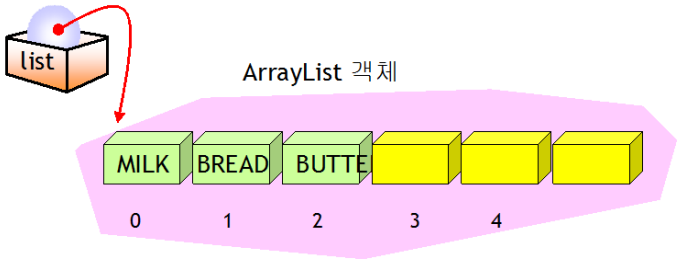
그림 15-6 • 리스트

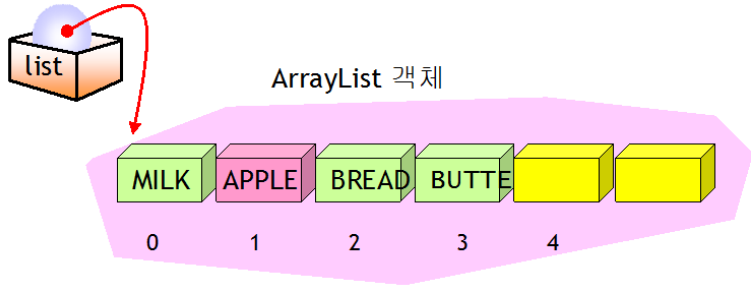
ArrayList

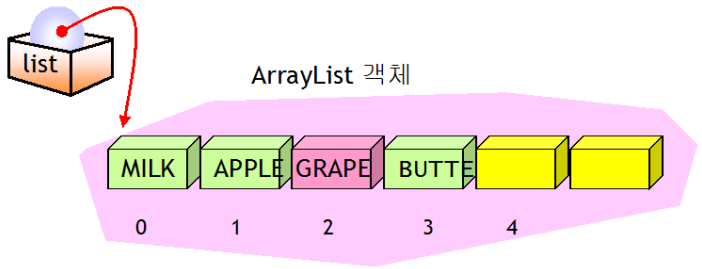
- ArrayList를 배열(Array)의 향상된 버전 또는 가변 크기의 배열이라고 생각하면 된다.
- ArrayList의 생성
  - `ArrayList<String> list = new ArrayList<String>();`
- 원소 추가
  - `list.add( "MILK" );`
  - `list.add( "BREAD" );`
  - `list.add( "BUTTE`

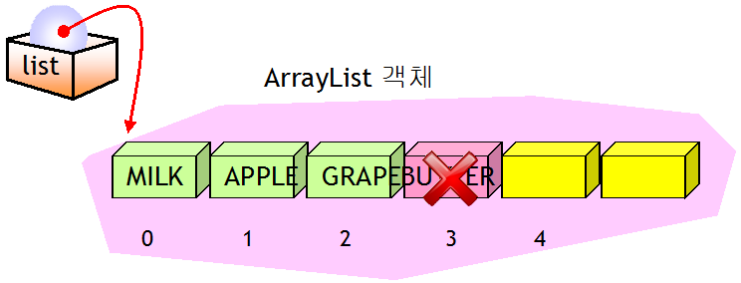


```
ArrayList<String> list = new ArrayList<String>();
list.add( "MILK" );
list.add( "BREAD" );
list.add( "BUTTER" );
list.add( 1, "APPLE" ); // 인덱스 1에 "APPLE"을 삽입
list.set( 2, "GRAPE" ); // 인덱스 2의 원소를 "GRAPE"로 대체
list.remove( 3 ); // 인덱스 3의 원소를 삭제한다.
```



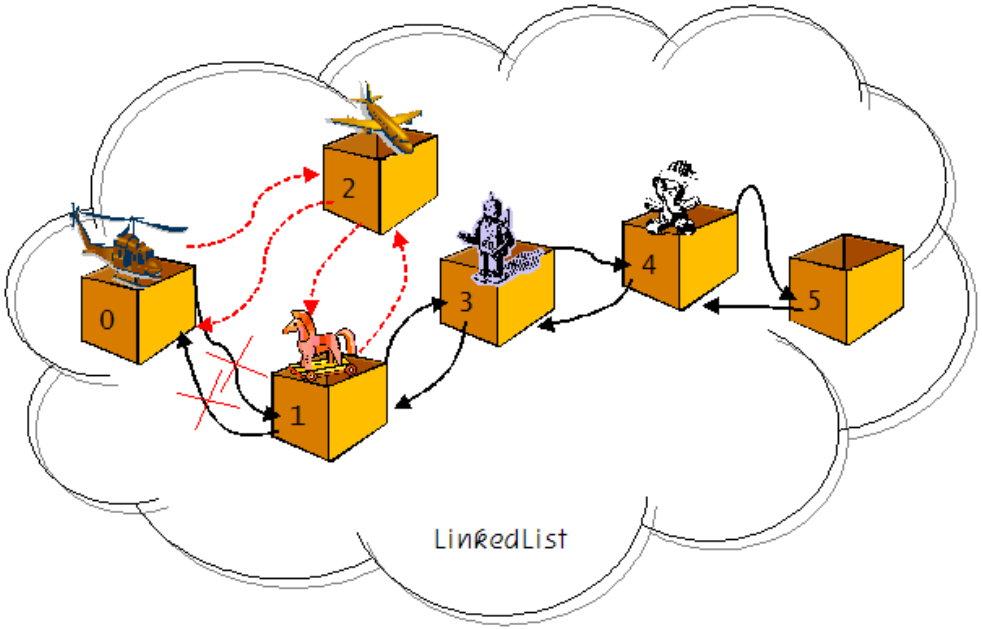






LinkedList

- 빈번하게 삽입과 삭제가 일어나는 경우에 사용



예제

```
import java.util.*;
public class LinkedListTest {
        public static void main(String args[]) {
                LinkedList<String> list = new LinkedList<String>();
                list.add("MILK");
                list.add("BREAD");
                list.add("BUTTER");
                list.add(1, "APPLE");    // 인덱스 1에 "APPLE"을 삽입
                list.set(2, "GRAPE");    // 인덱스 2의 원소를 "GRAPE"로 대체
                list.remove(3);          // 인덱스 3의 원소를 삭제한다.
                for (int i = 0; i < list.size(); i++)
                        System.out.println(list.get(i));
        }
}
```

반복자 사용하기

```
ArrayList<String> list = new ArrayList<String>();
list.add("하나 “);
list.add("둘 “);
list.add("셋 “);
list.add("넷 “);
String s;
Iterator e = list.iterator();
while(e.hasNext())
{
        s = (String)e.next();    // 반복자는 Object 타입을 반환!
        System.out.println(s);
}
```


```
MILK
APPLE
GRAPE
```

배열을 리스트로 변환하기

- List<String> list = Arrays.asList(new String[size]);
  - 일반적인 배열을 리스트로 변환한다.

큐(queue)

- 큐는 후단(tail)에서 원소를 추가하고 전단(head)에서 원소를 삭제한다.

그림 15-11 • 큐

예제

```
import java.util.*;
public class QueueTest {
        public static void main(String[] args) throws InterruptedException {
                int time = 10;
                Queue<Integer> queue = new LinkedList<Integer>();
                for (int i = time; i >= 0; i--)
                        queue.add(i);
                while (!queue.isEmpty()) {
                        System.out.print(queue.remove()+" ");
                        Thread.sleep(1000);    // 현재의 스레드를 1초간 재운다.
                }
        }
}
```


```
10 9 8 7 6 5 4 3 2 1 0
```

우선순위큐

- 우선 순위큐는 원소들이 무작위로 삽입되었더라도 정렬된 상태로 원소들을 추출한다. 즉 remove()를 호출할 때마다 가장 작은 원소가 추출된다.
- 우선 순위큐는 히프(heap)라고 하는 자료 구조를 내부적으로 사용한다.

예제

```
import java.util.*;

public class PriorityQueueTest {
	public static void main(String[] args) {
		PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
		pq.add(30);
		pq.add(80);
		pq.add(20);

		for (Integer o : pq)
			System.out.println(o);
		System.out.println("원소 삭제");
		while (!pq.isEmpty())
			System.out.println(pq.remove());
	}
}
```


```
20
80
30
원소 삭제
20
30
80
```

Map

- Map은 많은 데이터 중에서 원하는 데이터를 빠르게 찾을 수 있는 자료 구조이다.
- 맵은 사전과 같은 자료 구조이다.

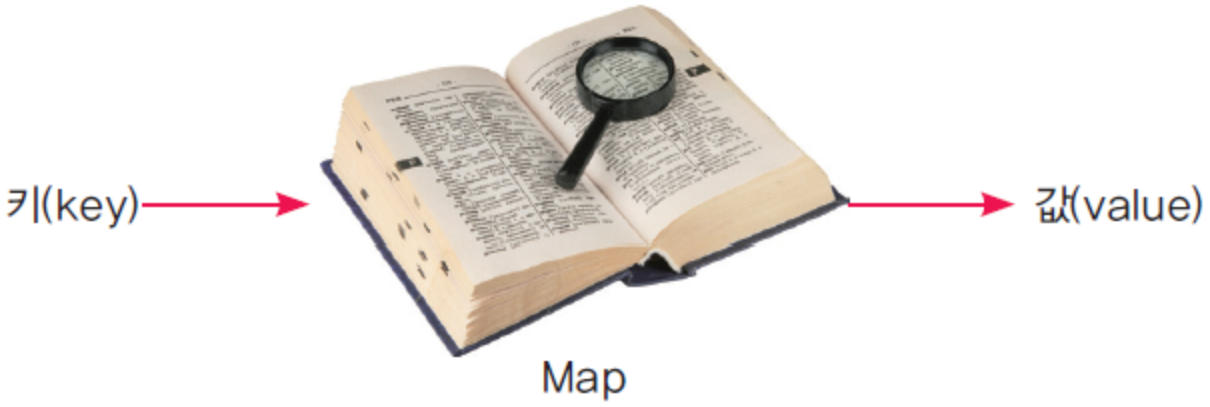


그림 15-17 • Map의 개념

예제

```
import java.util.*;

class Student {
    int number;
    String name;

    public Student(int number, String name) {
        this.number = number;
        this.name = name;
    }

    public String toString() {
        return name;
    }
}
```

```
public class MapTest {
    public static void main(String[] args) {
        Map<String, Student> st = new HashMap<String, Student>();
        st.put("20090001", new Student(20090001, "구준표"));
        st.put("20090002", new Student(20090002, "금잔디"));
        st.put("20090003", new Student(20090003, "윤지후"));

        // 모든 항목을 출력한다.
        System.out.println(st);

        // 하나의 항목을 삭제한다.
        st.remove("20090002");
        // 하나의 항목을 대치한다.
        st.put("20090003", new Student(20090003, "소이정"));
        // 값을 참조한다.
        System.out.println(st.get("20090003"));
        // 모든 항목을 방문한다.
        for (Map.Entry<String, Student> s : st.entrySet()) {
            String key = s.getKey();
            Student value = s.getValue();
            System.out.println("key=" + key + ", value=" + value);
        }
    }
}
```

예제


```
{20090001=구준표, 20090002=금잔디, 20090003=윤지후}
소이정
key=20090001, value=구준표
key=20090003, value=소이정
```

Collections 클래스

- Collections 클래스는 여러 유용한 알고리즘을 구현한 메소드들을 제공한다.

- 정렬(Sorting)
- 섞기(Shuffling)
- 탐색(Searching)

정렬

- 정렬은 데이터를 어떤 기준에 의하여 순서대로 나열하는 것이다.

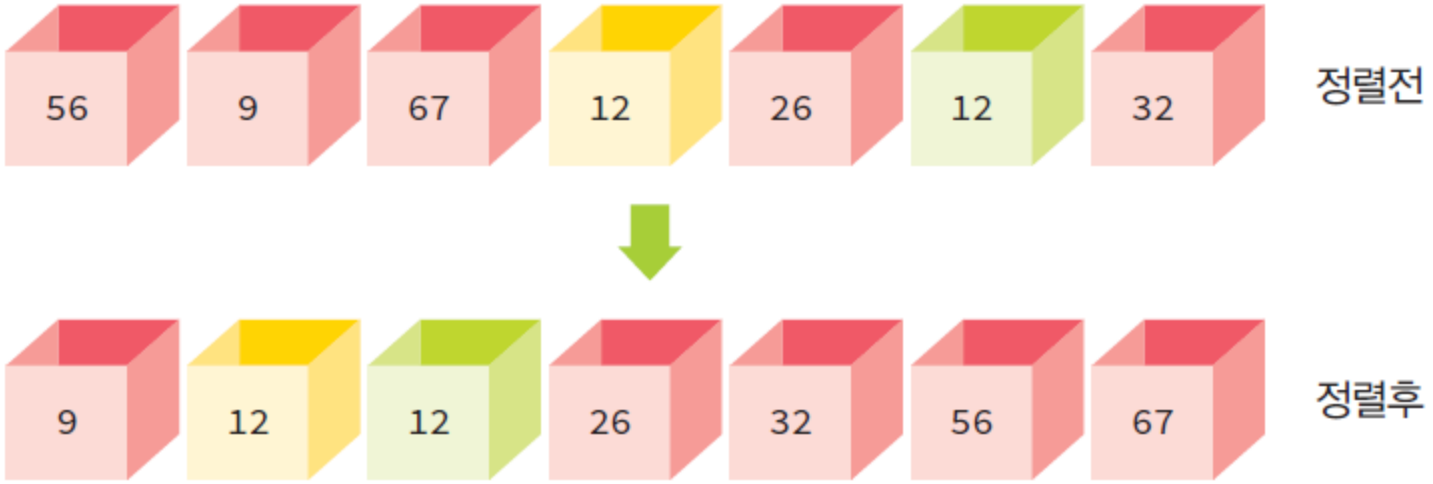


그림 15-13 • 안정된 정렬

예제

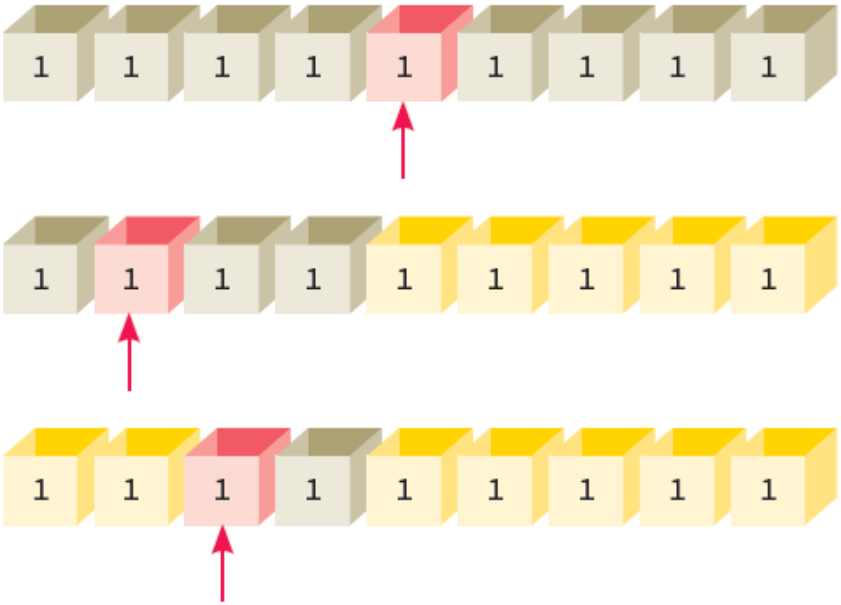
```
import java.util.*;

public class Sort {
        public static void main(String[] args) {
                String[] sample = { "i", "walk", "the", "line" };
                List<String> list = Arrays.asList(sample);          // 배열을 리스트로 변경
                Collections.sort(list);
                System.out.println(list);
        }
}
```


```
[i, line, the, walk]
```

탐색

- 탐색이란 리스트 안에서 원하는 원소를 찾는 것이다.



예제

```
import java.util.*;

public class Search {
	public static void main(String[] args) {
		int key = 50;
		List<Integer> list = new ArrayList<Integer>();
		for (int i = 0; i < 100; i++)
			list.add(i);
		int index = Collections.binarySearch(list,key);
		System.out.println("탐색의 반환값 =" + index);
	}
}
```

실행결과

```
탐색의 반환값 =50
```

LAB: 영어사전의 구현

- 여기서는 Map을 사용하여서 영어 사전을 구현하여 보자. 사용자가 단어를 입력하면 단어의 설명을 보여준다.


```
영어 단어를 입력하시오:map
단어의 의미는 지도
영어 단어를 입력하시오:school
단어의 의미는 학교
영어 단어를 입력하시오:quit
```

예제

```
import java.util.*;

public class EnglishDic {
      public static void main(String[] args) {
          Map<String, String> st = new HashMap<String, String>();

          st.put("map", "지도");
            st.put("java", "자바");
            st.put("school", "학교");

            Scanner sc = new Scanner(System.in);
            do {
                System.out.print("영어 단어를 입력하시오:");
                String key = sc.next();
                if( key.equals("quit") ) break;
                System.out.println("단어의 의미는 " + st.get(key));
            } while(true);
       }
}
```