

제네릭과 컬렉션

강사 : 강병준

컬렉션 프레임워크(collection framework)이란?

▶ 컬렉션 프레임워크(collection framework)

- 데이터 군(群)을 저장하는 클래스들을 표준화한 설계
- 다수의 데이터를 쉽게 처리할 수 있는 방법을 제공하는 클래스들로 구성
- JDK1.2부터 제공

▶ 컬렉션(collection)

- 다수의 데이터, 즉, 데이터 그룹을 의미한다.

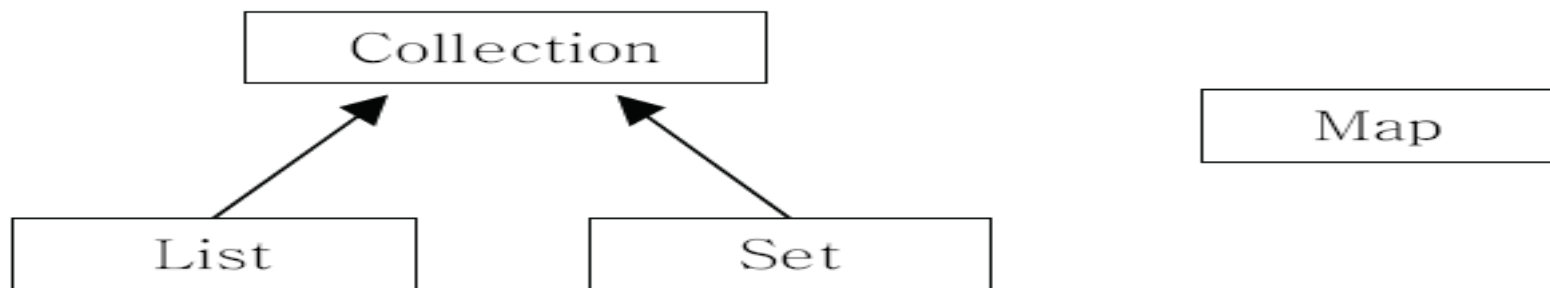
▶ 프레임워크(framework)

- 표준화, 정형화된 체계적인 프로그래밍 방식

▶ 컬렉션 클래스(collection class)

- 다수의 데이터를 저장할 수 있는 클래스(예, Vector, ArrayList, HashSet)

컬렉션 프레임워크의 핵심 인터페이스



컬렉션 프레임워크의 핵심 인터페이스간의 상속계층도

인터페이스	특징
List	순서가 있는 데이터의 집합. 데이터의 중복을 허용한다. 예) 대기자 명단
	구현클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다. 예) 양의 정수집합, 소수의 집합
	구현클래스 : HashSet, TreeSet 등
Map	키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용한다. 예) 우편번호, 지역번호(전화번호)
	구현클래스 : HashMap, TreeMap, Hashtable, Properties 등

컬렉션 프레임워크의 동기화(synchronization)

- 멀티쓰레드 프로그래밍에서는 컬렉션 클래스에 동기화 처리가 필요하다.
- Vector와 같은 구버전 클래스들은 자체적으로 동기화처리가 되어 있다.
- ArrayList와 같은 신버전 클래스들은 별도의 동기화 처리가 필요하다.
- Collections 클래스는 다음과 같은 동기화 처리 메서드를 제공한다.

[주의] java.util.Collection은 인터페이스이고, java.util.Collections는 클래스이다.

```
static Collection synchronizedCollection(Collection c)
static List synchronizedList(List list)
static Map synchronizedMap(Map m)
static Set synchronizedSet(Set s)
static SortedMap synchronizedSortedMap(SortedMap m)
static SortedSet synchronizedSortedSet(SortedSet s)
```

```
List list = Collections.synchronizedList(new ArrayList(...));
```

컬렉션 클래스와 제네릭

메소드	설 명
int size()	요소가 몇 개 들었는지를 반환
boolean isEmpty()	컬렉션이 비었는지를 반환
boolean add(Object element)	요소 추가 성공시 true 반환
boolean remove(Object obj)	요소 삭제 성공시 true 반환
boolean removeAll(Collection other)	요소 전체 삭제
boolean contains(Object obj)	해당 객체가 컬렉션 클래스에 포함되어 있으면 true, 그렇지 않으면 false
Iterator iterator()	Iterator 인터페이스를 얻어냄
Object[] toArray()	컬렉션에 들어 있는 요소를 객체 배열로 바꿈

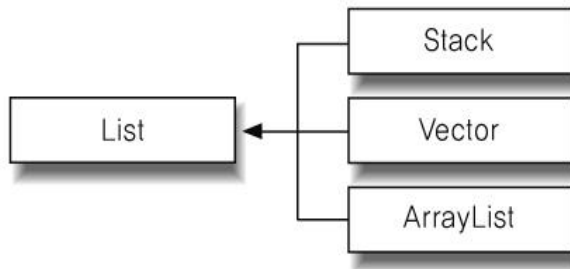
확장 for 문

형식

```
for (자료형 접근 변수명 : 배열이나 컬렉션 변수명) {  
    반복 코드  
}
```

❖ List 인터페이스

⇒ List 구조는 Sequence라고도 하며 시작과 끝이 선정되어 저장되는 요소들을 일괄적인 정렬상태를 유지하면서 요소들의 저장이 이루어진다. 이런 점 때문에 List 구조하면 배열을 영상 하게 되는데 무리는 아니다. 어떻게 보면 배열과 컬렉션의 List 구조는 같다고 볼 수 있으며 다르다면 배열은 크기가 고정되어 있는 것이고 컬렉션의 List 구조는 가변적 길이를 가진다는 것이다. 다음은 List 구조에서 알려진 구현 클래스들이다.



[그림 7-13] List의 구조



[그림 7-14] List의 도식화

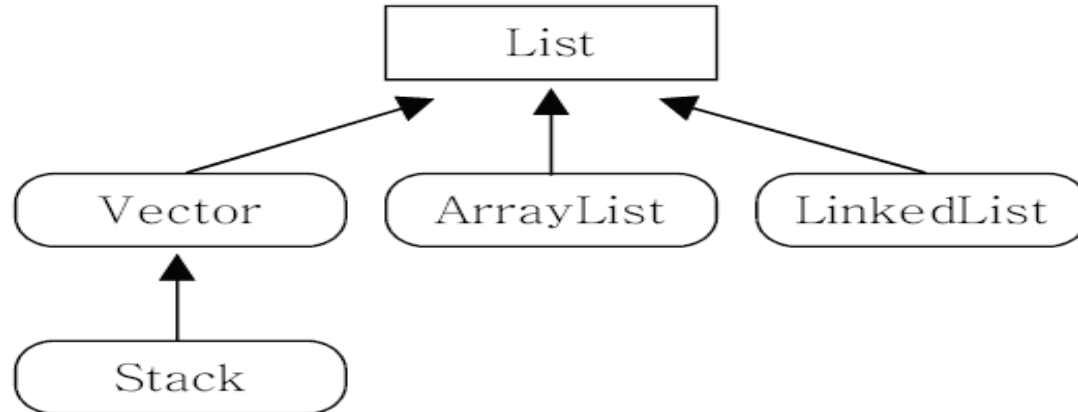
List의 구현 클래스

구현 클래스	설명
Stack	Stack 클래스는 객체들의 last-in-first-out(LIFO) 스택을 표현한다. 그리고 Vector 클래스로부터 파생된 클래스다. 요소를 저장할 때의 push() 메서드와 요소를 빼낼 때의 pop() 메서드 등 총 5개의 메서드를 제공한다.
Vector	배열과 같이 정수 인덱스로 접근할 수 있는 접근 소자를 가지고 있다. 하지만 배열과는 달리 Vector의 크기는 Vector가 생성된 후에 요소를 추가하는 경우에 따라 증대되고 또는 제거할 때에 따라 감소할 수 있다. 그리고 요소들의 작업을 위해 Iterator로 작업할 수 있으며 나중에 배우는 스레드 동기화가 지원되는 List 구조다.
ArrayList	List 인터페이스를 구현하고 있는 것뿐 아니라 ArrayList는 배열의 크기를 조작하기 위하여 메서드들이 제공된다. 공백을 포함한 모든 요소들을 저장할 수 있으며 Vector와 유사하지만 ArrayList는 스레드의 동기화는 지원하지 않는다.

- 이렇게 몇 개의 구현 클래스들을 확인해 보았다. 아무래도 List구조인 객체들은 Set과는 다르게 정렬상태를 유지하면서 각 요소들의 접근력을 Set보다는 쉽게 가지는 구조라 할 수 있겠다

Vector와 ArrayList

- ArrayList는 기존의 Vector를 개선한 것으로 구현원리와 기능적으로 동일
- List인터페이스를 구현하므로, 저장순서가 유지되고 중복을 허용한다.
- 데이터의 저장공간으로 배열을 사용한다.(배열기반)
- Vector는 자체적으로 동기화처리가 되어 있으나 ArrayList는 그렇지 않다.



```
public class Vector extends AbstractList
    implements List, RandomAccess, Cloneable, java.io.Serializable
{
    ...
    protected Object elementData[];
    ...
}
```

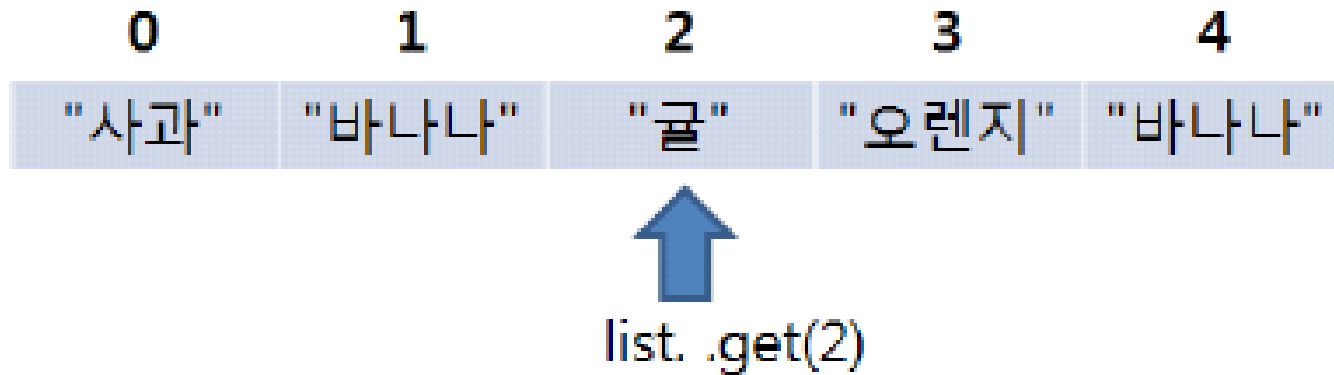

ArrayList 클래스

```
ArrayList<String> list;  
list = new ArrayList<String>( );  
list.add("사과");  
list.add("바나나");  
list.add("귤");  
list.add("오렌지");  
list.add("바나나");
```

0	1	2	3	4
"사과"	"바나나"	"귤"	"오렌지"	"바나나"

get 메소드

```
String item=list.get(2);
```



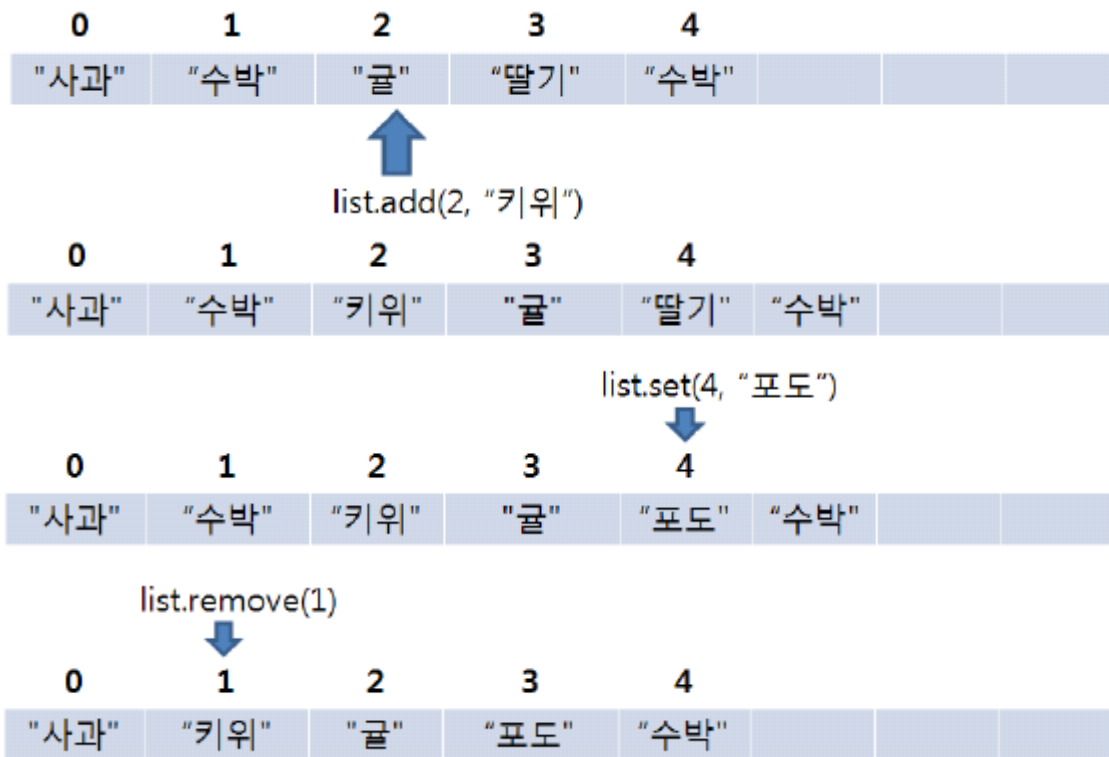
```
import java.util.ArrayList;
import java.util.Iterator;
public class Ex02 {
    public static void main(String[] args) {
        ArrayList<String> list=new ArrayList<String>( );
        list.add("사과");    list.add("바나나"); list.add("귤");
        list.add("오렌지"); list.add("바나나");
        System.out.println("요소의 개수->" + list.size());
        System.out.println(">> Iterator 객체로 요소 얻기 <<");
        Iterator elements=list.iterator();
        while(elements.hasNext())        //요소가 있다면
            System.out.print(elements.next()+"\t"); //요소를 얻어내어 출력
        System.out.println("\n");

        System.out.println(">> get 메소드로 요소 얻기 <<");
        for(int i=0; i<list.size(); i++)
            System.out.print(list.get(i)+"\t");
        System.out.println();
    }
}
```

ArrayList에서 원소 추가, 변경, 삭제하기

메소드	설 명
<code>void add(int index, E element)</code>	index 위치에 element로 주어진 객체를 저장한다. 해당 위치의 객체는 뒤로 밀려난다.
<code>E set(int index, E element)</code>	index 위치의 요소를 element로 주어진 객체를 대체한다.
<code>E remove(int index)</code>	index 위치의 객체를 지운다.

ArrayList에서 원소 추가, 변경, 삭제하기



ArrayList에서 데이터 검색

메소드	설 명
<code>int indexOf(Object o)</code>	전달 인자로 준 객체를 앞에서부터 찾아 해당 위치를 반환. 못 찾으면 -1 반환
<code>int lastIndexOf(Object o)</code>	객체를 마지막 위치부터 찾음

```
import java.util.ArrayList;    import java.util.Enumeraion;
public class Ex03 {
    public static void main(String[] args) {
        ArrayList list=new ArrayList( );
        list.add("사과");    list.add("수박");    list.add("귤");
        list.add("딸기");    list.add("수박");    prn(list);
        System.out.println(">>인덱스 2인 위치에 키위 삽입<<");
        list.add(2, "키위");                prn(list);
        System.out.println(">>인덱스 4인 위치의 데이터를 포도로 변경<<");
        list.set(4, "포도");                prn(list);
        System.out.println(">>인덱스 1인 위치의 데이터를 제거<<");
        list.remove(1);                prn(list);
        System.out.println(">>사과 란 데이터를 찾아서 제거<<");
        list.remove("사과");                prn(list);
    }
    static void prn(ArrayList<String> list)    {
        for(int i=0; i<list.size(); i++)
            System.out.print(list.get(i)+"\t\t");
            System.out.println("\n");
        }
    }
```

Vector 클래스

메소드	설 명
<code>public Vector()</code>	디폴트 생성자로 빈 벡터 객체를 생성한다.
<code>public Vector(int initialCapacity)</code>	<code>initialCapacity</code> 로 지정한 크기의 벡터 객체를 생성한다.
<code>public Vector(int initialCapacity, int capacityIncrement)</code>	<code>initialCapacity</code> 로 지정한 크기의 벡터 객체를 생성하되, 새로운 요소가 추가되어 원소가 늘어나야 하면 <code>capacityIncrement</code> 만큼 늘어난다.

Vector – 주요메서드

메서드	설 명
Vector()	크기가 10인 Vector를 생성한다.
Vector(Collection c)	주어진 컬렉션을 저장할 수 있는 생성자
Vector(int initialCapacity)	Vector의 초기용량을 지정할 수 있는 생성자
Vector(int initialCapacity, int capacityIncrement)	초기용량과 용량의 증분을 지정할 수 있는 생성자
void add(int index, Object element)	지정된 위치(index)에 객체(element)를 저장한다.
boolean add(Object o)	객체(o)를 저장한다.
boolean addAll(Collection c)	주어진 컬렉션의 모든 객체를 저장한다.
boolean addAll(int index, Collection c)	지정된 위치부터 주어진 컬렉션의 모든 객체를 저장한다.
void addElement(Object obj)	객체(obj)를 저장한다. 반환값은 없다.
int capacity()	Vector의 용량을 반환한다.
void clear()	Vector를 비운다.
Object clone()	Vector를 복제한다.
boolean contains(Object elem) boolean containsAll(Collection c)	지정된 객체(elem) 또는 컬렉션(c)의 객체들이 Vector에 포함되어 있는지 확인한다.
void copyInto(Object[] anArray)	Vector에 저장된 객체들을 anArray배열에 저장한다.
Object elementAt(int index)	지정된 위치(index)에 저장된 객체를 반환한다.
Enumeration elements()	Vector에 저장된 모든 객체들을 반환한다.
void ensureCapacity(int minCapacity)	Vector의 용량이 최소한 minCapacity가 되도록 한다.
boolean equals(Object o)	주어진 객체(o)와 같은지 비교한다.
Object firstElement()	첫 번째로 저장된 객체를 반환한다.
Object get(int index)	지정된 위치(index)에 저장된 객체를 반환한다.
int hashCode()	해시 코드를 반환한다.

Vector — 주요메서드

<code>int indexOf(Object elem)</code>	지정된 객체가 저장된 위치를 찾아 반환한다.
<code>int indexOf(Object elem, int index)</code>	지정된 객체가 저장된 위치를 찾아 반환한다.(지정된 위치부터 찾는다.)
<code>void insertElementAt(Object obj, int index)</code>	객체(obj)를 주어진 위치(index)에 삽입한다.
<code>boolean isEmpty()</code>	Vector가 비어있는지 확인한다.
<code>Object lastElement()</code>	Vector에 저장된 마지막 객체를 반환한다.
<code>int lastIndexOf(Object elem)</code>	객체(elem)가 저장된 위치를 끝에서부터 역방향으로 찾는다.
<code>int lastIndexOf(Object elem, int index)</code>	객체(elem)가 저장된 위치를 지정된 위치(index)부터 역방향으로 찾는다.
<code>Object remove(int index)</code>	지정된 위치(index)에 있는 객체를 제거한다.
<code>boolean remove(Object o)</code>	지정한 객체를 제거한다.
<code>boolean removeAll(Collection c)</code>	지정한 컬렉션에 저장된 것과 동일한 객체들을 Vector에서 제거한다.
<code>void removeAllElements()</code>	<code>clear()</code> 와 동일하다.
<code>boolean removeElement(Object obj)</code>	지정된 객체를 삭제한다.
<code>void removeElementAt(int index)</code>	지정된 위치(index)에 저장된 객체를 삭제한다.
<code>boolean retainAll(Collection c)</code>	Vector에 저장된 객체 중에서 주어진 컬렉션과 공통된 것들만을 남기고 나머지는 삭제한다.
<code>Object set(int index, Object element)</code>	주어진 객체(element)를 지정된 위치(index)에 저장한다.
<code>void setElementAt(Object obj, int index)</code>	주어진 객체(obj)를 지정된 위치(index)에 저장한다.
<code>void setSize(int newSize)</code>	Vector의 크기를 지정한 크기(newSize)로 변경한다.
<code>int size()</code>	Vector에 저장된 객체의 개수를 반환한다.
<code>List subList(int fromIndex, int toIndex)</code>	fromIndex부터 toIndex사이에 저장된 객체를 반환한다.
<code>Object[] toArray()</code>	Vector에 저장된 모든 객체들을 객체배열로 반환한다.
<code>Object[] toArray(Object[] a)</code>	Vector에 저장된 모든 객체들을 객체배열 a에 담아 반환한다.
<code>String toString()</code>	저장된 모든 객체를 문자열로 출력한다.
<code>void trimToSize()</code>	용량을 크기에 맞게 줄인다.(빈 공간을 없앤다.)

```
import java.util.*;
class IJ {}
class KL {}
public class Exam_13 {
    public static void main(String[] ar) {
        IJ a = new IJ();           KL b = new KL();
        String c = new String("C"); Vector vc = new Vector();
        ArrayList al = new ArrayList();
        vc.add(a);                 vc.add(b);                 vc.add(c);
        al.add(a);                 al.add(b);                 al.add(c);
        for(int i = 0; i < vc.size(); ++i) {
            Object obj = vc.elementAt(i);
            System.out.println(i + " --> " + obj);
        }
        System.out.println();
        for(int i = 0; i < al.size(); ++i) {
            Object obj = al.get(i);
            System.out.println(i + " --> " + obj);
        }
    }
}
```

```
import java.util.*;
class VectorEx1{
    public static void main(String[] args) {
        Vector v = new Vector(5);      // 용량(capacity)이 5인 Vector를 생성한다.
        v.add("1");          v.add("2");          v.add("3");
        print(v);
        v.trimToSize();      // 빈 공간을 없앤다.(용량과 크기가 같아진다.)
        System.out.println("=== After trimToSize() ===");
        print(v);
        v.ensureCapacity(6); System.out.println("=== After ensureCapacity(6) ===");
        print(v);
        v.setSize(7);        System.out.println("=== After setSize(7) ===");
        print(v);
        v.clear();           System.out.println("=== After clear() ===");
        print(v);
    }

    public static void print(Vector v) {
        System.out.println(v);
        System.out.println("size :"+v.size());
        System.out.println("capacity :"+v.capacity());
    }
}
```

Vector의 크기(size)와 용량(capacity)

// 1. 용량(capacity)이 5인 Vector를 생성한다.

```
Vector v = new Vector(5);
```

```
v.add("1");
```

```
v.add("2");
```

```
v.add("3");
```

// 2. 빈 공간을 없앤다.(용량과 크기가 같아진다.)

```
v.trimToSize();
```

// 3. capacity가 6이상 되도록 한다.

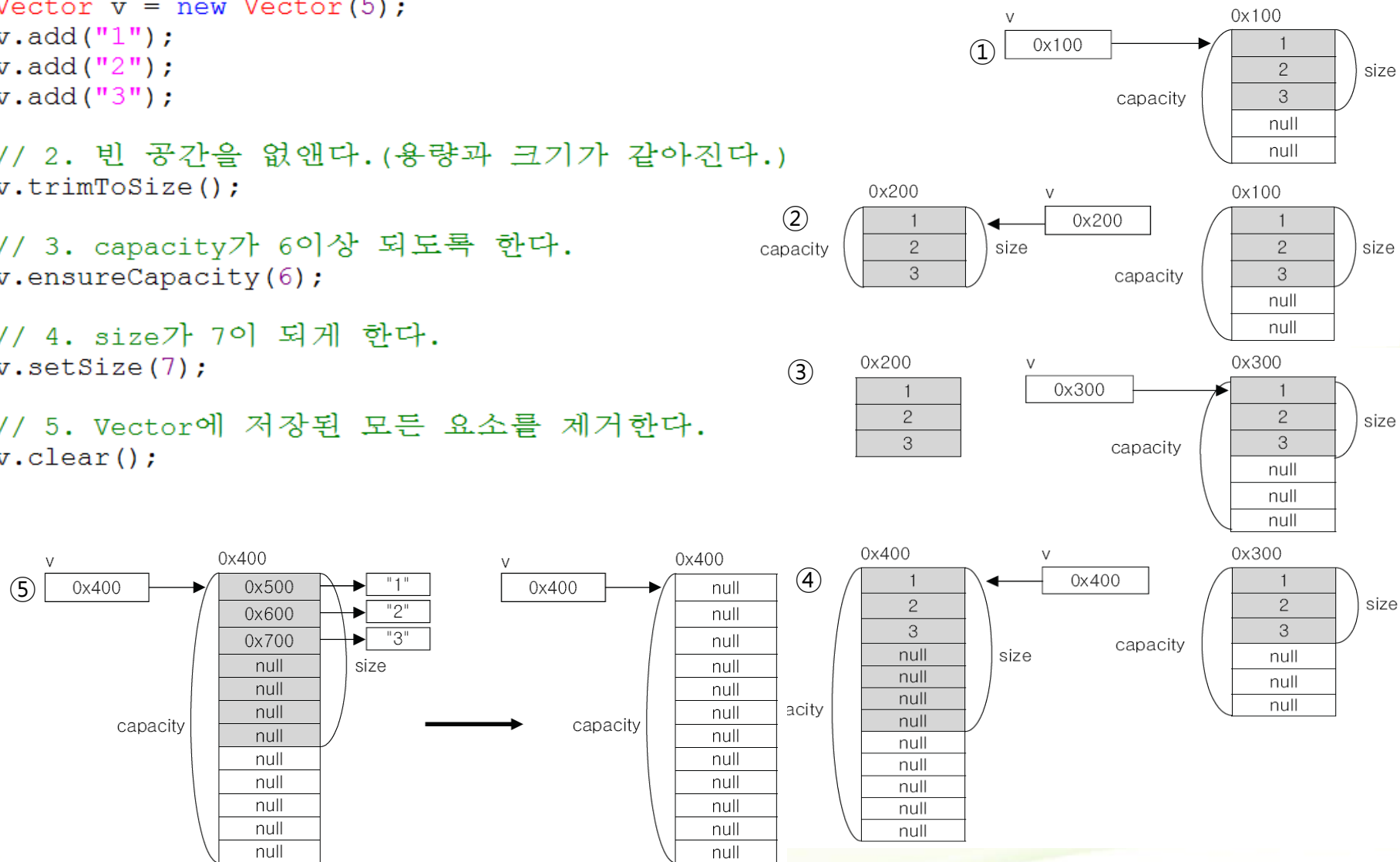
```
v.ensureCapacity(6);
```

// 4. size가 7이 되게 한다.

```
v.setSize(7);
```

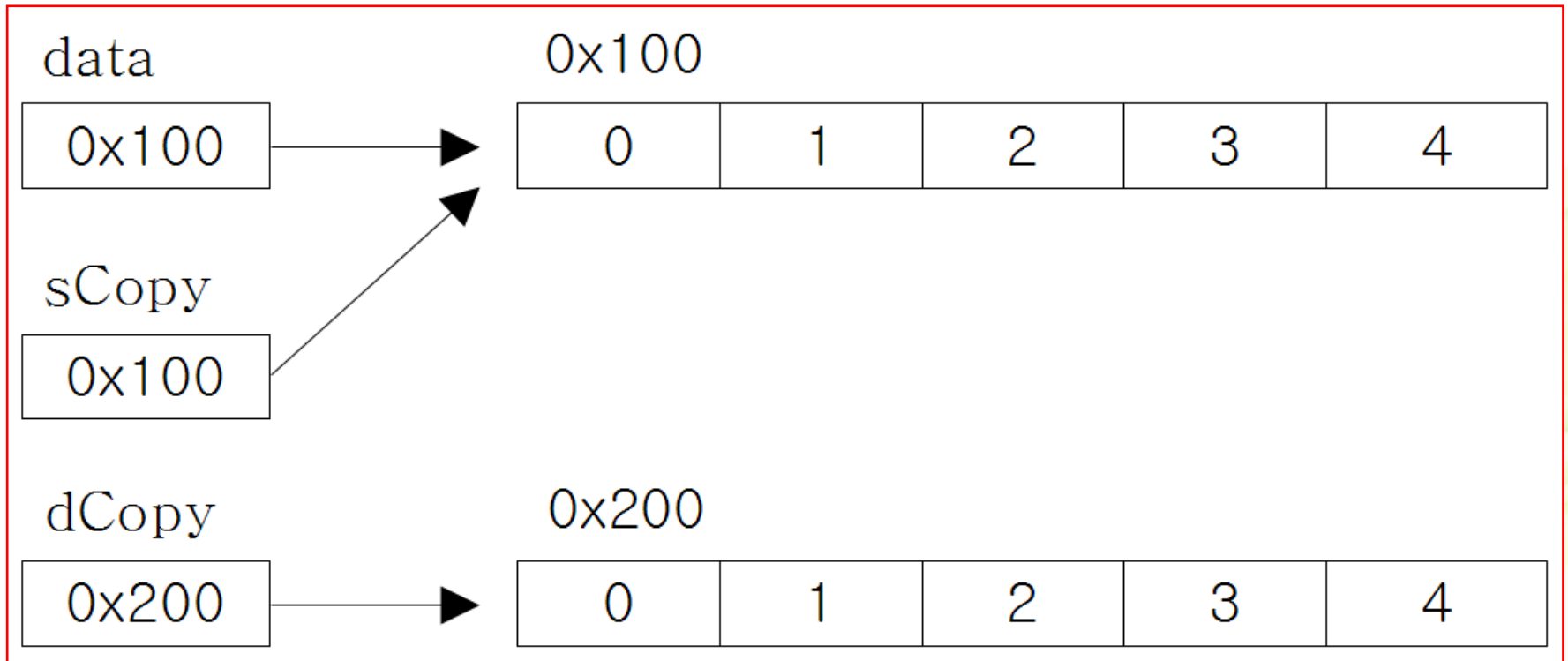
// 5. Vector에 저장된 모든 요소를 제거한다.

```
v.clear();
```



Deep copy vs. Shallow copy

- ▶ Deep copy(깊은 복사) : 같은 내용의 새로운 객체를 생성.
원본의 변화에 복사본이 영향을 받지 않는다.
- ▶ Shallow copy(얕은 복사) : 참조변수만 복사. 원본의 변화에 복사본이 영향을 받는다.



```
import java.util.Arrays;
class CopyTest {
    public static void main(String[] args) {
        int[] data = {0,1,2,3,4};      int[] sCopy = null;      int[] dCopy = null;
        sCopy = shallowCopy(data); dCopy = deepCopy(data);
        System.out.println("Original:" + Arrays.toString(data));
        System.out.println("Shallow :" + Arrays.toString(sCopy));
        System.out.println("Deep   :" + Arrays.toString(dCopy));
        System.out.println();
        data[0] = 5;
        System.out.println("Original:" + Arrays.toString(data));
        System.out.println("Shallow :" + Arrays.toString(sCopy));
        System.out.println("Deep   :" + Arrays.toString(dCopy));
    }
    public static int[] shallowCopy(int[] arr) {
        return arr;
    }
    public static int[] deepCopy(int[] arr) {
        if(arr==null) return null;
        int[] result = new int[arr.length];
        System.arraycopy(arr, 0, result, 0, arr.length);
        return result;
    }
}
```


ArrayList의 단점 - 배열의 단점

- 배열은 구조가 간단하고 데이터를 읽어오는 데 걸리는 시간(접근시간, access time)이 가장 빠르다는 장점이 있지만 단점도 있다.

▶ 단점 1 : 크기를 변경할 수 없다.

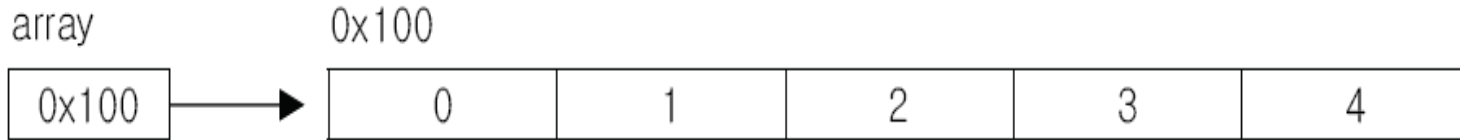
- 크기를 변경해야 하는 경우 새로운 배열을 생성하고 데이터를 복사해야 한다.(비용이 큰 작업)
- 크기 변경을 피하기 위해 충분히 큰 배열을 생성하면, 메모리 낭비가 심해진다.

▶ 단점 2 : 비순차적인 데이터의 추가, 삭제에 시간이 많이 걸린다.

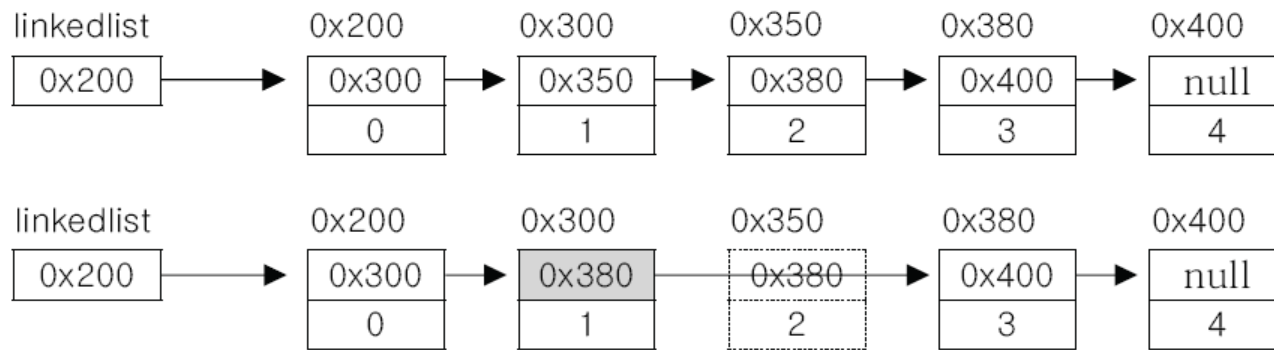
- 데이터를 추가하거나 삭제하기 위해, 많은 데이터를 옮겨야 한다.
- 그러나, 순차적인 데이터 추가(마지막에 추가)와 순차적으로 데이터를 삭제하는 것(마지막에서부터 삭제)은 빠르다.

LinkedList - 배열의 단점을 보완

- 배열과 달리 링크드 리스트는 불연속적으로 존재하는 데이터를 연결(link)

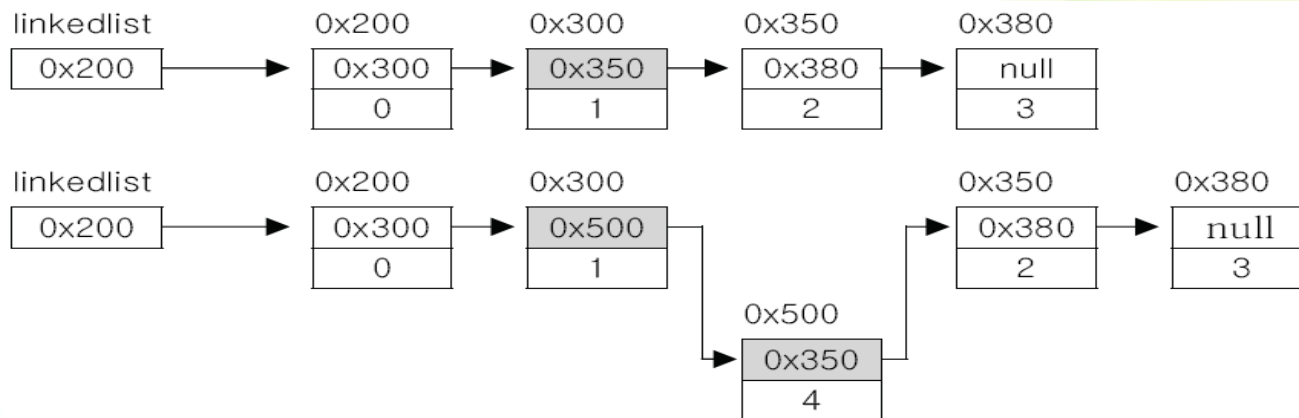


- ▶ 데이터의 삭제 : 단 한 번의 참조변경만으로 가능



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

- ▶ 데이터의 추가 : 하나의 Node 객체생성과 한 번의 참조변경만으로 가능



LinkedList — 주요메서드

생성자 또는 메서드	설 명
LinkedList()	LinkedList객체를 생성한다.
LinkedList(Collection c)	주어진 컬렉션을 포함하는 LinkedList객체를 생성한다.
void add(int index, Object element)	지정된 위치(index)에 객체(element)를 추가한다.
boolean add(Object o)	지정된 객체(o)를 LinkedList의 끝에 추가한다.
boolean addAll(Collection c)	주어진 컬렉션에 포함된 모든 요소를 LinkedList의 끝에 추가한다.
boolean addAll(int index, Collection c)	지정된 위치(index)에 주어진 컬렉션에 포함된 모든 요소를 추가한다.
void addFirst(Object o)	객체(o)를 LinkedList의 첫 번째 요소 앞에 추가한다.
void addLast(Object o)	객체(o)를 LinkedList의 마지막 요소 뒤에 추가한다.
void clear()	LinkedList의 모든 요소를 삭제한다.
Object clone()	LinkedList를 복제해서 반환한다.
boolean contains(Object o)	지정된 객체가 LinkedList에 포함되어 있는지 알려준다.
Object get(int index)	지정된 위치(index)의 객체를 반환한다.
Object getFirst()	LinkedList의 첫 번째 요소를 반환한다.
Object getLast()	LinkedList의 마지막 요소를 반환한다.
int indexOf(Object o)	지정된 객체가 저장된 위치(앞에서 몇 번째)를 반환한다.
int lastIndexOf(Object o)	지정된 객체가 저장된 위치(뒤에서 몇 번째)를 반환한다.

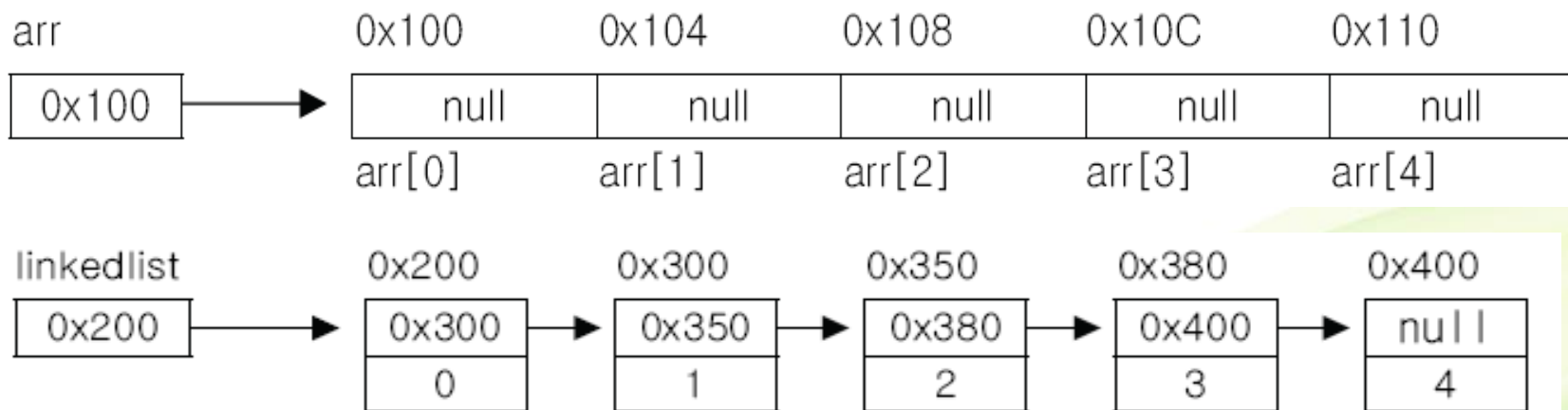
LinkedList — 주요메서드

생성자 또는 메서드	설 명
ListIterator listIterator(int index)	지정된 위치에서부터 시작하는 ListIterator를 반환한다.
Object remove(int index)	지정된 위치(index)의 객체를 LinkedList에서 제거한다.
boolean remove(Object o)	지정된 객체를 LinkedList에서 제거한다.
Object removeFirst()	LinkedList의 첫 번째 요소를 제거한다.
Object removeLast()	LinkedList의 마지막 요소를 제거한다.
Object set(int index, Object element)	지정된 위치(index)의 객체를 주어진 객체로 바꾼다.
int size()	LinkedList에 저장된 객체의 수를 반환한다.
Object[] toArray()	LinkedList에 저장된 객체를 배열로 반환한다.
Object[] toArray(Object[] a)	LinkedList에 저장된 객체를 주어진 배열에 저장하여 반환한다.
E element()	LinkedList의 첫 번째 요소를 반환한다.
boolean offer(E o)	지정된 객체(o)를 LinkedList의 끝에 추가한다.
E peek()	LinkedList의 첫 번째 요소를 반환한다.
E poll()	LinkedList의 첫 번째 요소를 반환한다.(LinkedList에서는 제거된다.)
E remove()	LinkedList의 첫 번째 요소를 제거한다.

ArrayList vs. LinkedList

- 순차적으로 데이터를 추가/삭제하는 경우, ArrayList가 빠르다.
- 비순차적으로 데이터를 추가/삭제하는 경우, LinkedList가 빠르다.
- 접근시간(access time)은 ArrayList가 빠르다.

n번째 데이터의 주소 = 배열의 주소 + n * 데이터 타입의 크기



컬렉션	읽기(접근시간)	추가 / 삭제	비 고
ArrayList	빠르다	느리다	순차적인 추가삭제는 빠름. 비효율적인 메모리사용
LinkedList	느리다	빠르다	데이터가 많을수록 접근성이 떨어짐

```
import java.util.*;
public class ArrayListLinkedListTest {
    public static void main(String args[]) {
        // 추가할 데이터의 개수를 고려하여 충분히 잡아야한다.
        ArrayList al = new ArrayList(1000000);
        LinkedList ll = new LinkedList();
        System.out.println("= 순차적으로 추가하기 =");
        System.out.println("ArrayList :"+add1(al));
        System.out.println("LinkedList :"+add1(ll));
        System.out.println();
        System.out.println("= 중간에 추가하기 =");
        System.out.println("ArrayList :"+add2(al));
        System.out.println("LinkedList :"+add2(ll));
        System.out.println();
        System.out.println("= 중간에서 삭제하기 =");
        System.out.println("ArrayList :"+remove2(al));
        System.out.println("LinkedList :"+remove2(ll));
        System.out.println();
        System.out.println("= 순차적으로 삭제하기 =");
        System.out.println("ArrayList :"+remove1(al));
        System.out.println("LinkedList :"+remove1(ll));
    }
}
```

```
public static long add1(List list) {
    long start = System.currentTimeMillis();
    for(int i=0; i<100000;i++) list.add(i+"");
    long end = System.currentTimeMillis();
    return end - start;
}
public static long add2(List list) {
    long start = System.currentTimeMillis();
    for(int i=0; i<1000;i++) list.add(500, "X");
    long end = System.currentTimeMillis();
    return end - start;
}
public static long remove1(List list) {
    long start = System.currentTimeMillis();
    for(int i=list.size()-1; i > 0;i--) list.remove(i);
    long end = System.currentTimeMillis();
    return end - start;
}
public static long remove2(List list) {
    long start = System.currentTimeMillis();
    for(int i=0; i<1000;i++) list.remove(i);
    long end = System.currentTimeMillis();
    return end - start;
}
}
```


- Stack은 객체를 후입선출(後入先出), last-in-first-out(LIFO)이며 객체의 저 정시의 push()메서드와 검출 시 사용하는 pop()과 Stack의 가장 위쪽 객체를 의미하는 peek() 메서드 그리고 Stack이 비어 있는지 판별 해 주는 empty()와 객체를 검색 해 주는 search()메서드들로 Vector라는 클래스를 확장하였다.

[표 7-8] Stack 생성자에 대한 요약

생성자명	설명
Stack()	새로운 Stack 객체를 생성하고 초기화한다.

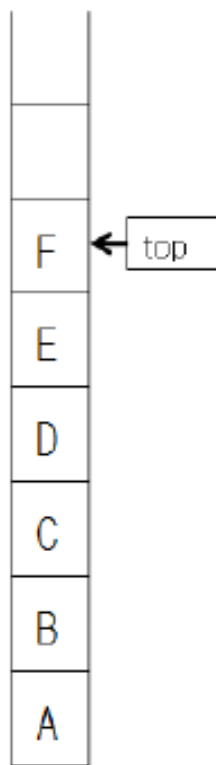
[표 7-9] Stack의 주요 메서드

반환형	메서드명	설명
boolean	empty()	Stack이 비었는지 비교하여 비어 있으면 true를 반환한다.
E	peek()	Stack의 가장 위쪽에 있는 객체를 반환한다.
	pop()	Stack의 가장 위쪽에 있는 객체를 삭제하고 그 객체를 반환한다.
	push(E item)	Stack의 가장 위쪽에서 객체를 추가한다.
int	search(Object o)	현재 Stack의 구조에서 인자로 전달받은 객체의 인덱스값을 반환한다(참고로 인덱스값은 1부터 시작한다).

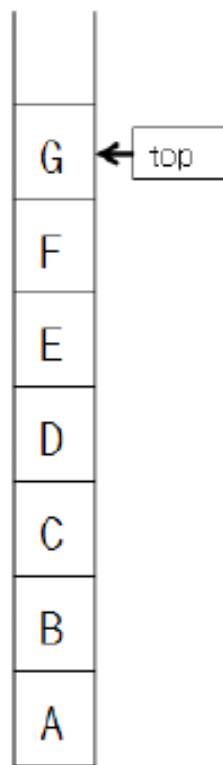
**:: Stack은 List 구조이지만
가방과 같은 구조라고
생각하면 된다.
입구가 하나라서 제일 먼저
넣은 물건(객체)이 가장
아래에
위치하므로 꺼낼 때는
가장 나중에 나오게 된다.**

Stack 클래스

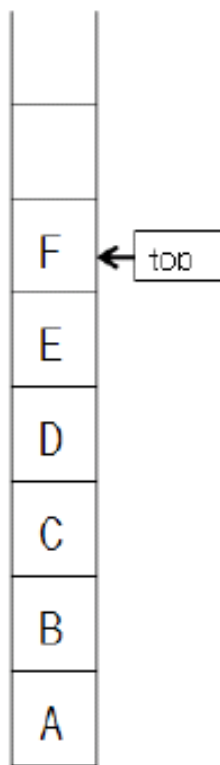
초기상태



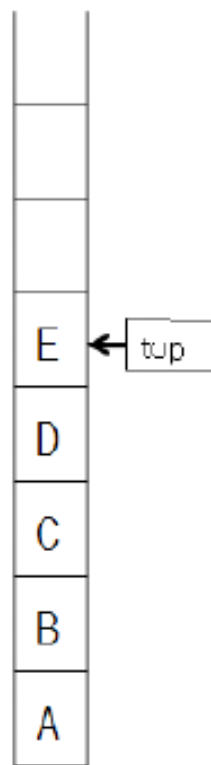
①Push(G)



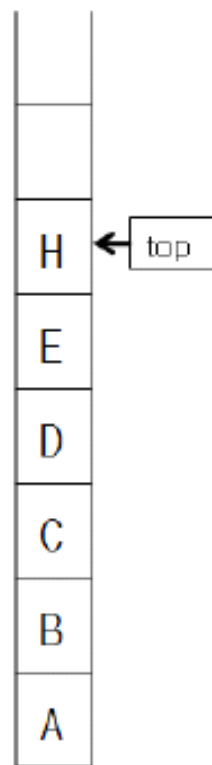
②Pop(data)



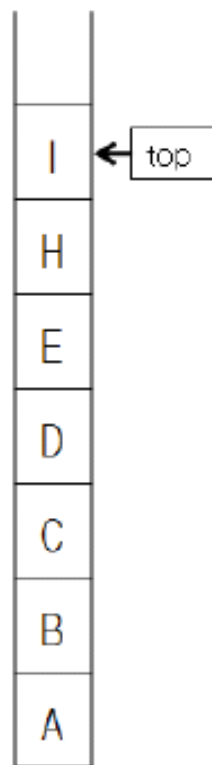
③Pop(data)



④Push(H)



⑤Push(I)



Stack 클래스의 주요 메소드

메소드	설 명
pop()	스택의 맨 위에서 객체를 제거하고 그 객체를 반환한다.
push(E item)	스택의 맨 위에 객체를 추가한다.
peek()	스택의 맨 위에 있는 객체를 반환한다. 이때 객체를 스택에서 제거하지는 않는다.
Boolean empty()	현재 스택이 비어 있는지를 확인한다. isEmpty도 동일한 기능을 한다.

```
import java.util.Stack;

public class StackEx1{
    public static void main(String[] args) {
        String[] groupA =
            {"우즈베키스탄","쿠웨이트","사우디","대한민국"};

        Stack<String> stack = new Stack<String>();
        for(String n : groupA)
            stack.push(n);

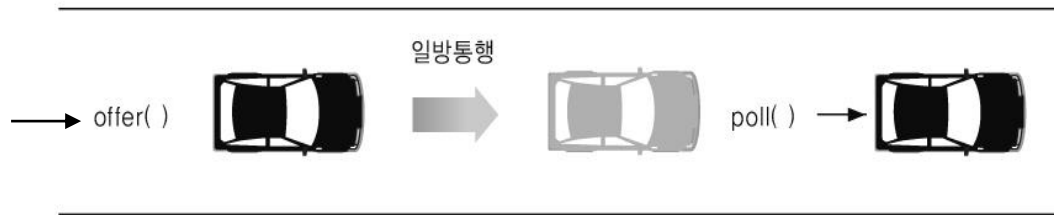
        while(!stack.isEmpty())
            System.out.println(stack.pop());
    }
}
```

실행결과

```
----- Java Run -----
대한민국
사우디
쿠웨이트
우즈베키스탄
Normal Termination
출력 완료 (0초 경과).
```

Queue 인터페이스

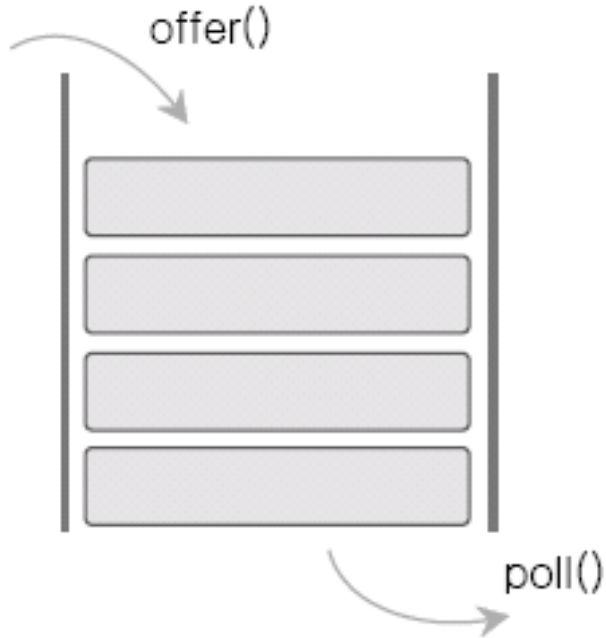
- ⇒ JDK5.0에 오면서 새롭게 추가된 인터페이스이며 Queue의 구조는 도로의 일정구간인 일방통행(一方通行)과 같다. 요소(Element)가 들어가는 입구와 요소(Element)가 나오는 출구가 따로 준비 되어 있어 가장 먼저 들어간 요소(Element)가 가장 먼저 나오는 선입선출(先入先出), first-in-first-out(FIFO)방식이다.



[그림 7-23] Queue의 도식화

- API문서를 참조해 보면 Queue를 구현하는 클래스는 여러 가지가 있는 것을 알 수 있으며 여기서는 그 중에서 LinkedList에 대해서 알아보도록 하겠다.

Queue 인터페이스 : LinkedList 클래스



Queue 인터페이스 LinkedList 클래스

메소드	설 명
boolean offer(E o)	큐에 객체를 넣는다.
E poll()	큐에서 데이터를 꺼내 온다. 만일 큐가 비어 있다면 null을 반환한다.
E peek()	큐의 맨 위에 있는 객체를 반환한다. 이때 객체를 큐에서 제거하지는 않는다. 그리고 큐가 비어 있다면 null을 반환한다.

QueueEx1.java

```
import java.util.Vector;
public class QueueEx1 {

    public static void main(String[] args) {
        String[] item = {"소나타", "렉스톤", "제규어"};
        LinkedList<String> q = new LinkedList<String>();

        for(String n : item)
            q.offer(n); //요소 추가
        System.out.println("q의 크기:"+q.size());

        String data="";
        while((data = q.poll()) != null)
            out.println(data+"삭제!");

        System.out.println("q의 크기:"+q.size());
    }
}
```

실행결과

```
----- Java Run -----
q의 크기:3
소나타삭제!
렉스톤삭제!
제규어삭제!
q의 크기:0
Normal Termination
출력 완료 (0초 경과).
```

Enumeration & Iterator

❖ 컬렉션에 저장되어 있는 객체들을 저장방법에 상관없이 접근 할 수 있도록 한 인터페이스로 컬렉션 객체들의 `enumerable()`이나 `iterable()` 메서드에 의해 구현됩니다.

❖ Enumeration 인터페이스

❖ `boolean hasMoreElements()`: 다음 요소가 있는지 없는지 여부를 판단해주는 메서드

❖ `E nextElement ()`: 열거에 1 개 이상의 요소가 남아 있는 경우는 다음의 요소를 리턴

❖ Iterator 인터페이스

❖ `boolean hasNext()`: 다음 요소가 있는 지 여부를 리턴

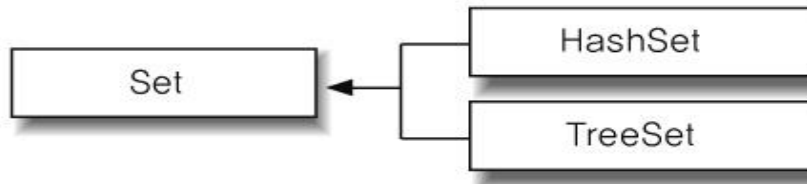
❖ `Enext()`: 반복 처리로 다음의 요소를 리턴

❖ `void remove()`: 마지막으로 리턴된 요소를 삭제

```
import java.util.*;
class LinkedListTest {
    public static void main(String[] args) {
        System.out.println("리스트에 배열");
        LinkedList<String> list = new LinkedList<String>();
        //데이터를 추가
        list.add("포도");    list.add("딸기");    list.add("복숭아");
        list.add(2,"키위");    list.set(0,"오렌지");
        list.remove(1);//삭제할 인덱스번호
        list.remove("키위");//삭제할 데이터값
        //검색
        for(int i=0;i<list.size();i++){
            String str=list.get(i);
            System.out.println("str="+str);
        }
        //추가->Iterator
        Iterator<String> i = list.iterator();
        while(i.hasNext()){
            String str=i.next();
            System.out.println("str="+str);
        }//while
    }
}
```

Set 인터페이스

- ⇒ Set내에 저장되는 객체들은 특별한 기준에 맞춰서 정렬되지 않는다. 그리고 저장되는 객체들간의 중복된 요소가 발생하지 못하도록 내부적으로 관리되고 있다. 다음은 Set인터페이스를 구현하고 있는 클래스들이다. 더 자세한 것은 API문서를 참조하길 바라면 다음 그림은 기본적으로 알려진 구현 클래스들이므로 참조하기 바란다.



[그림 7-10] Set의 구조

[표 7-4] Set의 구현 클래스

구현 클래스	설명
HashSet	Set 인터페이스를 구현하고 있으며 내부적으로 HashMap을 사용하고 있다. 얻어지는 Iterator의 정렬 상태를 보장하지 못하므로 특정한 기준으로 정렬을 이루고 있지 않으며 저장 및 검출과 같은 동작에는 일정한 시간을 필요로 한다.
TreeSet	내부적으로 Set 인터페이스를 구현하고 있으며 TreeMap에 의해 후원을 받는다. 그리고 기본적으로 얻어지는 Iterator의 요소들은 오름차순 정렬 상태를 유지하고 있다.

○HashSet

- 기본적인 Set인터페이스를 구현하고 있으며 정렬순서나 반복처리 시 처리순서에 대한 기준은 없다. 그리고 반복 처리에 대해서는 저장된 요소(Element)의 수와는 별도로 용량에 비례하는 시간이 필요하므로 반복 처리하는 성능이 중요한 응용프로그램에서는 초기용량을 너무 높게 설정하지 않는 것이 중요하다.

[표 7-5] HashSet 생성자 요약

생성자명	설명
HashSet()	새로운 HashSet 객체를 생성하고 초기화한다.

[표 7-6] HashSet 메서드 요약

반환형	메서드명	설명
boolean	add(E o)	제네릭 타입으로 넘어온 객체가 Set 구조에 없다면 추가하고 true를 반환한다.
void	clear()	Set 구조에 있는 모든 요소들을 삭제한다.
boolean	contains(Object o)	인자로 전달된 객체를 현 Collection에서 요소로 가지고 있으면 true를 반환한다.
	isEmpty()	현 Collection에 저장된 요소가 없다면 true를 반환한다.
Iterator<E>	iterator()	현 Set 구조의 요소들을 순서대로 처리하기 위해 Iterator 객체로 반환한다.
boolean	remove(Object o)	현 Set 구조에서 인자로 전달된 객체를 삭제한다. 이때 삭제에 성공하면 true를 반환한다.
int	size()	현 Set 구조에 저장된 요소의 수를 반환한다.

```
import java.util.HashSet;
import java.util.Iterator;
public class Ex01 {
    public static void main(String[] args) {
        HashSet<String> set=new HashSet<String>( );
        set.add("사과");
        set.add("바나나");
        set.add("귤");
        set.add("오렌지");
        set.add("바나나");
        System.out.println("요소의 개수->" + set.size());

        Iterator elements=set.iterator();
        while(elements.hasNext())           //요소가 있다면
            System.out.println(elements.next());
            //요소를 얻어내어 출력
    }
}
```

[예제 7-10] HashSetEx1.java

```
import java.util.*;
public class HashSetEx1 {
    public static void main(String args[]) {
        String[] str = {"Java", "Beans", "Java", "XML"};

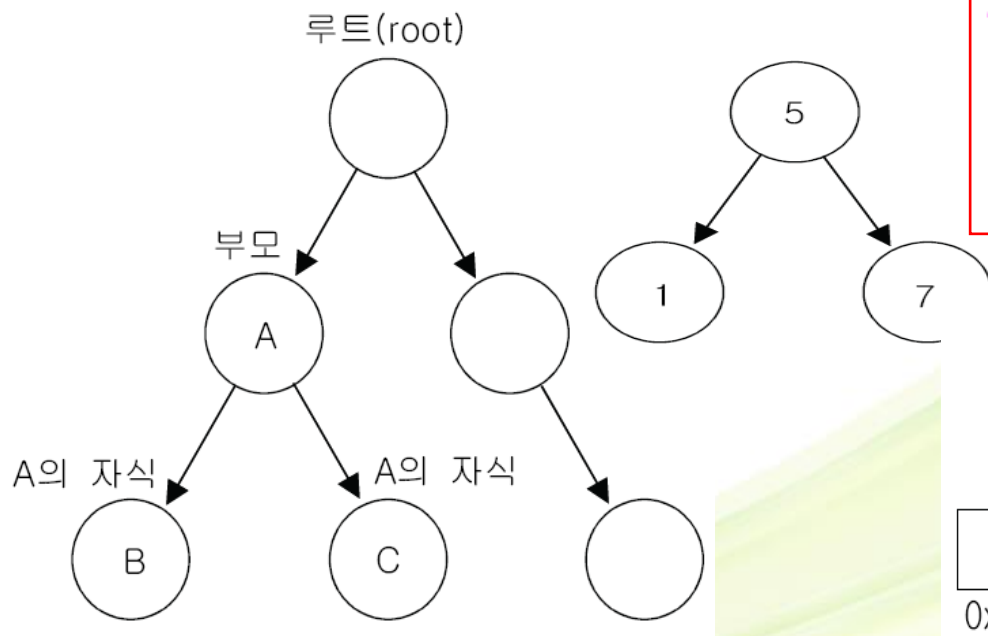
        HashSet<String> hs1 = new HashSet<String>();
        HashSet<String> hs2 = new HashSet<String>();
        for (String n : str){
            if (!hs1.add(n))
                hs2.add(n);
        }
        System.out.println("hs1 : " + hs1);
        hs1.removeAll(hs2);
        System.out.println("hs1 : " + hs1);
        System.out.println("hs2 : " + hs2);
    }
}
```

실행결과

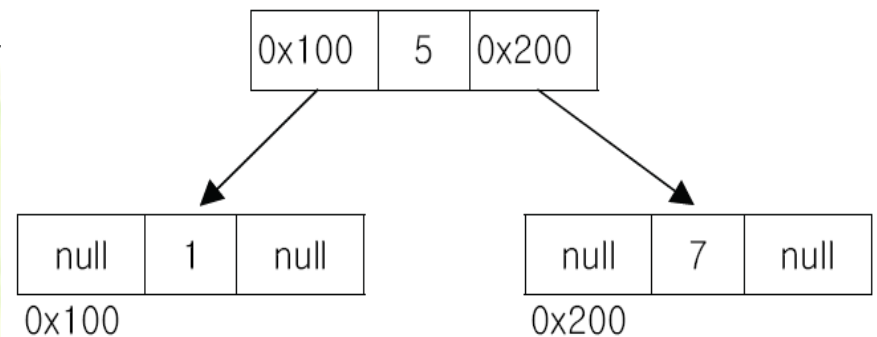
```
----- Java Run -----
hs1 : [Java, Beans, XML]
hs1 : [Beans, XML]
hs2 : [Java]
Normal Termination
출력 완료 (0초 경과).
```

TreeSet — 검색과 정렬에 유리

- Set인터페이스를 구현한 컬렉션 클래스.(중복허용x, 순서유지x, 정렬저장○)
- 이진검색트리(binary search tree - 정렬, 검색에 유리)의 구조로 되어있다.
- 링크드리스트와 같이 각 요소(node)가 나무(tree)형태로 연결된 구조
- 모든 트리는 하나의 루트(root node)를 가지며, 서로 연결된 두 요소를 ‘부모-자식관계’에 있다 하고, 하나의 부모에 최대 두 개의 자식을 갖는다.
- 왼쪽 자식의 값은 부모의 값보다 작은 값을, 오른쪽 자식의 값은 부모보다 큰 값을 저장한다.
- 검색과 정렬에 유리하지만, HashSet보다 데이터 추가, 삭제시간이 더 걸린다.



```
class TreeNode {
    TreeNode left;    // 왼쪽 자식노드
    Object element;   // 저장할 객체
    TreeNode right;   // 오른쪽 자식노드
}
```

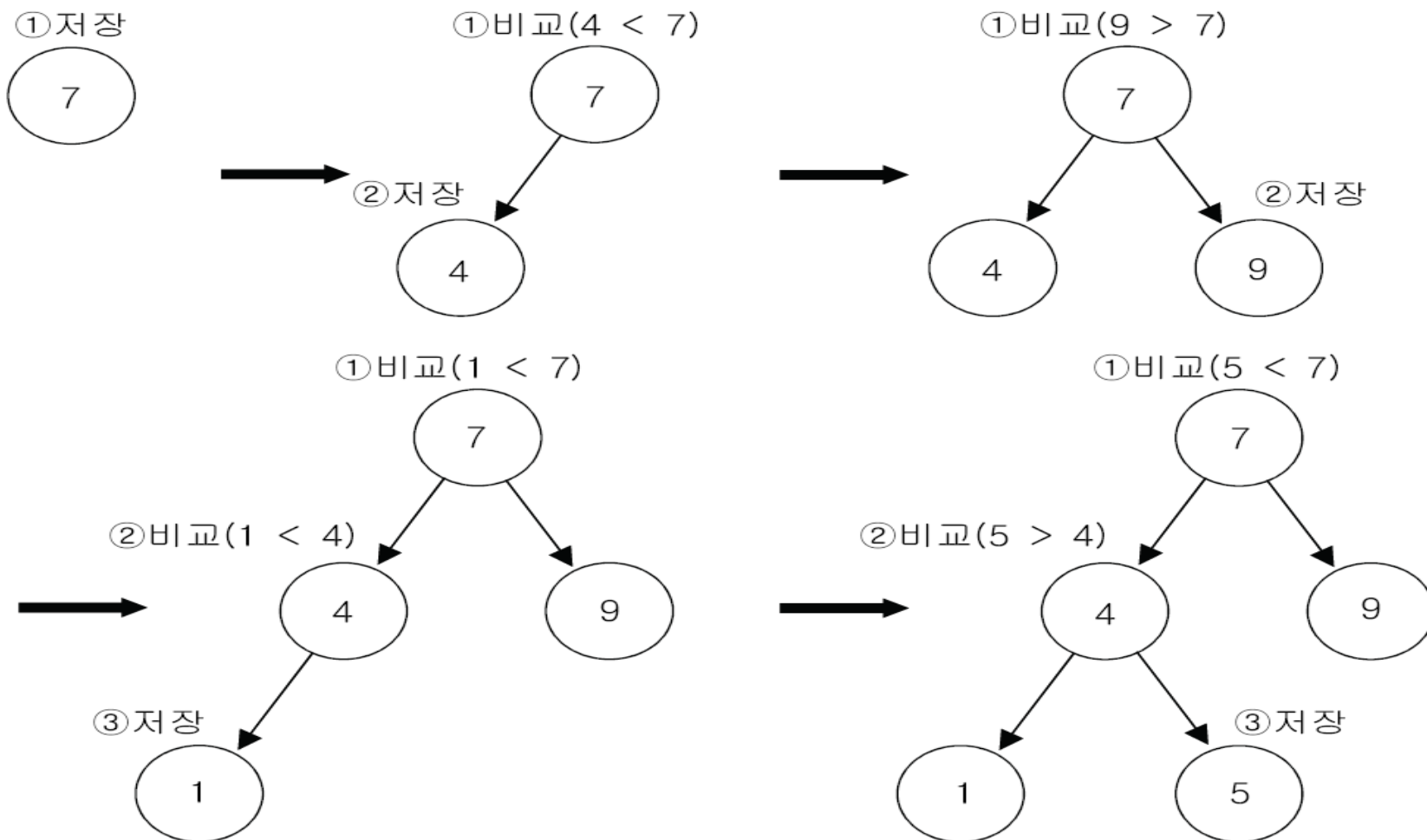


TreeSet – 주요메서드

생성자 또는 메서드	설 명
TreeSet()	기본생성자
TreeSet(Collection c)	주어진 컬렉션을 저장하는 TreeSet을 생성
TreeSet(Comparator c)	주어진 정렬조건으로 정렬하는 TreeSet을 생성
TreeSet(SortedSet s)	주어진 SortedSet을 구현한 컬렉션을 저장하는 TreeSet을 생성
boolean add(Object o) boolean addAll(Collection c)	지정된 객체(o) 또는 Collection(c)의 객체들을 Collection에 추가한다.
void clear()	저장된 모든 객체를 삭제한다.
Object clone()	TreeSet을 복제하여 반환한다.
Comparator comparator()	TreeSet의 정렬기준(Comparator)를 반환한다.
boolean contains(Object o) boolean containsAll(Collection c)	지정된 객체(o) 또는 Collection의 객체들이 포함되어 있는지 확인한다.
Object first()	정렬된 순서에서 첫 번째 객체를 반환한다.
SortedSet headSet(Object toElement)	지정된 객체보다 작은 값의 객체들을 반환한다.
boolean isEmpty()	TreeSet이 비어있는지 확인한다.
Iterator iterator()	TreeSet의 Iterator를 반환한다.
Object last()	정렬된 순서에서 마지막 객체를 반환한다.
boolean remove(Object o)	지정된 객체를 삭제한다.
boolean retainAll(Collection c)	주어진 컬렉션과 공통된 요소만을 남기고 삭제한다.(교집합)
int size()	저장된 객체의 개수를 반환한다.
SortedSet subSet(Object fromElement, Object toElement)	범위 검색(fromElement와 toElement사이)의 결과를 반환한다.(끝 범위인 toElement는 범위에 포함되지 않음)
SortedSet tailSet(Object fromElement)	지정된 객체보다 큰 값의 객체들을 반환한다.
Object[] toArray()	저장된 객체를 객체배열로 반환한다.
Object[] toArray(Object[] a)	저장된 객체를 주어진 객체배열에 저장하여 반환한다.

TreeSet – 데이터 저장과정

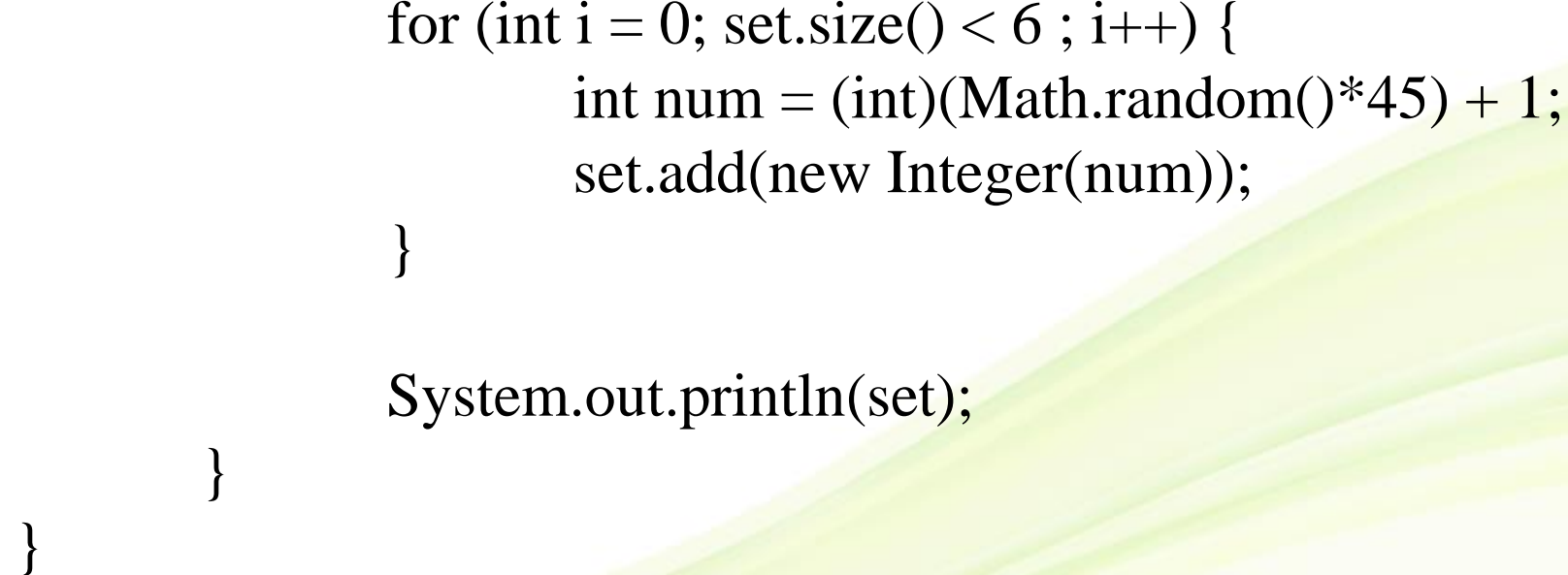
※만일 TreeSet에 7,4,9,1,5의 순서로 데이터를 저장한다면, 다음과 같은 과정을 거치게 된다.





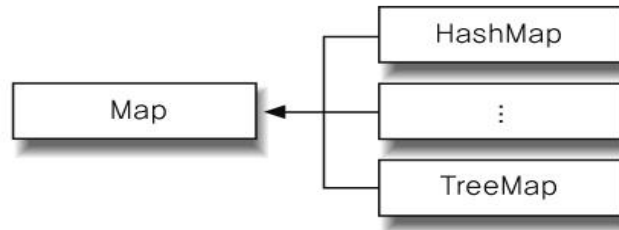
```
import java.util.*;
```

```
class TreeSetLotto {  
    public static void main(String[] args) {  
        Set set = new TreeSet();  
  
        for (int i = 0; set.size() < 6 ; i++) {  
            int num = (int)(Math.random()*45) + 1;  
            set.add(new Integer(num));  
        }  
  
        System.out.println(set);  
    }  
}
```



○Map인터페이스

- **Key**와 **Value**를 매핑하는 객체이다. 여기에 사용되는 **Key**는 절대 중복될 수 없으며 각 **Key**는 1개의 **Value**만 매핑할 수 있다. 정렬의 기준이 없으며 이는 마치 각 **Value**에 열쇠 고리를 달아서 큰 주머니에 넣어두고 오로지 **Key**로 각 **Value**를 참조 할 수 있도록 해둔 구조라 할 수 있다.



[그림 7-25] Map의 구조

- 사용자가 원하는 **Value**의 **Key**를 알고 있다면 **Key**를 당겨(**get**) 해당 **Key**와 매핑되어 있는 **Value**를 얻을 수 있는 구조이다. 즉, 검색을 **Key**로 해야 하므로 **Key**를 모르면 원하는 **Value**를 얻어내는 못하게 된다.

[표 7-15] Map의 구현 클래스

구현 클래스	설명
Hashtable	정렬의 기능을 가지지 않는 Map 인터페이스를 구현하고 있다. 이는 Key가 null을 가질 수 없으며, Value 또한 null을 허용하지 않는다. 중복 또한 불가능하며 스레드 동기화를 지원하는 특징을 가지고 있다.
HashMap	앞의 Hashtable과 거의 동일한 객체로 다른 점이 있다면 Key와 Value에 있어 null을 허용한다는 점과 스레드 동기화를 지원하지 않는다는 것이다.

○HashMap

- **Key**와 **Value**를 하나의 쌍으로 저장되는 구조이며 저장되는 **Value**와 **Key**가 **null**을 허용한다. 하지만 중복은 허용하지 않으므로 **null**을 가지는 **Key**가 2개일 수는 없다. 그리고 동기화가 포함되지 않았으므로 나중에 배우는 **Multi-Thread**환경에서의 구현이 아니라면 **Hashtable**에 비해서 처리 속도가 빠른 장점을 가지고 있다.

[표 7-16] HashMap의 주요 생성자

생성자명	설명
HashMap()	초기용량을 16으로 하고 적재율은 0.75로 하여 새로운 HashMap 객체가 생성된다.
HashMap (int initialCapacity)	전달된 인자를 통해 객체를 저장할 수 있는 초기용량으로 설정되고 기본 적재율인 0.75로 HashMap 객체가 생성된다.
HashMap(int initialCapacity, float loadFactor)	전달된 인자인 용량과 적재율로 새로운 HashMap 객체가 생성된다.

Map 인터페이스 Hashtable 클래스

키(이름)	값(전화번호)
"홍길동"	"010-111-1111"
"성춘향"	"010-222-2222"
"한석봉"	"010-333-3333"

키에 대한 데이터 타입 키에 대한 데이터 타입
↓ ↓
`Hashtable<String, String> ht=new Hashtable<String, String>;`
↑ ↑
값에 대한 데이터 타입 값에 대한 데이터 타입

키(이름)	값(전화번호)
"홍길동"	"010-111-1111"
"성춘향"	"010-222-2222"
"한석봉"	"010-333-3333"

키에 대한 데이터 타입 키에 대한 데이터 타입

↓ ↓

```
Hashtable<String, String> ht=new Hashtable<String, String>;
```

↑ ↑

값에 대한 데이터 타입 값에 대한 데이터 타입

```
import java.util.*;
public class HashtableEX {
    public static void main(String[] args) {
        Hashtable<String, String> ht =
            new Hashtable<String, String>();
        ht.put("홍길동", "010-1111-1111");
        ht.put("전우치", "010-2222-22222");
        ht.put("박씨부인", "010-3333-3333");

        System.out.println(ht.get("홍길동"));
        System.out.println(ht.get("박씨부인"));
    }
}
```

HashMap의 주요 메서드

반환형	메서드명	설명
void	clear()	모든 매핑을 맵으로부터 삭제한다.
V	get(Object key)	인자로 전달된 key 객체와 매핑되고 있는 Value를 반환한다.
boolean	isEmpty()	현재 맵이 비어있다면 true를 반환한다.
Set <K>	keySet()	맵에 저장되고 있는 Key들을 Set 인터페이스로 반환한다.
V	put(K key, V value)	인자로 전달된 Key와 Value를 현재 맵에 저장한다.
	remove(Object key)	인자로 전달된 Key에 대한 것이 있다면 현재 맵에서 삭제하고 매핑된 Value를 반환한다. 전달된 Key에 대한 정보가 없다면 null을 반환한다.
int	size()	맵에 저장된 Key와 Value로 매핑된 수를 반환한다.
Collection <V>	values()	현재 맵에 저장된 Value들만 Collection 인터페이스로 반환한다.

MapEx1.java

```
import java.util.*;
import static java.lang.System.out;
public class MapEx1{
    public static void main(String[] args) {
        String[] msg = {"Berlin","Dortmund","Frankfurt",
            "Gelsenkirchen","Hamburg"};

        HashMap<Integer,String> map =
            new HashMap<Integer,String>(); // HashMap 생성

        for(int i=0 ; i<msg.length ; i++)
            map.put(i,msg[i]); //맵에 저장

        Set<Integer> keys = map.keySet();
        for(Integer n : keys)
            out.println(map.get(n)); //맵에서 읽어오기
        }
    }
}
```

실행결과

```
----- Java Run -----
Frankfurt
Hamburg
Dortmund
Gelsenkirchen
Berlin
Normal Termination
출력 완료 (0초 경과).
```

Map 컬렉션

Properties 특징

키와 값을 String 타입으로 제한한 Map 컬렉션

Properties는 프로퍼티(~.properties) 파일을 읽어 들일 때 주로 사용

프로퍼티(~.properties) 파일

옵션 정보, 데이터베이스 연결 정보, 국제화(다국어) 정보를 기록

텍스트 파일로 활용

애플리케이션에서 주로 변경이 잦은 문자열을 저장

유지 보수를 편리하게 만들어 줌

키와 값이 = 기호로 연결되어 있는 텍스트 파일

ISO 8859-1 문자셋으로 저장

한글은 유니코드(Unicode)로 변환되어 저장

Map 컬렉션

1. Hashtable의 하위 클래스
2. Hashtable은 키와 값을 다양한 타입으로 저장 가능한데 비해 Properties는 키와 값을 String 타입으로 제한한 컬렉션
3. 애플리케이션의 옵션 정보, 데이터베이스 연결 정보 그리고 국제화(다국어) 정보가 저장된 프로퍼티(~.properties) 파일을 읽을 때 주로 사용
4. 프로퍼티 파일은 키와 값이 = 기호로 연결되어 있는 텍스트 파일로 ISO 8859-1 문자셋으로 저장. 한글은 유니코드로 변환되어 저장
5. 이클립스에서 유니코드로 변환된 내용을 다시 한글로 보려면 마우스를 유니코드 위에 올려놓으면 됨
6. 프로퍼티 파일을 읽기 위해서는 Properties 객체를 생성하고 load()메서드를 호출
7. load() 메서드는 프로퍼티 파일로부터 데이터를 읽기 위해 FileReader 객체를 매개값으로 받음

8. 사용 방법

```
Properties pro = new Properties();  
pro.load(new FileReader("프로퍼티 파일 경로"));
```

Map 컬렉션

1. 프로퍼티 파일은 일반적으로 클래스파일(~.class)과 함께 저장
2. 클래스 파일을 기준으로 상대 경로를 이용해서 프로퍼티 파일의 경로를 얻으려면 Class 의 getResource() 메서드를 사용
3. getResource()는 주어진 파일의 상대 경로를 URL로 리턴
URL 의 getPath()는 파일의 절대 경로를 리턴
4. **String path = 클래스.class.getResource("database.properties").getPath();**
// 경로에 한글이 있을 경우에 한글을 복원.
path = URLDecoder.decode(path, "UTF-8");
Properties pro = new Properties();
pro.load(new FileReader(path));
5. 다른 패키지에 프로퍼티 파일이 있는 경우는 경로 구분자로 "/" 를 사용
6. A클래스가 com.naver 패키지에 있고 database.properties 파일이
com.naver.www 패키지에 있을 경우
String path = A.class.getResource("www/database.properties").getPath();
7. Properties 객체에서 해당 키의 값을 읽으려면 getProperty() 메서드를 사용
8. Properties도 Map 컬렉션이므로 get() 메서드로 값을 얻을 수도 있지만,
get() 메서드는 값을 object 타입으로 리턴하므로 형변환해서 String을 얻어야
하기 때문에 일반적으로 getProperty()메서드를 사용
String val = properties.getProperty("key");

Map 컬렉션

database.properties

driver=oracle.jdbc.OracleDriver

url=jdbc:oracle:thin:@localhost:1521:orcl

username=scott

password=tiger

```
public class PropertiesExample {  
    public static void main(String[] args) throws Exception {  
        Properties properties = new Properties();  
        String path = PropertiesExample.class.getResource("database.properties").getPath();  
        path = URLDecoder.decode(path, "utf-8");  
        properties.load(new FileReader(path));  
  
        String driver = properties.getProperty("driver");  
        String url = properties.getProperty("url");  
        String username = properties.getProperty("username");  
        String password = properties.getProperty("password");  
  
        System.out.println("driver : " + driver);  
        System.out.println("url : " + url);  
        System.out.println("username : " + username);  
        System.out.println("password : " + password);  
    }  
}
```

Arrays 클래스

❖ Arrays

- 배열 조작 기능을 가지고 있는 클래스 - 배열 복사, 항목 정렬, 항목 검색
- 제공하는 정적 메소드

리턴타입	메소드 이름	설명
int	binarySearch(배열, 찾는값)	전체 배열 항목에서 찾는값이 있는 인덱스 리턴
타겟배열	copyOf(원본배열, 복사할길이)	원본배열의 0 번 인덱스에서 복사할 길이만큼 복사한 배열 리턴, 복사할 길이는 원본배열의 길이보다 크도 되며, 타겟배열의 길이가 된다.
타겟배열	copyOfRange(원본배열, 시작인덱스, 끝인덱스)	원본배열의 시작인덱스에서 끝인덱스까지 복사한 배열 리턴
boolean	deepEquals(배열, 배열)	두 배열의 깊은 비교(중첩 배열의 항목까지 비교)
boolean	equals(배열, 배열)	얕은 비교(중첩 배열의 항목은 비교하지 않음)
void	fill(배열, 값)	전체 배열 항목에 동일한 값을 저장
void	fill(배열, 시작인덱스, 끝인덱스, 값)	시작인덱스부터 끝인덱스까지의 항목에만 동일한 값을 저장
void	sort(배열)	배열의 전체 항목을 올림차순으로 정렬

Arrays 클래스

배열 복사

- `Arrays.copyOf(원본배열, 복사할 길이)`

⇒ 0 ~ (복사할 길이-1)까지 항목 복사

⇒ 복사할 길이는 원본 배열의 길이보다 커도 되며 타겟 배열의 길이

```
char[] arr1 = {'J', 'A', 'V', 'A'};  
char[] arr2 = Arrays.copyOf(arr1, arr1.length);
```

- `copyOfRange(원본 배열, 시작 인덱스, 끝 인덱스)`

⇒ 시작인덱스 ~ (끝 인덱스-1)까지 항목 복사

```
char[] arr1 = {'J', 'A', 'V', 'A'};  
char[] arr2 = Arrays.copyOfRange(arr1, 1, 3);
```

- `System.arraycopy()`

```
System.arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
```

원본배열 원본시작인덱스 타겟배열 타겟시작인덱스 복사개수

Arrays 클래스

```
import java.util.Arrays;
public class Array1 {
    public static void main(String[] args) {
        String[] arr1 = {"J","A","V","A"};
        String[] arr2 = Arrays.copyOf(arr1, arr1.length);
        String[] arr3 = Arrays.copyOfRange(arr1,1, 3);
        String[] arr4 = new String[arr1.length];

        System.arraycopy(arr1,0, arr4, 0, arr1.length);
        System.out.println(Arrays.toString(arr1));
        System.out.println(Arrays.toString(arr2));
        System.out.println(Arrays.toString(arr3));
        System.out.println(Arrays.toString(arr4));
    }
}
```


Arrays 클래스

■ 배열 항목 비교

- `Arrays.equals(배열, 배열)` - 1차 항목의 값만 비교
- `Arrays.deepEquals(배열, 배열)` - 중첩된 배열의 항목까지 비교

■ 배열 항목 정렬

- `Arrays.sort(배열)`- 항목 오름차 순으로 정렬
⇒ 기본 타입이거나 `String` 배열 자동 정렬
- `Arrays.sort(score, Collections.reverseOrder());` 항목 내림차 순으로 정렬
- 사용자 정의 클래스 배열은 `Comparable` 인터페이스를 구현해야만 정렬

■ 배열 항목 검색

- 특정 값 위치한 인덱스 얻는 것
- `Arrays.sort(배열)`로 먼저 정렬
- `Arrays.binarySearch(배열, 찾는 값)` 메소드로 항목을 찾아야

Arrays 클래스

```
import java.util.Collections;

public class Array2 {
    public static void main(String[] args) {
        Integer[] score = {43,98,56,78,34};
        System.out.println(Arrays.toString(score));
        Arrays.sort(score);
        System.out.println(Arrays.toString(score));
        Arrays.sort(score, Collections.reverseOrder());
        System.out.println(Arrays.toString(score));
        String[] name = {"수지","설현","하니","재석"};
        System.out.println(Arrays.toString(name));
        Arrays.sort(name);
        System.out.println(Arrays.toString(name));
        Arrays.sort(name, Collections.reverseOrder());
        System.out.println(Arrays.toString(name));
    }
}
```

Arrays 클래스

```
public class Member implements Comparable<Member> {  
    String name;  
    Member(String name) {  
        this.name = name;  
    }  
    public int compareTo(Member o) {  
        return name.compareTo(o.name);  
    }  
}
```

Arrays 클래스

```
public class SearchExample {  
    public static void main(String[] args) {  
        int[] scores = { 99, 97, 98 };  
        Arrays.sort(scores);  
        System.out.println(Arrays.toString(scores));  
        int index = Arrays.binarySearch(scores, 99);  
        System.out.println("찾은 인덱스: " + index);  
        String[] names = { "홍길동", "박동수", "김민수" };  
        Arrays.sort(names);  
        System.out.println(Arrays.toString(names));  
        index = Arrays.binarySearch(names, "홍길동"); //문자열 검색  
        System.out.println("찾은 인덱스: " + index);  
        //객체 검색  
        Member1 m1 = new Member1("홍길동");  
        Member1 m2 = new Member1("박동수");  
        Member1 m3 = new Member1("김민수");  
        Member1[] members = { m1, m2, m3 };  
        Arrays.sort(members);  
        System.out.println(Arrays.toString(members));  
        index = Arrays.binarySearch(members, m1);  
        System.out.println("찾은 인덱스: " + index);  
        for(Member1 m : members) {  
            System.out.print(m+" ");  
        }  
    }  
}
```

Arrays 클래스

```
public class SortExample {  
    public static void main(String[] args) {  
        int[] scores = { 99, 97, 98 };  
        Arrays.sort(scores);  
        for(int i=0; i<scores.length; i++) {  
            System.out.println("scores[" + i + "]= " + scores[i]);  
        }  
        System.out.println();  
        String[] names = { "홍길동", "박동수", "김민수" };  
        Arrays.sort(names);  
        for(int i=0; i<names.length; i++) {  
            System.out.println("names[" + i + "]= " + names[i]);  
        }  
        System.out.println();  
        Member m1 = new Member("홍길동");  
        Member m2 = new Member("박동수");  
        Member m3 = new Member("김민수");  
        Member[] members = { m1, m2, m3 };  
        Arrays.sort(members);  
        for(int i=0; i<members.length; i++) {  
            System.out.println("members[" + i + "].name=" + members[i].name);  
        }  
    }  
}
```

Arrays 클래스

```
package ch11;
public class ArraysEx {
    public static void main(String[] args) {
        Integer[] arr = {32,87,36,27,98,54};
        System.out.println(Arrays.toString(arr));
        Arrays.sort(arr);
        System.out.println(Arrays.toString(arr));
        Arrays.sort(arr,Collections.reverseOrder());
        System.out.println(Arrays.toString(arr));
        String[] name = {"효리","수지","혜리","원빈","설현"};
        System.out.println(Arrays.toString(name));
        Arrays.sort(name);
        System.out.println(Arrays.toString(name));
        Arrays.sort(name,Collections.reverseOrder());
        System.out.println(Arrays.toString(name));
    }
}
```

Arrays 클래스

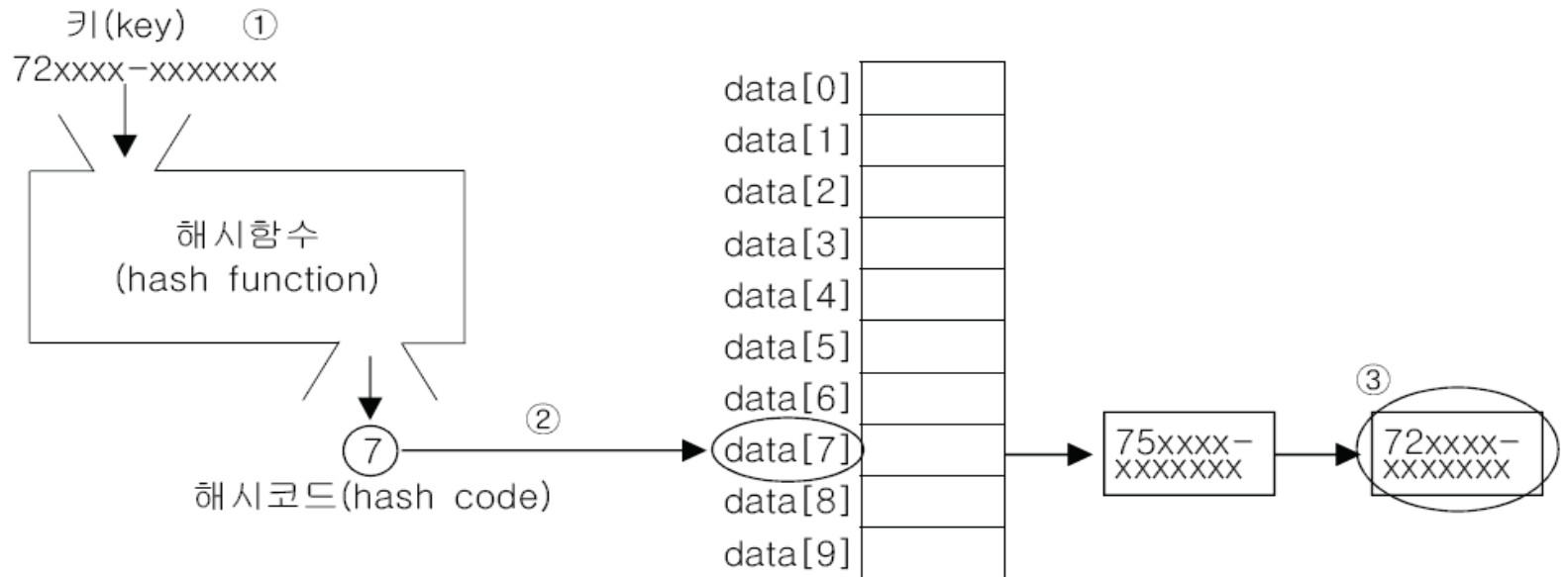
```
package ch11;
public class Member implements Comparable<Member> {
    String name;
    int age;
    Member(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public int compareTo(Member o) {
        String k1 = age+"";
        String k2 = o.age+"";
        return k1.compareTo(k2);
        // return name.compareTo(o.name);
    }
    public String toString() {
        return name+": "+age;
    }
}
```

Arrays 클래스

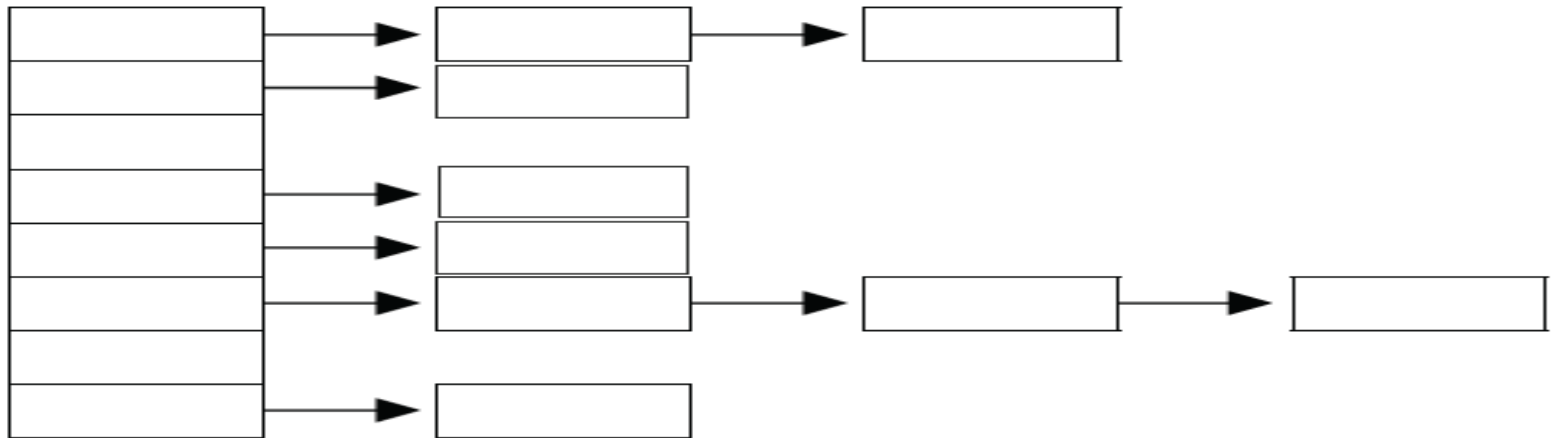
```
package ch11;
import java.util.Arrays;
import java.util.Collections;
public class MemberEx {
    public static void main(String[] args) {
        Member m1 = new Member("수지", 21);
        Member m2 = new Member("재석", 41);
        Member m3 = new Member("혜리", 25);
        Member m4 = new Member("원빈", 34);
        Member m5 = new Member("경규", 57);
        Member[] mem = {m1,m2,m3,m4,m5};
        System.out.println(Arrays.toString(mem));
        Arrays.sort(mem);
        System.out.println(Arrays.toString(mem));
        Arrays.sort(mem,Collections.reverseOrder());
        System.out.println(Arrays.toString(mem));
    }
}
```


해싱(hashing)

- 해시함수(hash function)를 이용해서 해시테이블(hash table)에 저장하고 검색하는 기법

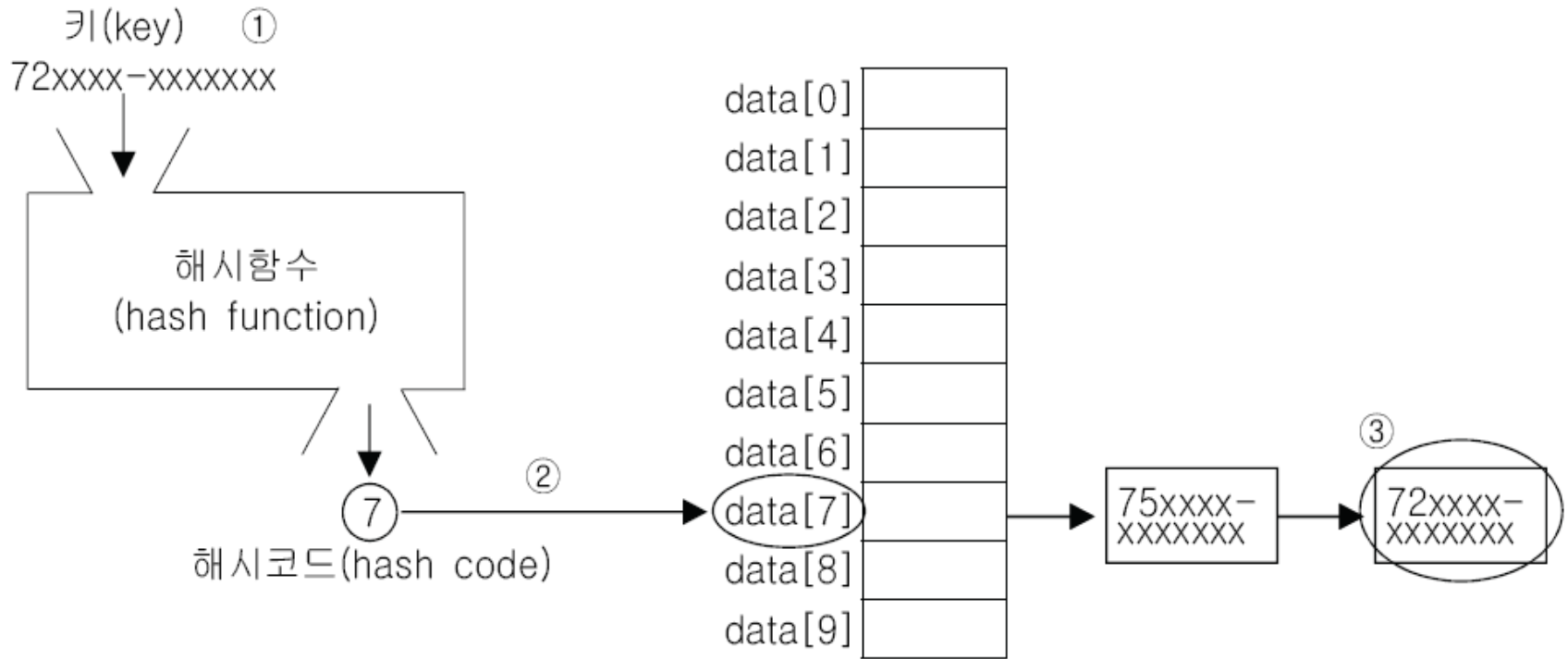


- 해싱에 사용되는 자료구조는 배열과 링크드리스트가 조합된 형태이다.



해싱(hashing)

- ▶ 키를 이용해서 해시테이블로부터 데이터를 가져오는 과정



- ① 키를 이용해서 해시함수를 호출한다.
- ② 해시함수의 호출결과인 해시코드에 대응하는 배열에 저장된 링크드리스트를 찾는다.
- ③ 링크드리스트에서 키와 일치하는 데이터를 찾는다.

※ 해시함수는 같은 키값에 대해 항상 같은 해시코드를 반환해야한다.
서로 다른 키값일지라도 같은 값의 해시코드를 반환할 수 있다.

제 네 릭

❖ 제네릭

⇒ Generics는 컬렉션(자료구조) 즉, 쉽게 말해서 객체들을 저장(수집)하는 구조적인 성격을 보장하기 위해 제공되는 것이다.

간단히 예를 들자면 컵이라는 특정 객체가 있다. 이 컵은 물만 담을 수 있는 물컵, 또는 이 컵은 주스만 담을 수 있는 주스 컵! 이렇게 상징적인 것이 바로 Generics다!



○ 제네릭의 필요성

- 컴파일 단계'에서 '잘못된 타입 사용될 수 있는 문제' 제거 가능
- 자바5부터 새로 추가 !
- 컴파일 시 강한 타입 체크 가능
- 실행 시 타입 에러가 나는 것 방지
- 컴파일 시에 미리 타입을 강하게 체크해서 에러 사전 방지
- 타입변환 제거 가능

```
List list = new ArrayList();  
list.add("hello");  
String str = (String) list.get(0);
```



```
List<String> list = new ArrayList<String>();  
list.add("hello");  
String str = list.get(0);
```

○ 제네릭 타입이란?

- 타입을 파라미터로 가지는 클래스와 인터페이스
- 선언 시 클래스 또는 인터페이스 이름 뒤에 “<>” 부호 붙임
- “<>” 사이에는 타입 파라미터 위치
- 타입 파라미터
일반적으로 대문자 알파벳 한 문자로 표현
개발 코드에서는 타입 파라미터 자리에 구체적인 타입을 지정

GenericTest1

```
import java.util.*;
public class GenericTest1 {
    public static void main(String[] args) {
        Vector<Integer> v = new Vector<Integer>(5,3);
        v.add(1); v.add(4); v.add(2); v.add(8); v.add(3);
        // v.add("abc"); // 예러
        prints(v)
    }
    public static void prints(Vector<Integer> vi) {
        int num = vi.size();
        int sum = 0;
        for (int i = 0; i < num; i++) {
            sum += vi.get(i);
        }
        System.out.println(" 합은 = " + sum);
    }
}
```

GenericTest2

```
import java.util.*;
class Car {
    public void print() {
        System.out.println("ㄴㅏ ㅅㅏ 0ㅏ");
    }
}
class Bus extends Car {
    public void print() {
        System.out.println("ㄴㅏ ㅅㅏ 0ㅏ");
    }
    public void move() {
        System.out.println("ㅅㅏ 60 0/고 승객은 60 명 0/ㅏ");
    }
}
class Taxi extends Car {
    public void print() {
        System.out.println("ㄴㅏ 노란 택시 0ㅏ");
    }
}
```



```
public class GenericTest2 {
```

```
    public static void main(String[] args) {
```

```
        ArrayList<Car> car = new ArrayList<>();
```

```
        car.add(new Bus()); car.add(new Taxi()); car.add(new Taxi());
```

```
        ArrayList<Bus> al = new ArrayList<Bus>(); // 객체도 제네릭스 사용
```

```
        al.add(new Bus()); // al.add(new Taxi());
```

```
        al.get(0).print(); al.add((Bus)car.get(0));
```

```
        // al.add((Bus)car[1]);
```

```
        for(Car c: car) {
```

```
            c.print();
```

```
            if (c instanceof Bus)
```

```
                ((Bus)c).move();
```

```
        }
```

```
    }
```

```
}
```



○ <1글자로 된 영문대문자>

- API에서는 전달되는 객체가 현 객체 내에서 자료형(Type)으로 쓰일 때 <T>로 유도를 하고 있으며 만약 전달되는 객체가 현 객체 내에서 하나의 요소(Element)로 자리를 잡을 때는 <E>로 그리고 전달되는 객체가 현 객체 내에서 Key값으로 사용될 때는 <K>로, 만약 전달되는 객체가 현 객체 내에서 Value값으로 사용될 때 <V>로 표현하고 있다.

[접근제한] class 클래스명 <유형1, 유형2...유형n>

T s;
또는
T[]
arr;

객체가 생성시 전달된 상징적 자료형(Generic Type)이 String형이었다면 왼쪽의 코드는 다음과 같이 대체(代替)된다.

```
String s;  
String[] arr;
```


○ 제네릭 타입 사용하기

- 앞서 간단한 Generic class를 우리가 직접 정의해 보았다. 이제 이것을 사용하려면 Generic class의 변수 선언과 생성 법을 익히면 된다. 다음의 구성을 보자!


```
Generic_class명<적용할_Generic_Type> 변수명; // 선언  
변수명 = new Generic_class생성자명<적용할_Generic_Type>(); // 생성
```

GenericEx1Main.java

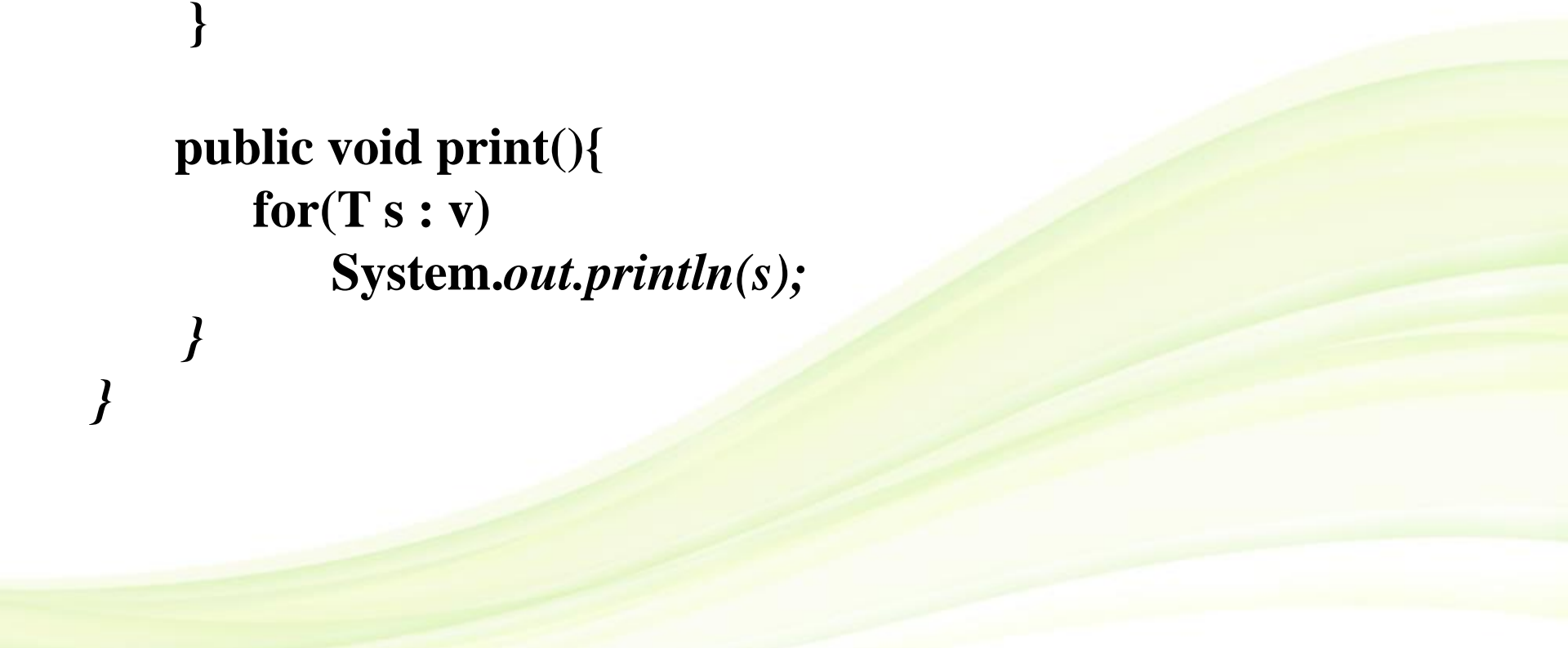
```
public class GenericEx1Main{  
    public static void main(String[] args){  
        GenericEx1<String> t = new GenericEx1<String>();  
  
        String[] ss = {"애","아","서"};  
        t.set(ss);  
        t.print();  
        // 좋은 방법이 아님  
        GenericEx1 t1 = new GenericEx1();  
        Integer[] s = {1,2,3};  
        t1.set(s);  
        t1.print();  
    }  
}
```

실행결과

```
----- Java Run -----  
애  
아  
서  
Normal Termination  
출력 완료 (0초 경과).
```



```
class GenericEx1<T> {  
  
    T[] v;  
  
    public void set(T[] n){  
        v = n;  
    }  
  
    public void print(){  
        for(T s : v)  
            System.out.println(s);  
    }  
}
```



```
class Orange {
    int sugarContent;    // 당분 함량
    public Orange(int sugar) { sugarContent=sugar; }
    public void showSugarContent() {
        System.out.println("당도 "+sugarContent);
    }
}

class FruitBox {
    Object item;
    public void store(Object item) { this.item=item; }
    public Object pullOut() { return item; }
}

class ObjectBaseFruitBox {
    public static void main(String[] args) {
        FruitBox fBox1=new FruitBox();
        fBox1.store(new Orange(10));
        Orange org1=(Orange)fBox1.pullOut();
        org1.showSugarContent();
        FruitBox fBox2=new FruitBox();
        fBox2.store("오렌지");
        Orange org2=(Orange)fBox2.pullOut();
        org2.showSugarContent();
    }
}
```

```

class Orange {
    int sugarContent; // 당분 함량
    public Orange(int sugar) { sugarContent=sugar; }
    public void showSugarContent() { System.out.println("당도 "+sugarContent); }
}

class Apple {
    int weight; // 사과 무게
    public Apple(int weight) { this.weight=weight; }
    public void showAppleWeight() { System.out.println("무게 "+weight); }
}

class FruitBox<T> {
    T item;
    public void store(T item) { this.item=item; }
    public T pullOut() { return item; }
}

class GenericBaseFruitBox {
    public static void main(String[] args) {
        FruitBox<Orange> orBox=new FruitBox<Orange>();
        orBox.store(new Orange(10)); Orange org=orBox.pullOut(); org.showSugarContent();
        FruitBox<Apple> apBox=new FruitBox<Apple>();
        apBox.store(new Apple(20)); Apple app=apBox.pullOut(); app.showAppleWeight();
    }
}

```

연습문제

다음은 정수집합 1,2,3,4와 3,4,5,6의 교집합, 차집합, 합집합을 구하는 코드이다. 코드를 완성하여 실행결과와 같은 결과를 출력하시오.

[Hint] ArrayList클래스의 `addAll()`, `removeAll()`, `retainAll()`을 사용하라.

```
import java.util.*;
class Exercise11_1 {
    public static void main(String[] args) {
        ArrayList list1 = new ArrayList();  ArrayList list2 = new ArrayList();
        ArrayList kyo = new ArrayList(); // 교집합
        ArrayList cha = new ArrayList(); // 차집합
        ArrayList hap = new ArrayList(); // 합집합
        list1.add(1); list1.add(2); list1.add(3); list1.add(4);
        list2.add(3); list2.add(4); list2.add(5); list2.add(6);
        kyo.addAll(list1); // list1의 모든 요소를 kyo에 저장한다.
        kyo.retainAll(list2); // list2와 kyo의 공통요소만 남기고 삭제한다.
        cha.addAll(list1);  cha.removeAll(list2); // cha에서 list2와 공통된 요소들을 모두 삭제한다.
        hap.addAll(list1); // list1의 모든 요소를 hap에 저장한다.
        hap.removeAll(kyo); // hap에서 kyo와 공통된 모든 요소를 삭제한다.
        hap.addAll(list2); // list2의 모든 요소를 hap에 저장한다.
        System.out.println("list1="+list1);  System.out.println("list2="+list2);
        System.out.println("kyo="+kyo);      System.out.println("cha="+cha);
        System.out.println("hap="+hap);
    }
}
```

고객관리

강사 강 병준

Customer

```
package customer;
import java.util.Date;
public class Customer {
    private String id;           private String pass;
    private String email;       private String name;
    private Date reg_date;
    public Customer() { };
    public Customer(String id,String pass,String email,String name,Date reg_date) {
        this.id=id; this.pass = pass; this.email = email;
        this.name = name; this.reg_date = reg_date;
    }
    public String toString() {
        return "아이디:"+id+", 암호:"+pass+", 이메일:"+email+
            ", 이름:"+name+", 가입일:"+reg_date;
    }
    public boolean passChk(String pass, String confirmPass) {
        return pass.equals(confirmPass);
    }
    public String getId() {           return id;           }
    public void setId(String id) { this.id = id;           }
```

Customer

```
public String getPass() {  
    return pass;  
}  
public void setPass(String pass) {  
    this.pass = pass;  
}  
public String getEmail() {          return email;      }  
public void setEmail(String email) {  
    this.email = email;  
}  
public String getName() {          return name;      }  
public void setName(String name) {  
    this.name = name;  
}  
public Date getReg_date() {          return reg_date;  }  
public void setReg_date(Date reg_date) {  
    this.reg_date = reg_date;  
}  
}
```


CustomerDaoImpl

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
public class CustomerDaoImpl {
    private Map<String, Customer> map=new HashMap<String, Customer>();
    public Customer select(String email) {
        return map.get(email);
    }
    public int insert(Customer member) {
        map.put(member.getId(), member);
        return 1;
    }
    public Collection<Customer> list() {
        return map.values();
    }
    public int update(Customer member) {
        map.put(member.getId(), member);
        return 1;
    }
    public int delete(String id) {
        map.remove(id);
        return 1;
    }
}
```

CustomerServiceImpl

```
package customer;
import java.util.Collection;
public class CustomerServiceImpl {
    private static CustomerDaoImpl md = new CustomerDaoImpl();
    public int insert(Customer customer) {
        int result = 0;
        Customer ct = md.select(customer.getId());
        if (ct == null) {
            result = md.insert(customer);
        }else System.out.println("이미 있는 데이터 입니다");
        return result;
    }
    public Customer select(String id) {
        return md.select(id);
    }
    public Collection<Customer> list() {
        return md.list();
    }
}
```

CustomerServiceImpl

```
public int update(Customer customer) {
    int result = 0;
    Customer ct = md.select(customer.getId());
    if (ct != null) {
        result = md.update(customer);
    }else System.out.println("없는 데이터는 수정 못합니다");
    return result;
}

public int delete(String id) {
    int result = 0;
    Customer customer = md.select(id);
    if (customer != null) {
        result = md.delete(id);
    }else System.out.println("없는 데이터는 삭제 못합니다");
    return result;
}

}
```

Ex01

```
public class Ex01 {  
    private static CustomerServiceImpl ms = new CustomerServiceImpl();  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        while(true) {  
            System.out.println("명령어를 입력하세요");  
            String command = sc.nextLine();  
            if (command.equals("exit")) break;  
            else if (command.startsWith("insert")) {  
                insert(command.split(","));  
            } else if (command.startsWith("select")) {  
                select(command.split(","));  
            } else if (command.equals("list"))  
                list();  
            else if (command.startsWith("update"))  
                update(command.split(","));  
            else if (command.startsWith("delete"))  
                delete(command.split(","));  
            else help();  
        }  
        sc.close();  
        System.out.println("프로그램 종료");  
    }  
}
```

Ex01

```
private static void delete(String[] str) {
    if (str.length != 2) {
        help(); return;
    }
    int result = ms.delete(str[1]);
    if (result > 0) System.out.println("삭제 됐네");
}

private static void update(String[] str) {
    if (str.length != 6) {
        help(); return;
    }
    Customer customer = new Customer();

    customer.setId(str[1]);    customer.setEmail(str[2]);
    customer.setName(str[3]); customer.setPass(str[4]);
    customer.setReg_date(new Date());
    if (!customer.passChk(str[4],str[5])) {
        System.out.println("암호와 암호 확인이 다릅니다");
        return;
    }
    int result = ms.update(customer);
    if (result > 0) System.out.println("회원수정 완료");
}
```

Ex01

```
private static void list() {  
    Collection<Customer> list = ms.list();  
    if (list == null || list.size() == 0)  
        System.out.println("없는 데이터입니다");  
    else  
        for(Customer customer : list) {  
            System.out.println(customer);  
        }  
}  
private static void select(String[] str) {  
    if (str.length != 2) {  
        help(); return;  
    }  
    Customer customer = ms.select(str[1]);  
    if (customer == null) System.out.println("없는 데이터 입니다");  
    else System.out.println(customer);  
}
```

Ex01

```
private static void insert(String[] str) {  
    if (str.length != 6) {  
        help(); return;  
    }  
    Customer customer = new Customer();  
    customer.setId(str[1]);    customer.setEmail(str[2]);  
    customer.setName(str[3]); customer.setPass(str[4]);  
    customer.setReg_date(new Date());  
    if (!customer.passChk(str[4],str[5])) {  
        System.out.println("암호와 암호 확인이 다릅니다");  
        return;  
    }  
    int result = ms.insert(customer);  
    if (result > 0) System.out.println("회원등록 완료");  
}
```

Ex01

```
private static void help() {  
    System.out.println();  
    System.out.println("잘못된 명령어 입니다");  
    System.out.println("다음 명령어 중에서 하나를 사용하세요");  
    System.out.println("insert,id,이메일,이름,암호,암호확인");  
    System.out.println("update,id,이메일,이름,암호,암호확인");  
    System.out.println("select,id");  
    System.out.println("delete,id");  
    System.out.println("list");  
    System.out.println("exit");  
    System.out.println();  
}  
}
```