11. 제네릭과 컬렉션

강사 강태광

I] 컬렉션 프레임웍(collection framework)이란

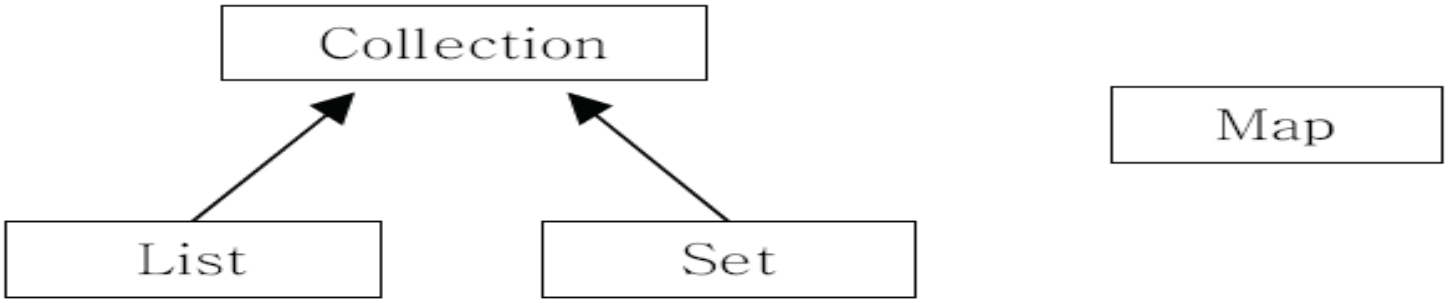
▶ 컬렉션 프레임웍(collection framework)
- 데이터 군(群)을 저장하는 클래스들을 표준화한 설계
- 다수의 데이터를 쉽게 처리할 수 있는 방법을 제공하는 클래스들로 구성
- JDK1.2부터 제공

▶ 컬렉션 (collection)
- 다수의 데이터, 즉, 데이터 그룹을 의미한다.

▶ 프레임웍(framework) - 표준화, 정형화된 체계적인 프로그래밍 방식

▶ 컬렉션 클래스(collection class)
- 다수의 데이터를 저장할 수 있는 클래스(예, Vector, ArrayList, HashSet)

2] 컬렉션 프레임웍의 핵심 인터페이스



[그림11-1] 컬렉션 프레임웍의 핵심 인터페이스간의 상속계층도

인터페이스	특 징
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

[표11-1] 컬렉션 프레임웍의 핵심 인터페이스와 특징

3]컬렉션-프레임웍의 동기화(synchronization)

- 멀티쓰레드 프로그래밍에서는 컬렉션 클래스에 동기화 처리가 필요하다.
- Vector와 같은 구버젼 클래스들은 자체적으로 동기화처리가 되어 있다.
- ArrayList와 같은 신버젼 클래스들은 별도의 동기화 처리가 필요하다.
- Collections클래스는 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(...));
```

3] 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

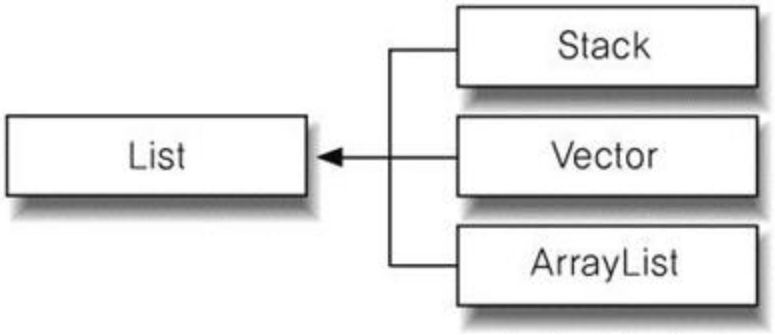
확장 for 문

형식

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

4] 컬렉션-List 인터페이스

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



4] 컬렉션-List 인터페이스

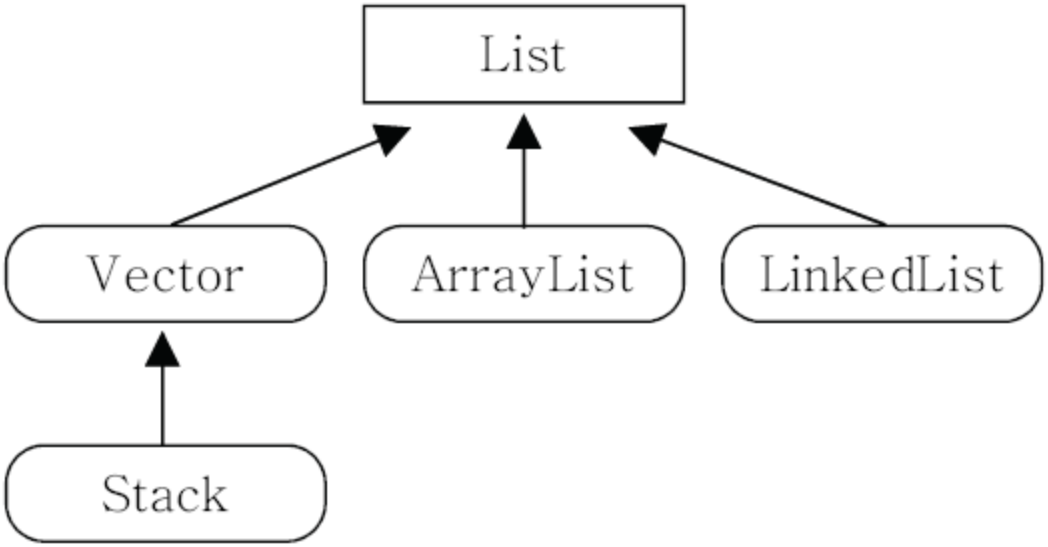
[표 7-7] List의 구현 클래스

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 구조

4] 컬렉션-List (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



4] 컬렉션-List (Vector와 ArrayList)

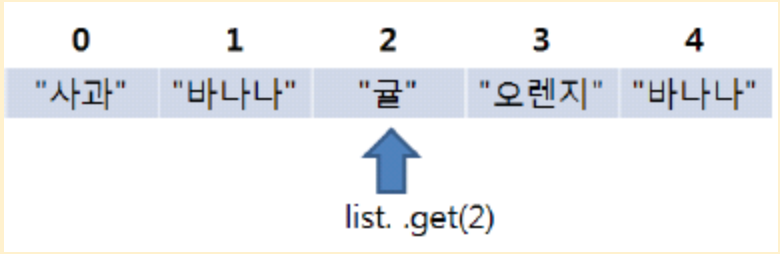
1. ArrayList 클래스

```
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"

2. ArrayList 클래스 -> get 메소드

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

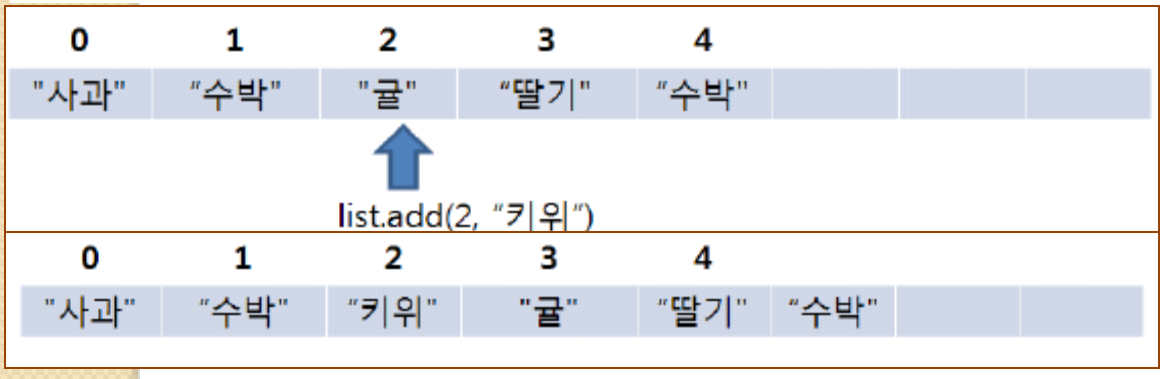


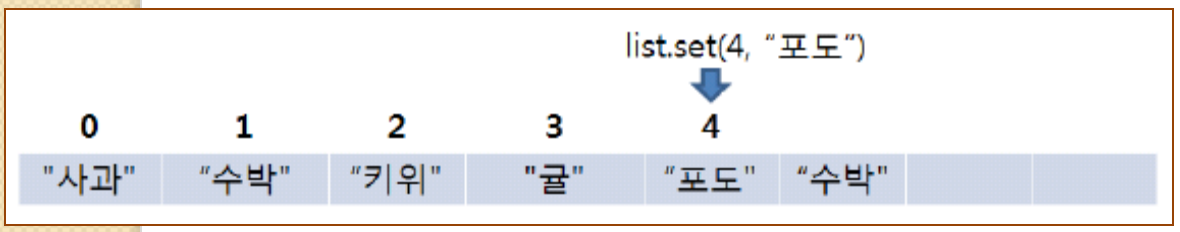
4] ArrayList 적용 예시

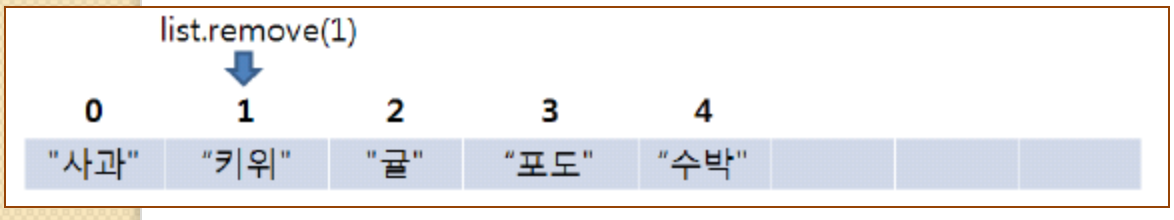
```
package ch11;

import java.util.ArrayList;
import java.util.Iterator;
public class ArrayList1 {
	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();
	}
}
```

4]ArrayList에서 원소 추가, 변경, 삭제하기

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







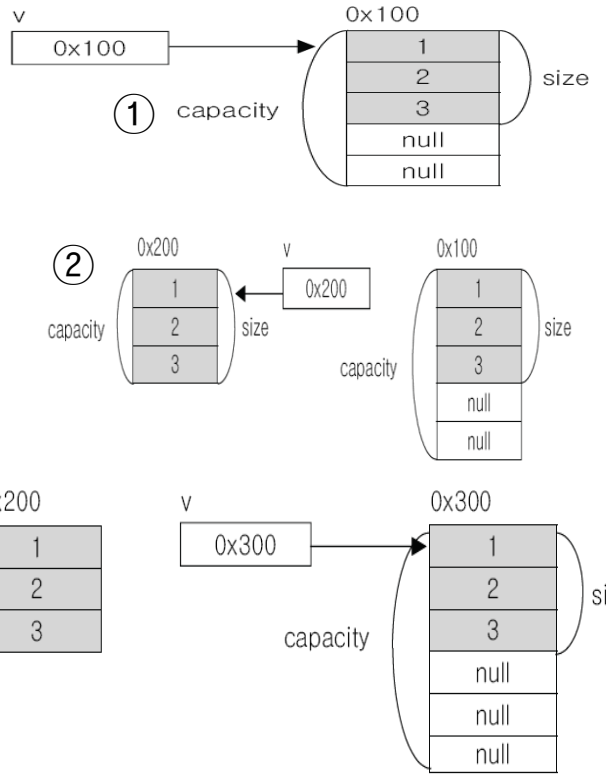
4] ArrayList 적용 예시

```
package ch11;

import java.util.ArrayList;
import java.util.Iterator;
public class ArrayList1 {
	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();
	}
}
```

5] 컬렉션-List (Vector)1

메소드	설 명
public Vector()	디폴트 생성자로 빈 벡터 객체를 생성한다.
public Vector(int initialCapacity)	initialCapacity로 지정한 크기의 벡터 객체를 생성한다.
public Vector(int initialCapacity, int capacityIncrement)	initialCapacity로 지정한 크기의 벡터 객체를 생성하되, 새로운 요소가 추가되어 원소가 늘어나야 하면 capacityIncrement만큼 늘어난다.

```
// 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();
```



5] 컬렉션-List (Vector)2

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()	해시코드를 반환한다.

5] 컬렉션-List (Vector)3

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

5] 컬렉션-List (Vector) 예시

```
package ch11;

import java.util.Vector;

public class Vector2 {
    public static void main(String[] args) {
        Vector<String>  vc = new Vector<>();
        vc.add("수박");
        vc.add("사과");
        vc.add("바나나");
        vc.add("수박");
        vc.add("대추");
        vc.add("바나나");
        System.out.println("갯수 : " + vc.size());
        print(vc);
        // 1번 자리에 끼워 넣기 가능하다
        vc.add(1,"키위");
        print(vc);
        vc.set(4,"봉숭아");
        print(vc);
        vc.remove(0);
        print(vc);
```

5] 컬렉션-List (Vector) 예시(계속)

```
System.out.println("3번 인덱스 -->" + vc.get(3));
System.out.println("바나나 문의 : " + vc.contains("바나나"));
System.out.println("바나나 문의 : " + vc.indexOf("바나나"));
System.out.println("바나나 문의 : " + vc.lastIndexOf("바나나"));
for (int i = 0 ; i < vc.size(); i++ ) {
        if (vc.get(i).equals("바나나")) {
            vc.set(i, "딸기");
        }
}
print(vc);
}
public static void print(Vector<String> vc) {
    for ( String str : vc) {
            System.out.print(str + "₩t");
    }
    System.out.println("₩n----------------------------------------");
    }

}
```

5] 컬렉션-List (ArrayList) 단점

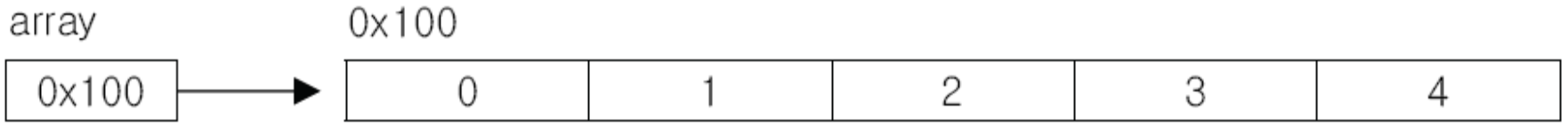
1) 배열은 구조가 간단하고 데이터를 읽어오는 데 걸리는 시간(접근시간, access time)이 가장 빠르다는 장점이 있지만 단점도 있다.

▶ 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.

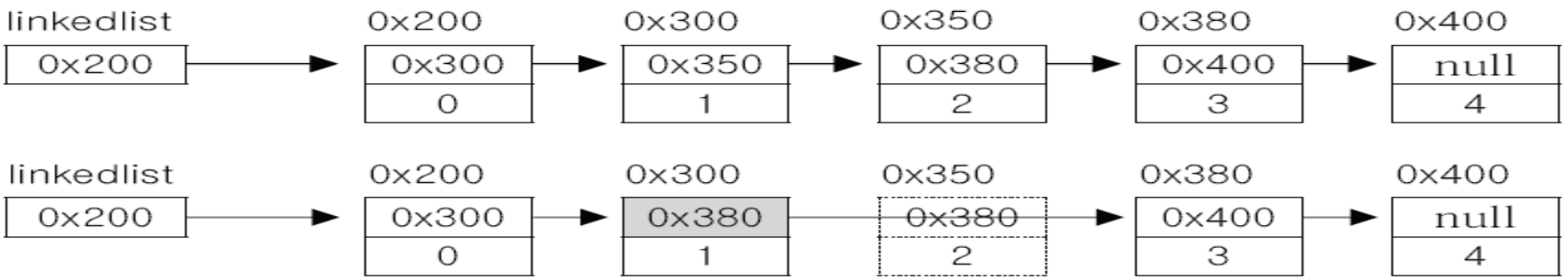
▶ 단점 2 : 비순차적인 데이터의 추가, 삭제에 시간이 많이 걸린다.
- 데이터를 추가하거나 삭제하기 위해, 많은 데이터를 옮겨야 한다
- 그러나, 순차적인 데이터 추가(마지막에 추가)와 순차적으로 데이터를 삭제하는 것(마지막에서부터 삭제)은 빠르다.

5] 컬렉션-List (LinkedList)

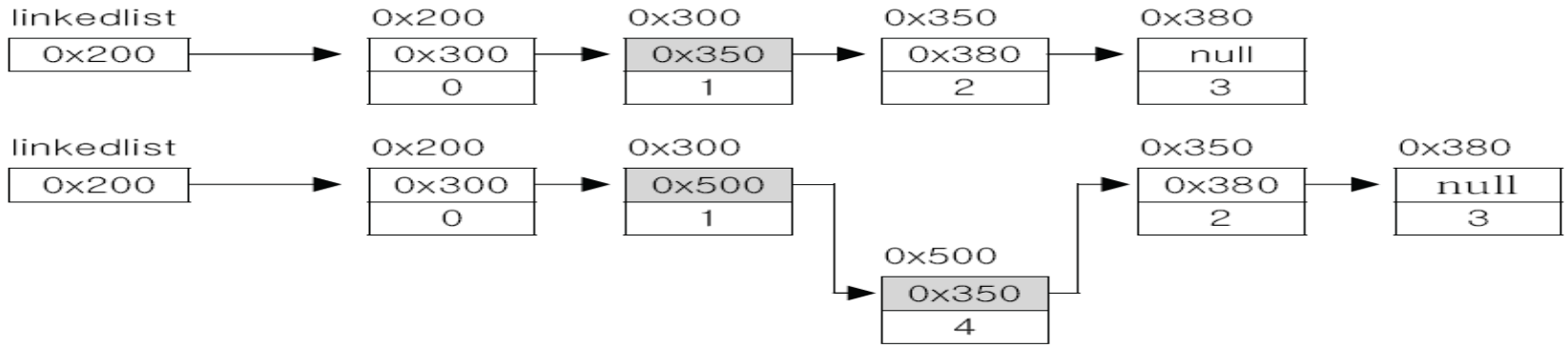
1. 배열과 달리 링크드 리스트는 불연속적으로 존재하는 데이터를 연결(link) 하며 배열의 단점을 보완



- 데이터의 삭제 : 단 한 번의 참조변경만으로 가능



- 데이터의 추가 : 하나의 Node객체생성과 한 번의 참조변경만으로 가능



5] 컬렉션-List (LinkedList)-주요 메서드1

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.

5] 컬렉션-List (LinkedList)-주요 메서드2

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.

5] 컬렉션-List (LinkedList)-주요 메서드3

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.

5] 컬렉션-List (ArrayList vs. LinkedList)

- 순차적으로 데이터를 추가/삭제하는 경우, ArrayList가 빠르다.

- 비순차적으로 데이터를 추가/삭제하는 경우, LinkedList가 빠르다.

- 접근시간(access time)은 ArrayList가 빠르다.

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

5] 컬렉션-List (Stack)

-객체를 후입선출(後入先出), last-in-first-out(LIFO)이며 객체의 저장시의 push()메서드와 검출 시 사용하는 pop()과 Stack의 가장 위쪽 객체를 의미하는 peek()메서드 그리고 Stack이 비어있는지 판별해주는 empty()와 객체를 검색해주는 search()메서드들로 Vector라는 클래스를 확장하였다

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

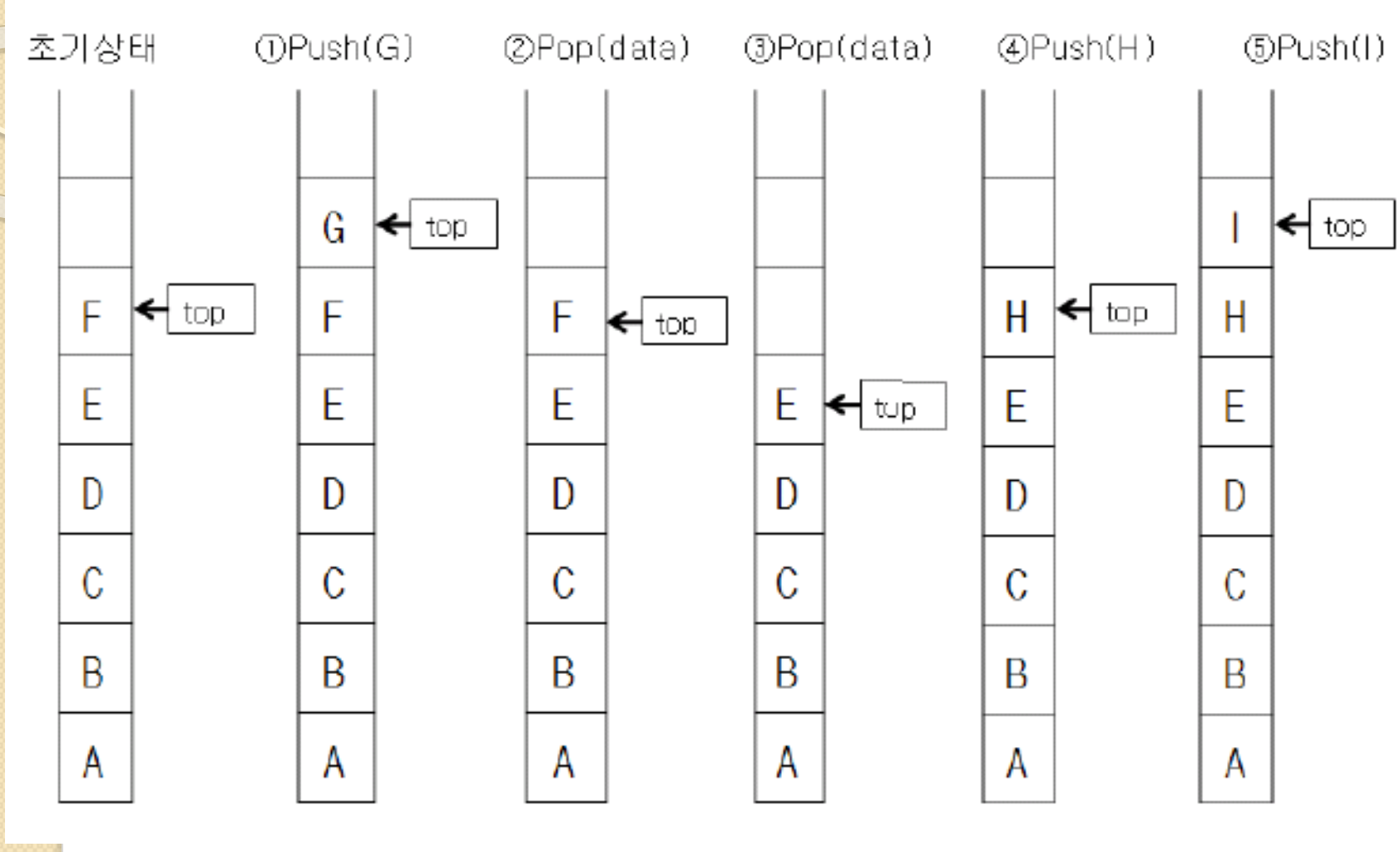
[표 7-8] Stack 생성자에 대한 요약

생성자명	설명
Stack( )	새로운 Stack 객체를 생성하고 초기화한다.

[표 7-9] Stack의 주요 메서드

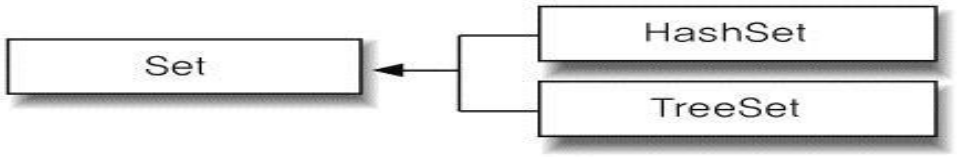
반환형	메서드명	설명
boolean	empty( )	Stack이 비었는지 비교하여 비어 있으면 true를 반환한다.
E	peek( )	Stack의 가장 위쪽에 있는 객체를 반환한다.
	pop( )	Stack의 가장 위쪽에 있는 객체를 삭제하고 그 객체를 반환한다.
	push(E item)	Stack의 가장 위쪽에서 객체를 추가한다.
int	search(Object o)	현재 Stack의 구조에서 인자로 전달받은 객체의 인덱스값을 반환한다(참고로 인덱스값은 1부터 시작한다).

5] 컬렉션-List (Stack) Push / POP



5] 컬렉션-Set인터페이스

- 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.

5] 컬렉션-Set(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

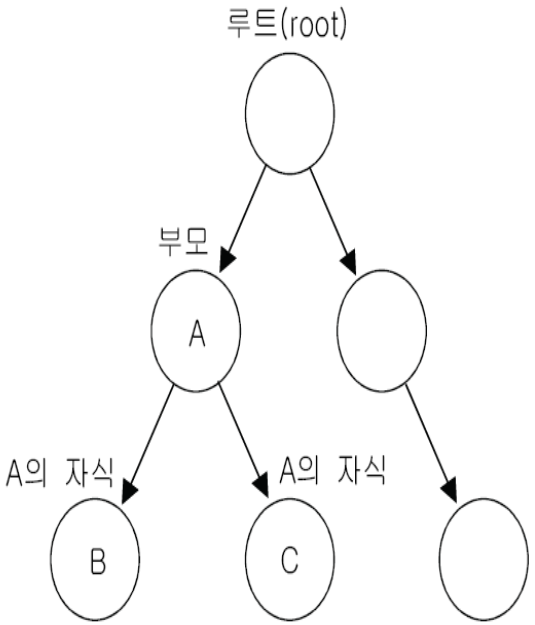
생성자명	설명
HashSet( )	새로운 HashSet 객체를 생성하고 초기화한다.

[표 7-6] HashSet 메서드 요약

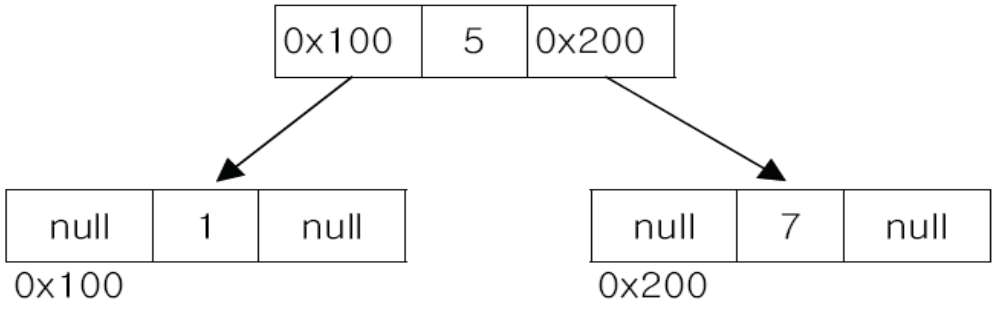
반환형	메서드명	설명
boolean	add(E o)	제네릭 타입으로 넘어온 객체가 Set 구조에 없다면 추가하고 true를 반환한다.
void	clear( )	Set 구조에 있는 모든 요소들을 삭제한다.
boolean	contains(Object o)	인자로 전달된 객체를 현 Collection에서 요소로 가지고 있으면 true를 반환한다.
	isEmpty( )	현 Collection에 저장된 요소가 없다면 true를 반환한다.
Iterator<E>	iterator( )	현 Set 구조의 요소들을 순서대로 처리하기 위해 Iterator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

5] 컬렉션-Set(TreeSet)

- Set인터페이스를 구현한 컬렉션 클래스.(중복허용×, 순서유지×, 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-자식 관계'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..

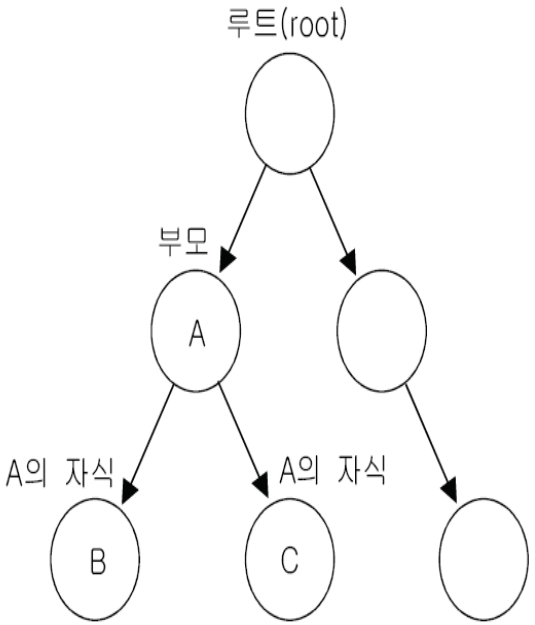



```
class TreeNode {
    TreeNode left;     // 왼쪽 자식노드
    Object   element; // 저장할 객체
    TreeNode right;    // 오른쪽 자식노드
}
```

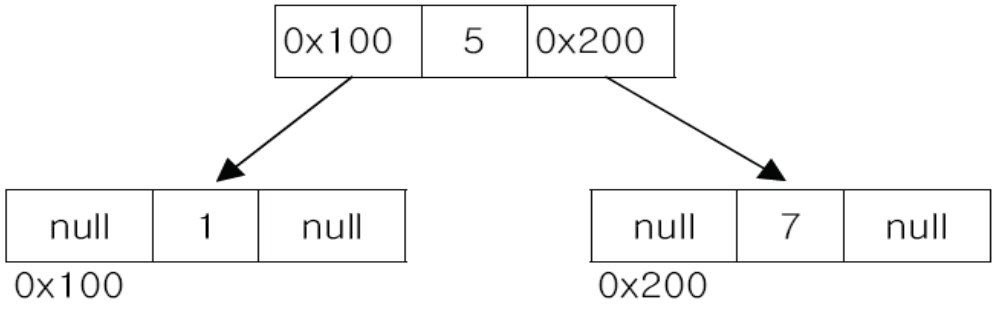


5] 컬렉션-Set(TreeSet)

- Set인터페이스를 구현한 컬렉션 클래스.(중복허용×, 순서유지×, 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-자식 관계'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
}
```

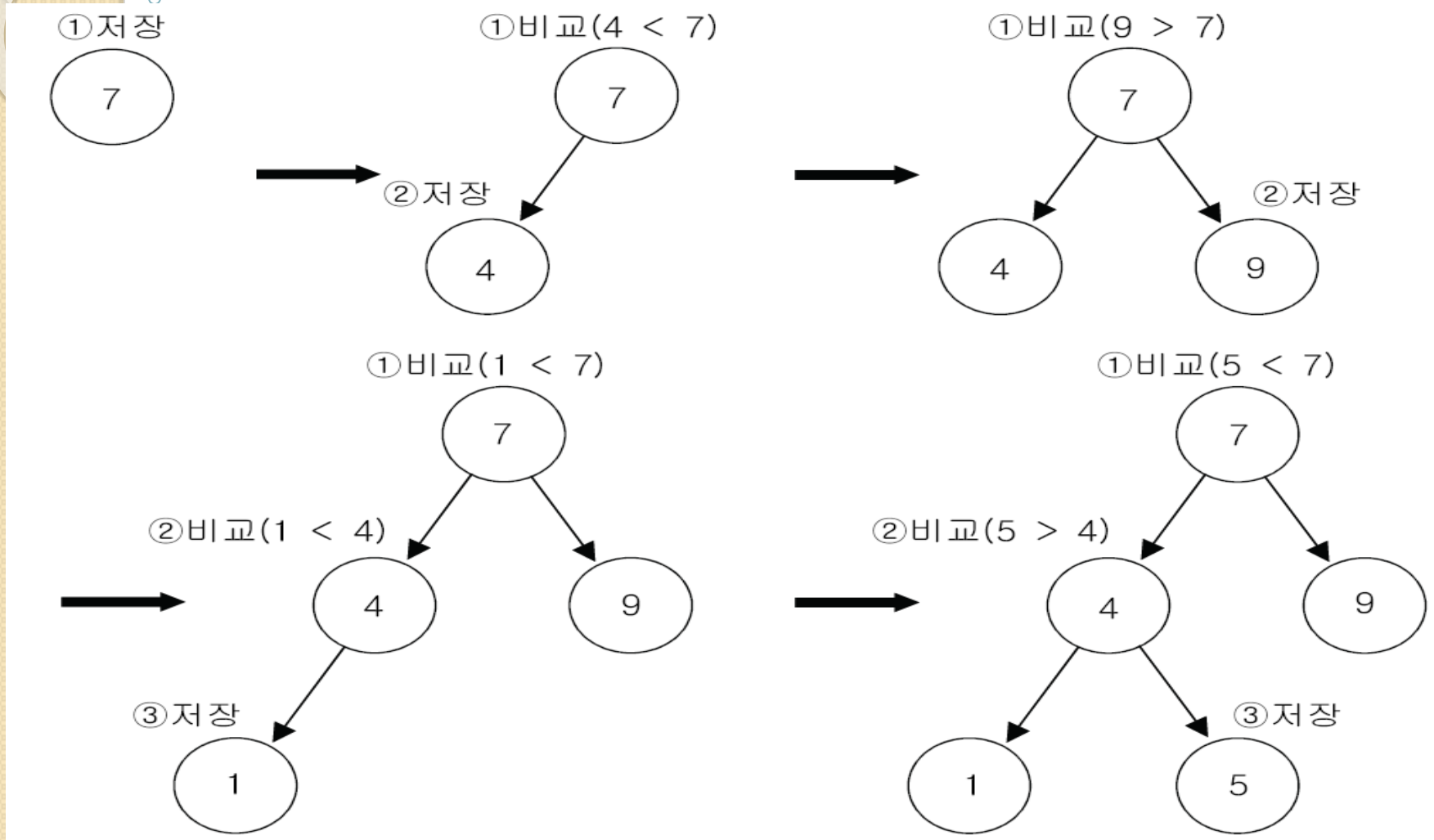


5] 컬렉션-Set(TreeSet)-주요메서드

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.

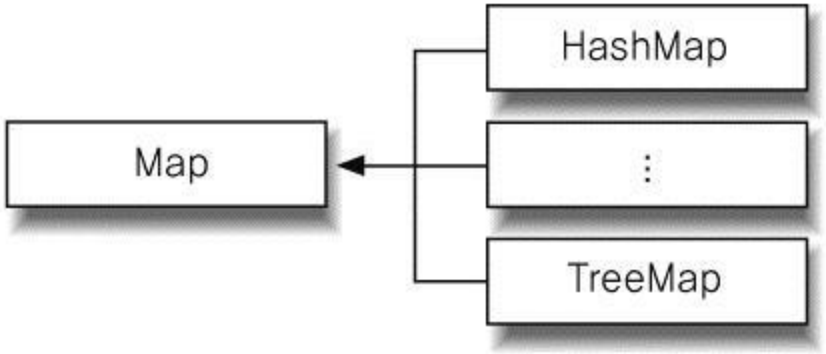
5] 컬렉션-Set(TreeSet)-데이터 저장과정

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



6] MAP인터페이스

Key와 Value를 매핑하는 객체
여기에 사용되는 Key는 절대 중복될 수 없으며 각 Key는 1개의 Value만 매핑.
정렬의 기준이 없으며 이는 마치 각 Value에 열쇠 고리를 달아서 큰 주머니에 넣어두고 오로지 Key로 각 Value를 참조 할 수 있도록 해둔 구조



[그림 7-25] Map의 구조

사용자가 원하는 Value의 Key를 알고 있다면 Key를 당겨(get) 해당 Key와 매핑되어 있는 Value를 얻을 수 있는 구조이다. 즉, 검색을 Key로 해야 하므로 Key를 모르면 원하는 Value를 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.

6] MAP인터페이스-HashMap

Key와 Value를 하나의 쌍으로 저장되는 구조이며 저장되는 Value와 Key가 null을 허용.
하지만 중복은 허용하지 않으므로 null을 가지는 Key가 2개일 수는 없다.
그리고 동기화가 포함되지 않았으므로 나중에 배우는 Multi-Thread환경에서의 구현이 아니라면 Hashtable에 비해서 처리 속도가 빠른 장점

[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.

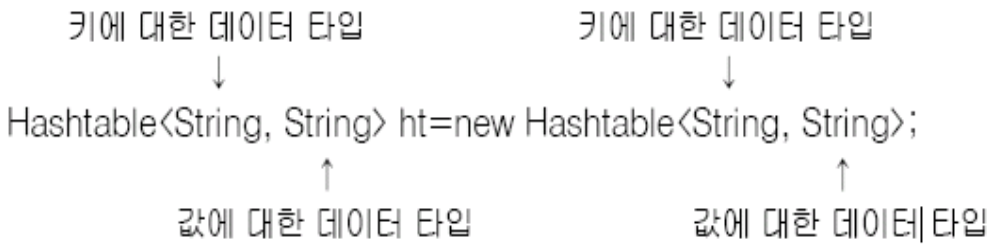
6] MAP인터페이스-HashMap 주요 메서드

[표 7-17] 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.

6] MAP인터페이스-Hashtable 클래스

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




```
package ch11;

import java.util.HashMap;

public class HashMapEx {
   public static void main(String[] args) {
        HashMap <String , String > hm = new HashMap<>();
        hm.put("이승우", "010-2222-3333");
        hm.put("손흥민", "010-3333-3333");
        hm.put("전지현", "010-4455-3333");
        System.out.println("강수지 TEL:" +hm.get("이승우"));
        System.out.println("손흥민 TEL:" +hm.get("손흥민"));
        System.out.println("전지현 TEL:" +hm.get("전지현"));

   }
}
```

6] MAP인터페이스-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

6] MAP인터페이스-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

6] MAP인터페이스-Map 컬렉션

```
package ch11;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.net.URLDecoder;
import java.util.Properties;
public class PropertiesExample {
   public static void main(String[] args)     throws FileNotFoundException, IOException
   {
                Properties pt = new Properties();
                 String path = PropertiesExample.class.getResource("database.properties").getPath();
                  path = URLDecoder.decode(path,"utf-8");
                  pt.load(new FileReader(path));
                  String driver = pt.getProperty("driver");
                  String url     = pt.getProperty("url");
                  String userName     = pt.getProperty("userame");
                  String password     = pt.getProperty("password");

                  System.out.println("드라이버 : "     + driver);
                  System.out.println("url : "        + url);
                  System.out.println("username : " + userName);
                  System.out.println("password : " + password);

                }
}
```

7] 제 네 릭

Generics는 컬렉션(자료구조) 즉, 쉽게 말해서 객체
들을 저장(수집)하는 구조적인 성격을 보강하기 위해
제공되는 것
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!

Generics 필요성

1) JDK5.0에 와서 Generics가 포함되면서 이제 프로그래머가 특정 컬렉션(자료구조)에 원하는 객체 타입을 명시하여 실행하기 전에 컴파일단계에서 지정된 객체가 아니면 절대 저장이 불가능하게 할 수 있다.

2) 이전 버전까지는 반드시 실행하여 컬렉션(자료구조)에 있는 자원들을 하나씩 검출하여 확인할 수 밖에 없었다. Object로부터 상속받은 객체는 모두 저장이 가능했던 이전의 버전들과는 달리 보다 체계(體系)적이라 할 수 있다.

3) 실행 시 자원 검출을 하게 되면 별도의 형 변환(Casting)이 필요 없이 <>사이에 선언하였던 객체자료형으로 검출되어 편리하다.

GenericTest1.java 참조