yonggyo1981 동영상 강의 URL 업데이트 708f66a · 2 years ago History

| Name | Name | Last commit date |
|---|---|---|
| .. | | |
| day13_14 | 컬렉션 프레임워크 강의 작성 | 2 years ago |
| images | 컬렉션 프레임워크 강의 작성 | 2 years ago |
| README.md | 동영상 강의 URL 업데이트 | 2 years ago |

README.md

# 강의 동영상 링크

동영상 링크

# 컬렉션 프레임워크(Collections Framework)

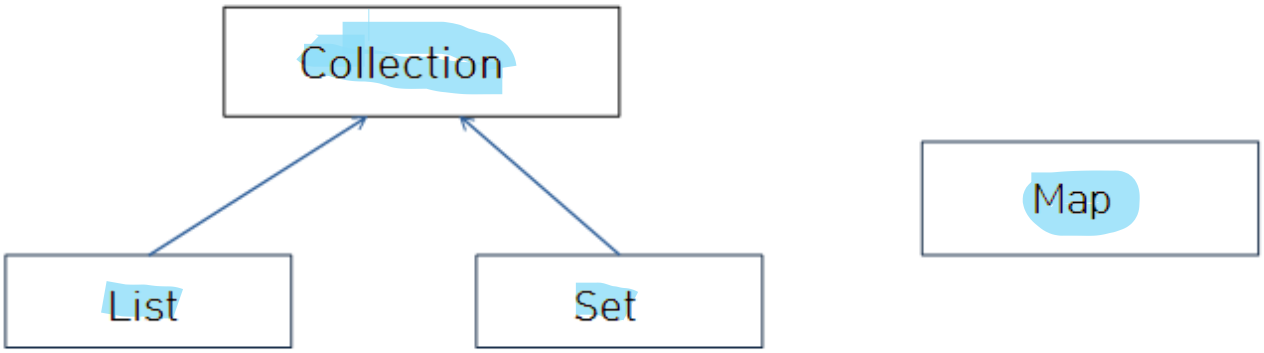
- 컬렉션 프레임워크란, '데이터 군을 저장하는 클래스등을 표준화한 설계'를 뜻한다.
- JDK1.2 이전까지는 Vector, Hashtable, Properties와 같은 컬렉션 클래스, 다수의 데이터를 저장할 수 있는 클래스들을 서로 다른 각자의 방식으로 처리해야 했다.
- JDK1.2부터 컬렉션 프레임워크가 등장하면서 다양한 종류의 컬렉션 클래스가 추가되고 모든 컬렉션 클래스를 표준화된 방식으로 다룰 수 있도록 체계화되었다.

> Vector와 같이 다수의 데이터를 저장할 수 있는 클래스를 컬렉션 클래스라고 한다.

- 컬렉션 프레임워크는 컬렉션(다수의 데이터)을 다루는 데 필요한 다양하고 풀부한 클래스를 제공한다.
- 인터페이스의 다형성을 이용한 객체지향적 설계를 통해 표준화되어 있기 때문에 사용법을 익히기에도 편리하고 재사용성이 높은 코드를 작성할 수 있다는 장점이 있다.

# 컬렉션 프레임워크 핵심 인터페이스

- 컬렉션 프레임워크에서는 컬렉션데이터 그룹을 크게 3가지 타입이 존재한다고 인식하고 컬렉션을 다루는데 필요한 기능을 가진 3개 인터페이스(List, Set, Map)를 정의하였다.
- 인터페이스 List와 Set의 공통된 부분을 다시 뽑아서 새로운 인터페이스인 Collection을 추가로 정의하였다.




> 인터페이스의 List와 Set을 구현한 컬렉션 클래스들은 서로 많은 공통부분이 있어서, 공통된 부분을 다시 뽑아 Collection인터페이스를 정의할 수 있었지만 Map 인터페이스는 이들과는 전혀 다른 형태로 컬렉션을 다루기 때문에 같은 상속계층도에 포함되지 못했다.

| 인터페이스 | 특징 |
|---|---|
| List | 순서가 있는 데이터의 집합. 데이터의 중복을 허용한다.<br>구현클래스:ArrayList, LinkedList, Stack,Vector등 |
| Set | 순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다.<br>구현클래스: HashSet, TreeSet 등 |
| Map | 키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합<br>순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용한다.<br>구현클래스: HashMap, TreeMap, Hashtable, Properties |

> 키(Key)란, 데이터 집합 중에서 어떤 값(value)을 찾는데 열쇠(key)가 된다는 의미에서 붙여진 이름이다. 그래서 키(Key)는 중복을 허용하지 않는다.

- 컬렉션 프레임워크의 모든 컬렉션 클래스들을 List, Set, Map중 하나를 구현하고 있다.
- 구현한 인터페이스의 이름이 클래스의 이름에 포함되어 있어서 이름만으로도 클래스의 특징을 쉽게 알 수 있도록 되어 있다.
- 그러나 Vector, Stack, Hashtable, Properties와 같은 클래스들은 컬렉션 프레임워크가 만들어지기 이전부터 존재하던 것이기 때문에 컬렉션 프레임워크의 명명법을 따르지 않는다.
- Vector나 Hashtable과 같은 기존의 컬렉션 클래스들은 호환을 위해, 설계를 변경해서 남겨두었지만 가능하면 사용하지 않는 것이 좋다. 그 대신 새로 추가된 ArrayList와 HashMap을 사용하는 것이 좋다.

## Collection인터페이스

- Collection은 List와 Set의 상위 인터페이스 이다.
- Collection 인터페이스는 컬렉션 클래스에 저장된 데이터를 읽고, 추가하고, 삭제하는 등 컬렉션을 다루는데 가장 기본이적인 메서드들을 정의하고 있다.

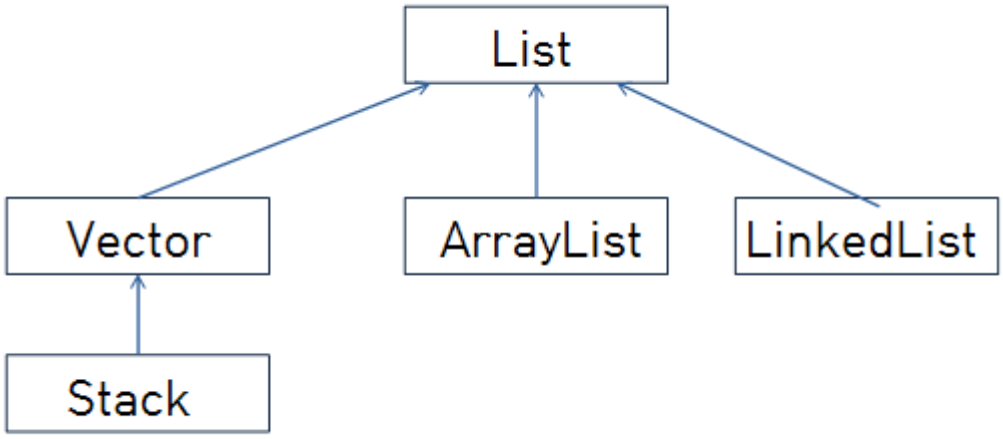
**Collection 인터페이스에 정의된 메서드**

| 메서드 | 설명 |
|---|---|
| boolean add(Object o)<br>boolean addAll(Collection c) | 지정된 객체(o) 또는 Collection(c)의 객체들을 Collection에 추가한다. |
| void clear() | Collection의 모든 객체를 삭제한다. |
| boolean contains(Object o)<br>boolean containsAll(Collection c) | 지정된 객체(o) 또는 Collection의 객체들이 Collection에 포함되어 있는지 확인한다. |
| boolean equals(Object o) | 동일한 Collection인지 비교한다. |
| int hashCode() | Collection의 hash code를 반환한다. |
| boolean isEmpty() | Collection이 비어 있는지 확인한다. |
| Iterator iterator() | Collection의 iterator를 얻어서 반환한다. |
| boolean remove(Object o) | 지정된 객체를 삭제한다. |
| boolean removeAll(Collection c) | 지정된 Collection에 포함된 객체들을 삭제한다. |
| boolean retainAll(Collection c) | 지정된 Collection에 포함된 객체만 남기고 다른 객체들은 Collection에서 삭제한다. 이 작업으로 인해 Collection에 변화가 있으면 true를 그렇지 않다면 false를 반환한다. |
| int size() | Collection에 저장된 객체의 개수를 반환한다. |
| Object[] toArray() | Collection에 저장된 객체를 객체배열(Object[])로 반환한다. |
| Object[] toArray(Object[] a) | 지정된 배열에 Collection의 객체를 지정해서 반환한다. |

## List 인터페이스

List 인터페이스는 **중복을 허용**하면서 **저장순서가 유지**되는 컬렉션을 구현하는데 사용된다.

**List의 상속계층도**




**List인터페이스에 정의된 메서드**

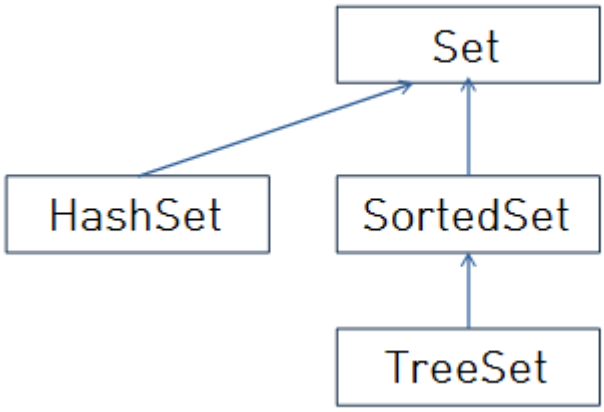
Collection 인터페이로부터 상속받은 것은 제외하였다.

| 메서드 | 설명 |
|---|---|
| void add(int index, Object element)<br>boolean addAll(int index, Collection c) | 지정된 위치(index)에 객체(element) 또는 컬렉션에 포함된 객체들을 추가한다. |
| Object get(int index) | 지정된 위치(index)에 있는 객체를 반환한다. |
| int indexOf(Object o) | 지정된 객체의 위치(index)를 반환한다.<br>(List의 첫 번째 요소로부터 순방향으로 찾는다.) |
| int lastIndexOf(Object o) | 지정된 객체의 위치(index)를 반환한다.<br>(List의 마지막 요소로 부터 역방향으로 찾는다.) |
| ListIterator listIterator()<br>ListIterator listIterator(int index) | List의 객체에 접근할 수 있는 ListIterator를 반환한다. |
| Object remove(int index) | 지정된 위치(index)에 있는 객체를 삭제하고 삭제된 객체를 반환한다. |
| Object set(int index, Object element) | 지정된 위치(index)에 객체(element)를 저장한다. |
| void sort(Comparator c) | 지정된 비교자(Comparator)로 List를 정렬한다. |
| List subList(int fromIndex, int toIndex) | 지정된 범위(fromIndex부터 toIndex)에 있는 객체를 반환한다. |

## Set 인터페이스

- Set 인터페이스는 중복을 허용하지 않고 저장순서가 유지되지 않는 컬렉션 클래스를 구현하는데 사용된다.
- Set 인터페이스를 구현한 클래스로는 HashSet, TreeSet 등이 있다.

Set의 상속계층도

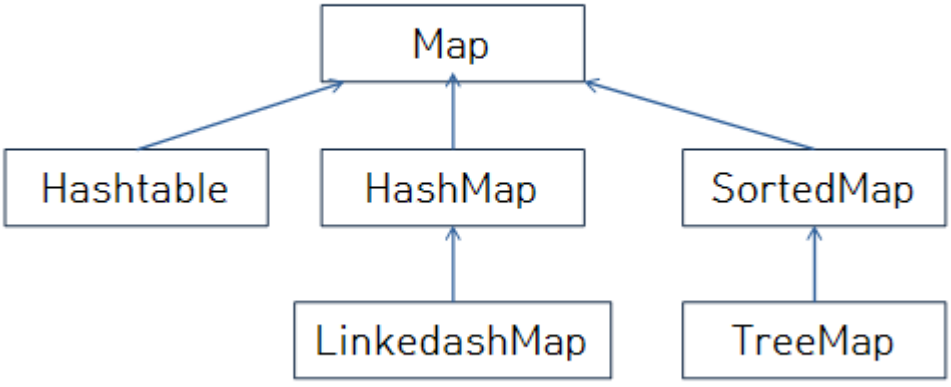



## Map 인터페이스

- Map인터페이스는 키(key)와 값(value)을 하나의 쌍으로 묶어서 저장하는 컬렉션 클래스를 구현하는 데 사용된다.
- 키는 중복될 수 없지만 값은 중복을 허용한다.
- 기존에 저장된 데이터와 중복된 키와 값을 저장하면 기존의 값은 없어지고 마지막에 저장된 값이 남게된다.
- Map인터페이스를 구현한 클래스로는 Hashtable, HashMap, LinkedHashMap, SortedMap, TreeMap등이 있다.

> Map이란 개념은 어떤 두 값을 연결한다는 의미에서 붙여진 이름이다.

Map의 상속계층도




Map인터페이스의 메서드

| 메서드 | 설명 |
| --- | --- |
| void clear() | Map의 모든 객체를 삭제한다. |
| boolean containsKey(Object key) | 지정된 key객체와 일치하는 Map의 key객체가 있는지 확인한다. |

| 메서드 | 설명 |
| --- | --- |
| boolean containsValue(Object value) | 지정된 value객체와 일치하는 Map의 value객체가 있는지 확인한다. |
| Set entrySet() | Map에 저장되는 있는 key-value쌍을 Map.Entry타입의 객체로 저장한 Set으로 반환한다. |
| boolean equals(Object o) | 동일한 Map인지 비교한다. |
| Object get(Object key) | 지정된 key객체에 대응하는 value객체를 찾아서 반환한다. |
| int hashCode() | 해시코드를 반환한다. |
| boolean isEmpty() | Map이 비어있는지 확인한다. |
| Set keySet() | Map에 저장된 모든 key객체를 반환한다. |
| Object put(Object key, Object value) | Map에 value객체를 key객체에 연결(mapping)하여 저장한다. |
| void putAll(Map f) | 지정된 Map의 모든 key-value쌍을 추가한다. |
| Object remove(Object key) | 지정한 key객체와 일치하는 key-value객체를 삭제한다. |
| int size() | Map에 저장된 key-value쌍의 개수를 반환한다. |
| Collection values() | Map에 저장된 모든 value객체를 반환한다. |

> Map 인터페이스에서 값(values)은 중복을 허용하기 때문에 Collection타입으로 반환하고, 키(key)는 중복을 허용하지 않기 때문에 Set타입으로 반환한다.

## Map.Entry 인터페이스

- Map.Entry인터페이스는 Map인터페이스의 내부 인터페이스이다.
- 내부 클래스와 같이 인터페이스도 인터페이스 안에 인터페으스를 정의하는 것이 가능하다.
- Map에 저장되는 key-value쌍을 다루기 위해 내부적으로 Entry 인터페이스를 정의해 놓은 것

Map 인터페이스의 소스코드 일부

```
public interface Map {
        ...
        interface Entry {
                Object getKey();
                Object getValue();
                Object setValue(Object value);
                boolean equals(Object o);
                int hashCode();
                ...
        }
}
```

### Map.Entry 인터페이스와 메서드

| 메서드 | 설명 |
| --- | --- |
| boolean equals(Object o) | 동일한 Entry인지 비교한다. |
| Object getKey() | Entry의 key 객체를 반환한다. |
| Object getValue() | Entry의 value객체를 반환한다. |
| int hashCode() | Entry의 해시코드를 반환한다. |
| Object setValue(Object value) | Entry의 value객체를 지정된 객체로 바꾼다. |

## ArrayList

- ArrayList는 컬렉션 프레임워크에서 가장 많이 사용되는 컬렉션 클래스이다.
- ArrayList는 List인터페이스를 구현하기 때문에 데이터의 저장순서가 유지되고 중복을 허용한다는 특징을 갖는다.
- ArrayList는 기존의 Vector를 개선한 것으로 Vector와 구현원리와 기능적인 측면에서 동일하다고 할 수 있다.
  (Vector는 기존에 작성된 소스와의 호환성을 위해서 남겨둔 것이므로 가능하면 Vector보다는 ArrayList를 사용하는 것이 좋다.)
- ArrayList는 Object배열을 이용해서 데이터를 순차적으로 자정한다.
- 첫 번째로 저장한 객체는 Object배열의 0번째 위치에 저장되고 그 다음에 저장하는 객체는 1번째 위치에 저장된다(배열에 순서대로 저장된다.)
- 배열에 더 이상 저장할 공간이 없으면 **보다 큰 새로운 배열을 생성**해서 **기존의 배열에 저장된 내용을 새로운 배열로 복사**한 다음에 저장된다.

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

- ArrayList는 elementData라는 이름의 Object 배열을 멤버변수로 선언하고 있다.
- 선언된 배열의 타입이 모든 객체의 최상위 상위 클래스인 Object이기 때문에 모든 종류의 객체를 담을 수 있다.

### ArrayList의 생성자와 메서드

| 메서드 | 설명 |
| --- | --- |
| ArrayList() | 크기가 10인 ArrayList를 생성 |
| ArrayList(Collection c) | 주어진 컬렉션이 저장된 ArrayList를 생성 |

| 메서드 | 설명 |
| --- | --- |
| ArrayList(int initialCapacity) | 지정된 초기용량을 갖는 ArrayList를 생성 |
| boolean add(Object o) | ArrayList의 마지막 객체를 추가, 성공하면 true |
| void add(int index, Object element) | 지정된 위치(index)에 객체를 저장 |
| boolean addAll(Collection c) | 주어진 컬렉션의 모든 객체를 저장한다. |
| boolean addAll(int index, Collection c) | 지정된 위치부터 주어진 컬렉션의 모든 객체를 저장한다. |
| void clear() | ArrayList를 완전히 비운다. |
| Object clone() | ArrayList를 복제한다. |
| boolean contains(Object c) | 지정된 객체(o)가 ArrayList에 포함되어 있는지 확인 |
| void ensureCapacity(int minCapacity) | ArrayList의 용량이 최소한 minCapacity가 되도록 한다. |
| Object get(int index) | 지정된 위치(index)에 저장된 객체를 반환한다. |
| int indexOf(Object o) | 지정된 객체가 저장된 위치를 찾아 반환한다. |
| boolean isEmpty() | ArrayList가 비어있는지 확인한다. |
| Iterator iterator() | ArrayList의 Iterator 객체를 반환 |
| int lastIndexOf(Object o) | 객체(o)가 저장된 위치를 끝부터 역방향으로 검색해서 반환 |
| ListIterator listIterator() | ArrayList의 ListIterator를 반환 |
| ListIterator listIterator(int index) | ArrayList의 지정된 위치부터 시작하는 ListIterator를 반환 |
| Object remove(int index) | 지정된 위치(index)에 있는 객체를 제거한다. |
| boolean remove(Object o) | 지정된 객체를 제거한다.(성공하면 true, 실패하면 false) |
| boolean removeAll(Collection c) | 지정된 컬렉션에 저장된 것과 동일한 객체들을 ArrayList에서 제거한다. |
| boolean retainAll(Collection c) | ArrayList에 저장된 객체중 주어진 컬렉션과 공통인 것만 남기고 나머지는 삭제한다. |
| Object set(int index, Object element) | 주어진 객체(element)를 지정된 위치(index)에 저장한다. |
| int size() | ArrayList에 저장된 객체의 개수를 반환한다. |
| void sort(Comparator c) | 지정된 정렬기준(c)으로 ArrayList를 정렬 |
| List subList(int fromIndex, int toIndex) | fromIndex부터 toIndex사이에 저장된 객체를 반환한다. |

| 메서드 | 설명 |
| --- | --- |
| Object[] toArray() | ArrayList에 저장된 모든 객체들을 객체배열로 반환한다. |
| Object[] toArray(Object[] a) | ArrayList에 저장한 모든 객체들을 객체배열 a에 담아 반환한다. |
| void trimToSize() | 용량을 크기에 맞게 줄인다.(빈 공간을 없앤다.) |

**day13_14/ArrayListEx1.java**

```
package day13_14;

import java.util.*;

public class ArrayListEx1 {
	public static void main(String[] args) {
		ArrayList list1 = new ArrayList(10);
		list1.add(Integer.valueOf(5));
		list1.add(Integer.valueOf(4));
		list1.add(Integer.valueOf(2));
		list1.add(Integer.valueOf(0));
		list1.add(Integer.valueOf(1));
		list1.add(Integer.valueOf(3));

		ArrayList list2 = new ArrayList(list1.subList(1, 4));
		print(list1, list2);

		Collections.sort(list1);
		Collections.sort(list2);
		print(list1, list2);

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

		list2.add("B");
		list2.add("C");
		list2.add(3, "A");
		print(list1, list2);

		list2.set(3, "AA");
		print(list1, list2);

		// list1과 list2와 겹치는 부분만 남기ㄷ고 나머지는 삭제한다.
		System.out.println("list1.retainAll(list2):" +
list1.retainAll(list2));

		print(list1, list2);


		// list2에서 list1에 포함된 객체들을 삭제한다.
		for(int i = list2.size() - 1; i >=0; i--) {
			if (list1.contains(list2.get(i)))
				list2.remove(i);
		}
```

```
                print(list1, list2);
        }

        static void print(ArrayList list1, ArrayList list2) {
                System.out.println("list1:" + list1);
                System.out.println("list2:" + list2);
                System.out.println();
        }
}
```

실행결과

```
list1:[5, 4, 2, 0, 1, 3]
list2:[4, 2, 0]

list1:[0, 1, 2, 3, 4, 5]
list2:[0, 2, 4]

list1.containsAll(list2):true
list1:[0, 1, 2, 3, 4, 5]
list2:[0, 2, 4, A, B, C]

list1:[0, 1, 2, 3, 4, 5]
list2:[0, 2, 4, AA, B, C]

list1.retainAll(list2):true
list1:[0, 2, 4]
list2:[0, 2, 4, AA, B, C]

list1:[0, 2, 4]
list2:[AA, B, C]
```

- list2의 각 요소를 접근하기 위해 get(int index)메서드와 for문을 사용하였는데, for문의 변수 i를 0부터 증가시킨 것이 아니라 **list2.size() - 1** 부터 감소시키면서 거꾸로 반복 시켰다.
- 변수 i를 증가시켜가면서 삭제하면, 한 요소가 삭제될 때마다 빈 공간을 채우기 위해 나머지 요소들이 자리이동을 하기 때문에 올바른 결과를 얻을 수 없다.
- 그래서 제어변수를 감소시켜가면서 삭제를 해야 자리이동이 발생해도 영향을 받지 않고 작업이 가능하다.

**day13_14/ArrayListEx2.java**

```
package day13_14;

import java.util.*;

public class ArrayListEx2 {
        public static void main(String[] args) {
                final int LIMIT = 10;
                String source = "0123456789abcdefghijABCDEFGHIJ!@#$%^&*()ZZZ";
                int length = source.length();

                List list = new ArrayList(length / LIMIT + 10); // 크기를 약간 여유
있게 잡는다.
```

```
                for(int i = 0; i < length; i+=LIMIT) {
                        if (i+LIMIT < length) {
                                list.add(source.substring(i, i+LIMIT));
                        } else {
                                list.add(source.substring(i));
                        }
                }

                for (int i = 0; i < list.size(); i++) {
                        System.out.println(list.get(i));
                }
        }
}
```

실행결과

```
0123456789
abcdefghij
ABCDEFGHIJ
!@#$%^&*()
ZZZ
```

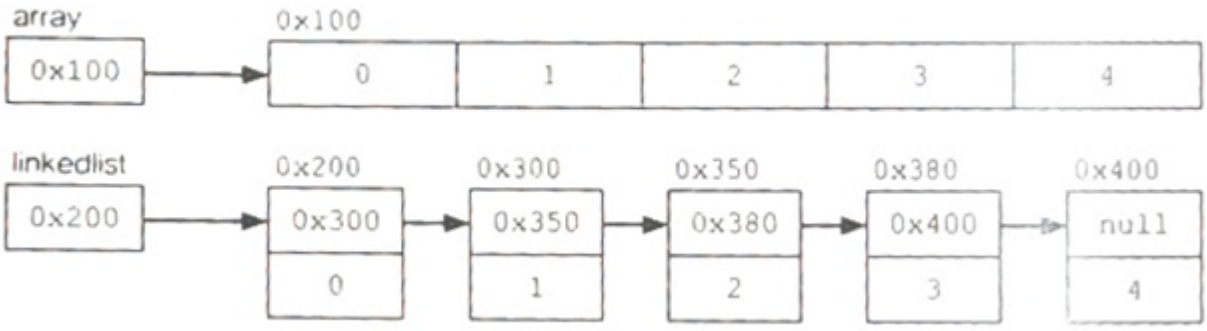
- ArrayList를 생성할 때, 저장할 요소의 개수를 고려해서 실제 저장할 개수보다 약간 여유 있는 크기로 하는것이 좋다.
- 생설할 때 지정한 크기보다 더 많은 객체를 저장하면 자동적으로 크기가 늘어나기는 하지만 이 과정에서 처리시간이 많이 소요되기 때문이다.

> ArrayList나 Vector같이 배열을 이용한 자료구조는 데이터를 읽어오고 저장하는 데는 효율이 좋지만, 용량을 변경해야 할 때는 새로운 배열을 생성한 후 기존의 배열로 부터 새로 생성된 배열로 데이터를 복사해야하기 때문에 상당히 효율이 떨어진다는 단점을 가지고 있다. 따라서 처음에 인스턴스를 생성할 때, 저장한 데이터의 개수를 잘 고려하여 충분한 용량의 인스턴스를 생성하는 것이 좋다.

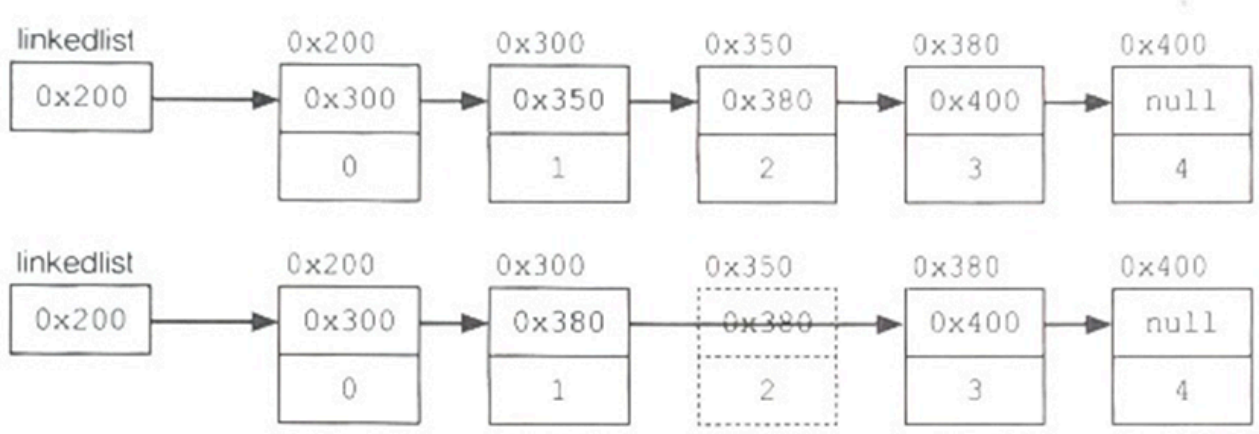
## LinkedList

- 배열은 가장 기본적인 형태의 자료구조로 구조가 간단하며 사용하기 쉽고 데이터를 읽어오는 데 걸리는 시간(접근 시간, access time)이 가장 빠르다라는 장점을 가지고 있다.
- 그러나 다음과 같은 단점도 가지고 있다.
  - 크기를 변경할 수 없다.
    - 크기를 변경할 수 없으므로 새로운 배열을 생성해서 데이터를 복사해야 한다.
    - 실행속도를 향상시키기 위해서는 충분히 큰 크기의 배열을 생성해야 하므로 메모리가 낭비된다.
  - 비순차적인 데이터의 추가 또는 삭제에 시간이 많이 걸린다.
    - 차례대로 데이터를 추가하고 마지막에서부터 데이터를 삭제하는 것은 빠르지만
    - 배열의 중간에 데이터를 추가하려면, 빈자리를 만들기 위해 다른 데이터를 복사해서 이동해야 한다.

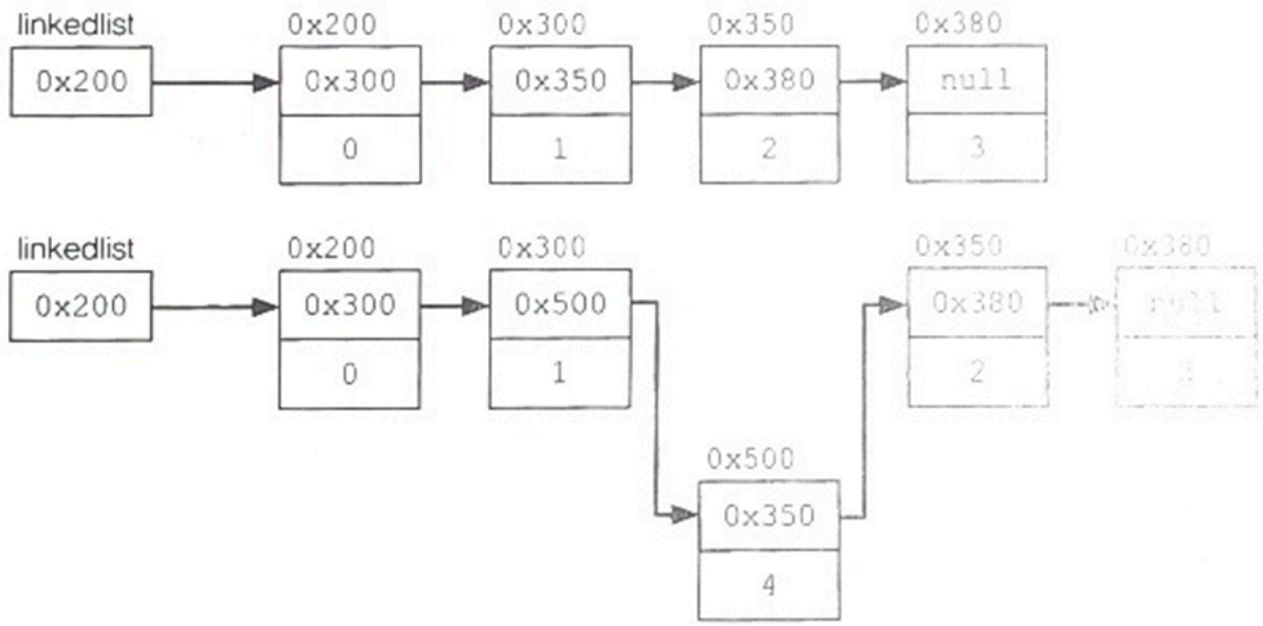
- 이러한 배열의 단점을 보완하기 위해서 링크드 리스트(linked list)라는 자료구조가 고안되었다.
- 배열은 모든 데이터가 연속적으로 존재하지만 링크드 리스트는 불연속적으로 존재하는 데이터를 서로 연결(link)한 형태로 구성되어 있다.



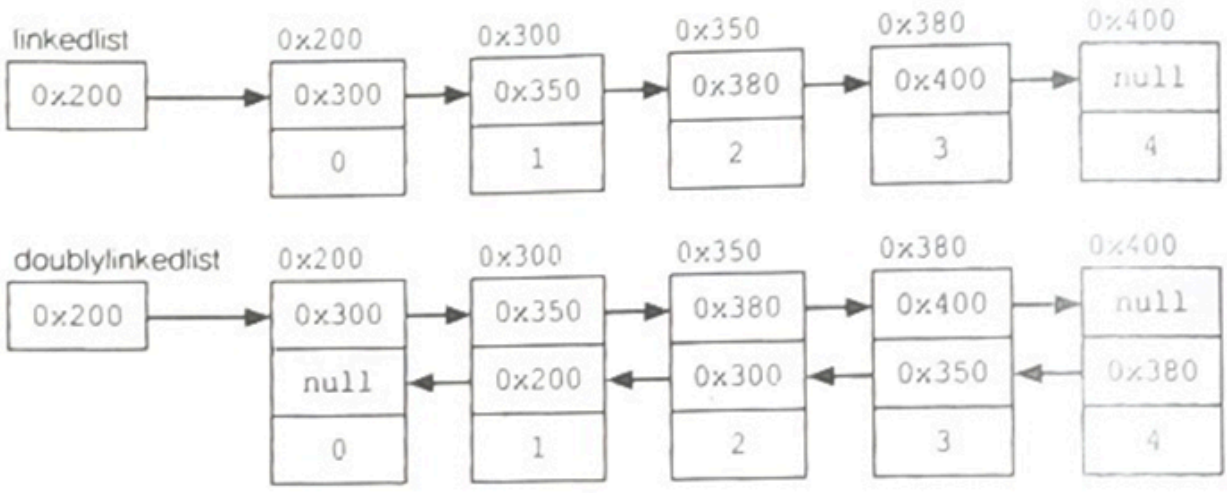

- 상기 그림에서 알수 있는 것 처럼, 링크드 리스트의 각 요소들은 자신과 연결된 다음 요소에 대한 참조(주소값)와 데이터로 구성되어 있다.
- 링크드 리스트에서 삭제는 삭제하고자 하는 요소의 이전 요소가 삭제하고자 하는 요소의 다음 요소를 참조하도록 변경하기만 하면 된다. 단 하나의 참조만 변경하면 삭제가 이루어지므로 배열처럼 데이터를 이동하기 위해 복사하는 과정이 없기 때문에 처리속도가 매우 빠르다.



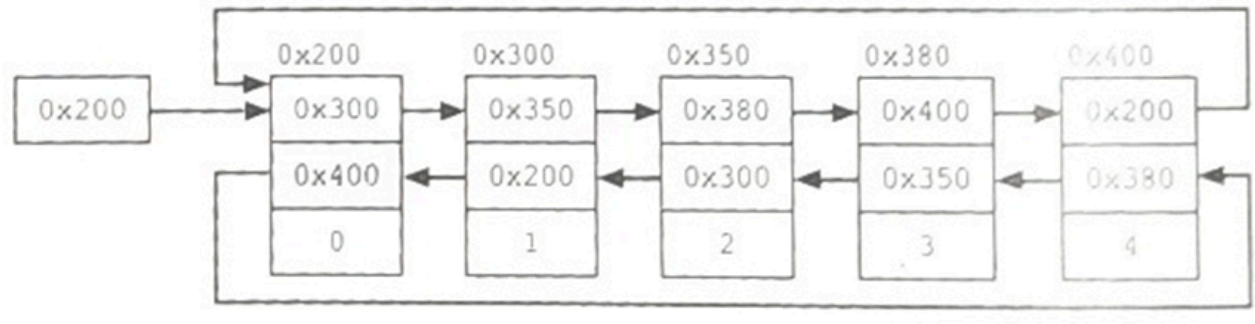

- 새로운 데이터를 추가할 때는 새로운 요소를 생성한 다음 추가하고자 하는 위치의 이전 요소의 참조를 새로운 요소에 대한 참조로 변경해주고, 새로운 요소가 그 다음 요소를 참조하도록 변경하기만 하면 되므로 처리 속도가 매우 빠르다.



- 링크드 리스트는 이동방향이 단방향이기 때문에 다음 요소에 대한 접근은 쉽지만 이전요소에 대한 접근은 어렵다. 이 덤을 보완한 것이 더블 링크드 리스트(이중 연결리스트, doubly linked list)이다.
- 더블 링크드 리스트는 단순히 링크드 리스트에 참조변수를 하나 더 추가하여 다음 요소에 대한 참조뿐 아니라 이전 요소에 대한 참조가 가능하도록 했을 뿐, 링크드 리스트와 같다.




- 더블 링크드 리스트의 접근성을 보다 향상시킨 것이 '더블 써큘러 링크드 리스트(이중 원형 연결리스트, doubly circular linked list)'이다. 이는 단순히 더블 링크드 리스트의 첫 번째 요소와 마지막 요소를 서로 연결시킨 것이다.
- 이렇게 하면 마지막요소의 다음요소가 첫번째 요소가 되고, 첫 번째 요소의 이전 요소가 마지막 요소가 된다.




- 실제로 구현된 LinkedList 클래스는 이름과 달리 '링크드 리스트'가 아닌 **더블 링크드 리스트**로 구현되어 있다.

## LinkedList의 생성자와 메서드

| 생성자 또는 메서드 | 설명 |
|---|---|
| LinkedList() | LinkeList 객체를 생성 |
| LinkList(Collection c) | 주어진 컬렉션을 포함하는 LinkedList객체를 생성 |
| boolean add(Object o) | 지정된 객체(o)를 LinkedList의 끝에 추가, 저장에 성공하면 true, 실패하면 false |
| void add(int index, Object element) | 지정된 위치(index)에 객체(element)를 추가 |

| 생성자 또는 메서드 | 설명 |
| --- | --- |
| boolean addAll(Collection c) | 주저진 컬렉션에 포함된 모든 요소를 LinkedList의 끝에 추가한다. 성공하면 true, 실패하면 false) |
| void clear() | LinkedList의 모든 요소를 삭제 |
| boolean contains(Object o) | 지정된 객체가 LinkedList에 포함되어 있는지 알려줌 |
| boolean containsAll(Collection c) | 지정된 컬렉션의 모든 요소가 포함되어 있는지 알려줌 |
| Object get(int index) | 지정된 위치(index)의 객체를 반환 |
| int indexOf(Object o) | 지정된 객체가 저장된 위치(앞에서 몇 번째)를 반환 |
| boolean isEmpty() | LinkedList가 비어 있는지 알려 준다. 비어있으면 true |
| Iterator iterator() | Iterator를 반환한다. |
| int lastIndexOf(Object o) | 지정된 객체의 위치(index)를 반환(끝부터 역순검색) |
| ListIterator listIterator() | ListIterator를 반환한다. |
| ListIterator listIterator(int index) | 지정된 위치에서부터 시작하는 ListIterator를 반환 |
| Object remove(int index) | 지정된 위치(index)의 객체를 LinkedList에서 제거 |
| booleam remove(Object o) | 지정된 객체를 LinkedList에서 제거, 성공하면 true, 실패하면 false |
| boolean removeAll(Collection c) | 지정된 컬렉션의 요소와 일치하는 요소를 모두 삭제 |
| boolean retainAll(Collection c) | 지정된 컬렉션의 모든 요소가 포함되어 있는지 확인 |
| Object set(int index, Object element) | 지정된 위치(index)의 객체를 주어진 객체로 바꿈 |
| int size() | LinkedList에 저장된 객체의 수를 반환 |
| List subList(int fromIndex, int toIndex) | LinkedList의 일부를 List로 반환 |
| Object[] toArray() | LinkedList에 저장된 객체를 배열로 반환 |
| Object[] toArray(Object[] a) | LinkedList에 저장된 객체를 주어진 배열에 저장하여 반환 |
| Object element() | LinkedList의 첫 번째 요소를 반환 |
| boolean offer(Object o) | 지정된 객체(o)를 LinkedList의 끝에 추가, 성공하면 true, 실패하면 false |
| Object peek() | LinkedList의 첫 번째 요소를 반환 |
| Object poll() | LinkedList의 첫 번째 요소를 반환, LinkedList에서는 제거된다. |

| 생성자 또는 메서드 | 설명 |
|---|---|
| Object remove() | LinkedList의 첫 번째 요소를 제거 |
| void addFirst(Object o) | LinkeList의 맨 앞에 객체(o)를 추가 |
| void addLast(Object o) | LinkedList의 맨 끝에 객체(o)를 추가 |
| Iterator descendingIterator() | 역순으로 조회하기 위한 DescendingIterator를 반환 |
| Object getFirst() | LinkedList의 첫번째 요소를 반환 |
| Object getLast() | LinkedList의 마지막 요소를 반환 |
| boolean offerFirst(Object o) | LinkedList의 맨 앞에 객체(o)를 추가, 성공하면 true |
| boolean offerLast(Object o) | LinkedList의 맨 끝에 객체(o)를 추가, 성공하면 true |
| Object peekFirst() | LinkedList의 첫번째 요소를 반환 |
| Object peekLast() | LinkedList의 마지막 요소를 반환 |
| Object pollFirst() | LinkedList의 첫번째 요소를 반환하면서 제거 |
| Object pollLast() | LinkedList의 마지막 요소를 반환하면서 제거 |
| Object pop() | removeFirst()와 동일 |
| void push(Object o) | addFirst()와 동일 |
| Object removeFirst() | LinkedList의 첫번째 요소를 제거 |
| Object removeLast() | LinkedList의 마지막 묘소를 제거 |
| boolean removeFirstOccurence(Object o) | LinkedList에서 첫번째로 일치하는 객체를 제거 |
| boolean removeLastOccurence(Object o) | LinkedList에서 마지막으로 일치하는 객체를 제거 |

> LinkedList는 Queue인터페이스(JDK1.5)와 Deque인터페이스(JDK1.6)를 구현하도록 변경되었는데, 마지막 22개 메서드는 Queue인터페이스와 Deque 인터페이스를 구현하면서 추가된 것이다.

LinkedList 역시 List 인터페이스를 구현했기 때문에 ArrayList의 내부구현방법만 다를 뿐 제공하는 메서드의 종류와 기능은 거의 같다.

**day13_14/ArrayListLinkedListTest.java**

```
package day13_14;

import java.util.*;

public class ArrayListLinkedListTest {
	public static void main(String[] args) {
		// 추가할 데이터의 개수를 고려하여 충분히 잡아야한다.
```

```
        ArrayList al = new ArrayList(2000000);
        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 t삭제하기 =");
        System.out.println("ArrayList : " + remove1(al));
        System.out.println("LinkedList : " + remove1(ll));
        System.out.println();
}

public static long add1(List list) {
        long start = System.currentTimeMillis();

        for(int i = 0; i < 1000000; 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 < 10000; 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 < 10000; i++) {
                        list.remove(i);
                }

                long end = System.currentTimeMillis();
                return end - start;
        }
}

실행결과
= 순차적으로 추가하기 =
ArrayList : 256
LinkedList : 362

= 중간에 추가하기 =
ArrayList : 3760
LinkedList : 24

= 중간에서 삭제하기 =
ArrayList : 3143
LinkedList : 244

= 순차적으로 t삭제하기 =
ArrayList : 21
LinkedList : 61
```

### day13_14/ArrayListLinkedListTest2.java

```
package day13_14;

import java.util.*;

public class ArrayListLinkedListTest2 {
        public static void main(String[] args) {
                ArrayList al = new ArrayList(1000000);
                LinkedList ll = new LinkedList();
                add(al);
                add(ll);

                System.out.println("= 접근시간 테스트 =");
                System.out.println("ArrayList : " + access(al));
                System.out.println("LinkedList : " + access(ll));
        }

        public static void add(List list) {
                for(int i = 0; i < 100000; i++) {
                        list.add(i + "");
                }
        }

        public static long access(List list) {
```

```
                long start = System.currentTimeMillis();

                for (int i = 0; i < 10000; i++) {
                        list.get(i);
                }

                long end = System.currentTimeMillis();
                return end - start;
        }
}
```

```
실행결과
= 접근시간 테스트 =
ArrayList : 1
LinkedList : 241
```

- LinkedList는 불연속적으로 위치한 각 요소들이 서로 연결된 것이라 처음부터 n번째 데이터까지 차례대로 따라가야만 원하는 값을 얻을 수 있다.
- 따라서 LinkedList는 저장해야하는 데이터의 개수가 많아질수록 데이터를 읽어 오는 시간, 즉 접근시간(access time)이 길어진다는 단점이 있다.

**ArrayList와 LinkedList 비교**

| 컬렉션 | 읽기(접근시간) | 추가/삭제 | 비고 |
|---|---|---|---|
| ArrayList | 빠르다 | 느리다 | 순차적인 추가삭제는 더 빠름.<br>비효율적인 메모리사용 |
| LinkedList | 느리다 | 빠르다 | 데이터가 많을수록 접근성이 떨어진다. |

# Stack과 Queue

## Stack

- 스택(Stack)은 마지막에 저장한 데이터를 가장 먼저 꺼내게 되는 LIFO(Last In First Out)구조로 되어 있다.
- 스택(Stack)은 동전 통과 같은 구조로 양 옆과 바닥이 막혀 있어 한 방향으로만 뺄수 있는 구조이다.
- 스택(Stack)에 0, 1, 2의 순서로 데이터를 넣으면 꺼낼 때는 2, 1, 0의 순서로 꺼내게 된다. 즉 넣은 순서와 꺼낸 순서가 바뀐다.
- 스택에는 ArrayList와 간은 배열기반의 컬렉션 클래스가 적합하다.

**Stack의 메서드**

| 메서드 | 설명 |
|---|---|
| boolean empty() | Stack이 비어있는지 알려준다. |
| Object peek() | Stack의 맨 위에 저장된 객체를 반환, pop()과 달리 Stack에서 객체를 꺼 |

| 메서드 | 설명 |
| --- | --- |
| | 내지는 않음.(비었을 때는 EmptyStackException이 발생) |
| Object pop() | Stack의 맨 위에 저장된 객체를 꺼낸다.(비었을 때는 EmptyStackException발생) |
| Object push(Object item) | Stack에 객체(item)를 저장한다. |
| int search(Object o) | Stack에서 주어진 객체(o)를 찾아서 그 위치를 반환, 못찾으면 -1을 반환한다.(배열과 달리 위치는 0이 아닌 1부터 시작) |

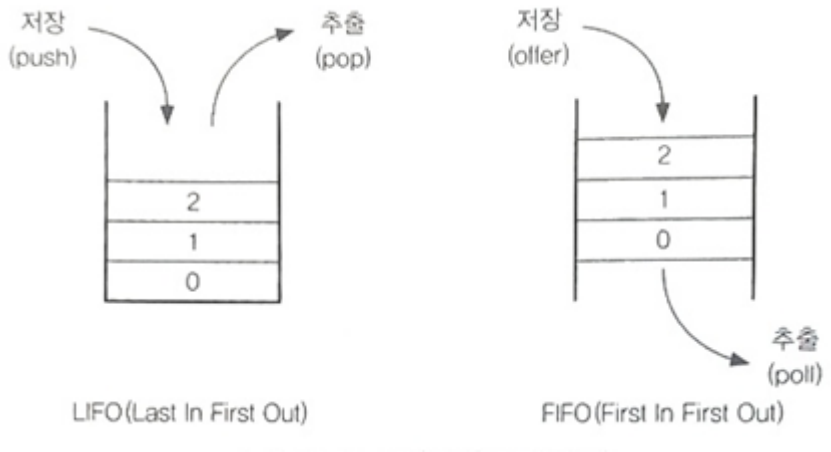
## Queue

- 큐(Queue)는 처음에 저장한 데이터를 가장 먼저 꺼내게 되는 FIFO(First In First Out)구조로 되어 있다.
- 큐(Queue)는 양 옆만 막혀 있고, 위 아래로 뚫려 있어서 한 방향으로는 넣고 한 방향으로는 빼는 파이프와 같은 구조로 되어 있다.
- 큐(Queue)에 0, 1, 2의 순서로 데이터를 넣었다면 꺼낼 때 역시 0, 1, 2 순서로 꺼내게 된다. 순서의 변경 없이 먼저 넣은 것을 먼저 꺼내게 된다.
- 큐는 ArrayList보다는 데이터의 추가/삭제가 쉬운 LinkedList로 구현하는 것이 더 적합하다.

> ArrayList와 같은 배열 기반의 컬렉션 클래스를 사용한다면 데이터를 꺼낼 때마다 빈 공간을 채우기 위해 데이터의 복사가 발생하므로 비효율적이다.

**Queue의 메서드**

| 메서드 | 설명 |
| --- | --- |
| boolean add(Object o) | 지정된 객체를 Queue에 추가한다. 성공하면 true를 반환, 저장공간이 부족하면 IllegalStateException발생 |
| Object remove() | Queue에서 객체를 꺼내 반환, 비어있으면 NoSuchElementException 발생 |
| Object element() | 삭제없이 요소를 읽어온다. peek과 달리 Queue가 비었을 때, NoSuchElementException발생 |
| boolean offer(Object o) | Queue에 객체를 저장, 성공하면 true, 실패하면 false를 반환 |
| Object poll() | Queue에서 객체를 꺼내서 반환, 비어있으면 null을 반환 |
| Object peek() | 삭제없이 요소를 읽어온다. Queue가 비어있으면 null을 반환 |




### day13_14/StackQueueEx.java

```
package day13_14;

import java.util.*;

public class StackQueueEx {
	public static void main(String[] args) {
		Stack st = new Stack();
		Queue q = new LinkedList(); // Queue 인터페이스를 구현한 LinkedList
를 사용

		st.push("0");
		st.push("1");
		st.push("2");

		q.offer("0");
		q.offer("1");
		q.offer("2");

		System.out.println("= Stack =");
		while(!st.empty()) {
			System.out.println(st.pop());
		}

		System.out.println("= Queue =");
		while(!q.isEmpty()) {
			System.out.println(q.poll());
		}
	}
}

실행결과
= Stack =
2
1
0
= Queue =
0
1
2
```

## PriorityQueue

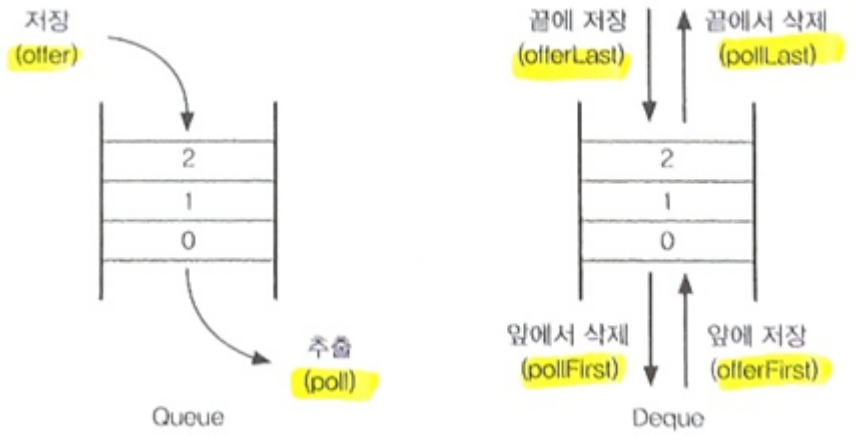
- Queue 인터페이스의 구현체 중에 하나로, 저장한 순서에 관계없이 우선순위(Priority)가 높은 것부터 꺼내게 된다는 특징이 있다.
- 우선순위는 클래스에 지정된 Comparable 인터페이스의 compareTo에 의해 결정된다.

**day13_14/PriorityQueueEx.java**

```
package day13_14;

import java.util.*;

public class PriorityQueueEx {
	public static void main(String[] args) {
		Queue pq = new PriorityQueue();
		pq.offer(3);
		pq.offer(1);
		pq.offer(5);
		pq.offer(2);
		pq.offer(4);

		System.out.println(pq);

		Object obj = null;

		// PriorityQueue에 저장한 요소를 하나씩 꺼낸다.
		while((obj = pq.poll()) != null) {
			System.out.println(obj);
		}
	}
}

실행결과
[1, 2, 5, 3, 4]
1
2
3
4
5
```

## Deque(Double-Ended Queue)

- Queue의 변형으로 한 쪽 끝으로만 추가/삭제할 수 있는 Queue와 달리 Deque는 양쪽 끝에 추가/삭제가 가능하다.
- Deque의 상위 인터페이스는 Queue이며, 구현체로는 ArrayDeque와 LinkedList등이 있다.
- Deque는 Stack과 Queue를 합쳐놓은 것과 같으며 Stack으로 사용할 수도 있고, Queue로 사용할 수도 있다.




## Iterator, ListIterator, Enumeration

- 컬렉션에 저장된 요소를 접근하는데 사용되는 인터페이스이다.
- Enumeration은 Iterator의 구버전이고, ListIterator는 Iterator의 기능을 향상 시킨 것이다.

### Iterator

- 컬렉션 프레임워크에서는 컬렉션에 저장된 요소들을 읽어오는 방법을 표준화하였다.
- 컬렉션에 저장된 각 요소에 접근하는 기능을 가진 Iterator인터페이스를 정의하고, Collection 인터페이스에는 Iterator(Iterator를 구현한 클래스의 인스턴스)를 반환하는 iterator()를 정의하고 있다.
- iterator()는 Collection인터페이스에 정의된 메서드이므로 Collection 인터페이스의 하위 인터페이스인 List와 Set에도 포함되어 있으며, 각 컬렉션의 특징에 알맞게 작성되어 있다.
- 컬렉션 클래스에 대해 iterator()를 호출하여 Iterator를 얻은 다음 반복문, 주로 while문을 사용해서 컬렉션 클래스의 요소를 읽어 올 수 있다.

#### Iterator 인터페이스 메서드

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

```
List list = new ArrayList();
Iterator it = list.iterator();

while(it.hasNext()) {
        System.out.println(it.next());
}
```

- Map 인터페이스를 구현한 컬렉션 클래스는 키(key)와 값(value)을 쌍(pair)으로 저장하고 있기 때문에 iterator()를 직접 호출할 수 없다.
- 그 대신 keySet()이나 entrySet()과 같은 메서드를 통해서 키와 값을 각각 따로 Set의 형태로 얻어 온 후에 다시 iterator()를 호출해야 Iterator를 호출해야 Iterator를 얻을 수 있다.

```
Map map = new HashMap();
...
Iterator it = map.keySet().iterator();
```

> Iterator list = map.entrySet().iterator();은 아래 두 문장을 하나로 합친 것으로 이해하면 된다.
> Set eSet = map.entrySet();
> Iterator list = eSet.iterator();

**day13_14/IteratorEx1.java**

```
package day13_14;

import java.util.*;

public class IteratorEx1 {
	public static void main(String[] args) {
		ArrayList list = new ArrayList();
		list.add("1");
		list.add("2");
		list.add("3");
		list.add("4");
		list.add("5");

		Iterator it = list.iterator();

		while(it.hasNext()) {
			Object obj = it.next();
			System.out.println(obj);
		}
	}
}
```

실행결과
1
2
3
4
5

## ListIterator와 Enumeration

### Enumeration

- 컬렉션 프레임워크가 만들어지기 이전에 사용하던 것으로 Iterator의 구버전이다.
- 이전 버전으로 작성된 소스와의 호환을 위해 남겨둔 것

**Enumeration 인터페이스의 메서드**

| 메서드 | 설명 |
|---|---|
| boolean hasMoreElements() | 읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다. Iterator의 hasNext()와 같다. |
| Object nextElement() | 다음 요소를 읽어 온다. nextElement()를 호출하기 전에 hasMoreElements()를 호출해서 읽어올 요소가 남아있는지 확인하는 것이 안전한다. Iterator의 next()와 같다. |

**ListIterator**

- Iterator를 상속받아서 기능을 추가한 것
- Iterator는 단방향으로만 이동할 수 있는데 반해 ListIterator는 양방향으로의 이동이 가능하다.
- ArrayList, LinkedList와 같이 List인터페이스를 구현한 컬렉션에서만 사용할 수 있다.

**ListIterator의 메서드**

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

**day13_14/ListIteratorEx1.java**

```
package day13_14;

import java.util.*;

public class ListIteratorEx1 {
	public static void main(String[] args) {
		ArrayList list = new ArrayList();
		list.add("1");
		list.add("2");
		list.add("3");
		list.add("4");
		list.add("5");

		ListIterator it = list.listIterator();

		while(it.hasNext()) {
			System.out.print(it.next()); // 순방향으로 진행하면서 읽어온
다.
		}
		System.out.println();

		while(it.hasPrevious()) {
			System.out.print(it.previous()); // 역방향으로 진행하면서 읽
어온다.
		}
		System.out.println();
	}
}

실행결과
12345
54321
```

### day13_14/ListIteratorEx2.java

```
package day13_14;

import java.util.*;

public class ListIteratorEx2 {
	public static void main(String[] args) {
		ArrayList original = new ArrayList(10);
		ArrayList copy1 = new ArrayList(10);
		ArrayList copy2 = new ArrayList(10);

		for(int i = 0; i < 10; i++) {
			original.add(i+"");
		}

		Iterator it = original.iterator();

		while(it.hasNext()) {
			copy1.add(it.next());
		}
```

```
                System.out.println("= Original에서 copy1로 복사(copy) =");
                System.out.println("original:" + original);
                System.out.println("copy1:" + copy1);
                System.out.println();

                it = original.iterator(); // Iterator는 재사용이 안되므로, 다시 얻어
와야 한다.
                while(it.hasNext()) {
                        copy2.add(it.next());
                        it.remove();
                }

                System.out.println("= Original에서 copy2로 이동(copy) =");
                System.out.println("original:" + original);
                System.out.println("copy2:" + copy2);
        }
}
```

실행결과

```
= Original에서 copy1로 복사(copy) =
original:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
copy1:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

= Original에서 copy2로 이동(copy) =
original:[]
copy2:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## Arrays

Arrays클래스에는 배열을 다루는데 유용한 메서드가 정의되어 있다.

### 배열의 복사 - copyOf(), copyOfRange()

copyOf()는 배열 전체를, copyOfRange()는 배열의 일부를 복사해서 새로운 배열을 만들어 반환한다. 지정된 범위의 끝은 포함되지 않는다.

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

### 배열 채우기 - fill(), setAll()

- fill() : 배열의 모든 요소를 지정된 값으로 채운다.
- setAll() : 배열을 채우는데 사용할 함수형 인터페이스를 매개변수로 받는다.

```
int[] arr = new int[5];
Arrays.fill(arr, 9); // arr = [9,9,9,9,9]
```

```
Arrays.setAll(arr, i -> (int)(Math.random() * 5) + 1); // arr = [1,5,2,1,1]  1~5 사이의 랜덤 숫자
```

> **i -> (int)(Math.random() * 5) + 1**은 람다식(lambda expression)이다. 람다식 참고

## 배열의 정렬과 검색 - sort(), binarySearch()

- sort() : 배열을 정렬할 때 사용
- binarySearch() : 배열에 저장된 요소를 검색할 때 사용
- binarySearch()는 배열에서 지정된 값이 저장된 위치(index)를 찾아서 반환하는데, 반드시 배열이 정렬된 상태이어야 올바른 결과를 얻는다.
- binarySearch()는 검색한 값과 일치하는 요소들이 여러 개 있다면, 이 중에서 어떤 것의 위치가 반환될지는 알 수 없다.

```
int[] arr = { 3, 2, 0, 1, 4 };
int idx = Arrays.binarySearch(arr, 2); // idx = -5 - 잘못된 결과

Arrays.sort(arr); // 배열 arr을 정렬한다.
System.out.println(Arrays.toString(arr)); // [0,1,2,3,4]
int idx = Arrays.binarySearch(arr, 2); // idx = 2  - 올바른 결과
```

## 문자열 비교와 출력 - equals(), toString()

- toString() : 배열의 모든 요소를 문자열로 편하게 출력할 수 있다. 일차원 배열에서만 사용할 수 있다.
- deepToString() : 배열의 모든 요소를 재귀적으로 접근해서 문자열을 구성하므로 다차원 배열에서도 사용할 수 있다.

```
int[] arr = {0,1,2,3,4};
int[][] arr2D = {{11,22}, {21,22}};

System.out.println(Arrays.toString(arr)); // [0,1,2,3,4]
System.out.println(Arrays.deepToString(arr2D)); // [[11,12], [21,22]]
```

- equals() : 두 배열에 저장된 모든 요소를 비교해서 같으면 true, 다르면 false를 반환한다. 일차원 배열에서만 사용가능하다.
- deepToEquals() : 다차원 배열에서도 사용할 수 있다.

```
String[][] str2D = new String[][]{{"aaa", "bbb"}, {"AAA", "BBB"}};
String[][] str2D2 = new String[][]{{"aaa", "bbb"}, {"AAA", "BBB"}};

System.out.println(Arrays.equals(str2D, str2D2)); // false
System.out.println(Arrays.deepEquals(str2D, str2D2)); // true
```

## 배열을 List로 변환 - asList(Object... a)

- asList()는 배열을 List에 담아서 반환합니다.
- 매개변수 타입이 가변인수이므로 배열 생성 없이 저장할 요소들만 나열하는 것도 가능하다.

```
List list = Arrays.asList(new Integer[]{1,2,3,4,5});
List list = Arrays.asList(1,2,3,4,5);
```

- asList()가 반환한 List의 크기를 변경할 수 없다(추가 삭제가 불가능)
- 추가 삭제가 필요하다면 다음과 같이 새로운 List를 생성하면 된다.

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

## stream()

stream()은 컬렉션을 스트림으로 변환합니다. 스트림 참고

**day13_14/ArraysEx.java**

```
package day13_14;

import java.util.*;

public class ArraysEx {
	public static void main(String[] args) {
		int[] arr = {0,1,2,3,4};
		int[][] arr2D = {{11,12,13}, {21,22,23}};

		System.out.println("arr=" + Arrays.toString(arr));
		System.out.println("arr2D=" + Arrays.deepToString(arr2D));

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

		System.out.println("arr2=" + Arrays.toString(arr2));
		System.out.println("arr3=" + Arrays.toString(arr3));
		System.out.println("arr4=" + Arrays.toString(arr4));
		System.out.println("arr5=" + Arrays.toString(arr5));
		System.out.println("arr6=" + Arrays.toString(arr6));

		int[] arr7 = new int[5];
		Arrays.fill(arr7, 9);
		System.out.println("arr7=" + Arrays.toString(arr7));

		Arrays.setAll(arr7, i -> (int)(Math.random()*6) + 1);
		System.out.println("arr7=" + Arrays.toString(arr7));

		for(int i : arr7) {
			char[] graph = new char[i];
			Arrays.fill(graph, '*');
```

```
                        System.out.println(new String(graph) + i);
                }

                String[][] str2D = new String[][] {{"aaa","bbb"},{"AAA", "BBB"}};
                String[][] str2D2 = new String[][] {{"aaa","bbb"},{"AAA", "BBB"}};

                System.out.println(Arrays.equals(str2D, str2D2)); // false
                System.out.println(Arrays.deepEquals(str2D, str2D2)); // true

                char[] chArr = {'A', 'D', 'C', 'B', 'E'};

                System.out.println("chArr=" + Arrays.toString(chArr));
                System.out.println("index of B =" + Arrays.binarySearch(chArr,
'B'));
                System.out.println("= After sorting =");
                Arrays.sort(chArr);
                System.out.println("chArr="+Arrays.toString(chArr));
                System.out.println("index of B =" + Arrays.binarySearch(chArr,
'B'));
        }
}
```

실행결과

```
arr=[0, 1, 2, 3, 4]
arr2D=[[11, 12, 13], [21, 22, 23]]
arr2=[0, 1, 2, 3, 4]
arr3=[0, 1, 2]
arr4=[0, 1, 2, 3, 4, 0, 0]
arr5=[2, 3]
arr6=[0, 1, 2, 3, 4, 0, 0]
arr7=[9, 9, 9, 9, 9]
arr7=[2, 4, 5, 4, 1]
**2
****4
*****5
****4
*1
false
true
chArr=[A, D, C, B, E]
index of B =-2
= After sorting =
chArr=[A, B, C, D, E]
index of B =1
```

## Comparator와 Comparable

- Arrays.sort()를 호출만 하면 컴퓨터가 알아서 배열을 정렬하는 것처럼 보이지만, 사실은 Character클래스의 Comparable의 구현에 의해 정렬되었던 것이다.
- Comparator와 Comparable은 모두 인터페이스로 컬렉션을 정렬하는데 필요한 메서드를 정의하고 있다.
- Comparable을 구현하고 있는 클래스들은 같은 타입의 인스턴스끼리 서보 비교할 수 있는 클래스들, 주로 Integer와 같은 wrapper 클래스와 String, Date, File과 같은 것들이며 기본적으로 오

름차순, 즉 작은 값에서부터 큰 값의 순서로 정렬되도록 구현되어 있다.
- 따라서 Comparable을 구현한 클래스는 정렬이 가능하다는 것을 의미한다.

## java.util.Comparator

```
public interface Comparator {
        int compare(Object o1, Object o2);
        boolean equals(Object obj);
}
```

## java.lang.Comparable

```
public interface Comparable {
        public int compareTo(Object o);
}
```

- compare()와 compareTo()는 선언형태와 이름이 다를뿐 두 객체를 비교한다는 같은 기능을 가지고 있다.
- 두 객체가 같으면 0, 비교하는 값보다 작으면 음수, 크면 양수를 반환한다.
- 예를 들어 TreeSet에서 Integer 인스턴스를 저장한다면 이때 정렬되는 기준이 compareTo메서드에 의한 것이다.
- **Comparable** : 기본 정렬기준을 구현하는데 사용
- **Comparator** : 기본 정렬기준 외에 다른 기준으로 정렬하고자할 때 사용

**day13_14/ComparatorEx.java**

```
package day13_14;

import java.util.*;

class Descending implements Comparator {
        public int compare(Object o1, Object o2) {
                if (o1 instanceof Comparable && o2 instanceof Comparable) {
                        Comparable c1 = (Comparable)o1;
                        Comparable c2 = (Comparable)o2;
                        return c1.compareTo(c2) * -1; // 또는 c2.compareTo(c1);
                }
                return -1;
        }
}

실행결과
strArr=[Dog, cat, lion, tiger]
strArr=[cat, Dog, lion, tiger]
strArr=[tiger, lion, cat, Dog]
```

- 문자열의 오름차순 정렬은 공백, 숫자, 대문자, 소문자 순으로 정렬된다(문자의 유니코드의 숫자가 작은 값부터 큰 값으로 정렬된다.)

- 대소문자를 구분하지 않고 비교하는 Comparator를 String 클래스에서 상수의 형태로 제공한다 (static final Comparator Case_INSENSITIVE_ORDER)

## HashSet

- HashSet은 Set인터페이스를 구현한 가장 대표적인 컬렉션이다.
- Set인터페이스의 특징대로 HashSet은 중복된 요소를 저장하지 않는다.

### HashSet의 메서드

| 생성자 또는 메서드 | 설명 |
|---|---|
| HashSet() | HashSet객체를 생성한다. |
| HashSet(Collection c) | 주어진 컬렉션을 포함하는 HashSet객체를 생성한다. |
| HashSet(int initialCapacity) | 주어진 값을 초기용량으로 하는 HashSet객체를 생성한다. |
| HashSet(int initialCapacity, float loadFactor) | 초기용량과 load factor를 지정하는 생성자. |
| boolean add(Object o) | 새로운 객체를 저장한다. |
| boolean addAll(Collection c) | 주어진 컬렉션에 저장된 모든 객체들을 추가한다.(합집합) |
| void clear() | 저장된 모든 객체를 삭제한다. |
| Object clone() | HashSet을 복제해서 반환한다(얕은 복사) |
| boolean contains(Object o) | 지정된 객체를 포함하고 있는지 알려준다. |
| boolean containsAll(Collection c) | 주어진 컬렊현에 저장된 모든 객체들을 포함하고 있는지 알려준다. |
| boolean isEmpty() | HashSet이 비어있는지 알려준다. |
| Iterator iterator() | Iterator를 반환한다. |
| boolean remove(Object o) | 지정된 객체를 HashSet에서 삭제한다.(성공하면 true, 실패하면 false) |
| boolean retainAll(Collection c) | 주어진 컬렉션에 저장된 객체와 동일한 것만 남기고 삭제한다.(교집합) |
| int size() | 저장된 객체의 개수를 반환한다. |
| Object[] toArray() | 저장된 객체들을 객체배열의 형태로 반환한다. |
| Object[] toArray(Object[] a) | 저장된 객체들을 주어진 객체배열(a)에 담는다. |

> load factor는 컬렉션 클래스에 저장공간이 가득 차기 전에 미리 용량을 확보하기 위한 것으로 이 값을 0.8로 지정하면, 저장공간의 80%가 채워졌을 때 용량이 두 배로 늘어난다. 기본값은 0.75, 즉 75%이다.

### day13_14/HashSetEx1.java

```
package day13_14;

import java.util.*;

public class HashSetEx1 {
	public static void main(String[] args) {
		Object[] objArr = {"1", Integer.valueOf(1), "2", "2", "3", "3",
"4", "4", "4" };
		Set set = new HashSet();

		for(int i = 0; i < objArr.length; i++) {
			set.add(objArr[i]); // HashSet에 objArr의 요소를 저장한다.
		}

		System.out.println(set);
	}
}
```

실행결과

```
[1, 1, 2, 3, 4]
```

### day13_14/HashSetEx2.java

```
package day13_14;

import java.util.*;

public class HashSetEx2 {
	public static void main(String[] args) {
		Set set = new HashSet();

		for (int i = 0; set.size() < 6; i++) {
			int num = (int)(Math.random()*45) + 1;
			set.add(Integer.valueOf(num));
		}

		List list = new LinkedList(set);  // LinkedList(Collection c)
		Collections.sort(list); // Collections.sort(List list)
		System.out.println(list);
	}
}
```

실행결과

```
[5, 13, 25, 30, 37, 38]
```

- 번호를 크기순으로 정렬하기 위해서 Collections클래스의 sort(List list)를 사용했다.
- 이 메서드는 인자로 List인터페이스 타입을 필요로 하기 때문에 LinkedList클래스의 생성자 LinkedList(Collection c)를 이용해서 HashSet에 저장된 객체들을 LinkedList에 담아서 처리

## equals()와 hashCode()

- HashSet의 add메서드는 새로운 요소를 추가하기 전에 기존에 저장된 요소와 같은 것인지 판별하기 위해서 추가하려는 요소의 equals()와 hashCode()를 호출하기 때문에 equals()와 hashCode()를 목적에 맞게 재정의 해야 한다.
- 재정의 하지 않는다면, 기본 equals()와 hashCode()는 모든 클래스의 상위 클래스인 Object에 정의되어 있는 메서드 이며, hashCode()은 생성된 인스턴스의 주소이며, equals()역시 동일한 주소일때 true를 반환한다.
- 인스턴스의 동일 여부가 아닌 논리적인 동일성으로 중복을 판단하기 위해서는 equals()와 hashCode()를 재정의한다.

**day13_14/Person.java**

```
package day13_14;

public class Person {
	String name;
	int age;

	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}

	public String toString() {
		return name + ":" + age;
	}
}
```

**day13_14/HashSetEx3.java**

```
package day13_14;

import java.util.*;

public class HashSetEx3 {
	public static void main(String[] args) {
		HashSet set = new HashSet();

		set.add("abc");
		set.add("abc");
		set.add(new Person("David", 10));
		set.add(new Person("David", 10));

		System.out.println(set);
	}
}
```

```
실행결과
[abc, David:10, David:10]
```

Person 클래스는 name과 age를 멤버변수로 갖는다. 이름(name)과 나이(age)가 같으면 같은 사람으로 인식하려는 의도록 작성되었으나 실행결과는 두 인스턴스의 name과 age의 값이 같음에도 불구하고 서로 다른 것으로 인식하여 "David:10"이 두번 출력 되었다.

이를 인스턴스의 일치가 아닌 논리적인 일치로 인식되게 하려면 equals()와 hashCode()를 재정의해야 한다.

**day13_14/Person2.java**

```
package day13_14;

public class Person2 {
	String name;
	int age;

	public  Person2(String name, int age) {
		this.name = name;
		this.age = age;
	}

	public boolean equals(Object obj) {
		if (obj instanceof Person2) {
			Person2 tmp = (Person2)obj;
			return name.equals(tmp.name) && age == tmp.age;
		}

		return false;
	}

	public int hashCode() {
		return (name + age).hashCode();  // 또는 java.util.Objects 클래스의
hash를 이용해도 된다. Objects.hash(name+age);
	}

	public String toString() {
		return name + ":" + age;
	}
}
```

**day13_14/HashSetEx4.java**

```
package day13_14;

import java.util.*;

public class HashSetEx4 {
	public static void main(String[] args) {
		HashSet set = new HashSet();
```

```
                set.add(new String("abc"));
                set.add(new String("abc"));
                set.add(new Person2("David", 10));
                set.add(new Person2("David", 10));

                System.out.println(set);
        }
}
```

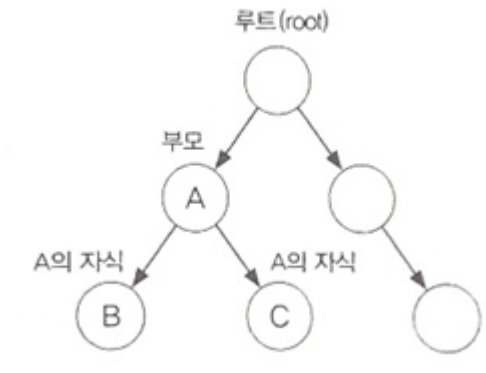
실행결과
```
[abc, David:10]
```

## TreeSet

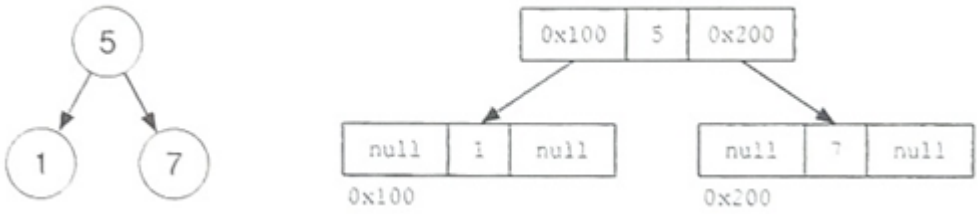
- TreeSet은 이진 검색트리(binary search tree)라는 자료구조의 형태로 데이터를 저장하는 컬렉션 클래스이다.
- 이진 검색 트리는 정렬, 검색, 범위검색(range search)에 높은 성능을 보이는 자료구조이다.
- Set 인터페이스를 구현했으므로 중복된 데이터의 저장을 허용하지 않으며, 정렬된 위치에 저장하므로 저장순서를 유지하지도 않는다.
- TreeSet은 정렬된 상태를 유지하기 때문에 단일 값 검색과 범위검색(range search)이 매우 빠르가.
- 다만 트리는 데이터를 순차적으로 저장하는 것이 아니라 저장위치를 찾아서 저장해야 하고, 삭제하는 경우 트리의 일부를 재구성해야 하므로 링크드 리스트보다 데이터의 추가/삭제 시간은 더 걸린다.
- 대신 배열이나 링크드리스트에 비해 검색과 정렬 기능이 더 뛰어나다.

### 이진 검색 트리(binary search tree)

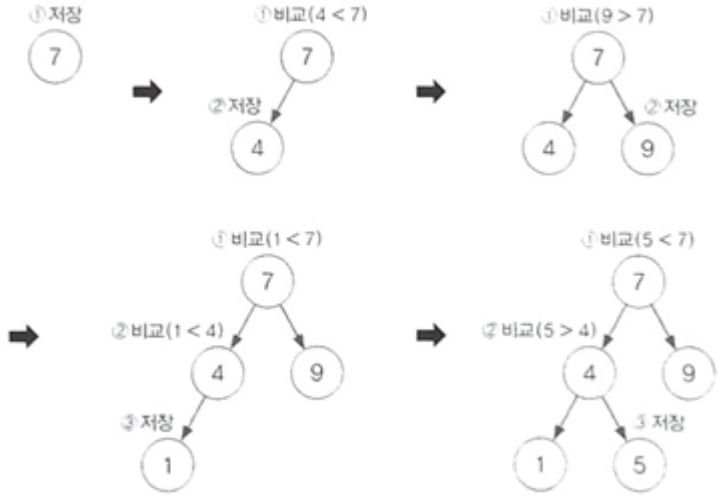
- 모든 노드는 최대 두 개의 자식노드를 가질 수 있다.
- 왼쪽 자식노드의 값은 부모노드의 값보다 작고 오른쪽자식노드의 값은 부모노드의 값모다 커야 한다.
- 노드의 추가 삭제에 시간이 걸린다.(순차적으로 저장하지 않으므로)
- 검색(범위검색)과 정렬에 유리하다.
- 중복된 값을 저장하지 못한다.




**이진검색트리(binary search tree)**




**이진 검색트리 저장과정**




## TreeSet의 생성자와 메서드

| 생성자 또는 메서드 | 설명 |
|---|---|
| TreeSet() | 기본생성자 |
| TreeSet(Collection c) | 주어진 컬렉션을 저장하는 TreeSet을 생성 |
| TreeSet(Comparator comp) | 주어진 정렬조건으로 정렬하는 TreeSet을 생성 |
| TreeSet(SortedSet s) | 주어진 SortedSet을 구현한 컬렉션을 저장하는 TreeSet을 생성 |
| boolean add(Object o)<br>boolean addAll(Collection c) | 지정된 객체(o) 또는 Collection(c)의 객체들을 Collection에 추가 |
| Object ceilling(Object o) | 지정된 객체와 같은 객체를 반환. 없으면 큰 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null |
| void clear() | 저장된 모든 객체를 삭제한다. |
| Object clone() | TreeSet을 복제하여 반환한다. |
| Comparator comparator() | TreeSet의 정렬기준(Comparator)을 반환한다. |
| boolean contains(Object o)<br>boolean containsAll(Collection c) | 지정된 객체(o) 또는 Collection의 객체들이 포함되어 있는지 확인한다. |
| NavigableSet descendingSet() | TreeSet에 저장된 요소들을 역순으로 정렬해서 반환 |

| 생성자 또는 메서드 | 설명 |
|---|---|
| Object first() | 정렬된 순서에서 첫 번쨰 객체를 반환한다. |
| Object floor(Object o) | 지정된 객체와 같은 객체를 반환. 없으면 작은 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null |
| SortedSet headSet(Object toElement) | 지정된 객체보다 작은 값의 객체들을 반환한다. |
| NavigableSet headSet(Object toElement, boolean inclusive) | 지정된 객체보다 작은 값의 객체들을 반환. inclusive가 true이면, 같은 값의 객체도 포함 |
| Object higher(Object o) | 지정된 객체보다 큰 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null |
| boolean isEmpty() | TreeSet이 비어있는지 확인한다. |
| Iterator iterator() | TreeSet의 Iterator를 반환한다. |
| Object last() | 정렬된 순서에서 마지막 객체를 반환한다. |
| Object lower(Object o) | 지정된 객체보다 작은 값을 가진 객체 중 제일 가까운 값의 객체를 반환. 없으면 null |
| Object pollFirst() | TreeSet의 첫번째 요소(제일 작은 값의 객체)를 반환 |
| Object pollLast() | TreeSet의 마지막 번째 요소(제일 큰 값의 객체)를 반환 |
| boolean remove(Object o) | 지정된 객체를 삭제한다. |
| boolean retainAll(Collection c) | 주어진 컬렉션과 공통 요소만 남기고 삭제한다.(교집합) |
| int size() | 저장된 객체의 개수를 반환한다. |
| SortedSet subSet(Object fromELement, Object toElement) | 범위검색(fromElement와 toElement사이)의 결과를 반환한다.(끝 범위인 toElement는 범위에 포함되지 않음) |
| NavigableSet<E> subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) | 범위 검색(fromElement와 toElement사이)의 결과를 반환한다.(fromInclusive가 true이면 시작값이 포함되고, toInclusive가 true면 끝값이 포함된다) |
| SortedSet tailSet(Object fromElement) | 지정된 객체보다 큰 값의 객체들을 반환한다. |
| Object[] toArray() | 지정된 객체를 객체배열로 반환한다. |
| Object[] toArray(Object[] a) | 저장된 객체를 주어진 객체배열에 저장하여 반환한다. |

**day13_14/TreeSetEx1.java**

```
package day13_14;

import java.util.*;

public class TreeSetEx1 {
	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(num);
		}

		System.out.println(set);
	}
}
```

실행결과

```
[2, 15, 17, 28, 32, 36]
```

### day13_14/TreeSetEx2.java

```
package day13_14;

import java.util.*;

public class TreeSetEx2 {
	public static void main(String[] args) {
		TreeSet set = new TreeSet();

		String from = "b";

		String to = "d";

		set.add("abc");
		set.add("alien");
		set.add("bat");

		set.add("car");
		set.add("Car");
		set.add("disc");

		set.add("dance");
		set.add("dZZZZ");
		set.add("dzzzz");

		set.add("elephant");
		set.add("elevator");
		set.add("fan");
		set.add("flower");

		System.out.println(set);
		System.out.println("range search : from " + from + " to " + to);
```

```
                System.out.println("result1 : " + set.subSet(from, to));
                System.out.println("result2 : " + set.subSet(from, to + "zzz"));
        }
}
```

```
실행결과
[Car, abc, alien, bat, car, dZZZZ, dance, disc, dzzzz, elephant, elevator, fan,
flower]
range search : from b to d
result1 : [bat, car]
result2 : [bat, car, dZZZZ, dance, disc]
```

### day13_14/TreeSetEx3.java

```
package day13_14;

import java.util.*;

public class TreeSetEx3 {
        public static void main(String[] args) {
                TreeSet set = new TreeSet();
                int[] score = {80, 95, 50, 35, 45, 65, 10, 100};

                for(int i = 0; i < score.length; i++) {
                        set.add(Integer.valueOf(score[i]));
                }

                System.out.println("50보다 작은 값 : " +
set.headSet(Integer.valueOf(50)));
                System.out.println("50보다 큰 값 : " +
set.tailSet(Integer.valueOf(50)));
        }
}
```

```
실행결과
50보다 작은 값 : [10, 35, 45]
50보다 큰 값 : [50, 65, 80, 95, 100]
```

## HashMap과 Hashtable

- Hashtable과 HashMap의 관계는 Vector와 ArrayList의 관계와 같아서 Hashtable보다는 새로운 버전인 HashMap을 사용할 것
- HashMap은 Map을 구현했으므로 앞에서 살펴본 Map의 특징, 키(key)와 값(value)를 묶어서 하나의 데이터(entry)로 저장한다는 특징을 갖는다.
- 해싱(hashing)을 사용하기 때문에 많은 데이터를 검색하는데 뛰어난 성능을 보인다.
- 해싱(hashing)으 사용한 컬렉션 클래스들은 저장순서를 유지하지 않는다.

```
public class HashMap extends AbstractMap implements Map, Cloneable, Serializable {
        transient Entry[] table;
        ...
```

```
        static class Entry implements Map.Entry {
                final Object key;
                Object value;
                ...
        }
        ...
}
```

- HashMap은 Entry라는 내부 클래스를 정의하고, 다시 Entry타입의 배열을 선언하고 있다.
- 키(key)와 값(value)은 별개의 값이 아니라 서로 관련된 값이기 때문에 각각의 배열로 선언하기보다는 하나의 클래스로 정의해서 하나의 배열로 다루는 것이 데이터 무결성(integrity)적인 측면에서 더 바람직하기 때문이다.
- HashMpa은 키와 값을 각각 Object 타입으로 저장한다, 즉 (Object, Object)의 형태로 저장하기 때문에 어떤 객체도 저장할 수 있지만 키는 주로 String을 대문자 또는 소문자로 통일해서 사용하곤 한다.
- 키는 저장된 값을 찾는데 사용되는 것이기 때문에 컬렉션 내에서 유일(unique)해야 한다. (HashMap에 저장된 데이터를 하나의 키로 검색했을 때 결과가 단 하나이어야 함을 뜻한다.)
- **키(key) : ** 컬렉션 내의 키(key) 중에서 유일해야 한다.
- **값(value) : ** 키(key)와 달리 데이터의 중복을 허용한다.

## HashMap의 생성자와 메서드

| 생성자 또는 메서드 | 설명 |
|---|---|
| HashMap() | HashMap객체를 생성 |
| HashMap(int initialCapacity) | 지정된 값을 초기용량으로 하는 HashMap객체를 생성 |
| HashMap(int initialCapacity, float loadFactor) | 지정된 초기용량과 load factor의 HashMap객체를 생성 |
| HashMap(Map m) | 지정된 Map의 모든 요소를 포함하는 HashMap객체를 생성 |
| void clear() | HashMap에 저장된 모든 객체를 제거 |
| Object clone() | 현재 HashMap을 복제해서 반환 |
| boolean containsKey(Object key) | HashMap에 지정된 키(key)가 포함되어있는지 알려준다.(포함되어 있다면 true) |
| boolean containsValue(Object value) | HashMap에 지정된 값(value)이 포함되어있는지 알려준다.(포함되어 있으면 true) |
| Set entrySet() | HashMap에 저장된 키와 값을 엔트리(키와 값의 결합)의 형태로 Set에 저장해서 반환 |

| 생성자 또는 메서드 | 설명 |
| --- | --- |
| Object get(Object key) | 지정된 키(key)의 값(객체)를 반환, 못찾으면 null 반환 |
| Object getOrDetail(Object key, Object defaultValue) | 지정된 키(key)의 값(객체)를 반환한다. 키를 못찾으면 기본값(defaultValue)로 지정한 객체를 반환 |
| boolean isEmpty() | HashMap이 비어있는지 알려준다. |
| Set keySet() | HashMap에 저장된 모든 키가 저장된 Set을 반환 |
| Object put(Object key, Object value) | 지정된 키와 값을 HashMap에 저장 |
| void putAll(Map m) | Map에 저장된 모든 요소를 HashMap에 저장 |
| Object remove(Object key) | HashMap에서 지정된 키로 저장된 값(객체)를 제거 |
| Object replace(Object key, Object value) | 지정된 키와 값을 지정된 객체(value)로 대체 |
| boolean replace(Object key, Object oldValue, Object newValue) | 지정된 키와 객체(oldValue)가 모두 일치하는 경에만 새로운 객체(newValue)로 대체 |
| int size() | HashMap에 저장된 요소의 개수를 반환 |
| Collection values() | HashMap에 저장된 모든 값을 컬렉션 형태로 반환 |

**day13_14/HashMapEx1.java**

```
package day13_14;

import java.util.*;

public class HashMapEx1 {
	public static void main(String[] args) {
		HashMap map = new HashMap();
		map.put("myId", "1234");
		map.put("asdf", "1111");
		map.put("asdf", "1234");

		Scanner s = new Scanner(System.in); // 화면으로부터 라인단위로 입력
을 받는다.

		while(true) {
			System.out.println("id와 password를 입력해주세요.");
			System.out.print("id : ");
			String id = s.nextLine().trim();
			System.out.println();

			System.out.print("password :");
			String password = s.nextLine().trim();
			System.out.println();

			if (!map.containsKey(id)) {
```

```
                                System.out.println("입력하신 id는 존재하지 않습니다.
다시 입력해주세요.");

                                continue;
                        } else {
                                if (!(map.get(id)).equals(password)) {
                                        System.out.println("비밀번호가 일치하지 않습
니다. 다시 입력해주세요.");
                                } else {
                                        System.out.println("id와 비밀번호가 일치합니
다.");

                                        break;
                                }
                        }
                }
        }
}
```

```
실행결과
id와 password를 입력해주세요.
id : asdf

password :1111

비밀번호가 일치하지 않습니다. 다시 입력해주세요.
id와 password를 입력해주세요.
id : asdf

password :1234

id와 비밀번호가 일치합니다.
```

### day13_14/HashMapEx2.java

```
package day13_14;

import java.util.*;

public class HashMapEx2 {
        public static void main(String[] args) {
                HashMap map = new HashMap();
                map.put("이병헌", Integer.valueOf(90));
                map.put("김태리", Integer.valueOf(100));
                map.put("유연석", Integer.valueOf(100));
                map.put("김민정", Integer.valueOf(80));
                map.put("변요한", Integer.valueOf(90));

                Set set = map.entrySet();
                Iterator it = set.iterator();

                while(it.hasNext()) {
                        Map.Entry e = (Map.Entry)it.next();
                        System.out.println("이름 : " + e.getKey() + ", 점수 : " +
e.getValue());
```

```
                }

                set = map.keySet();
                System.out.println("참가자 명단 : " + set);

                Collection values = map.values();
                it = values.iterator();

                int total = 0;
                while(it.hasNext()) {
                        Integer i = (Integer)it.next();
                        total += i.intValue();
                }

                System.out.println("총점 : " + total);
                System.out.println("평균 : " + (float)total / set.size());
                System.out.println("최고점수 : " + Collections.max(values));
                System.out.println("최저점수 : " + Collections.min(values));
        }
}
```

```
실행결과
이름 : 변요한, 점수 : 90
이름 : 김태리, 점수 : 100
이름 : 유연석, 점수 : 100
이름 : 이병헌, 점수 : 90
이름 : 김민정, 점수 : 80
참가자 명단 : [변요한, 김태리, 유연석, 이병헌, 김민정]
총점 : 460
평균 : 92.0
최고점수 : 100
최저점수 : 80
```

# TreeMap

- TreeMap은 이진검색트리 형태로 키와 값의 쌍으로 이루어진 데이터를 저장한다
- 검색과 정렬에 적합한 컬렉션 클래스이다.
- 검색에 관련한 대부분의 경우에서 HashMap이 TreeMap보다 더 뛰어나므로 HashMap을 사용하는 것이 좋다. 다만 범위 검색이나 정렬이 필요한 경우에는 TreeMap을 사용하는 것이 좋다.

## TreeMap의 생성자와 메서드

| 생성자 또는 메서드 | 설명 |
|---|---|
| TreeMap() | TreeMap 객체를 생성 |
| TreeMap(Comparator c) | 지정된 Comparator를 기준으로 정렬하는 TreeMap객체를 생성 |
| TreeMap(Map m) | 주어진 Map에 저장된 모든 요소를 포함하는 TreeMap을 생성 |

| 생성자 또는 메서드 | 설명 |
| --- | --- |
| TreeMap(SortedMap m) | 주어진 SortedMap에 저장된 모든 요소를 포함하는 TreeMap을 생성 |
| Map.Entry ceilingEntry(Object key) | 지정된 key와 일치하거나 큰 것중 제일 작은 것의 키와 값의 쌍(Map.Entry)를 반환. 없으면 null을 반환 |
| Object ceillingKey(Object key) | 지정된 key와 일치하거나 큰 것중 제일 작은 것의 키와 값의 쌍(Map.Entry)를 반환, 없으면 null을 반환 |
| void clear() | TreeMap에 저장된 모든 객체를 제거 |
| Object clone() | 현재 TreeMap을 복제해서 반환 |
| Comparator comparator() | TreeMap의 정렬기준이 되는 Comparator를 반환 Comparator가 지정되지 않았다면 null을 반환 |
| boolean containsKey(Object key) | TreeMap에 지정된 키(key)가 포함되어 있는지를 알려줌(포함되어 있으면 true) |
| boolean containsValue(Object value) | TreeMap에 저장된 값(value)이 포함되어 있는지 알려줌(포함되어 있으면 true) |
| NavigableSet descendingKeySet() | TreeMap에 저장된 키를 역순으로 정렬해서 NavigableSet에 담아서 반환 |
| Set entrySet() | TreeMap에 저장된 키와 값을 엔트리(키와 값의 결합)의 형태로 Set에 저장해서 반환 |
| Map.Entry firstEntry() | TreeMap에 저장된 첫번째(가장 작은) 키와 값의 쌍(Map.Entry)을 반환 |
| Object firstKey() | TreeMap에 저장된 첫번째(가장 작은) 키를 반환 |
| Map.Entry floorEntry(Object key) | 지정된 key와 일치하거나 작은 것 중에서 제일 큰 키를 반환. 없으면 null을 반환 |
| Object get(Object key) | 지정된 키(key)의 값(객체)를 반환 |
| SortedMap headMap(Object toKey) | TreeMap에 저장된 첫번째 요소부터 지정된 범위에 속한 모든 요소가 담긴 SortedMap을 반환 (toKey는 미포함) |
| NavigableMap headMap(Object toKey, boolean inclusive) | TreeMap에 저장된 첫번째 요소부터 지정된 범위에 속한 모든 요소가 담긴 SortedMap을 반환. inclusive의 값이 true면 toKey도 포함 |
| Map.Entry higherEntry(Object key) | 지정된 Key키 보다 큰 키 중에서 제일 작은 키의 쌍(Map.Entry)을 반환, 없으면 null을 반환 |
| boolean isEmpty() | TreeMap이 비어있는지 알려준다. |

| 생성자 또는 메서드 | 설명 |
| --- | --- |
| Set keySet() | TreeMap에 저장된 모든 키가 저장된 Set을 반환 |
| Map.Entry lastEntry() | TreeMap에 저장된 마지막 키(가장 큰 키)의 쌍을 반환 |
| Object lastKey() | TreeMap에 저장된 마지막 키(가장 큰 키)를 반환 |
| Map.Entry lowerEntry(Object key) | 지정된 key보다 작은 키 중에서 제일 큰 키의 쌍(Map.Entry)을 반환, 없으면 null을 반환 |
| NavigableSet navigableKeySet() | TreeMap의 모든 키가 담긴 NavigableSet을 반환 |
| Map.Entry pollFirstEntry() | TreeMap에서 제일 작은 키를 제거하면서 반환 |
| Map.Entry pollLastEntry() | TreeMap에서 제일 큰 키를 제거하면서 반환 |
| Object put(Object key, Object value) | 지정된 키와 값을 TreeMap에 저장 |
| void putAll(Map map) | Map에 저장된 모든 요소를 TreeMap에 저장 |
| Object remove(Object key) | TreeMap에 지정된 키로 저장된 값(객체)를 제거 |
| Object replace(Object k, Object v) | 기존의 키(k)의 값을 지정된 값(v)로 변경 |
| boolean replace(Object key, Object oldValue, Object newValue) | 기존의 키(key)값을 새로운 값(newValue)으로 변경. 단, 기존의 값과 지정된 값(oldValue)가 일치해야 함 |
| int size() | TreeMap에 저장된 요소의 개수를 반환 |
| NavigableMap subMap(Object fromKey, boolean fromInclusive, Object toKey, boolean toInclusive) | 지정된 두개의 키 사이에 있는 모든 요소들이 담긴 NavigableMap을 반환. fromInclusive가 true면 범위에 fromKey포함. toInclusive가 true면 범위에 toKey포함 |
| SortedMap subMap(Object fromKey, Object toKey) | 지정된 구 개의 키 사이에 있는 모든 요소들이 담긴 SortedMap을 반환(toKey는 포함되지 않는다.) |
| NavigableMap tailMap(Object fromKey) | 지정된 키부터 마지막 요소의 범위에 속한 요소가 담긴 SortedMap을 반환 |
| NavigableMap tailMap(Object fromKey, boolean inclusive) | 지정된 키부터 마지막 요소의 범위에 속한 요소가 담긴 NavigableMap을 반환. inclusive가 true면 fromKey 포함 |
| Collection values() | TreeMap에 저장된 모든 값을 걸렉션의 형태로 반환 |

**day13_14/TreeMapEx.java**

```
package day13_14;
```

```
import java.util.*;

public class TreeMapEx {
        public static void main(String[] args) {
                String[] data = {"A", "K", "A", "K", "D", "K", "A", "K", "K", "K",
"Z", "D"};

                TreeMap map = new TreeMap();

                for (int i = 0; i < data.length; i++) {
                        if (map.containsKey(data[i])) {
                                Integer value = (Integer)map.get(data[i]);
                                map.put(data[i], Integer.valueOf(value.intValue()
+ 1));
                        } else {
                                map.put(data[i], Integer.valueOf(1));
                        }
                }

                Iterator it = map.entrySet().iterator();

                System.out.println("= 기본정렬 =");
                while(it.hasNext()) {
                        Map.Entry entry = (Map.Entry)it.next();
                        int value = ((Integer)entry.getValue()).intValue();
                        System.out.println(entry.getKey() + " : " + printBar('#',
value) + " " + value);
                }

                System.out.println();

                // map을 ArrayList을 변환한 다음에 Collections.sort()로 정렬
                Set set = map.entrySet();
                List list = new ArrayList(set); //  ArrayList(Collection c)

                Collections.sort(list, new ValueComparator());

                it = list.iterator();

                System.out.println("= 값의 크기가 큰 순서로 정렬 =");
                while(it.hasNext()) {
                        Map.Entry entry = (Map.Entry)it.next();
                        int value = ((Integer)entry.getValue()).intValue();
                        System.out.println(entry.getKey() + " : " + printBar('#',
value) + " " + value);
                }
        }

        static class ValueComparator implements Comparator {
                public int compare(Object o1, Object o2) {
                        if (o1 instanceof Map.Entry && o2 instanceof Map.Entry) {
                                Map.Entry e1 = (Map.Entry)o1;
                                Map.Entry e2 = (Map.Entry)o2;

                                int v1 = ((Integer)e1.getValue()).intValue();
                                int v2 = ((Integer)e2.getValue()).intValue();
```

```
                        return v2 - v1;
                    }
                    return -1;
                }
        }

        public static String printBar(char ch, int value) {
                char[] bar = new char[value];

                for(int i = 0; i < bar.length; i++) {
                        bar[i] = ch;
                }

                return new String(bar);
        }
}

실행결과
= 기본정렬 =
A : ### 3
D : ## 2
K : ###### 6
Z : # 1

= 값의 크기가 큰 순서로 정렬 =
K : ###### 6
A : ### 3
D : ## 2
Z : # 1
```

# Properties

- Properties는 HashMap의 구버전인 Hashtable을 상속받아 구현한 것
- Hashtable은 키와 값을 (Object, Object)의 형태로 저장하는데 비해 Properties는 (String, String)의 형태로 저장하는 보다 단순화된 컬렉션 클래스이다.
- 주로 애플리케이션의 환경 설정과 관련된 속성(property)를 저장하는데 사용되며 데이터를 파일로 부터 읽고 쓰는 편리한 기능을 제공한다.

## Properties의 생상자와 메서드

| 생성자 또는 메서드 | 설명 |
|---|---|
| Properties() | Properties객체를 생성한다. |
| Properties(Properties defaults) | 지정된 Properties에 저장된 목록을 가진 Properties 객체를 생성한다. |
| String getProperty(String key) | 지정된 키(key)의 값(value)를 반환한다. |
| String getProperty(String key, String defaultValue) | 지정된 키(key)의 값(value)를 반환한다. 키를 못찾으면 defaultValue를 반환한다. |

| 생성자 또는 메서드 | 설명 |
| --- | --- |
| void list(PrintStream out) | 지정된 PrintStream에 저장된 목록을 출력한다. |
| void list(PrintWriter out) | 지정된 PrintWriter에 저장된 목록을 출력한다. |
| void load(InputStream inStream) | 지정된 InputStream으로부터 목록을 읽어서 저장한다. |
| void load(Reader reader) | 지정된 Reader으로 부터 목록을 읽어서 저장한다. |
| void loadFromXML(inputStream in) | 지정된 InputStream으로부터 XML 문서를 읽어서 XML문서에 저장된 목록을 읽어다 담는다(load & save) |
| Enumeration propertyNames() | 목록의 모든 키(key)가 담긴 Enumeration을 반환한다. |
| void save(OutputStream out, String header) | deprecated되었으므로 store()를 사용 |
| Object setProperty(String key, String value) | 지정된 키와 값을 저장한다.이미 존재하는 키(key)면 새로운 값(value)로 바뀐다. |
| void store(OutputStream out, String comments) | 저장된 목록을 지정된 OutputStream에 출력(저장)한다. comments는 목록에 대한 주석으로 저장된다. |
| void store(Writer writer, String comments) | 저장된 목록을 지정된 Writer에 출력(저장)한다. comments는 목록에 대한 설명(주석)으로 저장된다. |
| void storeToXML(OutputStream os, String comments) | 지정된 목록을 지정된 출력스트림에 XML문서로 출력(저장)한다. comment는 목록에 대한 설명(주석)으로 저장된다. |
| void storeToXML(OutputStream os, String comment, String encoding) | 지정된 목록을 지정된 출력스트림에 해당 인코딩의 XML문서로 출력(저장)한다. comment는 목록에 대한 설명(주석)으로 저장된다. |
| Set stringPropertyNames() | properties에 저장되어 있는 모든 키(key)를 Set에 담아서 반환한다. |

**day13_14/PropertiesEx1.java**

```
package day13_14;

import java.util.*;

public class PropertiesEx1 {
        public static void main(String[] args) {
                Properties prop = new  Properties();
```

```
                // prop에 키와 값(key, value)를 저장한다.
                prop.setProperty("timeout", "30");
                prop.setProperty("language", "kr");
                prop.setProperty("size", "10");
                prop.setProperty("capacity", "10");

                // prop에 저장된 요소들을 Enumeration을 이용해서 출력한다.
                Enumeration e = prop.propertyNames();

                while(e.hasMoreElements()) {
                        String element = (String)e.nextElement();
                        System.out.println(element + "=" +
prop.getProperty(element));
                }

                System.out.println();
                prop.setProperty("size", "20"); // size의 값을 20으로 변경한다.
                System.out.println("size=" + prop.getProperty("size"));
                System.out.println("capacity=" + prop.getProperty("capacity",
"20"));
                System.out.println("loadFactor=" + prop.getProperty("loadFactor",
"0.75"));

                System.out.println(prop); // prop에 저장된 요소들을 출력한다.
                prop.list(System.out); // prop에 저장된 요소들을 화면(System.out)에
출력한다.
        }
}
```

실행결과

```
capacity=10
size=10
timeout=30
language=kr

size=20
capacity=10
loadFactor=0.75
{size=20, language=kr, timeout=30, capacity=10}
-- listing properties --
size=20
language=kr
timeout=30
capacity=10
```

### day13_14/PropertiesEx2.java

```
package day13_14;

import java.io.*;
import java.util.*;

public class PropertiesEx2 {
        public static void main(String[] args) {
```

```
                // commandLine에서 inputfile을 지정해주지 않으면 프로그램을 종료한다.
                if (args.length != 1) {
                        System.out.println("USAGE: java PropertiesEx2
INPUTFILENAME");
                        System.exit(0);
                }

                Properties prop = new Properties();

                String inputFile = args[0];

                try {
                        prop.load(new FileInputStream(inputFile));
                } catch (IOException e) {
                        System.out.println("지정된 파일을 찾을 수 없습니다.");
                        System.exit(0);
                }

                String name = prop.getProperty("name");
                String[] data = prop.getProperty("data").split(",");
                int max = 0, min = 0;
                int sum = 0;

                for (int i = 0; i < data.length; i++) {
                        int intValue = Integer.parseInt(data[i]);

                        if (i==0) max = min = intValue;

                        if (max < intValue)
                                max = intValue;
                        else if (min > intValue)
                                min = intValue;

                        sum += intValue;
                }

                System.out.println("이름 :" + name);
                System.out.println("최대값 :" + max);
                System.out.println("최소값 :" + min);
                System.out.println("합계 :" + sum);
                System.out.println("평균 :" + (sum*100.0/data.length)/100);
        }
}

input.txt
# 주석 입니다.
name=Lee, Yonggyo
data=9,1,5,2,8,13,26,11,35,1
```

### day13_14/PropertiesEx3.java

```
package day13_14;

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

```
public class PropertiesEx3 {
	public static void main(String[] args) {
		Properties prop = new Properties();

		prop.setProperty("timeout", "30");
		prop.setProperty("language", "kr");
		prop.setProperty("size", "10");
		prop.setProperty("capacity", "10");

		try {
			prop.store(new FileOutputStream("output.txt"), "Properties
Excample");

			prop.storeToXML(new FileOutputStream("output.xml"),
"Properties Example");

		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

실행결과

output.txt

```
#Properties Excample
#Sun May 08 00:04:25 KST 2022
size=10
language=kr
timeout=30
capacity=10
```

output.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Properties Example</comment>
<entry key="size">10</entry>
<entry key="language">kr</entry>
<entry key="timeout">30</entry>
<entry key="capacity">10</entry>
</properties>
```

### day13_14/PropertiesEx4.java

```
package day13_14;

import java.util.*;

public class PropertiesEx4 {
	public static void main(String[] args) {
		Properties sysProp = System.getProperties();
		System.out.println("java.version :" +
sysProp.getProperty("java.version"));
```

```
                System.out.println("user.language :" +
sysProp.getProperty("user.language"));

                sysProp.list(System.out);
        }
}
```

실행결과

```
java.version :17.0.1
user.language :ko
-- listing properties --
java.specification.version=17
sun.cpu.isalist=amd64
sun.jnu.encoding=MS949
java.class.path=D:\javaEx\lecture\bin
java.vm.vendor=Eclipse Adoptium
sun.arch.data.model=64
user.variant=
java.vendor.url=https://adoptium.net/
java.vm.specification.version=17
os.name=Windows 10
sun.java.launcher=SUN_STANDARD
user.country=KR
sun.boot.library.path=D:\자바공부\eclipse\plugins\org.eclipse.j...
sun.java.command=day13_14.PropertiesEx4
jdk.debug=release
sun.cpu.endian=little
user.home=C:\Users\YONGGYO
user.language=ko
java.specification.vendor=Oracle Corporation
java.version.date=2021-10-19
java.home=D:\자바공부\eclipse\plugins\org.eclipse.j...
file.separator=\
java.vm.compressedOopsMode=Zero based
line.separator=

java.vm.specification.vendor=Oracle Corporation
java.specification.name=Java Platform API Specification
user.script=
sun.management.compiler=HotSpot 64-Bit Tiered Compilers
java.runtime.version=17.0.1+12
user.name=YONGGYO
path.separator=;
os.version=10.0
java.runtime.name=OpenJDK Runtime Environment
file.encoding=MS949
java.vm.name=OpenJDK 64-Bit Server VM
java.vendor.version=Temurin-17.0.1+12
java.vendor.url.bug=https://github.com/adoptium/adoptium-...
java.io.tmpdir=C:\Users\YONGGYO\AppData\Local\Temp\
java.version=17.0.1
user.dir=D:\javaEx\lecture
os.arch=amd64
java.vm.specification.name=Java Virtual Machine Specification
sun.os.patch.level=
native.encoding=MS949
```

```
java.library.path=D:\자바공부\eclipse\plugins\org.eclipse.j...
java.vm.info=mixed mode
java.vendor=Eclipse Adoptium
java.vm.version=17.0.1+12
sun.io.unicode.encoding=UnicodeLittle
java.class.version=61.0
```

# Collections

- Arrays가 배열과 관련된 메서드를 제공하는 것처럼, Collections는 컬렉션과 관련된 메서드를 제공한다.
- fill(), copy(), sort(), binarySearch()등의 메서드는 두 클래스 모두 포함되어 있고 같은 기능을 한다.

> java.util.Collection은 인터페이스이고, java.util.Collections는 클래스이다.

## 컬렉션의 동기화

멀티 쓰레드 프로그래미엥서는 하나의 객체를 여러 쓰레드가 동시에 접근할 수 있기 때문에 데이터의 일관성을 유지하기 위해서는 공유되는 객체에 동기화(synchronization)가 필요하다.

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

## 변경불가 컬렉션 만들기

```
static Collection unmodifiableCollection(Collection c)
static List unmodifiableList(List list)
static Set unmodifiableSet(Set s)
static Map unmodifiableMap(Map m)
static NavigableSet unmodifiableNavigableSet(NavigableSet s)
static SortedSet unmodifiableSortedSet(SortedSet s)
static NavigableMap unmodifiableNavigableMap(NavigableMap m)
static SortedMap unmodifiableSortedMap(SortedMap m)
```

## 싱글톤 컬렉션 만들기

```
static List singletonList(Object o)
static Set singleton(Object o)
static Map singletonMap(Object key, Object value)
```

## 한 종류의 객체만 저장하는 컬렉션 만들기

```
static Collection checkedCollection(Collection c, Class type)
static List checkedList(List list, Class type)
static Set checkedSet(Set s, Class type)
static Map checkedMap(Map m, Class keyType, Class valueType)
static Queue checkedQueue(Queue queue, Class type)
static NavigableSet checkedNavigableSet(NavigableSet s, Class type)
static SortedSet checkedSortedSet(SortedSet s, Class type)
static NavigableMap checkedNavigableMap(NavigableMap m, Class keyType, Class
valueType)
static SortedMap checkedSortedMap(SortedMap m, Class keyType, Class valueType)
```

### day13_14/CollectionsEx.java

```
package day13_14;

import java.util.*;
import static java.util.Collections.*;

public class CollectionsEx {
        public static void main(String[] args) {
                List list = new ArrayList();
                System.out.println(list);

                addAll(list, 1,2,3,4,5);
                System.out.println(list);

                rotate(list, 2); // 오른쪽으로 두 칸씩 이동
                System.out.println(list);

                swap(list, 0, 2); // 첫 번째와 세 번째를 교환
                System.out.println(list);

                shuffle(list); //  저장된 요소와 위치를 임의로 변경
                System.out.println(list);

                sort(list); // 정렬
                System.out.println(list);

                sort(list, reverseOrder()); // 역순 정렬, reverse(list); 와 동일
                System.out.println(list);

                int idx = binarySearch(list, 3); // 3이 저장된 위치(index)를 반환
                System.out.println("index of 3 = " + idx);

                System.out.println("max=" + max(list));
                System.out.println("min=" + min(list));
                System.out.println("min=" + max(list, reverseOrder()));

                fill(list, 9); // list를 9로 채운다.
                System.out.println("list="+list);

                // List와 같은 크기의 새로운 list를 생성하고 2로 채운다. 단, 결과는
변경불가
                List newList = nCopies(list.size(), 2);
```

```
                System.out.println("newList=" + newList);

                System.out.println(disjoint(list, newList)); // 공통요소가 없으면
true


                copy(list, newList);
                System.out.println("newList="+newList);
                System.out.println("list="+list);

                replaceAll(list, 2, 1);
                System.out.println("list="+list);

                Enumeration e = enumeration(list);
                ArrayList list2 = list(e);

                System.out.println("list2=" + list2);
        }
}
```

실행결과

```
[]
[1, 2, 3, 4, 5]
[4, 5, 1, 2, 3]
[1, 5, 4, 2, 3]
[4, 2, 3, 5, 1]
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
index of 3 = 2
max=5
min=1
min=1
list=[9, 9, 9, 9, 9]
newList=[2, 2, 2, 2, 2]
true
newList=[2, 2, 2, 2, 2]
list=[2, 2, 2, 2, 2]
list=[1, 1, 1, 1, 1]
list2=[1, 1, 1, 1, 1]
```