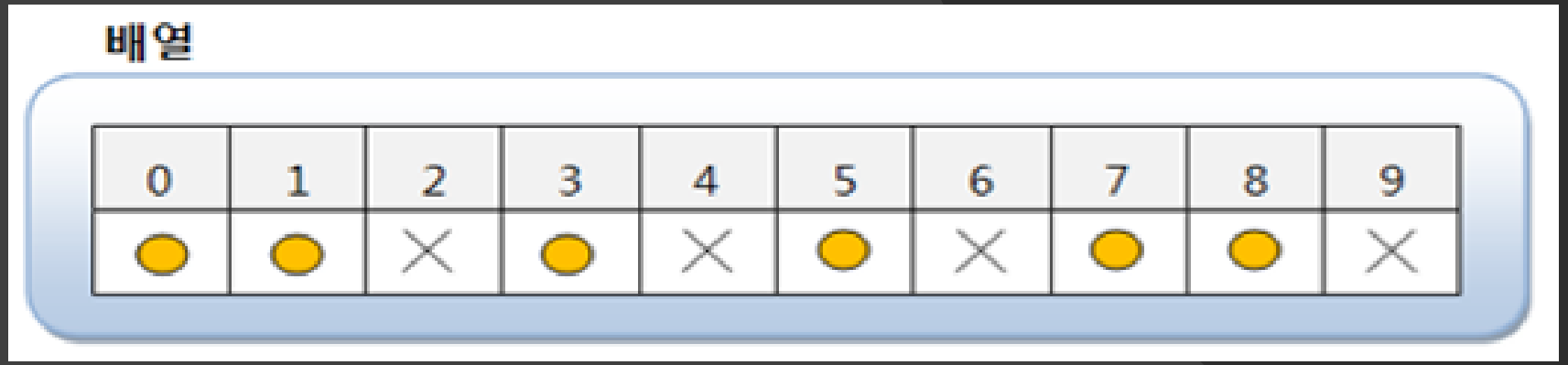
Java 기초

콜렉션(Collection)

콜렉션(Collection)

- 기존 배열의 문제점
  - 저장할 수 있는 객체 수가 배열을 생성할 때 결정 => 불특정 다수의 객체를 저장하기에는 문제점이 많다.
  - 각체 삭제했을 때 해당 인덱스가 비게 됨 => 객체를 저장하려면 어디가 비어있는지 확인이 필요하다.

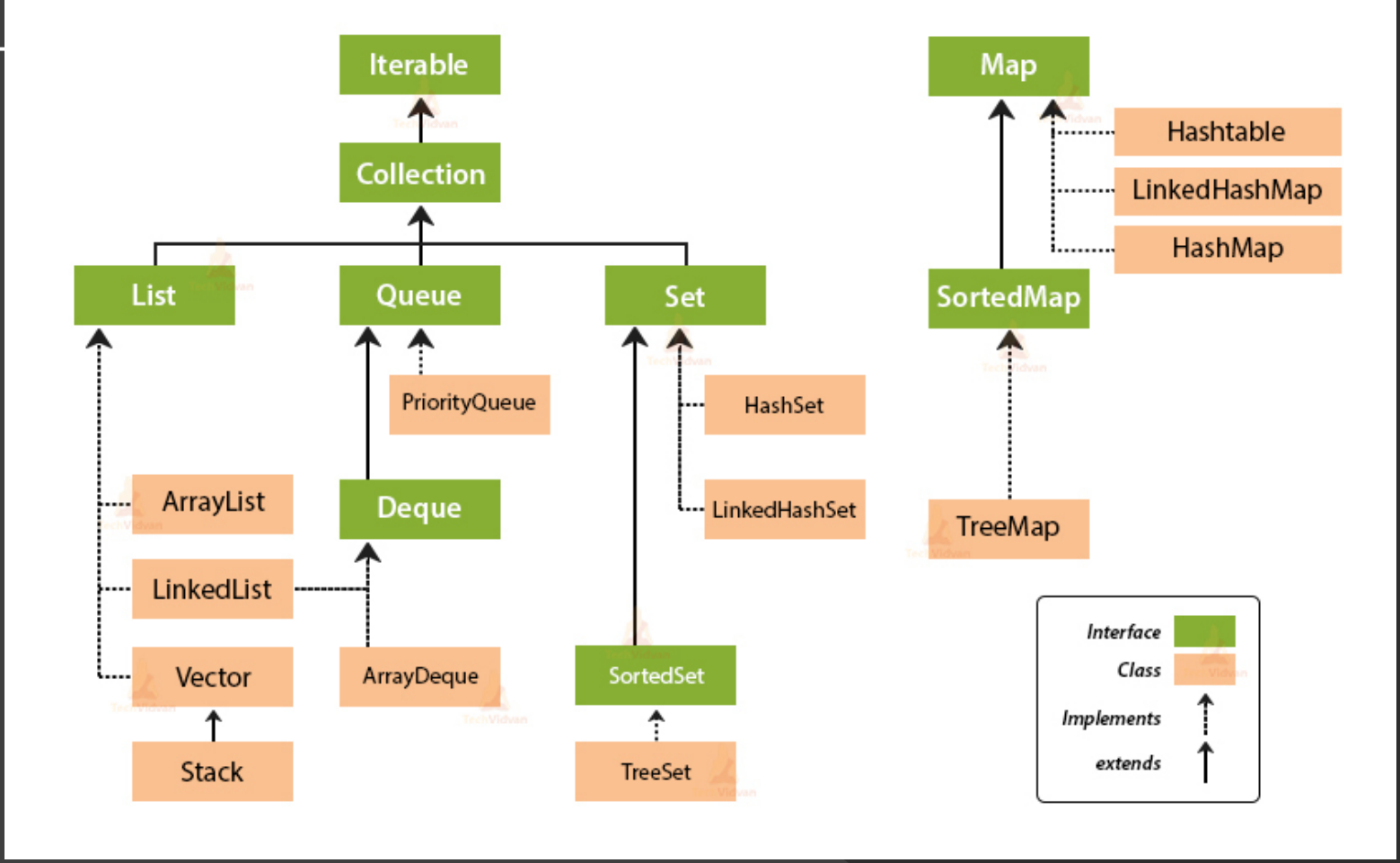


콜렉션(Collection)

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

콜렉션(Collection)

- 컬렉션 주요 인터페이스



콜렉션(Collection)

- 컬렉션 주요 인터페이스

인터페이스	구현클래스	특징
Set	HashSet TreeSet	순서를 유지하지 않는 데이터의 집합으로 데이터의 중복을 허용하지 않는다.
List	LinkedList Vector ArrayList	순서가 있는 데이터의 집합으로 데이터의 중복을 허용한다.
Queue	LinkedList PriorityQueue	List와 유사
Map	Hashtable HashMap TreeMap	키(Key), 값(Value)의 쌍으로 이루어진 데이터으 집합으로, 순서는 유지되지 않으며 키(Key)의 중복을 허용하지 않으나 값(Value)의 중복은 허용한다.

콜렉션(Collection)

- List 주요 메서드

메서드	설 명
void add(int index, Object element) 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. (List의 첫 번째 요소부터 순방향으로 찾는다.)
int lastIndexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 마지막 요소부터 역방향으로 찾는다.)
ListIterator listIterator( ) 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.

콜렉션(Collection)

- ArrayList
  - Vector를 개선한 컬렉션 프레임워크
  - List의 대표적인 클래스다.(자주 사용 됨)
  - Object배열을 통해 데이터를 순차적으로 저장함.
  - 배열에 저장할 공간이 없을 시 큰 배열을 새로 생성하여 저장.
  - 순차적인 데이터 접근이 필요할 경우 많이 사용된다.
  - 저장순서가 유지되고 데이터의 중복을 허용한다.
  - 데이터의 저장 공간으로 배열을 사용한다.
  - vector는 멀티스레드에 대한 동기화가 되어있으나 ArrayList는 그렇지 않다.

콜렉션(Collection)

- ArrayList 관련 함수

리턴형	함수 이름 및 설명
boolean	add (E e) 리스트의 마지막으로, 지정된 요소를 추가합니다.
void	add (int index, E element) 리스트내의 지정된 위치로 지정된 요소를 삽입합니다.
boolean	addAll(Collection <? extends E > c) 지정된 컬렉션내의 모든 요소를, 지정된 컬렉션의 반복자에 의해 반환되는 순서로 리스트의 마지막에 추가합니다.
boolean	addAll (int index, Collection <? extends E > c) 지정된 컬렉션내의 모든 요소를, 리스트의 지정된 위치에 삽입합니다.
void	clear () 리스트로부터 모든 요소를 삭제합니다.
Object	clone () ArrayList 의 인스턴스의 shallow 복사를 돌려줍니다.
boolean	contains (Object o) 리스트로 지정된 요소가 있는 경우에 true 를 돌려줍니다.
void	ensureCapacity (int minCapacity) 필요에 따라서, 이 ArrayList 의 인스턴스의 사이즈를 확대해, 적어도 최소 사이즈 인수로 지정된 수의 요소를 포함할 수 있도록(듯이) 합니다.
E	get (int index) 리스트내의 지정된 위치에 있는 요소를 돌려줍니다.
int	indexOf (Object o) 지정된 요소가 리스트내에서 최초로 검출된 위치의 인덱스를 돌려줍니다.

콜렉션(Collection)

- ArrayList 관련 함수

리턴형	함수 이름 및 설명
boolean	isEmpty () 리스트에 요소가 없는 경우에 true 를 돌려줍니다.
int	lastIndexOf (Object o) 지정된 요소가 리스트내에서 마지막에 검출된 위치의 인덱스를 돌려줍니다.
E	remove (int index) 리스트의 지정된 위치에 있는 요소를 삭제합니다.
boolean	remove (Object o) 존재하는 경우는, 최초로 검출된 지정 요소를 이 리스트로부터 삭제합니다.
protected Void	removeRange (int fromIndex, int toIndex) 이 리스트로부터,fromIndex 로 시작되어,toIndex 의 직전에 끝나는 인덱스를 가지는 모든 요소를 삭제합니다.
E	set (int index, E element) 리스트의 지정된 위치에 있는 요소를, 지정된 요소로 옮겨놓습니다.
int	size () 리스트내에 있는 요소의 수를 돌려줍니다.
Object[]	toArray () 리스트내의 모든 요소를 적절한 순서 (최초의 요소로부터 마지막 요소에)로 포함하고 있는 배열을 돌려줍니다
<T> T[]	toArray (T[] a) 리스트내의 모든 요소를 적절한 순서 (최초의 요소로부터 마지막 요소에)로 포함하고 있는 배열을 돌려줍니다.
void	trimToSize () 이 ArrayList 의 인스턴스의 사이즈를 리스트의 현재의 사이즈에 축소합니다.

콜렉션(Collection)

- ArrayList 예제

```
---------- java ----------
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]
```

```
ArrayList list1 = new ArrayList(10);
list1.add(new Integer(5));
list1.add(new Integer(4));
list1.add(new Integer(2));
list1.add(new Integer(0));
list1.add(new Integer(1));
list1.add(new Integer(3));

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

Collections.sort(list1);    // list1과 list2를 정렬한다.
Collections.sort(list2);    // Collections.sort(List l)
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);

// list1에서 list2와 겹치는 부분만 남기고 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);
```

콜렉션(Collection)

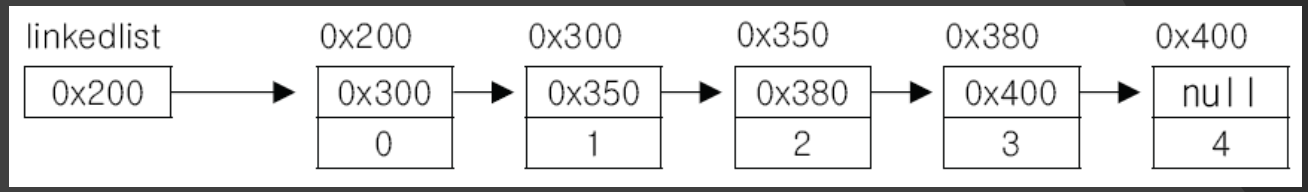
- ArrayList 장단점
  - 배열은 구조가 간단하고 데이터를 읽어오는 데 걸리는 시간(접근시간, access time)이 가장 빠르다는 장점이 있다.

콜렉션(Collection)

- ArrayList 단점
  - 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.

콜렉션(Collection)

- LinkedList
  - ArrayList의 단점을 보완한 클래스
  - 링크드리스트는 불연속적으로 존재하는 데이터를 연결한 형태로 구성
  - 각 요소(node)들은 자신과 연결된 다음 요소에 대한 참조값과 데이터로 구성되어 있다.




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

콜렉션(Collection)

- LinkedList 특징
  - ArrayList의 단점을 보완한 콜렉션 클래스
  - 링크드리스트는 불연속적으로 존재하는 데이터를 연결한 형태로 구성
  - 각 요소(node)들은 자신과 연결된 다음 요소에 대한 참조값과 데이터로 구성되어 있다.
  - 중간 데이터의 삭제나 추가가 용이 하다.
  - 대부분 메서드는 ArrayList와 크게 다르지 않다.
  - 순차적으로 추가/삭제하는 경우에는 ArrayList가 빠르며 중간 데이터를 추가/삭제 시 LinkedList가 빠르다

콜렉션(Collection)

- LinkedList 메서드

리턴형	함수 이름 및 설명
boolean	add (E e) 리스트의 마지막으로, 지정된 요소를 추가합니다.
void	add (int index, E element) 리스트내의 지정된 위치로 지정된 요소를 삽입합니다.
boolean	addAll (Collection <? extends E > c) 지정된 컬렉션내의 모든 요소를, 지정된 컬렉션의 반복자에 의해 반환되는 순서로 리스트의 마지막에 추가합니다.
boolean	addAll (int index, Collection <? extends E > c) 지정된 컬렉션내의 모든 요소를, 리스트의 지정된 위치에 삽입합니다.
void	addFirst (E e) 리스트의 선두에, 지정된 요소를 삽입합니다.
void	addLast (E e) 리스트의 마지막으로, 지정된 요소를 추가합니다.
void	clear () 리스트로부터 모든 요소를 삭제합니다.
Object	clone () LinkedList 의 shallow 복사를 돌려줍니다.
boolean	contains (Object o) 리스트로 지정된 요소가 있는 경우에 true 를 돌려줍니다.
Iterator <E>	descendingIterator () 이 양단 큐내의 요소를 역순서로 반복 처리 하는 반복자를 돌려줍니다.
E	element () 이 리스트의 선두 (최초의 요소)를 가져옵니다만, 삭제는 하지 않습니다.
E	get (int index) 리스트내의 지정된 위치에 있는 요소를 돌려줍니다.
E	getFirst () 리스트내의 최초의 요소를 돌려줍니다.

콜렉션(Collection)

- LinkedList 메서드

리턴형	함수 이름 및 설명
E	getLast () 리스트내의 마지막 요소를 돌려줍니다.
int	indexOf (Object o) 지정된 요소가 리스트내에서 최초로 검출된 위치의 인덱스를 돌려줍니다.
int	lastIndexOf (Object o) 지정된 요소가 리스트내에서 마지막에 검출된 위치의 인덱스를 돌려줍니다.
ListIterator<E>	listIterator (int index) 리스트내의 지정된 위치에서 시작되는, 리스트내의 요소를 적절한 순서로 반복하는 리스트 반복자를 돌려줍니다.
boolean	offer (E e) 지정된 요소를 이 리스트의 말미 (마지막 요소)에 추가합니다.
boolean	offerFirst (E e) 리스트의 선두에, 지정된 요소를 삽입합니다.
boolean	offerLast (E e) 리스트의 말미에, 지정된 요소를 삽입합니다.
E	peek () 이 리스트의 선두 (최초의 요소)를 가져옵니다만, 삭제는 하지 않습니다.
E	peekFirst () 리스트의 최초의 요소를 가져옵니다만, 삭제하지 않습니다.
E	peekLast () 리스트의 마지막 요소를 가져옵니다만, 삭제하지 않습니다.
E	poll () 이 리스트의 선두 (최초의 요소) 취득해, 삭제합니다.
E	pollFirst () 리스트의 최초의 요소를 취득 및 삭제합니다.
E	pollLast () 리스트의 마지막 요소를 취득 및 삭제합니다.

콜렉션(Collection)

- LinkedList 메서드

리턴형	함수 이름 및 설명
E	pop () 이 리스트가 나타내는 스택으로부터 요소를 팝 합니다.
void	push (E e) 이 리스트가 나타내는 스택상에 요소를 푸쉬 합니다.
E	remove () 이 리스트의 선두 (최초의 요소) 취득해, 삭제합니다.
E	remove (int index) 리스트의 지정된 위치에 있는 요소를 삭제합니다.
boolean	remove (Object o) 지정된 요소가 이 리스트에 있으면, 그 최초의 것을 리스트로부터 삭제합니다.
E	removeFirst () 리스트로부터 최초의 요소를 삭제해 돌려줍니다.
boolean	removeFirstOccurrence (Object o) 이 리스트내에서 최초로 검출된, 지정된 요소를 삭제합니다 (리스트를 선두로부터 말미의 방향으로 횡단(traverse) 했을 경우).
E	removeLast () 리스트로부터 마지막 요소를 삭제해 돌려줍니다.
boolean	removeLastOccurrence (Object o) 이 리스트내에서 마지막에 검출된, 지정된 요소를 삭제합니다 (리스트를 선두로부터 말미의 방향으로 횡단(traverse) 했을 경우).
E	set (int index, E element) 리스트의 지정된 위치에 있는 요소를, 지정된 요소로 옮겨놓습니다.
int	size () 리스트내에 있는 요소의 수를 돌려줍니다.
Object []	toArray () 리스트내의 모든 요소를 적절한 순서로 (최초의 요소로부터 마지막 요소에) 포함하고 있는 배열을 돌려줍니다.
<T> T[]	toArray (T[] a) 리스트내의 모든 요소를 적절한 순서로 (최초의 요소로부터 마지막 요소에) 포함하고 있는 배열을 돌려줍니다.

Enumeration, Iterator, ListIterator

- Iterator
  - 컬렉션에서 저장된 요소들에 대한 순차적인 접근을 하기 위한 클래스
  - 많이 사용이 되며 Set과 List방식에서는 Iterator 클래스로 변환할 수 있는 메소드를 제공한다
  - Map방식은 정렬방식을 Set방식으로 바꾼 다음 Iterator로 변환한다.
  - Enumeration의 발전형이 Iterator고 ListIterator는 Iterator를 상속받아 사용되는 클래스이다.
  - ListIterator는 Iterator에 이전방향에 대한 접근을 추가한 것 외엔 Iterator와 크게 다르지 않다.

리턴형	함수 이름 및 설명
boolean	hasNext () iterator안에 읽어 올 요소가 있는지 판단한다.
Object	next () iterator안에 다음 요소를 읽어온다. 해당 요소를 읽은 후에 node는 출력했던 요소로 이동한다.
void	remove() iterator에 해당 노드가 가르키고 있는 값을 제거한다.

Enumeration, Iterator, ListIterator

- Iterator 예제

```java
public class TestIterator {
    public static void main(String[] args) {
        ArrayList<Integer> al =
                new ArrayList<Integer>();
        al.add(1);
        al.add(3);
        al.add(2);
        Iterator<Integer> it1 = al.iterator();
        System.out.print("출력값 : ");
        while(it1.hasNext()){
            System.out.print(it1.next()+" ");
        }
        System.out.println();
        Iterator<Integer> it2 = al.iterator();
        System.out.print("출력값 : ");
        if(it2.hasNext()){
            System.out.print(it2.next()+" ");
        }
        if(it2.hasNext()){
            it2.next();
            it2.remove();
        }
        if(it2.hasNext()){
            System.out.print(it2.next());
        }
    }
}
```

Result)

```
출력값 : 1 3 2
출력값 : 1 2
```

Set

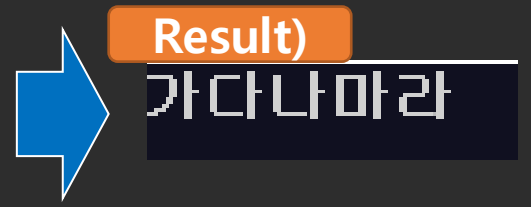
- HashSet
  - Set인터페이스를 구현한 가장 대표적인 컬렉션
  - 중복된 요소를 저장하지 않는다.
  - 새로운 요소를 추가할 때 중복된 데이터가 저장될 경우 메서드가 false를 반환하고 데이터를 추가하지 않는다.
  - 만약 저장순서를 유지하고자 한다면 LinkedHashSet을 이용한다.

리턴형	함수 이름 및 설명
boolean	add (E e) 지정된 요소가 이 세트의 요소로서 존재하지 않는 경우에, 그 요소를 세트에 추가합니다.
void	clear () 모든 요소를 세트로부터 삭제합니다.
Object	clone () HashSet 의 인스턴스의 shallow 복사를 돌려줍니다.
boolean	contains (Object o) 세트가, 지정된 요소를 보관 유지하고 있는 경우에 true 를 돌려줍니다.
boolean	isEmpty () 세트가 요소를 1 개나 보관 유지하고 있지 않는 경우에 true 를 돌려줍니다.
Iterator<E >	iterator () 세트내의 각 요소에 대한 반복자를 돌려줍니다.
boolean	remove (Object o) 지정된 요소가 이 세트에 존재하는 경우에, 요소를 세트로부터 삭제합니다.
int	size () 세트내의 요소수 (그 카디나리티)를 돌려줍니다.

Set

- HashSet 예제

```
public class StartSet {
    public static void main(String[] args) {
        Set<String> s = new HashSet<String>();
        s.add("가");
        s.add("나");
        s.add("가");//에러 : 존재하는 값
        s.add("다");
        s.add("라");
        s.add("마");
        Iterator<String> it = s.iterator();
        while(it.hasNext()){
            System.out.print(it.next());
        }
    }
}
```



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-자식관계'에 있다 하고, 하나의 부모에 최대 두 개의 자식을 갖는다.
  - 왼쪽 자식의 값은 부모의 값보다 작은 값을, 오른쪽 자식의 값은 부모보다 큰 값을 저장한다.
  - 검색과 정렬에 유리하지만, HashSet보다 데이터 추가, 삭제시간이 더 걸린다.

Set

- TreeSet 메서드

리턴형	함수 이름 및 설명
boolean	add (E e) 지정된 요소가 이 세트의 요소로서 존재하지 않는 경우에, 그 요소를 세트에 추가합니다.
boolean	addAll (Collection <? extends E > c) 지정된 컬렉션내의 모든 요소를 세트에 추가합니다.
E	ceiling (E e) 이 세트내에서, 지정된 요소와 동일한가 그것보다 큰 요소 속에서 최소의 것을 돌려줍니다.
void	clear () 모든 요소를 세트로부터 삭제합니다.
Object	clone () TreeSet 의 인스턴스의 shallow 복사를 돌려줍니다.
Comparator <? super E >	comparator () 이 세트내의 요소를 순서 붙이고 하는데 사용하는 Comparator를 돌려줍니다.
boolean	contains (Object o) 이 세트로 지정된 요소가 포함되어 있는 경우에 true 를 돌려줍니다.
Iterator <E>	descendingIterator () 이 세트의 요소의 반복자를 내림차순으로 돌려줍니다.
NavigableSet<E >	descendingSet () 이 세트에 포함되는 요소의 역순서의 뷰를 돌려줍니다.
E	first () 세트내에 현재 있는 최초 (하단)의 요소를 돌려줍니다.
E	floor (E e) 이 세트내에서, 지정된 요소와 동일한가 그것보다 작은 요소 속에서 최대의 것을 돌려줍니다.
SortedSet <E>	headSet (E toElement) 세트의 toElement 보다 작은 요소를 가지는 부분의 뷰를 돌려줍니다.

Set

- TreeSet 메서드

리턴형	함수 이름 및 설명
NavigableSet<E >	headSet (E toElement, boolean inclusive) 이 세트의 toElement 보다 작은 요소 (inclusive 가 true 의 경우는 그것보다 작은가 그것과 동일한 요소)를 포함한 부분의 뷰를 돌려줍니다.
E	higher (E e) 이 세트내에서, 지정된 요소보다 확실히 큰 요소 속에서 최소의 것을 돌려줍니다.
boolean	isEmpty () 이 세트에 요소가 포함되지 않은 경우에 true 를 돌려줍니다.
Iterator <E>	iterator () 이 세트의 요소의 반복자를 승순으로 돌려줍니다.
E	last () 세트내에 현재 있는 최후 (상단)의 요소를 돌려줍니다.
E	lower (E e) 이 세트내에서, 지정된 요소보다 확실히 작은 요소 속에서 최대의 것을 돌려줍니다.
E	pollFirst () 최초 (하단)의 요소를 취득해 삭제합니다.
E	pollLast () 최후 (상단)의 요소를 취득해 삭제합니다.
boolean	remove (Object o) 지정된 요소가 이 세트에 존재하는 경우에, 요소를 세트로부터 삭제합니다.
int	size () 세트내의 요소수 (그 카디나리티)를 돌려줍니다.
NavigableSet<E >	subSet (E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) 세트의 fromElement ~ toElement 의 요소 범위를 가지는 부분의 뷰를 돌려줍니다.

Set

- TreeSet 메서드

리턴형	함수 이름 및 설명
SortedSet <E>	subSet (E fromElement, E toElement) 세트의 fromElement (이것을 포함한다) ~ toElement (이것을 포함하지 않는다)의 요소 범위를 가지는 부분의 뷰를 돌려줍니다.
SortedSet <E>	tailSet (E fromElement) 세트의 fromElement 에 동일한가 이것보다 큰 요소를 가지는 부분의 뷰를 돌려줍니다.
NavigableSet<E >	tailSet (E fromElement, boolean inclusive) 이 세트의 fromElement 보다 큰 요소 (inclusive 가 true 의 경우는 그것보다 큰가 그것과 동일한 요소)를 포함한 부분의 뷰를 돌려줍니다.

Set

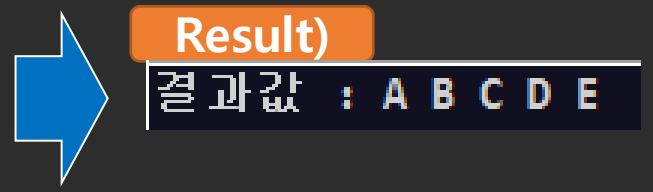
- TreeSet 메서드

```java
public class TreeSetTest {

    public static void main(String[] args) {
        Set<String> set = new TreeSet<String>();
        set.add("D");
        set.add("C");
        set.add("E");
        set.add("B");
        set.add("A");
        Iterator<String> it = set.iterator();
        System.out.print("결과값 : ");
        while(it.hasNext()){
            System.out.print(it.next()+" ");
        }

    }

}
```



Map

- HashMap
  - Map방식 컬렉션 클래스 중 가장 많이 사용된다
  - 해싱 방식으로 구현되어 있어 많은 데이터를 검색할 때 유리하다
  - HashMap은 Map인터페이스를 구현하였으며, 데이터를 키와 값의 쌍으로 저장한다.
  - HashTable의 신버전
  - 키는 겹칠 수 없으며 값은 중복이 되어도 상관이 없다.

```
public class HashMapTest {
    public static void main(String[] args) {
        HashMap<Integer,String> hm = new HashMap<Integer, String>();
        hm.put(1, "가");
        hm.put(1, "나");
        hm.put(2, "다");
        System.out.println(hm.toString());
    }
}
```

```
{1=나, 2=다}
```

Map

- HashMap 메소드

리턴형	함수 이름 및 설명
void	clear () 모든 매핑을 맵으로부터 삭제합니다.
Object	clone () HashMap 의 인스턴스의 shallow 복사를 돌려줍니다.
boolean	containsKey (Object key) 맵이 지정된 키의 매핑을 보관 유지하는 경우에 true 를 돌려줍니다.
boolean	containsValue (Object value) 맵이 1 개 또는 복수의 키와 지정된 값을 매핑 하고 있는 경우에 true 를 돌려줍니다.
Set<Map.Entry<K ,V >>	entrySet () 이 맵에 포함되는 맵 Set 뷰를 돌려줍니다.
V	get (Object key) 지정된 키가 맵 되고 있는 값을 돌려줍니다.
boolean	isEmpty () 맵이 키와 값의 매핑을 보관 유지하지 않는 경우에 true 를 돌려줍니다.
Set <K >	keySet () 이 맵에 포함되는 키 Set 뷰를 돌려줍니다.
V	put (K key, V value) 지정된 값과 지정된 키를 이 맵에 관련짓습니다.
void	putAll (Map <? extends K ,? extends V > m) 지정된 맵으로부터 모든 매핑을 맵에 카피합니다.
V	remove (Object key) 지정된 키의 매핑이 있으면 맵으로부터 삭제합니다.
int	size () 맵내의 키치 매핑의 수를 돌려줍니다.
Collection<V >	values () 이 맵에 포함되는 값 Collection 뷰를 돌려줍니다.

Map

- HashMap 예제

Ex)

```
public class HashMapTest {
    public static void main(String[] args) {
        HashMap<Integer,String> hm =
                new HashMap<Integer, String>();
        hm.put(1, "가");
        hm.put(1, "나");
        hm.put(2, "다");
        System.out.println(hm.toString());
        System.out.println(hm.keySet());
        System.out.println(hm.remove(1));
        System.out.println(hm.toString());
        hm.put(1, "가");
        hm.put(3, "라");
        System.out.println(hm.toString());
        System.out.println(hm.size());
        System.out.println(hm.get(2));
    }
}
```

Result)

```
{1=나, 2=다}
[1, 2]
나
{2=다}
{1=가, 2=다, 3=라}
3
다
```

Map

- TreeMap
  - 이진 검색트리의 형태로 키와 값이 쌍으로 이루어진 데이터를 저장한다.
  - 이진트리 방식으로 검색과 정렬에 적합한 컬렉션 클래스다
  - 검색 능력은 HashMap이 TreeMap보다 뛰어나다.
  - HashMap에서 가지고있던 기능 외에 Tree에 사용되는 메서드가 존재한다.

Ex)

```
public class TreeMao {
    public static void main(String[] args) {
        TreeMap<Integer, String> tm =
                new TreeMap<Integer, String>();
        tm.put(2, "나");
        tm.put(1, "가");
        tm.put(3, "다");
        System.out.println(tm.toString());
    }
}
```

Result)

```
{1=가, 2=나, 3=다}
```

Map

- TreeMap 메서드

리턴형	함수 이름 및 설명
Map.Entry <K,V >	ceilingEntry (K key) 지정된 키와 동일한가 그것보다 큰 키 속에서 최소의 것에 관련지을 수 있던, 키와 값의 매핑을 돌려줍니다.
K	ceilingKey (K key) 지정된 키와 동일한가 그것보다 큰 키 속에서 최소의 것을 돌려줍니다.
void	clear () 모든 매핑을 맵으로부터 삭제합니다.
Object	clone () TreeMap 의 인스턴스의 shallow 복사를 돌려줍니다.
Comparator<? super K >	comparator () 이 맵내의 키를 순서 붙이고 하는데 사용하는 Comparator를 돌려줍니다.
boolean	containsKey (Object key) 맵이 지정된 키의 매핑을 보관 유지하는 경우에 true 를 돌려줍니다.
boolean	containsValue (Object value) 맵이 1 개 또는 복수의 키와 지정된 값을 매핑 하고 있는 경우에 true 를 돌려줍니다.
NavigableSet<K >	descendingKeySet () 이 맵에 포함되는 키의 역순서 NavigableSet 뷰를 돌려줍니다.
NavigableMap<K ,V >	descendingMap () 맵내에 보관 유지되고 있는 매핑의 역순서의 뷰를 돌려줍니다.
Set<Map.Entry <K,V > >	entrySet () 이 맵에 포함되는 맵 Set 뷰를 돌려줍니다.
Map.Entry <K,V >	firstEntry () 이 맵내에서 최소의 키에 관련지을 수 있었던 키와 값의 매핑을 돌려줍니다.
K	firstKey () 맵내에 현재 있는 최초 (하단)의 키를 돌려줍니다.
Map.Entry <K,V >	floorEntry (K key) 지정된 키와 동일한가 그것보다 작은 키 속에서 최대의 것에 관련지을 수 있던, 키와 값의 매핑을 돌려줍니다.

Map

- TreeMap 메서드

리턴형	함수 이름 및 설명
K	floorKey (K key) 지정된 키와 동일한가 그것보다 작은 키 속에서 최대의 것을 돌려줍니다.
V	get (Object key) 지정된 키가 맵 되고 있는 값을 돌려줍니다.
SortedMap <K,V >	headMap (K toKey) 맵의 toKey 보다 작은 키를 가지는 부분의 뷰를 돌려줍니다.
NavigableMap<K ,V >	headMap (K toKey, boolean inclusive) toKey 보다 작은 키 (inclusive 가 true 의 경우는 그것보다 작은가 그것과 동일한 키)를 포함한 이 맵의 부분의 뷰를 돌려줍니다.
Map.Entry <K,V >	higherEntry (K key) 지정된 키보다 확실히 큰 키 속에서 최소의 것에 관련지을 수 있던, 키와 값의 매핑을 돌려줍니다.
K	higherKey (K key) 지정된 키보다 확실히 큰 키 속에서 최소의 것을 돌려줍니다.
Set <K >	keySet () 이 맵에 포함되는 키 Set 뷰를 돌려줍니다.
Map.Entry <K,V >	lastEntry () 이 맵내에서 최대의 키에 관련지을 수 있었던 키와 값의 매핑을 돌려줍니다.
K	lastKey () 맵내에 현재 있는 최후 (상단)의 키를 돌려줍니다.
Map.Entry <K,V >	lowerEntry (K key) 지정된 키보다 확실히 작은 키 속에서 최대의 것에 관련지을 수 있던, 키와 값의 매핑을 돌려줍니다.
K	lowerKey (K key) 지정된 키보다 확실히 작은 키 속에서 최대의 것을 돌려줍니다.
NavigableSet<K >	navigableKeySet () 이 맵에 포함되는 키 NavigableSet 뷰를 돌려줍니다.
Map.Entry <K,V >	pollFirstEntry () 이 맵내에서 최소의 키에 관련지을 수 있었던 키와 값의 매핑을 삭제해 돌려줍니다.

Map

- TreeMap 메서드

리턴형	함수 이름 및 설명
Map.Entry <K,V >	pollLastEntry () 이 맵내에서 최대의 키에 관련지을 수 있었던 키와 값의 매핑을 삭제해 돌려줍니다.
V	put (K key, V value) 지정된 값과 지정된 키를 이 맵에 관련짓습니다.
void	putAll (Map <? extends K ,? extends V > map) 지정된 맵으로부터 모든 매핑을 맵에 카피합니다.
V	remove (Object key) 키의 매핑이 있으면 TreeMap 로부터 삭제합니다.
int	size () 맵내의 키치 매핑의 수를 돌려줍니다.
NavigableMap<K ,V >	subMap (K fromKey, boolean fromInclusive, K toKey, boolean toInclusive) 맵의 fromKey ~ toKey 의 키 범위를 가지는 부분의 뷰를 돌려줍니다.
SortedMap <K,V >	subMap (K fromKey, K toKey) 맵의 fromKey (이것을 포함한다) ~ toKey (이것을 포함하지 않는다)의 키 범위를 가지는 부분의 뷰를 돌려줍니다.
SortedMap <K,V >	tailMap (K fromKey) 맵의 fromKey 이상의 키를 가지는 부분의 뷰를 돌려줍니다.
NavigableMap<K ,V >	tailMap (K fromKey, boolean inclusive) fromKey 보다 큰 키 (inclusive 가 true 의 경우는 그것보다 큰가 그것과 동일한 키)를 포함한 이 맵의 부분의 뷰를 돌려줍니다.
Collection<V >	values () 이 맵에 포함되는 값 Collection 뷰를 돌려줍니다.