검색 기능을 강화시킨 컬렉션

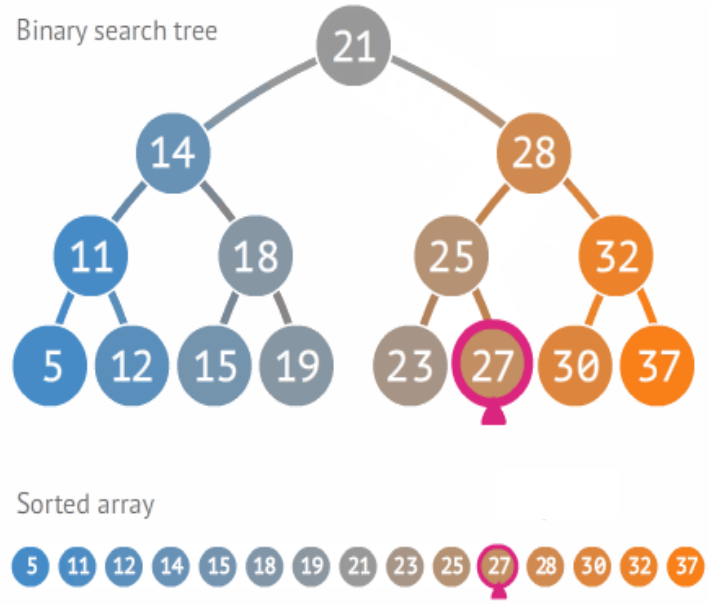
▣ 검색 기능을 강화시킨 컬렉션

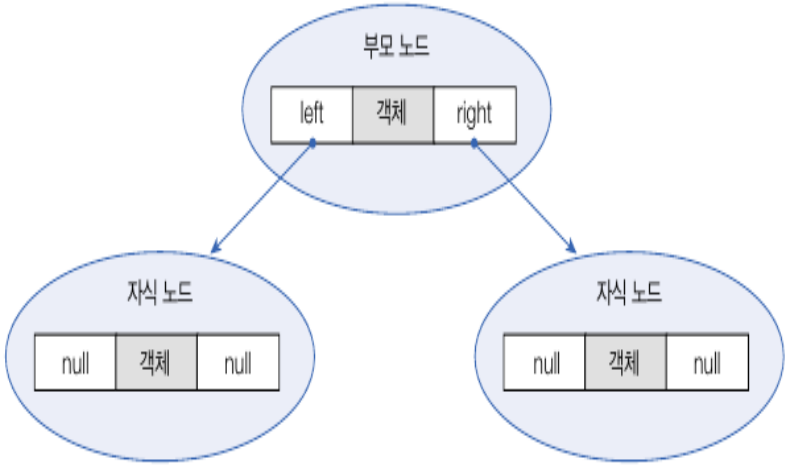
- 컬렉션 프레임워크는 검색 기능을 강화시킨 TreeSet 과 TreeMap를 제공한다.

● TreeSet

- TreeSet은 이진트리(binary tree)를 기반으로 한 Set 컬렉션이다.
- 이진 트리는 여러 개의 노드(node)가 트리형태로 연결된 구조로, 루트 노드(root node)라고 불리는 하나의 노드에서 시작해서 각 노드에 최대 2개의 노드를 연결할 수 있는 구조를 가지고 있다.

- TreeSet에 객체를 저장하면 다음과 같이 자동으로 정렬된다.
- 부모노드의 객체와 비교해서 낮은 것은 왼쪽 자식노드에, 높은 것은 오른쪽 자식노드에 저장한다.

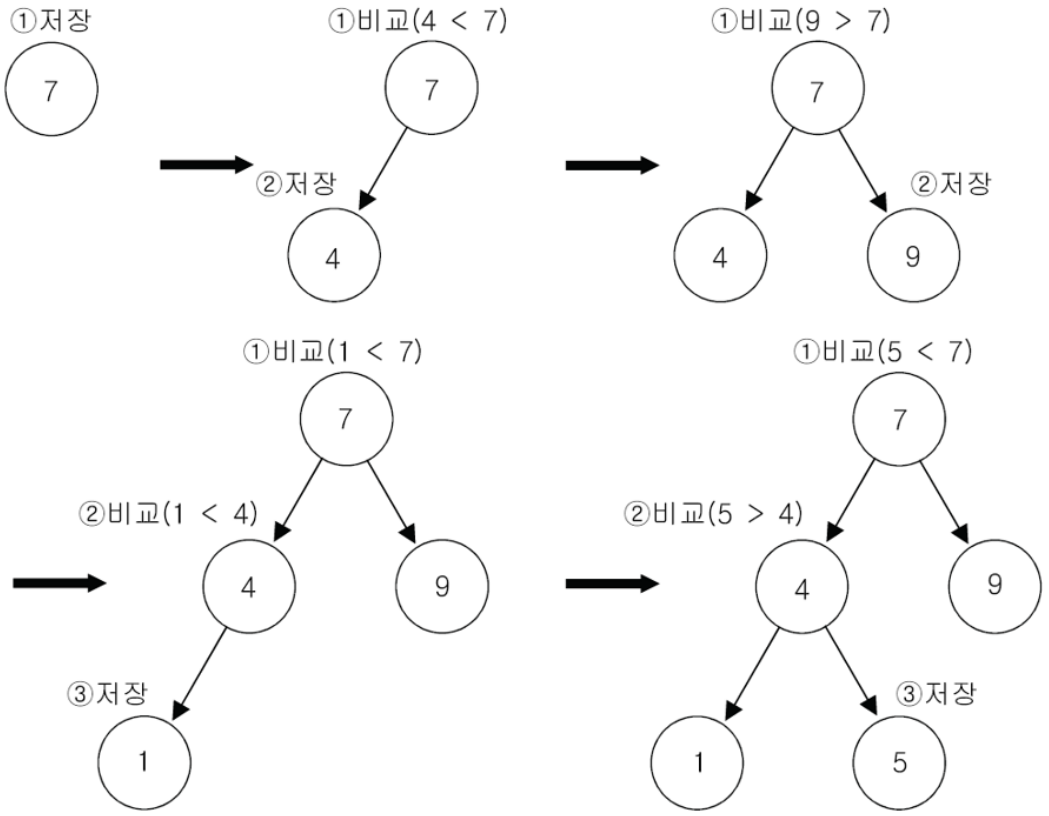





```
TreeSet<E> treeSet = new TreeSet<E>();
TreeSet<E> treeSet = new TreeSet<>();
```

boolean add(Object o)

TreeSet에 7,4,9,1,5의 순서로 데이터를 저장하면, 아래의 과정을 거친다.
(루트부터 트리를 따라 내려가며 값을 비교. 작으면 왼쪽, 크면 오른쪽에 저장)

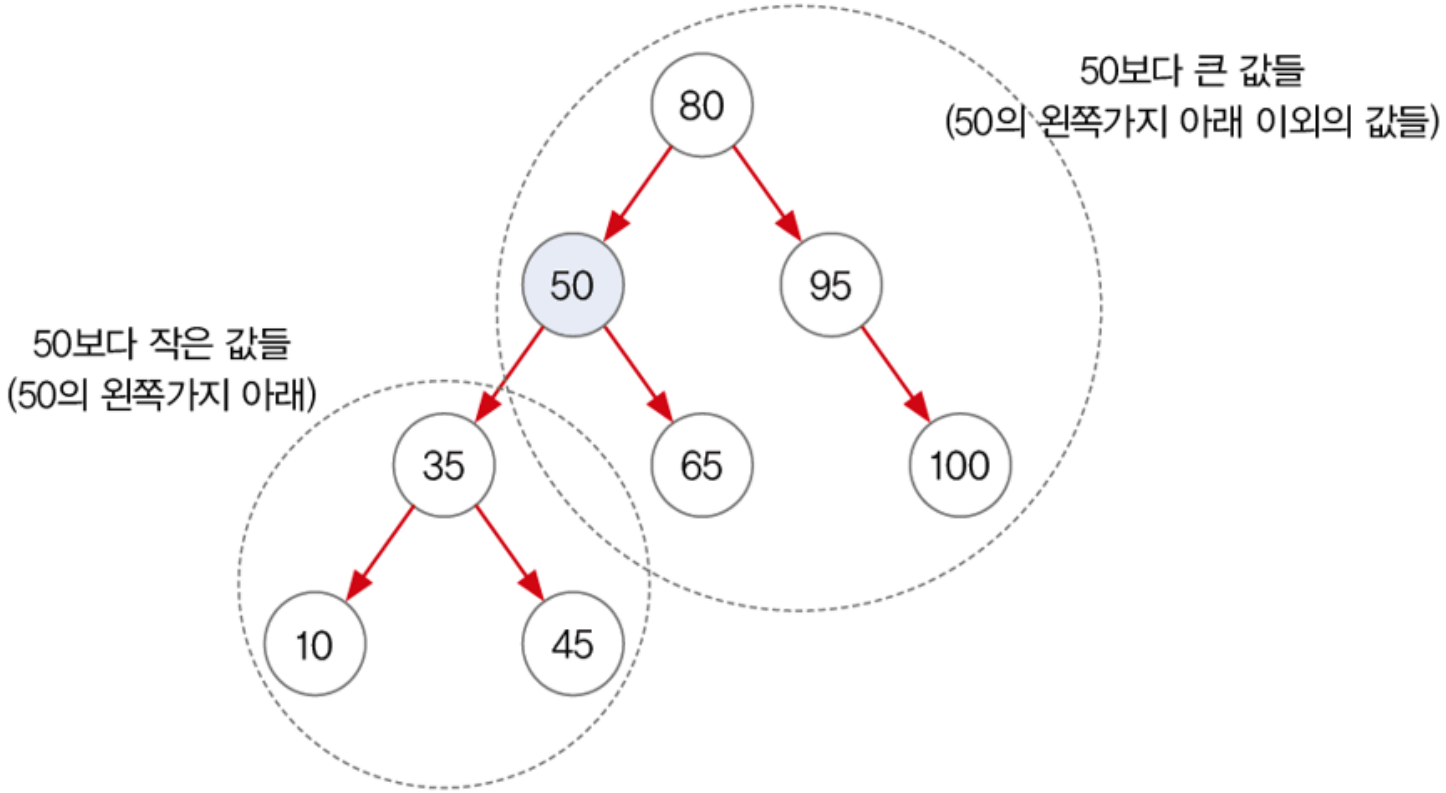


- Set 타입 변수에 대입해도 되지만,
  TreeSet 타입으로 대입한 이유는 검색 관련 메소드가 TreeSet에만 정의되어 있기 때문이다.

리턴 타입	메소드	설명
E	first( )	제일 낮은 객체를 리턴
E	last( )	제일 높은 객체를 리턴
E	lower(E e)	주어진 객체보다 바로 아래 객체를 리턴
E	higher(E e)	주어진 객체보다 바로 위 객체를 리턴
E	floor(E e)	주어진 객체와 동등한 객체가 있으면 리턴, 만약 없다면 주어진 객체의 바로 아래의 객체를 리턴
E	ceiling(E e)	주어진 객체와 동등한 객체가 있으면 리턴, 만약 없다면 주어진 객체의 바로 위의 객체를 리턴
E	pollFirst( )	제일 낮은 객체를 꺼내오고 컬렉션에서 제거함
E	pollLast( )	제일 높은 객체를 꺼내오고 컬렉션에서 제거함
Iterator<E>	descendingIterator( )	내림차순으로 정렬된 Iterator를 리턴
NavigableSet<E>	descendingSet( )	내림차순으로 정렬된 NavigableSet을 리턴
NavigableSet<E>	headSet(E toElement, boolean inclusive)	주어진 객체보다 낮은 객체들을 NavigableSet으로 리턴. 주어진 객체 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableSet<E>	tailSet(E fromElement, boolean inclusive)	주어진 객체보다 높은 객체들을 NavigableSet으로 리턴. 주어진 객체 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableSet<E>	subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	시작과 끝으로 주어진 객체 사이의 객체들을 NavigableSet으로 리턴. 시작과 끝 객체의 포함 여부는 두 번째, 네 번째 매개값에 따라 달라짐

자바에서 제공하는 NavigableSet 인터페이스의 대표적인 구현 클래스는 TreeSet 입니다.
따라서 TreeSet 객체를 생성 후에, NavigableSet 타입 변수에 할당하기면 하면 됩니다.
descendingSet 메소를 이용하여, 역순으로 정렬된 새로운 Set을 얻을 수도 있습니다.

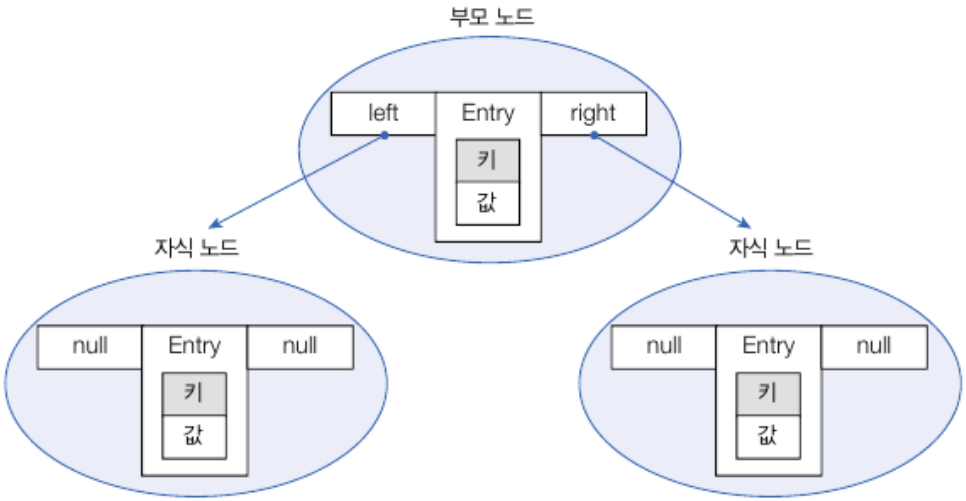
메서드	설 명
SortedSet subSet(Object fromElement, Object toElement)	범위 검색(fromElement와 toElement사이)의 결과를 반환한다.(끝 범위인 toElement는 범위에 포함되지 않음)
SortedSet headSet(Object toElement)	지정된 객체보다 작은 값의 객체들을 반환한다.
SortedSet tailSet(Object fromElement)	지정된 객체보다 큰 값의 객체들을 반환한다.



● TreeMap

- TreeMap은 이진 트리를 기반으로 한 Map 컬렉션이다.
- TreeSet과의 차이점은 키와 값이 저장된 Entry를 저장한다는 점이다.

- TreeMap에 엔트리를 저장하면 키를 기준으로 자동 정렬되는데,
  부모 키 값과 비교해서 낮은 것은 왼쪽, 높은 것은 오른쪽 자식 노드에 Entry 객체를 저장한다.




```
TreeMap<K, V> treeMap = new TreeMap<K, V>();
TreeMap<K, V> treeMap = new TreeMap<>();
```

- Map 타입 변수에 대입해도 되지만,
  TreeMap 타입으로 대입한 이유는 검색 관련 메소드가 TreeMap에만 정의되어 있기 때문이다.

리턴 타입	메소드	설명
Map.Entry<K,V>	firstEntry( )	제일 낮은 Map.Entry를 리턴
Map.Entry<K,V>	lastEntry( )	제일 높은 Map.Entry를 리턴
Map.Entry<K,V>	lowerEntry(K key)	주어진 키보다 바로 아래 Map.Entry를 리턴
Map.Entry<K,V>	higherEntry(K key)	주어진 키보다 바로 위 Map.Entry를 리턴
Map.Entry<K,V>	floorEntry(K key)	주어진 키와 동등한 키가 있으면 해당 Map.Entry를 리턴, 없다면 주어진 키 바로 아래의 Map.Entry를 리턴
Map.Entry<K,V>	ceilingEntry(K key)	주어진 키와 동등한 키가 있으면 해당 Map.Entry를 리턴, 없다면 주어진 키 바로 위의 Map.Entry를 리턴
Map.Entry<K,V>	pollFirstEntry( )	제일 낮은 Map.Entry를 꺼내오고 컬렉션에서 제거함
Map.Entry<K,V>	pollLastEntry( )	제일 높은 Map.Entry를 꺼내오고 컬렉션에서 제거함
NavigableSet<K>	descendingKeySet( )	내림차순으로 정렬된 키의 NavigableSet을 리턴
NavigableMap<K,V>	descendingMap( )	내림차순으로 정렬된 Map.Entry의 NavigableMap을 리턴
NavigableMap<K,V>	headMap(K toKey, boolean inclusive)	주어진 키보다 낮은 Map.Entry들을 NavigableMap으로 리턴. 주어진 키의 Map.Entry 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableMap<K,V>	tailMap(K fromKey, boolean inclusive)	주어진 객체보다 높은 Map.Entry들을 NavigableSet으로 리턴. 주어진 객체 포함 여부는 두 번째 매개값에 따라 달라짐
NavigableMap<K,V>	subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)	시작과 끝으로 주어진 키 사이의 Map.Entry들을 NavigableMap 컬렉션으로 반환. 시작과 끝 키의 Map.Entry 포함 여부는 두 번째, 네 번째 매개값에 따라 달라짐

☆ Comparable (기본 정렬기준을 구현하는데 사용) – this 객체와 인자로 받은 객체를 비교

- TreeSet에 저장되는 객체와 TreeMap에 저장되는 키 객체는 저장과 동시에 오름차순으로 정렬되는데, 어떤 객체든 정렬될 수 있는 것은 아니고 개체가 Comparable 인터페이스를 구현하고 있어야 가능하다.

- Integer, Double, String 타입은 모두 Comparable을 구현하고 있기 때문에 상관 없지만, 사용자 정의 객체를 저장할 때에는 반드시 Comparable을 구현하고 있어야 한다.

- Comparable 인터페이스에는 compareTo() 메소드가 정의되어 있다. 따라서 사용자 정의 클래스에서 이 메소드를 재정의해서 비교 결과를 정수 값으로 리턴해야 한다.

리턴 타입	메소드	설명
int	compareTo(T o)	주어진 객체와 같으면 0을 리턴 주어진 객체보다 적으면 음수를 리턴 주어진 객체보다 크면 양수를 리턴

```
public class Sample_String {
    public static void main(String[] args) {
        String str = "a";
        System.out.println("str.compareTo("b"));

        str = "b";
        System.out.println("str.compareTo("a"));

        str = "b";
        System.out.println("str.compareTo("b"));
    }
}

[Output]
-1
1
0
```

☆ Comparator (기본 정렬기준 외에 다른 기준으로 정렬하고자할때 사용) – 인자로 받은 두 객체를 비교

- 비교 기능이 있는 Comparable 구현 객체를 TreeSet에 저장하거나 TreeMap의 키로 저장하는 것이 원칙이지만,

- 비교 기능이 없는 Comparable 비구현 객체를 저장하고 싶다면,
  TreeSet과 TreeMap을 생성할 때 비교자(Comparator)를 제공하면 된다.

- 비교자는 Comparator 인터페이스를 구현한 객체를 말하는데, Comparator 인터페이스에는 compare() 메소드가 정의되어 있다. 비교자는 이 메소드를 재정의해서 비교 결과를 정수 값으로 리턴하면 된다.

```
TreeSet<E> treeSet = new TreeSet<E>( new ComparatorImpl() );
                                     비교자
TreeMap<K,V> treeMap = new TreeMap<K,V>( new ComparatorImpl() );
```

리턴 타입	메소드	설명 (왼쪽 o1기준으로)
int	compare(T o1, T o2)	o1과 o2가 동등하다면 0을 리턴 o1이 o2보다 앞에 오게 하려면 음수를 리턴 o1이 o2보다 뒤에 오게 하려면 양수를 리턴

```
import java.lang.Integer;
public class example {
    public static void main(String args[])
    {
        int x = 20;
        int y = 30;
        System.out.println(Integer.compare(x, y));

        int p = 40;
        int q = 40;
        System.out.println(Integer.compare(p, q));

        int r = 20;
        int s = 7;
        System.out.println(Integer.compare(r, s));
    }
}

[Output]
-1
0
1
```