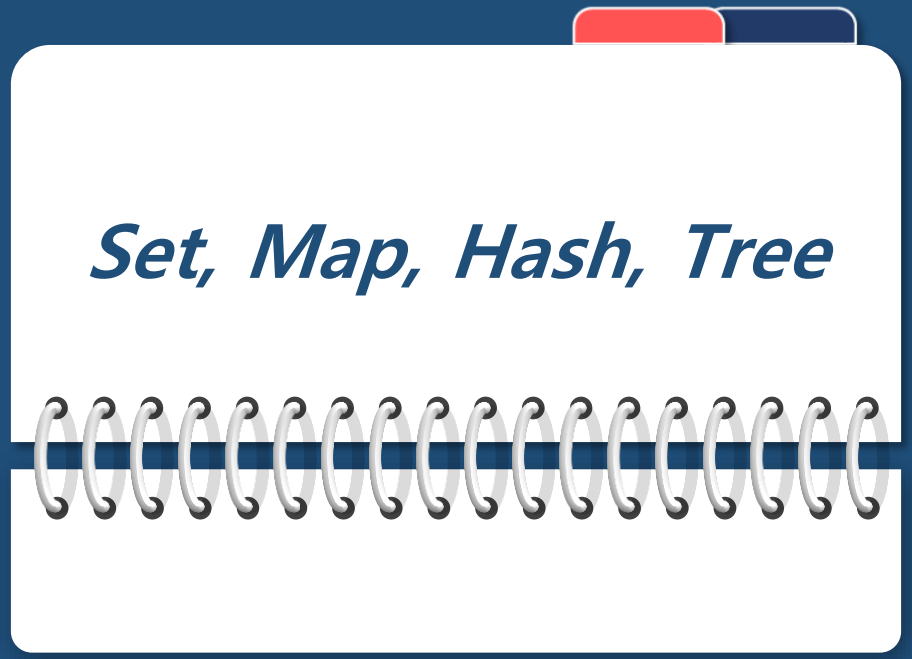

Set, Map, Hash, Tree

Set , Map

	Set	Map
자료 형태	Value 만 존재	Key, Value 쌍으로 존재
중복 여부	중복 불가	Key 중복 불가
contains	contains(value)	containsKey(key)
get	불가	get(key)

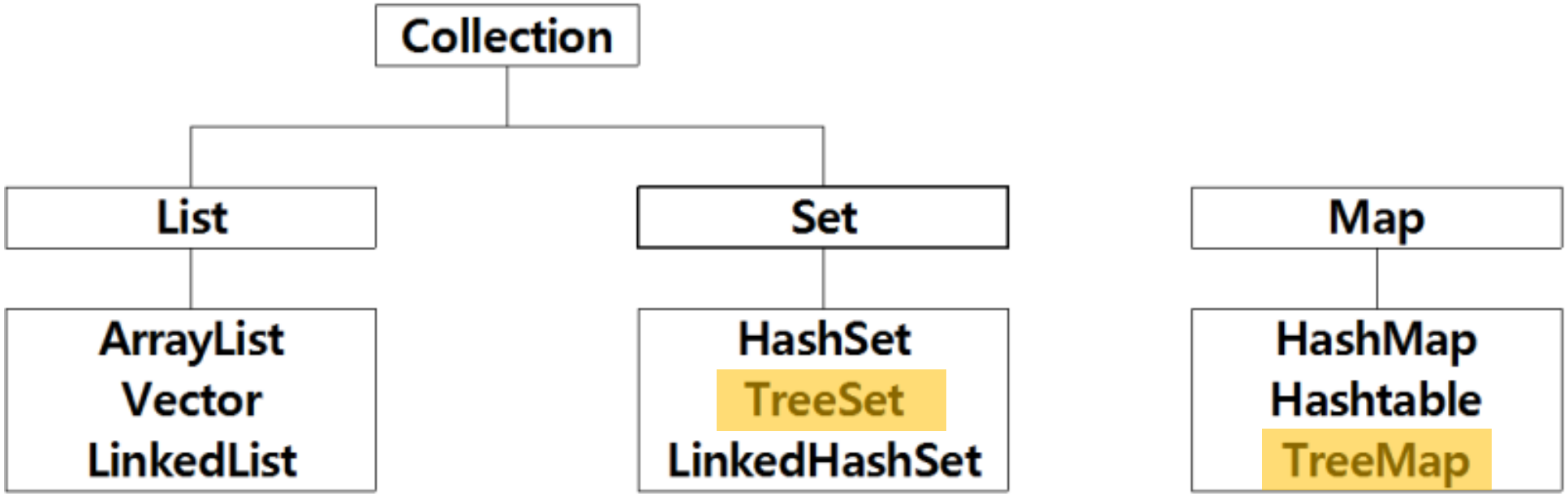
- Set은 특정 요소를 get 하려면 iterator를 통해 얻어야 한다.

Hash, Tree

	Hash	Tree
순서	순서 없음	정렬 순서 유지
시간 복잡도	O(1)	O(log n)

- Hash는 순서를 유지하지 않지만, 빠른 시간을 보장.
- Tree는 순서를 유지해서 약간 느리다.
- 속도 : Hash, 정렬 : Tree 로 보고 선택하면 된다.

- HashSet : 속도가 빠르고, Value만 존재, 존재 여부만 판별 가능
- HashMap : 속도가 빠르고, Key·Value 존재, get 가능
- TreeSet : 정렬 순서 유지, Value만 존재, 존재 여부만 판별 가능
- TreeMap : 정렬 순서 유지, Key·Value 존재, get 가능


Collection
List
Set
Map
ArrayList
Vector
LinkedList
HashSet
TreeSet
LinkedHashSet
HashMap
Hashtable
TreeMap


TreeSet

TreeSet 사용

TreeSet 선언

```
TreeSet<Integer> set1 = new TreeSet<Integer>();//TreeSet생성
TreeSet<Integer> set2 = new TreeSet<>();//new에서 타입 파라미터 생략가능
```

-> Collections.reverseOrder() 메서드를 사용하면 내림차순 정렬로 바꿀 수 있다.

TreeSet 값추가 : .add()

```
TreeSet<Integer> set = new TreeSet<Integer>();//TreeSet생성
set.add(7); //값추가
set.add(4);
set.add(9);
set.add(1);
set.add(5);
```

TreeSet 사용

TreeSet 값 삭제 : .remove() / .clear()

```
TreeSet<Integer> set = new TreeSet<Integer>();//TreeSet생성
set.remove(1);//값 1 제거
set.clear();//모든 값 제거
```

-> .remove() : 값이 존재하면 삭제 후 true, 없으면 예외 발생
-> .clear() : 모든 값을 제거

TreeSet 크기 구하기 : .size()

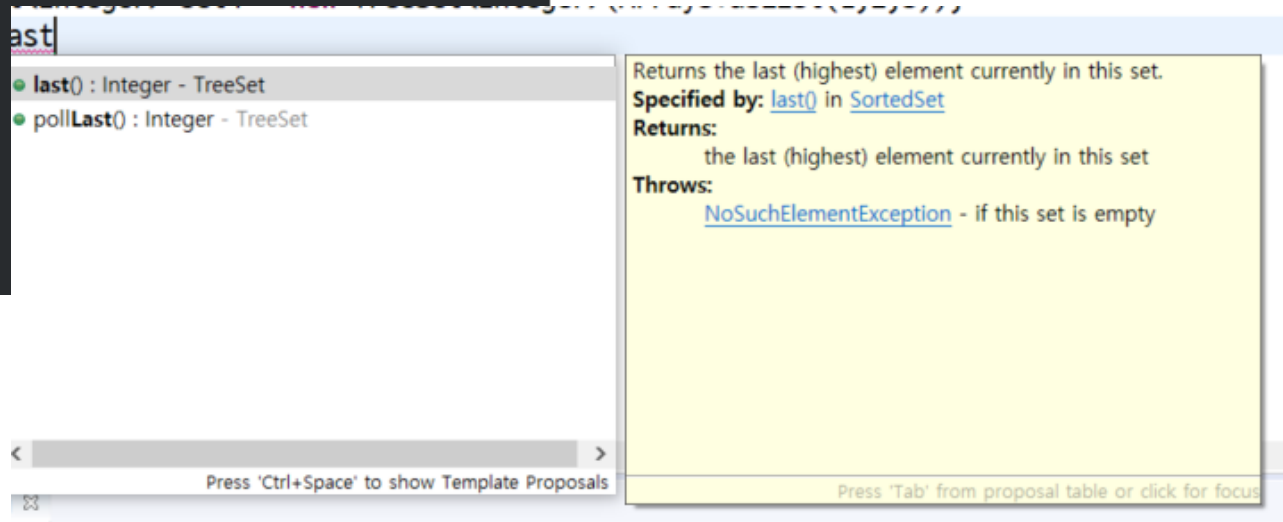
```
TreeSet<Integer> set = new TreeSet<Integer>();//TreeSet생성
set.add(7); //값추가
set.add(4);
set.add(9);
set.add(1);
set.add(5);
```

TreeSet 사용

TreeSet 값 출력 : .first() / .last() / .higher() / .lower()

```
TreeSet<Integer> set = new TreeSet<Integer>(Arrays.asList(4,2,3));//초기값 지정
System.out.println(set); //전체출력 [2,3,4]
System.out.println(set.first());//최소값 출력
System.out.println(set.last());//최대값 출력
System.out.println(set.higher(3));//입력값보다 큰 데이터중 최소값 출력 없으면 null
System.out.println(set.lower(3));//입력값보다 작은 데이터중 최대값 출력 없으면 null

Iterator iter = set.iterator(); // Iterator 사용
while(iter.hasNext()) {//값이 있으면 true 없으면 false
    System.out.println(iter.next());
}
```



-> .first() : 최소값 출력
-> .last() : 최대값 출력
-> .higher() : 입력값보다 큰 데이터 중 최소값
-> .lower() : 입력값보다 작은 데이터 중 최대값

-> iterator : 객체 전체를 대상으로 한 번씩 반복해서 가져오는 반복자.
.next() 나 .hasNext() 등과 함께 이용

TreeMap

TreeMap

TreeMap ?

-> TreeMap에 객체를 저장하면 자동으로 정렬된다.

-> 키는 저장과 동시에 자동 오름차순으로 정렬된다.
-> 숫자 타입일 경우에는 값으로, 문자열 타입일 경우에는 유니코드로 정렬.

-> 정렬 순서
부모 키 값과 비교해서 키 값이 낮은 것은 왼쪽 자식 노드,
키 값이 높은 것은 오른쪽 자식 노드에 저장.

TreeSet 사용

TreeSet 선언

```
TreeMap<Integer,String> map1 = new TreeMap<Integer,String>();//TreeMap생성
TreeMap<Integer,String> map2 = new TreeMap<>();//new에서 타입 파라미터 생략가능
TreeMap<Integer,String> map3 = new TreeMap<>(map1);//map1의 모든 값을 가진 TreeMap생성
TreeMap<Integer,String> map6 = new TreeMap<Integer,String>(){{//초기값 설정
    put(1,"a");
}};
```

-> 선언 시 크기 지정을 할 수 없다.

TreeSet 값추가 : .put()

```
TreeMap<Integer,String> map = new TreeMap<Integer,String>();//TreeMap생성
map.put(1, "사과");//값 추가
map.put(2, "복숭아");
map.put(3, "수박");
```

TreeMap 사용

TreeSet 값 삭제 : .remove() / .clear()

```
TreeSet<Integer> set = new TreeSet<Integer>();//TreeSet생성
set.remove(1);//값 1 제거
set.clear();//모든 값 제거
```

-> .remove() : 값이 존재하면 삭제 후 true, 없으면 예외 발생
-> .clear() : 모든 값을 제거

TreeSet 크기 구하기 : .size()

```
TreeSet<Integer> set = new TreeSet<Integer>();//TreeSet생성
set.add(7); //값추가
set.add(4);
set.add(9);
set.add(1);
set.add(5);
```

TreeMap 사용

TreeMap 값 출력 : .get(key) / .firstEntry() / .firstKey() / .lastEntry() / .lastKey()

```
TreeMap<Integer,String> map = new TreeMap<Integer,String>(){{//초기값 설정
    put(1, "사과");//값 추가
    put(2, "복숭아");
    put(3, "수박");
}};

System.out.println(map); //전체 출력 : {1=사과, 2=복숭아, 3=수박}
System.out.println(map.get(1));//key값 1의 value얻기 : 사과
System.out.println(map.firstEntry());//최소 Entry 출력 : 1=사과
System.out.println(map.firstKey());//최소 Key 출력 : 1
System.out.println(map.lastEntry());//최대 Entry 출력: 3=수박
System.out.println(map.lastKey());//최대 Key 출력 : 3
```

-> .get(key) : 특정 key 값의 value
-> .firstEntry() : 최소 entry 값
-> .firstKey() : 최소 Key 값
-> .lastEntry() : 최대 Entry 값
-> .lastKey() : 최대 Key 값

Entry : 키=value