**HashMap과 HashSet의 주요 차이점**

**해시맵, HashMap**

키-값 쌍을 저장하는 컬렉션 클래스

각 키가 고유한 값에 매핑되는 키를 기반으로 데이터를 저장하고 검색

해시 코드를 사용하여 데이터를 인덱싱하므로 대부분의 작업에 대해 일정한 시간 성능을 제공

키가 해시맵에 추가되면 해시 코드가 계산되어 데이터를 인덱싱

동일한 해시 코드(충돌이라고 함)를 가진 키가 여러 개 있으면 인덱스된 위치의 링크된 목록에 저장

**해시셋, HashSet**

고유한 요소를 저장하는 컬렉션 클래스

중복 요소를 허용하지 않으며 각 요소는 해시 코드를 기반으로 저장

요소가 HashSet에 추가되면 해당 요소의 해시 코드가 계산되어 내부 데이터 구조에서 해당 요소의 위치를 결정

동일한 해시 코드를 가진 요소가 여러 개 있으면 인덱스된

위치의 연결된 목록에 저장

요소의 존재를 추가, 제거 및 테스트하는 방법을 제공

**사용법**

* **해시맵**

일반적으로 키-값 쌍을 저장하여 키를 통한 캐싱, 인덱싱 및 데이터 저장에 자주 사용

* **해시셋**

고유한 요소를 저장하는 데 사용되며 종종 컬렉션에서 요소의 존재를 확인하는 데 사용

**2. 특정 작업의 성능**

해시맵

키를 사용하여 데이터를 인덱싱하므로 키를 기반으로 특정 요소를 검색하는 것에 최적화

* 해시셋

컬렉션에서 요소의 존재를 확인하는 데 최적화

* **HashSet 함수**
* add() : Data 삽입할 때 사용
* next() : Data 추출할 때 사용
* remove() : Data를 삭제할 때 사용
* contains() : Data의 포함여부를 알기 위해 사용
* size() : HashSet의 요소 개수를 얻어낼 때 사용

**HashMap 함수**

* 객체 생성 : Map<String, Integer> map = new HashMap<String, Integer>();
* put(key,value) : Data 삽입할 때 사용
* get(key,value) : Data를 추출할 때 사용, argument값은 Key를 사용
* void clear() : Map의 모든 객체를 삭제 boolean
* containsKey(Object key) : 지정된 key 객체와 일치하는 Map의 key객체가 있는지 확인
* boolean containsValue(Object value) : 지정된 value 객체와 일치하는 Map의 value객체가 있는지 확인
* Set entrySet() : Map에 저장되어 있는 key-value쌍을 Map.Entry 타입의 객체로 저장한 "Set"으로 반환
* boolean equals(Object o) : 동일한 Map인지 비교

Object get(Object key) : 지정한 key객체에 대응하는 value객체를 찾아서 반환

* int hashCode() : 해시코드를 반환
* boolean isEmpty() : Map이 비어있는지 확인
* Set keySet() : Map에 저장된 모든 key객체를 반환한다.
* Object put(Object key, Object Value) : Map에 value객체를 key객체에 연결(mapping)하여 저장
* void putAll(Map t) : 지정된 Map의 모든 key-value쌍을 추가
* Object remove(Object key) : 지정한 key객체와 일치하는 key-value 객체를 삭제
* int size() : Map에 저장된 key-value쌍의 개수를 반환
* Collection values() : Map에 저장된 모든 value 객체를 반환

**TreeMap이란?**

TreeMap은 이진트리를 기반으로 한 Map 컬렉션입니다. 같은 Tree구조로 이루어진 TreeSet과의 차이점은 TreeSet은 그냥 값만 저장한다면 TreeMap은 키와 값이 저장된 Map, Etnry를 저장한다는 점입니다. TreeMap에 객체를 저장하면 자동으로 정렬되는데, 키는 저장과 동시에 자동 오름차순으로 정렬되고 숫자 타입일 경우에는 값으로, 문자열 타입일 경우에는 유니코드로 정렬합니다. 정렬 순서는 기본적으로 부모 키값과 비교해서 키 값이 낮은 것은 왼쪽 자식 노드에 키값이 높은 것은 오른쪽 자식 노드에 Map.Etnry 객체를 저장합니다. TreeMap은 일반적으로 Map으로써의 성능이 HashMap보다 떨어집니다. TreeMap은 데이터를 저장할 때 즉시 정렬하기에 추가나 삭제가 HashMap보다 오래 걸립니다. 하지만 정렬된 상태로 Map을 유지해야 하거나 정렬된 데이터를 조회해야 하는 범위 검색이 필요한 경우 TreeMap을 사용하는 것이 효율성면에서 좋습니다.

**TreeMap 사용법**

**TreeMap 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"); }};

TreeMap을 생성하기 위해서는 키로 저장할 객체타입과 값으로 저장할 객체 타입을 타입 파라미터로 주고 기본 생성자를 호출하면 됩니다. 생성하는 명령어는 HashMap과 크게 다르지 않으나 선언 시 크기를 지정해줄 수는 없습니다.

**TreeMap 값 추가**

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

map.put(2, "복숭아");

map.put(3, "수박");

TreeMap은 구조만 HashMap과 다를뿐이지 기본적으로 Map인터페이스를 같이 상속받고 있으므로 기본적인 메소드의 사용법 자체는 HashMap과 동일합니다. TreeMap에 값을 추가하려면 put(key, value) 메소드를 사용하면 됩니다. TreeMap의 타입 파라미터와 같은 타입의 key와 value값을 넣어야 정상적으로 값이 input 되며 만약 입력되는 키 값이 TreeMap 내부에 존재한다면 기존의 값은 새로 입력되는 값으로 대치됩니다.

**TreeMap 값 삭제**

TreeMap<Integer, String> map = new TreeMap<Integer,String>(){{//초기값 설정

put(1, "사과");//값 추가

put(2, "복숭아");

put(3, "수박"); }};

map.remove(1); //key값 1 제거

map.clear(); //모든 값 제거

TreeMap에 값을 제거하려면 remove(key) 메소드를 사용하면 됩니다. 오직 키값으로만 TreeMap의 요소를 삭제할 수 있습니다. 모든 값을 제거하려면 clear() 메소드를 사용하면 됩니다.

**TreeMap 단일 값 출력**

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));//key값1의value얻기:사과 System.out.println(map.firstEntry());//최소Entry출력:1=사과 System.out.println(map.firstKey());//최소 Key 출력 : 1

System.out.println(map.lastEntry());//최대Entry출력:3=수박 System.out.println(map.lastKey());//최대 Key 출력 : 3

TreeMap을 그냥 print하게 되면 {}로 묶어 Map의 전체 key값, value가 출력됩니다. 특정 key값의 value를 가져오고 싶다면 get(key)를 사용하면 됩니다. 그리고 TreeMap은 HashMap과는 달리 Tree구조로 이루어져 있기에 항상 정렬이 되어있어 최솟값, 최댓값을 바로 가져오는 다양한 메소드를 지원합니다. firstEntry는 최소 Entry값, firstKey는 최소 Key값, lastEntry는 최대 Entry값, lastKey는 최대 Key값을 리턴합니다.

**TreeSet 이란?**

TreeSet은 Set 인터페이스를 구현한 클래스로써 객체를 중복해서 저장할 수 없고 저장 순서가 유지되지 않는다는 Set의 성질을 그대로 가지고 있다.

TreeSet은 이진 탐색 트리(BinarySearchTree)의 구조로 이루어져 있다. 이진 탐색 트리는 추가와 삭제에는 시간이 조금 더 걸리지만 정렬, 검색에 높은 성능을 보이는 자료구조다.

TreeSet은 이진탐색트리의 형태로 데이터를 저장하기에 기본적으로 nature ordering을 지원하며 생성자의 매개변수로 Comparator 객체를 입력하여 정렬 방법을 임의로 지정해 줄 수도 있다.

이터레이터

**Iterator<데이터타입> iterator명 = 컬렉션.iterator();**

먼저 컬렉션 프레임워크는 List, Set, Map, Queue 등을 말한다.

장점

**Iterator는 모든 컬렉션 프레임워크에 공통으로 사용 가능**

**컬렉션 프레임워크에서 쉽게 값을 가져오고 제거할 수 있음**

**3개의 메소드만 알면 되어서 사용하기 매우 쉽다**

단점

**처음부터 끝까지의 단방향 반복만 가능**

**값은 변경하거나 추가가 불가능**

**대량의 데이터를 제어할 때 속도가 느림**

메소드

**Iterator.hasNext()**

- Iterator 안에 다음 값이 들어있는지 확인

들었으면 true, 안들었음 false

**Iterator.next()**

- iterator의 다음 값 가져오기

**Iterator.remove()**

- iterator에서 next() 시에 가져왔던 값을

컬렉션(List, Set, Map) 등에서 삭제

반드시 next() 후에 사용해야 함

Set과 사용

Iterator iter = set.iterator();// Iterator 사용 while(iter.hasNext()) {//값이 있으면 true 없으면 false

System.out.println(iter.next()); }