컬렉션

컴퓨터공학전공
박요한

수업내용

- 컬렉션 프레임워크의 이해
- Collection<E> 인터페이스
  - ✓List
  - ✓Set
  - ✓Queue
- Map(K,V) 인터페이스
  - ✓Hash

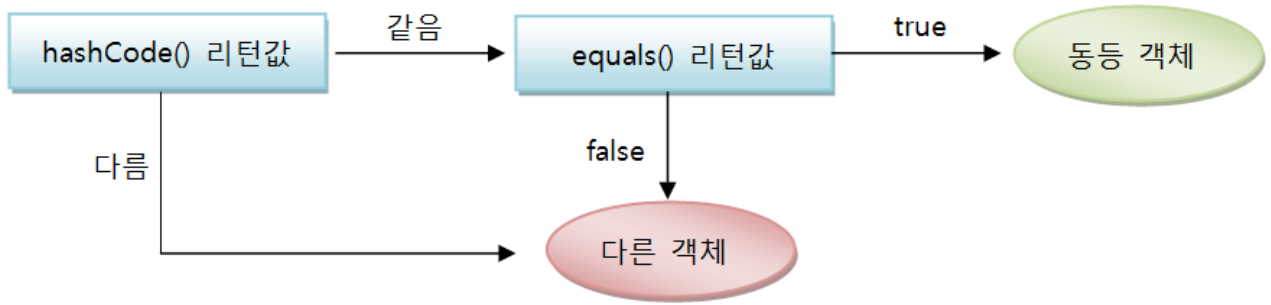
Set 컬렉션

- HashSet

```
Set<E> set = new HashSet<E>();
```

✓특징

- 동일 객체 및 동등 객체는 중복 저장하지 않음
- 동등 객체 판단 방법



HashSet<E>

- List<E>를 구현하는 클래스들과 달리 Set<E>를 구현하는 클래스들은 데이터의 저장순서를 유지하지 않는다.
- List<E>를 구현하는 클래스들과 달리 Set<E>를 구현하는 클래스들은 데이터의 중복저장을 허용하지 않는다. 단, 동일 데이터에 대한 기준은 프로그래머가 정의
- 즉, Set<E>를 구현하는 클래스는 '집합'의 성격을 지닌다.

```
public static void main(String[] args)
{
    HashSet<String> hSet=new HashSet<String>();
    hSet.add("First");
    hSet.add("Second");
    hSet.add("Third");
    hSet.add("First");

    System.out.println("저장된 데이터 수 : "+hSet.size());

    Iterator<String> itr=hSet.iterator();
    while(itr.hasNext())
        System.out.println(itr.next());
}
```

동일한 문자열 인스턴스는 저장되지 않았다. 그렇다면 동일 인스턴스를 판단하는 기준은?

실행결과

```
저장된 데이터 수 : 3
Third
Second
First
```

HashSet<E> 동일 인스턴스 판단 기준

```
class SimpleNumber
{
    int num;
    public SimpleNumber(int n)
    {
        num=n;
    }
    public String toString()
    {
        return String.valueOf(num);
    }
}
```

HashSet<E> 클래스의 인스턴스 동등비교 방법

Object 클래스에 정의되어 있는 equals 메소드의 호출결과와 hashCode 메소드의 호출결과를 참조하여 인스턴스의 동등비교를 진행

```
public static void main(String[] args)
{
    HashSet<SimpleNumber> hSet=new HashSet<SimpleNumber>();
    hSet.add(new SimpleNumber(10));
    hSet.add(new SimpleNumber(20));
    hSet.add(new SimpleNumber(20));

    System.out.println("저장된 데이터 수 : "+hSet.size());

    Iterator<SimpleNumber> itr=hSet.iterator();
    while(itr.hasNext())
        System.out.println(itr.next());
}
```

실행결과

```
저장된 데이터 수 : 3
20
10
20
```

실행결과를 보면, 동일 인스턴스의 판단기준이 별도로 존재함을 알 수 있다.

HashSet<E> 동일 인스턴스 판단 기준

- 검색 1단계
  Object 클래스의 hashCode 메소드의 반환 값을 해시 값으로 활용하여 검색의 그룹을 선택한다.
- 검색 2단계
  그룹내의 인스턴스를 대상으로 Object 클래스의 equals 메소드의 반환 값의 결과로 동등을 판단

HashSet<E>의 인스턴스에 데이터의 저장을 명령하면, 우선 다음의 순서를 거치면서 동일 인스턴스가 저장되었는지를 확인한다.

따라서 아래의 두 메소드 적절히 오버라이딩 해야 함.

```
public int hashCode( )
public boolean equals(Object obj)
```

hashCode 메소드의 구현에 따라서 검색의 성능이 달라진다. 그리고 동일 인스턴스를 판단하는 기준이 맞게 equals 메소드를 정의해야 한다.

TreeSet<E>

- TreeSet<E> 클래스는 트리라는 자료구조를 기반으로 데이터를 저장한다.
- 데이터를 정렬된 순서로 저장하며, HashSet<E>와 마찬가지로 데이터의 중복저장 않는다.
- 정렬의 기준은 프로그래머가 직접 정의한다.

```
class Person  implements Comparable<Person> {
    private String name;
    private int age;

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

    @Override
    public String toString() {
        return name + " : " + age;
    }
@Override
    public int compareTo(Person p) {
        return this.age - p.age;
    }
}
```

TreeSet<E>

```
class StringComparator implements Comparator<String> {
    public int compare(String s1, String s2) {
        return s1.length() - s2.length();
    }
}
```

```
        TreeSet<String> tree = new TreeSet<>();
        tree.add("Box");
        tree.add("Rabbit");
        tree.add("Zoom");

        TreeSet<String> tree2 = new TreeSet<>(new StringComparator());
        tree2.add("Box");
        tree2.add("Rabbit");
        tree2.add("Zoom");
```

Map 인터페이스

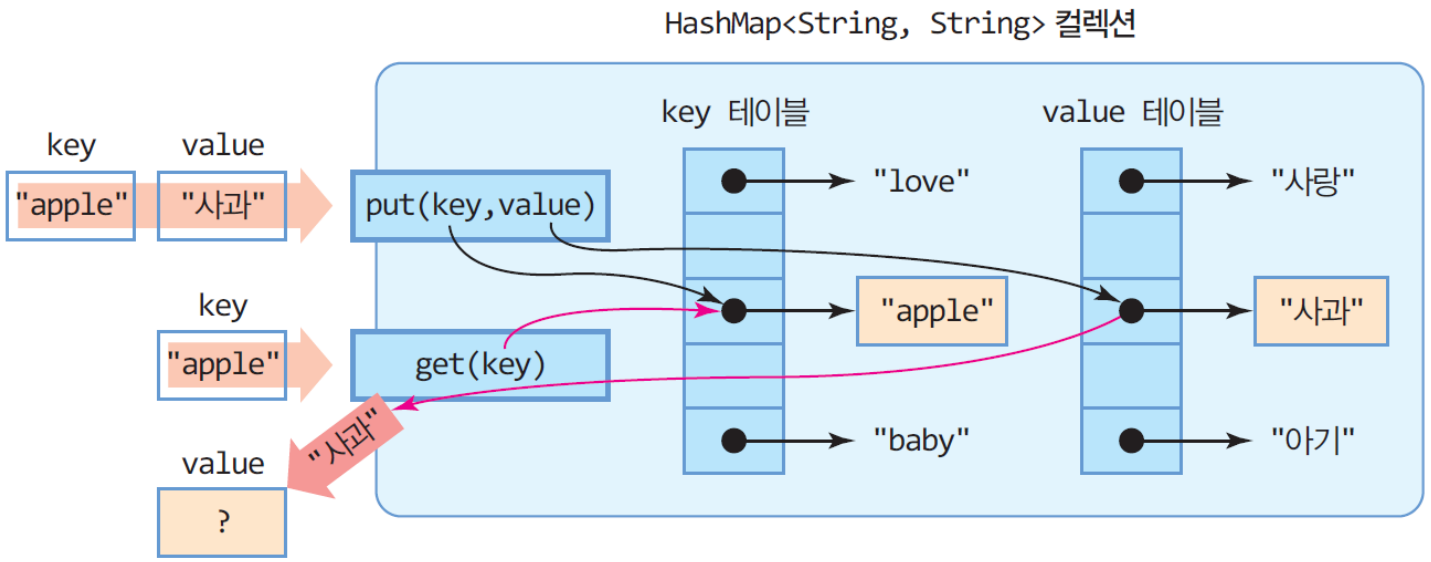
HashMap<K,V>

- HashMap<K,V>
  - ✓키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
    - java.util.HashMap
    - K는 키로 사용할 요소의 타입, V는 값으로 사용할 요소의 타입 지정
    - 키와 값이 한 쌍으로 삽입
    - 키는 해시맵에 삽입되는 위치 결정에 사용
    - 값을 검색하기 위해서는 반드시 키 이용
  - ✓삽입, 삭제, 검색이 빠른 특징
    - 요소 삽입 : put() 메소드
    - 요소 검색 : get() 메소드
  - ✓예) HashMap<String, String> 생성, 요소 삽입, 요소 검색

```
HashMap<String, String> h = new HashMap<String, String>();
h.put("apple", "사과"); // "apple" 키와 "사과" 값의 쌍을 해시맵에 삽입
String kor = h.get("apple"); // "apple" 키로 값 검색. kor는 "사과“
```

HashMap<String, String>의 내부 구성

```
HashMap<String, String> map = new HashMap<String, String>();
```

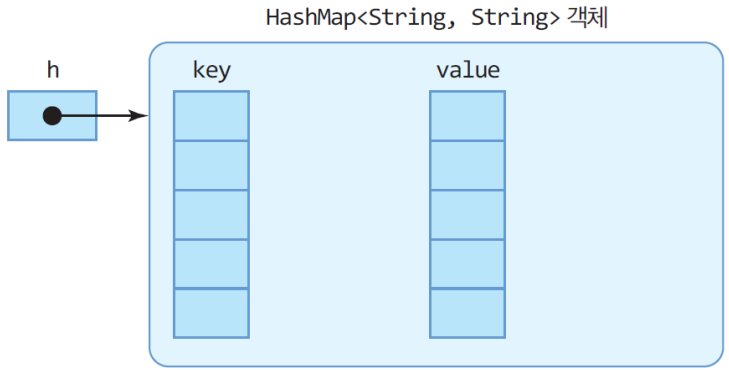


HashMap<K,V>의 주요 메소드

메소드	설명
void clear()	해시맵의 모든 요소 삭제
boolean containsKey(Object key)	지정된 키(key)를 포함하고 있으면 true 리턴
boolean containsValue(Object value)	지정된 값(value)에 일치하는 키가 있으면 true 리턴
V get(Object key)	지정된 키(key)의 값 리턴, 키가 없으면 null 리턴
boolean isEmpty()	해시맵이 비어 있으면 true 리턴
Set<K> keySet()	해시맵의 모든 키를 담은 Set<K> 컬렉션 리턴
V put(K key, V value)	key와 value 쌍을 해시맵에 저장
V remove(Object key)	지정된 키(key)를 찾아 키와 값 모두 삭제
int size()	HashMap에 포함된 요소의 개수 리턴

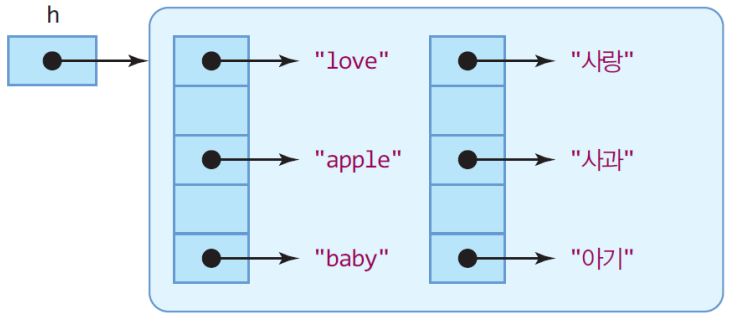
해시맵 생성

```
HashMap<String, String> h =
new HashMap<String, String>();
```



(키, 값) 삽입

```
h.put("baby", "아기");
h.put("love", "사랑");
h.put("apple", "사과");
```



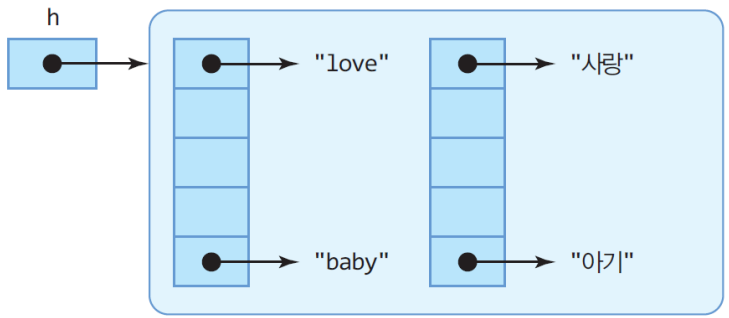
키로 값 읽기

```
String kor = h.get("love");
```

kor = "사랑"

키로 요소 삭제

```
h.remove("apple");
```



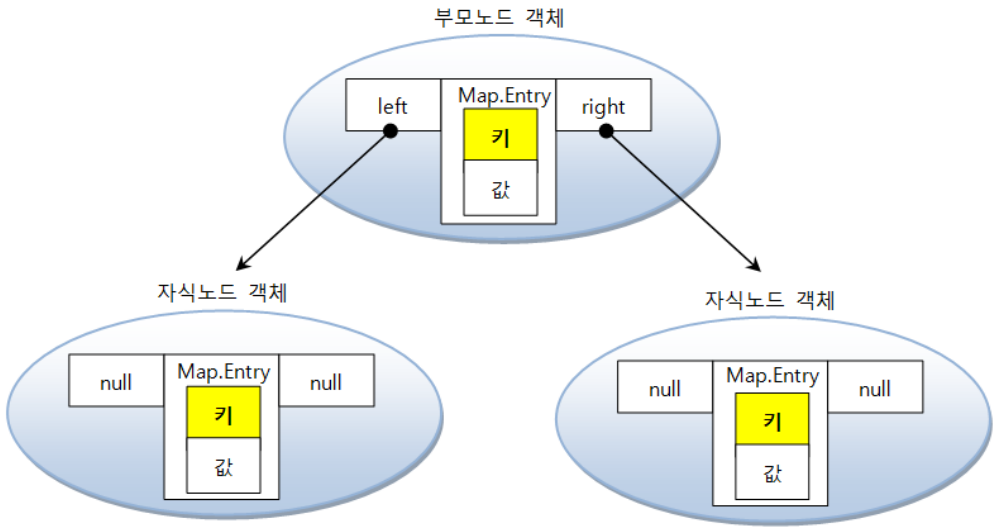
요소 개수

```
int n = h.size();
```

n = 2

TreeMap<K,V>

- TreeMap
  - ✓특징
    - 이진 트리(binary tree) 를 기반으로 한 Map 컬렉션
    - 키와 값이 저장된 Map.Entry를 저장
    - 왼쪽과 오른쪽 자식 노드를 참조하기 위한 두 개의 변수로 구성



```
 TreeMap<Integer, String> map = new TreeMap<>();

        // Key-Value 기반 데이터 저장
        map.put(45, "Brown");
        map.put(37, "James");
        map.put(23, "Martin");

        // Key만 담고 있는 컬렉션 인스턴스 생성
        Set<Integer> ks = map.keySet();

        // 전체 Key 출력 (for-each문 기반)
        for(Integer n : ks)
            System.out.print(n.toString() + '\t');
        System.out.println();

        // 전체 Value 출력 (반복자 기반)
        for(Iterator<Integer> itr = ks.iterator(); itr.hasNext(); )
            System.out.print(map.get(itr.next()) + '\t');
        System.out.println();
    }
}
```

```
TreeMap<Integer, String> map = new TreeMap<>(new AgeComparator());

        // Key-Value 기반 데이터 저장
        map.put(45, "Brown");
        map.put(37, "James");
        map.put(23, "Martin");

        // Key만 담고 있는 컬렉션 인스턴
        Set<Integer> ks = map.keySet();

        // 전체 Key 출력 (for-each문 기반)
        for(Integer n : ks)
            System.out.print(n.toString() + '\t');
        System.out.println();

        // 전체 Value 출력 (반복자 기반)
        for(Iterator<Integer> itr = ks.iterator(); itr.hasNext(); )
            System.out.print(map.get(itr.next()) + '\t');
        System.out.println();
    }
}
```

```
class AgeComparator implements Comparator<Integer> {
    public int compare(Integer n1, Integer n2) {
        return n2.intValue() - n1.intValue();
    }
}
```