Map 컬렉션

❖ Map 컬렉션의 특징 및 주요 메소드

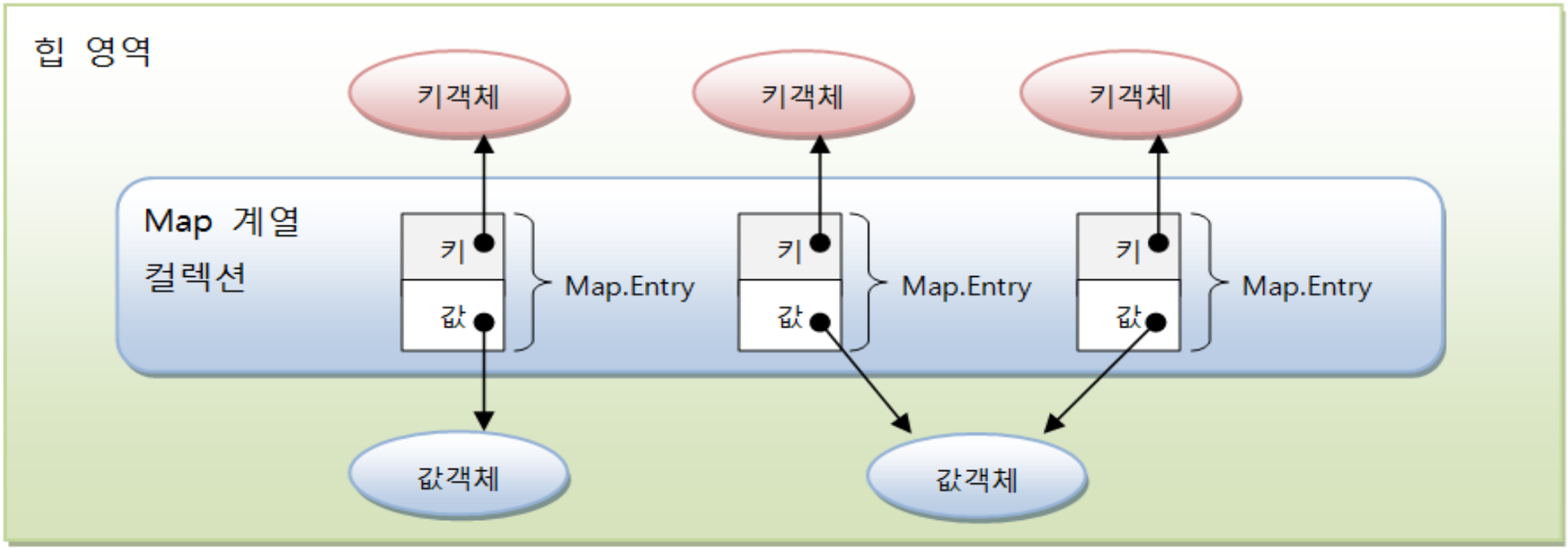
- 특징
  - 키(key)와 값(value)으로 구성된 Map.Entry 객체를 저장하는 구조
  - 키와 값은 모두 객체
  - 키는 중복될 수 없지만 값은 중복 저장 가능

- 구현 클래스
  - HashMap, Hashtable, LinkedHashMap, Properties, TreeMap

Map 컬렉션

❖ Map 컬렉션의 특징 및 주요 메소드

- 키(key)와 값(value)으로 구성된 Entry 객체를 저장하는 구조
- 키, 값 모두 객체



Map 컬렉션

❖ Map 컬렉션의 특징 및 주요 메소드

ㅇ 주요 메소드

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키와 값을 추가, 저장이 되면 값을 리턴
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 여부
	boolean containsValue(Object value)	주어진 값이 있는지 여부
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set 에 담아서 리턴
	V get(Object key)	주어진 키의 값을 리턴
	boolean isEmpty()	컬렉션이 비어있는지 여부
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값 Collection 에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry 삭제, 삭제가 되면 값을 리턴

Map 컬렉션

❖ HashMap

- ㅇ 기본 사용방법

```
Map<String, Integer> map = ~;
map.put("홍길동", 30);                    //객체 추가
int score = map.get("홍길동");            //객체 찾기
map.remove("홍길동");                     //객체 삭제
```

Map 컬렉션

❖ HashMap

- ○ 순회 방법 - 키 집합으로 순회하기

```
Map<K, V> map = ~;
Set<K> keySet = map.keySet();
Iterator<K> keyIterator = keySet.iterator();
while(keyIterator.hasNext()) {
    K key = keyIterator.next();
    V value = map.get(key);
}
```

❖ HashMap

- ○ 순회 방법 – Entry 집합으로 순회하기

```
Set<Map.Entry<K, V>> entrySet = map.entrySet();
Iterator<Map.Entry<K, V>> entryIterator = entrySet.iterator();
while(entryIterator.hasNext()) {
    Map.Entry<K, V> entry = entryIterator.next();
    K key = entry.getKey();
    V value = entry.getValue();
}
```

Map 컬렉션

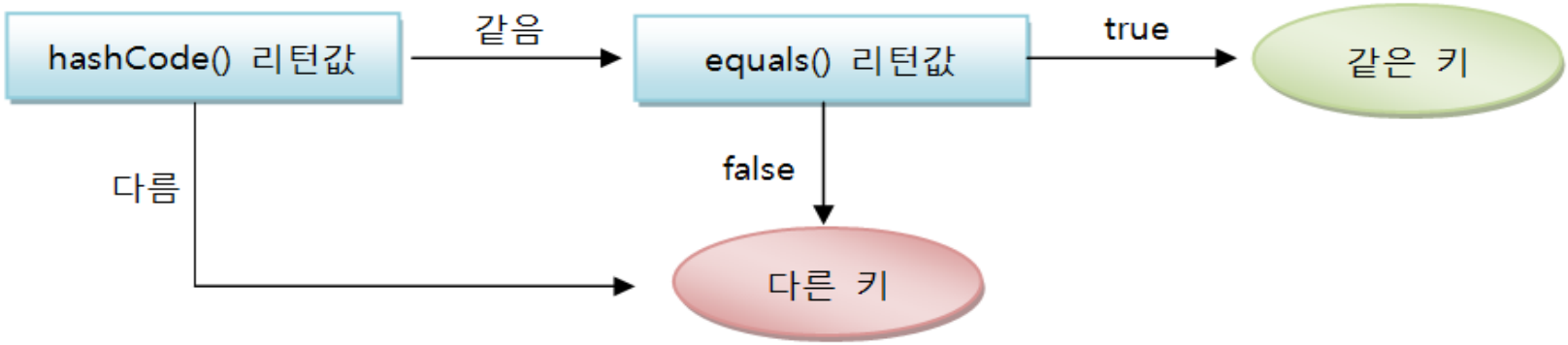
❖ HashMap

- 특징

```
Map<K, V> map = new HashMap<K, V>();
```

키 타입 값 타입 키 타입 값 타입

- 키 객체는 hashCode()와 equals() 를 재정의해 동등 객체가 될 조건을 정해야



  - 키 타입은 String 많이 사용
    - String은 문자열이 같을 경우 동등 객체가 될 수 있도록 hashCode()와 equals() 메소드가 재정의되어 있기 때문

❖ 이름을 키로 접수를 값으로 저장하기: HashMapExample1.java

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapExample1 {
    public static void main(String[] args) {
        //Map 컬렉션 생성
        Map<String, Integer> map = new HashMap<String, Integer>();

        //객체 저장
        map.put("신용권", 85);
        map.put("홍길동", 90);
        map.put("동장군", 80);
        map.put("홍길동", 95);
        System.out.println("총 Entry 수: " + map.size());

        //객체 찾기
        System.out.println("\t홍길동 : " + map.get("홍길동"));
        System.out.println();
```

❖ 이름을 키로 접수를 값으로 저장하기: HashMapExample1.java

```
//객체를 하나씩 처리
Set<String> keySet = map.keySet();
Iterator<String> keyIterator = keySet.iterator();
while(keyIterator.hasNext()) {
  String key = keyIterator.next();
  Integer value = map.get(key);
  System.out.println("\t" + key + " : " + value);
}
System.out.println();

//객체 삭제
map.remove("홍길동");
System.out.println("총 Entry 수: " + map.size());
```

❖ 이름을 키로 접수를 값으로 저장하기: HashMapExample1.java

```
        //객체를 하나씩 처리
        Set<Map.Entry<String, Integer>> entrySet = map.entrySet();
        Iterator<Map.Entry<String, Integer>> entryIterator =
                                                    entrySet.iterator();
        while(entryIterator.hasNext()) {
          Map.Entry<String, Integer> entry = entryIterator.next();
          String key = entry.getKey();
          Integer value = entry.getValue();
          System.out.println("\t" + key + " : " + value);
        }
        System.out.println();

        //객체 전체 삭제
        map.clear();
        System.out.println("총 Entry 수: " + map.size());
    }
}
```

❖ 키로 사용할 객체 – hashCode()와 equals() 재정의: Student.java

```
public class Student {
    public int sno;
    public String name;

    public Student(int sno, String name) {
        this.sno = sno;
        this.name = name;
    }

    public boolean equals(Object obj) {
        if(obj instanceof Student) {
            Student student = (Student) obj;
            return (sno==student.sno) && (name.equals(student.name)) ;
        } else {
            return false;
        }
    }

    public int hashCode() {
        return sno + name.hashCode();
    }
}
```

❖ 학번과 이름이 동일한 경우 같은 키로 인식: HashMapExample2 .java

```
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapExample2 {
    public static void main(String[] args) {
        Map<Student, Integer> map = new HashMap<Student, Integer>();

        map.put(new Student(1, "홍길동"), 95);
        map.put(new Student(1, "홍길동"), 95);

        System.out.println("총 Entry 수: " + map.size());
    }
}
```

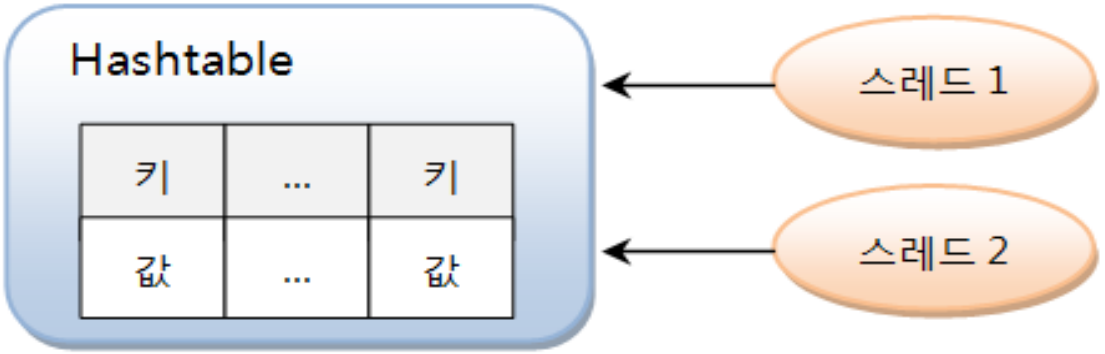
❖ Hashtable

```
Map<K, V> map = new Hashtable<K, V>();
```

키 타입 값 타입 키 타입 값 타입

❖ 특징

- 키 객체 만드는 법은 HashMap과 동일
- Hashtable은 스레드 동기화(synchronization)가 된 상태
  - 복수의 스레드가 동시에 Hashtable에 접근해서 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)



스레드 동기화 적용됨

❖ 아이디와 비밀번호 검사하기: HashtableExample .java

```
import java.util.*;

public class HashtableExample {
    public static void main(String[] args) {
        Map<String, String> map = new Hashtable<String, String>();

        map.put("spring", "12");
        map.put("summer", "123");
        map.put("fall", "1234");
        map.put("winter", "12345");

        Scanner scanner = new Scanner(System.in);
```

❖ 아이디와 비밀번호 검사하기: HashtableExample .java

```
        while(true) {
            System.out.println("아이디와 비밀번호를 입력해주세요");
            System.out.print("아이디: ");
            String id = scanner.nextLine();

            System.out.print("비밀번호: ");
            String password = scanner.nextLine();
            System.out.println();

            if(map.containsKey(id)) {
                if(map.get(id).equals(password)) {
                    System.out.println("로그인 되었습니다");
                    break;
                } else {
                    System.out.println("비밀번호가 일치하지 않습니다.");
                }
            } else {
                System.out.println("입력하신 아이디가 존재하지 않습니다");
            }
        }
    }
}
```

❖ Properties

- 특징
  - 키와 값을 String 타입으로 제한한 Map 컬렉션
  - Properties는 프로퍼티(~.properties) 파일을 읽어 들일 때 주로 사용

- 프로퍼티(~.properties) 파일
  - 옵션 정보, 데이터베이스 연결 정보, 국제화(다국어) 정보를 기록
    - 텍스트 파일로 활용
  - 애플리케이션에서 주로 변경이 잦은 문자열을 저장
    - 유지 보수를 편리하게 만들어 줌

  - 키와 값이 = 기호로 연결되어 있는 텍스트 파일
    - ISO 8859-1 문자셋으로 저장
    - 한글은 유니코드(Unicode)로 변환되어 저장

❖ 키=값으로 구성된 프로퍼티: database.properties

```
driver=oracle.jdbc.OracleDirver
url=jdbc:oracle:thin:@localhost:1521:orcl
username=scott
password=tiger
```

❖ 키=값으로 구성된 프로퍼티: PropertiesExample.java

```
import java.io.FileReader;
import java.net.URLDecoder;
import java.util.Properties;

public class PropertiesExample {
    public static void main(String[] args) throws Exception {
        Properties properties  = new Properties();
        String path = PropertiesExample.class.getResource(
                                              "database.properties").getPath();
        path = URLDecoder.decode(path, "utf-8");
        properties.load(new FileReader(path));

        String driver = properties.getProperty("driver");
        String url = properties.getProperty("url");
        String username = properties.getProperty("username");
        String password = properties.getProperty("password");

        System.out.println("driver : " + driver);
        System.out.println("url : " + url);
        System.out.println("username : " + username);
        System.out.println("password : " + password);
    }
}
```