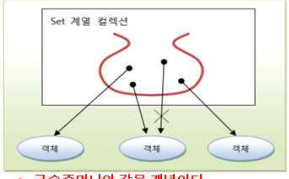
제24장
컬렉션 프레임워크-Part2

3. Set 컬렉션

❖ **Set인터페이스를 구현한 컬렉션의 특징(순서 유지안함, 중복허용 안함)**

- **특징**
  - 수학의 집합에 비유할 수 있다.
  - 저장 순서가 유지되지 않는다.
  - 객체를 중복 저장 불가능하다.
  - 하나의 null만 저장 가능하다.
- **구현 클래스**
  - HashSet, TreeSet(저장되면서 정렬됨)
  - SortedSet(TreeSet의 조상)
- **주요 메서드**



- 구슬주머니와 같은 개념이다.
- 역시 객체가 저장 되는 것이 아니라 참조가 저장되는 것임

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 저장, 객체가 성공적으로 저장되면 true 를 리턴하고 중복 객체면 false 를 리턴
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	isEmpty()	컬렉션이 비어 있는지 조사
	Iterator<E> iterator()	저장된 객체를 한번씩 가져오는 반복자 리턴
	int size()	저장되어있는 전체 객체수 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

- Collection인터페이스에서 추가되는 메서드가 하나도없다. 하여 인덱스를 이용해서 가져오는 방법이 없다. 대신 Iterator(반복자)가 있어서 객체 검색이 가능하다.

3. Set 컬렉션

- **객체 추가 및 삭제**

```
Set<String> set = ...;

set.add("홍길동");          //객체 추가
set.add("이민기");

set.remove("홍길동");       //객체 삭제
set.remove("이민기");
```

- **Set컬렉션은 인덱스로 객체를 검색해서, 가져오는 메서드가 없다.**
  ※대신, 전체객체를 대상으로 한번씩 반복해서 가져오는 반복자(Iterator)를 제공한다.

```
Set<String> set = ...;
Iterator<String> iterator = set.iterator();
```

리턴 타입	메서드명	설명
boolean	hasNext()	가져올 객체가 있으면, true를 반환하고, 없으면 false를 리턴 한다.
E	next()	컬렉션에서 하나의 객체를 가져온다.
void	remove()	Set컬렉션에서 객체를 제거한다.

```
Set<String> set = ...;
Iterator<String> iterator = set.iterator();
while(iterator.hasNext()) {
      String str = iterator.next(); //String객체를 하나 가져온다.
}
```

```
Set<String> set = ...;
for(String str : set)
{}
```

3. Set 컬렉션

- **반복자를 통한 제거**

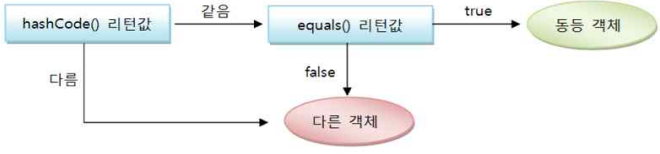
```
Set<String> set = ...;
Iterator<String> iterator = sec.iterator();

while(iterator.hasNext()) {

      String str = iterator.next(); //String객체를 하나 가져온다.

      if(str.equals("홍길동"))
       {
          iterator.remove();
       }
}
```

3. Set 컬렉션

❖ HashSet

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

- 특징
  - 동일 객체 및 동등 객체는 중복 저장하지 않는다.
  - 동등 객체 판단 방법



  - 1번째 호출 : hashCode()
  - 2번째 호출 : equals()

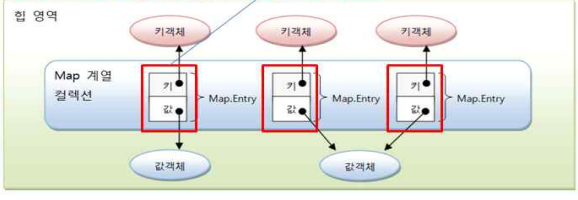
* 1번째와 2번째가 둘다 true가 나와야 동등객체로 판단함.

4. Map 컬렉션

❖ **Map 컬렉션의 특징 및 주요 메소드**

(순서 유지안함, 키중복 허용안함, 값은 허용)

- **특징**
  - **키(key)와 값(value)으로 구성된 [Map.Entry 객체] 를 저장하는 구조이다.**
  - **키와 값은 모두 객체이다.**
  - **키는 중복될 수 없지만 값은 중복 저장 가능하다.**



- **구현 클래스**
  - **HashMap(신버전), Properties, TreeMap(저장되면서 정렬됨),**
    Hashtable(Map이 나오기 이전 구버전)

4. Map 컬렉션

❖ Map 컬렉션의 특징 및 주요 메서드

▪ 주요 메소드

저장 되는 형태

키(Key	값(Value)
myId	12345
Perpear	78110

하나의 Map.Entry라고 부른다.

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() * Set의 Iterator를 이용하기 위함, 키,값 모두	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 Set의 Iterator를 이용, 키만 저장
	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

4. Map 컬렉션

- **객체 추가 및 삭제**

```
Map<String, Integer> map = ...;

map.put("홍길동", 30);                    //객체 추가  //자동 박싱

int score = map.get("홍길동");            //객체 찾기  //자동 언박싱

map.remove("홍길동");                     //객체 삭제
```

- **전체객체를 대상으로 반복해서 얻기**

```
Map<K, V> map  = ...;
Set<K> keySet = map.keySet();   //키값만 Set으로 담는다.
Iterator<K> keyIterator = keySet.iterator();

While(keyIterator.hasNext() ) {
    K key = keyIterator.next();
    //키로 값을 찾는다.
    V value = map.get(key);
}
```

```
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();
}
```

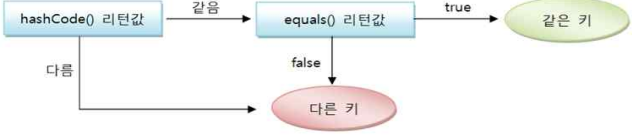
4. Map 컬렉션

❖ HashMap

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

키 타입 — 값 타입 — 키 타입 — 값 타입

- 특징
  - 키 객체는 hashCode()와 equals() 를 재정의해 동등 객체가 될 조건을 정해야 한다. 동등 키는 한 번만 저장 된다는 것에 주목 하자.



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

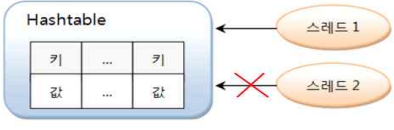
4. Map 컬렉션

❖ Hashtable

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

키 타입 — 값 타입 — 키 타입 — 값 타입

- 특징
  - 키 객체는 hashCode()와 equals()를 재 정의해서 동등 객체가 될 조건을 정해야 한다.
  - Hashtable은 스레드 동기화(synchronization)가 되어 있기 때문에, 복수의 스레드가 동시에 Hashtable에 접근해서 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)하다.(List계열의 Vector와 동일) -- 싱글 스레드의 경우는 HashMap을 권장함.



스레드 동기화 적용됨

4. Map 컬렉션

❖ Properties

- 특징
  - 키와 값을 String타입으로 제한한 Map컬렉션이다.
  - Properties는 프로퍼티(~.properties) 파일을 읽어 들일 때 주로 사용한다.

- 프라퍼티(~.properties) 파일
  - 옵션 정보, 데이터베이스 연결 정보, 국제화(다국어)정보를 기록한 텍스트 파일로 활용
  - 애플리케이션에서 주로 변경이 잦은 문자열을 저장해서 유지 보수를 편리하게 만들어 줌
    - 프라퍼티 파일이 없다면, 소스 코드를 변경될 때 마다 수정해서 릴리즈를 해야 됨(불편)

[database.properties]키 = 값으로 구성된 프라퍼티	
1	driver = oracle.jdbc.OracleDriver
2	url = jdbc:oracle:thin@localhost:1521:orcl
3	username = scott
4	password = tiger

  - 키와 값이 = 기호로 연결되어 있는 텍스트 파일로 ISO 8859-1(유럽) 문자셋으로 저장
    한글(EUC-KR)은 유니코드(Unicode)로 변환되어 저장됨.

```
contry=대한민국
language=한글
```

⇒

```
contry=₩uB300₩uD55C₩uBBFC₩uAD6D
language=₩uD55C₩uAE00
```

* 파일 저장 시 위와 같이 변환

4. Map 컬렉션

- **Properties객체 생성**
  - 파일 시스템 경로를 이용

```
Properties properties = new Properties();

Properties.load(new  FileReader("C:/~/database.properties"));    //프라퍼티 파일 경로
```

* FileReader는 문자기반 파일입력스트림의 일종으로 io에서 배움.

  - 클래스 패스(ClassPath)를 이용

```
//class파일이 있는곳에 프라퍼티 파일이 같이 존재하는 경우
String path = 클래스.class.getResource("database.properties").getPath();

Path = URLDecoder.decode(path, "utf-8");      ← 경로에 한글이 있을 경우 한글을 복원

Properties properties = new Properties();

Properties.load(new  FileReader(path));
```

```
//A.class가 있는 폴더의 서브폴더로 config가 있고 그 안에 프라퍼티 파일이 존재할 경우
String path = A.class.getResource("config/database.properties").getPath();
```

- **값 읽기**

```
String value = properties.getProperties("key");
```

감사합니다.