13-1. 컬렉션 프레임워크

혼자 공부하는 자바 (신용권 저)

목차

시작하기 전에

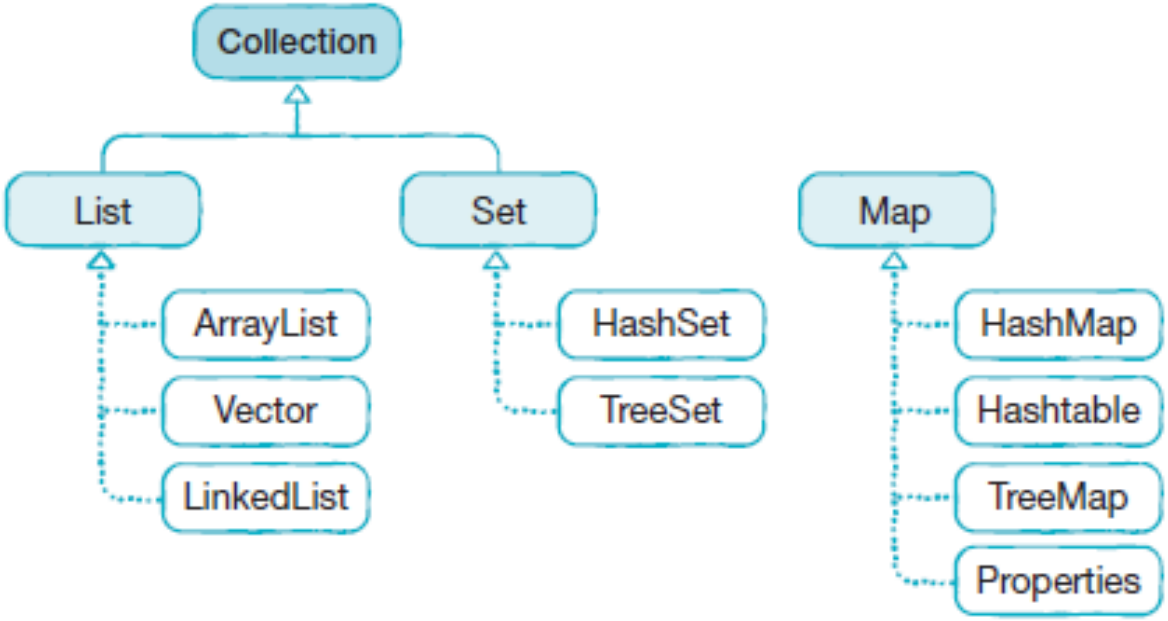
[핵심 키워드] : 컬렉션 프레임워크, List 컬렉션, Set 컬렉션, Map 컬렉션

[핵심 포인트]
배열이 가지는 불편함을 해결하기 위해 제공되는 컬렉션 프레임워크에 대해 알아본다.

시작하기 전에

❖ 컬렉션 프레임워크 (Collection Framework)

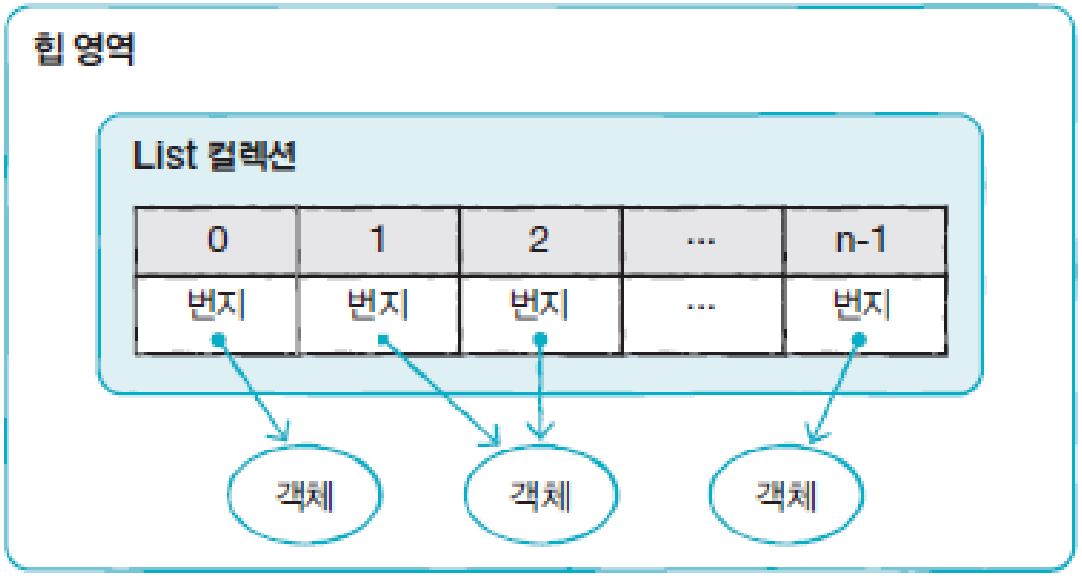
- 자료구조를 사용해서 객체들을 효율적으로 관리할 수 있도록 인터페이스와 구현 클래스를 java.util 패키지에서 제공함
- 프레임워크 : 사용 방법을 정해놓은 라이브러리
- 주요 인터페이스로 List, Set, Map이 있음



❖ List 컬렉션

- 객체를 인덱스로 관리
- 저장용량이 자동으로 증가하며 객체 저장 시 자동 인덱스가 부여
- 추가, 삭제, 검색 위한 다양한 메소드 제공
- 객체 자체를 저장하는 것이 아닌 객체 번지 참조

  null도 저장 가능



List 컬렉션

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨 끝에 추가합니다.
	void add(int index, E element)	주어진 인덱스에 객체를 추가합니다.
	E set(int index, E element)	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿉니다.
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 조사합니다.
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴합니다.
	boolean isEmpty()	컬렉션이 비어 있는지 조사합니다.
	int size()	저장되어 있는 전체 객체 수를 리턴합니다.
객체 삭제	void clear()	저장된 모든 객체를 삭제합니다.
	E remove(int index)	주어진 인덱스에 저장된 객체를 삭제합니다.
	boolean remove(Object o)	주어진 객체를 삭제합니다.

```
List<String> list = …;
list.add("홍길동");          //맨 끝에 객체 추가
list.add(1, "신용권");       //지정된 인덱스에 객체 삽입
String str = list.get(1);    //인덱스로 객체 검색
list.remove(0);              //인덱스로 객체 삭제
list.remove("신용권");       //객체 삭제
```

List 컬렉션

- List 컬렉션에 저장된 모든 객체를 대상으로 하나씩 가져와 처리하려는 경우

인덱스 이용

```
List<String> list = …;
for(int i=0; i<list.size(); i++) {
  String str = list.get(i);
}
```

← 저장된 총 객체 수만큼 루핑

i 인덱스에 저장된 String 객체를 가져옴

for문 이용

String 객체를 하나씩 가져옴

```
for(String str : list) {
}
```

← 저장된 총 객체 수만큼 루핑

List 컬렉션

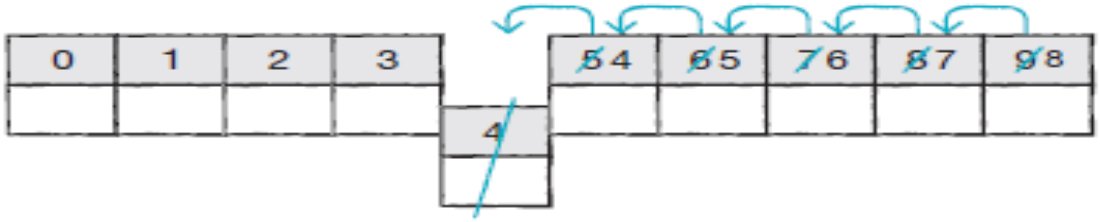
❖ ArrayList

- List 인터페이스의 대표적 구현 클래스
- ArrayList 객체 생성


```
List<String> list = new ArrayList<String>();
List<String> list = new ArrayList<>();
```

List 컬렉션

- ArrayList에 객체 추가하면 0번 인덱스부터 차례로 저장
- 객체 제거하는 경우 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨짐



- 예시 – String 객체를 저장하는 ArrayList

```
01  package sec01.exam01;
02
03  import java.util.*;
04
05  public class ArrayListExample {
06    public static void main(String[] args) {
07      List<String> list = new ArrayList<String>();
08
09      list.add("Java");
10      list.add("JDBC");
11      list.add("Servlet/JSP");
12      list.add(2, "Database");
13      list.add("iBATIS");
14
```

String 객체를 저장

List 컬렉션

```
15        int size = list.size();<-------------- 저장된 총 객체 수 얻기
16        System.out.println("총 객체수: " + size);
17        System.out.println();
18
19        String skill = list.get(2);<---------- 2번 인덱스의 객체 얻기
20        System.out.println("2: " + skill);
21        System.out.println();
22
23        for(int i=0; i<list.size(); i++) {<---- 저장된 총 객체 수만큼 루핑
24          String str = list.get(i);
25          System.out.println(i + ":" + str);
26        }
27        System.out.println();
28
29        list.remove(2);<------------------------ 2번 인덱스 객체(Database) 삭제됨
30        list.remove(2);<------------------------ 2번 인덱스 객체(Servlet/JSP) 삭제됨
31        list.remove("iBATIS");
32
33        for(int i=0; i<list.size(); i++) {<---- 저장된 총 객체 수만큼 루핑
34          String str = list.get(i);
35          System.out.println(i + ":" + str);
36        }
37      }
38    }
```

실행결과

```
총 객체수: 5

2: Database

0:Java
1:JDBC
2:Database
3:Servlet/JSP
4:iBATIS

0:Java
1:JDBC
```

❖ Vector

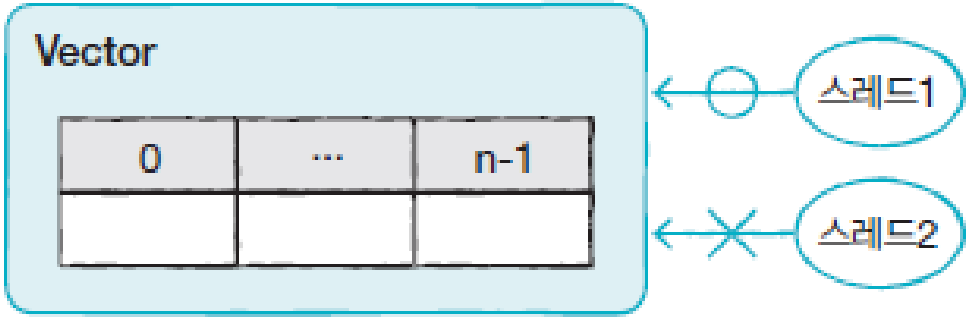
- 저장할 객체 타입을 타입 파라미터로 표기하고 기본 생성자 호출하여 생성

```
List<E> list = new Vector<E>();
List<E> list = new Vector<>();
```

Vector의 E 타입 파라미터를 생략하면 왼쪽 List에 지정된 타입을 따라 감

- 동기화된 메소드로 구성되어 멀티 스레드가 동시에 Vector의 메소드들 실행할 수 없고, 하나의 스레드가 메소드 실행 완료해야만 다른 스레드가 메소드 실행할 수 있음
- 멀티 스레드 환경에서 안전하게 객체 추가 및 삭제할 수 있음

  스레드에 안전 (thread safe)



List 컬렉션

- 예시 – Board 객체를 저장하는 Vector

```
01  package sec01.exam02;
02
03  import java.util.*;
04
05  public class VectorExample {
06    public static void main(String[] args) {
07      List<Board> list = new Vector<Board>();
08
09      list.add(new Board("제목1", "내용1", "글쓴이1"));
10      list.add(new Board("제목2", "내용2", "글쓴이2"));
11      list.add(new Board("제목3", "내용3", "글쓴이3"));
12      list.add(new Board("제목4", "내용4", "글쓴이4"));
13      list.add(new Board("제목5", "내용5", "글쓴이5"));
14
15      list.remove(2);
16      list.remove(3);
```

Board 객체를 저장

15 ← 2번 인덱스 객체(제목3) 삭제(뒤의 인덱스는 1씩 앞으로 당겨짐)

16 ← 3번 인덱스 객체(제목5) 삭제

List 컬렉션

```
17
18        for(int i=0; i<list.size(); i++) {
19          Board board = list.get(i);
20          System.out.println(board.subject + "\t" + board.content + "\t" + board.writer);
21        }
22      }
23    }
```

실행결과

```
제목1   내용1   글쓴이1
제목2   내용2   글쓴이2
제목4   내용4   글쓴이4
```

List 컬렉션

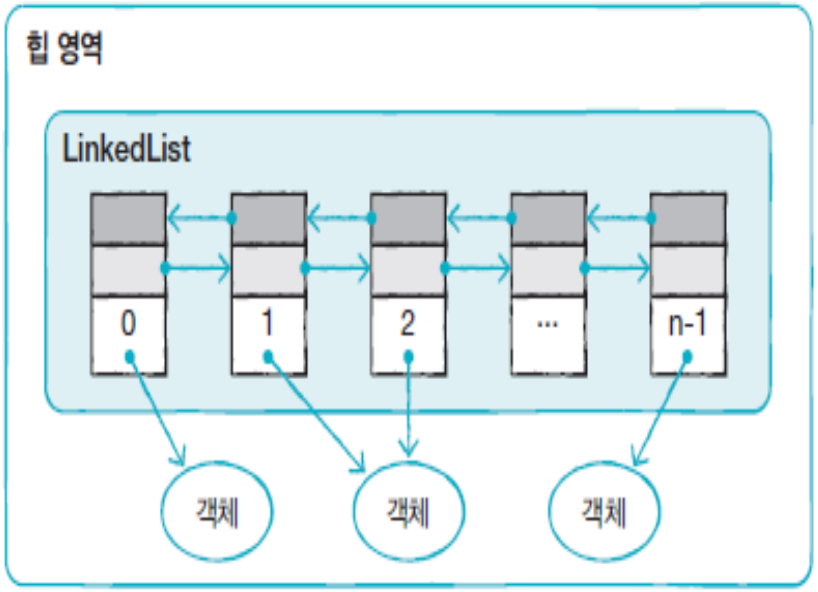
- 예시 – 게시물 정보 객체

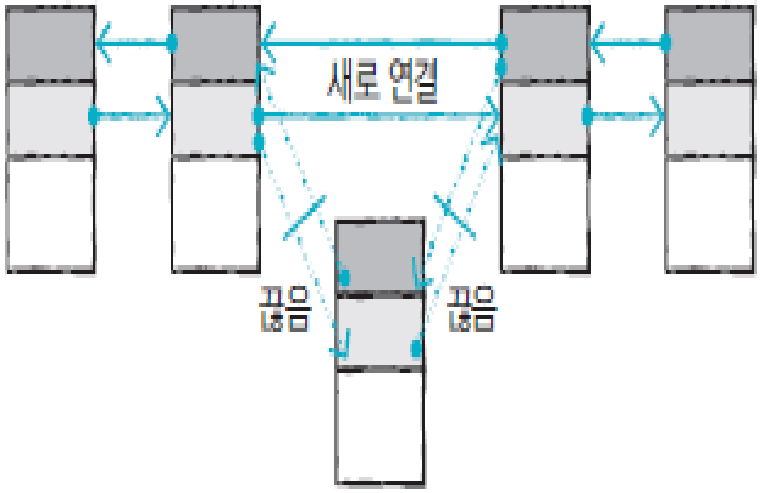
```
01  package sec01.exam02;
02
03  public class Board {
04    String subject;
05    String content;
06    String writer;
07
08    public Board(String subject, String content, String writer) {
09      this.subject = subject;
10      this.content = content;
11      this.writer = writer;
12    }
13  }
```

List 컬렉션

❖ LinkedList

- ArrayList와 사용 방법은 같으나 내부 구조가 다름
- 인접 참조를 링크하여 체인처럼 객체를 관리
- 특정 인덱스 객체 제거하거나 삽입하면 앞뒤 링크만 변경되고 나머지 링크는 변경되지 않음





List 컬렉션

- 저장할 객체 타입을 타입 파라미터에 표기하고 기본 생성자 호출하여 생성

```
List<E> list = new LinkedList<E>();
List<E> list = new LinkedList<>();
```

LinkedList의 E 타입 파라미터를 생략하면 왼쪽 List에 지정된 타입을 따라 감

- 예시 – ArrayList와 LinkedList의 실행 성능 비교

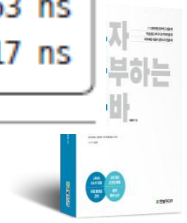
```
01  package sec01.exam03;
02
03  import java.util.*;
04
05  public class LinkedListExample {
06    public static void main(String[] args) {
07      List<String> list1 = new ArrayList<String>();
08      List<String> list2 = new LinkedList<String>();
09
10      long startTime;
11      long endTime;
12
```

List 컬렉션

```
13        startTime = System.nanoTime();
14        for(int i=0; i<10000; i++) {
15          list1.add(0, String.valueOf(i));
16        }
17        endTime = System.nanoTime();
18        System.out.println("ArrayList 걸린시간: " + (endTime-startTime) + " ns");
19
20        startTime = System.nanoTime();
21        for(int i=0; i<10000; i++) {
22          list2.add(0, String.valueOf(i));
23        }
24        endTime = System.nanoTime();
25        System.out.println("LinkedList 걸린시간: " + (endTime-startTime) + " ns");
26      }
27    }
```

실행결과

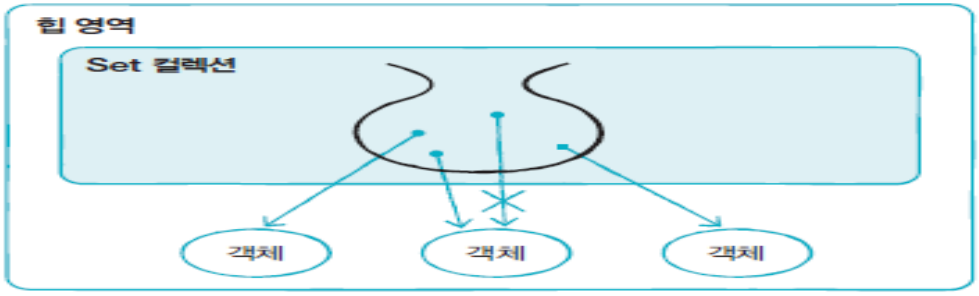
```
ArrayList 걸린시간: 20248953 ns
LinkedList 걸린시간: 4279517 ns
```



Set 컬렉션

❖ Set 컬렉션

- 저장 순서 유지되지 않으며, 객체 중복하여 저장할 수 없고 하나의 null만 저장할 수 있다.



- HashSet, LinkedHashSet, TreeSet 등

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 저장합니다. 객체가 성공적으로 저장되면 true를 리턴하고 중복 객체면 false를 리턴합니다.
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 조사합니다.
	boolean isEmpty()	컬렉션이 비어 있는지 조사합니다.
	Iterator<E> iterator()	저장된 객체를 한 번씩 가져오는 반복자를 리턴합니다.
	int size()	저장되어 있는 전체 객체 수를 리턴합니다.
객체 삭제	void clear()	저장된 모든 객체를 삭제합니다.
	boolean remove(Object o)	주어진 객체를 삭제합니다.

Set 컬렉션

- Set 컬렉션에 객체 저장 및 삭제

```
Set<String> set = …;
set.add("홍길동");        //객체 추가
set.add("신용권");
set.remove("홍길동");     //객체 삭제
```

- iterator() 메소드 호출하여 반복자 얻고, 반복자로 검색 기능 대체

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

- Iterator 인터페이스 메소드

리턴 타입	메소드	설명
boolean	hasNext()	가져올 객체가 있으면 true를 리턴하고 없으면 false를 리턴합니다.
E	next()	컬렉션에서 하나의 객체를 가져옵니다.
void	remove()	Set 컬렉션에서 객체를 제거합니다.

Set 컬렉션

- String 객체들 반복해서 하나씩 가져오기

```
Set<String> set = …;
Iterator<String> iterator = set.iterator();
while(iterator.hasNext()) {
  //String 객체 하나를 가져옴
  String str = iterator.next();
}
```

저장된 객체 수만큼 루핑

향상된 for문 이용하여 전체 객체 대상으로 반복

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

저장된 객체 수만큼 루핑

Set 컬렉션

- remove() 메소드로 객체 제거

```
while(iterator.hasNext()) {
  String str = iterator.next();
  if(str.equals("홍길동")) {
    iterator.remove();
  }
}
```

Set 컬렉션

❖ HashSet

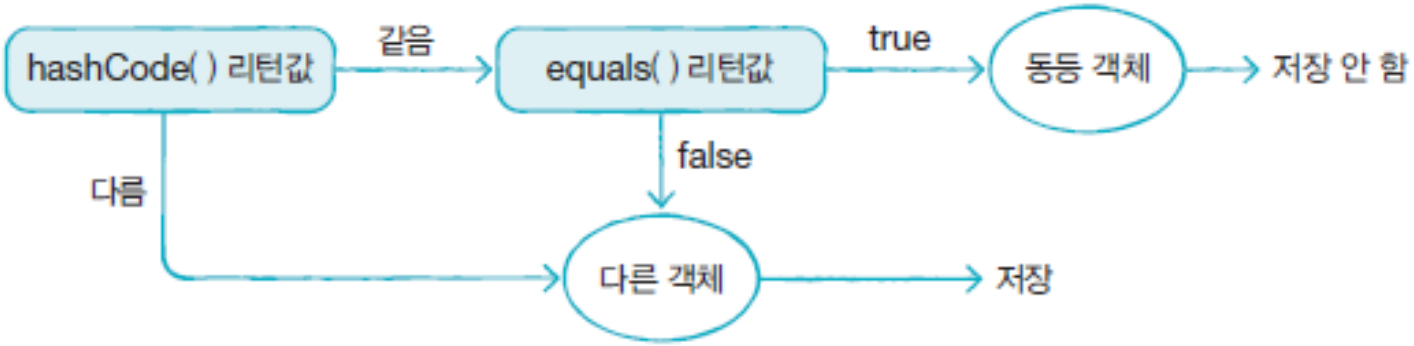
- Set 인터페이스의 구현 클래스
- 기본 생성자 호출하여 생성

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

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

HashSet의 E 타입 파라미터를 생략하면 왼쪽 Set에 지정된 타입을 따라 감

- 객체를 순서 없이 저장하되 동일한 객체는 중복 저장하지 않음
- 객체 저장 전 객체의 hashCode() 메소드 호출하여 해시코드 얻어내고 이미 저장된 객체의 해시코드와 비교



Set 컬렉션

- 예시 – String 객체를 중복 없이 저장하는 HashSet

```
01  package sec01.exam04;
02
03  import java.util.*;
04
05  public class HashSetExample {
06    public static void main(String[] args) {
07      Set<String> set = new HashSet<String>();
08
09      set.add("Java");
10      set.add("JDBC");
11      set.add("Servlet/JSP");
12      set.add("Java");
13      set.add("iBATIS");
14
15      int size = set.size();
16      System.out.println("총 객체수: " + size);
17
```

"Java"는 한 번만 저장됨

저장된 객체 수 얻기

Set 컬렉션

```
18        Iterator<String> iterator = set.iterator();
19        while(iterator.hasNext()) {
20          String element = iterator.next();
21          System.out.println("\t" + element);
22        }
23
24        set.remove("JDBC");
25        set.remove("iBATIS");
26
27        System.out.println("총 객체수: " + set.size());
28
29        iterator = set.iterator();
30        for(String element : set) {
31          System.out.println("\t" + element);
32        }
33
34        set.clear();
35        if(set.isEmpty()) { System.out.println("비어 있음"); }
36      }
37    }
```

반복자 얻기

객체 수 만큼 루핑

1개의 객체를 가져옴

1개의 객체 삭제

1개의 객체 삭제

저장된 객체 수 얻기

반복자 얻기

객체 수 만큼 루핑

모든 객체를 제거하고 비움

실행결과

```
총 객체수: 4
   Java
   JDBC
   Servlet/JSP
   iBATIS
총 객체수: 2
   Java
   Servlet/JSP
비어 있음
```

Set 컬렉션

- 예시 – hashCode()와 equals() 메소드 재정의

```
01  package sec01.exam05;
02
03  public class Member {
04    public String name;
05    public int age;
06
07    public Member(String name, int age) {
08      this.name = name;
09      this.age = age;
10    }
11
12    @Override
13    public boolean equals(Object obj) {
14      if(obj instanceof Member) {
15        Member member = (Member) obj;
16        return member.name.equals(name) && (member.age==age) ;
17      } else {
18        return false;
19      }
20    }
21
22    @Override
23    public int hashCode() {
24      return name.hashCode() + age;
25    }
26  }
```

name과 age 값이 같으면 true를 리턴

name과 age 값이 같으면 동일한 hashCode를 리턴

String의 hashCode() 이용

Set 컬렉션

- 예시 - Member 객체를 중복 없이 저장하는 HashSet

```
01  package sec01.exam05;
02
03  import java.util.*;
04
05  public class HashSetExample2 {
06    public static void main(String[] args) {
07      Set<Member> set = new HashSet<Member>();
08
09      set.add(new Member("홍길동", 30));
10      set.add(new Member("홍길동", 30));
11
12      System.out.println("총 객체수 : " + set.size());
13    }
14  }
```

인스턴스는 다르지만 내부 데이터가 동일하므로 객체 1개만 저장

저장된 객체 수 얻기

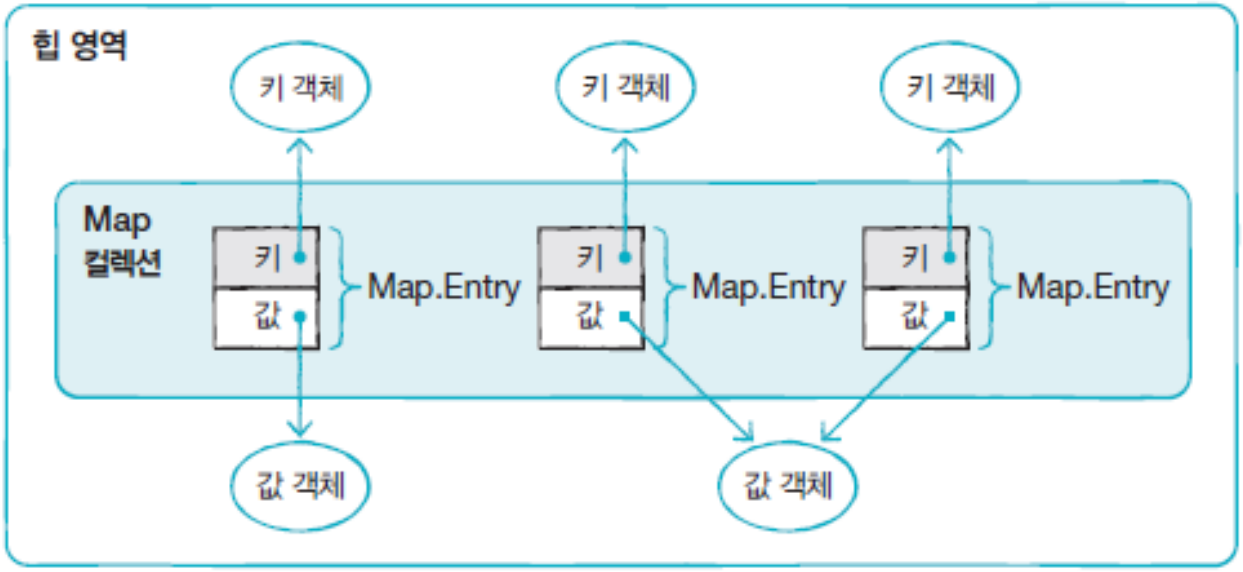
실행결과

```
총 객체수 : 1
```

❖ Map 컬렉션

- 키와 값으로 구성된 Map.Entry 객체 저장하는 구조 가짐
- 키는 중복 저장될 수 없으나 값은 중복 저장될 수 있음

  기존 저장된 키와 동일한 키로 값을 저장하면 기존 값 없어지고 새로운 값으로 대체



- HashMap, Hashtable, LinkedHashMap, Properties, TreeMap 등

Map 컬렉션

- Map 인터페이스 메소드

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키로 값을 저장합니다. 새로운 키일 경우 null을 리턴하고 동일한 키가 있을 경우 값을 대체하고 이전 값을 리턴합니다
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 여부를 확인합니다.
	boolean containsValue(Object value)	주어진 값이 있는지 여부를 확인합니다.
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set에 담아서 리턴합니다.
	V get(Object key)	주어진 키가 있는 값을 리턴합니다.
	boolean isEmpty()	컬렉션이 비어 있는지 여부를 확인합니다.
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴합니다.
	int size()	저장된 키의 총 수를 리턴합니다.
	Collection<V> values()	저장된 모든 값을 Collection에 담아서 리턴합니다.
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제합니다.
	V remove(Object key)	주어진 키와 일치하는 Map.Entry를 삭제하고 값을 리턴합니다.

Map 컬렉션

- 키 타입이 String, 값 타입이 Integer인 Map 컬렉션 생성하고 put() 메소드로 키와 값을 저장, 키로 값 얻거나 제거하기 위해 get()과 remove() 메소드 사용

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

- 저장된 전체 객체를 대상으로 하나씩 얻고 싶은 경우

  keySet() 메소드로 모든 키를 Set 컬렉션으로 얻은 뒤 반복자 통해 키 하나씩 얻고 get() 메소드 통해 값 얻음

```
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);
}
```

Map 컬렉션

entrySet() 메소드로 모든 Map.Entry를 Set 컬렉션으로 얻은 뒤 반복자 통해 Map.Entry() 하나씩 얻고 getKey() 와 getValue() 메소드 이용해 키와 값 얻음

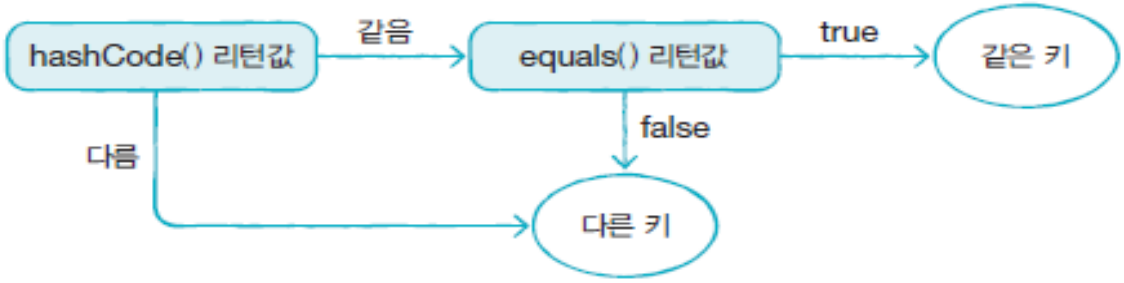
```
Set<Map.Entry<K, V>> entrySet = map.entrySet();
Iterator<Map.Entry<K, V>> entryIterator = entrySet.iterator();
while(entryIterator.hasNext()) {
  Map.Entry<K, V> entry = entryIterator.next();
  K key = entry.getKey();
  V value = entry.getValue();
}
```

Map 컬렉션

❖ HashMap

- 대표적인 Map 컬렉션
- HashMap의 키로 사용할 객체는 hashCode()와 equals() 메소드 재정의하여 동등 객체가 될 조건 정해야

  hashCode() 리턴값 같고 equals() 메소드가 true 리턴해야 함



- HashMap 생성하려면 키 타입과 값 타입을 타입 파라미터로 주고 기본 생성자 호출

```
Map<K, V> map = new HashMap<K, V>();
    ↑  ↑              ↑  ↑
  키 타입 값 타입    키 타입 값 타입
```

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

HashMap의 K와 V 타입 파라미터를 생략하면 왼쪽 Map에 지정된 타입을 따라 감

Map 컬렉션

- 예시 – 이름을 키로 점수를 값으로 저장하기

```
01  package sec01.exam06;
02
03  import java.util.HashMap;
04  import java.util.Iterator;
05  import java.util.Map;
06  import java.util.Set;
07
08  public class HashMapExample {
09    public static void main(String[] args) {
10      //Map 컬렉션 생성
11      Map<String, Integer> map = new HashMap<String, Integer>();
12
13      //객체 저장
14      map.put("신용권", 85);
15      map.put("홍길동", 90);
16      map.put("동장군", 80);
17      map.put("홍길동", 95);
18      System.out.println("총 Entry 수: " + map.size());
19
20      //객체 찾기
21      System.out.println("\t홍길동 : " + map.get("홍길동"));
22      System.out.println();
```

"홍길동" 키가 같기 때문에 제일 마지막에 저장한 값으로 대체

저장된 총 Entry 수 얻기

이름(키)으로 점수(값)를 검색

Map 컬렉션

```
23
24        //객체를 하나씩 처리
25        Set<String> keySet = map.keySet();
26        Iterator<String> keyIterator = keySet.iterator();
27        while(keyIterator.hasNext()) {
28          String key = keyIterator.next();
29          Integer value = map.get(key);
30          System.out.println("\t" + key + " : " + value);
31        }
32        System.out.println();
33
34        //객체 삭제
35        map.remove("홍길동");
36        System.out.println("총 Entry 수: " + map.size());
37
38        //객체를 하나씩 처리
39        Set<Map.Entry<String, Integer>> entrySet = map.entrySet();
40        Iterator<Map.Entry<String, Integer>> entryIterator = entrySet.iterator();
41
42        while(entryIterator.hasNext()) {
43          Map.Entry<String, Integer> entry = entryIterator.next();
44          String key = entry.getKey();
45          Integer value = entry.getValue();
46          System.out.println("\t" + key + " : " + value);
47        }
48        System.out.println();
49
```

Key Set 얻기

반복해서 키를 얻고
값을 Map에서 얻어냄

키로 Map.Entry를 제거

Map.Entry Set 얻기

반복해서
Map.Entry를 얻고
키와 값을 얻어냄

Map 컬렉션

```
50        //객체 전체 삭제
51        map.clear();  ←———————— 모든 Map.Entry 삭제
52        System.out.println("총 Entry 수: " + map.size());
53      }
54    }
```

실행결과

```
총 Entry 수: 3
  홍길동 : 95

  홍길동 : 95
  신용권 : 85
  동장군 : 80

총 Entry 수: 2
  신용권 : 85
  동장군 : 80

총 Entry 수: 0
```

Map 컬렉션

- 예시 – 키로 사용할 객체 – hashCode()와 equals() 재정의

```
01  package sec01.exam07;
02
03  class Student {
04    public int sno;
05    public String name;
06
07    public Student(int sno, String name) {
08      this.sno = sno;
09      this.name = name;
10    }
11
12    public boolean equals(Object obj) {  <-- 학번과 이름이 같다면 true를 리턴
13      if(obj instanceof Student) {
14        Student student = (Student) obj;
15        return (sno==student.sno) && (name.equals(student.name));
16      } else {
17        return false;
18      }
19    }
20
21    public int hashCode() {  <-- 학번과 이름이 같다면 동일한 값을 리턴
22      return sno + name.hashCode();
23    }
24  }
```

Map 컬렉션

- 예시 – 학번과 이름이 동일한 경우 같은 키로 인식

```
01  package sec01.exam07;
02
03  import java.util.*;
04
05  public class HashMapExample {
06    public static void main(String[] args) {
07      Map<Student, Integer> map = new HashMap<Student, Integer>();
08
09      map.put(new Student(1, "홍길동"), 95);
10      map.put(new Student(1, "홍길동"), 95);
11
12      System.out.println("총 Entry 수: " + map.size());
13    }
14  }
```

학번과 이름이 동일한 Student를 키로 저장

저장된 총 Map.Entry 수 얻기

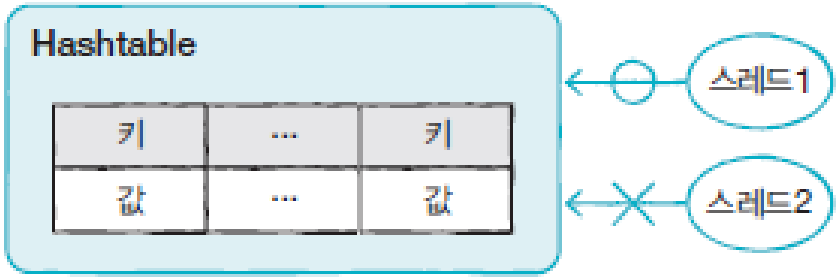
실행결과

```
총 Entry 수: 1
```

Map 컬렉션

❖ Hashtable

- HashMap과 동일한 내부 구조
- 동기화된 메소드로 구성되어 멀티 스레드가 동시에 Hashtable 메소드 실행할 수 없으며, 하나의 스레드가 실행을 완료해야만 다른 스레드 실행할 수 있음
- 키로 사용할 객체를 hashCode()와 equals() 메소드 재정의하여 동등 객체 될 조건 정해야



- 키 타입과 값 타입 지정하고 기본 생성자 호출하여 생성

```
Map<K, V> map = new Hashtable<K, V>();
    ↑  ↑                  ↑  ↑
  키 타입 값 타입        키 타입 값 타입
```

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

← Hashtable의 K와 V 타입 파라미터를 생략하면 왼쪽 Map에 지정된 타입을 따라 감

Map 컬렉션

- 예시 – 아이디와 비밀번호 검사하기

```
01  package sec01.exam08;
02
03  import java.util.*;
04
05  public class HashTableExample {
06    public static void main(String[] args) {
07      Map<String, String> map = new Hashtable<String, String>();
08
09      map.put("spring", "12");
10      map.put("summer", "123");
11      map.put("fall", "1234");
12      map.put("winter", "12345");
13
14      Scanner scanner = new Scanner(System.in);
15
16      while(true) {
17        System.out.println("아이디와 비밀번호를 입력해주세요.");
18        System.out.print("아이디: ");
19        String id = scanner.nextLine();
20
```

아이디와 비밀번호를 미리 저장

키보드로부터 입력된 내용을 받기 위해 생성

키보드로 입력한 아이디를 읽음

Map 컬렉션

```
21          System.out.print("비밀번호: ");
22          String password = scanner.nextLine();
23          System.out.println();
24
25          if(map.containsKey(id)) {
26            if(map.get(id).equals(password)) {
27              System.out.println("로그인되었습니다.");
28              break;
29            } else {
30              System.out.println("비밀번호가 일치하지 않습니다.");
31            }
32          } else {
33            System.out.println("입력하신 아이디가 존재하지 않습니다");
34          }
35        }
36      }
37    }
```

키보드로 입력한 비밀번호를 읽음

아이디인 키가 존재하는지 확인

비밀번호를 비교

실행결과

```
아이디와 비밀번호를 입력해주세요
아이디: summer
비밀번호: 123

로그인되었습니다.
```

키워드로 끝내는 핵심 포인트

- 컬렉션 프레임워크 : 널리 알려진 자료구조 사용하여 객체를 효율적으로 추가, 삭제, 검색할 수 있도록 인터페이스와 구현 클래스를 java.util 패키지에서 제공하는데 이들을 총칭하여 컬렉션 프레임워크라 한다

- List 컬렉션 : List 컬렉션은 배열과 비슷하게 객체를 인덱스로 관리한다. 차이점은 저장용량이 자동으로 증가하며 객체 저장 시 자동 인덱스가 부여된다는 것이다. 또한 추가, 삭제, 검색을 위한 다양한 메소드가 제공된다

- Set 컬렉션 : Set 컬렉션은 저장 순서 유지되지 않으며, 객체를 중복해서 저장할 수 없고, 하나의 null만 저장할 수 있다.

- Map 컬렉션 : Map 컬렉션은 키와 값으로 구성된 Map.Entry 객체를 저장하는 구조를 가지고 있으며, 여기서 키와 값은 모두 객체이다. 키는 중복 저장될 수 없지만 값은 중복 저장될 수 있다.

확인문제

- 자바의 컬렉션 프레임워크에 대한 설명으로 맞는 것에 O, 틀린 것에 X 하세요
  - List 컬렉션은 인덱스로 객체를 관리하며 중복 저장을 허용한다 ( )
  - Set 컬렉션은 순서 유지하지 않으며 중복 저장을 허용하지 않는다 ( )
  - Map 컬렉션은 키와 값으로 구성된 Map.Entry를 저장한다 ( )
  - List와 Set은 모두 하나의 null만 저장 가능하다 ( )

- 싱글 스레드 환경에서 Board 객체를 저장 순서에 맞게 읽고 싶습니다. 가장 적합한 컬렉션을 생성하도록 밑줄 친 부분에 코드를 적성해보세요

_____ 변수 = new _____

(타입) (생성자 호출)

- 싱글 스레드 환경에서 학번(String)을 키로, 점수(Integer)를 값으로 저장하는 가정 적합한 컬렉션을 생성하도록 밑줄 친 부분에 코드를 작성해보세요

_____ 변수 = new _____

(타입) (생성자 호출)

확인문제

❖ HashSet에 Student 객체를 저장하려 합니다. 학번이 같으면 동일한 Student 라고 가정하고 중복 저장이 되지 않도록 하고 싶습니다. Student 클래스에서 재정의해야 하는 hashCode()와 equals() 메소드의 내용을 채워보세요. Student의 해시코드는 학번이라 가정합니다.

Student 중복 저장 방지 소스 코드 HashSetExample.java

```
01  package sec01.verify.exam08;
02
03  import java.util.HashSet;
04  import java.util.Iterator;
05  import java.util.Set;
06
07  public class HashSetExample {
08    public static void main(String[] args) {
09      Set<Student> set = new HashSet<Student>();
10
11      set.add(new Student(1, "홍길동"));
12      set.add(new Student(2, "신용권"));
13      set.add(new Student(1, "조민우"));  <-------- 학번이 같으므로 저장되지 않음
14
15      Iterator<Student> iterator = set.iterator();
16      while(iterator.hasNext()) {
17        Student student = iterator.next();
18        System.out.println(student.studentNum + ":" + student.name);
19      }
20    }
21  }
```

실행결과

```
1:홍길동
2:신용권
```

확인문제

hashCode( )와 equals() 재정의 소스 코드 Student.java

```
01  package sec01.verify.exam08;
02
03  public class Student {
04    public int studentNum;
05    public String name;
06
07    public Student (int studentNum, String name) {
08      this.studentNum = studentNum;
09      this.name = name;
10    }
11
12    @Override
13    public int hashCode() {
14      //코드 작성
15    }
16
17    @Override
18    public boolean equals(Object obj) {
19      //코드 작성
20    }
21  }
```

확인문제

❖ HashMap에 아이디(String)와 점수(Integer)가 저장되어 있습니다. 실행결과와 같이 평균 점수를 출력하고 최고 점수와 최고 점수를 받은 아이디를 출력해보세요

점수 관리 소스 코드 MapExample.java

```
01  package sec01.verify.exam09;
02
03  import java.util.HashMap;
04  import java.util.Map;
05  import java.util.Set;
06
07  public class MapExample {
08  public static void main(String[] args) {
09      Map<String,Integer> map = new HashMap<String,Integer>();
10      map.put("blue", 96);
11      map.put("hong", 86);
12      map.put("white", 92);
13
14      String name = null;    //최고 점수를 받은 아이디 저장
15      int maxScore = 0;      //최고 점수 저장
16      int totalScore = 0;    //점수 합계 저장
17
18      //작성 위치
19    }
20  }
```

실행결과

```
평균점수: 91
최고점수: 96
최고점수를 받은 아이디: blue
```


혼자 공부하는 자바
혼자 공부하는 자바
1:1 과외처럼 공부하고 싶을 때
작심삼일, 하다가 포기하게 될 때
어제 배운 내용이 생각나지 않을 때
신용권 지음
JAVA 8&11 지원
245개의 손코딩 예제
무료 동영상 강의
별책 용어 노트
한빛미디어

Thank You!