13-1. 컬렉션 프레임워크

혼자 공부하는 자바 (신용권 저)

목차

시작하기 전에

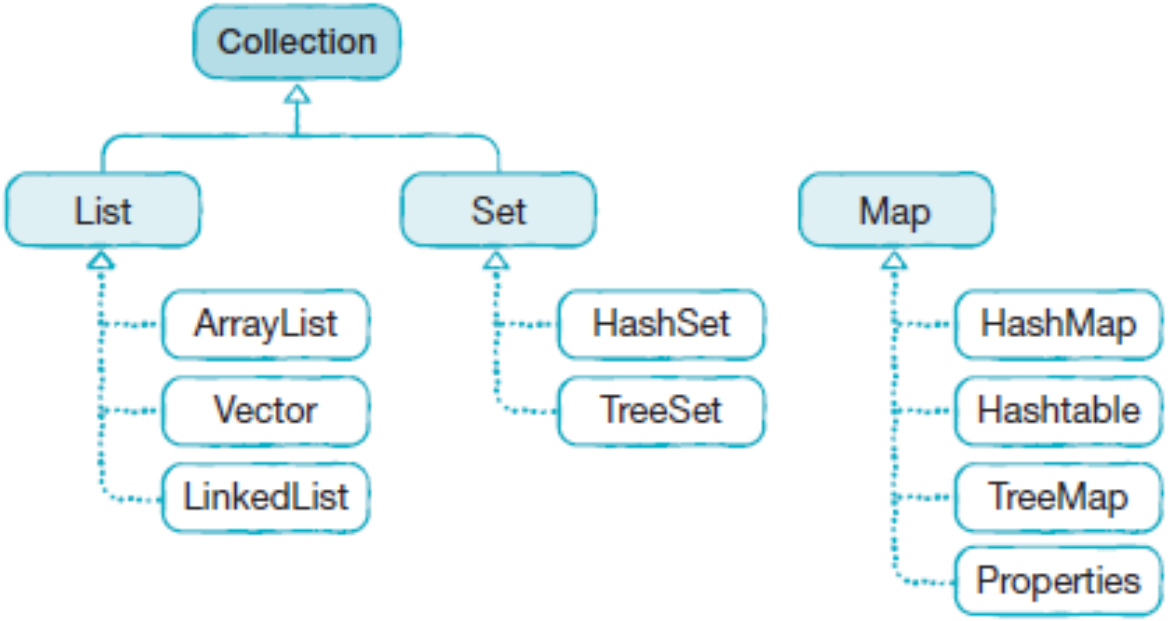
[핵심 키워드] : 컬렉션 프레임워크, List 컬렉션, Set 컬렉션, Map 컬렉션

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

시작하기 전에

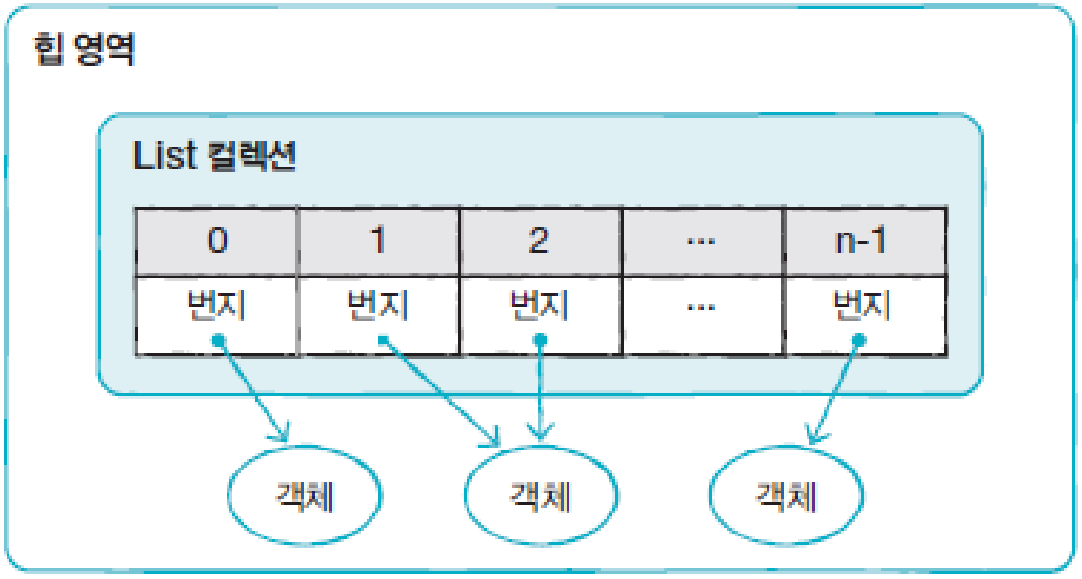
❖ 컬렉션 프레임워크 (Collection Framework)

- 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++) {   ← 저장된 총 객체 수만큼 루핑
  String str = list.get(i);
}
      ↑
i 인덱스에 저장된 String 객체를 가져옴
```

  - for문 이용

```
    String 객체를 하나씩 가져옴
        ↓
for(String str : list) {   ← 저장된 총 객체 수만큼 루핑
}
```

List 컬렉션

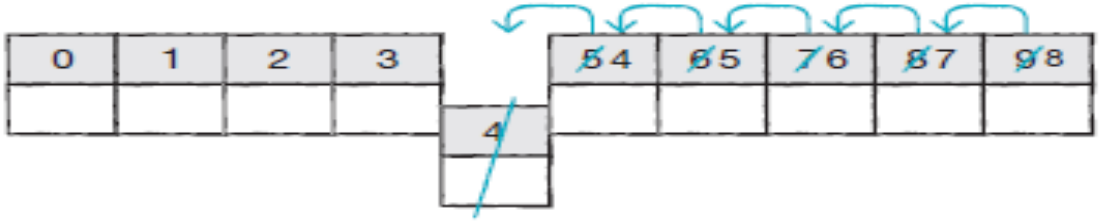
❖ ArrayList

- 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



List 컬렉션

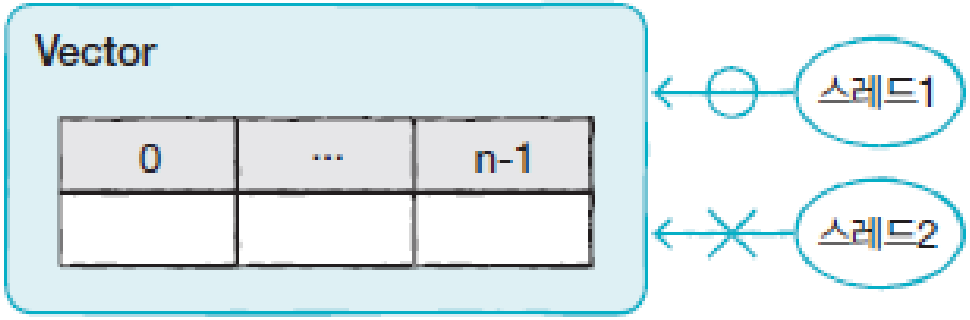
❖ Vector

- 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

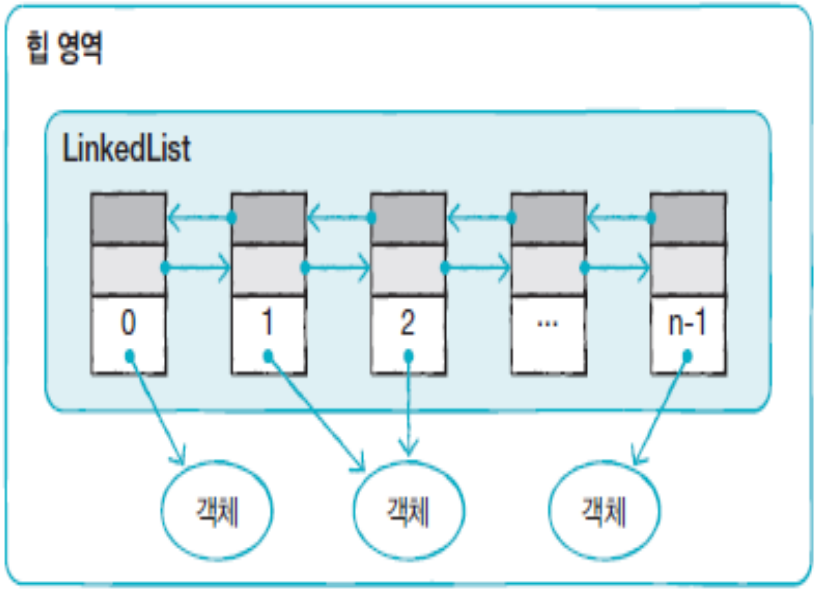
- 동기화된 메소드로 구성되어 멀티 스레드가 동시에 Vector의 메소드들 실행할 수 없고, 하나의 스레드가 메소드 실행 완료해야만 다른 스레드가 메소드 실행할 수 있음
- 멀티 스레드 환경에서 안전하게 객체 추가 및 삭제할 수 있음
  - 스레드에 안전 (thread safe)

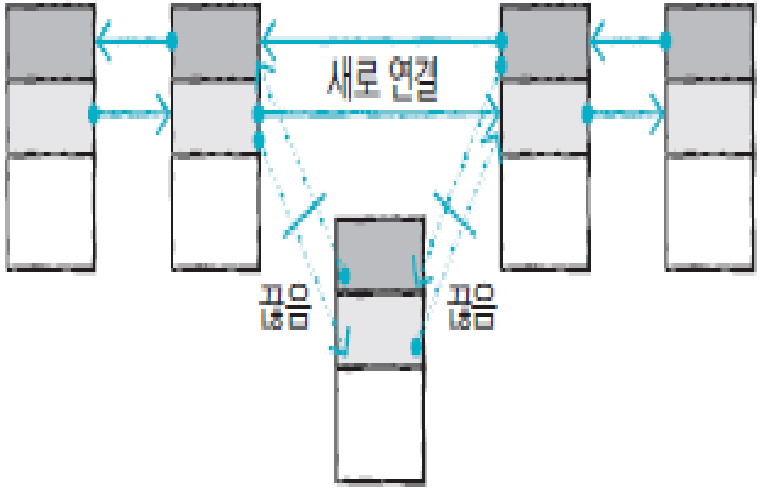


List 컬렉션

❖ 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

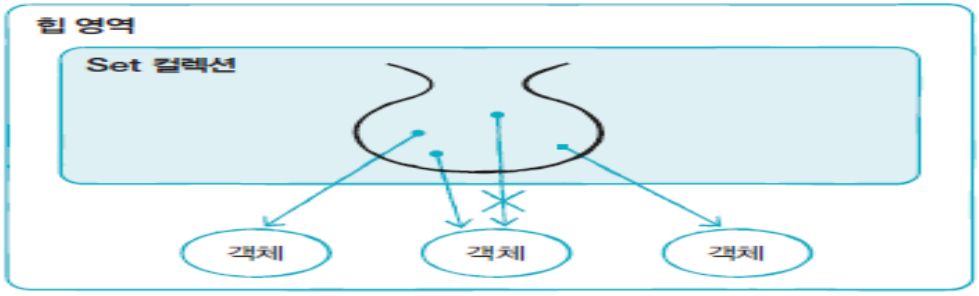
```
List<E> list = new LinkedList<E>();
List<E> list = new LinkedList<>();
```

LinkedList의 E 타입 파라미터를 생략하면 왼쪽 List에 지정된 타입을 따라 감

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

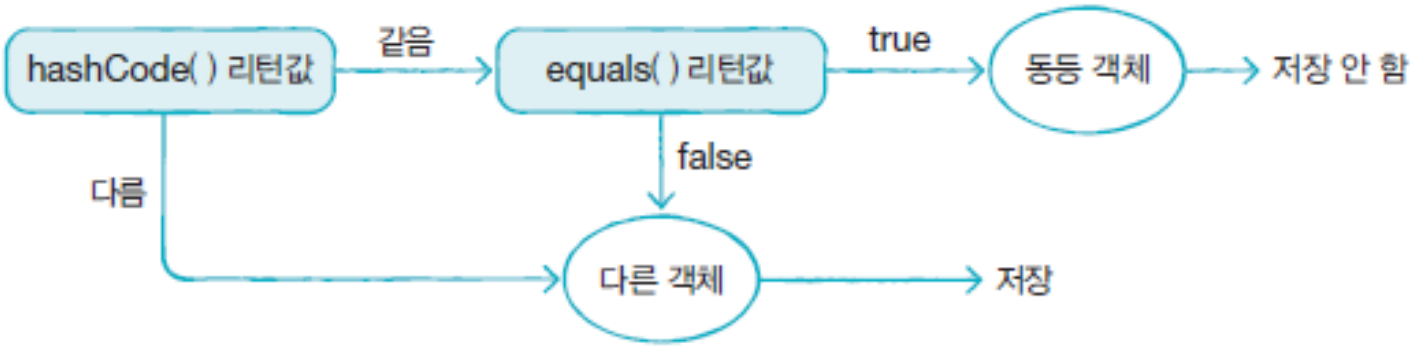
- Set 인터페이스의 구현 클래스
- 기본 생성자 호출하여 생성

```
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

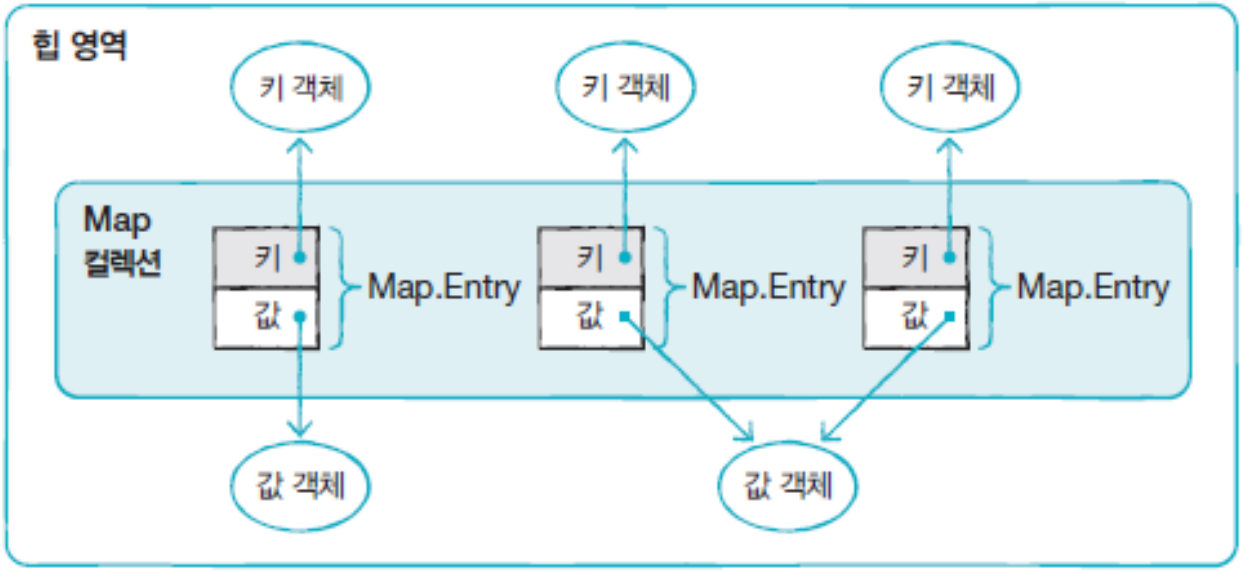
- 객체를 순서 없이 저장하되 동일한 객체는 중복 저장하지 않음
- 객체 저장 전 객체의 hashCode() 메소드 호출하여 해시코드 얻어내고 이미 저장된 객체의 해시코드와 비교



Map 컬렉션

❖ 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

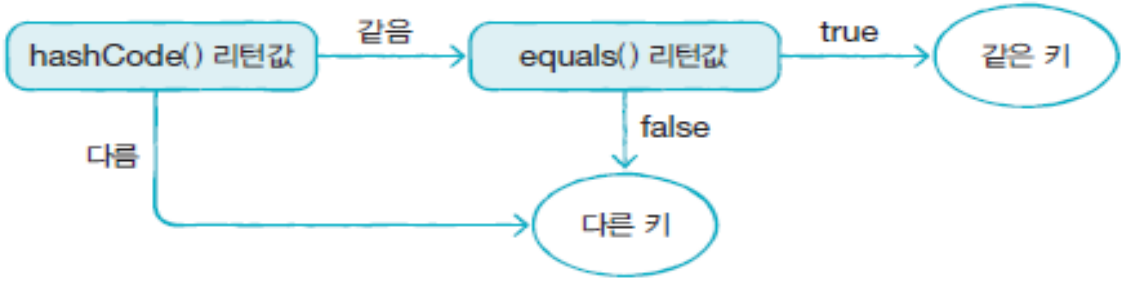
- entrySet() 메소드로 모든 Map.Entry를 Set 컬렉션으로 얻은 뒤 반복자 통해 Map.Entry() 하나씩 얻고 getKey()와 getValue() 메소드 이용해 키와 값 얻음

```
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
```

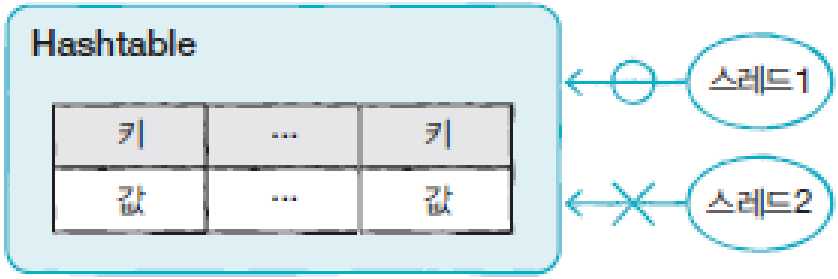
```
Map<String, Integer> map = new HashMap<String, Integer>();
Map<String, Integer> map = new HashMap<>();
```

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

Map 컬렉션

❖ 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

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.

13-2. LIFO와 FIFO 컬렉션

혼자 공부하는 자바 (신용권 저)

한빛미디어 Hanbit Media, Inc.

목차

시작하기 전에

[핵심 키워드] : Stack, Queue

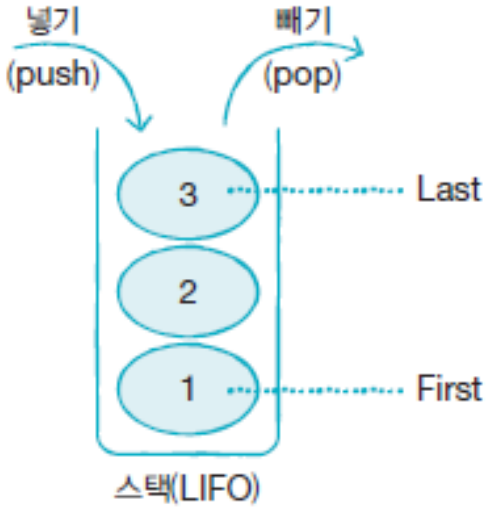
[핵심 포인트]

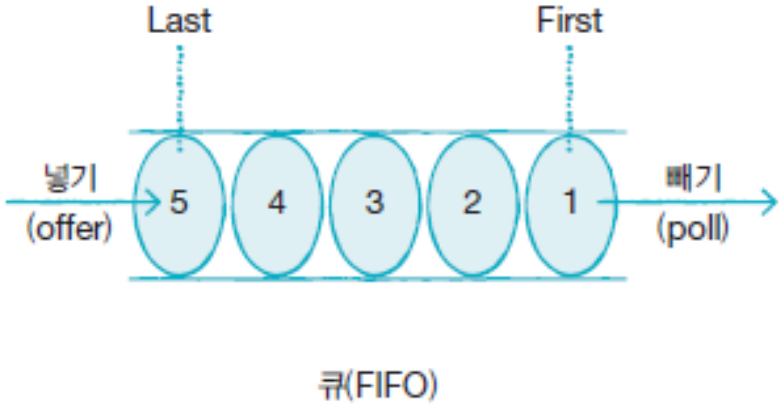
컬렉션 프레임워크에는 LIFO(후입선출) 자료구조를 제공하는 Stack 클래스와 FIFO(선입선출) 자료구조를 제공하는 Queue 인터페이스가 있습니다. 이번 절에서는 Stack 클래스와 Queue 인터페이스에 대해 살펴본다.

시작하기 전에

- 후입선출 (LIFO : List In First Out)
  - 나중에 넣은 객체가 먼저 빠져나가는 자료구조
- 선입선출 (FIFO : First In First Out)
  - 먼저 넣은 객체가 먼저 빠져나가는 자료구조

- 컬렉션 프레임워크에는 LIFO 자료구조 제공하는 Stack 클래스와 FIFO 자료구조 제공하는 Queue 인터페이스 제공됨





Stack

❖ Stack

- LIFO 자료구조 구현한 클래스

리턴 타입	메소드	설명
E	push(E item)	주어진 객체를 스택에 넣습니다.
E	peek()	스택의 맨 위 객체를 가져옵니다. 객체를 스택에서 제거하지 않습니다.
E	pop()	스택의 맨 위 객체를 가져옵니다. 객체를 스택에서 제거합니다.

- Stack 객체 생성하려면 저장할 객체 타입을 E 타입 파라미터 자리에 표기하고 기본 생성자를 호출

```
Stack<E> stack = new Stack<E>();
Stack<E> stack = new Stack<>();
```

Stack의 E 타입 파라미터를 생략하면 왼쪽 Stack에 지정된 타입을 따라 감

Queue

❖ Queue

- FIFO 자료구조에서 사용되는 메소드 정의

리턴 타입	메소드	설명
boolean	offer(E e)	주어진 객체를 넣습니다.
E	peek()	객체 하나를 가져옵니다. 객체를 큐에서 제거하지 않습니다.
E	poll()	객체 하나를 가져옵니다. 객체를 큐에서 제거합니다.

- LinkedList 클래스

```
Queue<E> queue = new LinkedList<E>();
Queue<E> queue = new LinkedList<>();
```

LinkedList의 E 타입 파라미터를 생략하면 왼쪽 Queue에 지정된 타입을 따라 감

키워드로 끝내는 핵심 포인트

- Stack : 후입선출을 구현한 클래스.
- Queue : 선입선출에 필요한 메소드를 정의한 인터페이스. 구현 클래스로 LinkedList가 있음

Thank You!

혼자 공부하는 자바 (신용권 저)

한빛미디어 Hanbit Media, Inc.