[summary] 컬렉션 프레임워크

노트북:	0050_java		
만든 날짜:	2021-12-19 오후 8:17	수정한 날짜:	2022-08-23 오후 5:06
작성자:	nomad.lab.kr@gmail.com		
URL:	about:blank		

컬렉션 프레임워크 : 자바에서 제공하는 라이브러리.
널리 알려져 있는 **자료 구조 Data Structure**를 사용해서
객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 인터페이스와 구현 클래스를 java.util 패키지에서 제공.

* 자료 구조 data structure : **프로그램 실행 중 메모리에 자료를 유지, 관리하기 위해 사용.**

컬렉션 Collection : 객체의 저장
프레임워크 Framework : 사용 방법을 정해 놓은 라이브러리

컬렉션 프레임워크의 전체 구조는 Collection 인터페이스와 Map 인터페이스 기반으로 이루어져 있음.

Collection 인터페이스	하나의 자료를 모아서 관리하는 데 필요한 기능을 제공
Map 인터페이스	쌍(pair)으로 된 자료들을 관리하는데 유용한 기능을 제공

1. Collection 인터페이스

하위에는 List 인터페이스와 Set 인터페이스가 있음.

분류	설명
List 인터페이스	순서가 있는 자료 관리, 중복 허용. 이 인터페이스를 구현한 클래스는 ArrayList, Vectior, LinkedList, Stack, Queue 등이 있음.

Set 인터페이스	순서가 정해져 있지 않음, 중복을 허용하지 않음. 이 인터페이스를 구현한 클래스는 HashSet, TreeSet 등이 있음.

Collection 인터페이스에 선언된 메서드 중 자주 사용하는 메서드.

메서드	설명
boolean add(E e)	Collection에 객체를 추가.
void clear()	Collection의 모든 객체를 제거.
Iterator<E> iterator()	Collection을 순환할 반복자 Iterator를 반환.
boolean remove(Object o)	Collection에 매개변수에 해당하는 인스턴스가 존재하면 제거.
int size()	Collection에 있는 요소의 개수를 반환.

1. List 컬렉션

배열과 비슷하게 객체를 인덱스로 관리.
배열과의 차이점은 저장 용량 capacity 이 자동으로 증가하며,
객체를 저장할 때 자동으로 인덱스가 부여됨.
그리고 추가, 삭제, 검색을 위한 다양한 메소드를 제공.

객체 자체를 저장하는 것이 아니라 객체의 번지를 참조함.
그래서 동일한 객체를 중복 저장할 수 있고 이 경우 동일한 번지가 참조.

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨 끝에 추가
	void add(int index, E element)	주어진 인덱스에 객체를 추가
	E set(int index, E	주어진 인덱스에 저장된 객

	element)	체를 주어진 객체로 바꿈
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 조사
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴
	boolean isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어 있는 전체 객체 수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	E remove(int index)	주어진 인덱스에 저장된 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

우리가 알고 있는 순차 자료 구조의 대표적인 예는 배열.
자바에서 배열을 구현한 대표 클래스로는 ArrayList, Vector가 있고,
배열과 다르지만 순차 자료 구조를 구현한 LinkedList가 있음.

1) ArrayList
기본 생성자로 ArrayList 객체를 생성하면 내부에 10개의 객체를 저장할 수 있는 초기 용량 capacity을 가지게 됨.
저장되는 객체 수가 늘어나면 용량이 자동으로 증가.

2) Vector
ArrayList와 동일한 내부 구조를 가지고 있슴
ArrayList와 다른 점은 Vector는 동기화된 synchronized 메소드로 구성이 되어 있어서,
멀티 스레드가 동시에 메소드를 실행할 수 없고, 하나의 스레드가 메소드를 실행을 완료해야만
다른 스레드가 메소드를 실행할 수 있슴

그래서 **멀티 스레드 환경에서 안전하게 객체를 추가, 삭제** 할 수 있슴

3) LinkedList

ArrayList와 사용 방법은 동일한데, 내부 구조는 완전 다름

* 배열과 LinkedList의 다른 점.
배열은 생성할 때 용량을 지정하고, 용량보다 더 많은 요소가 추가된 경우에 용량을 늘여가며 수행.
LinkedList는 요소를 추가할 때마다 동적으로 요소의 메모리를 생성하기 때문에
배열처럼 용량을 늘리고 요소 값을 복사하는 번거러움이 없음.

또한 LinkedList는 자료를 중간에 추가하거나 삭제할 때 자료의 이동이 배열보다 적음.

배열이 LinkedList 보다 효율적인 경우는 어떤 요소의 위치(i번째)를 찾는 경우.
배열은 물리적으로 연결된 자료 구조이므르 i번째 요소 메모리 위치를 바로 계산할 수 있어 접근이 빠름.
예를 들어 int 배열의 두 번째 자료는 처음 주소부터 8바이트 떨어진 위치.

ArrayList는 내부 배열에 객체를 저장해서 관리하지만, LinkedList는 인접 참조를 링크해서 체인처럼 관리
LinkedList에서 특정 인덱스의 객체를 제거하면 앞뒤 링크만 변경되고 나머지 링크는 변경되지 않음

끝에서 부터(순차적으로) 추가 또는 삭제하는 경우에는 ArrayList가 빠르지만
중간에 추가, 삭제하는 경우는 앞뒤 링크 정보만 변경하므로 LinkedList가 더 빠름

구분	순차적으로 추가/삭제	중간에 추가/삭제	검색

ArrayList	빠르다	느리다	빠르다
LinkedList	느리다	빠르다	느리다

사용하는 자료의 변동(삽입, 삭제)이 많은 경우는 LinkedList를, 자료 변동이 거의 없는 경우에는 배열을 사용하는 것이 효율적.

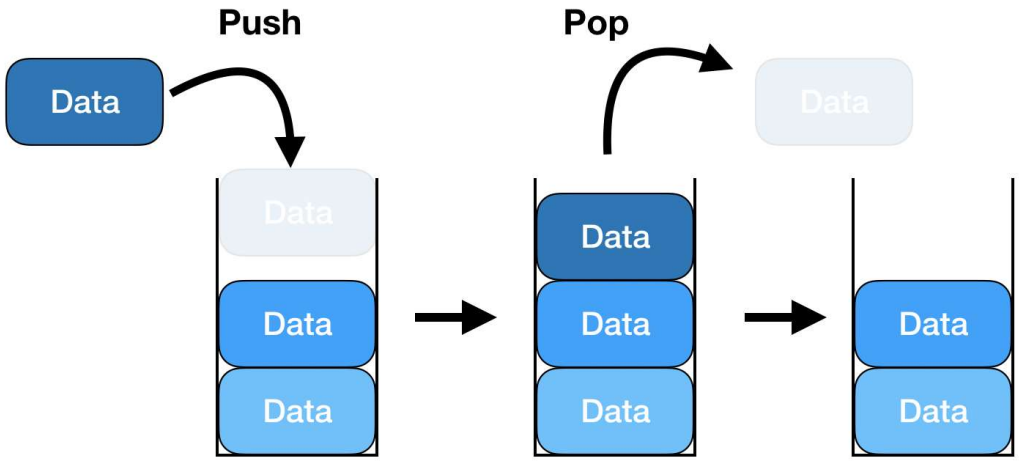
2. LIFO와 FIFO 컬렉션

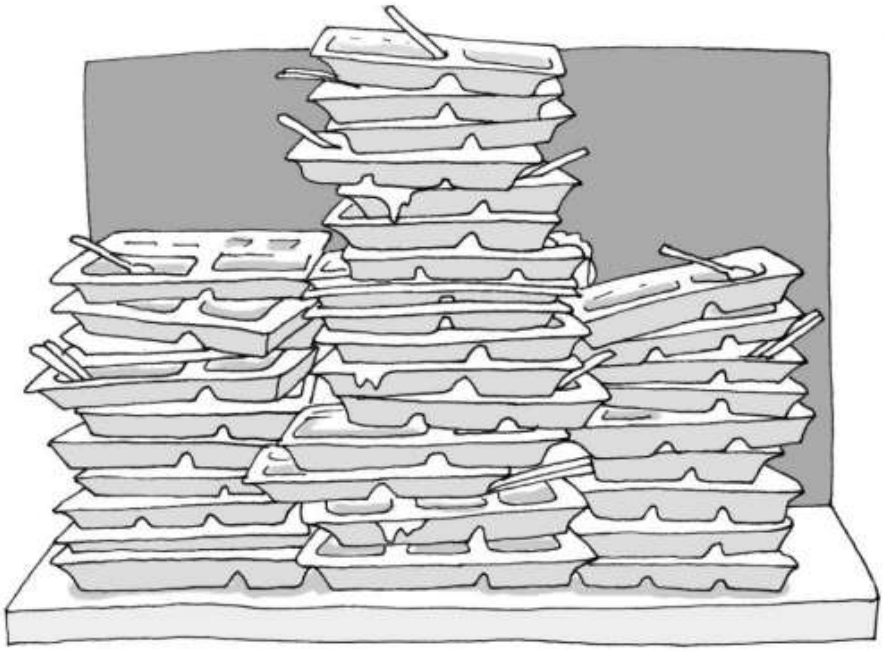
후입선출 LIFO Last In First Out 은 나중에 넣은 객체가 먼저 빠져나가는 구조
선입선출 FIFO First In First Out 은 먼저 넣은 객체가 먼저 빠져나가는 구조

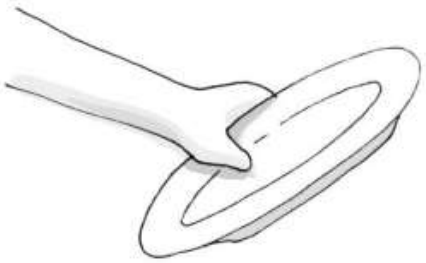
1) Stack

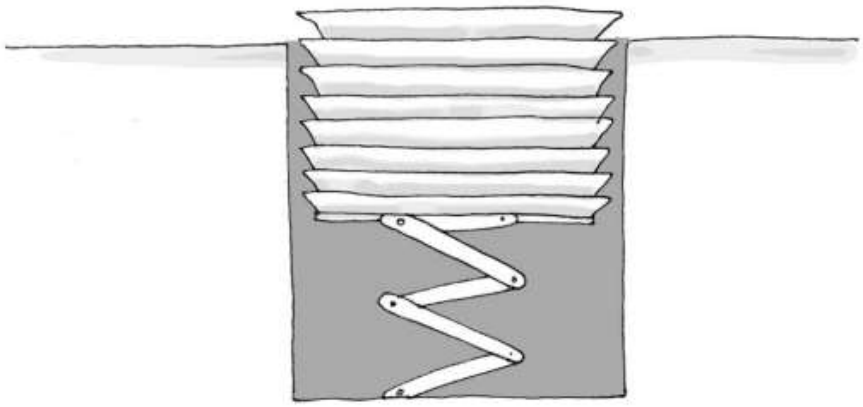
Stack 클래스는 LIFO 자료구조를 구현한 클래스

리턴 타입	메소드	설명
E	push(E Item)	주어진 객체를 스택에 넣음
E	peek()	스택의 맨 위 객체를 가져옴. 객체를 스택에서 제거하지 않음
E	pop()	스택의 맨 위 객체를 가져옴. 객체를 스택에서 제거







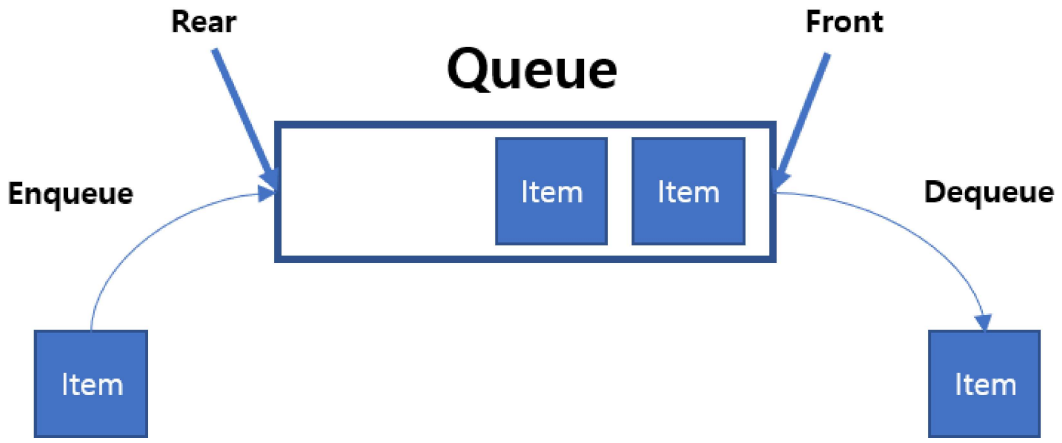


2) Queue

Queue 인터페이스는 FIFO 자료구조에서 사용되는 메소드를 정의

리턴 타입	메소드	설명
boolean	offer(E e)	주어진 객체를 넣음
E	peek()	객체 하나를 가져옴. 객체를 큐에서 제거하지 않음
E	poll()	객체 하나를 가져옴. 객체를 큐에서 제거

Queue 인터페이스를 구현한 대표적인 클래스는 LinkedList임



3. Set 컬렉션 : 중복되지 않게 자료를 관리

List 컬렉션은 객체의 저장 순서를 유지하지만, Set 컬렉션은 **저장 순서가 유지되지 않음**

객체를 **중복해서 저장할 수 없고, 하나의 Null만 저장할 수 있음**

인덱스로 관리하지 않기 때문에 인덱스를 매개값으로 갖는 메소드가 없음

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

* Collection 요소를 순회하는 Iterator
요소의 순회란? 컬렉션 프레임워크에 저장된 요소들을 하나씩 차례로 참조하는것

Set 컬렉션은 인덱스로 객체를 검색해서 가져오는 메소드가 없는 대신
전체 객체를 대상으로 한 번씩 반복해서 가져오는 반복자 Iterator를 제공함
(순서가 있는 List인터페이스의 경우는 Iterator를 사용 하지 않고 get(i) 메서드를 활용할 수 있음)
반복자는 Iterator 인터페이스를 구현한 객체를 말하는데, iterator() 메소드를 호출하면 얻을 수 있슴

Iterator 인터페이스에 선언된 메소드들

리턴 타입	메소드	설명
boolean	hasNext()	가져올 객체가 있으면 true를 리턴하고 없으면 false를 리턴
E	next()	컬렉션에서 하나의 객체를 가져옴
void	remove()	Set 컬렉션에서 객체를 제거 함

1) HashSet
Set 인터페이스의 구현 클래스
객체들을 **순서 없이 저장**하고 동일한 객체는 **중복 저장하지 않음**

HashSet이 판단하는 **동일한 객체란 꼭 같은 인스턴스를 뜻하지 않음**
HashSet은 객체를 **저장하기 전**에 먼저 객체의 **hashCode() 메소드들 호출해서 해시코드**를 얻어내고,
이미 저장되어 있는 객체들의 해시코드와 비교
만약 동일한 해시코드가 있다면 다시 **equals()메소드로 두 객체를 비교**해서
true가 나오면 동일한 객체로 판단하고 중복 저장을 하지 않음

2) TreeSet
자바의 Collection 인터페이스나 Map 인터페이스를 구현한 클래스 중 **Tree로 시작**하는
클래스는 데이터를 추가한 후 결과를 출력하면 **결과 값이 정렬**이 됨.
Tree + Set은 **자료의 중복을 허용하지 않으면서 출력 결과 값을 정렬**하는 클래스.

2. Map 인터페이스

하나가 아닌 쌍 pair으로 되어 있는 자료를 관리하는 메서드들이 선언되어 있음.
key-value 쌍이라고 표현하는데 이 때 키 값은 중복될 수 없음.
학번과 학생 이름처럼 쌍으로 되어 있는 자료를 관리할 때 사용하면 편리.

key	value
이름	손민수
나이	30세
직업	회사원, 프리랜서
취미	수영, 등산
특기	수영

Map은 기본적으로 검색용 자료 구조.
즉 어떤 Key 값을 알고 있을 때 value를 찾기 위한 자료 구조.
Map 인터페이스에 선언된 메소드 중 주요 메서드.

메서드	설명
V put(K key, V value)	key에 해당하는 value 값을 map에 넣음.
V get(K key)	key에 해당하는 value 값을 반환.
boolean isEmpty()	Map이 비었는지 여부를 반환.
boolean containsKey(Object Key)	Map에 해당 key가 있는지 여부를 반환.
boolean containsValue(Object Value)	Map에 해당 value가 있는지 여부를 반환.
Set keyset()	key 집합을 Set으로 반환. (중복 안 되므로 Set).
Collection values()	value를 Collection으로 반환 (중복 무관).
V remove(Object key)	key가 있는 경우 삭제.
boolean remove(Object key, Object value)	key가 있는 경우 key에 해당하는 value가 매개변수와 일치할 때 삭

	제.

1. Map 컬렉션 : 검색을 위한 자료구조

키 key 와 값 value 으로 구성된 **Map.Entry 객체**를 저장하는 구조를 가지고 있음
Entry는 Map 인터페이스 내부에 선언된 중첩 인터페이스이고 키와 값은 모두 객체임
키는 중복 저장될 수 없지만 값은 중복 저장될 수 있고, 만약 **기존에 저장된 키와 동일한 키로 값을 저장하면**
기존의 값은 없어지고 **새 값으로 대체**됨

Map 인터페이스의 메소드.
키로 객체를 관리하기 때문에 키를 매개값으로 갖는 메소드가 많음

기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키로 값을 저장. 새로운 키일 경우 null을 리턴하고 동일한 키가 있을 경우 값을 대체하고 이전 값을 리턴
객체 검색	boolean containsKey(Object Key)	주어진 키가 있는지 여부를 확인
	boolean containsValue(Object Value)	주어진 값이 있는지 여부를 확인
	Set<Map.Entry<K, V> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set에 담아서 리턴
	V get(Object key)	주어진 키가 있는 값을 리턴
	boolean isEmpty()	컬렉션이 비어 있는지 여부

		를 확인
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 모든 값을 Collection에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry를 삭제하고 값을 리턴

1) HashMap

Map 인터페이스를 구현한 대표적인 Map 컬렉션
HashMap의 키로 사용할 객체는 hashCode()와 equals() 메소드 재정의해서 동등 객체가 될 조건을 정해야 함
객체가 달라도 동등 객체라면 같은 키로 간주하고 중복 저장되지 않도록 하기 위함
동등 객체의 조건은 hashCode()의 리턴 값이 같아야 하고, equals() 메소드가 true를 리턴 해야 함

주로 키 타입은 String을 많이 사용하는데, String은 문자열이 같을 경우 동등 객체가 될 수 있도록 hashCode()와
equals() 메소드가 재정의 되어 있음

2) HashTable

HashMap과 동일한 구조를 가지고 있어서, 키로 사용할 객체
는 hashCode()와 equals() 메소드를 재정의해서
동등 객체가 될 조건을 정해야 함

차이점은 Hashtable은 동기화된 synchronized 메소드로 구성이 되어 있기 때문에
멀티 스레드가 동시에 메소드를 실행할 수 없고, 하나의 스레드가 메소드를 실행을 완료해야만

다른 스레드가 메소드를 실행할 수 있슴
그래서 멀티 스레드 환경에서 안전하게 객체를 추가, 삭제할 수 있기 때문에 HashTable은
스레드에 안전함

3) TreeMap

Map 인터페이스를 구현한 클래스 중 key 값으로 자료를 정렬할 때 사용.
key 값으로 정렬하므로 key 값에 해당하는 클래스에 Comparable이나 Comparator 인터페이스를 구현해야 함.