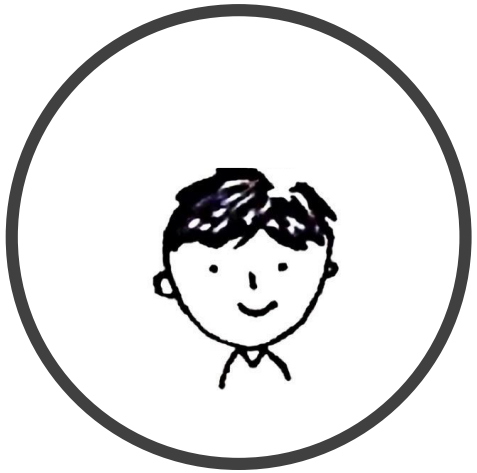
Collection API

Javacafe 지대철

소개

- Job : 4학년
- Email : puri8467@gmail.com
- Blog : https://blog.naver.com/puri8467
- 발표 예제 : https://github.com/Danpatpang/javacafe



문제

오늘 홈페이지에 접속한 사람들의 아이디 목록을 뽑아내세요.

접속 이력은 아래의 형식으로 되어있고. 이 파일을 읽어서 처리하는 프로그램을 만드세요.

```
접속 날짜                  접속자 IP    접속자 ID  URL
2019-04-27 10:00:20_192.168.1.1_tomato_http://00000.com/login
2019-04-27 10:00:22_192.168.1.1_banana_http://00000.com/login
2019-04-27 14:21:00_192.168.1.1_orange_http://00000.com/login
2019-04-27 17:00:00_192.168.1.1_mango_http://00000.com/login
2019-04-27 17:00:02_192.168.1.1_tomato_http://00000.com/login
                                    ⋮
```

배열로 만들어보자.

1. 배열 생성 시 크기를 미리 할당해야 한다.

2. 중복되는 접속자는 조회 후 삭제해야 한다.

3. 새로운 접속자는 배열에 저장해야 한다.

배열로 만들어보자.

1. 배열 생성 시 크기를 미리 할당해야 한다.

   ➔ 얼만큼??? 일단 크게 주자!

2. 중복되는 접속자는 조회 후 삭제해야 한다.

   ➔ 비어 있는 인덱스 발생.

3. 새로운 접속자는 어디에 저장할 것인가?

   ➔ 마지막으로 저장된 인덱스 다음? 비어 있는 인덱스?

Collection Framework

Collection - 객체를 수집해서 저장하는 역할.

Framework - 사용 방법을 미리 정해 놓은 라이브러리.

Collection Framework

널리 알려져 있는 자료구조를 바탕으로,

객체들을 효율적으로 추가, 삭제, 검색할 수 있도록

Java.util 패키지에 컬렉션과 관련된 인터페이스와 클래스들을 포함시켜 놓은 것.

참고 링크

자료 구조

https://www.edwith.org/datastructure-2019s2

https://www.edwith.org/datastructure-2019s

Java 알고리즘

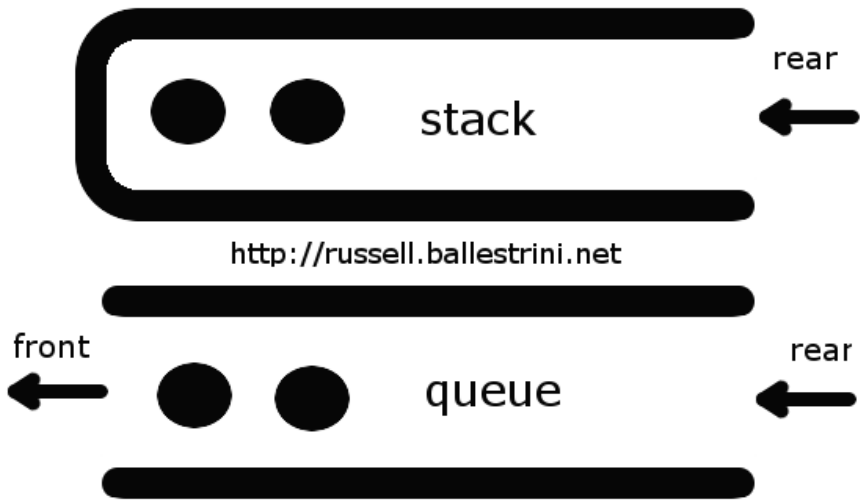
https://github.com/TheAlgorithms/Java

알고리즘 시각화

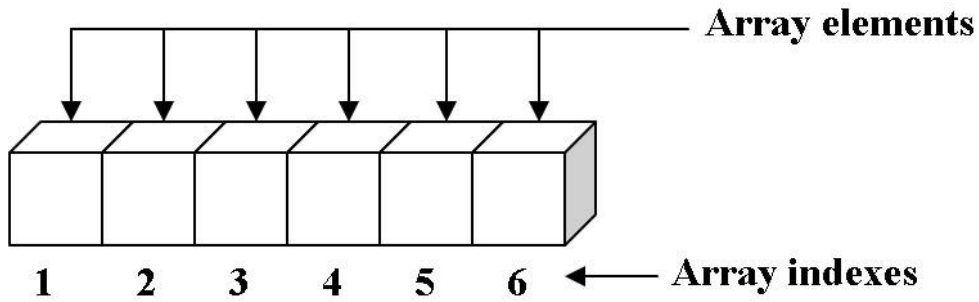
https://visualgo.net/ko

https://github.com/hahnlee/ipytracer

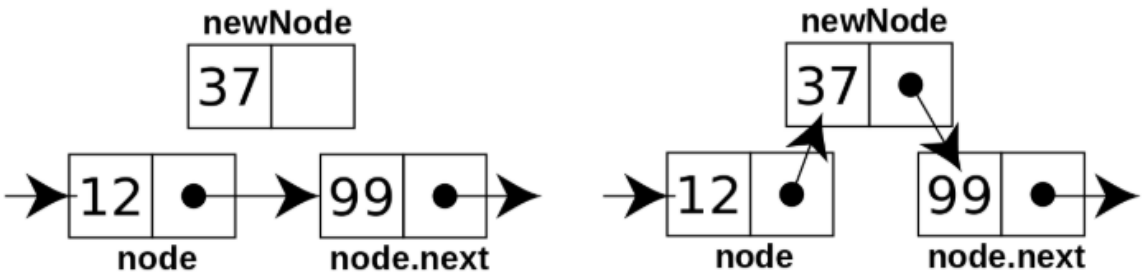
자료 구조



출처 : http://russell.ballestrini.net/occupy-wall-street-stack-vs-queue/

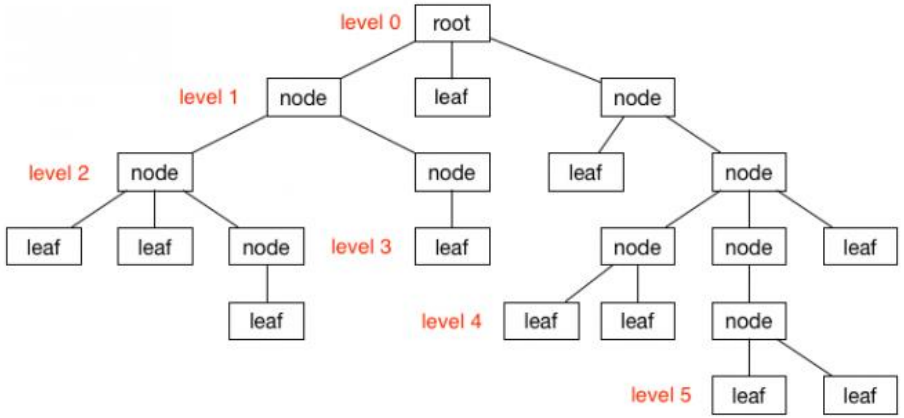


One-dimensional array with six elements

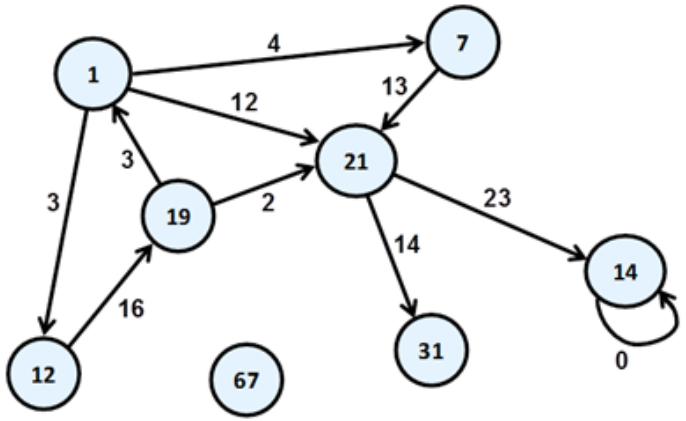


출처 : https://en.wikipedia.org/wiki/Linked_list

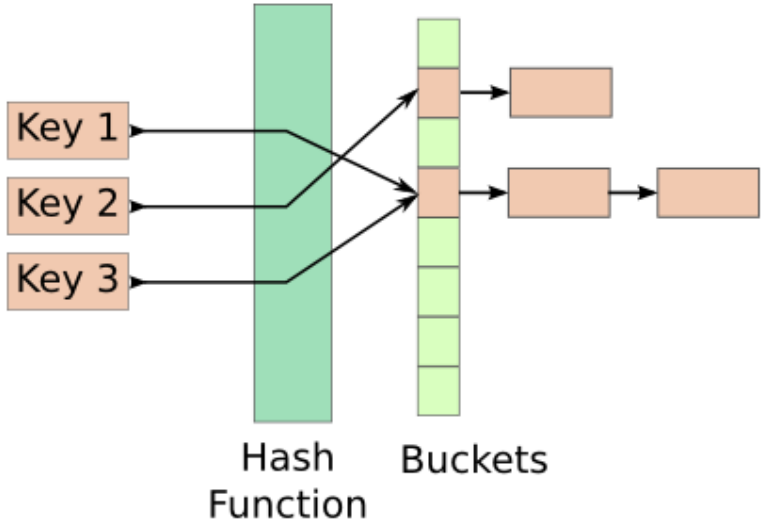
자료 구조



출처 : https://www.raywenderlich.com/138190/swift-algorithm-club-swift-tree-data-structure

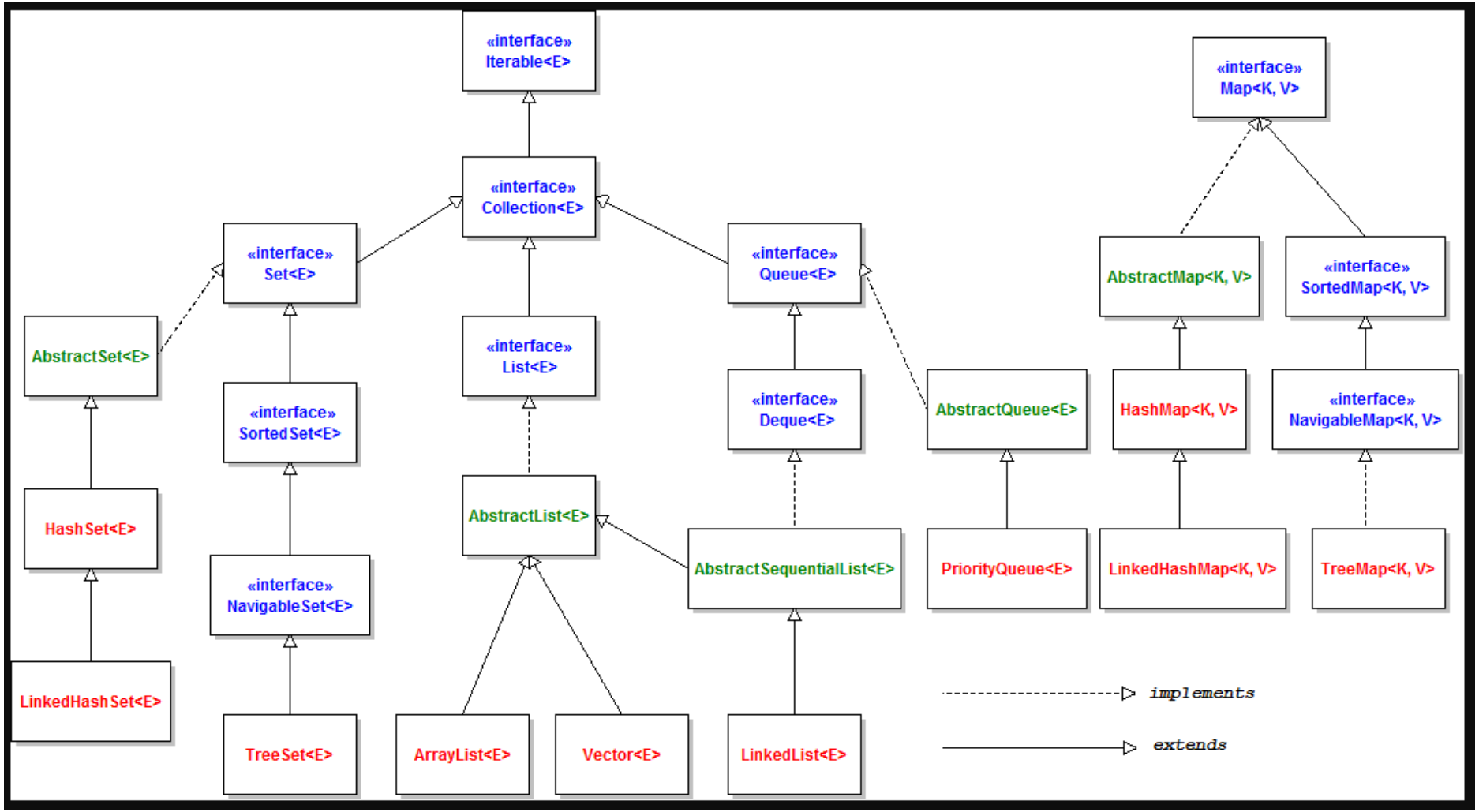


출처 : http://www.introprogramming.info/english-intro-csharp-book/read-online/chapter-17-trees-and-gra



출처 : http://vhanda.in/blog/2012/07/shared-memory-hash-table/

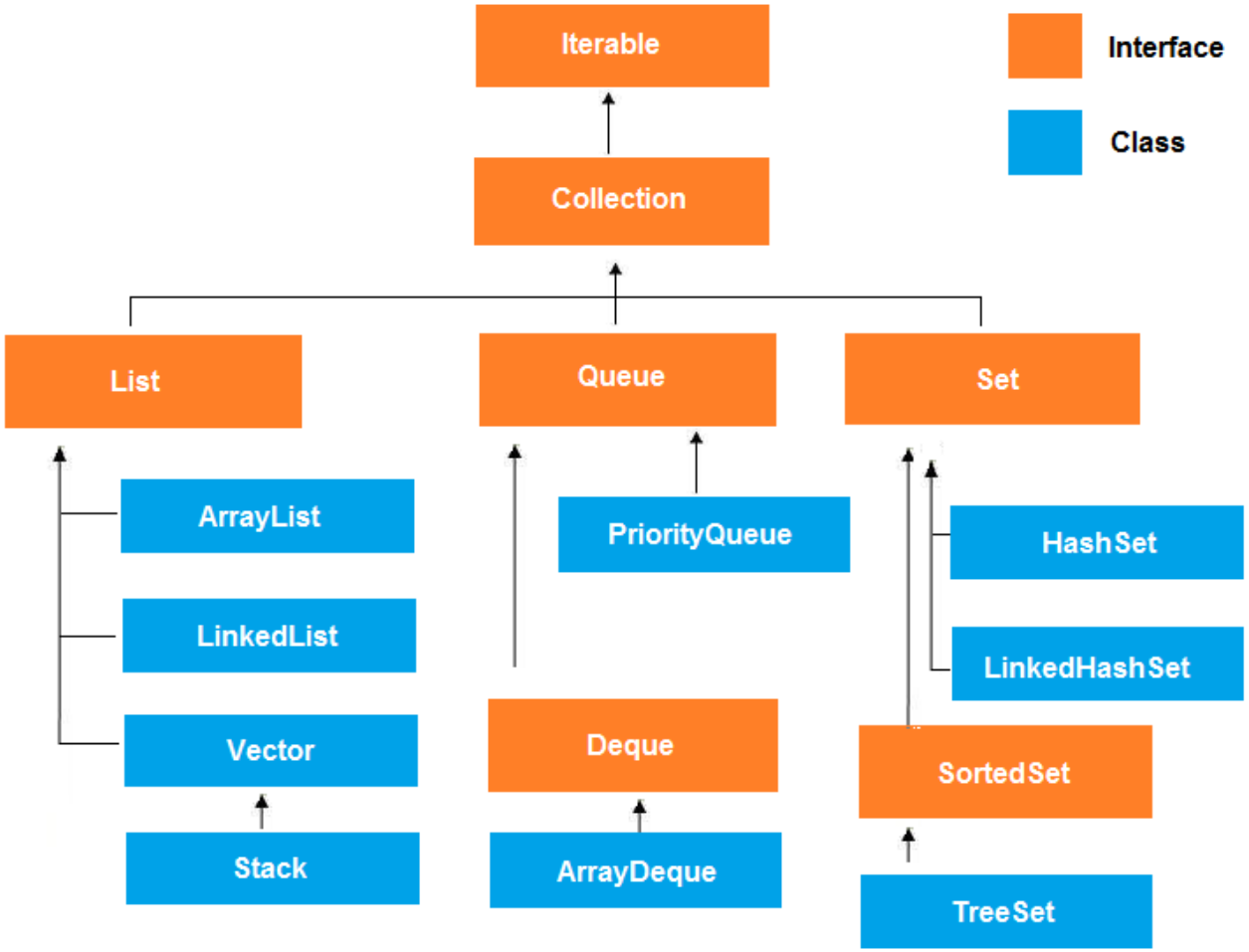
Diagram



https://www.codejava.net/java-core/collections/overview-of-java-collections-framework-api-uml-diagram

Collection

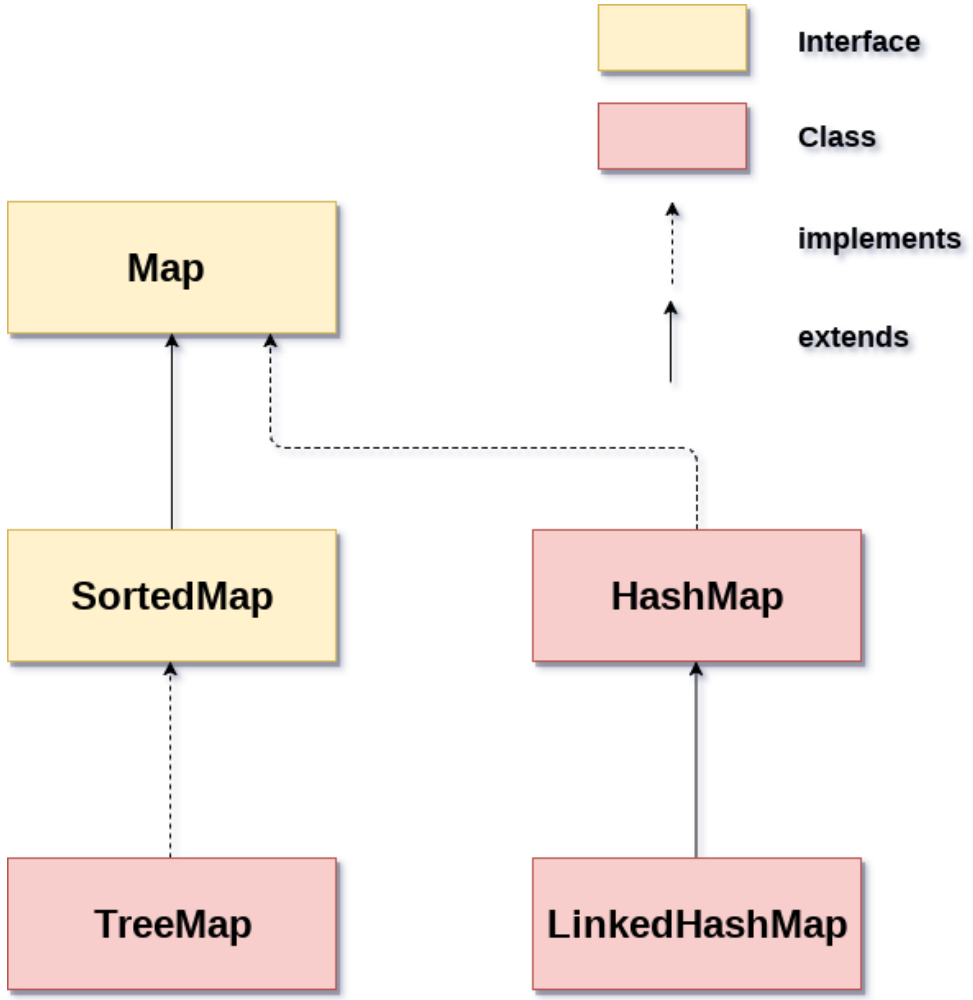
https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html



https://javatutorial95.blogspot.com/2017/03/Java-Collection-Framework.html

Map

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html



https://www.javatpoint.com/java-map

특징

인터페이스 분류		특징	구현 클래스
Collection	List	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList LinkedList Vector
	Set	- 순서를 유지하지 않고 저장 - 중복 저장 안됨	HashSet TreeSet
	Queue	- FIFO 구조	ArrayDeque PriorityQueue
Map		- 키와 값의 쌍으로 저장 - 키는 중복 안됨	HashMap HashTable TreeMap Properties

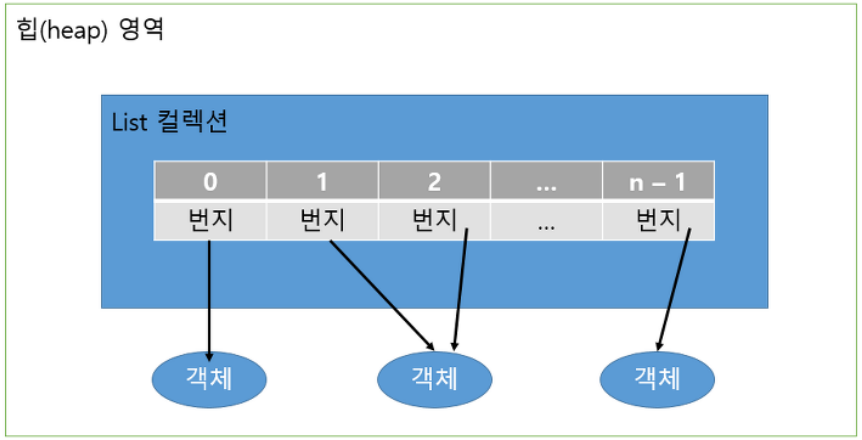
List

객체를 일렬로 늘어놓은 구조.

객체를 인덱스로 관리. 저장 시 자동으로 인덱스 부여.

인덱스로 객체를 검색, 삭제할 수 있는 기능 제공.

객체의 번지를 참조하기 때문에 중복 시 동일한 객체를 참조.



List

기능	메소드	설명
객체 추가	boolean add(E a)	주어진 객체를 맨 끝에 추가.
	void add(int index, E element)	주어진 인덱스에 객체를 추가.
	set(int index, E element)	주어진 인덱스에 저장된 객체 변경.
객체 검색	boolean contaions(Object o)	주어진 객체가 저장되어 있는지에 대한 여부.
	E get(int index)	주어진 인덱스에 저장된 객체 리턴.
	boolean isEmpty()	컬렉션이 비어 있는지 조사.
	int size()	저장되어 있는 전체 객체 수를 리턴.
객체 삭제	void clear()	저장된 모든 객체 삭제.
	E remove(int index)	주어진 인덱스에 저장된 객체 삭제.
	boolean remove(Object o)	주어진 객체를 삭제.

ArrayList

저장 용량을 초과한 객체들이 들어오면 자동적으로 저장 용량을 늘린다.

deafult 저장 공간 : 10

[~jdk1.4]

```
List list = new ArrayList();

list.add("보라돌이");

Object obj = list.get(0);

String name = (String) obj;
```

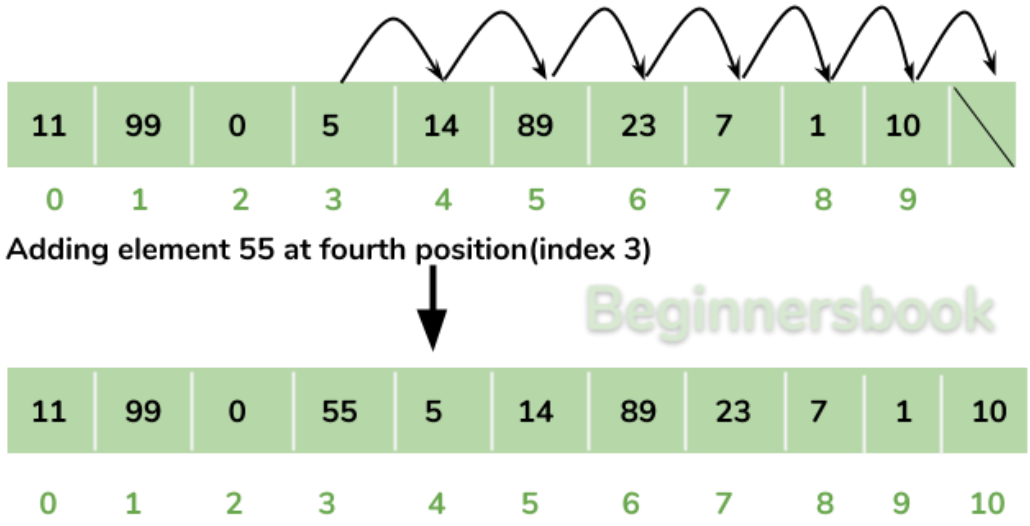
[jdk1.5~]

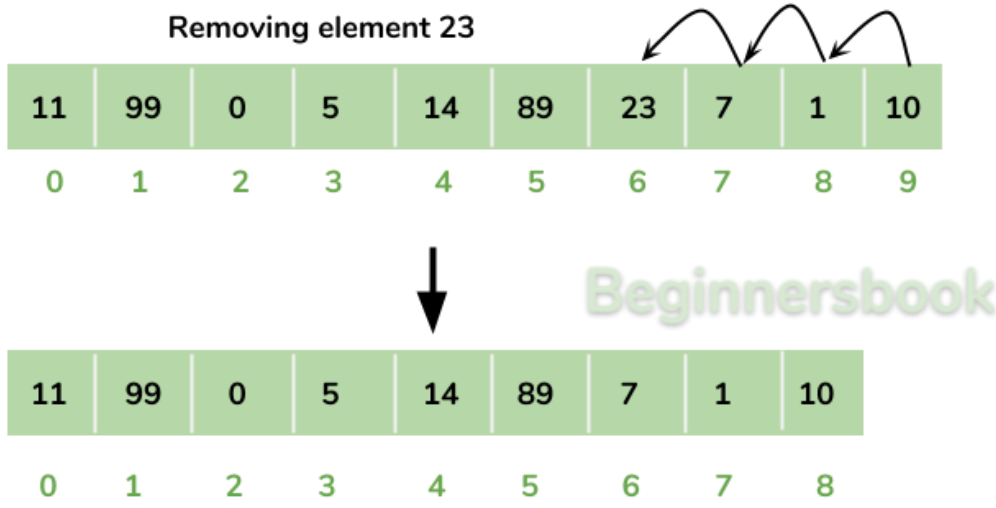
```
List<String> list = new ArrayList<Strng>();

list.add("보라돌이");

String name = list.get(0);
```

ArrayList

빈번한 객체 삭제와 삽입이 일어나는 곳에서는 사용하지 말 것. (LinkedList가 더 좋음)

인덱스 검색이나, 맨 마지막에 추가하는 경우는 ArrayList가 더 좋음.





ArrayList

collection.list.ArrayListExample.java

ArrayList

ExamArray01.java | Queue.java | ArrayListExample.java | Arrays.java

```
3804     * @serial include
3805     */
3806    private static class ArrayList<E> extends AbstractList<E>
3807        implements RandomAccess, java.io.Serializable
3808    {
3809        private static final long serialVersionUID = -2764017481108945198L;
3810        private final E[] a;
3811
3812        ArrayList(E[] array) {
3813            a = Objects.requireNonNull(array);
3814        }
3815
3816        @Override
3817        public int size() { return a.length; }
3820
3821        @Override
3822        public Object[] toArray() { return a.clone(); }
3825
3826        @Override
3827        /unchecked/
3828        public <T> T[] toArray(T[] a) {
3829            int size = size();
3830            if (a.length < size)
3831                return Arrays.copyOf(this.a, size,
```

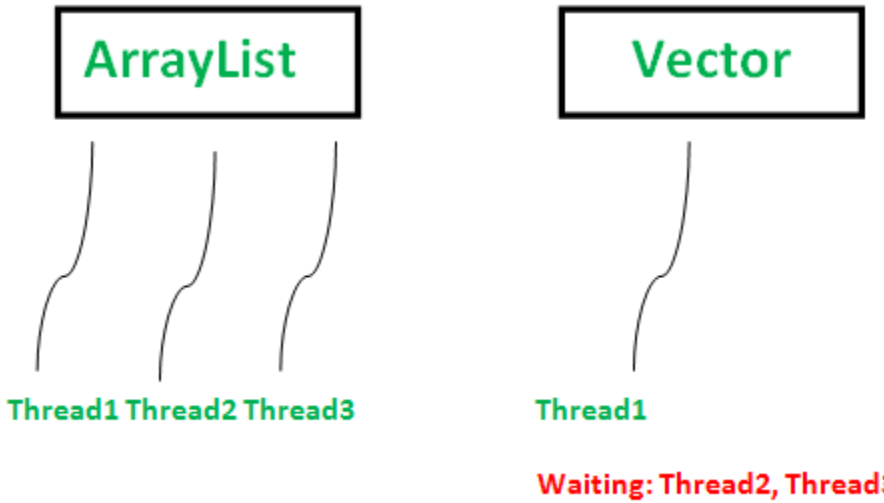
ArrayList.java

```
105     */
106
107    public class ArrayList<E> extends AbstractList<E>
108            implements List<E>, RandomAccess, Cloneable, java.io.Serializable
109    {
110        private static final long serialVersionUID = 8683452581122892189L;
111
112        /**
113         * Default initial capacity.
114         */
115        private static final int DEFAULT_CAPACITY = 10;
116
117        /**
118         * Shared empty array instance used for empty instances.
119         */
120        private static final Object[] EMPTY_ELEMENTDATA = {};
121
122        /**
123         * Shared empty array instance used for default sized empty instances. We
124         * distinguish this from EMPTY_ELEMENTDATA to know how much to inflate when
125         * first element is added.
126         */
127        private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA = {};
```

Vector

ArrayList와 동일한 내부 구조.

동기화된 메소드로 구성되어 있으므로, 멀티 스레드 환경에서 안전하게 작업 가능.

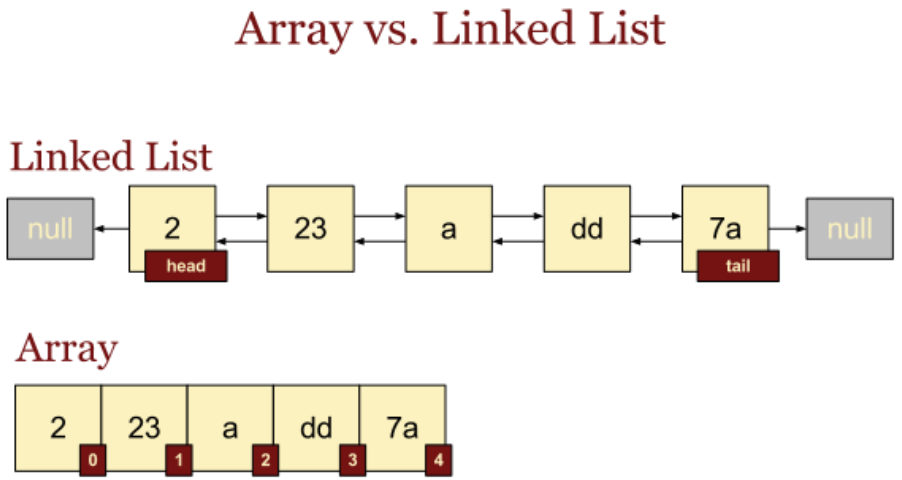


LinkedList

ArrayList와 사용 방법은 동일하지만 내부 구조는 다르다.

인접 참조를 링크해서 체인처럼 관리.

특정 인덱스의 객체를 삽입, 제거 시 앞, 뒤 링크만 변경.



LinkedList

collection.list.LinkedListExample.java

평가

구분	순차적으로 추가/삭제	중간에 추가/삭제	검색
ArrayList	빠르다	느리다	빠르다
LinkedList	느리다	빠르다	느리다

구분	순차적으로 추가/삭제	중간에 추가/삭제	검색
ArrayList	느리다	느리다	중간
Vector	느리다	느리다	중간
LinkedList	빠르다(약 2배)	빠르다(약 10배)	중간

평가

1. 삽입의 경우 Vector는 동기화를 하면서 삽입하므로 ArrayList보다 많은 시간이 걸린다.

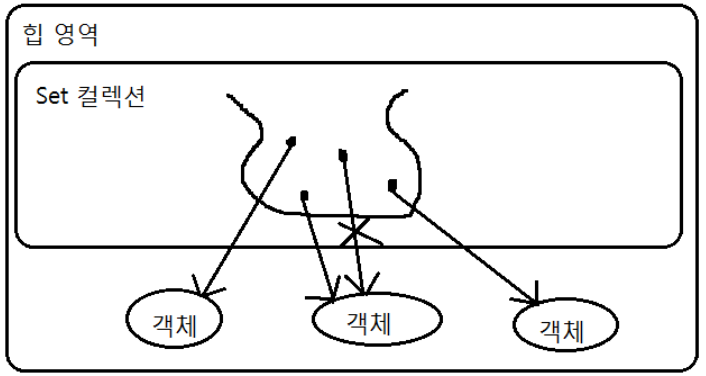
2. 싱글 스레드에서는 ArrayList 사용을 권한다.

3. 중간 데이터 삽입의 경우 Vector, ArrayList는 LinkedList에 비해 시간이 오래 걸린다.

4. 성능이 중요한 경우 일반 배열을 사용해서 구현하는 것을 추천한다.

Set

수학의 집합과 유사.



저장 순서 유지 X. 반복 저장 X. 하나의 null만 가능.

인덱스로 관리하지 않기 때문에 인덱스를 매개 값으로 갖는 메소드 X.

ex. 구슬 주머니

Set

기능	메소드	설명
객체 추가	boolean add(E a)	주어진 객체를 저장.
객체 검색	boolean contaions(Object o)	주어진 객체가 저장되어 있는지에 대한 여부.
	boolean isEmpty()	컬렉션이 비어 있는지 조사.
	int size()	저장되어 있는 전체 객체 수 리턴.
	Iterator<E> iterator()	저장된 객체를 한 번씩 가져오는 반복자 리턴.
객체 삭제	void clear()	저장된 모든 객체 삭제.
	boolean remove(Object o)	주어진 객체를 삭제.

Iterator

https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

컬렉션에 대한 반복자.

1. 반복하는 동안 컬렉션에서 객체를 제거할 수 있다.

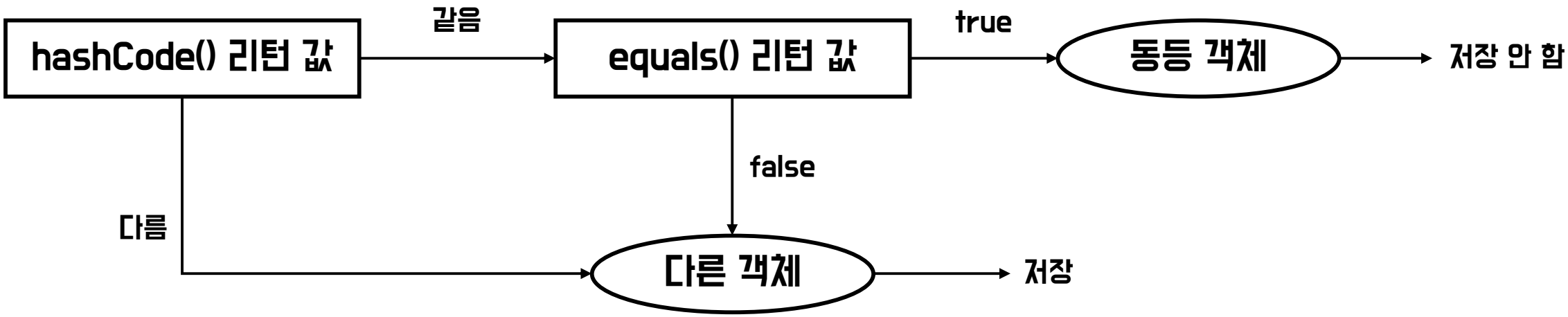
2. Method names have been improved.

리턴 타입	메소드	설명
boolean	hasNext()	가져올 객체가 있으면 true, 없으면 false
E	next()	컬렉션에서 하나의 객체를 가져온다.
void	remove()	next()로 읽어온 객체를 제거한다.

Iterator

iterator.IteratorExample.java

Set



종류	설명
HashSet	데이터 중복 저장 X, 순서를 보장하지 않는다.
TreeSet	데이터 중복 저장 X, 순서를 보장하지 않는다. (오름차순으로 정렬)
LinkedHashSet	데이터 중복 저장 X, 입력된 순서대로 데이터 관리.

Set

collection.set.HashSetExample.java
collection.set.TreeSetExample.java
collection.set.TreeSetExample2.java
collection.set.TreeSetExample3.java
collection.set.LinkedHashSetExample.java
collection.set.Member.java

평가

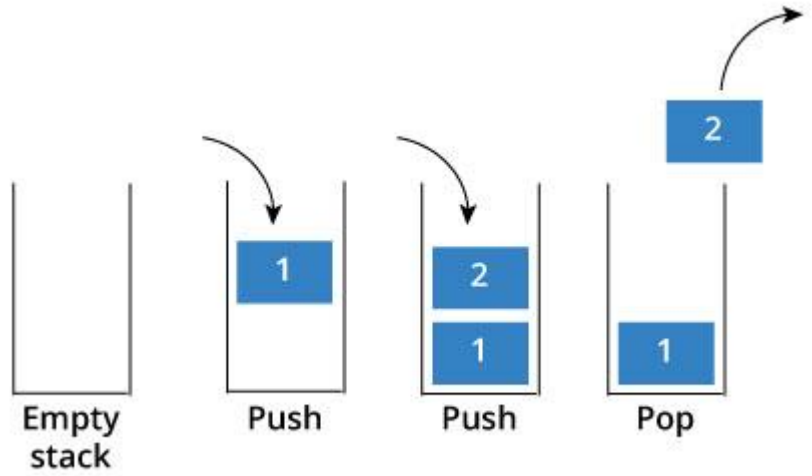
구분	추가	조회
HashSet	빠르다	중간(약 2.5배)
TreeSet	느리다(약 2배)	빠르다
LinkedHashSet	빠르다	느리다(약 3배)

1. 저장 용도로 많이 사용. (자주 꺼내 쓰지 않는 경우)

2. 중복을 제외하고 데이터를 조회하는 경우 사용.

Stack

1. LIFO (Last In First Out)
2. ex. JVM 스택 메모리



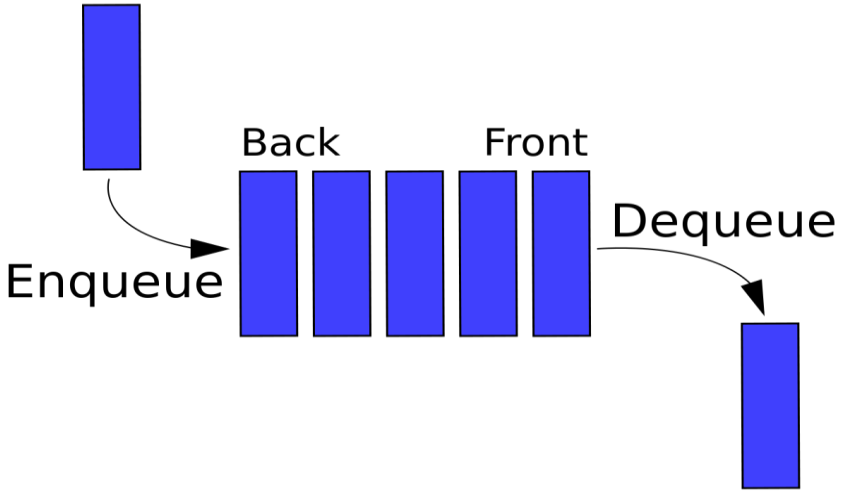
리턴 타입	메소드	설명
E	push(E item)	객체를 스택에 삽입.
	peek()	스택 맨 위의 객체를 가져온다. (삭제 X)
	pop()	스택 맨 위의 객체를 가져온다. (삭제 O)

Stack

collection.list.StackExample.java

Queue

1. FIFO (First In First Out)
2. 대표적으로 LinkedList로 구현.
3. ex. 스레드 풀



리턴 타입	메소드	설명
boolean	offer(E item)	객체를 큐에 삽입.
E	peek()	큐 맨 앞의 객체를 가져온다. (삭제 X)
	poll()	큐 맨 앞의 객체를 가져온다. (삭제 O)

Queue

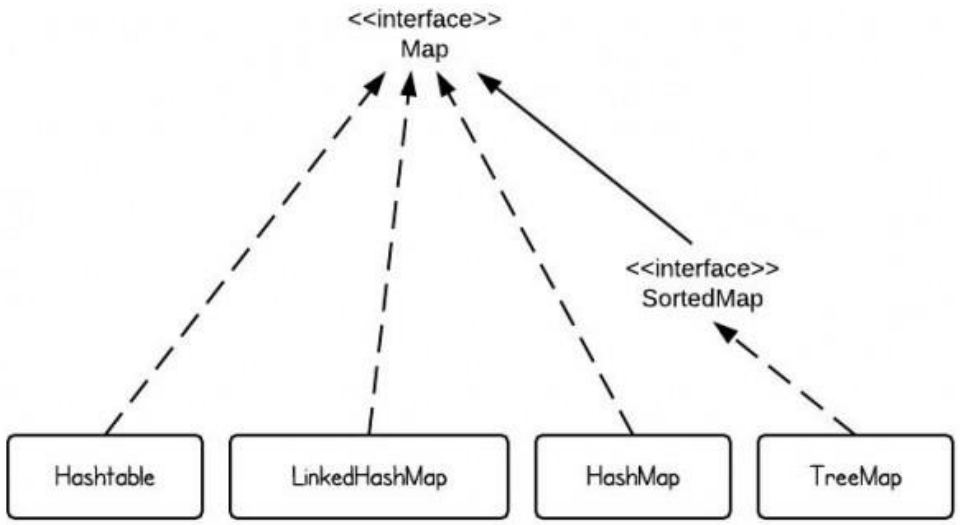
collection.queue.QueueExample.java
collection.queue.PriorityQueueExample.java

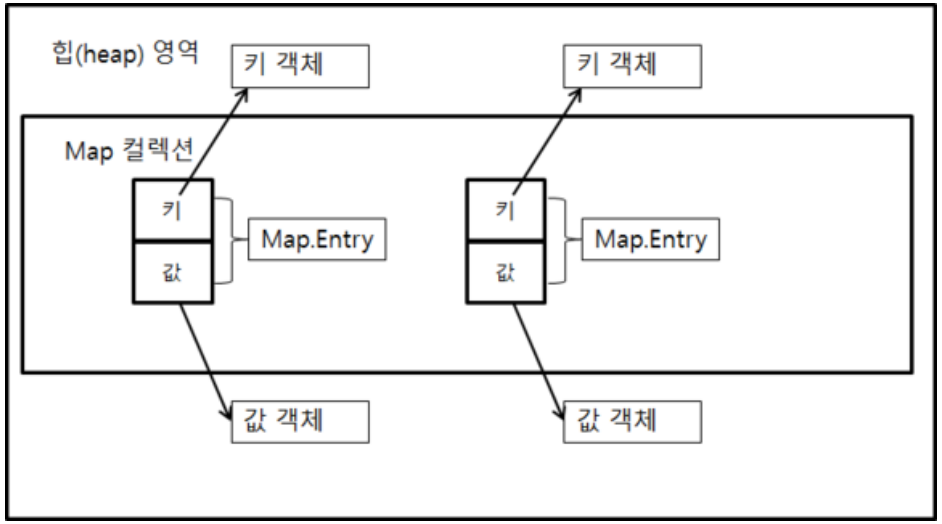
Map

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

key와 value로 구성된 Entry 객체를 저장하는 구조.

Key와 value는 모두 객체이며, key는 중복 저장 불가, value는 중복 저장 허용.





Map

종류	설명
HashMap	hash table을 구현한 것으로 정렬되지 않은 키와 값을 가진다.
Hashtable	HashMap과 달리 동기적. (동기화를 위한 오버헤드 발생)
LinkedHashMap	값의 입력된 순서를 유지한다.
TreeMap	red-black 트리로 구현한 것으로 정렬된 키를 가지고 있다. (TreeSet과 유사)

Map

기능	메소드	설명
객체 추가	V put(K key. V value)	주어진 키와 값을 추가
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 확인
	boolean containsValue(Object value)	주어진 값이 있는지 확인
	Set<Map.Entry<K,V>> entrySet()	키와 쌍으로 구성된 모든 Entry를 Set으로 리턴
	V get(Object key)	주어진 키가 있는 값을 리턴
	boolean isEmpty()	컬렉션이 비어 있는지 확인
	Set<K> keySet()	모든 키를 Set으로 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값을 Collection에 담아서 리턴
객체 삭제	void clear()	모든 Entry 삭제
	V remove(Object key)	주어진 키와 일치하는 Entry 삭제

Map

map.Dog.java
map.HashMapExample.java
map.LinkedHashMapExample.java
map.TreeMapExample.java

평가

1. Java 컬렉션에서 가장 많이 쓰이는 클래스.

2. 각 Map 클래스의 가장 큰 차이점은 데이터를 출력하는 순서.

3. HashMap, LinkedHashMap의 검색 성능은 O(1), TreeMap은 O(log n)

4. 특별히 정렬할 필요가 없다면 TreeMap은 피할 것.

정리

Collection : 컬렉션 인터페이스

List : 배열의 확장판으로 순서가 있는 집합을 처리하기 위한 인터페이스

Set : 중복을 허용하지 않는 집합을 처리하기 위한 인터페이스

Queue : FIFO

Map : 키와 쌍으로 구성된 객체의 집합을 처리하기 위한 인터페이스

정리

ArrayList : 크기를 지정할 필요가 없는 배열. (동기화 X)

Vector : 동기화 처리가 되는 ArrayList

LinkedList : LinkedList로 구현된 List

정리

HashSet : 순서에 상관없이 저장되는 Set

TreeSet : red-black 트리를 이용해 값에 따라 정렬되는 Set

LinkedHashSet : 저장되는 순서에 따라 저장되는 Set

정리

HashMap : null을 지원하는 Hashtable (동기화 X)

Hashtable : 데이터를 해시 테이블에 저장. (동기화 처리 O)

TreeMap : 키 값에 따라 순서가 정해진다. (TreeSet과 유사)

LinkedHashMap : 입력된 값의 순서를 유지 (LinkedHashSet과 유사)

정리

컬렉션들은 각각의 용도가 다르다.

필요한 용도에 따라서 가장 적합한 클래스를 선택하여 사용하는 것이 가장 적절하다.

List - ArrayList

Set - HashSet

Map - HashMap

Queue - LinkedList

http://blog.naver.com/PostView.nhn?blogId=2feelus&logNo=220622485558

Comparable & Comparator

TreeSet의 객체와 TreeKey의 키는 저장과 동시에 오름차순으로 정렬.

정렬을 위해 Comparable을 구현한 객체를 요구.

사용자 정의 클래스를 정렬하려면?

➔ Comparable을 구현하자!

Comparable

```
public class Wallet implements Comparable<T> {
        public String name;
        public int coin;
        ...

        @Override
        public int compareTo(T o) {
                return this.coin - o.coin
        }
}
```

주어진 객체와 같으면 0을 리턴

주어진 객체보다 작으면 음수를 리턴

주어진 객체보다 크면 양수를 리턴

Comparable & Comparator

Comparable - 이 인터페이스를 구현한 객체 스스로에게 부여하는 한 가지 기본 정렬 규칙을 설정하는 목적. (int compareTo(Object o))

Comparator - 이 인터페이스를 구현한 클래스는 정렬 규칙 그 자체를 의미. 기본 정렬 규칙과 다르게 원하는 대로 정렬순서를 지정하고 싶을 때 사용. (int compare(Object o1, Object o2))

➔ 기본적으로 인터페이스가 영향을 미치는 범위와 그에 따른 사용목적이 조금 다르다.

Comparable & Comparator

collections.Wallet.java
collections.ComparactorExample.java

Collections

https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html

static 메소드로만 구성된 utility 클래스.

Collections를 통해 Java Collection Framework의 알고리즘을 제공한다.

주요 메소드

리턴 타입	메소드	설명
void	sort(List list)	List안의 객체를 오름차순으로 정렬.
void	sort(List list, Comparator c)	List안의 객체를 정의된 Comparator에 따라 정렬.
void	reverse(List list)	List안 객체의 순서를 완전히 뒤집기.
void	shuffle(List list)	List안 객체의 순서를 완전히 섞기.
int	binarySearch(List list, T key)	List안 주어진 객체의 위치를 이진 탐색.
int	frequency(List list, Object o)	List안 주어진 객체의 출현도.

Collections

collections.CollectionExample.java
collections.CollectionExample2.java

주의 사항

https://www.programcreek.com/2014/05/top-10-mistakes-java-developers-make/

Arrays.asList()는 Arrays의 private 정적 클래스인 ArrayList를 리턴한다.

➔ java.util.ArrayList와 java.util.Arrays.ArrayList는 다르다!!

```
String[] str = {“보라돌이”, “뚜비 " };

// List<String> list = Arrays.asList(str);

List<String> list = new ArrayList<String>(Arrays.asList(str));
```

주의 사항

일반 배열에 특정 값 확인하기

➔ Set으로 변환 코드를 많이 쓰지만, 할 필요 없음... (비효율적)

```
String[] str = {“보라돌이”, “뚜비 " };
// Set<String> set = new HashSet<String>(Arrays.asList(str));

Arrays.asList(str).contains(“보라돌이”);
// 또는
for(String s : str) {
        if(s.equals(“보라돌이”) { return true; }
}
return false;
```

주의 사항

Loop 안에서 원소 제거하기

➔ List의 사이즈가 줄어들면서, index도 바뀌버린다.

collection.list.ArrayListBadExample.java

주의 사항

알고리즘적으로 Hashtable은 자료구조 이름이지만, Java에서의 이름은 HashMap이다.

➔ Hashtable과 HashMap의 차이는 동기화.

https://www.programcreek.com/2013/03/hashmap-vs-treemap-vs-hashtable-vs-linkedhashmap/

Q&A

감사합니다!