컬렉션(Collection) 프레임워크

기초

의미

- 유사한 객체의 집단을 효율적으로 관리할 수 있도록 컬렉션 프레임워크를 제공
- 컬렉션: 데이터를 한곳에 모아 편리하게 저장 및 관리하는 가변 크기의 객체 컨테이너
- 컬렉션 프레임워크: 객체를 한 곳에 모아 효율적으로 관리하고 편리하게 사용할 수 있도록 제공하는 환경

필요성

- 유사한 객체를 여러개 저장하고 조작해야 할 때가 빈번 * 고정된 크기의 배열의 불편함을 연결리스트로 해결

구조

- 컬렉션 프레임워크: 인터페이스와 클래스로 구성
- 인터페이스는 컬렉션에서 수행할 수 있는 각종 연산을 제네릭 타입으로 정의해 유사한 클래스에 일관성 있게 접근하게 함
- 클래스는 컬렉션 프레임워크 인터페이스를 구현한 클래스
- java.util 패키지에 포함된다.

컬렉션 인터페이스

- List: 객체의 순서가 있고, 원소가 중복될 수 있다.
  (구현 클래스: ArrayList, Stack, Vector, LinkedList)
- Queue: 객체를 입력한 순서대로 저장하며, 원소가 중복될 수 있다.
  (구현 클래스: DelayQueue, PriorityQueue, LinkedList)
- Set: 객체의 순서가 없으며, 동일한 원소를 중복할 수 없다.
  (구현 클래스: HashSet, TreeSet, EnumSet)

주요 메소드

```
boolean add(E e)            // 객체를 맨 끝에 추가한다.
void clear()                // 저장된 모든 객체를 제거한다.
boolean contains(Object o)  // 명시한 객체의 저장 여부를 조사한다.
boolean isEmpty()           // 리스트가 비어 있는지 조사한다.
Iterator<E> iterator()      // Iterator를 반환한다.
boolean remove(Object o)    // 명시한 첫 번째 객체를 제거하고, 제거 여부를 반환한다.
```

```
int size()                  // 저장된 전체 객체의 개수를 반환한다.
T[] toArray(T[] a)          // 리스트를 배열로 반환한다.
```

+) 유용한 디폴트 메소드를 제공한다.

```
default void forEach (Consumer<? super T> action)
default boolean removeIF (Predicate <? super E> filter)
default <T> T[] to Array (IntFunction <T[]> generator)
```

컬렉션의 반복 처리

- ArrayList에 문자열을 저장했다가 꺼내는 다양한 방법

```
String a[] = new String[] {“A”, “B”, “C”, “D”, “E”};
List<String> list = Arrays.asList(a);
```

1) for 구문 사용

```
for(int i=0; i<list.size(); i++)
    System.out.println(list.get(i));
```

2) for-each 구문 사용

```
for (String s: list)
    System.out.println(s);
```

3) Collection인터페이스는iterator() 메소드 사용

- iterator 인터페이스가 제공하는 주요 메소드

```
boolean hasNext()       // 다음 원소의 존재 여부를 반환한다.
E next()                // 다음 원소를 반환한다.
default void remove()   // 마지막에 순회한 컬렉션의 원소를 삭제한다.
```

반복자는 되돌리기 기능이 없다.

List 컬렉션

순서가 있는 객체를 중복 여부와 상관없이 저장하는 리스트 자료구조를 지원한다.
배열과 유사하지만 크기가 가변적이다.
순서가 있어 원소를 저장하거나 읽어올 때 인덱스를 사용한다.

주요 메소드

```
void add(int index, E element)      // 객체를 인덱스 위치에 추가한다.
E get(int index)                    // 인덱스에 있는 객체를 반환한다.
int indexOf(Object o)               // 명시한 객체가 있는 첫 번째 인덱스를 반환한다.
E remove(int index)                 // 인덱스에 있는 객체를 제거한다.
E set(int index, E element)         // 인덱스에 있는 객체와 주어진 객체를 교체한다.
List<E> subList(int from, int to)   // 범위에 해당하는 객체를 리스트로 반환한다.
```

디폴트 메소드

```
default void replaceAll (UnaryOperator<E> operator)
default void sort (Comparaotr<? super E> c)
```

팩토리 메소드

```
static <E> List<E> of (E ... elements)
```

List타입과 배열사이에는 다음 메소드를 사용해 상호 변환

```
public static <T> List <T> asList ( T ... a)    // java.util.Arrays 클래스의 정적 메소드
<T> T[ ] toArray(T [ ] a)                       // java.util.List 클래스의 메소드
```

대표적인 List 구현 클래스: ArrayList, Vector, LinkedList, Stack

- ArrayList 클래스
  List 컬렉션처럼 인덱스로 객체를 관리한다. List 컬렉션과 차이점으로는 크기를 동적으로 늘릴 수 있다.

- Vector 클래스
  Vector클래스는 ArrayList와 동일한 기능을 제공하지만 ArrayList와 달리 동기화 된 메서드로 구현해 스레드에 안전하지만 성능이 떨어질 수 있다.

- Stack 클래스
  후입선출 방식으로 객체를 관리하며, Vector의 자식클래스이다. 대부분의 인덱스가

0부터 시작하지만 stack 클래스는 1부터 시작한다.

```
boolean empty()         // 스텍이 비어있는지 조사한다.
E peek()                // 스텍의 최상위 원소를 제거하지 않고 엿본다.
E pop()                 // 스텍의 최상위 원소를 반환하며, 스택에서 제거한다.
E push(E item)          // 스텍의 최상위에 원소를 추가한다.
int search(Object o)    // 주어진 원소의 인덱스 값(1부터 시작)을 반환한다.
```

LinkedList 클래스와 ArrayList 클래스 차이

```
구분                    / ArrayList 클래스  / LinkedList 클래스
구현                    / 가변크기 배열     / 이중 연결 리스트
초기 용량               / 10              / 0
get()연산           / 빠름            / 느림
add(), remove() 연산 / 느림           / 빠름
메모리 부담          / 적음            / 많음
Iterator            / 순방향             / 순방향, 역방향
```

Queue 컬렉션

선입선출 방식을 지원하고 입구와 출구를 후단(tail)과 전단(head)라고 하며, tail에서 원소를 추가하고 head에서 원소를 제거 한다.
중간에 원소를 추가하거나 제거할 수 없다.

추가된 메소드

삽입

```
boolean add(E e)    //예외를 던짐
boolean offer(E e)  // null 또는 false를 반환
```

삭제

```
E remove()      //예외를 던짐
E poll()        // null 또는 false를 반환
```

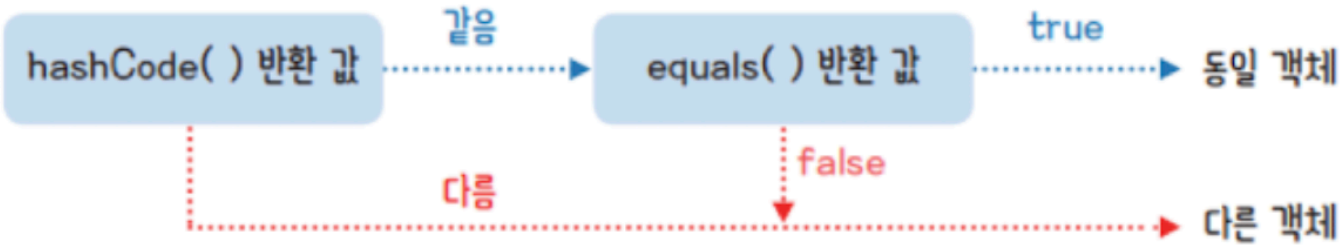
검색

```
E element()     //예외를 던짐
E peek()        // null 또는 false를 반환
```

Set 컬렉션

순서가 없으며, 중복되지 않는 객체를 저장하는 자료구조를 지원한다.
추가된 메소드는 없지만 중복 원소를 배제한다.
인덱스가 없어 저장이 되는 순서는 무시한다.

Set 컬렉션의 같은 객체



Set 팩토리 메소드

```
static <E> Set<E> of (E ... elements)
```

대표적인 Set: hashSet, TreeSet

Map 인터페이스

특징

- 키와 값, 이렇게 쌍으로 구성된 객체를 저장하는 자료구조이다.
- 맵이 사용하는 키와 값도 모두 객체이다.
- 키는 중복되지 않고 하나의 값에만 매핑되어 있으므로 키가 있으면 대응하는 값을 얻을 수 있다.
- Map 객체에 같은 키로 중복 저장되지 않도록 하려면 Set 객체처럼 키로 사용할 클래스에 대한 hashCode() 와 equals() 메소드를 오버로딩 해야 한다.

구현 클래스 : HashMap, Hashtable, TreeMap, Properties

Map 인터페이스가 제공하는 메소드

```
void clear()                            // 모든 매핑을 삭제한다.
boolean containsKey(Object Key)         // 주어진 키의 존재 여부를 반환한다.
boolean containsValue(Object value)     // 주어진 값의 존재 여부를 반환한다.
Set<Map.Entry<K, V>> entrySet()         // 모든 매핑을 Set 타입으로 반환한다.
V get(Object key)                       // 주어진 키에 해당하는 값을 반환한다.
boolean isEmpty()                       // 컬렉션이 비어 있는지 여부를 반환한다.
Set<K> keySet()                         // 모든 키를 Set 타입으로 반환한다.
V put(K key, V value)                   // 주어진 키-값을 저장하고 값을 반환한다.
V remove(Object key)                    // 키와 일치하는 원소를 삭제하고 값을 반환한다.
int size()                              // 컬렉션의 크기를 반환한다.
Collection<V> values()                  // 모든 값을 Collection 타입으로 반환한다.
```

Map.Entry<K, V> 인터페이스가 제공하는 메소드

```
K getKey()          // 원소에 해당하는 키를 반환한다.
V getValue()        // 원소에 해당하는 값을 반환한다.
V setValue()        // 원소의 값을 교체한다.
```

팩토리 메소드

```
static <K, V> Map<K, V> of (K k1, V v1)
```

디폴트 메소드

```
default void forEach(BiConsumer action)
default void replaceAll(BiFunction function)
```

HashMap과 Hashtable

Hashtable은 HashMap과 달리 동기화 된 메소드로 구현되어 스레드에 안전하다.
HashMap에서는 키와 값으로 null을 사용할 수 있지만 Hashtable에서는 사용할 수 없다.

컬렉션 클래스

특징

- 컬렉션을 다루는 다양한 메소드를 제공하는 java.util 패키지의 클래스
- 컬렉션 원소 정렬, 섞기, 탐색 등 문제를 쉽게 해결할 수 있다.

정렬

```
static void reverse(List list)
static void sort(List list)
static void sort(List list, Comparator c)
static Comparator reverseOrder()
static Comparator reverseOrder(Comparator c)
```

돌리기 및 섞기

```
static void rotate(List<?> list, int distance)
static void shuffle(List<?> list)
static void shuffle(List<?> list, Random r)
```

탐색하기

```
static <T> int binarySearch(List<T> list, T key)
static <T> int binarySearch(List<T> list, T key, Comparator<T> c)
```

기타 메소드

```
addAll()        // 명시된 원소들을 컬렉션에 삽입한다.
copy()          // 리스트를 다른 리스트로 복사한다.
disjoint()      // 2개의 컬렉션에서 공통된 원소가 있는지 조사한다.
fill()          // 리스트의 모든 원소를 특정 값으로 덮어쓴다.
frequency()     // 컬렉션에 주어진 원소의 빈도 수를 반환한다.
max()           // 리스트에서 최댓값을 반환한다.
min()           // 리스트에서 최솟값을 반환한다.
nCopies()       // 매개변수 값으로 주어진 객체를 주어진 횟수만큼 복사해 List 객체를 반환한
다.
reverse()       // 리스트의 원소들을 역순으로 정렬한다.
swap()          // 리스트에서 주어진 위치에 있는 두 원소를 교체한다.
```