CF

Collection Framework

다수의 데이터를 다루기 위한 표준화된 프로그래밍 방식이다.
핵심 인터페이스로는 List와 Set, Map이 있고 List와 Set의 공통적인 부분을 모아놓은 인터페이스인 Collection이 있다.

Collection

List와 Set의 조상으로 다음의 메서드들이 정의되어 있다.

메서드	설명
boolean add(Object o) boolean add(Collection c)	Collection에 지정된 객체를 추가하는 메서드
void clear()	Collection에 모든 객체를 삭제하는 메서드
boolean contains(Object o) boolean containAll(Collection c)	Collection에 지정된 객체가 포함되어 있는지 확인하는 메서드
boolean equals(Object o)	동일한 Collection인지 비교하는 메서드
boolean isEmpty()	Collection이 비어있는지 확인하는 메서드
Iterator iterator()	Iterator를 반환하는 메서드
boolean remove(Object o) boolean removeAll(Collection c)	지정된 객체를 삭제하는 메서드
boolean retainAll(Collection c)	지정된 Collection에 포함된 객체만 남기고 나머지는 삭제하는 메서드
int size()	Collection에 저장된 객체의 수를 반환하는 메서드
Object[] toArray()	Collection에 저장된 객체를 객체배열로 반환하는 메서드

List

순서가 있고 데이터의 중복을 허용하는 인터페이스이다.
Collection의 메서드와 다음의 메서드들이 정의되어 있다.

메서드	설명
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 위치에 객체를 추가하는 메서드
Object get(int index)	지정된 위치에 있는 객체를 반환하는 메서드
int indexOf(Object o)	지정된 객체의 위치를 첫 번째 요소부터 찾아서 반환하는 메서드
int lastIndexOf(Object o)	지정된 객체의 위치를 마지막 요소부터 찾아서 반환하는 메서드

메서드	설명
Object remove(int index)	지정된 위치에 있는 객체를 삭제하고 반환하는 메서드
Object set(int index, Object element)	지정된 위치에 객체를 저장하는 메서드
void sort(Comparator c)	지정된 비교자로 List를 정렬하는 메서드
List subList(int fromIndex, int toIndex)	지정된 범위에 있는 객체를 반환하는 메서드

ArrayList

Object 배열을 이용해서 데이터를 순차적으로 저장한다.
기존의 Vector를 개선한 것이며, Vector는 자체적으로 동기화 처리가 되어 있으나, ArrayList는 그렇지 않다.
배열에 순서대로 저장되며, 배열에 더 이상 저장할 공간이 없으면 보다 큰 새로운 배열을 생성해서 기존의 배열에 저장된 내용을 새로운 배열로 복사한 다음에 저장한다.
다음의 메서드들이 정의되어 있다.

메서드	설명
ArrayList()	크기가 0인 ArrayList를 생성하는 생성자
ArrayList(Collection c)	주어진 Collection이 저장된 ArrayList를 생성하는 생성자
ArrayList(int initalCapacity)	지정된 초기용량을 갖는 ArrayList를 생성하는 생성자
ArrayList(int initalCapacity, int capacityIncrement)	지정된 초기용량과 용량의 증분을 갖는 ArrayList를 생성하는 생성자
boolean add(Object o) boolean addAll(Collection c)	ArrayList의 마지막에 객체를 추가하는 메서드
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 위치에 객체를 저장하는 메서드
void clear()	ArrayList를 비우는 메서드
Object clone()	ArrayList를 복제하는 메서드
boolean contains(Object o)	지정된 객체가 ArrayList에 포함되어 있는지 확인하는 메서드
void ensureCapacity(int minCapacity)	ArrayList의 용량이 최소한 minCapacity가 되도록 하는 메서드
void trimToSize()	ArrayList의 빈 공간을 없애는 메서드

ArrayList는 배열을 이용해 데이터를 저장하기 때문에 크기를 변경할 수 없으므로 새로운 배열을 생성해서 데이터를 복사하는 작업과
비순차적인 데이터의 추가 또는 삭제에서는 데이터를 복사하고 이동해야 해서 시간이 많이 걸린다.

LinkedList

linked list는 ArrayList의 단점을 보완하기 위해서 나온 자료구조이다.
배열과 달리 불연속적으로 존재하는 데이터를 연결해서 저장한다.
linked list는 데이터의 추가, 삭제할 때 처리속도가 매우 빠르지만 이동방향이 단반향이여서 이전 요소에 대한

접근은 어렵다.
이 점을 보완한 것이 doubly linked list이다. 기존 linked List에서 이전 요소에 대한 참조가 가능하도록 하였다.
여기서 접근성을 보다 향상시킨 것이 doubly circular linked list인데 doubly linked list에서 첫 번째 요소와 마지막 요소를 연결시켜준 것이다.
LinkedList는 이 doubly circular linked list를 구현했는데, 접근성을 높이기 위해서이다.
다음의 메서드들이 정의되어 있다.

메서드	설명
LinkedList()	LinkedList를 생성하는 생성자
LinkedList(Collection c)	주어진 Collection이 저장된 LinkedList를 생성하는 생성자
boolean add(Object o) boolean addAll(Collection c) boolean offer(Object o)	LinkedList의 마지막에 객체를 추가하는 메서드
void add(int index, Object element) boolean addAll(int index, Collection c)	지정한 위치에 객체를 저장하는 메서드
void clear()	LinkedList를 비우는 메서드
boolean contains(Object o) boolean containsAll(Collection c)	지정된 객체가 LinkedList에 포함되어 있는지 확인하는 메서드
Object get(int index)	지정된 위치의 객체를 반환하는 메서드
int indexOf(Object o)	지정된 객체의 위치를 반환하는 메서드
Object element() Object peek() Object getFirst() Object peekFirst()	LinkedList의 첫 번째 요소를 반환하는 메서드
Object getLast() Object peekLast()	LinkedList의 마지막 요소를 반환하는 메서드
Object poll() Object pollFirst()	LinkedList의 첫 번째 요소를 제거하고 반환하는 메서드
Object pollLast()	LinkedList의 마지막 요소를 제거하고 반환하는 메서드
Object remove() Object pop() Object removeFirst()	LinkedList의 첫 번째 요소를 제거하는 메서드
Object removeLast()	LinkedList의 마지막 요소를 제거하는 메서드
Object addFirst(Object o) boolean offerFirst(Object o)	LinkedList의 맨 앞에 객체를 추가하는 메서드
Object addLast(Object o) boolean offerLast(Object o)	LinkedList의 마지막에 객체를 추가하는 메서드

메서드	설명
Iterator descendingIterator()	역순으로 조회하기 위한 DescendingIterator를 반환하는 메서드
boolean removeFirstOccurrence(Object o)	LinkedList에서 첫 번째로 일치하는 객체를 제거하는 메서드
boolean removeLastOccurrence(Object o)	LinkedList에서 마지막으로 일치하는 객체를 제거하는 메서드

ArrayList와 LinkedList 비교

Collection	데이터에 접근시간	추가 / 삭제	비 고
ArrayList	빠르다	느리다	순차적인 추가 및 삭제는 더 빠르다. 비효율적인 메모리를 사용한다.
LinkedList	느리다	빠르다	데이터가 많을수록 접근성이 떨어진다.

따라서 데이터의 개수가 변하지 않는 경우라면 ArrayList를 사용하고, 데이터의 개수의 변경이 잦다면 LinkedList를 사용하자.
아니면 데이터를 저장할 때는 ArrayList를 사용한 다음, 작업할 때는 LinkedList로 데이터를 옮겨서 작업하자.

Stack

LIFO구조로 마지막에 저장된 것을 제일 먼저 꺼내게 되는 클래스이다.
ArrayList로 구현하며, 수식 계산이나 수식 괄호 검사, undo/redo, 뒤로 가기/앞으로 가기를 구현할 때 사용한다.
다음의 메서드들이 정의되어 있다.

메서드	설명
boolean empty()	Stack이 비어있는 지 확인하는 메서드
Object peek()	Stack의 맨 위에 저장된 객체를 반환하는 메서드(pop()과 달리 Stack에서 꺼내지는 않는다.)
Object pop()	Stack의 맨 위에 저장된 객체를 꺼내서 반환하는 메서드
Object push(Object item)	Stack에 객체를 저장하는 메서드
int search(Object o)	Stack에서 지정된 객체를 찾아 그 위치를 반환하는 메서드(배열과 달리 위치는 1부터 시작하고 못찾으면 -1을 반환한다.)

Queue

FIFO구조로 제일 먼저 저장한 것을 제일 먼저 꺼내게 되는 인터페이스이다.
LinkedList로 구현하며, 최근 사용 문서나 인쇄 작업 대기목록, 버퍼를 구현할 때 사용한다.
다음의 메서드들이 정의되어 있다.

메서드	설명
boolean add(Object o)	지정된 객체를 Queue에 저장하는 메서드
Object remove()	Queue에서 객체를 꺼내 반환하는 메서드
Object element()	삭제없이 요소를 읽어오는 메서드
boolean offer(Object o)	Queue에 객체를 저장하는 메서드
Object poll()	Queue에서 객체를 꺼내서 반환하는 메서드
Object peek()	삭제없이 요소를 읽어오는 메서드(Queue가 비어있다면 null을 반환)

add()와 remove(), element()는 예외가 발생할 수 있으므로 try-catch문을 이용하고,
poll과 peek는 null을 반환하기 때문에 if(obj == null)로 확인한다.

Queue의 종류로는 Deque, PriorityQueue, BlockingQueue가 있다.
Deque는 stack과 Queue의 결합으로 양 끝에서 저장(offer)과 삭제(poll)가 가능하다.
구현한 클래스로는 ArrayDeque와 LinkedList가 있다.

PriorityQueue는 우선 순위가 높은 것부터 꺼내는 Queue이며 null은 저장할 수 없다.
저장공간으로 배열을 사용하며, 각 요소를 힙이라는 자료구조의 형태로 저장한다.
힙은 이진 트리의 한 종륲 가장 큰 값이나 가장 작은 값을 빠르게 찾을 수 있다는 특징이 있다.

BlockingQueue는 멀티쓰레드에서 사용하며, 비어있을 때 꺼내기와 가득 차 있을 때 넣기를 지정된 시간동안 지연시키는 Queue이다.

Set

순서가 없고 데이터의 중복을 허용하지 않는 인터페이스이다.
순서가 없어서 정렬이 불가능하다.
Collection의 메서드와 동일하다.

HashSet

Set 인터페이스를 구현한 대표적인 컬렉션이다.
순서를 유지하려면 LinkedHashSet 클래스를 사용하면 된다.

HashSet의 add()는 사용할 때 equals()와 hashCode()를 호출한다.
따라서, equals()와 hashCode()가 오버라이딩이 되어 있어야 한다.

hashCode()의 오버라이딩 조건은 다음과 같다.

1. 동일 객체에 대해 hashCode()를 여러 번 호출해도 동일한 값을 반환해야 한다.
2. equals()로 비교해서 true로 얻은 두 객체의 hashCode()값은 일치해야 한다.
3. equals()로 비교한 결과가 false인 두 객체의 hashCode()값이 같을 수도 있지만 서로 다른 값을 반환하도록 작성한다.(성능 향상)

오버라이딩을 할 때 java.util.Objects 클래스의 hash() 괄호 안에 클래스의 인스턴스 변수들을 넣으면 된다.

TreeSet

이진 검색 트리라는 자료구조의 형태로 데이터를 저장하는 컬렉션 클래스이다.
범위 검색과 정렬에 유리한 컬렉션 클래스이고, HashSet보다 데이터 추가, 삭제에 시간이 더 걸린다.
이진 검색 트리는 부모보다 작은 값을 왼쪽에, 큰 값은 오른쪽에 저장을 한다.

TreeSet에 추가되는 객체는 비교 기준을 가져야하거나, TreeSet이 비교 기준을 가져야한다.
그렇지 않으면, 두 번째 객체를 저장할 때 예외가 발생한다.

TreeSet의 메서드는 다음과 같다.

메서드	설명
TreeSet(Collection c)	주어진 컬렉션을 저장하는 TreeSet을 생성하는 생성자
TreeSet(Comparator comp)	주어진 정렬기준으로 정렬하는 TreeSet을 생성하는 생성자
Object first()	정렬된 순서에서 첫 번째 객체를 반환하는 메서드
Object last()	정렬된 순서에서 마지막 객체를 반환하는 메서드
Object pollFirst()	TreeSet의 첫 번째 요소를 반환하는 메서드
Object pollLast()	TreeSet의 마지막 요소를 반환하는 메서드
Object ceiling(Object o)	지정된 객체와 같은 객체를 반환하는 메서드(없으면 큰 값을 가진 객체 중 제일 가까운 값의 객체를 반환한다.)
Object floor(Object o)	지정된 객체와 같은 객체를 반환하는 메서드(없으면 작은 값을 가진 객체 중 제일 가까운 값의 객체를 반환한다.)
SortedSet subSet(Object fromElement, Object toElement)	범위 검색의 결과를 반환하는 메서드(toElement는 범위에 포함되지 않는다.)
SortedSet headSet(Object toElement)	지정된 객체보다 작은 값의 객체들을 반환하는 메서드
SortedSet tailSet(Object fromElement)	지정된 객체보다 큰 값의 객체들을 반환하는 메서드

Map

key와 value의 쌍으로 이루어져 있으며, 순서는 없고 키는 중복을 허용하지 않고, 값은 중복을 허용하는 인터페이스이다.
다음의 메서드들이 정의되어 있다.

메서드	설명
void clear()	Map의 모든 객체를 삭제하는 메서드
boolean containsKey(Object key)	지정된 key와 일치하는 Map의 key가 있는지 확인하는 메서드
boolean containsValue(Object value)	지정된 value와 일치하는 Map의 value가 있는지 확인하는 메서드

메서드	설명
Set entrySet()	Map에 저장되어 있는 key와 value의 쌍을 Map.Entry타입의 객체로 저장한 Set으로 반환하는 메서드
boolean equals(Object o)	동일한 Map인지 비교하는 메서드
Object get(object key)	지정된 key에 대응하는 value를 반환하는 메서드
boolean isEmpty()	Map이 비어있는지 확인하는 메서드
Set keySet()	Map에 저장된 key를 Set으로 반환하는 메서드
Object put(Object key, Object value) void putAll(Map t)	Map에 key와 value의 쌍을 추가하는 메서드
Object remove(Object key)	지정한 key에 일치하는 key와 value의 쌍을 삭제하는 메서드
int size()	Map에 저장된 key와 value의 쌍의 개수를 반환하는 메서드
Collection values()	Map에 저장된 value를 반환하는 메서드

Map.Entry 인터페이스는 Map 인터페이스의 내부 인터페이스로 Map 인터페이스를 구현하는 클래스에서는 Map.Entry 인터페이스도 함께 구현해야 한다.

HashMap

해싱 기법으로 데이터를 저장하는 클래스이며 데이터가 많아도 검색이 빠르다.

해싱 기법

해시 함수를 이용해서 해시 테이블에서 데이터를 저장 및 검색하는 기법이다.
해시 테이블은 배열과 링크드리스트가 조합된 형태인데, 배열의 장점인 접근성과 링크드리스트의 장점인 변경 유리를 채용한 것이다.

해시 테이블에 저장된 데이터를 가져오는 과정은 다음과 같다.

1. 키로 해시 함수를 호출해서 해시 코드(배열의 index)를 얻는다.
2. 해시 코드에 대응하는 링크드리스트를 배열에서 찾는다.
3. 링크드리스트에서 키와 일치하는 데이터를 찾는다.

equals()는 hashCode()의 반환값이 같아야 같은 객체로 인식하기 때문에 equals()를 오버라이딩해야 된다면 hashCode()도 같이 오버라이딩해줘야 한다.

HashMap의 실제 소스는 다음과 같다.

```
public class HashMap extends AbstractMap implements Map, Cloneable, Serializable {
    transient Entry[] table;
    ... // 일부 생략
    static class Entry implements Map.Entry {
        final Object key;
        Object value;
```

```
        ... // 일부 생략
    }
}
```

Entry라는 내부 클래스를 정의하고, Entry타입의 배열을 선언하고 있는 이유는 키와 값이 서로 관련된 값이므로 하나의 배열로 다루는 것이 데이터의 무결성적인 측면에서 바람직하기 때문이다.

HashMap의 메서드는 다음과 같다.

메서드	설명
HashMap(int initialCapacity)	지정된 값을 초기 용량으로 갖는 HashMap 객체를 생성하는 생성자
HashMap(Map m)	지정된 Map의 모든 요소를 포함하는 HashMap 객체를 생성하는 생성자
Object put(Object key, Object value)	지정된 키와 값을 저장하는 메서드
Object remove(Object key)	지정된 키로 저장된 값을 제거하는 메서드
Object replace(Object key, Object value)	지정된 키의 값을 지정된 값으로 변경하는 메서드
boolean containsKey(Object key)	HashMap에 키가 포함되어 있는지 확인하는 메서드
boolean containsValue(Object value)	HashMap에 값이 포함되어 있는지 확인하는 메서드
void putAll(Map m)	Map에 저장된 모든 요소를 HashMap에 저장하는 메서드

TreeMap

이진 검색 트리의 형태로 키와 값의 쌍으로 이루어진 데이터를 저장하는 클래스이다.
대부분의 경우에서 HashMap이 TreeMap보다 더 뛰어나므로 HashMap을 사용하는 것이 좋지만, 범위검색이나 정렬이 필요한 경우에는 TreeMap을 사용한다.

Properties

HashMap의 구 버전인 Hashtable을 상속받아 구현한 것으로 키와 값을 String으로 저장하는 클래스이다.
주로 애플리케이션의 환경설정과 관련된 속성을 저장하는데 사용되며 데이터를 파일로부터 읽고 쓰는 편리한 기능을 제공한다.

Properties의 메서드는 다음과 같다.

메서드	설명
Properties(ProPerties defaults)	지정된 Properties에 저장된 목록을 가진 Properties 객체를 생성하는 생성자
String getProperty(String key)	지정된 키의 값을 반환하는 메서드
void list(PrintStream out)	지정된 PrintStream에 저장된 목록을 출력하는 메서드
void load(InputStream inStream)	지정된 InputStream으로부터 목록을 읽어서 저장하는 메서드

메서드	설명
void store(OutputStream out, String comments)	저장된 목록을 지정된 OutputStream에 출력하는 메서드(comments는 목록에 대한 주석으로 저장된다.)
Set stringPropertyNames()	Propertyies에 저장되어 있는 모든 키를 Set에 담아서 반환하는 메서드

Iterator, ListIterator, Enumeration

Collection에 저장된 요소를 접근하는데 사용되는 인터페이스이다.

Iterator

컬렉션에 저장된 요소들을 읽어오는 방법을 표준화한 것이다.
표준화한 이유는 데이터를 저장하는 구조가 다르기 때문이다.
1회용이므로 사용하면 다시 호출을 해야한다.
Iterator의 메서드는 다음과 같다.

메서드	설명
boolean hasNext()	읽어 올 요소가 남아있는지 확인하는 메서드
Object next()	다음 요소를 읽어 오는 메서드
void remove()	next()로 읽어 온 요소를 삭제하는 메서드(반드시 구현하지 않아도 된다.)

Map 인터페이스를 구현한 클래스는 iterator()를 직접 호출할 수 없고 keySet()이나 entrySet()를 통해 Set의 형태로 얻어 온 후에 호출할 수 있다.

Enmeration

Enumeration는 iterator의 구 버전이므로 가능하면 사용하지 않는다.

ListIterator

ListIterator는 iterator를 상속받아서 기능을 추가한 것으로, iterator는 단방향만 이동할 수 있지만 ListIterator는 양방향으로 이동이 가능하다.
하지만 ListIterator는 List 인터페이스를 구현한 Collection에서만 사용할 수 있다.

ListIterator의 메서드는 다음과 같다.

메서드	설명
boolean hasPrevious()	읽어 올 이전 요소가 남아있는지 확인하는 메서드
Object previous()	이전 요소를 읽어 오는 메서드
int nextIndex()	다음 요소의 index를 반환하는 메서드
int previousIndex()	이전 요소의 index를 반환하는 메서드

Arrays

배열을 다루는데 유용한 메서드가 정의되어 있는 클래스이다.

다음의 메서드들이 정의되어 있다.

메서드	설명
String toString(arr)	배열을 출력하는 메서드
String deepToString(arr)	다차원 배열을 출력하는 메서드
Object[] copyOf(arr)	배열을 복사하는 메서드
Object[] copyOfRange(arr, int from, int to)	배열의 일부를 복사하는 메서드
fill()	배열의 모든 요소를 지정된 값으로 채우는 메서드
boolean equals(arr1, arr2)	배열의 모든 요소가 같은지 확인하는 메서드
boolean deepEquals(arr1, arr2)	다차원 배열의 모든 요소가 같은지 확인하는 메서드
Object[] sort()	배열을 정렬하는 메서드
int binarySearch(arr, Object)	배열에서 지정된 값이 저장된 index를 반환하는 메서드(반드시 배열이 정렬되어 있어야함)
List asList(Object... a)	배열을 List에 담아서 반환하는 메서드

Comparator와 Comparable

객체를 정렬하는데 필요한 메서드를 정의한 인터페이스이다.
정렬을 하려면 대상과 기준이 필요한데 기준을 제공하는 인터페이스이다.

Comparable

java.lang 패키지에 속하며 기본 정렬기준을 구현하는데 사용한다.

실제 소스는 다음과 같다.

```
public interface Comparable {
    public int compareTo(Object o);
}
```

Comparator

java.util 패키지에 속하며 기본 정렬기준 외에 다른 기준으로 정렬하고자할 때 사용한다.

실제 소스는 다음과 같다.

```
public interface Comparator {
    int compare(Object o1, Object o2);
```

```
    boolean equals(Object obj);
}
```

compare()와 compareTo()는 두 객체의 비교결과를 반환하도록 작성되었고, 같으면 0, 오른쪽이 크면 음수(-), 작으면 양수(+)를 반환한다.

Collections

컬렉션과 관련된 static 메서드를 제공한다.

1. 컬렉션 채우기, 복사, 정렬, 검색 - fill(), copy(), sort(), binarySearch() 등
2. 컬렉션 동기화 - synchronizedXXX()
3. 변경불가 컬렉션 만들기 - unmodifiableXXX()
4. 싱글톤 컬렉션 만들기 - singletonXXX()
5. 한 종류의 객체만 저장하는 컬렉션 만들기 - checkedXXX()

상황별 컬렉션 클래스

컬렉션	특 징
ArrayList	배열기반이며 데이터의 추가와 삭제에 불리하고, 순차적인 추가 및 삭제는 제일 빠르다. 임의의 요소에 대한 접근성이 뛰어나다.
LinkedList	연결기반이며 데이터의 추가와 삭제에 유리하고, 임의의 요소에 대한 접근성이 좋지 않다.
HashMap	배열과 연결이 결합된 형태이며 추가, 삭제, 검색, 접근성이 모두 뛰어나다. 검색에는 최고성능을 보인다.
TreeMap	연결기반이며 정렬과 범위 검색에 적합하다.
Stack	Vector를 상속받아서 구현했다.(LIFO)
Queue	LinkedList가 Queue 인터페이스를 구현했다.(FIFO)
Properties	Hashtable을 상속받아 구현했으며 키와 값을 문자열로 저장한다.
HashSet	HashMap을 이용해서 구현했다.
TreeSet	TreeMap을 이용해서 구현했다.
LinkedHashMap LinkedHashSet	HashMap과 HashSet에 저장 순서 유지기능을 추가하였다.