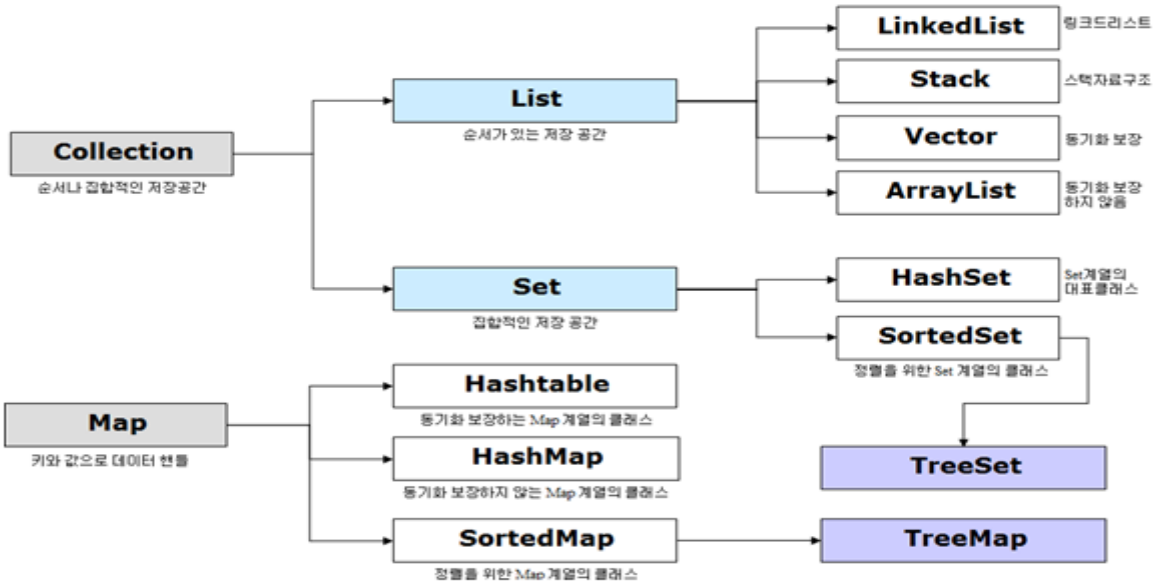
16.Java Collection

자료구조(collection)이란 쉽게 말해 다수의 데이터, 데이터의 그룹을 의미한다. 데이터 그룹을 다루는 방법 중 하나인 배열의 단점을 보완하고, 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법을 제공하는 클래스들의 집합 API이다 . 크게 list계열, set계열, map계열로 구분할 수 있는데 그 중 자주 사용되는 API를 정리해 보았다.



기본적으로 collection API는 배열과 다르게 하나의 컬렉션에 서로 다른 데이터형이 저장 가능하다며 참조 데이터 형만 저장가능하다. 기본 데이터형은 Wrapper클래스를 이용하거나, 오토 박싱(JVM이 자동으로 형변환 해주는 것)으로 저장 가능하다.

1) List계열

자료구조 중 가장 많이, 자주 사용되는 자료 구조형이며 대표적으로 ArrayList, LinkedList, Vector등이 있다. 다음과 같은 특징이 있다.

1. 동일한 데이터의 중복을 허용하며 크기를 고정하지 않아도 되며 그 크기는 유동적이다.
2. 데이터 저장 순서가 유지된다.
3. 힙(heap) 영역 내에서 List는 객체를 일렬로 늘어놓은 구조를 하고 있다.
4. 객체를 인덱스로 관리하기 때문에 객체를 저장하면 자동으로 인덱스가 부여되고 인덱스로 객체를 검색, 삭제할 수 있다. 이때 List 컬렉션은 객체 자체를 저장하여 인덱스를 부여하는게 아니라, 해당하는 인덱스에 객체의 주소값을 참조하여 저장한다.

- List 인터페이스의 주요 메소드

boolean **add**(E e)	해당 리스트(list)에 전달된 **요소를 추가**함. (선택적 기능)
void **add(int index, E e)**	해당 리스트의 **특정 위치에 전달된 요소를 추가**함. (선택적 기능)
void **clear()**	해당 리스트의 모든 **요소를 제거함**. (선택적 기능)
boolean **contains**(Object o)	해당 리스트가 전달된 **객체를 포함하고 있는지를 확인**함.
boolean **equals**(Object o)	해당 리스트와 전달된 **객체가 같은지를 확인**함.
E **get**(int index)	해당 리스트의 특정 위치에 존재하는 **요소를 반환**함.
boolean **isEmpty()**	해당 리스트가 **비어있는지를 확인**함.
Iterator<E> **iterator()**	해당 리스트의 **반복자(iterator)를 반환**함.
boolean **remove**(Object o)	해당 리스트에서 전달된 **객체를 제거**함. (선택적 기능)
boolean **remove(int index)**	해당 리스트의 **특정 위치에 존재하는 요소를 제거**함. (선택적 기능)
E **set**(int index, E e)	해당 리스트의 특정 위치에 존재하는 **요소를 전달받은 객체로 대체**함. (선택적 기능)
int **size()**	해당 리스트의 요소의 **총 개수를 반환함.**
Object[] **toArray()**	해당 리스트의 모든 요소를 Object 타입의 **배열로 반환함.**

1} ArrayList

ArrayList는 가장 자주 쓰이는 List중 하나며 배열과 비슷하다. 데이터의 중복이 허용되며 입력된 객체값의 인덱스가 존재한다. 그 크기 또한 제한적이지 않고 유동적이다.

Ex)

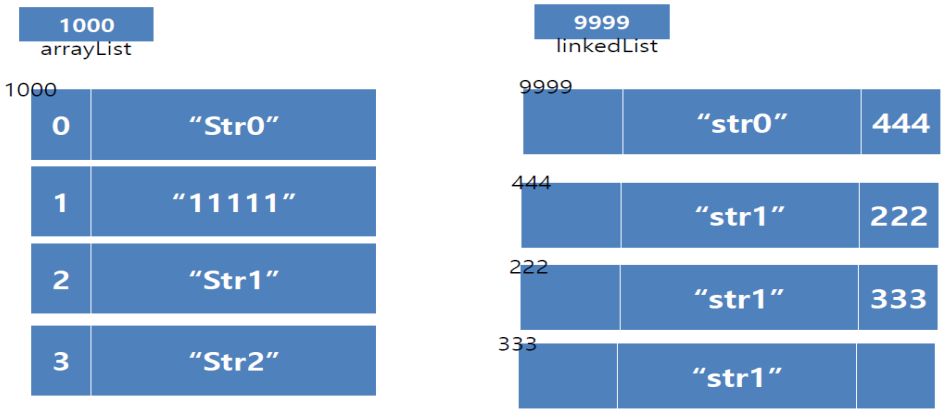
```java
package com.lec.ex1_list;
import java.util.ArrayList;
public class Ex01_ArrayList {
    public static void main(String[] args) {
        ArrayList<String> arrayList = new ArrayList<String>();
            // 제네릭 < >에 자료형 // String 형 Arraylist객체 생성
            //ArrayList는 기초데이터 타입이 아닌 객체(참조) 데이터 타입만 입력 가능하다.

        arrayList.add("str0");           // .add()메소드로 0번 인덱스 생성
        arrayList.add("str1");           // 1번 인덱스 생성
        arrayList.add("str2");           // 2번 인덱스 생성
        System.out.println(arrayList.get(0)); // .get() 메소드로 해당 인덱스의 값 가져올수 있음

        arrayList.add(1, "str11");      // 1번 인덱스에 "str11" 값추가
        System.out.println(arrayList.toString()); // 데이터가 추가된 객체 기준으로 하나씩 밀린다.
        arrayList.set(1, "11111");      // .set() 함수로 1번 인덱스의 값을 "11111"로 바꾼다
        System.out.println(arrayList);  //str0 11111 str1 str2

        for (String alist : arrayList) { //인덱스가 있기 때문에 for문으로 출력 가능하다.
            System.out.print(alist + "\t");
        }

        arrayList.remove(1);
      // 1번째 인덱스의 데이터 삭제 // 중간에 삭제하면 들여쓰기됨 (2->1번 인덱스, 3->2번 인덱스)
      // arrayList.remove("11111");     //인덱스 값이 아닌 데이터를 명시해서 삭제 가능하다
      //ArrayList는 데이터 중복이 가능하기때문에 데이터 값을 명시해서 삭제할때 중복된 데이터 중 하나만 삭제된다.
        arrayList.remove(arrayList.size() - 1); //.size()로 해당 인덱스 길이의 맨 마지막 인덱스 삭제

        for (int idx = 0; idx < arrayList.size(); idx++) {
            System.out.printf("%d번째 인덱스 값은 %s\t", idx, arrayList.get(idx));
        }

        arrayList.clear();          //.clear()메소드로 arrayList의 안의 모든 데이터 삭제

        if (arrayList.size() == 0) {        //.size() 해당 객체의 길이값 리턴
            System.out.println("arrayList의 데이터는 없습니다.");
        }
        if (arrayList.isEmpty()) {          //.isEmpty() 해당 객체의 데이터 유무 여부 불린값으로 리턴
            System.out.println("arrayList의 데이터는 없습니다.");
        }
        System.out.println(arrayList);  //데이터를 삭제해도 객체는 남아있기 때문에 ""[]"가 출력된다.
        arrayList = null;               // null값을 대입해 arrayList 객체 삭제
```

2} LinkedList

ArrayList와 거의 비슷하나 ArrayList는 접근시간(읽어오는데 걸리는 시간)은 빠르나 중간에서 데이터를 추가하거나 삭제 할때는 인덱스를 하나씩 조정해 줘야해서 작업 시간이 많이 걸린다는 단점이 있다. 이런 경우에는 각 데이터를 링크로 연결한 LinkedList가 더 빠르다. LinkedList는 각 데이터를 링크로 연결하며 데이터간 이중연결 리스트(다음 데이터와 이전 데이터를 가르킴)으로 되어있어 LinkedList에서 데이터가 추가, 삭제되는 경우엔 각 데이터가 가르키는 링크값만 변경하면 돼서 작업시간이 빠르다.

즉, 인덱스를 가지고 데이터에 접근하는 연산은 ArrayList의 성능이 더 좋고, 중간에 데이터를 삽입, 삭제가 빈번하게 일어나는 경우엔 LinkedList가 더 빠르다.

- ArrayList와 LinkedList



Ex)

```java
import java.util.ArrayList;
import java.util.LinkedList;

public class Ex02_linkedList {
    public static void main(String[] args) {

        ArrayList<String> arrayList = new ArrayList<>();
        LinkedList<String> linkedList = new LinkedList<>();
                                    //ArrayList와 메소드와 사용법이 비슷하다.
        linkedList.add("str0");      // 0번 인덱스
        linkedList.add("str1");      // 1번 인덱스
        linkedList.add("str2");      // 2번 인덱스
        linkedList.add("str3");      // 3번 인덱스

        linkedList.add(1, "str33");  // 1번 인덱스 자리에 "str33" 추가

        System.out.println(linkedList);  //str0, str33, str1, str2, str3

        for (String list : linkedList) {    //데이터 링크값이 있기때문에 for문 가능
            System.out.println(list);
        }

        linkedList.clear();         //linkedList 초기화
        System.out.println(linkedList.isEmpty() ? "데이터가 없습니다." : "데이터가 있습니다.");
        //.isEmpty()로 데이터 확인 // 데이터가 없습니다.
    }
```

2)Set 계열

Set계열의 자료구조에서는 대표적으로 HashSet, TreeSet이 있다.

- 데이터의 순서(인덱스)가 없으나 데이터의 중복은 허용 되지 않는다.
- 기본적으로 정렬이 불가능하다. 하지만 TreeSet클래스에선 가능하다.

-Set 인터페이스의 주요 메소드

boolean **add**(E e)	해당 집합(set)에 전달된 **요소를 추가**함. (선택적 기능)
void **clear**()	해당 집합의 **모든 요소를 제거**함. (선택적 기능)
boolean **contains**(Object o)	해당 집합이 전달된 객체를 **포함하고 있는지를 확인**함.
boolean **equals**(Object o)	해당 집합과 전달된 **객체가 같은지를 확인**함.
boolean **isEmpty**()	해당 집합이 **비어있는지를 확인**함.
Iterator<E> **iterator()**	해당 집합의 **반복자(iterator)를 반환**함.
boolean **remove**(Object o)	해당 집합에서 전달된 **객체를 제거**함. (선택적 기능)
int **size()**	해당 집합의 요소의 **총 개수를 반환**함.
Object[] **toArray()**	해당 집합의 모든 요소를 Object 타입의 **배열로 반환**함.

1} HashSet

Ex)

```java
import java.util.HashSet;
import java.util.Iterator;

public class Ex01_HashSet {

    public static void main(String[] args) {
        HashSet<String> hashSet = new HashSet<String>();
        hashSet.add("str0");        // 인덱스도 key값도 없음
        hashSet.add("str1");        // 하나의 값만 가져오는 get함수가 없다. iterator를 써야한다.

        System.out.println(hashSet); //str0 str1

        hashSet.add("str1");        // 중복된 값을 허용하지 않는다. //덮어쓰기
        System.out.println(hashSet); //str0 str1

        Iterator<String> iterator = hashSet.iterator();

        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }       //str1
    }           //str0
}
```

2}TreeSet

기본적으로 정렬이 되어있고 순서가 있다. 데이터의 중복이 허용 되지않는 Set인터페이스의 특징과 함께 특정 난수를 중복 되지 않게 출력해주고 정렬 또한 해주기에 로또 번호 추출 같은 로직에 유용하다

Ex)

```java
import java.util.Random;
import java.util.TreeSet;

public class Ex02_lotto {

    public static void main(String[] args) {

        TreeSet<Integer> lotto = new TreeSet<Integer>();
        //hasgset 클래스에서 중복 데이터를 덮어쓰기 때문에 다른수로 채워진다.
        Random random = new Random();
        int cnt = 0;
        while (lotto.size() < 6) {
            cnt++;              //몇번의 로직이 실행되었는지 카운트
            lotto.add(random.nextInt(45) + 1);      // 45사이의 정수난수
        }
        System.out.println(cnt + "" + lotto);
    }
}
```

5. iterator

iterator는 자바의 Collection 프레임 워크에서 Collection에 저장된 요소를 읽어오는 방법을 표준화 한것이다. 주로 인덱스가 없는 Map인터페이스나 Set인터페이스에서 주로 쓰이지만 모든 Collection프레임워크에서 사용 가능하다. 하지만 값을 변경하거나 추가가 불가능하고 대량의 데이터를 다룰 때 속도가 느리며 한번에 하나씩의 데이터만 가져온다는 단점이 있다.

- Iterator의 매소드

메서드	설명
boolean hasNext()	해당 이터레이션(iteration)이 다음 요소를 가지고 있으면 true를 반환하고, 더 이상 다음 요소를 가지고 있지 않으면 false를 반환함.
E next()	이터레이션(iteration)의 다음 요소를 반환함.
default void remove()	해당 반복자로 반환되는 마지막 요소를 현재 컬렉션에서 제거함. (선택적 기능)

Ex)예제

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
public class Ex04_Iterator {
    public static void main(String[] args) {

        ArrayList<String> list = new ArrayList<String>();
        list.add("STR");    // ArrayList 데이터의 중복을 허용함
        list.add("STR");
        System.out.println(list);
        Iterator<String> iterator1 = list.iterator();
                                //list계열은 배열이 있기때문에 iterator를 잘 사용하지 않는다.
        while (iterator1.hasNext()) {
            System.out.println(iterator1.next());
        }           // STR STR
        for (String l : list) {         //list계열은 for문을 주로 쓴다 .
            System.out.println(l);
        }           // STR STR

    // Map계열

        HashMap<String, String> map = new HashMap<String, String>();
        map.put("홍길동", "010-9999-9999");
        map.put("김길동", "010-9999-9999");//객체 출력할때 순서 상관관없을 때 iterator

        //Iterator형의 String자료형을 받는 iterator2객체 생성
        Iterator<String>iterator2 = map.keySet().iterator();
                                //HashMap map의 키값을 iterator넘긴다.
          while(iterator2.hasNext()) {
            String key = iterator2.next();
            System.out.println(key + " 키의 데이터 : "+ map.get(key));
                                                    //해당 key값의 데이터 리턴
            //김길동 키의 데이터 : 010-9999-9999
            //홍길동 키의 데이터 : 010-9999-9999
        }

    //Set 계열

        HashSet<String> set = new HashSet<String>();
        set.add("str0");
        set.add("str1");
        set.add("str1");

        Iterator<String> iterator3 = set.iterator();

        while(iterator3.hasNext()) {
            System.out.println(iterator3.next());
        }       //str1
                //str0
    }
}
```

3) Map

Map계열의 자료구조는 Collection과 다른 저장방식을 가지는데 Map인터페이스를 구현한 Map컬렉션 클라스들은 key와 값을 하나의 쌍으로 저장하는 방식을 사용한다. 여기서 key란 인덱스 대신 key 값으로 데이터를 액세스하는 역할을 한다. 이렇게 Map은 데이터를 찾아 갈수 있는 key와 실제 데이터를 저장한다고 해서 Dictionary라고도 한다. Map 인터페이스를 구현한 클래스는 HashMap, TreeMap, Hashtable이 있다.

1. 데이터의 저장순서를 유지하지 않는다. (순서를 보장하지 않는다.)
2. 저장순서의 불가능 때문에 역시 정렬 또한 불가능 하다.
3. Key는 해당 값을 연결하는 인덱스의 역할이기 때문에 중복을 허용하지 않지만 값은 중복 가능하다.
4. 만일 key값이 중복으로 입력될 경우 나중에 입력된 key값의 데이터가 기존의 값에 덮어씌어진다.

- Map 인터페이스의 주요 메소드

void **clear**()	해당 맵(map)의 **모든 매핑(mapping)을 제거**함.
boolean **containsKey**(Object key)	해당 맵이 **전달된 키를 포함하고 있는지를 확인**함.
boolean **containsValue**(Object value)	해당 맵이 **전달된 값에 해당하는 하나 이상의 키를 포함하고 있는지**를 확인함.
V **get(Object key)**	해당 맵에서 전달된 **키에 대응하는 값을 반환**함. 만약 해당 맵이 전달된 키를 포함한 매핑을 포함하고 있지 않으면 null을 반환함.
boolean **isEmpty**()	해당 **맵이 비어있는지를 확인함.**
Set<K> **keySet**()	해당 맵에 포함되어 있는 모든 **키로 만들어진 Set 객체를 반환**함.
V **put(K key, V value)**	해당 맵에 전달된 키에 대응하는 값으로 **특정 값을 매핑함**.
V **remove(Object key)**	해당 맵에서 전달된 키에 대응하는 **매핑을 제거함.**
V **replace(K key, V value)**	해당 맵에서 전달된 키에 대응하는 **값을 특정 값으로 대체함.**
boolean replace(K key, V oldValue, V newValue)	해당 맵에서 특정 값에 대응하는 전달된 키의 값을 새로운 값으로 대체함.
int **size()**	해당 맵의 **매핑의 총 개수를 반환함.**
boolean remove(Object key, Object value)	해당 맵에서 특정 값에 대응하는 특정 키의 매핑을 제거함.

1}HashMap

해시함수를 이용한 Map이며, 다른 Map의 클래스 중 가장 빈번하게 쓰인다 Map 자료 구조의 특징 대로 데이터 순서를 보장하지 않고 정렬이 불가능하다는 특징이 있다.

Ex)

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

public class Ex01_HashMap {

    public static void main(String[] args) {
        // HashMap<K,V> k = 인덱스가 어떤 타입인지 설정 // 객체 자료형만 올수있음
        HashMap<Integer, String> hashMap = new HashMap<Integer, String>();
                // 키   //데이터
        hashMap.put(11, "str11");               // 키값 11에 데이터 "str11" 추가 // 순서가 없다
        hashMap.put(new Integer(20), "str20");  // 키값 20에 데이터 "str20" 추가
        hashMap.put(33, "str33");               // 키값 33에 데이터 "str33" 추가
        System.out.println(hashMap.get(20));    // 키값이 20인 데이터 get()으로 불러온다

        hashMap.remove(20);                  // 키값 20인 데이터 삭제
        System.out.println("remove 후 :" + hashMap);
                                            //33=str33, 11=str11
        hashMap.clear();    //.clear()로 데이터 삭제
        System.out.println(hashMap.isEmpty() ? "데이터 모두 삭제" : "데이터 있음");
                //.isEmpty()함수로 데이터 여부 불린값 리턴 //데이터 모두 삭제
        hashMap.put(0, "Hong gildong");
        hashMap.put(10, "Kim dongil");
        hashMap.put(22, "Lee soonsin");
        hashMap.put(40, "Yu ain");
        System.out.println(hashMap);
        // {0=Hong gildong, 22=Lee soonsin, 40=Yu ain, 10=Kim dongil}

        Iterator<Integer> iterator = hashMap.keySet().iterator();
                            //인덱스가 없는 Collection 프레임워크는 Iterator로 요소를 읽어온다.
        while (iterator.hasNext()) {//인덱스가 없기에 while문으로 반복 출력
                                    //hasNext()는 해당 객체에 다음 요소가 있는지를 불린값 리턴
            Integer key = iterator.next();          //iterator의 키값을 key변수에
            System.out.println(key + "의 데이터는 " + hashMap.get(key));
        }                   //키값                //해당 키값의 데이터를 리턴
                    // 0의 데이터는 Hong gildong
                    // 22의 데이터는 Lee soonsin
                    // 40의 데이터는 Yu ain
                    // 10의 데이터는 Kim dongil
```