**[ 16 ] Java Collection**

오늘 알아보게 될 collection는 자료구조이다. 쉽게 말해서 배열과 다소 비슷하다고 생각하자. 어떠한 자료를 나중에 재활용하기 위해서 우리는 데이터를 차곡차곡 잘 정리해서 저장해 두어야 한다. 정리할 때 데이터의 속성에 맞춰서 저장해야 하므로 자료구조가 여러 가지 존재할 수 있다. 이러한 자료구조들을 collections라고 한다.

* 컬렉션이란
* List 계열, Map 계열, Set 계열
* Iterator(반복자) 자료구조 살펴보기

1. Cellection 이란?

컬렉션이란, 우리말로 쉽게 말해서 자료구조입니다. 쉽게 말해서 다수의 데이터, 즉 데이터그룹을 의미합니다.

데이터 군(群)을 저장하는 하기 위해서 우리는 배열을 사용해왔는데요, 배열과 비슷한 구조에다가 다수의 데이터를 쉽게 처리할 수 있는 방법(method)을 제공하는 API입니다.

우리가 배웠던 배열이 아주 훌륭한 자료구조입니다. 하지만 더욱 훌륭한 자료구조 형을 JAVA에서는 많이 준비했습니다. 그리고 우리는 어려운 자료구조 형의 내부를 공부할 필요 없이 API document를 보면서 그냥 사용만 하면 됩니다.

무수히 많은 데이터를 어떤 형식으로 잘 정리하느냐에 따라 추후 데이터를 찾아서 사용 할 때 업무의 효율성이 높아 질 수 있습니다. 그래서 JAVA에서는 다양한 자료구조형를 제공 하고 있습니다. **다양한 자료구조 형이 제공되는 이유는 데이터의 성질에 따라서 데이터를 관리(정리)해야 하는 방식이 다르기 때문**입니다. 자료구조형 안에서는 객체의 레퍼런스 만을 관리합니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 주소값 | 주소값 | 주소값 | . . . . . . | 주소값 |

2. 자바에서 제공되는 Collection 자료구조들

Collection

List

(순서, 중복 허용)

Map

(key, value)

Set

(순서X, 중복X)

**HashMap**

LinkedList

Vector

TreeSet

**ArrayList**

HashSet

Properties

(1) List계열 Collection 클래스 살펴보기

자료구조 중 장 많이 사용하고 쉽게 사용할 수 있는 자료구조형이 List계열입니다. List는 배열과 비슷하지만, 배열의 단점을 보완하였습니다. List는 처음 만들 때 크기를 고정하지 않아도 되고 그 크기는 유동적입니다

* ArrayList ; 배열과 매우 비슷. 인덱스가 존재하며 데이터는 중복을 허용. 인덱스가 가장 중요
  + - add(객체)
    - add(index, 객체)
    - set(index, 객체)
    - get(index)
    - size()
    - remove(index)
    - remove(객체)
    - contains(객체) 주어진 객체가 저장되어 있는지 여부를 boolean 값으로 반환
    - isEmpty()
    - clear()

**public** **static** **void** main(String[] args) {

String[] array = **new** String[5];

array[0] = "str0";

array[1] = "str1";

array[2] = "str2";

array[3] = "str3";

array[4] = "str4";

System.out.println(array);

**for**(String a : array)

System.***out***.print(a+"\t");

System.***out***.println();

ArrayList<String> arrayList = **new** ArrayList<String>();

arrayList.add("str0"); //0번째 인덱스

arrayList.add("str1"); //1번째 인덱스

arrayList.add("str2"); //2번째 인덱스

arrayList.add("str3"); //3번째 인덱스

arrayList.add("str4"); //4번째 인덱스

System.***out***.println(arrayList.toString());

String index3 = arrayList.get(3);

System.***out***.println("index3="+index3);

arrayList.set(2, "str2222");

System.***out***.println(arrayList);

**int** size = arrayList.size();

System.***out***.println("size = "+size);

arrayList.remove(2);

System.***out***.println(arrayList);

arrayList.clear();

System.***out***.println(arrayList);

arrayList = **null**;

System.***out***.println(arrayList);

System.***out***.println(arrayList.size());

}

* LinkedList ; ArrayList와 거의 비슷. ArrayList는 접근시간(읽어오는데 걸리는) 시간은 빠르나 데이터를 추가하거나 삭제하는데 많은 데이터를 옮겨야 해서 시간이 많이 걸린다는 단점이 있다. 이점을 보완한 LinkedList. ArrayList와 달리 불연속적으로 존재하는 데이터를 연결. 순차적으로 데이터를 추가/삭제할 경우 ArrayList가 빠르고 비순차적으로 데이터를 추가/삭제하는 경우 LinkedList가 빠르다. 접근시간도 ArrayList가 빠르다

**public** **static** **void** main(String[] args) {

LinkedList<String> linkedlist = **new** LinkedList<String>();

linkedlist.add("str0");

linkedlist.add("str1");

linkedlist.add("str2");

linkedlist.add("str3");

System.***out***.println(linkedlist);

linkedlist.add(2,"STR2");

System.***out***.println(linkedlist);

System.***out***.println("사이즈는 "+linkedlist.size());

linkedlist.remove(2);

System.***out***.println(linkedlist);

linkedlist.clear();

System.***out***.println(linkedlist);

System.***out***.println(linkedlist.isEmpty()? "비워졌다":"안 비워졌다");

linkedlist = **null**;

System.***out***.println(linkedlist);

}

* Vector

**class** AClass {

String stra = "A형";

@Override

**public** String toString() {

**return** stra;

}

}

**class** BClass {

String strb = "B형";

@Override

**public** String toString() {

**return** strb;

}

}

**public** **class** VectorEx {

**public** **static** **void** main(String[] args) {

AClass aObj = **new** AClass();

BClass bObj = **new** BClass();

String cObj = "C";

ArrayList<Object> list = **new** ArrayList<Object>();

Vector<Object> vec = **new** Vector<Object>();

list.add(aObj);list.add(bObj);list.add(cObj);

vec.add(aObj); vec.add(bObj); vec.add(cObj);

System.***out***.println(list);

System.***out***.println(vec);

**for**(**int** i=0 ; i<list.size() ; i++){

Object obj = list.get(i);

System.***out***.print(obj+"\t");

}

System.***out***.println();

**for**(**int** i=0 ; i<vec.size() ; i++){

Object obj = vec.elementAt(i);

// Object obj = vec.get(i);

System.***out***.print(obj+"\t");

}

}

}

(2) Map 계열 Collection 클래스 살펴보기

Map계열의 자료구조는 인덱스 대신 키 값으로 데이터를 액세스합니다.

List계열과 달리 인덱스가 없고, 키와 값만 있습니다. 그리고 키는 유니크 해야 합니다. 우리가 값을 관리하고자 한다면 키를 이용해서 값을 관리할 수 있습니다.

* HaspMap

**public** **static** **void** main(String[] args) {

HashMap<Integer, String> hashmap = **new** HashMap<Integer, String>();

hashmap.put(0, "str0");

hashmap.put(1, "sr1");

hashmap.put(2, "str2");

hashmap.put(3, "str3");

System.***out***.println(hashmap);

String str = hashmap.get(2);

System.***out***.println("Str:"+str);

hashmap.remove(2);

System.***out***.println("remove 한 후"+hashmap);

hashmap.clear();

System.***out***.println("clear 한 후"+hashmap);

hashmap.put(0, "Hong gildong");

hashmap.put(1, "Kim dongil");

hashmap.put(2, "Lee sumsin");

hashmap.put(3, "Yu ain");

System.***out***.println(hashmap);

Iterator<Integer> iterator = hashmap.keySet().iterator();

**while**(iterator.hasNext()){

**Integer key =** iterator.next();

String string = hashmap.get(key);

System.***out***.println(key+”번 “+string);

}

}

(3) Set 계열 Collection 클래스 살펴보기

Set계열 자료구조에서는 데이터의 순서는 없습니다(인덱스 없다). 하지만 중복된 데이터는 허락하지 않습니다.

중복된 데이터의 의미는 hashcode()값이 같거나 equal()메소드의 결과값에 의해 해석

* HashSet

**public** **static** **void** main(String[] args) {

HashSet<String> hashset = **new** HashSet<String>();

hashset.add("str0");

hashset.add("str1");

hashset.add("str2");

hashset.add("str3");

hashset.add("str2");

System.***out***.println(hashset);

hashset.remove("str0");

System.***out***.println(hashset);

System.***out***.println("사이즈 : " + hashset.size());

}

**public** **static** **void** main(String[] args) {

Random ran = **new** Random();

HashSet<Integer> lotto = **new** HashSet<>();

**while**(lotto.size()<6)

lotto.add(ran.nextInt(45)+1);

System.***out***.println(lotto);

}

**public** **class** Student {

**private** String name;

**private** **int** grade;

**public** Student(String name, **int** grade){

**this**.name = name;

**this**.grade = grade;

}

@Override

**public** String toString() {

**return** name + " : "+ grade;

}

@Override

**public** **boolean** equals(Object obj) {

String compareString = obj.toString();

String thisString = toString();

**return** thisStr.equals(compareStr);

}

@Override

**public** **int** hashCode() {

**return** toString().hashCode();

}

}

**public** **class** StudentMain {

**public** **static** **void** main(String[] args) {

HashSet<Student> hashSet = **new** HashSet<Student>();

hashSet.add(**new** Student("홍길동", 3));

hashSet.add(**new** Student("이순신", 6));

hashSet.add(**new** Student("장보고", 1));

System.***out***.println(hashSet);

Student student = **new** Student("이순신", 6);

hashSet.remove(student);

System.***out***.println(hashSet);

}

}

3. Iterator(반복자) 자료구조 살펴보기

JAVA에는 무수히 많은 자료구조형이 있다고 하였습니다. 그리고 지금까지 가장 많이 쓰이고, 중요한 몇 가지를 살펴 보았습니다.

다시 한번 말씀드리면. 많은 자료구조형이 존재하는 것은 결국은 데이터를 잘 관리하기 위해서입니다. 잘 관리한다는 것은 저장하고, 정렬하고, 검색하는 것입니다. 이중에서 데이터의 검색은 무엇보다도 중요 합니다. 그래서 JAVA에서는 Iterator라는 인터페이스가 있습니다. Iterator라는 의미는 ‘반복자＇라는 의미로 데이터를 반복적으로 검색하는데 아주 유용한 인터페이스 입니다. 참고로 모든 자료구조형은 iterator() 메소드를 지원 하고 있습니다.

* Iterator

**public** **static** **void** main(String[] args) {

ArrayList<String> arrayList = **new** ArrayList<String>();

arrayList.add("str0");

arrayList.add("str1");

arrayList.add("str2");

System.***out***.println(arrayList);

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

**while**(iterator.hasNext()){

System.***out***.println(iterator.next());

}

}

**public** **static** **void** main(String[] args) {

HashMap<Integer, String> hashMap = **new** HashMap<Integer, String>();

hashMap.put(0, "str0");

hashMap.put(1, "str1");

hashMap.put(2, "str2");

hashMap.put(3, "str3");

hashMap.put(4, "str4");

System.***out***.println(hashMap);

//Iterator<Integer> iterator = hashMap.keySet().iterator();

Set<Integer> set = hashMap.keySet();

Iterator<Integer> iterator = set.iterator();

**while**(iterator.hasNext()){

Integer key = iterator.next();

System.***out***.println(key + " : "+hashMap.get(key));

}

}

|  |  |
| --- | --- |
|  | 특 징 |
| List | 순서가 있는 데이터의 집합, 데이터의 중복을 허용한다  ex. 대기자명단 |
| 구현클래스 : ArrayList, LinkedList, Stack, Vector 등 |
| Set | 순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않는다. |
| 구현클래스 : HashSet, TreeSet 등 |
| Map | 키(key)와 값(value)의 쌍(pair)으로 이루어진 데이터의 집합  순서는 유지되지 않으며, 키는 중복을 허용하지 않고 값은 중복을 허용한다 |
| 구현클래스 : HashMap, HashTable 등 |

<그 밖의 TreeSet 예제>

TreeSet에 7,4,9,1,5를 저장하면 트리 구조로 좌측엔 작은 값이 우측엔 큰 값이 저장된다.

**public** **static** **void** main(String[] args) {

TreeSet<Integer> tree = **new** TreeSet<>();

**while**(tree.size()<6) {

tree.add((**int**)(Math.*random*()\*45)+1);

}

System.***out***.println(tree);

Iterator<Integer> iterator = tree.iterator();

**int**[] lotto = **new** **int**[6];

**int** i=0;

**while**(iterator.hasNext()) {

lotto[i++]=iterator.next();

}

**for**(i=0 ; i<lotto.length ; i++)

System.***out***.print((i+1)+"번째 로또\t"+lotto[i]);

}

4. 학습 예제

(1) TestMain1.java에서, N이나 n을 입력할 때까지 회원가입 정보(이름, 전화번호, 주소)를 ArrayList에 받고, N이나 n을 입력할 경우 가입한 모든 회원들의 정보를 아래와 같이 콘솔창에 출력하는 프로그램을 구현하시오

(2) TestMain2.java에서, N이나 n을 입력할 때까지 회원가입 정보(이름, 전화번호, 주소)를 HashMap에 받고, N이나 n을 입력할 경우 가입한 모든 회원들의 정보를 아래와 같이 콘솔창에 출력하는 프로그램을 구현하시오(단, HashMap의 킷값은 전화번호, 데이터는 회원정보로 한다)

[더이상 가입할 회원이 없어서 N이나 n을 입력했을 때, 콘솔창 화면]

|  |
| --- |
| 홍길동 010-9999-9999 서울시 용산구 용산동  김마동 010-8888-8888 서울시 마포구 노고산동  . . . . . |

[제출파일]

- Customer.java

- TestMain1.java

- TestMain2.java

- TestMain1.java의 실행화면.jpg또는 실행화면.png

- TestMain2.java의 실행화면.jpg 또는 실행화면.png