06/22 수

복습.

LinkedList

LinkedList는 추가와 삭제에 용이한 List

ArrayList는 모든 원소를 순환할 때, 모든 원소에 빨리 접근할 때 용이.

LinkedList 는 추가와 삭제가 빈번하다.

→ 첫번째 원소, 마지막 원소의 추가와 제거, 가져오기 메소드가 추가로 정의되어있음

< addFirst(); removeFirst(); getLast(); 등등>

220621-01 linked list의 Main3

→ Queue의 형태로 구현할 수 있음. List의 구현체 이면서, Queue의 구현체.

→ Queue는 선입선출

→ 제일 마지막과 제일 앞의 것의 추가와 제거에 집중한다.

< poll(); remove(); >

→ Deque(Stack) : Queue와 비슷하지만 다름. 후입선출.

< push(); pop(); >

Set → 집합

220622-01 set

List는 중복에 약하다.

Set - 중복에 집중

→ set은 인터페이스로 구현되어 있다. (util패키지)

→ add메소드로 추가된 원소는 객체의 형태를 띄고 있다.

→ 중복된 원소값이 입력되지 않는다.

→ index번호를 가지지 않음 → get메소드 자체가 없다.

→ iterator 상속받기 때문에 사용가능.

→ for-each문과 iterator로 모든 원소의 접근이 가능함.

→ 특징 : 삽입된 순서를 신경쓰지 않는다. Index도 존재하지 않음.

→ Set 안에 list의 원소를 넣어주면 알아서 중복을 걸러준다.

→ ArrayList의 생성자에는 List만 넣을 수 있는 것이 아니라, Collection을 상속받는 타입을 모두 집어 넣을 수 있다 →→→ set도 사용가능.

List<String> list = **new** ArrayList<>(Arrays.*asList*("가", "나", "다", "다", "다", "가", "라"));

Set<String> set = **new** HashSet<>(list);

→ 차집합을 구할 때, 하나의 set의 모든 원소를 넣고 다른 set의 모든 원소를 제거하면 차집합이 나옴.

→ set은 순서가 없기 때문에 equals로 동등함을 판단할 때, 원소만 모두 들어있다면 true로 반환된다.

LinkedHashSet → set에서 원소의 추가된 순서를 가질 뿐, index는 여전히 가지고 있지않다. → 편의상 사용. 여전히 set이기 때문에 index X

TreeSet → **대소 비교**를 할 수 있는 원소를 넣으면,

정렬된 순서(기본 오름차순)를 가진 원소를 가지며, 중복을 제거한다.

(나뭇가지 모양이라서 TreeSet)

<String> 일 땐, 알파벳 순.

→ String에 CompareTo가 정의되어 있기 때문에 기본 오름차순.

Hash

**Hash : 각 객체마다 가지고 있는 고유한 숫자값!**

→ 동등한 객체끼리는 같은 숫자값이 될 수 있도록 만들어 준다.

→ Java 1.7 or higher 사용가능

검사 순서 :

1차적으로 hash값 검사 → hash값이 같으면 → 2차 equals검사

→ 해시코드는 기본적으로 오브젝트에서 정의되어있다.

기본적으로 객체마다 해시코드가 생성됨.

인스턴스는 모두 다른 해시코드 값을 가지고 있다.

Objects의 hash메소드를 사용해서 편리하게 해시코드를 생성 가능.

< Objects.hash(); >

Map

220622-02 map

put으로 값을 집어넣는다.

Put(key, value) key와 value는 한쌍

hashmap은 map이라는 인터페이스로 구현이 가능하다.