제 37강

컬렉션 프레임웍2

교재:p225~231

목차

1. Set

Set(집합)?

: 요소들을 집합적으로 모아놓은 자료구조

- 특징
  - 중복을 허용하지 않음
  - 저장 순서를 유지하지 않음

1. Set

Set을 구현한 클래스: HashSet, TreeSet

[1] HashSet

: Set과 동일하게 중복이 없고, 순서가 없다.

생성자	설명
HashSet( )	HashSet 클래스의 기본 생성자
HashSet(Collection c)	컬렉션의 요소로 HashSet 객체 생성
HashSet(int capacity)	capacity 용량을 갖는 객체 생성

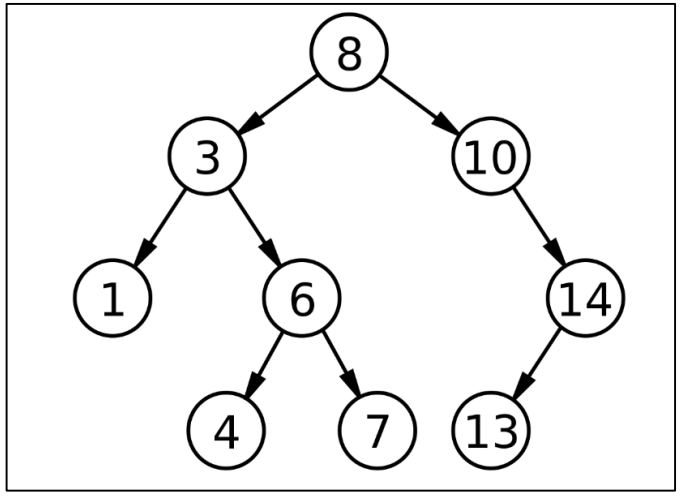
중복된 요소 add( )시, 집합에 추가 안되고 false 값 반환!

1. Set

Set을 구현한 클래스: HashSet, TreeSet

[1] TreeSet

: Set과 동일하게 중복이 없으나, 정렬기능이 추가된 형태



데이터 추가 시,
기준 데이터보다 작다면 기준의 왼쪽에
기준데이터보다 크다면 기준의 오른쪽에 위치

→ 이진 탐색 트리의 구조

1. Set

<실습> Exam-89.java

HashSet과 TreeSet의 요소를 저장한 결과를 비교

```java
public class TreeSet1 {
    public static void main(String[] args) {
            HashSet hs=new HashSet();
            hs.add("demon");
            hs.add("banana");
            hs.add("tomato");
            hs.add("apple");
            hs.add("cargo");

            TreeSet ts =new TreeSet();
            ts.add("demon");
            ts.add("banana");
            ts.add("tomato");
            ts.add("apple");
            ts.add("cargo");

            Iterator it= hs.iterator();
            System.out.println("<HashSet 출력>");
            while(it.hasNext()) {
                System.out.print(" "+it.next());
            }
            System.out.println();
            Iterator it2= ts.iterator();
            System.out.println("<TreesSet 출력>");
            while(it2.hasNext()) {
                System.out.print(" "+it2.next());
            }
            System.out.println();
            System.out.println("현재 Treeset의 크기:"+ts.size());
    }
}
```

<실행 결과>

```
<HashSet 출력>
 banana apple demon tomato cargo
<TreesSet 출력>
 apple banana cargo demon tomato
현재 Treeset의 크기:5
```

2. List

List(리스트)?

: 데이터를 일렬로 늘어놓은 구조

- 특징
- 순서가 있음
- 중복 허용

List 인터페이스를 구현한 클래스: ArrayList, LinkedList

2. List

List 인터페이스의 주요 메서드

메서드	설명
add(int index,E elem)	index 위치에 elem 추가
get(int index)	index 위치에 있는 요소 반환
indexOf(Object o)	요소 o가 있는 위치 반환
listiterator()	Listiterator( ) 반환
remove(int index)	index 위치 요소 삭제 후 삭제값 반환
set(int index,E elem)	index 위치 요소를 elem으로 변경

2. List

[1] ArrayList

: 요소 추가 시 0번 인덱스부터 차례대로 요소 저장(배열에서 발전된 형태)

<실습> Exam-90.java

```
public class ArrayList1 {
    public static void main(String[] args) {
        ArrayList list1= new ArrayList(10);
        list1.add("A");
        list1.add("C");
        list1.add("E");
        list1.add("D");

        System.out.println("초기상태: "+list1);
        System.out.print("인덱스 1위치에 B 추가:");
        list1.add(1,"B");
        System.out.println(list1);

        System.out.print("인덱스 2 위치의 값 삭제:");
        list1.remove(2);
        System.out.println(list1);

        System.out.println("인덱스2번 위치의 값 반환:"+list1.get(2));

    }
}
```

<실행 결과>

```
초기상태: [A, C, E, D]
인덱스 1위치에 B 추가:[A, B, C, E, D]
인덱스 2 위치의 값 삭제:[A, B, E, D]
인덱스2번 위치의 값 반환:E
```

2. List

[2] LinkedList

: 요소들이 서로 연결되어있는 리스트(각 요소가 다음 요소의 주소를 저장)

<실습> Exam-91.java

```
public class TimeCheck {
    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        LinkedList ll = new LinkedList();
        long start = System.currentTimeMillis();

        for(int i=0;i<100000;i++) {
            al.add(0,String.valueOf(i));
        }

        long end = System.currentTimeMillis();
        System.out.println("ArrayList 작업시간: "+ (end-start));
        start =System.currentTimeMillis();

        for(int i=0;i<100000;i++) {
            ll.add(0,String.valueOf(i));
        }
        end=System.currentTimeMillis();
        System.out.println("LinkedList 작업시간: "+ (end-start));
    }
}
```

<실행 결과>

```
ArrayList 작업시간: 536
LinkedList 작업시간: 22
```