Collection Framework

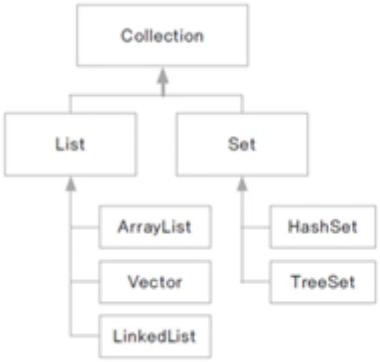
작성자: 신채린

컬렉션 프레임워크란?

- 프로그램 구현에 필요한 자료구조와 알고리즘을 구현해 놓은 라이브러리
- util 패키지에 구현되어 있음
- 개발에 소요되는 시간을 절약하고 최적화된 라이브러리를 사용할 수 있음
- Collection 인터페이스와 Map 인터페이스로 구성됨

Collection Interface

- 하나의 객체의 관리를 위해 선언된 인터페이스로 필요한 기본 메서드가 선언되어 있음
- 하위에 list, Set 인터페이스가 있음



분류	설명
List 인터페이스	순서가 있는 자료 관리, 중복 허용. 이 인터페이스를 구현한 클래스는 ArrayList, Vectior, LinkedList, Stack, Queue 등이 있음
Set 인터페이스	순서가 정해져 있지 않음, 중복을 허용하지 않음. 이 인터페이스를 구현한 클래스는 HashSet, TressSet 등이 있음

List Interface

- 하나의 자료형을 위한 인터페이스
- Collection 하위 인터페이스
- 객체를 **순서에 따라 저장하고 관리**하는데 필요한 메서드가 선언된 인터페이스
- 배열의 기능을 구현하기 위한 메서드가 선언됨
- ArrayList, Vector, LinkedList

ArrayList와 Vector

- 객체 배열 클래스
- Vector는 자바2부터 제공된 클래스
- 일반적으로 ArrayList를 더 많이 사용
- Vector는 멀티 쓰레드 프로그램에서 동기화를 지원
    - 동기화 (synchronization): 2개의 쓰레드가 동시에 하나의 리소스에 접근할 때 순서를 맞추어서 데이터의 오류가 방지하지 않도록 함
- capacity와 size는 다른 의미임

ArrayList와 LinkedList

- 둘다 자료의 순차적 구조를 구현한 클래스
- ArrayList는 배열을 구현한 클래스로 논리적 순서와 물리적 순서가 동일함
- LinkedList는 **논리적**으로 **순차적인 구조**지만, **물리적으로는 순차적이지 않을 수 있음**
- LinkedList 구조

- LinkedList에서 자료의 추가와 삭제

LinkedList 예시

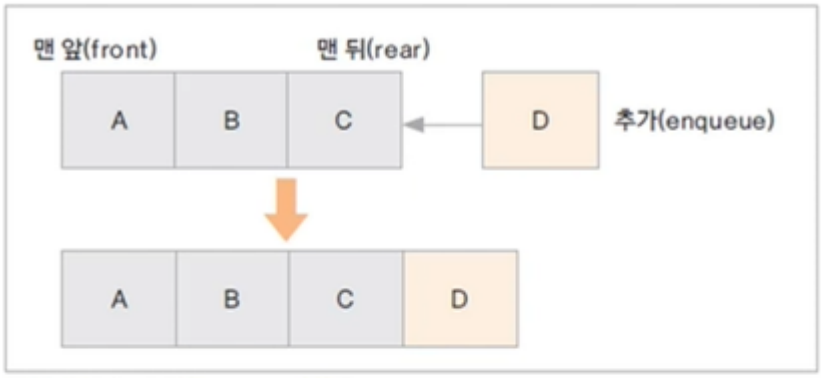
```
package Chapter11.collection;

import java.util.LinkedList;

public class LinkedListTest {
    public static void main(String[] args) {
        LinkedList<String> myList = new LinkedList<String>();

        myList.add("A");
        myList.add("B");
        myList.add("C");

        System.out.println(myList); // [A, B, C]
```

```
        myList.add(1, "D");
        System.out.println(myList); // [A, D, B, C]
        myList.removeLast() ;
        System.out.println(myList); //[A, D, B]

        for (int i=0; i< myList.size(); i++) {
            String s = myList.get(i);
            System.out.println(s);
        }


    }
}
```

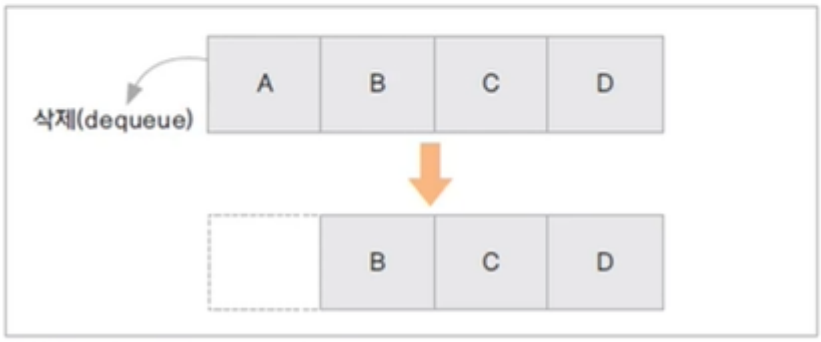
Stack & Queue 구현하기

Stack 구현하기

- Last in First Out (LIFO): 맨 마지막에 추가된 요소가 가장 먼저 꺼내지는 자료구조
- 선착순, 대기열 등을 구현할 때 가장 많이 사용되는 자료 구조
- ArrayList나 LinkedList로 구현할 수 있음



큐에서 요소 추가(enqueue)하기



큐에서 요소 삭제(dequeue)하기

- 예시

```
package Chapter11.collection;

import java.util.ArrayList;

class MyStack {
    private ArrayList<String> arrayStack = new ArrayList<String>();

    public void push(String data) {
        arrayStack.add(data);
    }

    public String pop() {
        int len = arrayStack.size();
        if (len == 0) {
            System.out.println("스택이 비었습니다.");
            return null;
        }
        return arrayStack.remove(len - 1);
    }
}

public class StackTest {
    public static void main(String[] args) {
        MyStack stack = new MyStack();
        stack.push("A");
        stack.push("B");
```

```
        stack.push("C");

        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
        System.out.println(stack.pop());
    }

}
```

Set 인터페이스

- set 인터페이스는 중복을 허용하지 않으며, 순서대로 출력되지 않음
- Collection 하위의 인터페이스
- List는 순서기반의 인터페이스이지만, Set은 순서가 없음
- get(i) 메서드가 제공되지 않음 (Iterator 로 순회)
- 저장된 순서와 출력순서는 다를 수 있음
- 아이디, 주민번호, 사번 등 유일한 값이나 객체를 관리할 때 사용
- HashSet, TreeSet 클래스

Iterator로 순회하기

- Collection의 개체를 순회하는 인터페이스
- iterator() 메서드 호출

o Iterator ir = memberArrayList.iterator();

Iterator에 선언된 메서드

- boolean hashNext()
  - 이후에 요소가 더 있는지를 체크하는 메서드이며, 요소가 있다면 true를 반환합니다.
- E next()
  - 다음에 있는 요소를 반환합니다.

예시

```
package Chapter11.collection.set;

import java.util.HashSet;

public class HashSetTest {
    public static void main(String[] args) {
        HashSet<String> set = new HashSet<String>();
        set.add("이순신");
        set.add("김유신");
        set.add("강감찬");
        set.add("이순신");
        System.out.println(set);
    }
}
```

HashSet 클래스

- Set 인터페이스를 구현한 클래스
- 중복을 허용하지 않으므로 저장되는 객체의 동일함 여부를 알기 위해 equals()와 hashCode() 메서드를 재정의 해야 함

TreeSet 클래스

- 객체의 정렬에 사용되는 클래스
- 중복을 허용하지 않으면서 오름차순이나 내림차순으로 객체를 정렬함
- 내부적으로 이진 검색 트리 (binary search tree)로 구현되어 있음
- 이진 검색 트리에 자료가 저장될 때 비교하여 저장될 위치를 정함
- 객체 비교를 위해 Comparable이나 Comparator 인터페이스를 구현해야 함
    - Comparator는 new 생성자를 사용해야 함

Comparable 인터페이스와 Comparator 인터페이스

- 정렬 대상이 되는 클래스가 구현해야 하는 인터페이스
- Comparable은 compareTo() 메서드를 구현
    - 매개 변수와 객체 자신(this)를 비교
- Comparator는 compare() 메서드를 구현

- 두 개의 매개 변수를 비교
- TreeSet 생성자에 Comparator가 구현된 객체를 매개변수로 전달

o TreeSet<Member> treeSet = new TreeSet<Member> (new Member());

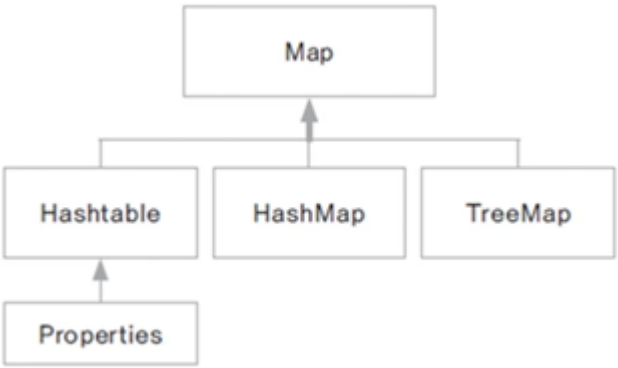
- 일반적으로 Comparable을 더 많이 사용
- 이미 Comparable이 구현된 경우 Comparator를 이용하여 다른 정렬 방식을 정의할 수 있음

Map 인터페이스

- **쌍으로 이루어진 객체를 관리**하는데 필요한 여러 메서드가 선언되어 있음
- Map을 사용하는 객체는 key-value 쌍으로 되어 있고, key는 중복될 수 없음
- 검색을 위한 자료 구조
- key를 이용하여 값을 저장하거나 검색, 삭제 할 때 사용하면 편리함
- 내부적으로 hash 방식으로 구현됨

o `index = hash(key)` //index는 저장 위치

- key가 되는 객체는 객체의 유일성함의 여부를 알기 위해 equals()와 hashCode() 메서드를 재정의함



HashMap 클래스

- Map 인터페이스를 구현한 클래스 중 가장 일반적으로 사용하는 클래스
- HashTable 클래스는 자바2부터 제공된 클래스로 Vector 처럼 동기화를 제공함

- pair 자료를 쉽고 빠르게 관리할 수 있음

TreeMap 클래스

- key 객체를 정렬하여 key-value를 pair로 관리하는 클래스
- key에 사용되는 클래스에 Comparable, Comparator 인터페이스를 구현
- java에 많은 클래스들은 이미 Comparable 이 구현되어 있음
- 구현된 클래스를 key로 사용하는 경우는 구현할 필요 없음

```
public final class Integer extends Number implements Comparable<integer> {
...
   public int compareTo(Integer anotherInteger) {
       return compare(this.value, anotherInteger.value);
   }
```