컬렉션 프레임워크

Vector 클래스

- java.util.Vector
- List 인터페이스를 구현한 클래스
- 객체들을 삽입, 삭제, 검색 할 수 있는 콘테이너 클래스
- 배열의 길이 제한 단점을 극복
- 아이템을 맨 마지막이나 중간에 삽입할 수 있음
- 객체수가 많아지면 자동으로 크기 조절
    - 맨 뒤에 객체 추가 : 공간이 모자르면 자동 늘림
    - 중간에 객체 추가 : 뒤에 존재하는 객체 한칸씩 뒤로이동
    - 객체 삭제 : 삭제 후 한칸씩 앞으로 자동이동

Vector 내부구조

- add() 메소드로 요소 삽입
- get() 메소드로 요소 검색
- String, Integer, Person 등 다양한 타입의 객체 삽입 가능
- 요소들은 인덱스로 관리 (0 부터 시작)

Vector 주요 메소드

- add(E e) : 벡터 맨 뒤에 요소추가
- add(int index, E e) : 지정된 인덱스에 지정된 객체 추가
- capacity() : 벡터 현재 용량 반환
- addAll(c) : c 가 지정하는 모든 요소 벡터 맨 뒤에 추가
- clear() : 벡터 모든 요소 삭제
- contains(Object o) : 벡터가 지정된 객체를 포함하고 있으면 true
- elementAt(int index) : 지정된 인덱스 요소 반환
- get(int index) : 지정된 인덱스 요소 반환
- indexOf(Object o) : 지정된 객체와 같은 첫 번째 요소 인스 반환. 없으면 -1 반ㄴ
- isEmpty() : 벡터가 비어있으면 true
- remove(int index) : 지정된 인덱스 요소 삭제
- remove(Objcect o) : 지정된 객체와 같은 첫째 요소 벡터에서 삭제
- removeAllElements() : 모돈 요소 삭제하고 크기를 0 으로 만듬
- size() : 벡터가 포함하는 요소의 개수 반환
- toArray() : 벡터의 모든 요소를 포함하는 배열 반환

ArrayList 란 무엇인가?

ArrayList 클래스

- java.util.ArrayList<E>
- 객체들을 삽입, 삭제, 검색할 수 있는 컨테이너 클래스.
- 배열 길이 제한단점을 극복할 수 있다.
    - 배열 선언 시 인덱스를 다 채우거나 못 채울 경우 해결
    - 배열은 인덱스가 꽉 차면 더 이상 값을 넣을 수 없다
    - 배열은 인덱스가 비어있으면 메모리가 낭비된다.
- 객체수가 많아지면 자동으로 크기조절
- 아이템을 벡터의 마지막이나 중간에 삽입할 수 있다.
    - 맨 뒤에 객체 추가 : 공간이 모자르면 자동 늘림
    - 중간에 객체 추가 : 뒤에 존재하는 객체 한칸씩 뒤로이동
    - 객체 삭제 : 삭제 후 한칸씩 앞으로 자동이동

ArrayList 내부구조

- add() 메소드로 요소 삽입
- get() 메소드로 요소 검색
- String, Integer, Person 등 다양한 타입의 객체 삽입 가능
- 요소들은 인덱스로 관리 (0 부터 시작)

ArrayList 주요 메소드

- aad(E e) : 맨 뒤에 요소추가
- add(int index, E e) : 지정된 인덱스에 지정된 객체 추가
- addAll(c) : c 가 지정하는 모든 요소 벡터 맨 뒤에 추가
- clear() : 모든 요소 삭제
- contains(Object o) : 지정된 객체를 포함하고 있으면 true
- elementAt(int index) : 지정된 인덱스 요소 반환
- get(int index) : 지정된 인덱스 요소 반환
- indexOf(Object o) : 지정된 객체와 같은 첫 번째 요소 인스 반환. 없으면 -1
- isEmpty() : 비어있으면 true
- remove(int index) : 지정된 인덱스 요소 삭제
- remove(Objcect o) : 지정된 객체와 같은 첫째 요소 벡터에서 삭제

- size() : 포함하는 요소의 개수 반환
- toArray() : 모든 요소를 포함하는 배열 반환

Vector 와 ArrayList 의 설명이나 주요 메소드 그리고 동작 결과를 보았을 때 큰 차이를 느낄 수 없을 정도로 비슷했다. 지금까지의 짧은 개발 경험이지만 생각해보면 ArrayList 를 사용하는 것은 굉장히 많이 봤는데 Vector 를 사용하는 경우는 거의 본적이 없는 것 같다. 둘은 왜 비슷한지.. 그리고 왜 ArrayList 가 많이 사용되어 지는지를 알아보도록 하자.

해당 링크의 Vector 에 관한 설명을 보도록 하자.

Vector 클래스

- JDK 1.0 부터 사용해왔다.
- ArrayList 클래스와 같은 동작을 수행하는 클래스이다.
- ArrayList 와 마찬가지로 List 인터페이스를 상속받는다.

Vector 클래스에서 사용할 수 있는 메소드는 ArrayList 클래스에서 사용할 수 있는 메소드와 거의 같다. 하지만 Vector 클래스는 기존 코드와의 호환성을 위해서만 남아있으므로, Vector 클래스 보다는 ArrayList 클래스를 사용하는 것이좋다.

기억해 둘 것

- Vector 와 ArrayList 는 List 인터페이스를 상속받는다.

- 둘은 같은 동작을 수행하는 클래스이다.

- 사용하는 메소드가 거의 비슷하다.

- Vector 는 기존 코드와의 호환성을 위해 남아있다.

- ArrayList 클래스를 사용하는 것이 좋다.

해당 링크의 Vector 와 ArrayList 주요 차이점을 설명을 보도록 하자.

1. 동기화(Synchronize)

- Vector 는 한번에 하나의 스레드만 접근 가능하다.
- ArrayList 는 동시에 여러 스레드가 작업할 수 있다.
    - 여러 스레드가 동시에 접근하드 경우 개발자가 명시적으로 동기화하는 코드를 추가해야 한다.

1. 스레드 안전(Thread Safe)

- 멀티 스레드 프로그래밍에서 여러 스레드가 동시에 접근이 이루어져도 프로그램 실행에 문제가 없음을 의미
- Vector 는 동기화 되어있기 때문에 한번에 하나의 스레드만 접근할 수 있으므로 Thread Safe 하다.
- ArrayList 는 동기화 되지 않았기 때문에 명시적으로 동기화 할 필요가 있다.

1. 성능

- ArrayList 는 동기화 되지 않았기 때문에 Vector 보다 빠르다.

1. 크기증가

- Vector 는 현재 배열의 크기의 100%가 증가
- ArrayList 는 현재 배열의 크기의 50% 증가

*멀티 스레드 환경이 아닌경우 ArrayList 사용이 바람직하다.

Vector 가 동기화 한다는 것은 복수의 스레드로부터 추가/삭제가 이루어져도 내부의 데이터 처리는 안전하게 한번에 하나의 스레드만 처리되도록 보장한다는 의미이다. 데이터 처리가 안정적으로 이루어지도록 보장하는 것이다.

단일 스레드의 경우 자동으로 동기화를 보장하는 것이 오히려 성능 저하를 일으킬

수 있기 때문에 동기화를 진행하지 않는 ArrayList 가 더 효율적인 성능을 보장한다고 할 수 있다.

LinkedList 란 무엇인가?

LinkedList 클래스

- java.util.LinkedList<E>
- ArrayList 의 단점을 극복하기 위해 고안되었다.
- 내부적으로 연결리스트를 이용하여 요소를 저장한다.
    - 배열은 저장된 요소가 순차적으로 저장된다.
    - 저장된 요소가 비 순차적으로 분포되며, 요소 사이를 링크로 연결하여 구성한다.
- List 인터페이스를 상속받기 때문에 ArrayList 의 메소드와 거의 같은 메소드를 사용할 수 있다.
- 단일 연결 리스트(singly linked list)
    - 요소의 저장과 삭제작업이 다음 요소를 가리키는 참조만 변경되어 빠르게 처리된다.
    - 현재 요소에서 이전 요소로 접근하기 매우 어렵다.
- 이중 연결 리스트(doubly linked list)
    - 이전 요소를 가리키는 참조도 가지고 있다.
    - 이전 요소로 접근하기 용이하다.

Vector - ArrayList - LinkedList 는 사용방법이나 동작 결과에 큰 차이는 없다. 내부적으로 저장하는 방법이 다른것이고 그것을 구분하여 설명할 줄 아는것이 중요한 것 같다.

정리 -

Vector 와 ArrayList 그리고 LinkedList 는 모두 같은 동작을 구현하는 클래스이며 List 인터페이스를 상속 받으므로 거의 비슷한 메소드를 사용한다. 사용방법과 동작 결과의 큰 차이는 없지만 내부 동작이 다르다.

Vector 와 ArrayList 의 주요 차이점중 하나는 동기화이다.
Vector 는 내부적으로 여러개의 스레드가 접근 할 때 데이터 안정성을 위해 한개의 스레드씩 순차적으로처리할 수 있도록 동기화 되어있다. 안정성을 보장하는 만큼 일을 많이 처리한다는 의미이고 메모리를 많이 사용한다고 볼 수 있다.
ArrayList 는 동기화되어있지 않기때문에 여러개의 스레드에서 접근 할 때 필요에 따라 동기화 처리를 해주어야 한다.

단일 스레드 작업시 동기화가 필요 없으므로 같은 동작을 하는 ArrayList 를 사용하는 것이 성능적인 면에서 용이하다.

현재 Vector 는 기존 코드와의 호환성을 위해 남아있다는 이야기가 있다.
Vector 에서 자동으로 해주는 동기화 보다 ArrayList 로 동기화를 직접 구현하는 것이 더 바람직하다는 의견도 있다.