**자바 Collections**

**-ArrayList**

ArrayList는 기본적으로 배열을 사용한다. 일반 배열은 메모리를 할당할 때 크기를 지정해주어야 하지만, ArrayList는 크기를 지정하지 않고 동적으로 값을 삽입하고 삭제할 수 있다.

조회: ArrayList는 각 데이터의 index를 가지고 있고 무작위 접근이 가능하기 때문에, 해당 index의 데이터를 한번에 가져올 수 있다.

데이터 삽입과 삭제: ArrayList는 데이터 삽입 시 사이즈를 늘려주는 연산이 추가되어야 하고,

삭제 시에는 순차적인 인덱스 구조로 인해 삭제된 빈 인덱스를 채워야 하기 때문에 연산이 추가되어야 한다.

예를들어 5개의 데이터가 있을 때 맨 앞의 2를 삭제했다면 나머지 뒤의 4개를 앞으로 한 칸씩 이동해야한다. 따라서 삽입과 삭제가 많다면 ArrayList는 비효율적이다.

ArrayList 삽입속도를 빠르게 하는법 : List의 size를 예측하여 size를 미리 설정해두면 속도가 더 빨라진다.

이유: 미리 size를 설정해두면 size를 늘려주는 연산을 안하고 데이터 삽입만 하면 되기 때문에 삽입 속도가 더 빨라진다.

**-LinkedList**

LinkedList는 내부적으로 양방향의 연결리스트로 구성되어 있어 참조하려는 원소에 따라 처음부터 정방향 또는 역순으로 순회가능.

조회: LinkedList는 순차적 접근이기 때문에 검색의 속도가 느리다. 그래서 ListIterator를 같이 사용하면 검색 속도가 빨라진다.

데이터 삽입과 삭제: LinkedList는 데이터를 추가/삭제시 가리키고 있는 주소값만 변경해주면 되기 때문에 ArrayList에 비해 상당히 효율적이다.

예를들어 2번째 값을 삭제하면 1번째 노드가 3번째 노드를 가리키게 하기만 하면 된다.

**-Vector**

Java1.0부터 이어져온 리스트 객체로 배열 형태를 이룬다.

데이터 추가 시 공간을 두 배로 확보하기 때문에 메모리를 많이 잡아먹는다.

동기화가 항상 이루어지기 때문에 하나의 스레드가 하나의 자원을 이용하는 경우 성능이 저하된다.

**HashMap**

-저장된 키 값은 꺼낼 때 무작위로 꺼내 키의 HashCode 값에 따라 데이터를 저장하고 키에 따라 직접 가져올 수 있어 빠른 접근 속도를 가지고 있다.

해시함수를 이용하기에 삽입, 삭제, 검색이 빠르다.

**LinkedHashMap**

-LinkedHashMap은 순서를 유지하기 위해 이중 연결 리스트를 사용한다.

-HashMap의 모든 기능을 그대로 사용할 수 있고, 순서를 유지하기 때문에 메모리 사용량이 HashMap보다 높다.

**Iterator**

hasNext() : 다음요소가 있는지 판단

next(): 다음요소를 가져옴

remove(): 가져온 요소를 삭제

**Iterator 사용이유**

LinkedList를 통해 예를 들면,

get(0)부터 get(100)까지를 수행하게 되는데 이는 0부터 100까지 총 101번의 요소를 조회하는게 아니라 get(int index) 메서드는 시작 주소부터 index 만큼 요소들을 밟아가며 조회하는 메서드이기 때문이다.

만약 5번째 값을 조회한다면 처음 시작주소부터 시작하여 다음주소를 타고 반복하여 총 5번을 반복해야한다.

get 메서드가 실행되며 i 값이 증가할 때마다 메모리적으로 조회해야 하는 요소는 1번, 2번, 3번, 4번... 101번까지 증가하는 것이다. 총 5151번을 조회해야 한다.

이에반해 Iterator는 1부터 101번째까지의 요소에 대해 내부적으로 객체로 생성 후 순차적으로 조회한다. 처음 주소로 돌아갈 필요가 없기 때문에 next 메서드를 통해 조회 시 요소의 개수인 101번만 조회를 하게 된다.

속도면에서 빠르지 않을까 생각했지만, Iterator를 구현하기 위해 객체를 생성하는 부분에 시간이 더 걸린다고 한다.