리스트 (List) 란?

배열과 같이 객체를 일렬로 늘어놓은 구조를 가지고 있다.
객체를 **인덱스(index)**로 관리하기 때문에 **객체를 저장하면 자동으로 인덱스가 부여**되고, 인덱스로 객체를 검색, 추가, 삭제할 수 있는 등의 기능을 제공한다.

List 인터페이스를 구현한 클래스로는 ArrayList, Vector, LinkedList, Stack 등이 있다. 가장 많이 사용되는 것은 ArrayList 와 LinkedList 이다.

리스트의 특징

- 순서가 있고 중복을 허용한다.
- 인덱스로 관리하기 때문에 인덱스로 접근이 가능하다.
- 크기가 가변적이다.

List 인터페이스

리스트 인터페이스에서 공통적으로 사용 가능한 메서드

add(int index, Object element)

주어진 인덱스에 객체를 추가한다.

addAll(int index, Collection c)

주어진 인덱스에 컬렉션을 추가한다. **boolean** 타입을 리턴한다.

set(int index, Object element)

주어진 인덱스에 객체를 저장. **Object** 타입을 리턴한다.

indexOf(Object o) / lastIndex(Object o)

순방향 / 역방향으로 탐색하여 주어진 객체의 위치를
반환한다. **int** 타입을 리턴한다.

listIterator() / listIterator(int index)

List 의 객체를 탐색할 수 있는 ListIterator 반환 / 주어진 index 부터
탐색할 수 있는 **ListIterator** 타입을 리턴한다.

subList(int fromIndex, int toIndex)

fromIndex 부터 toIndex 에 있는 객체를 반환한다. **List** 타입을 리턴한다.

remove(int index)

주어진 인덱스에 저장된 객체를 삭제하고 삭제된 객체를
반환한다. **Object** 타입을 리턴한다.

remove(Object o)

주어진 객체를 삭제한다. **boolean** 타입을 리턴한다.

sort(Comparator c)

주어진 비교자(comparator)로 List 를 정렬한다.

List 예제

```
// list 생성
List<String> list = new ArrayList<>();

// 일반적으로 리스트를 만든 뒤, add 메서드를 사용하여 객체를 하나씩 저장한다.

// 객체 추가
list.add("a");
list.add("b");
list.add("c");

// 조회
for(String ele : list) {
	System.out.println(ele);
}

/* 출력
a
b
c
*/
```

```
// 선언과 동시에 값을 할당하려면 asList 사용
List<String> list = Arrays.asList(new
String[]{"a","b","c"});

// 조회
for(String ele : list) {
	System.out.println(ele);
}

/* 출력
a
b
c
*/
```

ArrayList 란?

List 인터페이스를 구현한 클래스로 컬렉션 프레임워크에서 가장 많이 사용된다. 기능적으로는 Vector 와 동일하지만 Vector 를 개선한 것이므로 Vector 보다 많이 사용된다.

객체가 **인덱스로 관리**된다는 점에서 배열과 유사하다. 그러나 배열은 생성될 때 크기가 고정되며, 크기를 변경할 수 없지만 ArrayList 는 저장 용량을 초과하여 객체들이 추가되면, **자동으로 저장용량이 늘어나게 된다**. 또한 데이터가 연속적으로 존재하여 데이터의 순서가 **유지**된다,

```
ArrayList<타입 매개변수> 객체명 = new ArrayList<타입
매개변수>(초기 저장 용량);

ArrayList<String> example = new ArrayList<String>();
// String 타입의 객체를 저장하는 ArrayList 생성
// 초기 용량이 인자로 전달되지 않으면 기본적으로 10 으로 지정된다.
```

ArrayList 에 객체를 추가하게 되면 인덱스(index) 0 부터 차례대로 저장된다.

특정 인덱스의 객체를 제거하면, 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1 씩 당겨진다.

ArrayList 에 객체를 순차적으로 저장할 때에는 데이터를 이동하지 않아도 되므로, 작업 속도가 빠르지만, **중간에 위치한 객체를 추가, 삭제할 때에는 데이터 이동이 많이 일어나므로 속도가 저하된다.**

반면 인덱스가 n 인 요소인 데이터에 빠르게 접근이 가능하기 때문에 **검색(읽기) 측면**에서는 유리하다.

ArrayList 예제

```
// ArrayList 를 생성하여 list 에 할당
ArrayList<String> list = new ArrayList<String>();

// String 타입의 데이터를 ArrayList 에 추가
list.add("Java");
list.add("egg");
list.add("tree");

// 저장된 총 객체 수 얻기
int size = list.size();

// 0 번 인덱스의 객체 얻기
String skill = list.get(0);

// 저장된 총 객체 수 만큼 조회
for(int i = 0; i < list.size(); i++) {
```

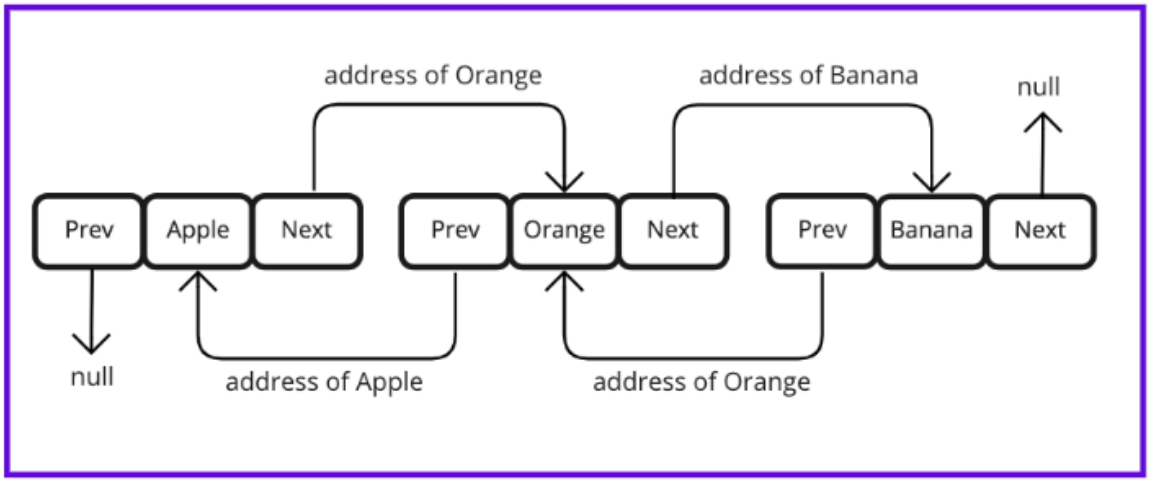
```
        String str = list.get(i);
    System.out.println(i + ":" + str);
}

// for - each 문으로 순회
for(String str : list) {
        System.out.println(str);
}

// 0 번 인덱스 객체 삭제
list.remove(0);
```

LinkedList 란?

ArrayList 와 배열은 모든 데이터가 연속적으로 존재하지만,
LinkedList 에는 **불연속적**으로 존재하며, 이 데이터는 서로 연결되어 있다.
LinkedList 컬렉션은 데이터를 **효율적으로 추가, 삭제, 변경하기 위해 사용한다.**



LinkedList 의 각 요소(node)들은 자신과 연결된 이전 및 다음 요소의 주소값과 데이터로 구성되어 있다.

LinkedList 에서 데이터를 삭제하려면, 삭제하고자 하는 요소의 이전 요소가 삭제하고자 하는 요소의 다음 요소를 참조하도록 변경한다. 링크를 끊어주는 방식이다. **배열처럼 데이터를 이동하기 위해 복사할 필요가 없기 때문에 처리 속도가 훨씬 빠르다.** 데이터를 추가할 때에도 마찬가지이다.

※ 위 그림의 Apple 과 Orange 사이에 Mango 라는 데이터를 추가하는 상황

- Mango 객체가 생성된다.
- Apple 의 Next 에 Mango 의 주소값이 저장된다. 이때, Mango 의 Prev 에 Apple 의 주소값이 저장된다.
- Mango 의 Next 에 Orange 의 주소값이 저장된다. 이때, Orange 의 Prev 에 Mango 의 주소값이 저장된다

이처럼 LinkedList 의 중간에 데이터를 추가하면 Next 와 Prev 에 저장되어 있는 주소값만 변경해주면 되므로, 각 요소들을 ArrayList 처럼 뒤로 밀어내지 않아도 된다. 마찬가지로, 삭제한 경우에도 삭제한 데이터의 뒤에 위치하는 요소들을 앞으로 당기지 않아도 된다.

LinkedList 예제

```
// Linked List 를 생성하여 list 에 할당
ArrayList<String> list = new LinkedList<>();

// String 타입의 데이터를 LinkedList 에 추가
list.add("Java");
list.add("egg");
list.add("tree");

// 저장된 총 객체 수 얻기
int size = list.size();
```

```
// 0번 인덱스의 객체 얻기
String skill = list.get(0);

// 저장된 총 객체 수 만큼 조회
for(int i = 0; i < list.size(); i++){
    String str = list.get(i);
    System.out.println(i + ":" + str);
}

// for-each 문으로 순회
for (String str: list) {
    System.out.println(str);
}

// 0번 인덱스 객체 삭제
list.remove(0);
```

ArrayList VS LinkedList

ArrayList 는 다음과 같은 상황에 **강점**을 지닌다.

- 데이터를 순차적으로 추가하거나 삭제하는 경우
  - 순차적으로 추가한다는 것은 0 번 인덱스에서부터 데이터를 추가하는 것을 의미한다.
  - 순차적으로 삭제한다는 것은 마지막 인덱스에서부터 데이터를 삭제하는 것을 의미한다.
- 데이터를 읽어들이는 경우
  - 인덱스를 통해 바로 데이터에 접근할 수 있으므로 검색이 빠르다.

다음과 같은 상황에서 **효율적이지 못하다.**

- 중간에 데이터를 추가하거나, 삭제하는 경우
  - 추가 또는 삭제 시, 해당 데이터의 뒤에 위치한 값들을 뒤로 밀어주거나 앞으로 당겨주어야 한다.

LinkedList 는 다음과 같은 상황에 강점을 가진다.

- 중간에 위치하는 데이터를 추가하거나, 삭제하는 경우
- 데이터를 중간에 추가, 삭제하는 경우 Prev 와 Next 의 주소값만 변경하면 되므로, 다른 요소들을 이동시킬 필요가 없다.

따라서 데이터를 중간에 추가, 삭제하는 경우, LinkedList 는 ArrayList 보다 빠른 속도를 보여준다.

데이터의 잦은 변경이 예상될 땐 LinkedList, 데이터의 개수가 변하지 않는다면 ArrayList 를 사용하는 것이 좋다.