[읽을거리] 2. 리스트

시간복잡도

리스트의 종류 별 시간복잡도는 다음과 같습니다.

	add()	remove()	get()	contains()
ArrayList	O(1)	O(n)	O(1)	O(n)
LinkedList	O(1)	O(1)	O(n)	O(n)
CopyOnWriteArrayList	O(n)	O(n)	O(1)	O(n)

ArrayList에서 add는 O(1) 이지만, capacity를 넘어서 배열복사가 일어날 경우 O(n)이 됩니다.

LinkedList 에서 특정 요소를 삭제하는 경우는 O(1) 이지만, 요소를 찾아서 삭제해야 할 경우에는 O(n) 이 됩니다.

간편 리스트 만들기

고정 값으로 List 만들기

리스트에 입력될 값들을 사용해서 리스트를 만들 수 있습니다.

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
```

Array로 List 만들기

배열로부터 리스트를 만들 수 있습니다.

```
Integer[] arr = new Integer[]{1, 2, 3, 4, 5};
List<Integer> list = Arrays.asList(arr);
```

하지만 리스트의 데이터타입과 다르면 만들 수 없다는 것에 주의하세요.

```
int[] arr = new int[]{1, 2, 3, 4, 5};
List<Integer> list = Arrays.asList(arr);
//error: incompatible types: inference variable T has incompatible bounds
```

List로부터 List만들기

기존의 리스트를 사용해서 동일한 값을 가진 새로운 리스트를 만들 수 있습니다.

```
List<Integer> list1 = Arrays.asList(1,2,3,4,5);
List<Integer> list2 = new LinkedList<>(list1);
```

⚠️ Fixed-size List

`Arrays.asList` 를 사용해서 간편하게 만들어낸 마치 배열처럼 고정된 크기의 리스트를 만들어 냅니다.

이 리스트는 값을 변경할 수는 있지만, 크기에 영향을 주는 추가/삭제 작업은 할 수 없습니다.

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
list.set(0, 6); // 가능
list.add(6);    // error!
list.remove(0); // error!
```

List를 Array로 변경하기

코딩테스트 문제를 풀다보면 반환값으로 array를 요구하는 경우가 많습니다. 풀이과정에서 List를 사용할 경우 반환값을 위해서 array로 변환할 필요가 있습니다.

List를 array로 만드는 방법들을 알아봅시다.

array에 복사하기

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
int[] arr = new int[list.size()];
for (int i = 0; i < list.size(); i++) {
    arr[i] = list.get(i);
}
```

toArray를 사용하기

```
List<String> list = Arrays.asList("A", "B", "C", "D", "E");
String[] arr = list.toArray(String[]::new);
```

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
Integer[] arr = list.toArray(Integer[]::new);
```

toArray를 사용할 때는 만들려고 하는 배열타입의 생성방법을 지정해 줘야 합니다.

하지만, 최종 배열타입이 다를경우 사용할 수 없습니다. (auto unboxing 이 적용되지 않습니다.)

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
int[] arr = list.toArray(int[]::new);
// error: no suitable method found for toArray(int[]::new)
```

stream 사용하기

```
List<String> list = Arrays.asList("A", "B", "C", "D", "E");
String[] arr = list.stream().toArray(String[]::new);
```

stream을 통해서 toArray 를 사용하는 방법은 List의 toArray를 사용하는 것과 같습니다.

하지만, primitive 로 unboxing 이 필요한 경우 편리하게 사용될 수 있습니다.

```
List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);
int[] arr = list.stream().mapToInt(Integer::intValue).toArray();
```

⚠️ 리스트 요소 삭제하기

리스트 사용 시 loop를 돌면서 삭제하게 되는 경우 주의할 것들이 있습니다.

```
List<Integer> list = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
for (int i = 0; i < list.size(); i++) {
    list.remove(i);
}
System.out.println(list);  // [2, 4]
```

Arrays.asList 로 생성하면 fixed-size list 가 되므로 LinkedList 를 새로 생성해서 삭제가 가능한 리스트로 만들어 냈습니다.

for-loop 에서 i 가 증가하면서 i 값을 index로 하는 요소를 삭제하지만, 삭제할 때마다 size 가 줄어들어 index 가 고정되지 않기 때문에 예상대로 동작하지 않게 됩니다.

```
List<Integer> list = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
for (int i = 0; i < list.size(); i++) {
    list.remove(0);
}
System.out.println(list);  // [2, 4]
```

항상 0번째의 값을 삭제하게 해도 마찬가지 입니다. i 가 증가하면서 size도 감소하기 때문에 어느정도 진행하고 나면 i의 값이 size 이상이 되는 경우 for-loop 가 멈추게 됩니다.

```
List<Integer> list = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
int size = list.size();
for (int i = 0; i < size; i++) {
    list.remove(0);
}
System.out.println(list);  // []
```

초기의 size 값을 캡춰해서 for-loop 의 반복회수를 고정시켜서 원하는 결과를 얻을 수 있습니다.

Immutable List

리스트를 함수의 인자로 전달할 때 call by reference로 전달되게 됩니다. 인자를 final로 선언한다 해도 리스트 인스턴스의 내용을 변경할 수 있게 됩니다.

```
void main() {
    List<Integer> list = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
    updateList(list);
    System.out.println(list);  // [2, 4, 6, 8, 10]
}

void updateList(final List<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        list.set(i, list.get(i) * 2);
    }
}
```

리스트를 updateList 에 인자로 전달했을 뿐인데, 그 내용이 변경되었습니다. 예기치 못한 이런 상황을 side-effect 라고 합니다.

이런 side-effect를 방지하기 위해서 함수에 리스트를 전달할 때는 원본 리스트가 훼손되지 못하는 변경되지 못하는 리스트의 형태로 를 만들 필요가 있습니다.

```
void main() {
    List<Integer> list = new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
    updateList(Collections.unmodifiableList(list));
    System.out.println(list);
}

void updateList(final List<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        list.set(i, list.get(i) * 2);  // ERROR!!
    }
}
```

`Collections.unmodifiableList` 를 사용하면 리스트의 내용 자체를 변경하지 못하게 만들 수 있습니다.