## List 인터페이스

데이터를 입력한 순서대로 유지하는 선형 자료구조이다. 데이터는 고유한 번호(인덱스)를 가지고 순서대로 나열된다.

주요 특징

1. 순서 유지 : 데이터가 입력된 순서대로 저장된다.
2. 중복 허용 : 같은 값을 가진 데이터를 여러개 저장할 수 있다.
3. 인덱스 기반 접근 : 각 데이터에 부여된 인덱스를 이용하여 특정 데이터에 접근한다.

List 인터페이스로 구현되는 클래스 ArrayList, LinkedList

List 인터페이스의 주요 메소드
`add(e)` : List의 끝에 요소를 추가한다.
`get(index)` : 지정된 인덱스의 요소를 반환한다.
`remove(index)` : 지정된 인덱스의 요소를 삭제한다.
`size()` : List의 크기(요소의 개수)를 반환한다.

## Queue 인터페이스

먼저 들어온 데이터가 먼저 나가는 선입선출(FIFO) 방식으로 데이터를 관리하는 선형 자료구조이다.

주요 특징

1. 순서 유지 : 데이터가 입력된 순서대로 처리된다.
2. 한쪽 끝에서 추가, 다른 쪽 끝에서 삭제 : 새로운 데이터는 뒤에 추가되고 가장 오래된 데이터는 앞에서 삭제된다.

Queue 인터페이스로 구현되는 클래스 LinkedList, ArrayDeque

Queue 인터페이스의 주요 메소드
`enqueue(e)` : 큐의 뒤에 데이터를 추가된다.
`dequeue()` : 큐의 맨 앞에 있는 요소를 제거하고 반환한다.
`peek()` : 큐의 맨 앞에 있는 요소를 확인한다.

---

## Deque 인터페이스

양쪽 끝에서 데이터를 추가하거나 삭제할 수 있는 선형 자료구조를 나타내는 인터페이스이다. 스택과 큐 기능이 모두 제공된다.

주요 특징

1. 양방향 접근 : 데이터를 앞쪽 또는 뒤쪽에서 삽입하고 삭제할 수 있다.
2. 유연성 : 다양한 자료구조를 구현하는 데 사용될 수 있다. 스택, 큐, 데크 자체로 사용된다.
3. 효율성 : 일반적으로 배열이나 연결 리스트를 기반으로 구현되며 삽입 및 삭제 연산의 시간 복잡도는 O(1)이다.

Deque 인터페이스로 구현되는 클래스 LinkedList, ArrayDeque

Deque 인터페이스의 주요 메소드
`addFirst(e)` : 맨 앞에 데이터를 추가한다.
`addLast(e)` : 맨 끝에 데이터를 추가한다.
`removeFirst()` : 맨 앞에 데이터를 제거하고 반환한다.
`removeLast()` : 맨 끝에 데이터를 제거하고 반환한다.
`getFirst()` : 맨 앞에 데이터를 조회한다.
`getLast()` : 맨 끝에 데이터를 조회한다.

---

## ArrayList 클래스

배열을 기반으로 구현된 동적 배열이다. 랜덤 접근이 빠르지만 중간에 데이터를 삽입,삭제할 때는 배열의 크기를 조절해야 하므로 성능이 저하될 수 있다. 연속적인 메모리 공간을 사용하여 캐시 효율성이 좋다(ArrayList를 사용할 때 CPU가 데이터에 더 빠르게 접근할 수 있다는 것을 의미). ArrayList는 데이터의 크기가 미리 예측 가능한 경우 순차적인 접근이 주된 경우 사용하는 것이 좋다.

### LinkedList 클래스

노드를 기반으로 객체를 연결하여 데이터를 저장한다. 중간에 데이터를 삽입,삭제하는 것이 빠르지만 특정 인덱스에 접근하려면 처음부터 순차적으로 찾아가야 하므로 랜덤 접근 성능이 떨어진다. 메모리 공간이 비연속적으로 분리되어 있어 캐시 효율성도 좋지 않다. LinkedList는 데이터를 자주 삽입,삭제해야 하는 경우 데이터의 크기가 자주 변하는 경우 스택이나 큐 처럼 양쪽 끝에서 데이터를 추가하거나 삭제해야하는 경우 사용하는 것이 좋다.

### ArrayDeque 클래스

양쪽 끝에서 삽입,삭제가 매우 빠르며 스택과 큐로 사용하기에 적합하다. LinkedList에 비해 캐시 효율성이 좋다. ArrayDeque는 양쪽 끝에서 데이터를 자주 추가하거나 삭제해야하는 경우 사용하는 것이 좋다.

### 정리표

| 자료구조 | 구현 방식 | 장점 | 단점 | 주요 사용 시나리오 |
|---|---|---|---|---|
| ArrayList | 배열 | 랜덤 접근 빠름, 캐시 효율성 좋음 | 중간 삽입/삭제 느림 | 데이터 자주 읽고 수정, 크기 미리 예측 가능 |
| LinkedList | 연결 리스트 | 중간 삽입/삭제 빠름 | 랜덤 접근 느림, 캐시 효율성 낮음 | 데이터 자주 삽입/삭제, 크기 자주 변함, 스택/큐 |
| ArrayDeque | 배열 | 양쪽 끝 삽입/삭제 빠름, 캐시 효율성 좋음 | 중간 삽입/삭제 느림 | 스택/큐, 양쪽 끝에서 데이터 자주 추가/삭제 |

### 선택 가이드

주요 연산 : 랜덤 접근, 중간 삽입삭제, 양쪽 끝 삽입/삭제 등
데이터 크기 : 미리 예측 가능한지, 자주 변하는지
메모리 사용 : 연속적인 메모리 공간이 필요한지
캐시 효율성 : 중요한 요소인지