ArrayList, LinkedList, ArrayDeque 상세조사

List 인터페이스
정렬된 데이터의 집합을 나타내는 인터페이스이다. 중복된 데이터를 허용하며 인덱스를 통해 접근할 수 있다.

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.

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

ArrayDeque 클래스
배열을 기반으로 구현된 Deque이다. 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	이중 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		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