팀 과제 보고서

1. 구성원 및 역할

\* 데이터 산출물 담당 : 제품기술팀 신현영 사원

\* 코드 관리 : 제품기술팀 김다현 사원

\* 문서화 담당 : 제품기술팀 김지혜 사원

\* 보고 : 제품품질팀 이상윤 사원

2. 방향

모두 공통적으로 성능을 측정한 HashMap, LinkedHashMap, ArrayList, LinkedList를 기준으로 중간 성능의 자료구조에 대해 정의 내렸다. 결과물을 취합한 결과, 위의 자료구조 중 LinkedList가 데이터를 입력할 때 가장 빠르고 탐색할 때 가장 느린 것으로 확인되었다. 이에 LinkedList의 구조를 변형시켜 입력시간을 늘리고 탐색시간을 줄이는 방향으로 중간 성능의 자료구조를 만들기로 결정하였다.

자료구조의 성능을 파악하기 위한 방법은 다음과 같다.

1. 입력과 탐색은 add(), get()으로 제한한다.

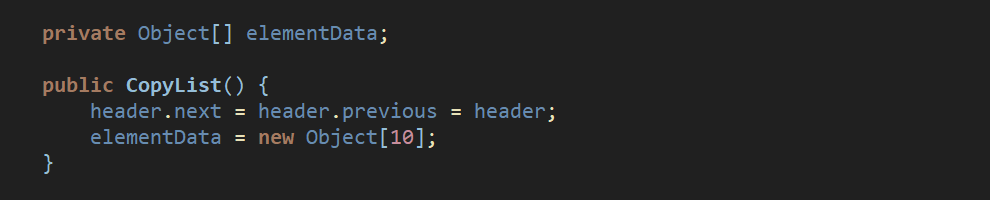
2. 데이터를 100,000건 입력하고 탐색한다.

3. Integer형으로 테스트한다.

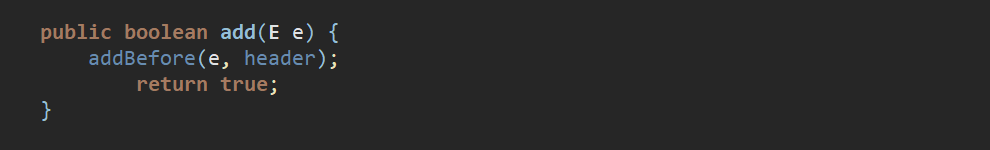
3. 진행사항

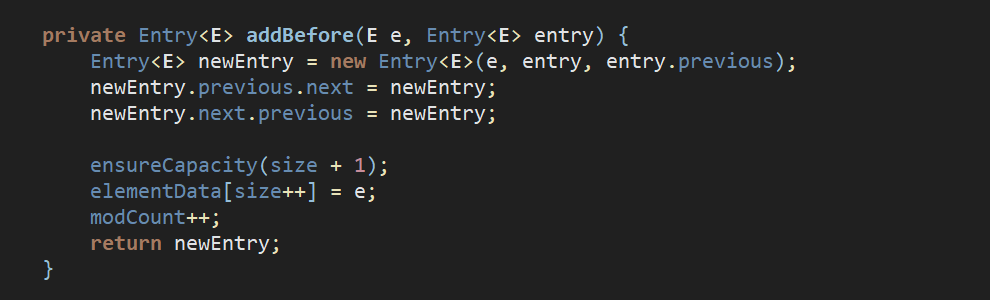
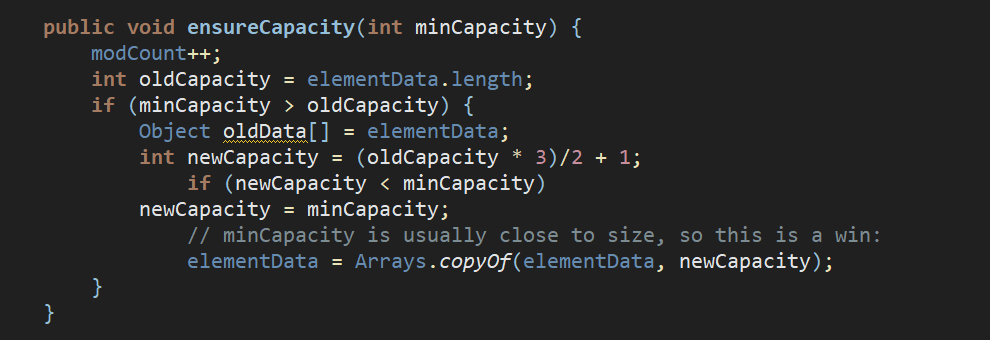
1) 배열에 값을 이중으로 저장

ArrayList의 구조를 참고하여 elementData에 값을 별도로 저장하였다. 배열의 초기 크기 값은 ArrayList와 마찬가지로 10으로 설정하였다.

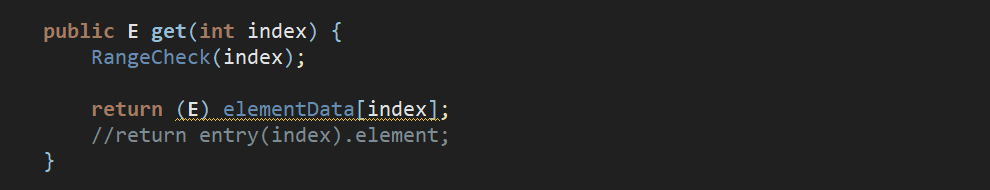


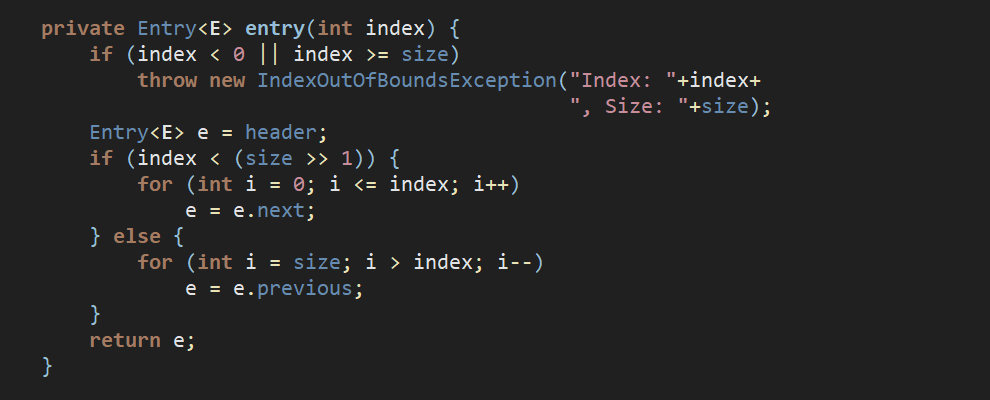
데이터를 입력할 때에는 ArrayList의 ensureCapacity()를 사용하여 elementData의 배열 길이를 늘렸으며, node간 연결 방식은 기존의 LinkedList와 동일하다.

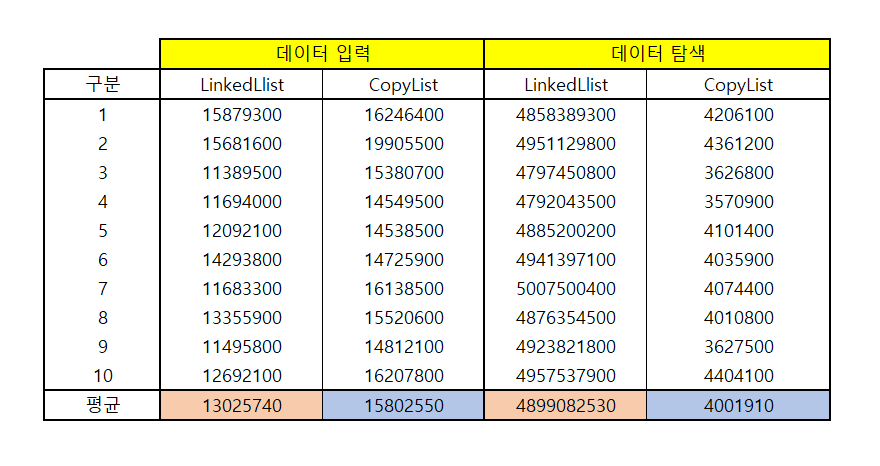


****

이 구조에서 데이터에 접근하는 방식으로 index를 통한 접근과 노드를 통한 접근이 있다. 둘 방식 중 index를 통해 접근하는 방식이 더 짧은 탐색시간을 갖는 것으로 확인되었다.

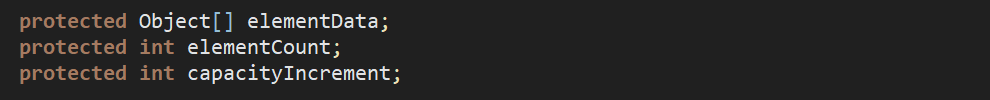


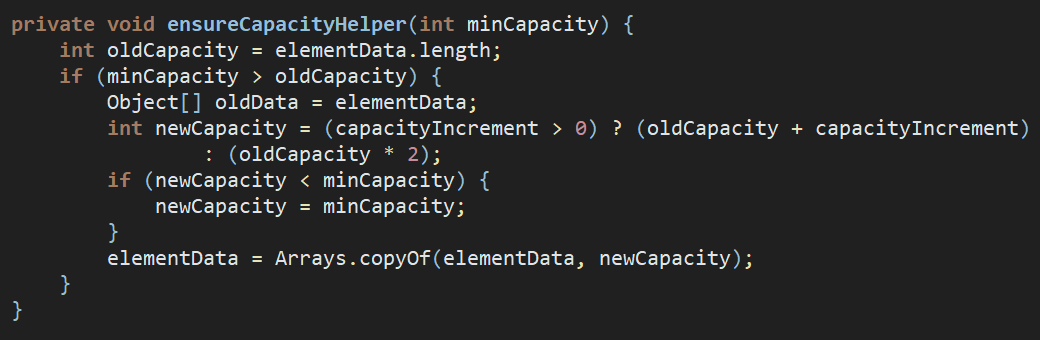


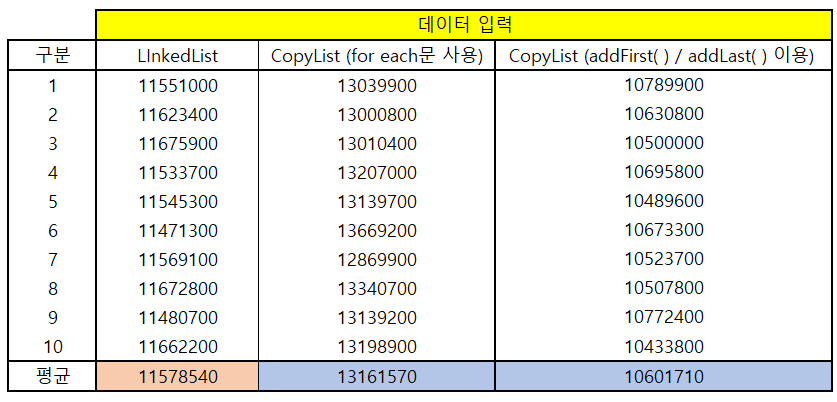


2) Vector 참고

위에서 ArrayList의 ensureCapacity()를 사용한 것과 달리 ensureCapacityHelper() 메소드를 따로 만들어 elementData의 길이를 늘리는 방식에 변형을 주었다.

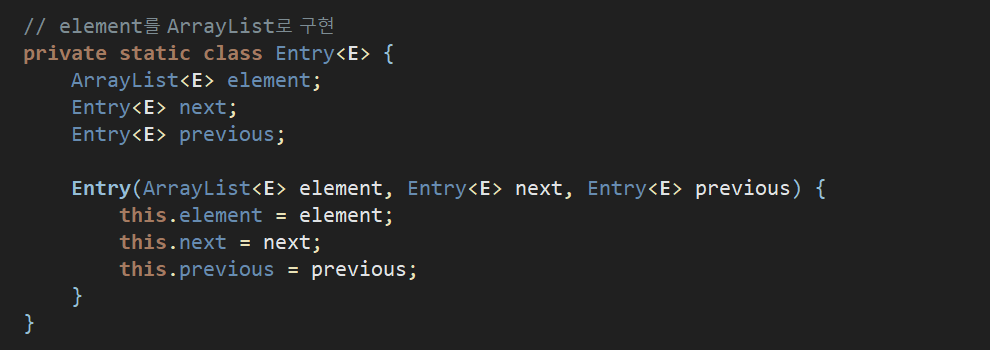




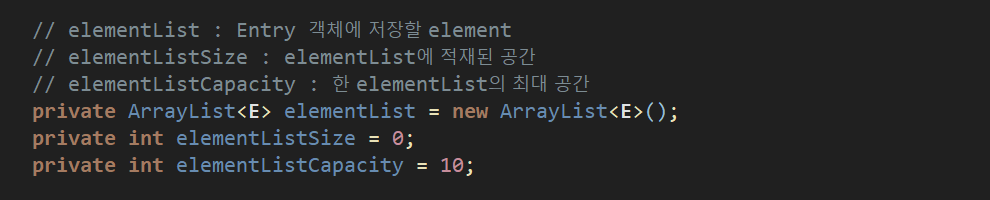


3) Entry의 element를 ArrayList로 변경

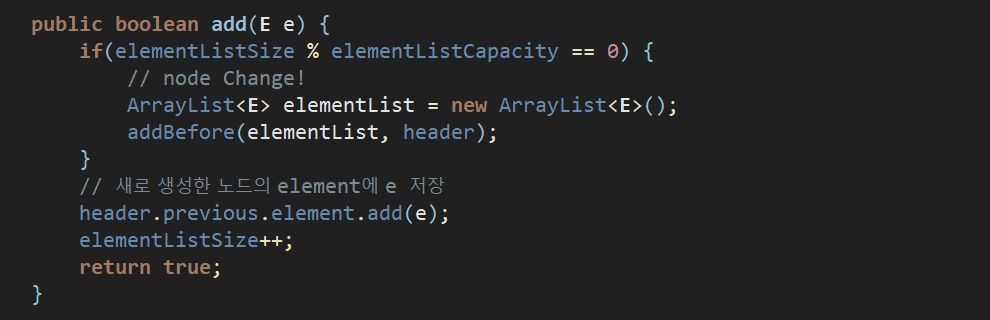
LinkedList의 Entry는 element로 데이터 하나를 넣는 방식이기 때문에 탐색시간이 오래 걸리는 것을 감안하여 데이터 하나 대신 ArrayList를 넣는 방식으로 변형시켰다.



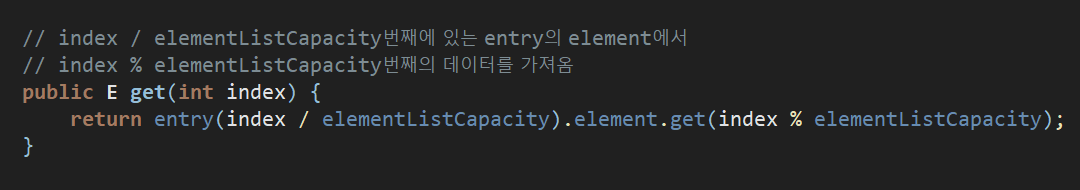
각 Entry의 elementList는 elementListCapacity 만큼의 길이를 갖는다.



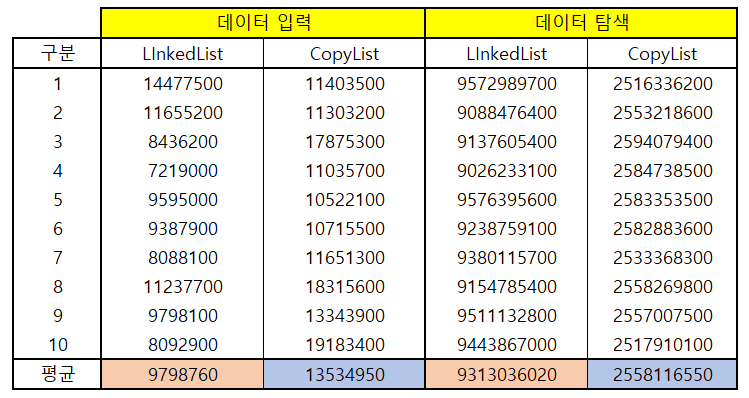
elementListSize가 elementListCapacity의 배수가 될 때마다 새로운 Entry를 생성하고, 그 노드에 ArrayList의 add()연산자를 사용하여 값을 저장한다.



기존에 indexed Entry를 찾는 방식을 사용하되, index를 elementList의 크기만큼 나눠 Entry 탐색횟수를 줄일 수 있었다.



elementList의 최대공간을 10으로 했을 때 결과는 다음과 같다.



elementList의 최대공간을 100으로 했을 때 결과는 다음과 같다.

