**자료구조 실습 보고서**

[제06주] 리스트의 성능 측정

**2021년 04월 11일**

**201702039 오명주**

1. **프로그램 설명서**
2. **프로그램의 전체 설계 구조**

* MVC (Model – View – Controller) 구조

Model : 프로그램이 “무엇”을 할 것인지 정의. 사용자의 요청에 맞는 알고리즘을 처리하고 DB와 상호작용하여 결과물을 산출하고 Controller에게 전달.

View : 화면에 무엇인가를 “보여주기 위한” 역할. 최종 사용자에게 “무엇”을 화면으로 보여줌.

Controller : 모델이 “어떻게” 처리할 지 알려주는 역할. 사용자로부터 입력을 받고 중개인 역할. Model과 View는 서로 직접 주고받을 수 없음. Controller을 통해 이야기함.

* 리스트 성능 비교 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

* Experiment : 실험 자체를 위한 추상 자료형. 출력될 실험변수와 실험방법을 구성
* MeasuredResult : 실험 결과를 받고 합과 최대값을 구함.
* SortedArrayList : ArrayList로 구현된 정렬된 리스트
* UnsortedArrayList : ArrayList로 구현된 정렬되지 않은 리스트
* SortedLinkedList : LinkedList로 구현된 정렬된 리스트
* UnsortedLinkedList : LinkedList로 구현된 정렬되지 않은 리스트
* Coin : 코인의 변수, 속성을 구성. 비교함수도 존재
* LinkedNode : Node의 변수, 속성을 구성

View :

* AppView : 프로그램의 입/출력을 담당한다.

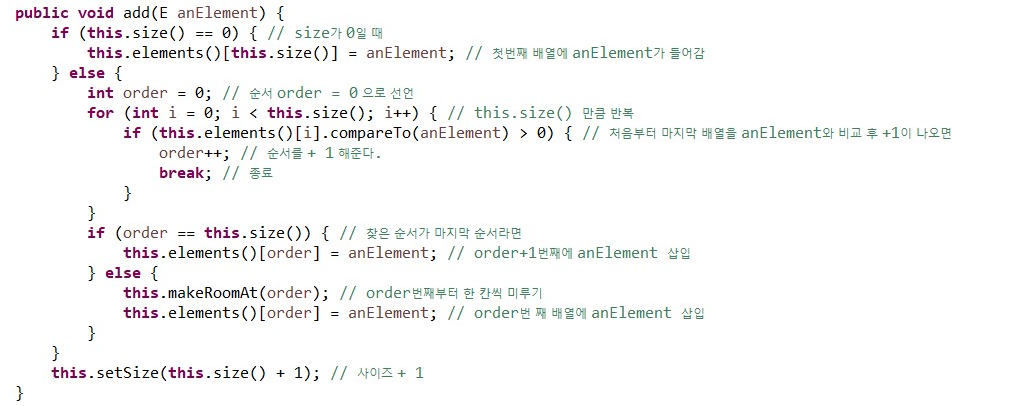
Controller :

* AppController : Model을 통해 결과물을 AppView를 통해 출력한다.

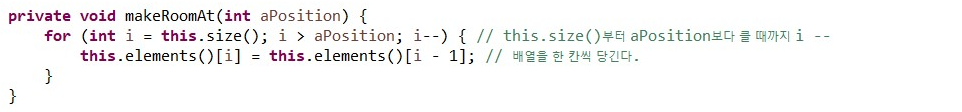
1. **함수 설명서**

* 주요 알고리즘

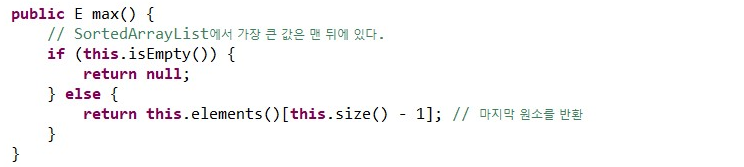
1. SortedArrayList



**Add** : Size를 확인하여 0이라면 첫번째 배열에 원소를 넣어준다. compareTo 함수를 이용하여 객체 값을 비교하여 삽입할 위치를 order에 저장한다. 만약 찾은 순서가 마지막이라면 그냥 넣어주고 중간 인덱스라면 makeRoomAt 함수를 이용하여 한 칸 씩 미루어 삽입한다.



주어진 인덱스 이후 한 칸 씩 뒤로 값을 밀어주는 함수이다.

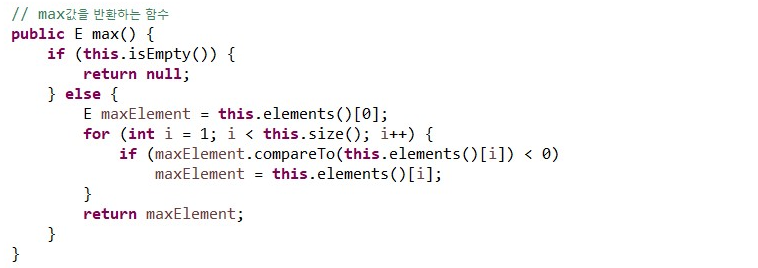


**Max** : 오름차순으로 정렬된 리스트이기 때문에 배열의 마지막 원소를 반환한다.

1. UnsortedArrayList



**Add** : 정열 되지 않는 리스트에 대한 구현이기 때문에 가장 마지막 순서에 원소를 삽입한다.

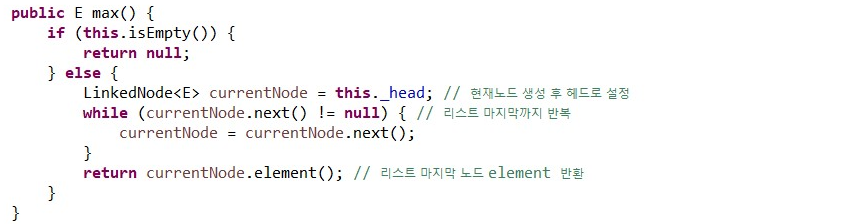


**Max** : 리스트가 비어 있다면 null을 반환. 배열의 크기만큼 max값 찾는 반복문을 이용하여 만약 max 변수에 저장된 값과 현재 배열 원소를 비교하여 더 큰 값을 max에 저장한다. 이 때도 객체의 비교는 compareTo 함수를 활용한다.

1. SortedLinkedList

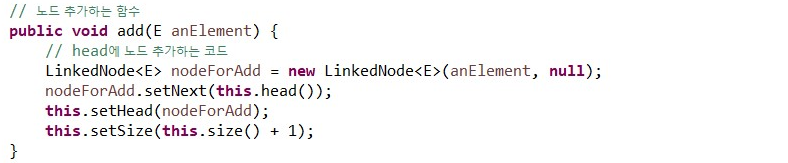


**Add** : 만약 리스트가 가득 찼다면 false를 반환. 삽입할 element를 가진 객체를 생성한다. 그리고 리스트가 비었다면 head에 삽입하고 그렇지 않다면 compareTo 함수를 이용하여 삽입할 위치를 검색한다. 위치를 검색했을 때 head라면 head로 삽입하고 중간 위치라면 previous의 next를 새로운 노드로 설정하여 삽입하여 준다.



**Max** : 리스트가 비어 있다면 null을 반환. 정렬 되어있는 LinkedList 이므로 while문을 이용하여 마지막 노드까지 걸어 나가 마지막 노드의 element를 반환한다.

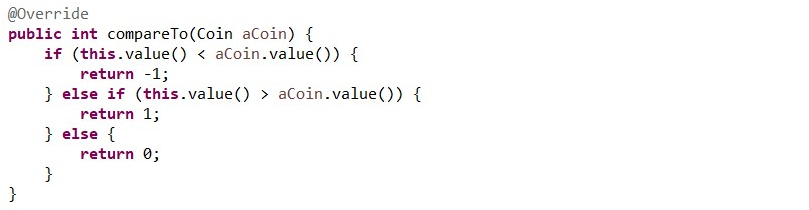
1. UnsortedLinkedList



**Add** : 정렬 되지 않은 리스트에 대한 구현이기 때문에 해당 element를 갖는 객체를 생성하여 head에 넣어준다. Size는 1 증가해준다.



**Max** : 리스트가 비어 있다면 null을 반환. 현재 노드와 최대값을 저장할 노드 객체를 생성해주고 마지막 노드까지 반복하는 while문을 이용하여 max노드와 현재 노드를 비교해준다. 더 큰 값을 max노드에 저장하고 걸어 나가 반복한다. max값을 반환한다.



객체 비교는 Coin 클래스의 campareTo 함수를 이용한다. 두 객체의 value의 차를 구하여 각각의 값을 비교하여 주어진 음수 양수 0 중 하나를 반환한다.

1. **종합 설명서**

* 프로그램 실행 순서대로 설명해보자.



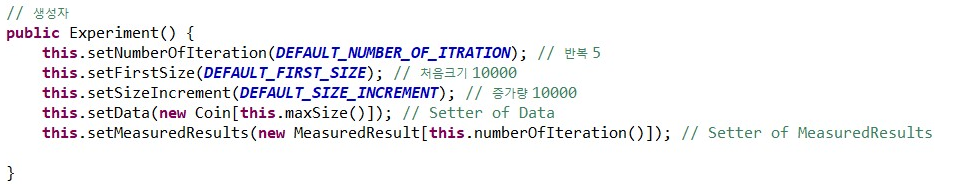
main에서 AppController의 객체를 생성하여 run 한다. 프로그램 실행.



AppConroller의 run 함수에서는 프로그램 시작과 진행을 알리는 출력문들과 experiment의 생성자를 만들어 각각의 리스트의 성능을 측정한 결과물을 출력해준다. 여기서 showExperimentResults 함수는 측정 결과를 출력하는 함수이다.



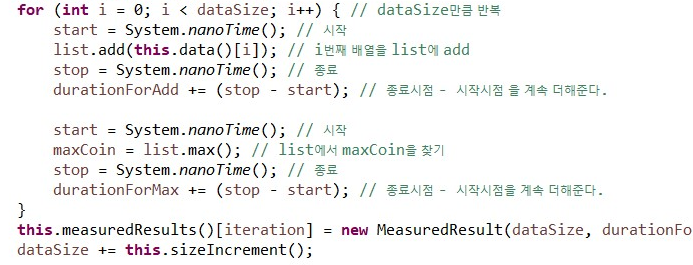
AppController은 Experiment 객체 선언과 experiment(), generateData() 함수를 실행한다.



크기가 주어지지 않을 때의 Experiment 클래스의 생성자이다. 처음 리스트의 크기는 10000으로 초기화하고 증가 량을 10000씩 주어 10000, 20000, 30000 .. 50000까지 5번 반복하여 측정한다.



난수 생성 - Random 클래스를 import 하여 random 객체를 만든다. maxSize() 만큼 반복하여 난수를 생성하고 data()[] 배열에 삽입한다. maxSize() = firstSize(10000) + sizeIncrement(10000) \* Interation 4 = 50000



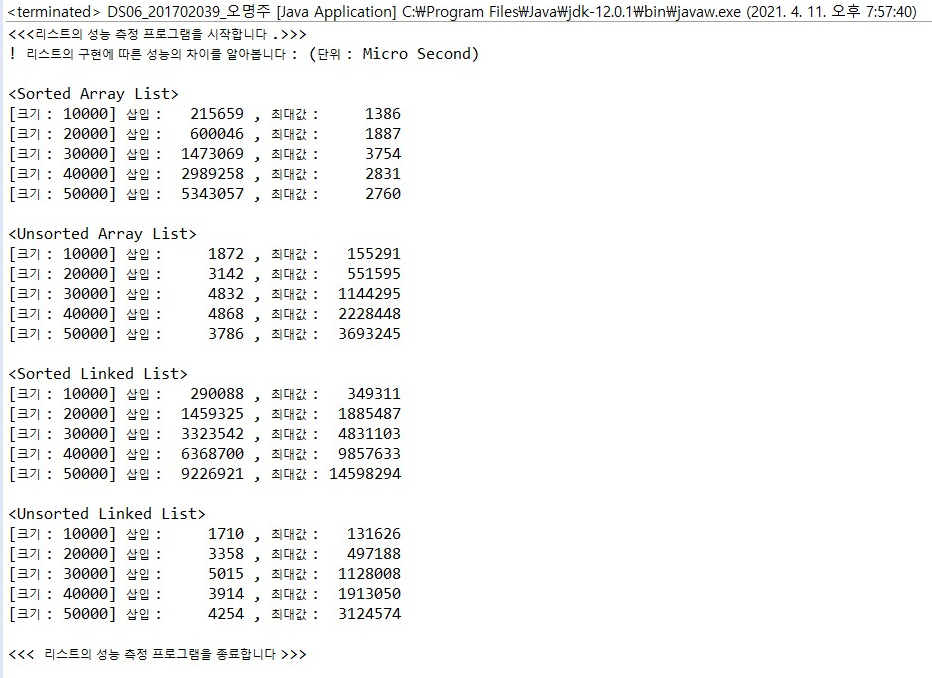
측정을 진행하는 함수의 반복문. dataSize만큼 반복한다. Start 측정 시작 – 리스트에 add – Stop 측정 종료, Start 측정 시작 – 리스트에서 max 찾기 – Stop 측정 종료를 반복한다.

Experiment 클래스에서 결과를 AppController로 넘겨 주어 화면에 출력한다.

1. **프로그램 장단점 / 특이점 분석**

* **장점**
* MVC 모델을 이용하여 가독성과 생산성이 뛰어나다. 각 클래스, 함수의 역할이 분명해서 코드와 프로그램을 잘 이해할 수 있다.
* 구현 프로그램의 실행 시간이나 성능을 측정할 수 있다는 게 편리하다고 느껴졌다. 앞으로 구현할 때나 프로그램을 개발할 때 시간을 측정하면서 효율적으로 개발할 수 있을 것 같다.
* 각각의 리스트에 대한 성능을 확인했으니 목적에 맞게 리스트들을 이용할 수 있다. 만약 정보를 찾을 때 시간이 적은 경우가 필요하다면 SortedList를 이용하면 될 것이다.
* **단점**
* 난수를 생성하여 사용하면서 코드 구현이 복잡해졌다. 실제로 정보를 저장할 때는 난수를 생성하지 않아도 되므로 해결될 것이다.
* SortedList는 삽입 할 때 시간이 오래 걸리는 것을 확인할 수 있었다.
* unSortedList는 max 값을 찾을 때 시간이 오래 걸리는 것을 확인할 수 있었다.
* LinkedList는 ArrayList에 비해 최대값 찾는 것에 시간이 걸리는 것을 확인하였다. 인덱스를 이용한 출력이 바로 가능한 Array에 비해 가장 마지막 노드까지 걸어 나가야 하기 때문이다.

1. **실행 결과 분석**
2. **입력과 출력 (화면 capture하여 제출)**

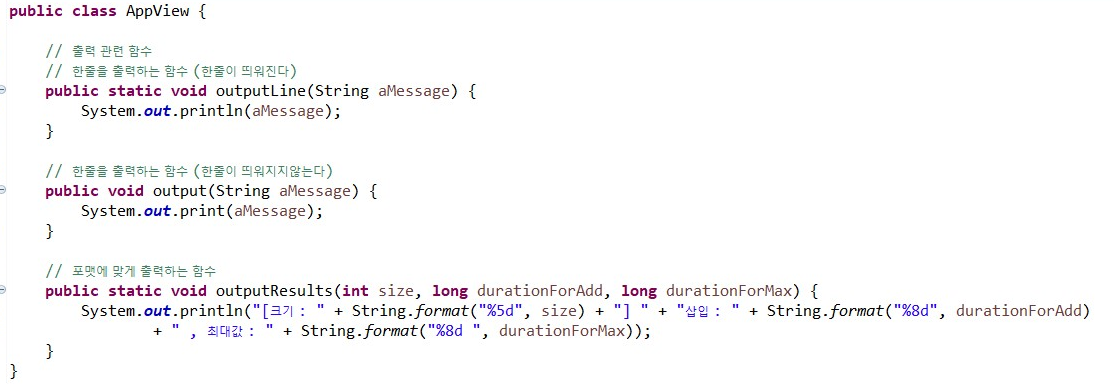
****

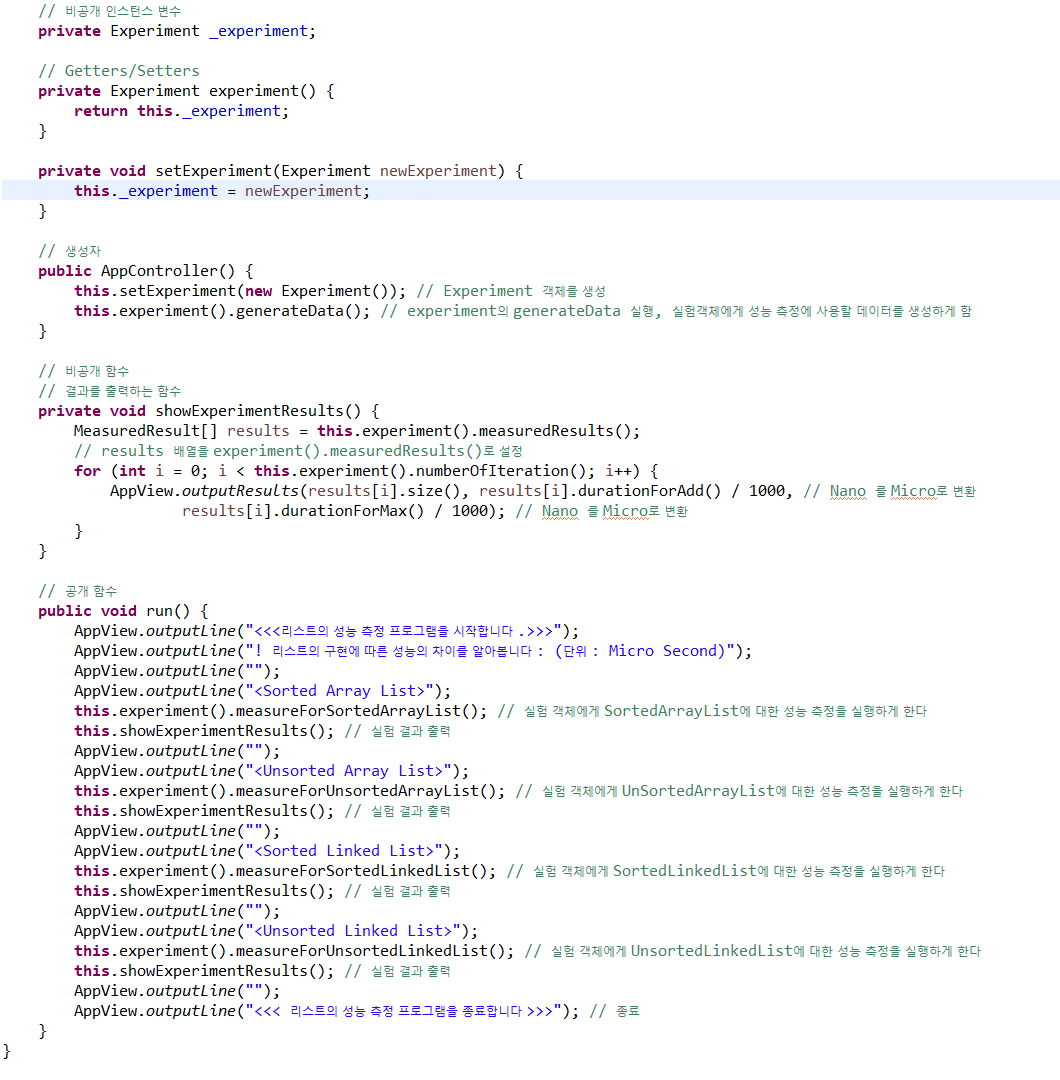
1. **결과 분석 (자신의 논리적 평가, 기타 느낀 점)**

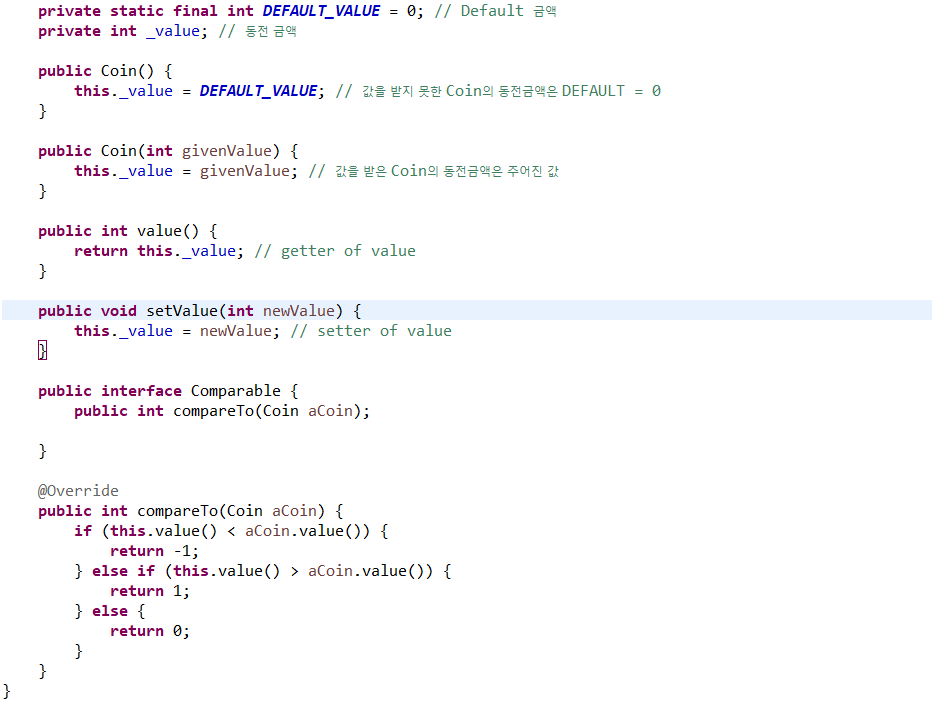
* 정렬된 리스트, 정렬되지 않은 리스트, 배열, 노드 차이를 시간 복잡도를 통해 알 수 있었다.  
  SortedArrayList – add O(n), max O(1) (add가 반복문을 통해 값을 삽입하기 때문)  
  UnsortedArrayList – add O(1), max O(n) (max 값을 찾을 때 반복문 이용)  
  SortedLinkedList – add O(n), max O(n) (max 값 찾을 때도 반복문으로 끝 노드까지 가기때문)  
  UnsortedLinkedList – add O(1), max O(n) (max 값 찾을 때 반복문 이용)
* UnsortedLinkedList를 구현할 때 처음엔 리스트의 마지막에 값을 삽입하는 것으로 구현하였는데 그러다 보니 반복문을 통해 마지막 노드까지 가야하는 복잡함이 있어 O(n) 만큼 걸리었다. 성능을 향상시키기 위해 head에 노드를 추가하는 형태로 바꾸었더니 O(1) 로 성능이 향상된 것을 확인할 수 있었다.
* 정렬을 하면 더 좋은 배열이라고 판단될 수 있지만 삽입하고 정렬하는데 시간이 오래 걸린다는 것을 알게 되었다. 프로그램을 실행하면서 크기를 10만으로 변경하여 실행하였더니 1분정도 더 오래 걸리는 것을 확인하였다.

1. **소스코드**















* 상단에 보고서에 첨부된 소스코드는 따로 캡쳐 하지 않았습니다.
* Experiment Class 에 대한 소스코드는 코드를 참고 부탁드립니다.