**자료구조 실습 보고서**

[제11주] 정렬 결과 검증

**2021년 05월 17일**

**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.

* 정렬 결과 검증 프로그램에서의 각 클래스 별 MVC 구조 역할

**Model** :

* DataGenerator : 데이터를 생성하고 배열을 만든다.
* Sort<E> (Abstract) : 삽입정렬과 퀵정렬의
* InsertionSort<E> : 삽입정렬의 기능을 구현한다.
* QuickSort<E> : 퀵정렬의 기능을 구현한다.
* ListOrder (Enum) : 오름차순/내림차순/랜덤값 세가지 방법에 대한 Enum 클래스.

**View** :

* AppView : 프로그램의 입/출력을 담당한다.

**Controller** :

* AppController : Model을 통해 생성된 결과물을 AppView를 통해 출력한다.

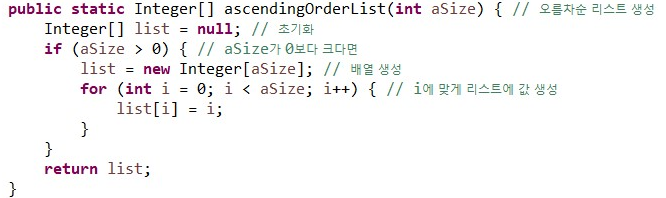
1. **함수 설명서**

* 주요 알고리즘

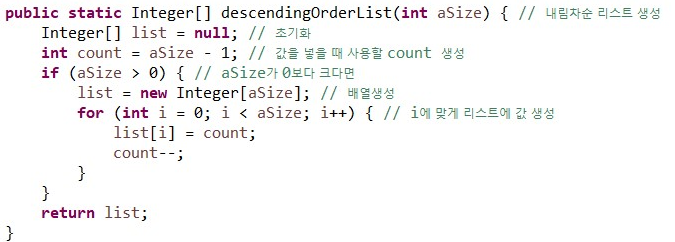
1. 오름차순/내림차순/무작위 리스트 생성



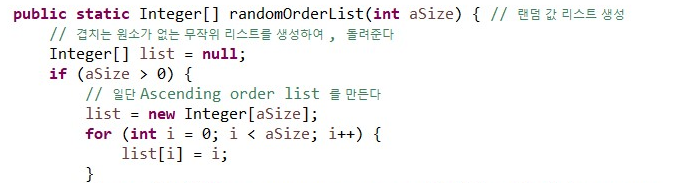
각각의 오름차순/내림차순/무작위 리스트를 출력하는 함수가 존재하는데, 이 때 DataGenerator을 통해 주어진 크기만큼의 리스트를 생성한다.



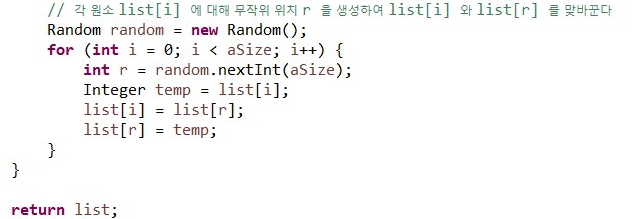
* 오름차순 리스트 생성.
* 먼저 배열 크기를 매개변수로 받아와서 유효한지 판단한다. (aSize>0)
* for문을 통해 1-aSize만큼 각각의 인덱스에 각각의 값을 삽입한다.



* 내림차순 리스트 생성
* 오름차순과 다른 부분은 aSize-1 만큼의 count 변수를 따로 생성하여 인덱스와 다르게 내림차순으로 수를 삽입한다.



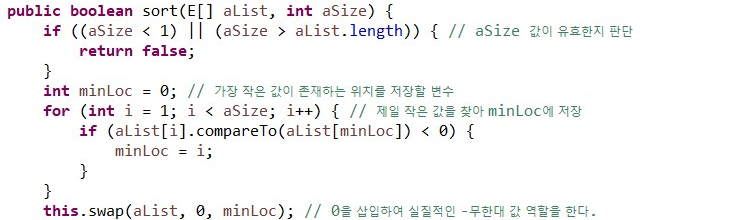
* 무작위로 랜덤값 리스트 생성
* 먼저 오름차순으로 리스트를 생성한다.



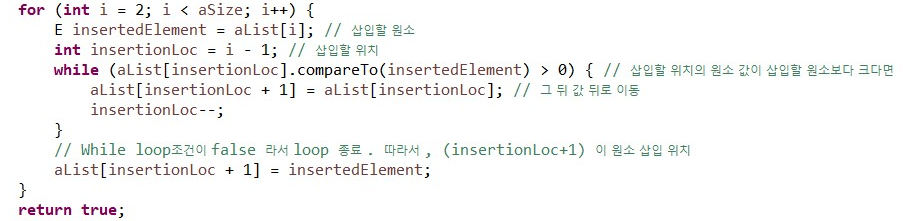
* 각 원소 list[i]에 대해 무작위 위치 r을 생성하여 바꾼다.
* 해당 리스트를 반환

1. 리스트 정렬

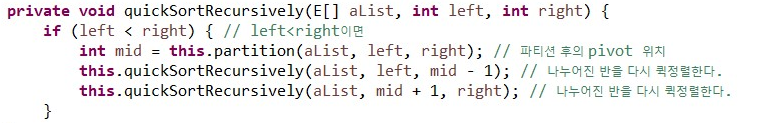
* 삽입정렬



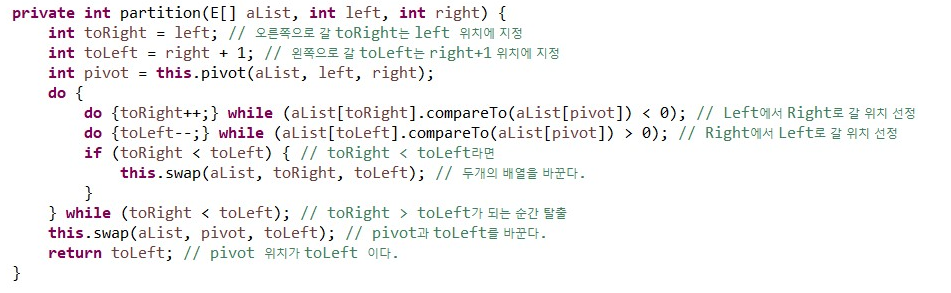
* 먼저 aSize값이 유효한지 판단
* 가장 작은 값이 존재하는 위치를 저장할 변수 minLoc을 선언하고 0으로 초기화한다.
* 주어진 배열에서 가장 작은 값을 찾고 0으로 삽입하여 -무한대 값 역할을 하도록 한다.



* 삽입할 원소를 i번째 값으로 선언하고 삽입할 위치를 i-1로 선언하여 둘을 비교한다.
* 삽입할 위치의 원소값이 삽입할 원소보다 크다면 그 후 모든 값을 한칸씩 뒤로 옮기고 해당 위치에 값을 삽입한다.
* 퀵정렬



* 반으로 나누어 각각의 반에 대해 퀵정렬을 실행한다. (재귀)



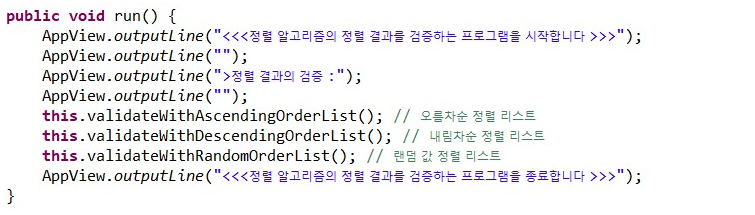
* partition 함수
* left를 toRight로, right+1 위치를 toLeft로 설정한다. pivot은 함수를 통해 left 값을 받는다.
* toRight은 left로 걸어가며 pivot보다 적은 값을 찾아 멈추고, toLeft는 right로 걸어가며 pivot보다 큰 값을 찾아 멈춘다.
* 둘을 비교하여 swap한다.
* toRight가 toLeft보다 커지면 pivot과 toLeft를 swap 한다.

1. **종합 설명서**

* 프로그램 실행 순서대로 설명해보자.



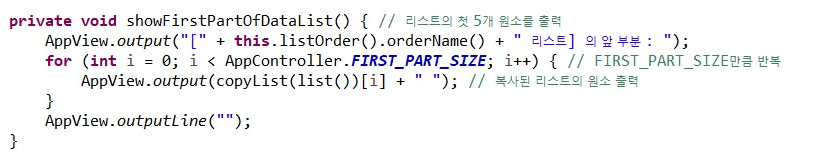
main에서 AppController 의 객체를 생성하여 run 한다. 프로그램을 실행한다.



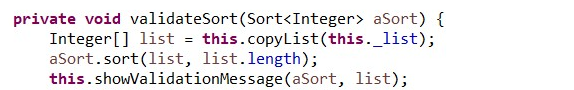
* AppController의 run 함수이다. 프로그램 실행
* 오름차순, 내림차순, 랜덤 값 리스트에 대한 실행을 각각의 함수를 통해 진행한다.



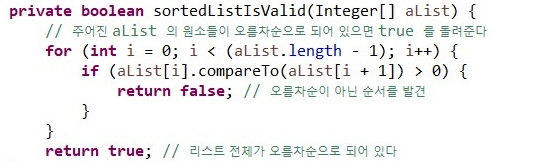
* 랜덤값 리스트를 실행하는 validateWithRandomOrderList 함수이다.
* ListOrder에서 Random으로 지정한다.
* 크기가 TEST\_SIZE인 크기의 배열을 생성한다.
* 처음 5개 원소를 출력한다.
* 결과값을 출력한다.



* 리스트의 첫 5개 원소를 출력하는 showFirshPartOfDataList 함수이다.
* 미리 정의한 FIRST\_PART\_SIZE (5개) 만큼 반복문을 반복하여 원소를 출력한다.



* 동일한 리스트로 2번 정렬한다 (Quick, Insert)
* 매번 원본 리스트를 복사하여 정렬한다.
* copyList : 주어진 배열 객체의 복사본을 만든다.



* 주어진 aList 원소들이 오름차순으로 되어있으면 True를 반환한다.
* 해당 함수가 true면 검증이 올바르다는 출력문을, false면 검증이 잘 못되었다는 출력문을 출력한다.

1. **프로그램 장단점 / 특이점 분석**

* **장점**
* 정렬에 공통부분을 추상클래스로 정의하여 새로운 정렬을 추가하거나 정렬을 수정하고 싶을 때, Sort나 Swap 함수에 대한 수정은 하지 않고 사용하여 구현할 수 있다.
* 각 정렬에 대한 검증을 해볼 수 있어서 구현한 정렬이 올바르게 작동하는지 확인할 수 있다.
* **단점**

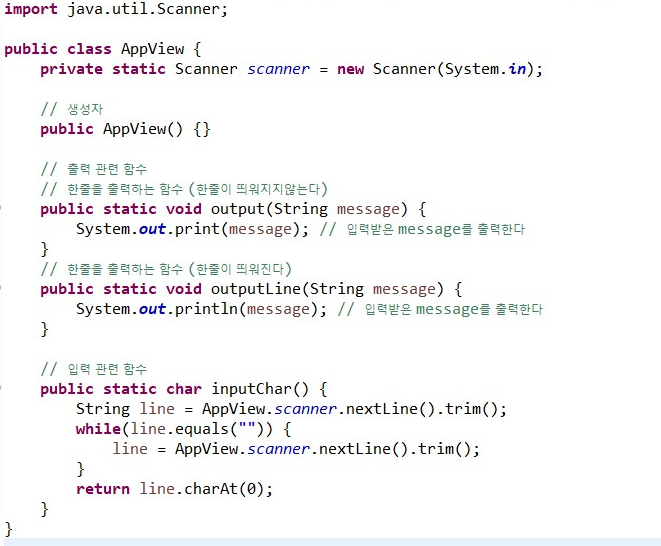
1. **실행 결과 분석**
2. **입력과 출력**



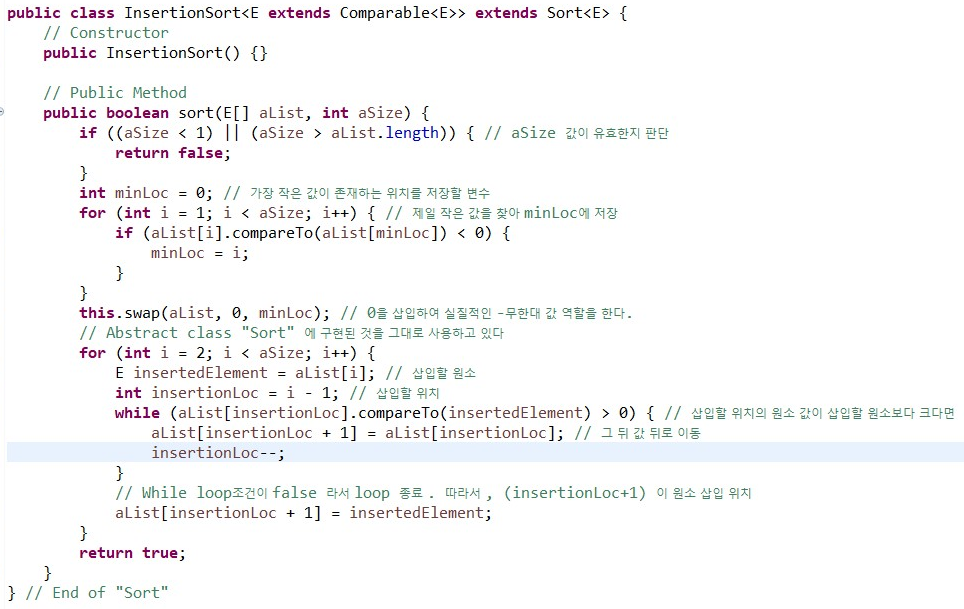
1. **결과 분석 (자신의 논리적 평가, 기타 느낀 점)**

* 구현한 코드에 대한 검증을 할 수 있어서 유용한 프로그램이라고 생각했다.
* .
* 생각해 볼 점
* 추상클래스와 인터페이스의 차이점
* Abstract : 추상 메소드가 하나이상 포함 된 경우. new 연산자를 이용하여 객체를 생성할 수 있다. 단일 상속만 가능하다.
* Interface : 모든 메소드가 추상 메소드인 경우. 구현 객체가 같은 동작을 하는 것을 보장한다. 다중 상속이 가능하다.

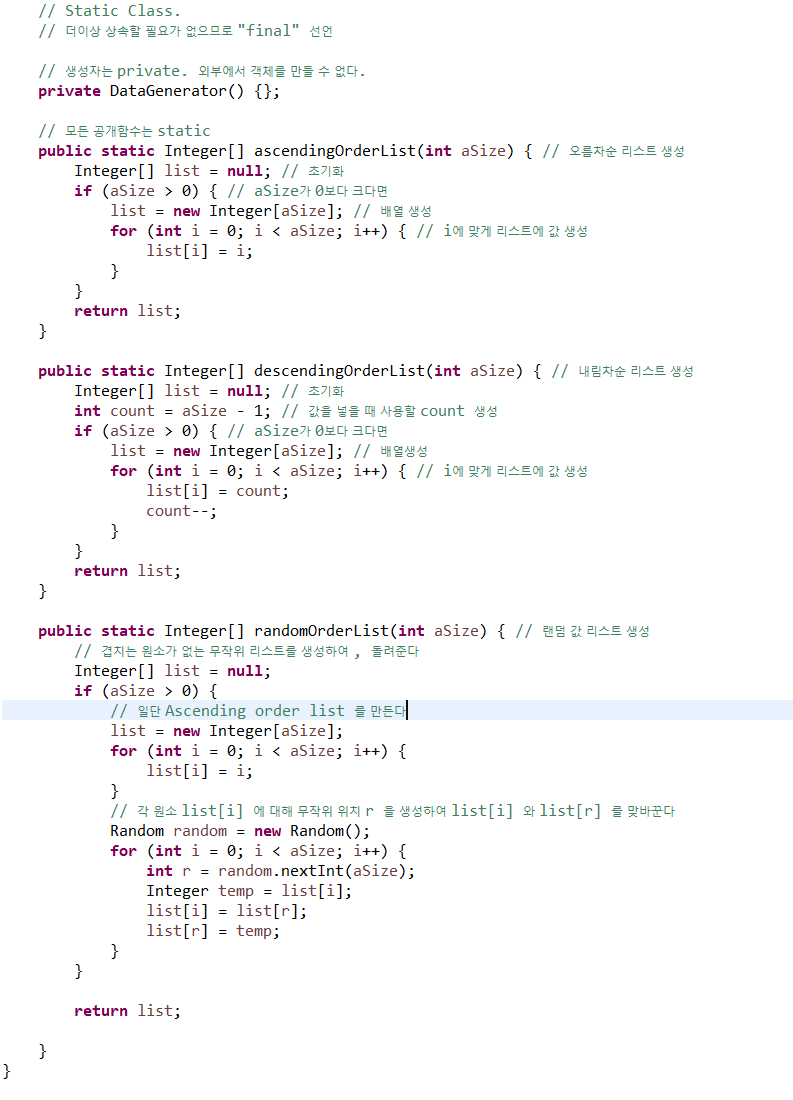
1. **소스코드**

****

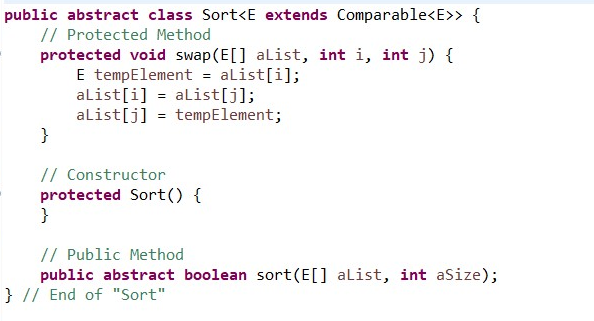
**[AppView]**

****

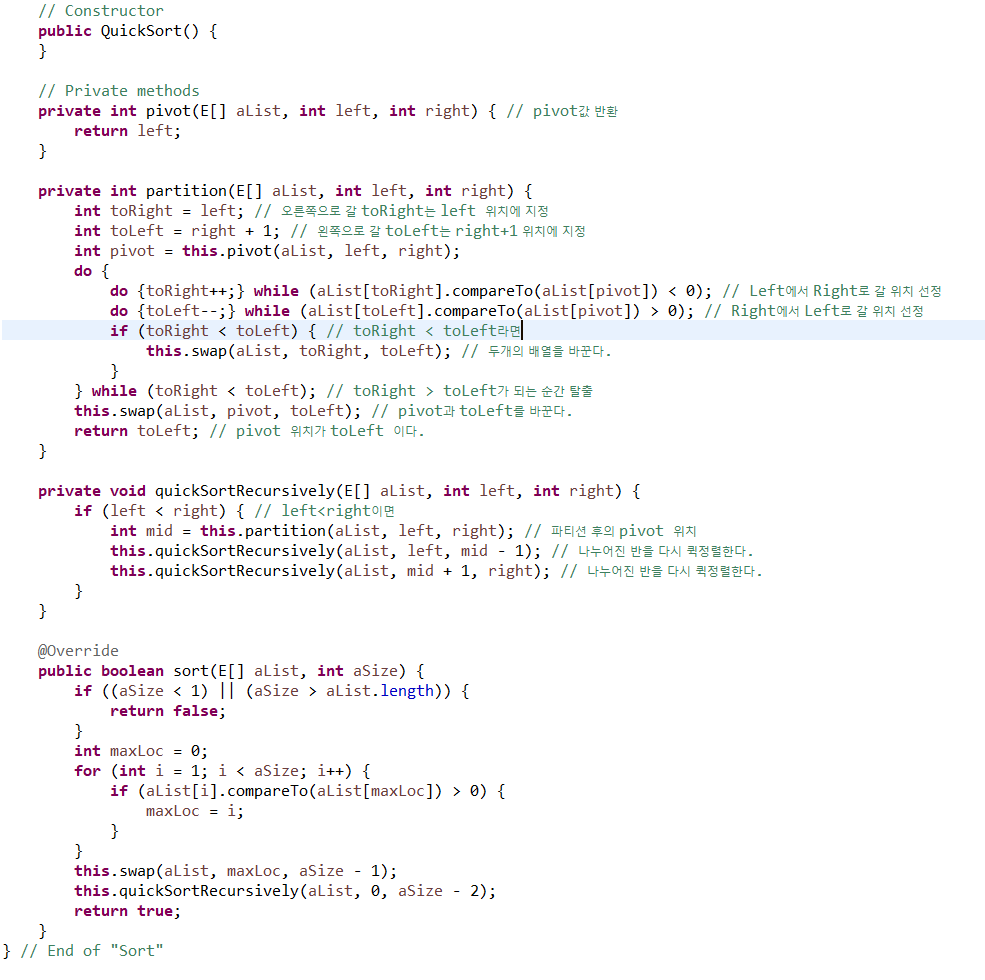
**[InsertionSort]**



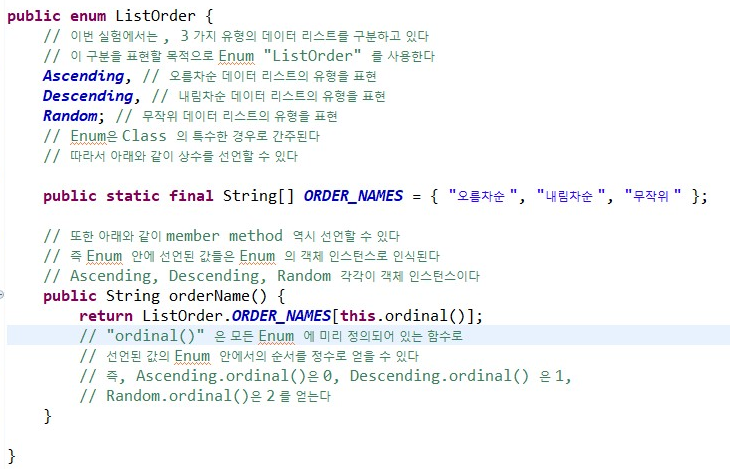
**[DataGenerator]**

****

**[Sort]**



**[QuickSort]**

****

**[ListOrder]**



**[AppController]**