**자료구조 실습 보고서**

[제13주] 이진검색트리로 구현된 사전\_성적처리

**2021년 05월 31일**

**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(Dictionary)** :

* DictionaryElement<Key, Obj> : Key와 Obj에 대한 속성과 getter/setter가 존재
* Dictionary : 추상클래스로, 사전 기능에 대한 정의
* DictionaryByBinarySearchTree<Key, Obj> : String과 Student가 쌍으로 저장되는 이진탐색트리
* BinaryNode<E> : 트리에 대한 BinaryNode를 정의

**Model(Ban)** :

* Student : 학생 객체 생성할 수 있는 클래스
* Ban : 학생 배열 받을 학급을 의미하는 클래스
* GradeCounter : 성적에 따른 학점을 정하고 학생 수 관리

**Model(Sort)** :

* Sort<E> (Abstract) : 정렬의 공통 기능.
* QuickSort<E> : 퀵정렬의 기능을 구현한다.

**Model(Stack)** :

* Stack<E> : 인터페이스로 스택 기본 기능 구성
* ArrayList<E> : 스택을 Array로 구현한 클래스

**Model(Iterator)** :

* Iterator<E> : 반복자 인터페이스. 반복자 기능을 구성

**View** :

* AppView : 프로그램의 입/출력을 담당한다.

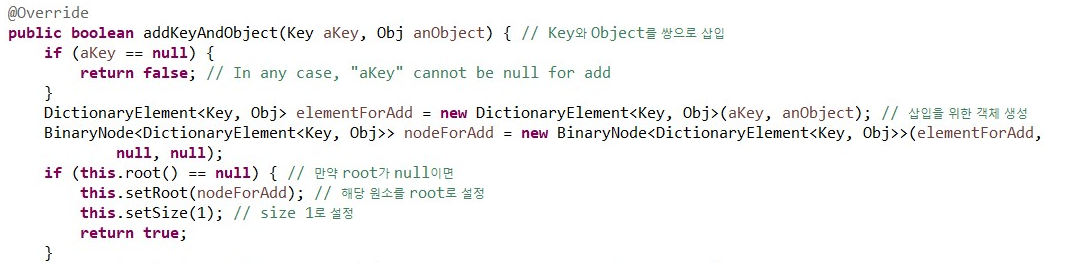
**Controller** :

* AppController : Model을 통해 생성된 결과물을 AppView를 통해 출력한다.

1. **함수 설명서**

* 주요 알고리즘

1. addKeyAndObject



* 이진 탐색 트리에 Key와 Object를 삽입하는 함수
* 삽입을 위한 DictionaryElement 타입의 변수를 생성하고 해당 aKey와 anObject를 값으로 넣는다.
* 만약 root가 null이면 해당 원소를 root로 설정하고 size를 1로 설정한다.



* 만약 원소가 하나라도 존재한다면 current 노드를 root로 설정하고 삽입할 원소의 key와 비교한다.
* while문 종료조건 : aKey와 current 원소의 키가 같은경우  
  - 만약 aKey가 root보다 작다면 (leftsubtree로)  
  - 만약 left()가 null이라면 left로 설정하고 size 증가 후 true를 return한다.  
  - left()가 null이 아니면 current 원소를 current.left()로 이동  
  - 만약 aKEy가 root보다 크다면 (rightsubtree로)  
  - 만약 right()가 null이라면 right로 설정하고 size 증가 후 true를 return한다.  
  - right()가 null이 아니면 current 원소를 current.right()로 이동

1. Iterator (중위탐색)



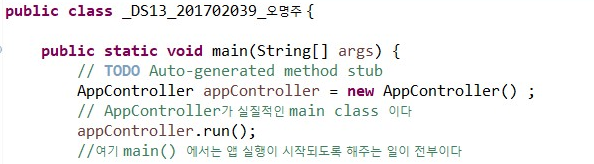
* Iterator의 hasNext() 함수. Interface Iterator를 구현
* stack이 Empty가 아니거나 nextNode가 null이 아니라면 true를 반환



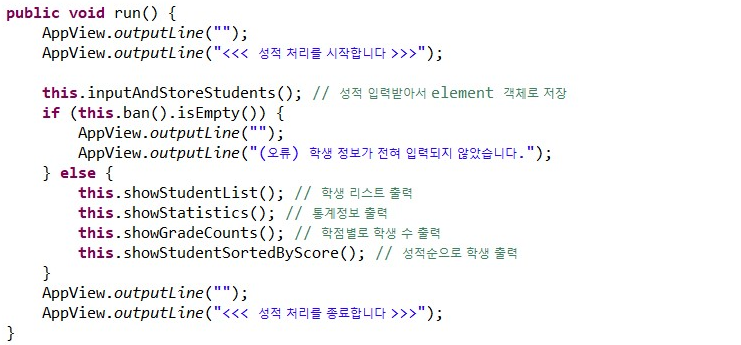
* 원소를 탐색하는 next() 함수.
* hasNext가 false라면 null을 반환.
* nextNode()가 null이 아니면 while문을 반복  
  - stack에 nextNode()를 push한다.  
  - nextNode의 left를 nextNode로 설정한다. (중위탐색에는 left->root->right순이기 때문)
* stack을 pop하여 BinaryNode로 저장하고 element를 반환한다.

1. **종합 설명서**

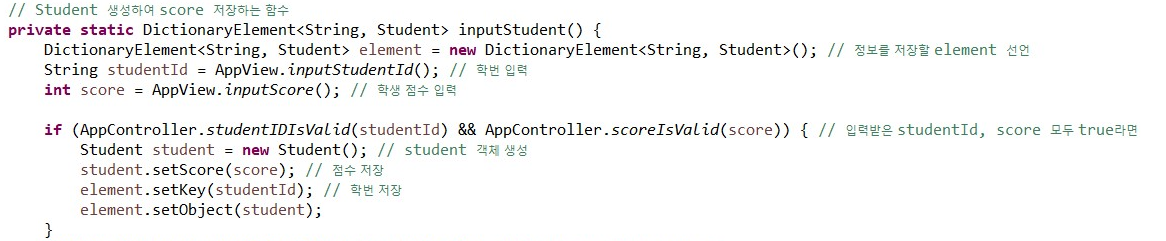
* 프로그램 실행 순서대로 설명해보자.



main에서 AppController 의 객체를 생성하여 run 한다. 프로그램을 실행한다.



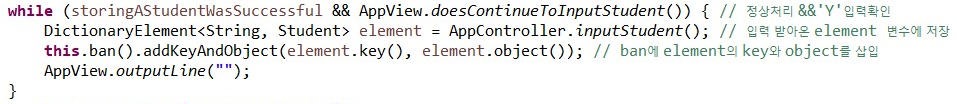
* AppController의 run 함수이다. 프로그램 실행
* inputAndStroeStudents() 함수로 학번과 성적을 입력받아 element 객체로 저장한다.
* 아무 정보도 없다면 오류문을 출력한다.
* 그렇지 않다면 학생리스트, 통계정보, 학점별로 학생수, 성적순 리스트를 출력한다.



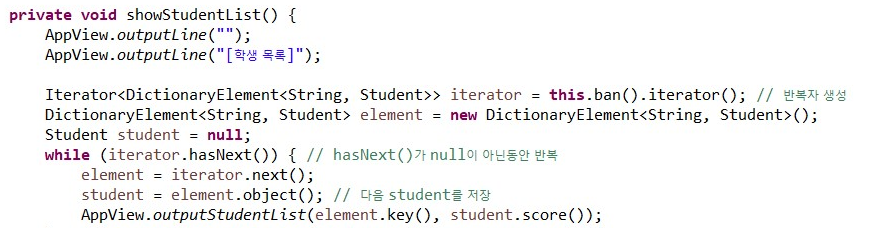
* 정보를 저장할 element를 선언하고 사용자로부터 학번과 학생 점수를 입력받는다.
* 입력받은 학번과 점수의 유효성을 확인하여 모두 true라면 Student 객체를 생성하여 score를 저장하고 element에 key와 object를 설정한다.



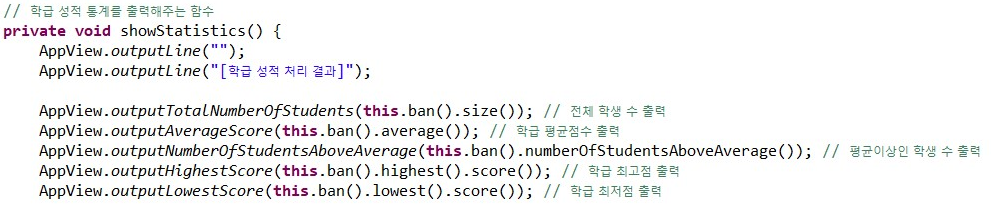
* 학번과 성적이 유효하지 않다면 오류문을 출력한다.



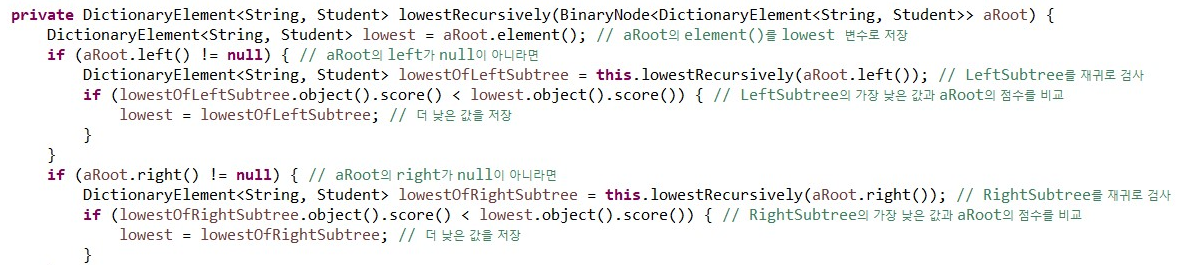
* 해당 원소를 삽입한다.



* 학생 리스트를 출력하는 showStudentList() 함수
* 반복자를 생성하고 원소를 저장할 element 변수 선언
* iterator가 null이 아닌동안 while문 반복  
  - element에 next() 원소를 저장  
  - student 객체에 element의 object를 저장 , 출력문 출력
* next가 중위탐색을 실행하므로 해당 반복문은 학번순으로 출력하게 된다.



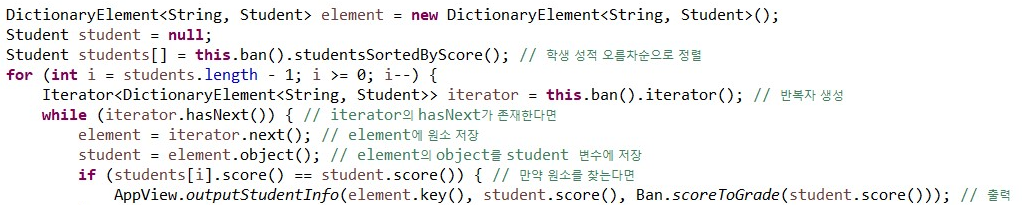
* 학급 성적 통계를 출력하는 showStatistics() 함수
* 전체 학생 수, 평균점수, 평균 이상의 학생 수, 학급 최고점, 최저점을 출력한다.



* 최저점을 재귀로 구하는 lowestRecursively 함수
* aRoot의 left, right를 재귀적으로 가장 작은 값을 구한다.
* aRoot의 left가 null이 아니라면 LeftSubtree를 재귀적으로 검사하여 가장 낮은 값과 aRoot 값을 비교하여 subtree의 가장 낮은 값을 저장한다. RightSubtree도 재귀적으로 검사하여 작은 값을 저장하여 반환한다.



* 학생들을 성적순으로 출력하는 showStudetnSortedByScore() 함수

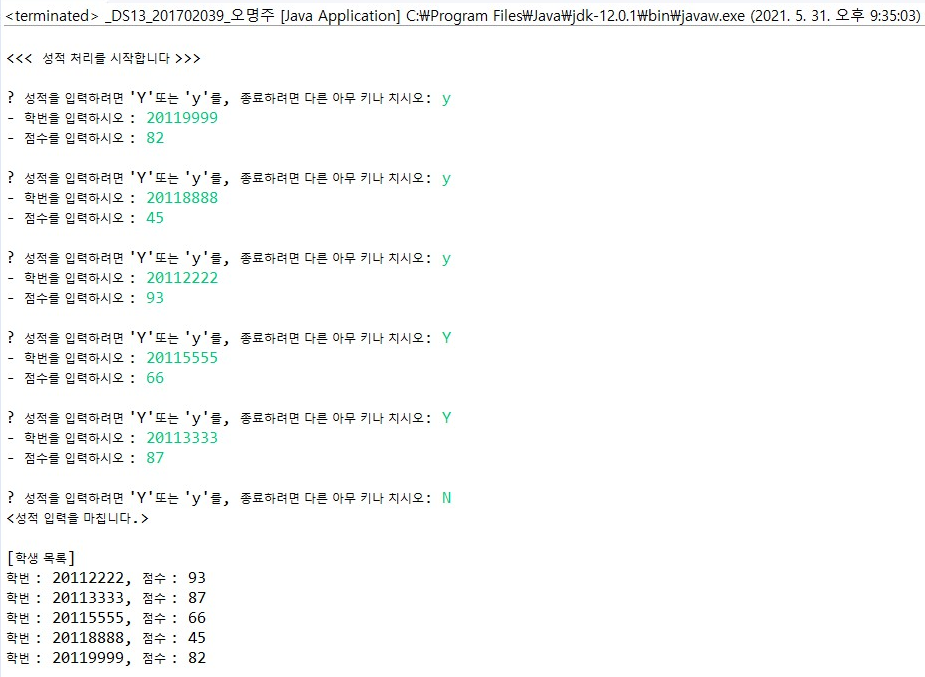


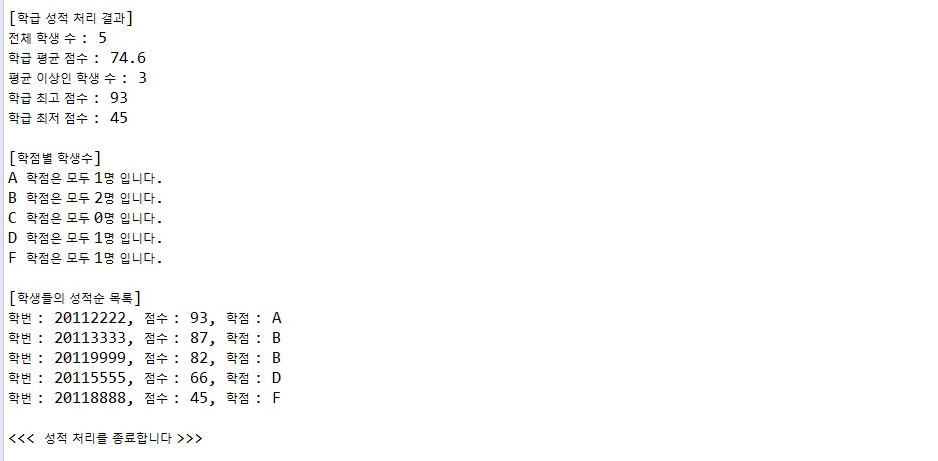
* 원소를 저장할 element 변수와 Student 객체 변수를 선언한다.
* students 배열 객체에 학생 성적을 오름차순으로 정렬한 결과를 저장한다.
* students 배열은 오름차순이므로 마지막 idx-0까지 for문을 반복한다.  
  - iterator을 통해 ban의 원소들과 students[i]를 비교하여 같은 점수를 발견하면 key와 score를 출력한다.

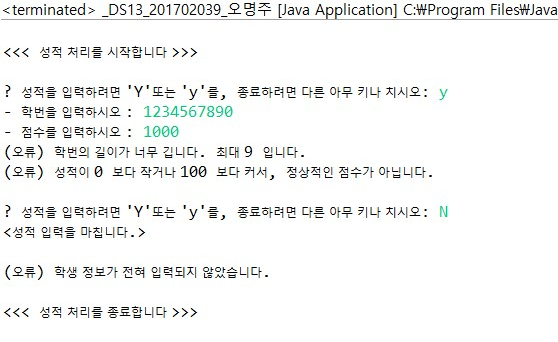
1. **프로그램 장단점 / 특이점 분석**

* **장점**
* 삽입, 삭제, 검색 모두 평균적으로 시간 복잡도가 O(logN)으로 트리의 높이에 비례하여 다소 빠르다.
* **단점**
* 최악의 경우, 시간 복잡도는 O(N)이 될 수도 있다. 한쪽으로만 기울어진 형태는 노드의 개수에 비례하여 삽입, 삭제, 검색이 이루어진다.

1. **실행 결과 분석**
2. **입력과 출력**





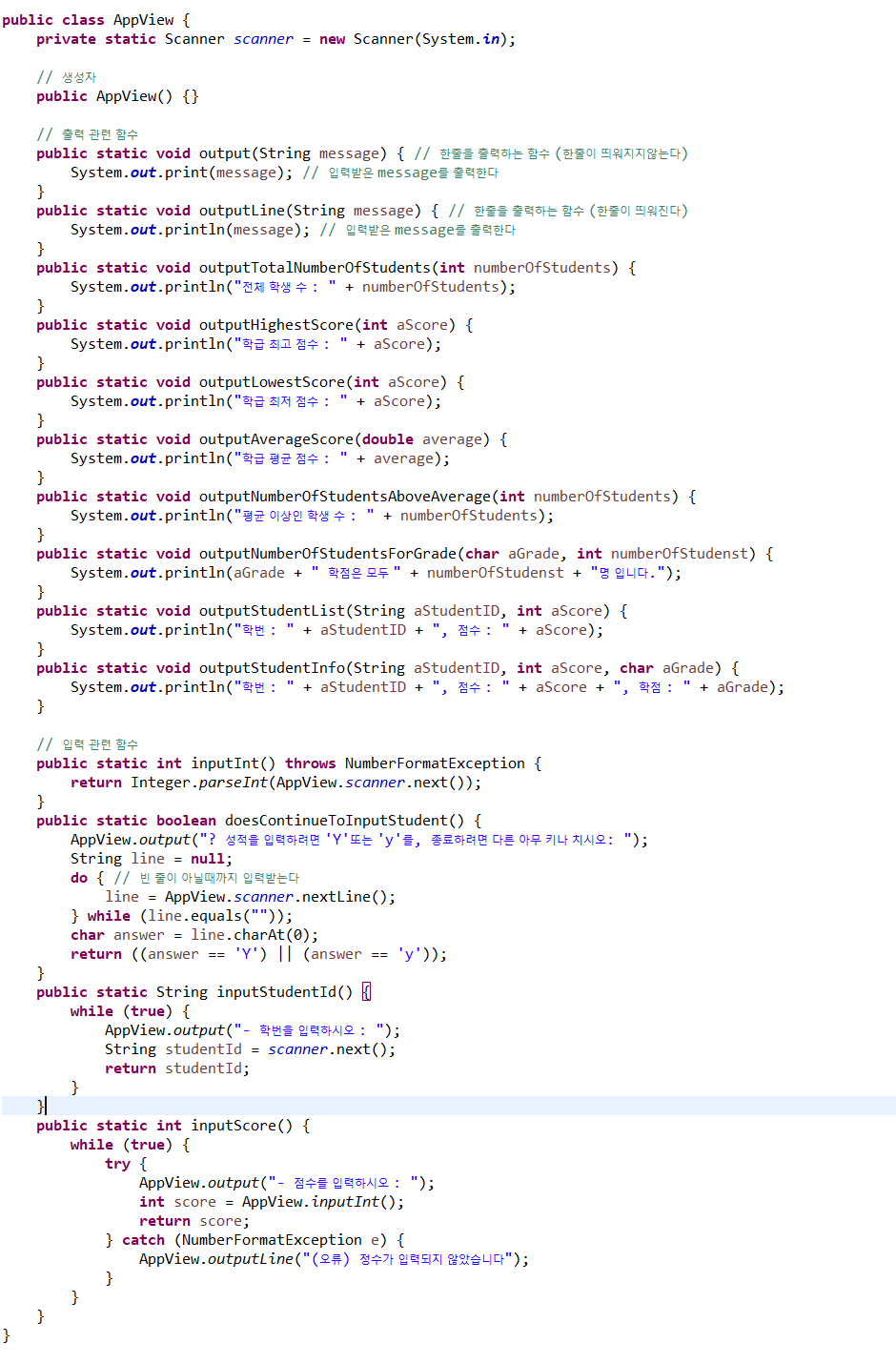


* 학번의 길이가 9보다 긴 경우, 성적이 정상적이지 않은 경우, 학생 정보가 입력되지 않고 종료된 경우 -> 오류처리

1. **결과 분석 (자신의 논리적 평가, 기타 느낀 점)**

* 이론적 시간 복잡도
* 이진 탐색 트리는 평균적으로 O(logN)으로 트리의 높이에 비례하지만 Left 혹은 Right 한쪽으로 쏠린 트리인 경우 시간복잡도가 O(N)으로 노드에 비례하여 이진탐색트리의 장점을 살리지 못 한다.
* 이 경우 AVL Tree로 해결할 수 있다.

1. **소스코드(대부분의 클래스 예전 과제 클래스이기 때문에 수정 부분만 캡쳐)**



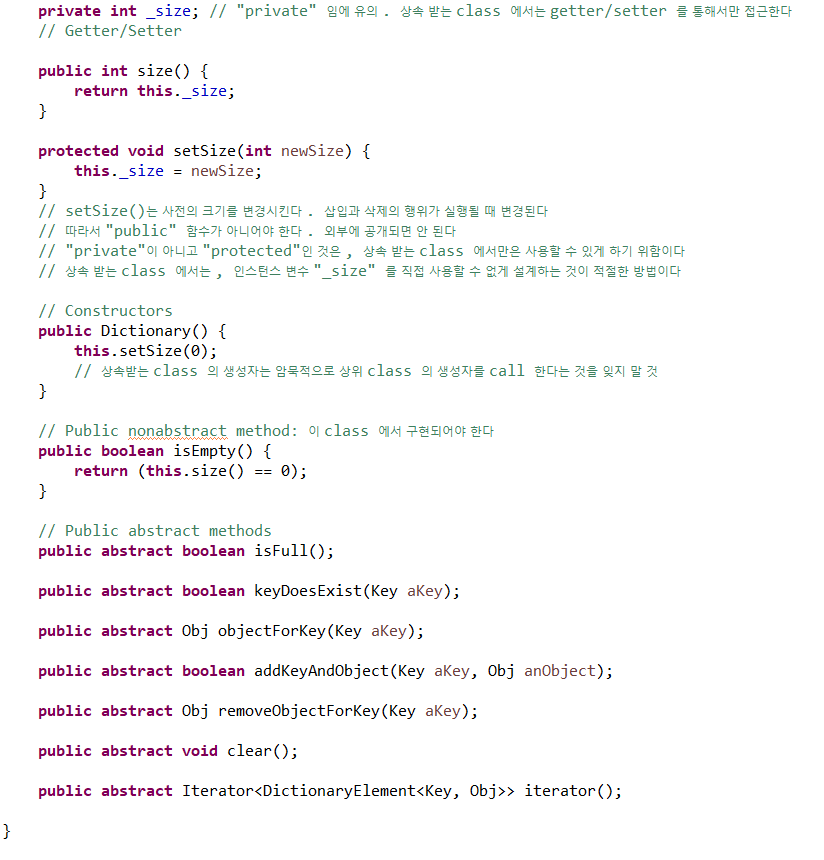
**[AppView]**

**[DictionaryByBST]**

**[AppController]**

****

**[DictionaryElement]**



**[Dictionary]**