**자료구조 실습 보고서**

[제14주] 이진검색트리 사전에서의

삽입, 삭제, Call Back

**2021년 06월 08일**

**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**:

* 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
* Stack<E> : 인터페이스로 스택 기본 기능 구성
* LinkedStack<E> : 스택을 LinkedList로 구현한 클래스
* LinkedNode<E> : LinkedList를 구성하는 LinkedNode에 대한 속성, 변수를 정의
* Iterator<E> : 반복자 인터페이스. 반복자 기능을 구성
* VisitDelegateForTraversal<E> : Visit 할 때마다 실행할 CallBack 함수사용볍 정의한 인터페이스.

**View** :

* AppView : 프로그램의 입/출력을 담당한다.

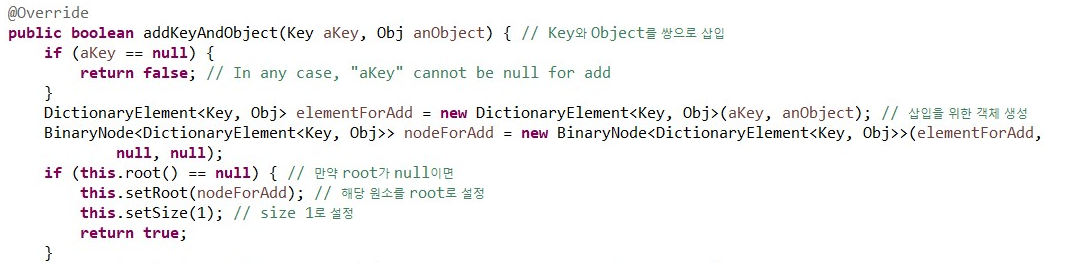
**Controller** :

* AppController : Model을 통해 생성된 결과물을 AppView를 통해 출력한다.

1. **함수 설명서**

* 주요 알고리즘

1. addKeyAndObject

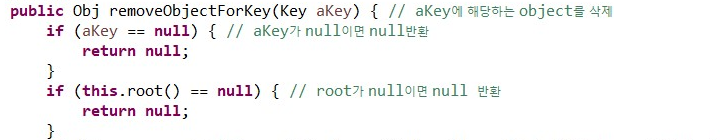


* 이진 탐색 트리에 Key와 Object를 삽입하는 함수
* 삽입을 위한 DictionaryElement 타입의 변수를 생성하고 해당 aKey와 anObject를 값으로 넣는다.
* 만약 root가 null이면 해당 원소를 root로 설정하고 size를 1로 설정한다.



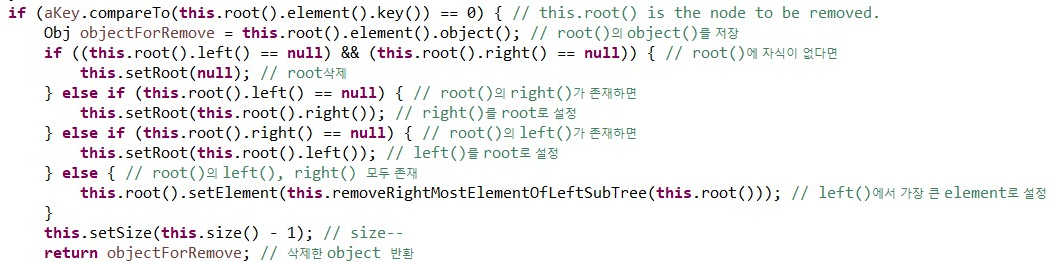
* 만약 원소가 하나라도 존재한다면 current 노드를 root로 설정하고 삽입할 원소의 key와 비교한다.
* while문 종료조건 : aKey와 current 원소의 키가 같은경우  
  - 만약 aKey가 root보다 작다면 (leftsubtree로)  
  - 만약 left()가 null이라면 left로 설정하고 size 증가 후 true를 return한다.  
  - left()가 null이 아니면 current 원소를 current.left()로 이동  
  - 만약 aKEy가 root보다 크다면 (rightsubtree로)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

1. removeObjectForKey



주어진 aKey에 해당하는 object를 삭제하는 removeObjectForKey 함수

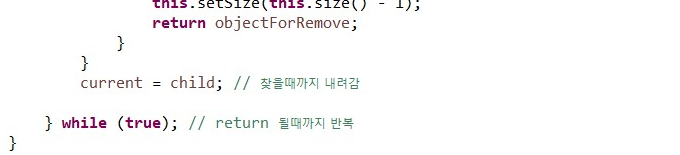
* aKey가 null이라면 null을 반환
* this.root()가 null이라면 null을 반환(비어있는 경우)



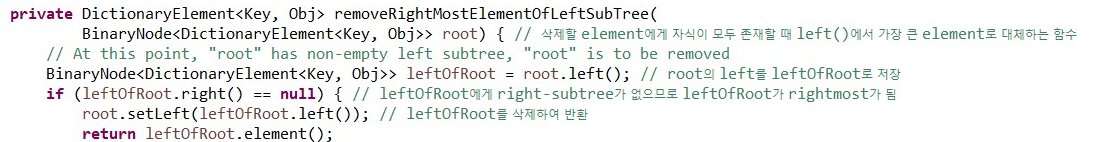
* aKey와 this.root().element().key()를 비교하여 같다면 반환해주기 위해 objectForRemove 변수에 root의 object를 저장한다.
* 만약 삭제할 root 원소에 자식이 하나도 없다면 root를 null로 설정한다.
* 만약 삭제할 root 원소에 left()가 null이라면 right()는 존재하므로 root에 root.right() 원소를 설정한다. 마찬가지로 root 원소에 right()가 null이라면 root에 root.left() 원소를 설정한다.
* 만약 삭제할 root 원소에 모든 자식이 존재하면 left-subtree에서 가장 큰 값을 찾아서 반환해주는 removeRightMostElementOfLeftSubTree를 호출한다.
* size를 하나 감소하고 삭제한 object 반환한다.



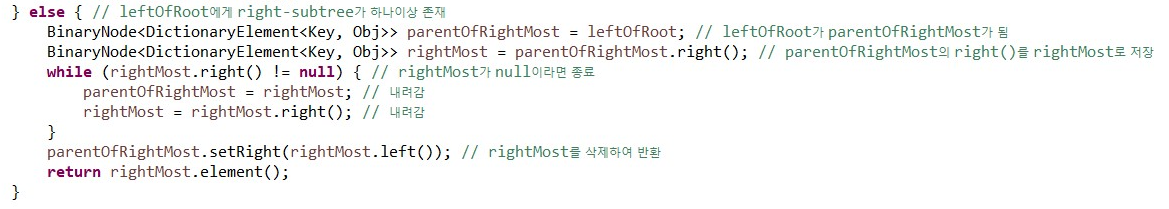
* root가 aKey에 해당하지 않는다면 subtree에서 찾아야하기 때문에 root를 의미하는 current와 실제 삭제할 값을 나타낼 child BinaryNode 변수를 선언한다.
* 찾을 때까지 반복하기 위해 do-while문을 사용한다.
* 만약 aKey가 current의 key보다 작다면 child를 current의 left()로 저장한다. 이때 child가 null이라면 찾으려는 값이 존재하지 않기 때문에 null을 반환한다.
* 만약 child가 aKey에 해당하면 root가 aKey인 경우와 마찬가지로 left(), right() 모두 존재하는 경우, left()만 존재하는 경우, right()만 존재하는 경우 자식 없는 경우로 나누어 처리한다.
* aKey가 current의 key보다 크면 child를 current의 right()로 저장하고 동일하게 진행한다.



* 만약 값을 찾지 못했다면 current를 child로 설정하여 내려간다.
* while문은 return될때까지 반복한다.



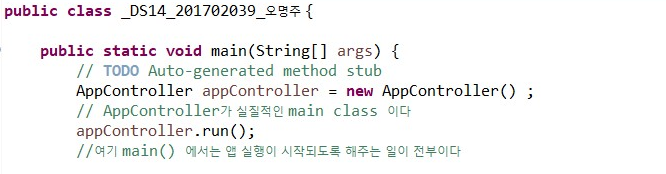
* 삭제하려는 노드에 자식이 모두 존재할 때 left-subtree에서 가장 right 값을 반환하는 removeRightMostElementOfLeftSubTree 함수.
* 전달받은 root가 삭제될 노드이며, left-subtree의 root를 leftOfRoot 변수에 저장한다.
* 만약 이 leftOfRoot 의 right()가 null이라면 leftOfRoot가 가장 큰 값이 되는 것이므로 root.left()에 leftOfRoot의 left()를 저장하고 leftOfRoot의 element를 반환한다.



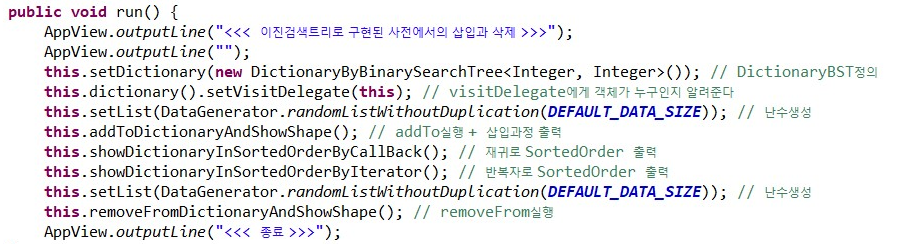
* leftOfRoot에게 right-subtree가 하나이상 존재하면 leftOfRoot를 parentOfRightMost에 저장하고 rightMost에 parentOfRightMost의 right()를 저장한다. rightMost가 실제 반환될 값을 의미한다.
* rightmost.right()가 null이 될 때까지 내려간다.
* parentOfRightMost의 right에 rightMost의 left를 설정하고 rightMost의 element를 반환한다.

1. **종합 설명서**

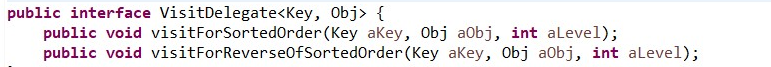
* 프로그램 실행 순서대로 설명해보자.



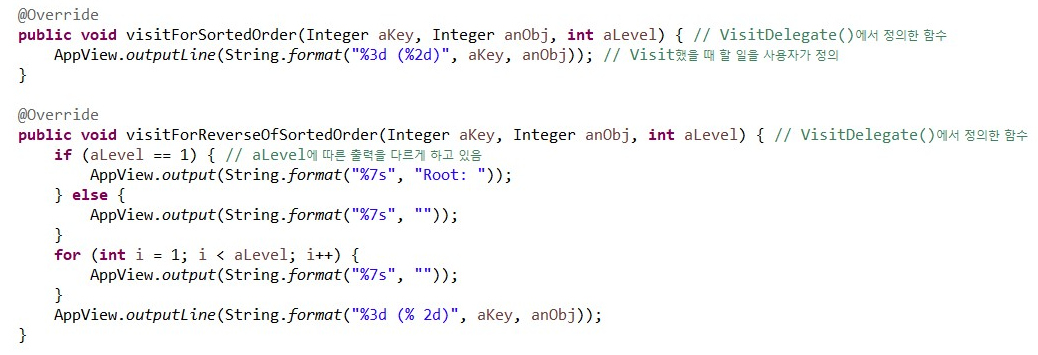
main에서 AppController 의 객체를 생성하여 run 한다. 프로그램을 실행한다.



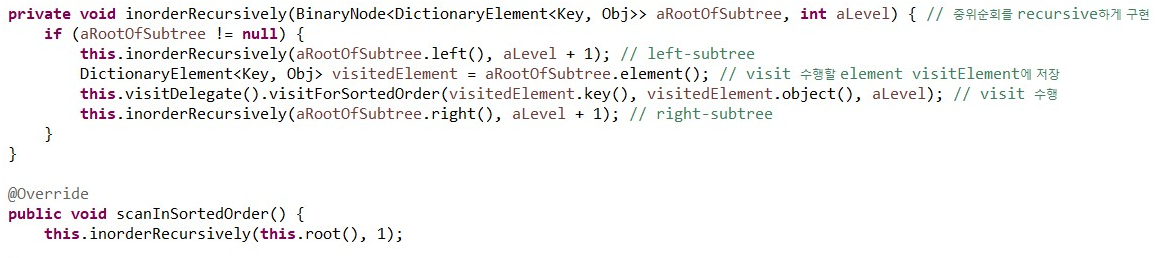
* 프로그램이 실행되는 AppController의 run() 함수.
* 새로운 DictionaryByBST객체 생성한다.
* dictionary 객체에게 VisitDelegate의 주체가 AppController(this)임을 알려준다.



* Visit일을 실행하는 VisitDelegate 인터페이스



* VisitDelegate 인터페이스의 구현은 사용자 객체인 AppController에서 하고있다.
* Visit 했을 때, 형식에 맞는 출력을 한다.

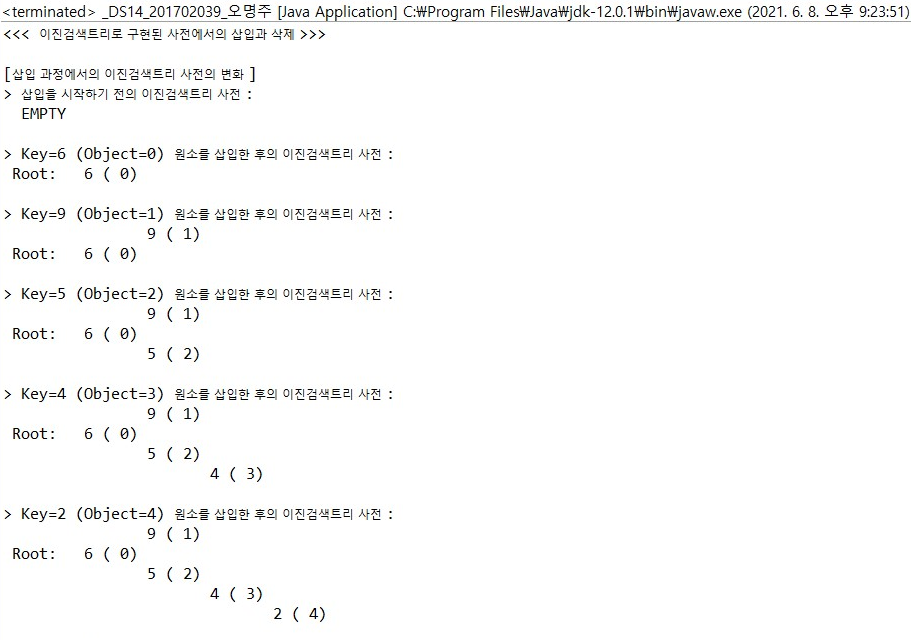


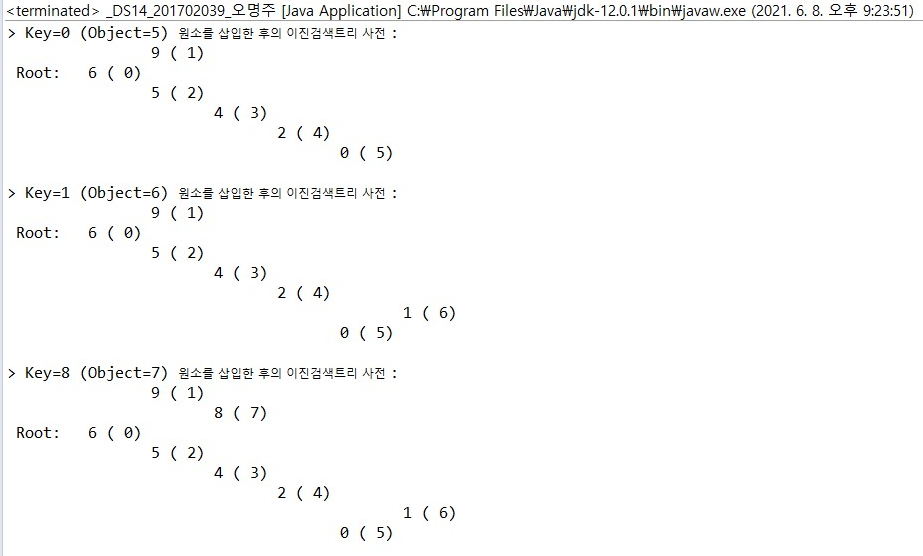
* Visit가 필요한 함수는 AppController에서 호출한 showDictionaryInSortedOrderByCallBack 함수에서 scanInSortedOrder 함수를 호출하면서 호출된다.
* 다시 AppController의 visitForSortedOrder() 함수를 호출하고 있다 => Call Back

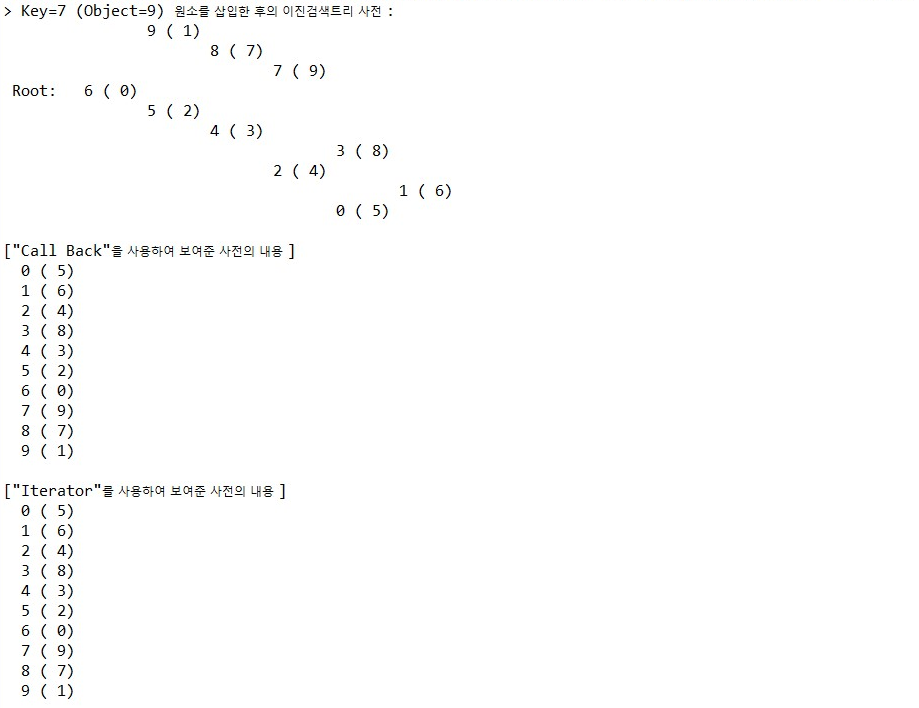
1. **프로그램 장단점 / 특이점 분석**

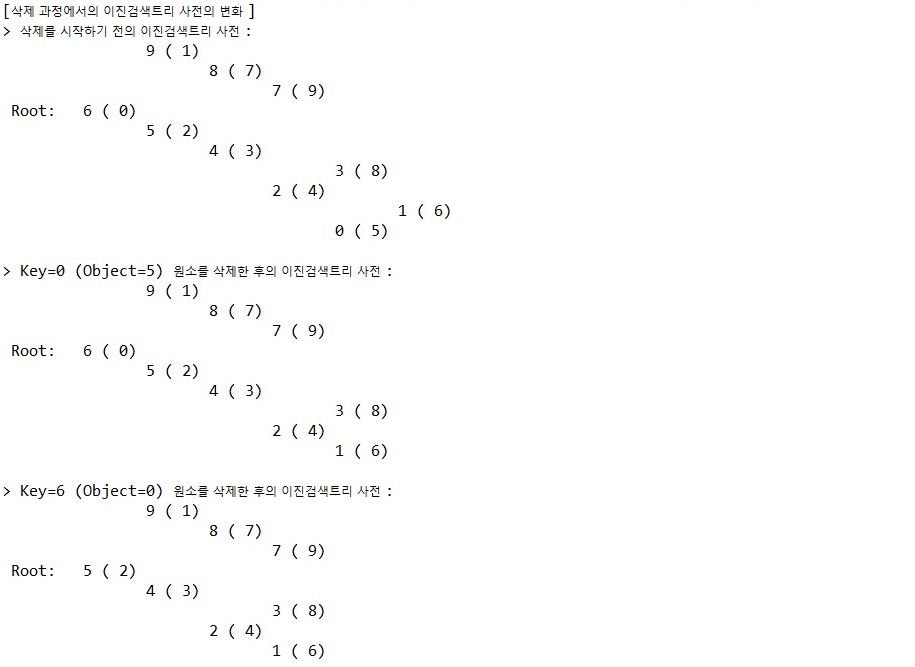
* **장점**
* VisitDelegate를 인터페이스로 만들어 사용자가 Visit함수를 구현할 수 있도록 함으로써 사용자가 Visit함수에 접근하면서도 MVC 구조를 유지할 수 있었다.
* **단점**
* 기능이 세분화 되어있어 코드가 길어질 수 있다.

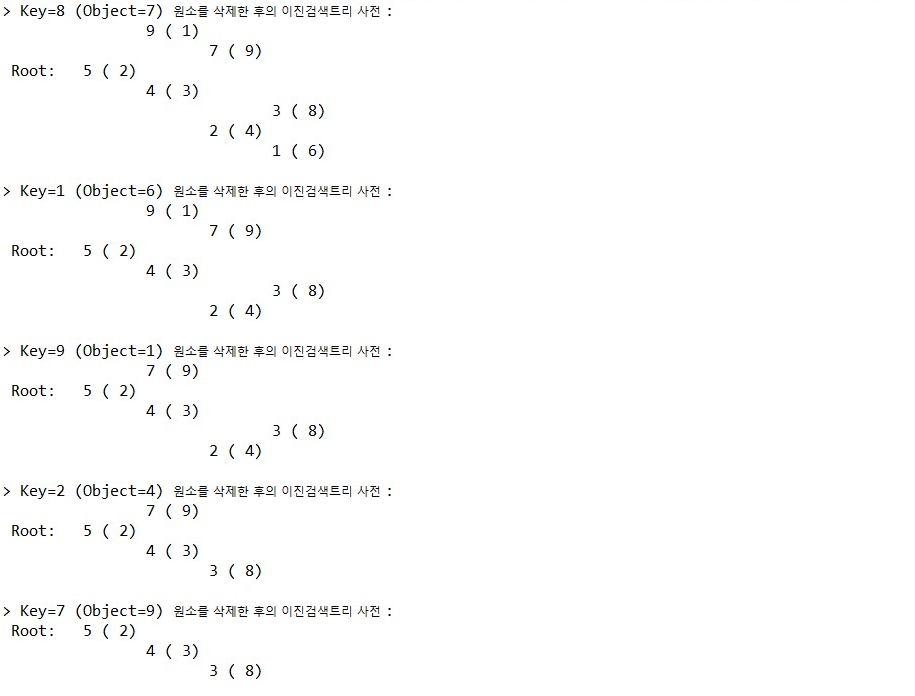
1. **실행 결과 분석**
2. **입력과 출력**

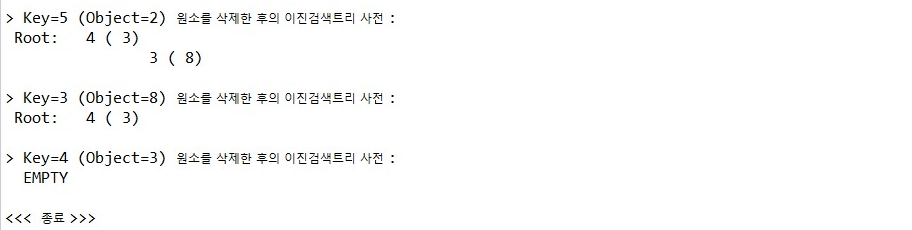












1. **결과 분석 (자신의 논리적 평가, 기타 느낀 점)**

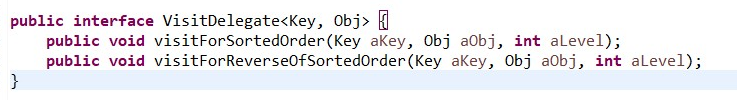
* VisitDelegate 인터페이스를 통해 사용자가 AppController를 통해 구현할 수 있으면서 MVC 중 Model에 해당하는 DictionaryBST에서 Visit를 수행할 수 있도록 구현함으로써 MVC 구조를 사용할 수 있었다.
* 재귀함수를 구현할때마다 느끼는 것이지만 짧은 코드로 구현할 수 있어서 효율적이라고 생각하였다.(구현자 입장에서, 메모리는 아니지만)

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

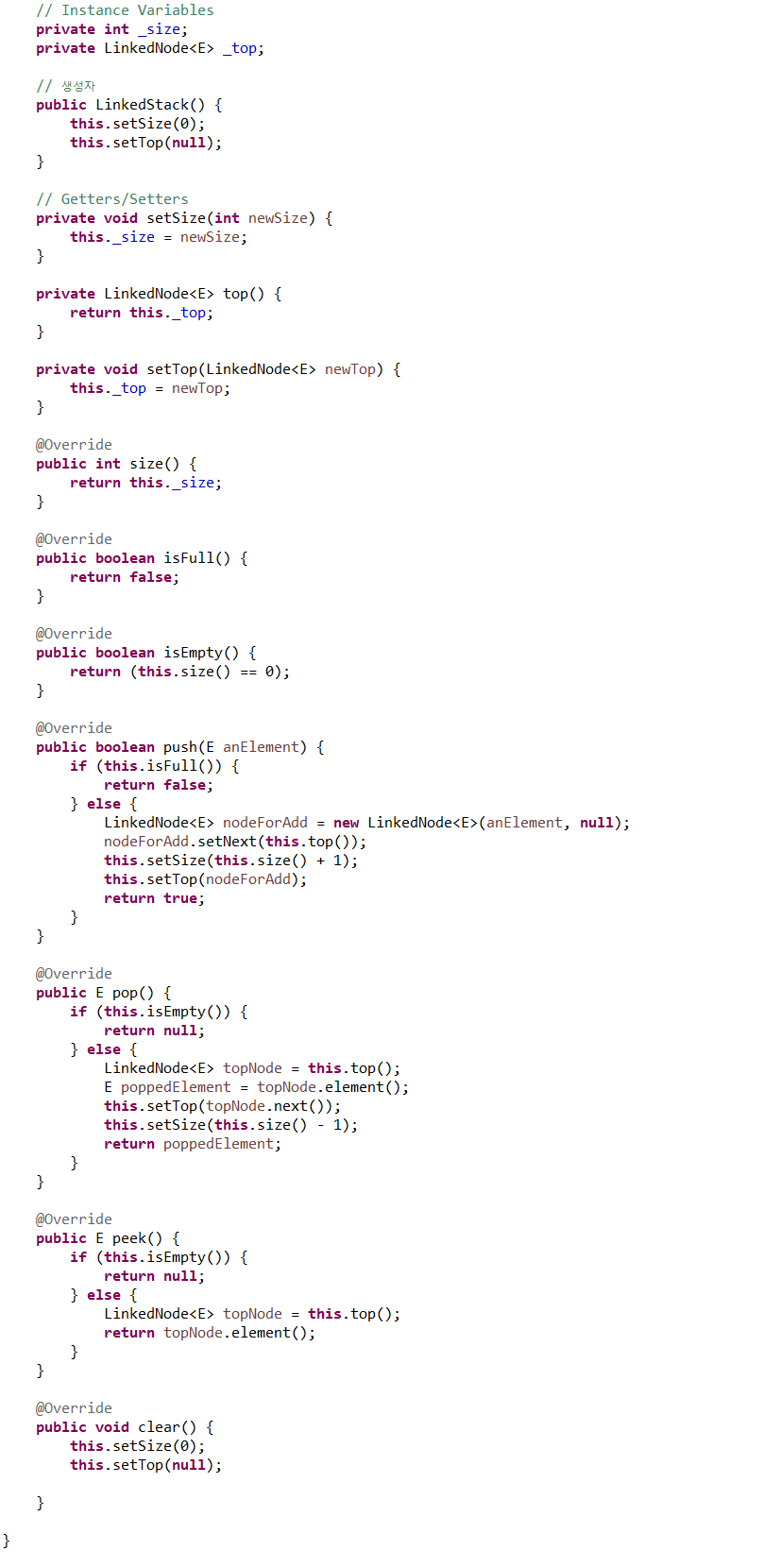
**[AppController]**



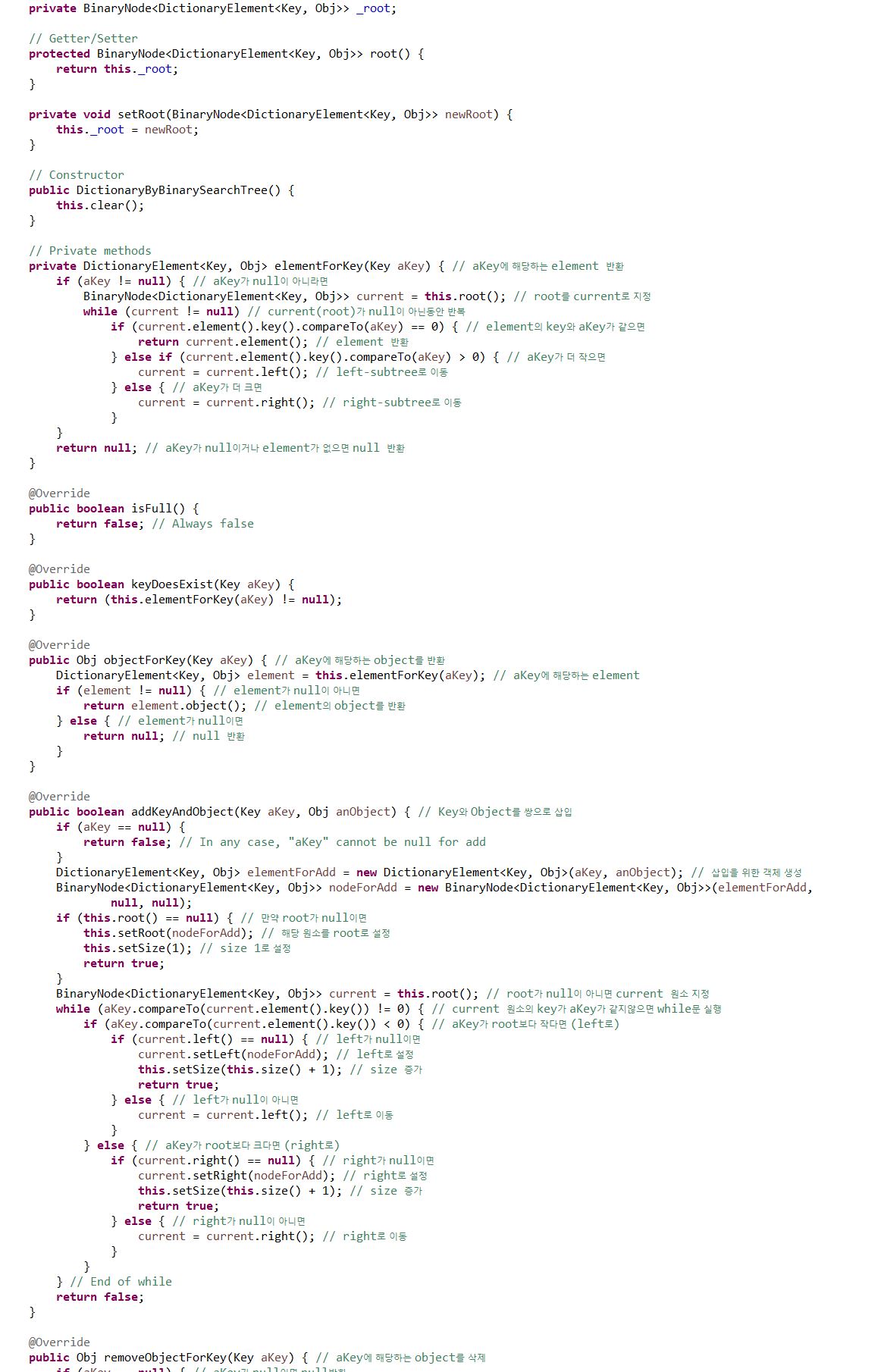
**[Dictionary]**

****

**[VisitDelegate]**



**[LinkedStack]**



**[DictionaryBST]-잘린코드는 코드 참조 바랍니다.**