

# Assignment 3: Binary Search Tree Implementation and Word Tracker

## Description

In this assignment, you'll be provided an ADT for a Binary Search Tree (BSTreeADT.java) and Iterators (Iterator.java) to be implemented as a Binary Search Tree (BST), which you'll then use in an application called WordTracker. This program will read text files, collects and stores all the unique words it finds in those files. The BST will be able to store information from multiple text files and will keep track of each occurrence of a word in a file and the line on which it was found. This information will be stored in a list for each word object stored in the BST. The program will also produce output, specified by the user at command line, to generate reports using a variety of iterators built into the BST.

## Equipment and Materials

For this assignment, you will need:

- Eclipse, Java 8 and JUnit 4

## Instructions

This assignment consists of three parts, to be completed **outside** of class time. See the course outline and Brightspace for due dates. Complete this assignment with your assigned group.

### Part A: Create an Implementation of a BST (90 marks)

Using the specifications below:

1. Create your own implementation of a Binary Search Tree based on the contract in the ADT.
2. Write a Word Tracker program that meets the requirements outlined in the *Program Specifications* section of this document.

### Part B: Complete an Evaluation as a Group (5 marks)

After completing your Word Tracker application, check your work against the provided marking criteria. Your instructor will refer to your group's self-evaluation when grading the assignment and will provide further feedback and grade adjustments as needed. Your instructor is responsible for awarding the group's final grade.

1. Open the Marking Criteria document (MarkingCriteria\_Assignment3.docx) and save a copy with your group's name.
2. As a group, discuss how well you met each criterion and assign yourselves a mark for each row in the table. You may include a short, point form, explanation for your mark in the Notes column.

**Note:** This is an opportunity for you to identify any missing pieces according to the marking criteria, make sure you do this step diligently to avoid losing any marks that is clearly stated in this document.

3. Save this file for submission to Brightspace along with your completed code.

### Part B: Complete a Peer Assessment (5 marks)

Each student must also complete a peer assessment of their group members. Your instructor will provide further submission details.

## Assignment Specifications

**Important:** Read the specifications very carefully. If you are uncertain about any of the requirements, discuss it with your instructor.

1. Import the assignment3StartingCode project into Eclipse.
2. Write an implementation for the utility class **BSTree.java** using the supplied **BSTreeADT.java** and **Iterator.java** interfaces provided by your instructor.
  - a. Your implementation should also include a **BSTreeNode.java** class to store each individual element in the tree.

**Note:** Do NOT modify the ADT provided in any way! Your instructor will test with the default ADT file.

3. Completely test the implementation **BSTree.java** for correct functionality using the set of JUnit tests from the starting code.

**Note:** Do NOT modify the tests provided in any way! Your instructor will test with the default test file.

**Important:** Move to the next step only when you are satisfied that the data structure above perform properly. Then implement your Word Tracker as described below.

4. Write a program called **WordTracker.java**, which performs the following:
  - a. Constructs a binary search tree with all words included from a text file (supplied at the command line)
  - b. Records the line numbers on which these words were used
  - c. Stores the line numbers with the file names, which in turn are associated with the nodes of the tree.
5. Using Java serialization techniques, store the tree in a binary file called **repository.ser**. Make sure you insert the class version UID to ensure backward compatibility with your repository should the class specification change with future enhancements.

6. Every time the program executes:

- a. It should check whether the binary file **repository.ser** exists, and if it does, restore the words tree.
- b. The results of the scanning the next file should be inserted in the appropriate nodes of the tree.

**Note:** Therefore, **repository.ser** will contain all words that occurred in all files scanned, along with the meta information about those word locations.

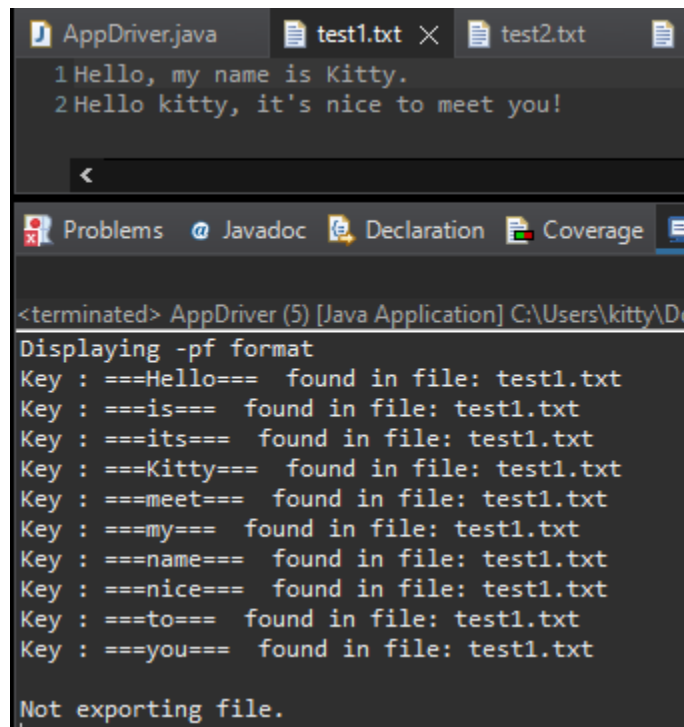
7. The user should be able to run the program via the command prompt as follows:

```
java -jar WordTracker.jar <input.txt> -pf/-pl/-po [-f<output.txt>]
```

- a. <input.txt> is the path and filename of the text file to be processed by the WordTracker program.
  - b. There are three mutually exclusive options at the command line:
    - **-pf** prints in alphabetic order all words, along with the corresponding list of files in which the words occur.
    - **-pl** prints in alphabetic order all words, along with the corresponding list of files and line numbers in which the word occur.
    - **-po** prints in alphabetic order all words, along with the corresponding list of files, line numbers in which the word occur and the frequency of occurrence of the words.
  - c. An optional argument to redirect the report in the previous step to the path and filename specified in **<output.txt>**.
  - d. Assume that the user will provide these arguments in the order shown.
8. You may use any additional Java libraries you wish to implement the word tracking functionality to store the filenames and line numbers (e.g. ArrayList, HashMap).

**Note:** The following images is intended to simply to give you an example of what the input to the application is and what the output would look like – your results will vary depending on the file being parsed. Your program output should be similar but does NOT have to match the exact formatting as long as the relevant information are shown. Your instructor will use a different test data file to evaluate your application.

Here is an example of the program with test1.txt -pf and no existing repo file:



```

AppDriver.java  test1.txt  test2.txt
1Hello, my name is Kitty.
2Hello kitty, it's nice to meet you!

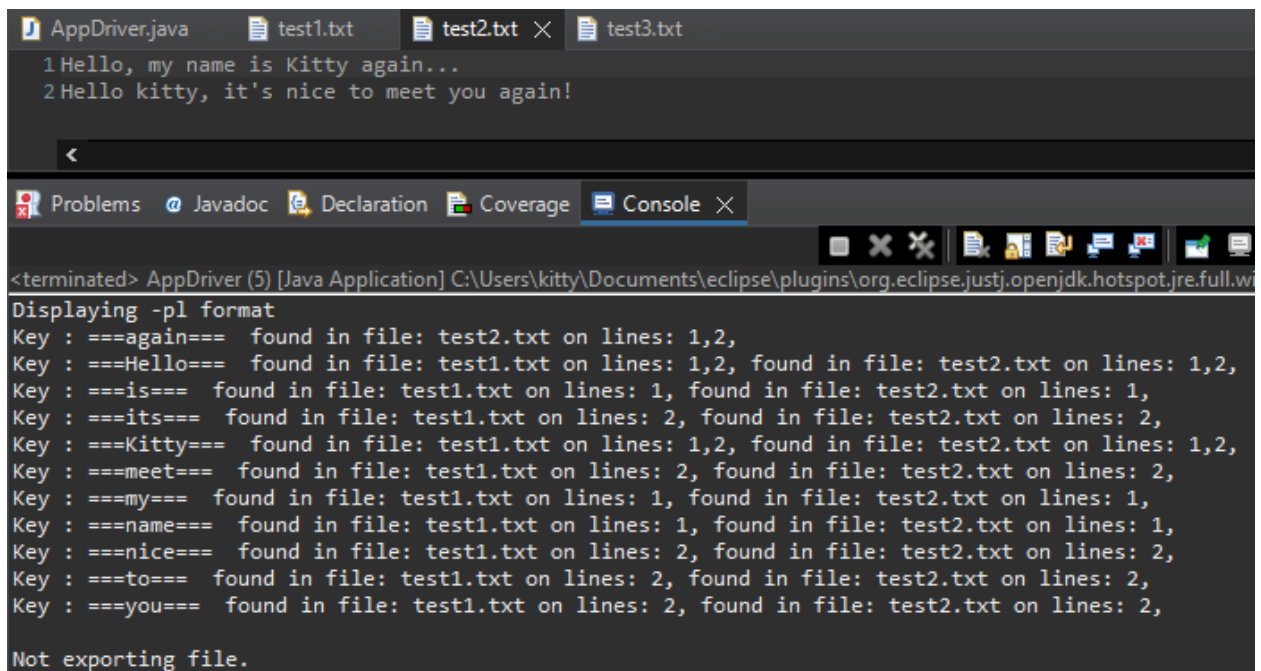
<
Problems  Javadoc  Declaration  Coverage

<terminated> AppDriver (5) [Java Application] C:\Users\kitty\Documents\workspace\SAIT\
Displaying -pf format
Key : ===Hello=== found in file: test1.txt
Key : ===is=== found in file: test1.txt
Key : ===its=== found in file: test1.txt
Key : ===Kitty=== found in file: test1.txt
Key : ===meet=== found in file: test1.txt
Key : ===my=== found in file: test1.txt
Key : ===name=== found in file: test1.txt
Key : ===nice=== found in file: test1.txt
Key : ===to=== found in file: test1.txt
Key : ===you=== found in file: test1.txt

Not exporting file.

```

Then the same program running again with test2.txt -pl and reading from the repo file first:



```

AppDriver.java  test1.txt  test2.txt  test3.txt
1Hello, my name is Kitty again...
2Hello kitty, it's nice to meet you again!

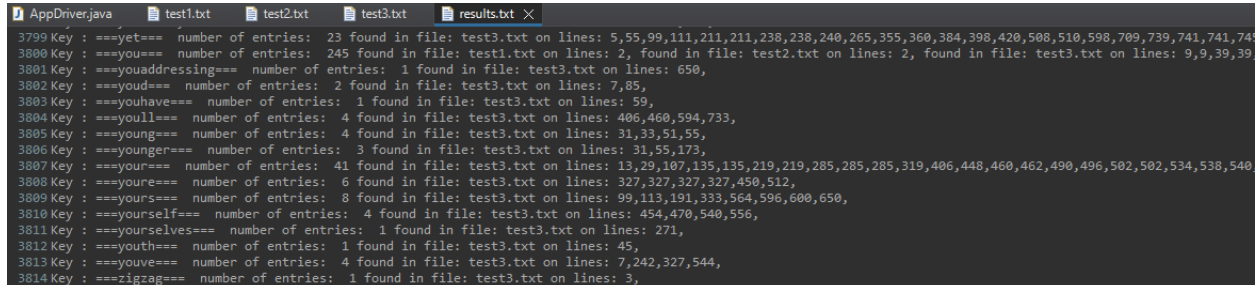
<
Problems  Javadoc  Declaration  Coverage  Console

<terminated> AppDriver (5) [Java Application] C:\Users\kitty\Documents\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.wi
Displaying -pl format
Key : ===again=== found in file: test2.txt on lines: 1,2,
Key : ===Hello=== found in file: test1.txt on lines: 1,2, found in file: test2.txt on lines: 1,2,
Key : ===is=== found in file: test1.txt on lines: 1, found in file: test2.txt on lines: 1,
Key : ===its=== found in file: test1.txt on lines: 2, found in file: test2.txt on lines: 2,
Key : ===Kitty=== found in file: test1.txt on lines: 1,2, found in file: test2.txt on lines: 1,2,
Key : ===meet=== found in file: test1.txt on lines: 2, found in file: test2.txt on lines: 2,
Key : ===my=== found in file: test1.txt on lines: 1, found in file: test2.txt on lines: 1,
Key : ===name=== found in file: test1.txt on lines: 1, found in file: test2.txt on lines: 1,
Key : ===nice=== found in file: test1.txt on lines: 2, found in file: test2.txt on lines: 2,
Key : ===to=== found in file: test1.txt on lines: 2, found in file: test2.txt on lines: 2,
Key : ===you=== found in file: test1.txt on lines: 2, found in file: test2.txt on lines: 2,

Not exporting file.

```

Then the same program running again with `test3.txt -po -fresults.txt` and reading from the repo file first and redirecting the output to a file called `results.txt`: (Note: the following image does not show the entire file)



```
3799 Key : ===yet=== number of entries: 23 found in file: test3.txt on lines: 5,55,99,111,211,211,238,238,240,265,355,360,384,398,420,508,510,598,709,739,741,741,741
3800 Key : ===you=== number of entries: 245 found in file: test1.txt on lines: 2, found in file: test2.txt on lines: 2, found in file: test3.txt on lines: 9,9,39,39
3801 Key : ===youaddressing=== number of entries: 1 found in file: test3.txt on lines: 650,
3802 Key : ===youad=== number of entries: 2 found in file: test3.txt on lines: 7,85,
3803 Key : ===youhave=== number of entries: 1 found in file: test3.txt on lines: 59,
3804 Key : ===youll=== number of entries: 4 found in file: test3.txt on lines: 406,460,594,733,
3805 Key : ===young=== number of entries: 4 found in file: test3.txt on lines: 31,33,51,55,
3806 Key : ===younger=== number of entries: 3 found in file: test3.txt on lines: 31,55,173,
3807 Key : ===your=== number of entries: 41 found in file: test3.txt on lines: 13,29,107,135,135,219,219,285,285,285,319,406,448,460,462,490,496,502,502,534,538,540
3808 Key : ===youre=== number of entries: 6 found in file: test3.txt on lines: 327,327,327,327,450,512,
3809 Key : ===yours=== number of entries: 8 found in file: test3.txt on lines: 99,113,191,333,564,596,600,650,
3810 Key : ===yourself=== number of entries: 4 found in file: test3.txt on lines: 454,470,540,556,
3811 Key : ===yourselves=== number of entries: 1 found in file: test3.txt on lines: 271,
3812 Key : ===youth=== number of entries: 1 found in file: test3.txt on lines: 45,
3813 Key : ===youve=== number of entries: 4 found in file: test3.txt on lines: 7,242,327,544,
3814 Key : ===zigzag=== number of entries: 1 found in file: test3.txt on lines: 3,
```

## Submission Deliverables

**Note:** This is a group assessment, so only one submission is required per group. If there are multiple submissions, your instructor will only evaluate the last submission. See Brightspace for the exact due date and time.

Your group's submission should include a zipped folder named as the assignment number and your group name (e.g., A3Zelda.zip). The zipped folder should contain the following items:

- 1) An executable Java Archive file (.jar) for your sorting application called **WordTracker.jar**.
- 2) A **readMe.txt** file with instructions on how to install and use the Word Tracker program.
- 3) The project should have completed javadoc using the "-private" option when generated, and the output placed in the **doc** directory of the project.
- 4) A folder containing the complete Eclipse project directory. Refer to the exact instructions of how to export your project from Lab 0.
- 5) The completed Marking Criteria document containing your group's evaluation of your application. All group members should be present when completing this document and it acts as your "sign-off" (approval) of the assignment submission.

**No late assignments will be accepted.**