Joanna Klukowska
joannakl@cs.nyu.edu

Programming Project 5:
Find Words

Due date: Dec 2, 11:55 PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own. Any sharing or copying of assignments will be considered cheating. If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures or features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will revisit your project 2 implementation of a program that produces all possible words given a set of letters and a dictionary. Your program should produce **all** anagrams of all different lengths of a given set of letters that are valid words in the provided language dictionary. For example, if the letters are

```
recounts
```

your program should print, in alphabetical order and one per line, all different words that can be made out of those letters (assuming that they are present in the dictionary that the program uses):

```
construe
counters
recounts
```

The program that you write has to be command line based (no graphical interface). It should use the dictionary from a file provided as a command line argument and it should prompt the user for a set of letters. The output should be printed to the terminal. Once the anagrams are displayed, the program should terminate.

Your program should use the implementation provided with this assignment. You should make changes only to the `Dictionary` class and provide classes that are required by the `Dictionary` class.

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- using/modifying existing code
- implementation of a generic binary search tree
- extending existing classes
- writing Java programs

The Program Input and Output

You should not change this part of the implementation.

Joanna Klukowska
joannakl@cs.nyu.edu

Computational Task

Once the user has entered the letters, the program displays all the words in the dictionary that can be formed as combinations of all the letters entered by the user.

Creating all Possible Words

The task of creating possible words should be achieved recursively using the backtracking technique.

The **`LetterBag`** class that performs this computation is provided for you and you should not be making any changes to it. Your **`Dictionary`** class need to provide all the tools required by this class to work properly.

Searching in the Dictionary

In order to determine if a given sequence is a valid word in a dictionary, your program needs to perform searches. As before, you need provide a **`Dictionary`** class that allows to search for particular words and for the prefixes to words.

Program Design

Your program must contain at least four classes (some of them are provided):

- **`BST <E>`** class that implements a binary search tree. All of its methods should be implemented recursively (adding, removing, searching).

  The **`BST <E>`** class should implement the following interface:

```
public interface BSTInterface <E extends Comparable <E> > {
  /**
    * Adds an item to this binary search tree.
    * @param item the item to be added
    */
  void insert( E item );

  /**
    * Removes an item from this binary search tree.
    * If item is not in the tree, the structure is unchanged.
    * @param item the item to be removed
    */
  void remove ( E item );

  /**
    * Determines if an item is located in this binary search tree.
    * @param item the item to be located
    * @return true if the item is in the tree, false otherwise
    */
  boolean contains ( E item );
}
```

  You will to define your own generic node class for this **`BST <E>`** class.

- **`Dictionary`** class to represents the collection of words read in from the input file (i.e., the dictionary used by the program). This class is responsible for performing queries in the dictionary.

  The **`Dictionary`** class should implement the following interface:

Joanna Klukowska
joannakl@cs.nyu.edu

```
public interface DictionaryInterface {
  /**
    * This method determines if a given word
    * is in this Dictionary.
    * @param word the word to be checked
    * @return true of the word is in this Dictionary,
    * false otherwise
    */
  boolean findWord ( String word );

  /**
    * This method determines if a given
    * prefix is a prefix of a word that exists in
    * this Dictionary.
    * @param prefix the prefix to be checked
    * @return true if the prefix is in this Dictionary,
    * false otherwise
    */
  boolean findPrefix ( String prefix );
}
```

The **Dictionary** class should extend the **BST<String>** class. This means that words should be stored in a binary search tree, not an **ArrayList**.

- **LetterBag** class to represent the letters entered by the user/player. This class is given and you should not make any changes to it.
- **FindWords** class that is the runnable program containing the **main()** method. This class is given and you should not make any changes to it.

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at http://cs.nyu.edu/~joannakl/cs102_f15/notes/CodeConventions.pdf.

You must document all your code using Javadoc. Your class documentation needs to provide a description of what it is used for and the name of its author. Your methods need to have description, specification of parameters, return values, exceptions thrown and any assumptions that they are making.

A class's data fields and methods should not be declared **static** unless they are to be shared by all instances of the class or provide access to such data.

You must use the provided classes and modify only the implementation of the **Dictionary** class.

Grading

Make sure that you are following all the rules in the **Programming Rules** section above.

If your program does not compile or crashes (almost) every time it is ran, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

30 points design, implementation and correctness of the **Dictionary** class,

40 points design, implementation and correctness of the **BST<E>** class and its node,

10 points use of existing classes,

Joanna Klukowska
joannakl@cs.nyu.edu

20 points proper documentation and program style.

How and What to Submit

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to NYU Classes. If you are unsure how to create a zip file or how to get to your source code files, you should ask long before the project is due.

If you wish to use your (one and only) freebie for this project (one week extension, no questions asked), then complete the form at `http://goo.gl/forms/YFDVBlscEB` **before the due date for the assignment**. All freebies are due seven days after the original due date and should be submitted to NYU Classes.