COSC 310 – Data Structures and Algorithms
Assignment 7 - Binary Search Trees

Due: November 9th at 2:00 am - 40 points.

The objectives of this assignment are to:

1) Gain further experience with using interfaces.
2) Gain further experience with recursion.
3) Gain understanding and experience with implementations of a binary search tree.
4) Gain experience with dictionary applications.
5) Continue to practice good programming techniques.

Task:

Step 1:

Create a binary search tree implementation of a `Dictionary`. The class must be named `BSTDictionary` where `Dictionary` is defined as:

```
/**
 * An abstract data type for a Dictionary that maps a set of keys to values.
 *
 * @author dtsmith
 *
 * @param <K> Data type for the keys
 * @param <V> Data type for the values
 */
interface Dictionary<K, V> {
	/**
	 * Put a key together with its associated value into the dictionary.
	 * If the key already exists then the new value replaces the current
	 * value associated with the key. Values can be retrieved using the
	 * get method.
	 *
	 * @param key the key
	 * @param value the new value
	 * @return the original value if the key already exists in the
	 *         dictionary, otherwise null.
	 */
	public V put(K key, V value);

	/**
	 * Get the current value associated with a given key.
	 *
	 * @param key the key
	 * @return the current value associated with the key in the dictionary
	 *         if found, otherwise null.
	 */
	public V get(K key);

	/**
	 * Remove the key and its associated value associated from the
	 * dictionary. The value associated with the key is returned.  If the
	 * key does not exist in the dictionary then null is returned.
	 *
```

```
         * @param key the key
         * @return the value associated with the removed key in the dictionary.
         *         If the key did not exist then null.
         */
        public V remove(K key);

        /**
         * Create an Iterator to iterate over the keys of the dictionary.
         *
         * @return an Iterator to iterator over the keys.
         */
        public Iterator<K> keys();

        /**
         * Test if the dictionary is empty
         *
         * @return true if the dictionary is empty, otherwise false
         */
        public boolean isEmpty();

        /**
         * Get the number of keys in the dictionary
         *
         * @return the number of keys in the dictionary
         */
        public int noKeys();
}
```

Your implementation of BSTDictionary must not use an inner/nested class of Nodes. Instead define your `BSTDictionary` to have a left and right instance variables that are each a `BSTDictionary`. This approach is going to be analogous to the Ls implementation of a linked list. When a `BSTDictionary` is created, it will be empty and its left and right instance variables must be null. The empty `BSTDictionary` is similar in purpose to the empty Ls at the end of an Ls list. On adding a key and value to an empty `BSTDictionary`, set the key and value instance variables accordingly and create an empty `BSTDictionary` for the left and create an empty `BSTDictionary` for the right. Note that each `BSTDictionary` must keep track of the number of keys in the given `BSTDictionary` including all descendants. Given this an empty `BSTDictionary` will have zero for the number of keys.

Note that keys() returns a Java Iterator, however you only need to implement the hasNext() and next() methods.

Step 3:

Create a jUnit test program to thoroughly test your `BSTDictionary`.

Step 4:

Create a word count program (**not in edu.iup.cosc310.util**). The word count program must open a text file passed to it on the command line. Your program must process the text file identifying the set of unique words in the file and the number of occurrences of each word. Separate words on spaces, commas, tabs, newlines, carriage returns, and other not alphanumeric characters (i.e. split on "\\W+").

After processing all the words in the input text file, your word count program must print a list of all the found words, in sorted order, together with the number of times each word was found within the file. The output is to be is sorted word order.

Deliverables:

1) A. zip file of your project. It will contain the sources and class files of your implementation, test, and word count program. The .zip file must be named [Your name]Asmt7.zip.
2) Printouts of source files for your interface, binary search implementation, test program, and word count program.
3) Screen shot of your captured output from executing your jUnit test program.
4) Text frle your captured output from executing your word count program against:
   supplied sylabus.txt.

Other requirements:

1) The iterator must return the keys in an in order traversal. The iterator implementation must not copy keys into a data structure holding all keys.
2) The `Dictionary` and `BSTDictionary` must be defined in package edu.iup.cosc310.util.
3) The word count program must be defined in package edu.iup.cosc310.wordcount
4) **Must provide JavaDoc**