Document Scanner

Due: November 14, 2021

Instructions: Create a document parser that will read and count all the words of a given document. The words in the document will be delimited by spaces " ", and sentences by periods ".". Once the document is read, the words, along with their counts, can be searched and printed out. The underlying data structure should be a Binary Search Tree (BST) that uses the provided hash generator for its keys. Words should be stored in the BST without spaces (“ “) or periods (“.”).

```
public int HashGenerator(String key) {
        double hash = 0.0;
        int base;
        int power;
        for (int i=0; i<key.length(); i++) {
                base = (int)key.charAt(i)-96;
                power = (i==0)?3:2;
                hash += Math.pow(base, power)*Math.PI/(i+1);
        }
        return (int)(hash*Math.E*5/key.length());
}
```

Notes:
1. All class members should be private. Accessor/Mutator methods should be used to access class members. There will be a 3 point deduction for each infraction.
2. Properly document each method and class. There will be point deductions for poorly documented code.

Create the following classes and methods

DocumentScannerDemo: Use the provided DocScannerDemo.java class to demo your DocumentScanner class. A sample output is provided.

DocumentScanner: Uses the BinarySearchTree class as the underlying data structure to store words		
Constructor	initializes the class	1 pts
ScanDocument(filename)	Scans and parses a given document (text file) and builds a binary search tree based on each individual word using BST `insert(key)` method hint: Read https://javacodex.com/Files/Read-File-Word-By-Word and use "[;\\r\\n]+" as the delimeter	5 pts
Search(word)	Searches for a word (using the BST class’ search) returns how many times it appears in the text.	2 pts
Delete(word)	Delete’s a word (using the BST class’ delete) if the word is found	1 pts
PrintMaxWord()	Searches for the word with the highest count and prints it along with its count	2 pts
PrintPreorder	Uses the BST `Print_Preorder` method to print the tree	1 pts
PrintInorder	Uses the BST `Print_Inorder` method to print the tree	1 pts
PrintPostorder	Uses the BST `Print_Postorder` method to print the tree	1 pts

BinarySearchTree: Uses the BSTNode class as the underlying data structure		
Constructor(NewNode)	initializes the class using the given node	2 pts
Insert(ParentNode, NewNode)	inserts a node (newnode) into the binary search tree using recursion	6 pts
Insert(key)	Uses the previous Insert function to insert a key into the binary search tree	1 pts
Node FindParent(key)	searches for and returns a node in the binary search tree that is the potential parent of a given key. Returns null if no parent is found	6 pts
Node Search(key)	searches for and returns a node in the binary search tree with the given key if found. Returns null if not found.	6 pts
Delete(key)	deletes a node from the binary search tree given a search key	12 pts
Print_Preorder	Prints the tree nodes in preorder format	2 pts
Print_Inorder	Prints the tree nodes in inorder format	2 pts
Print_Postorder	Prints the tree nodes in postorder format	2 pts

Suggested Helper functions (Optional)	
Node findSmallestRightChild (Node subtreeroot)	finds and returns the smallest right child of a subtree
Node pruneSmallestRightChild (Node subtreeroot)	prunes and returns the smallest right child of a subtree

BinarySearchTreeNode: Binary search tree node that holds data		
Constructor	initializes the class	1 pts
SetValue(key)	sets the value of the current node	1 pts
SetLeftChild(Node)	sets the left child pointer of the current node	1 pts
SetRightChild(Node)	set the right child pointer of the current node	1 pts
String GetValue	returns the value of the current node	1 pts
Node GetLeftChild()	returns the left child pointer of the current node	1 pts
Node GetRightChild()	returns the right child pointer of the current node	1 pts

Suggested Helper function (Optional)	
int GetNumberofChildren()	Returns the number of children a node has