Project

ICS 202

Jalal Ali Zainaddin | 202154790 | 11/22/2023

Objectives:

1- Create a Dictionary class
2- Creating four methods: addWord(), removeWord(), findSimilar(), findWord()
3- Apply changes to the dictionary file

Task 1: Initialize a dictionary:

We have three options:

1- Initialize a dictionary with no words
2- Initialize a dictionary with a word
3- Initialize a dictionary with a given file

In order to do that, we created a simple menu inside the main class to let the user choose one of these options, then using the switch statement according to the option chose.

Code:

```java
public class Main {
    no usages
    public static void main(String[] args) {
        Dictionary dict = new Dictionary();

        System.out.println("Choose one of Three options to initialize: ");
        System.out.println("1- Empty dictionary");
        System.out.println("2- Dictionary with initial String");
        System.out.println("3- Dictionary from file");

        Scanner input = new Scanner(System.in);
        String inputValue = input.next();
        // flag for second menu
        boolean flag = true;
        // initial menu
        switch(inputValue) {
            case "1":
                System.out.println("Empty dictionary created successfully");
                dict = new Dictionary();
                break;
            case "2":
                System.out.println("Enter an initial string: ");
                dict = new Dictionary(input.next());
                System.out.println("Dictionary with initial string created successfully");
                break;
            case "3":
                System.out.println("Enter a file name: ");
                try {
                    dict = new Dictionary(new File( pathname: "/src/" + input.next()));
                    System.out.println("Dictionary with a file created successfully");
                }
                catch (IOException e){
                    System.out.println("File not found");
                    flag = false;
                }
                break;
            default:
                System.out.println("Please enter a number from 1-3");
        }
```

Task 2: public void addWord(String s) throws WordAlreadyExistsException:

Using AVL insert method, which basically uses a normal insert from BST tree and balance it by using AVL rotations and height as comparison, because the BST throws IllegalArgumentException if the key is already found, we used try catch block, when it catches, it throws WordAlreadyExistsException, which will be caught in the main class.

Code:

```
public void addWord(String s) throws WordAlreadyExistsException{
    try{
        // Using AVL insert
        super.insertAVL(s);
    }
    catch (IllegalArgumentException ex){
        // throw to the main class
        throw new WordAlreadyExistsException();
    }
}
```

Task 3: public boolean findWord(String s):

Using BST search, which returns true if found and false if not, and basically its algorithm is when the given element is lesser that current, go to left side of current, else right until it is found or not found by reaching the end of the tree.

Code:

```
public boolean findWord(String s){
    // BST search
   return super.search(s);
}
```

Task 4: public void deleteWord(String s) throws WordNotFoundException:

Using delete AVL, which is simply using BST delete by copying but with balance after it, in delete by copying from BST, it throws two exceptions when happen, first if the key is not found, it throws NoSuchElementException(), or if tree is empty, it throws UnsupportedOperationException(). So when there's no such element, we catch it then we throw WordNotFoundException(), if the tree is already empty, we catch the exception and print "Dictionary is empty".

Code:

```
public void removeWord(String s) throws WordNotFoundException{
    try {
        super.deleteAVL(s);
        System.out.println("word removed successfully");
    }
    catch (NoSuchElementException noSuchElementException) {
        throw new WordNotFoundException();
    }
    catch (UnsupportedOperationException unsupportedOperationException){
        System.out.println("Dictionary is empty");
    }
}
```

Task 5: public String[ ] findSimilar (String s):

We create a stack called 'words' that contains the words that matches the condition, then we create a queue to traverse through the whole tree while the queue is not empty, each time we add left or right of the current element if they exist "not null", and we take a current node's data to a different method called "areWordsSimilar" which runs through multiple conditions and returns true if they are similar:
1 If they are the same, return false

2- If the difference in length is bigger than one, return false

3-else, we go to for loop, starting with the first character, if both characters between the two are same, go to next next character, if different, we have three situations:

- **a- If both words have the same length, compare all next characters of both by using substring, if they are the same, return true**
- **b- If they have different lengths, use similar algorithm but will start from the next character for the longer one.**

Code:

```
public String[] findSimilar (String s){
    Stack<String> words = new Stack<>();
    if (root == null)
        return new String[]{};
    Queue<BTNode<String>> queue = new Queue<>();
    queue.enqueue(root);
    while(!queue.isEmpty()){
        BTNode<String> current = queue.dequeue();

        if (areWordsSimilar(s,current.data))
            words.push(current.data);

        if (current.left != null)
            queue.enqueue(current.left);
        if (current.right != null)
            queue.enqueue(current.right);

    }

    String[] similar = new String[words.size()];
    for (int i = 0; i < similar.length; i++) {
        similar[i] = words.pop();
    }
    return similar;
}
```

Caption: findSimilar method

```
private static boolean areWordsSimilar(String s1, String s2){
    // check if both are the same
    if (s1.equals(s2))
        return false;

    // Check if the absolute difference in lengths is at most 1
    if (Math.abs(s1.length() - s2.length()) > 1) {
        return false;
    }

    // Iterate through the characters of the shorter string
    int minLength = Math.min(s1.length(), s2.length());
    for (int i = 0; i < minLength; i++) {
        // Check if the characters at the same position are different
        if (s1.charAt(i) != s2.charAt(i)) {
            // If the lengths are equal, check the rest of the strings
            if (s1.length() == s2.length()) {
                return s1.substring( beginIndex: i + 1).equals(s2.substring( beginIndex: i + 1));
            }

            // If s1 is longer, check the rest of s1 starting from the next character
            if (s1.length() > s2.length()) {
                return s1.substring( beginIndex: i + 1).equals(s2.substring(i));
            }

            // If s2 is longer, check the rest of s2 starting from the next character
            return s1.substring(i).equals(s2.substring( beginIndex: i + 1));
        }
    }
    // If we reach here, the words are similar
    return true;
}
```

Caption: areWordsSimilar method

Task 6: Save the updated dictionary as a text file:

Implementing "public void saveWords(String s)" method, the user can create a new file or overwrite by inputting the new file name in the main method, then we call helper method, which basically adds the AVL tree root in order to traverse. We use a similar technique to inorderTraversal in Binary Tree class to write it in original order, and we used FileWriter class to write the file.

Code:

```
public void saveWords(String s) {
    File file = new File(s);
    // using try catch because FileWriter requires it
    try {
        FileWriter fileWriter = new FileWriter(file);
        saveWords(fileWriter, super.root);
        fileWriter.close();
        System.out.println("Dictionary saved successfully.");
    }
    catch (IOException ioException){
        System.out.println("File not written successfully");
    }
}
3 usages
private void saveWords(FileWriter file, BTNode<String> s){
    if (s == null)
        return;
    try {
        saveWords(file,s.left);
        file.write( str: s.data+"\n");
        saveWords(file,s.right);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Task 7: test the program:

We created a small menu in a main method as a gateway to use all methods discussed.

Code:

```
inputValue = input.next();
while (!inputValue.equals("q")) {
    switch (inputValue) {
        case "1":
            System.out.println("check word> ");
            if (dict.findWord(input.next()))
                System.out.println("word found successfully\n");
            else
                System.out.println("word not found\n");
            break;
        case "2":
            System.out.println("add new word> ");
            try {
                dict.addWord(input.next());
                System.out.println("word added successfully\n");
            }
            catch (WordAlreadyExistsException wordAlreadyExistsException){
                System.out.println("WORD ALREADY EXISTED\n");
            }
            break;
        case "3":
            System.out.println("remove word> ");
            try {
                dict.removeWord(input.next());
                System.out.println();
            }
            catch (WordNotFoundException wordNotFoundException){
                System.out.println("word not found\n");
            }
            break;
        case "4":
            System.out.println("search for similar words> ");
            System.out.println(Arrays.toString(dict.findSimilar(input.next()))+"\n");
            break;
        case "5":
            System.out.println("Save Updated Dictionary (Y/N)> ");
            if (input.next().equals("Y")) {
                System.out.println("Enter Filename> ");
                dict.saveWords(input.next());
                System.out.println();
            }
            break;
        default:
            System.out.println("Please enter a number from 1-3");
    }
    System.out.println("Choose Five options to initialize: ");
    System.out.println("1- Find a word in the dictionary");
    System.out.println("2- Add a word to the dictionary");
    System.out.println("3- Remove a word to the dictionary");
    System.out.println("4- Search for similar words to a word in the dictionary");
    System.out.println("5- Save the updated dictionary as a text file");
    System.out.println("press 'q' to quit");
    inputValue = input.next();
}
```

Output:

```
Choose one of Three options to initialize:
1- Empty dictionary
2- Dictionary with initial String
3- Dictionary from file
2
Enter an initial string:
This
Dictionary with initial string created successfully
Choose Five options to initialize:
1- Find a word in the dictionary
2- Add a word to the dictionary
3- Remove a word to the dictionary
4- Search for similar words to a word in the dictionary
5- Save the updated dictionary as a text file
press 'q' to quit
2
add new word>
is
word added successfully
```

```
1
check word>
university
word not found

Choose Five options to initialize:
1- Find a word in the dictionary
2- Add a word to the dictionary
3- Remove a word to the dictionary
4- Search for similar words to a word in the dictionary
5- Save the updated dictionary as a text file
press 'q' to quit
2
add new word>
university
word added successfully
```

```
2
add new word>
mis
word added successfully

Choose Five options to initialize:
1- Find a word in the dictionary
2- Add a word to the dictionary
3- Remove a word to the dictionary
4- Search for similar words to a word in the dictionary
5- Save the updated dictionary as a text file
press 'q' to quit
4
search for similar words>
is
[mis]
```

```
5
Save Updated Dictionary (Y/N)>
Y
Enter Filename>
test_project
Dictionary saved successfully.

Choose Five options to initialize:
1- Find a word in the dictionary
2- Add a word to the dictionary
3- Remove a word to the dictionary
4- Search for similar words to a word in the dictionary
5- Save the updated dictionary as a text file
press 'q' to quit
q

Process finished with exit code 0
```

Conclusion:

In conclusion, this course is about learning very simple data structures such as linked list to very important ones that are used in many fields such as trees and hash table for data, graph for finding a fastest path from place to another and so on. It was enjoyable and challenging. This project used one of the structures called AVL tree, which is way more efficient than a normal array or list.