

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

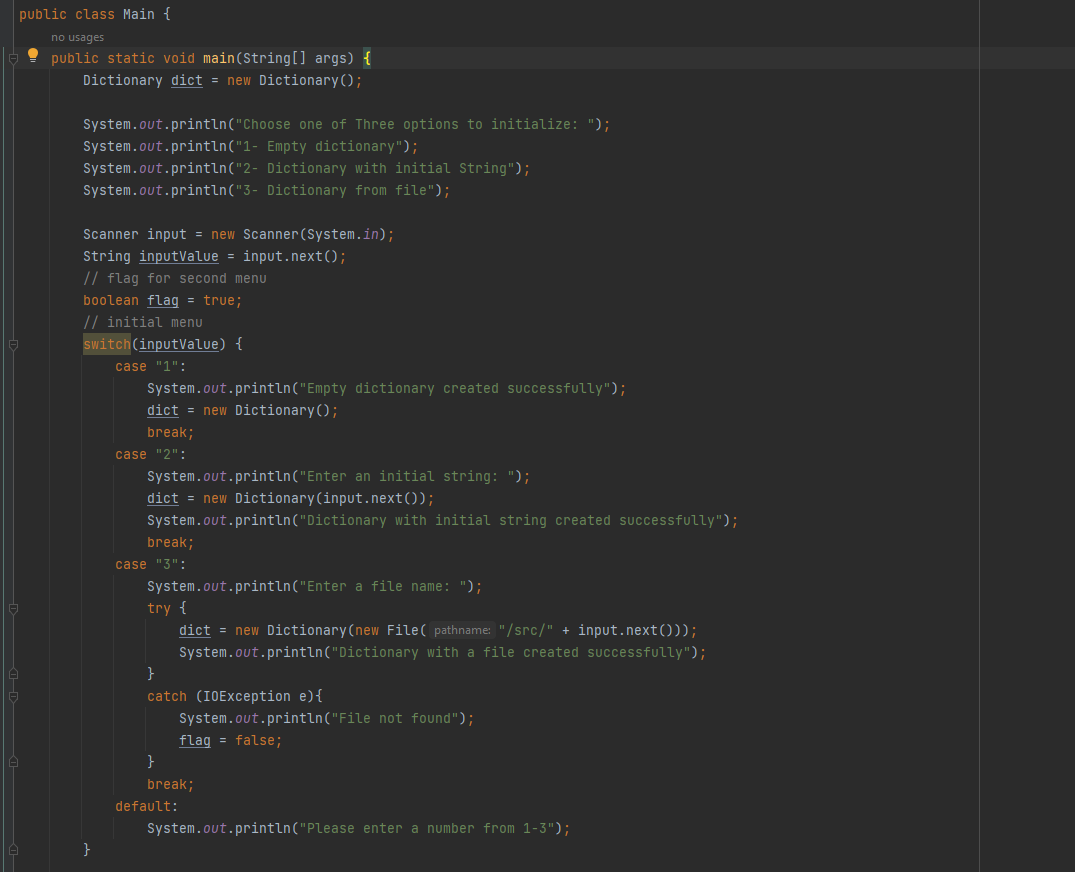
# Task 1: Initialize a dictionary:

**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:



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:

A screen shot of a computer program

Description automatically generated

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:

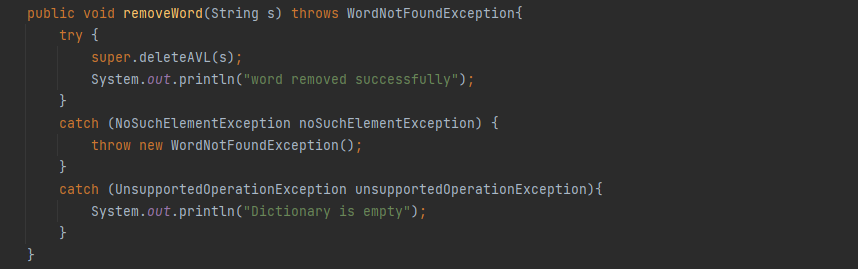
A computer screen shot of text

Description automatically generated

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:



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

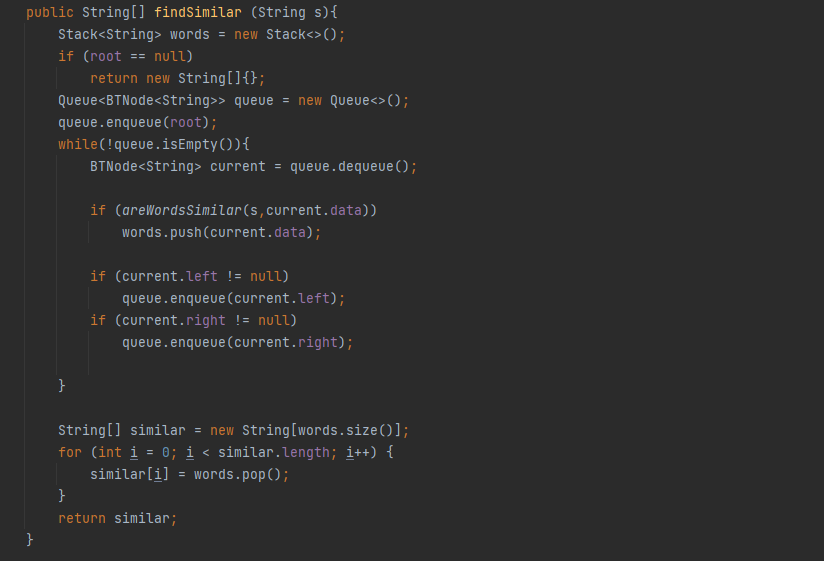
**We create a stack called ‘words’ that contains the words that matches the condition, then we create a queue to traverse through the whole tree while the queue is not empty, each time we add left or right of the current element if they exist “not null”, and we take a current node’s data to a different 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:**

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

Code:



Caption: findSimilar method

A screenshot of a computer program

Description automatically generated

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:

**A screen shot of a computer program

Description automatically generated**

Task 7: test the program:

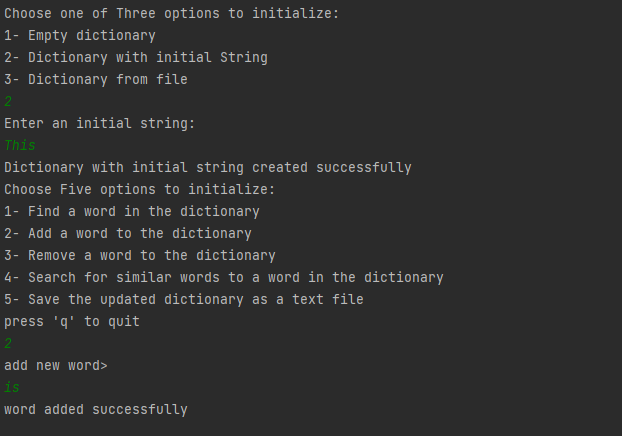
**We created a small menu in a main method as a gateway to use all methods discussed.**

Code:

**A screen shot of a computer

Description automatically generated**

Output:

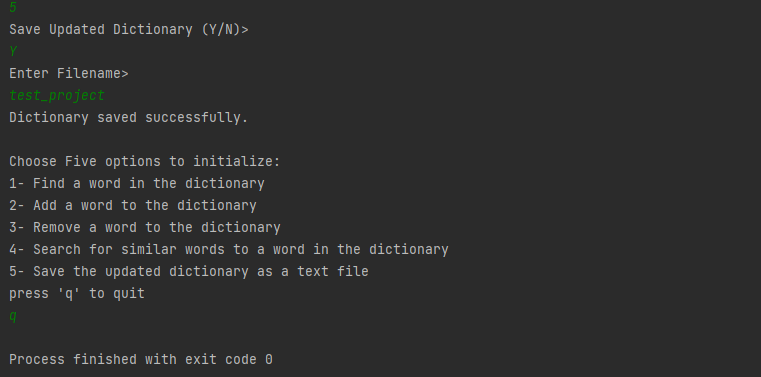


**A screenshot of a computer

Description automatically generated**

**A computer screen shot of white text

Description automatically generated**

****

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.**