King Fahd University of Petroleum & Minerals

College of Computer Sciences and Engineering

Information and Computer Science Department

ICS 202: Data Structures and Algorithms (3-3-4)

PROJECT

Lab Section: 57

Group Members:

ALI YAHYA ALSAIHATI – 202034820

Hussain Lutfi Alzayer – 202012120

Table of Content

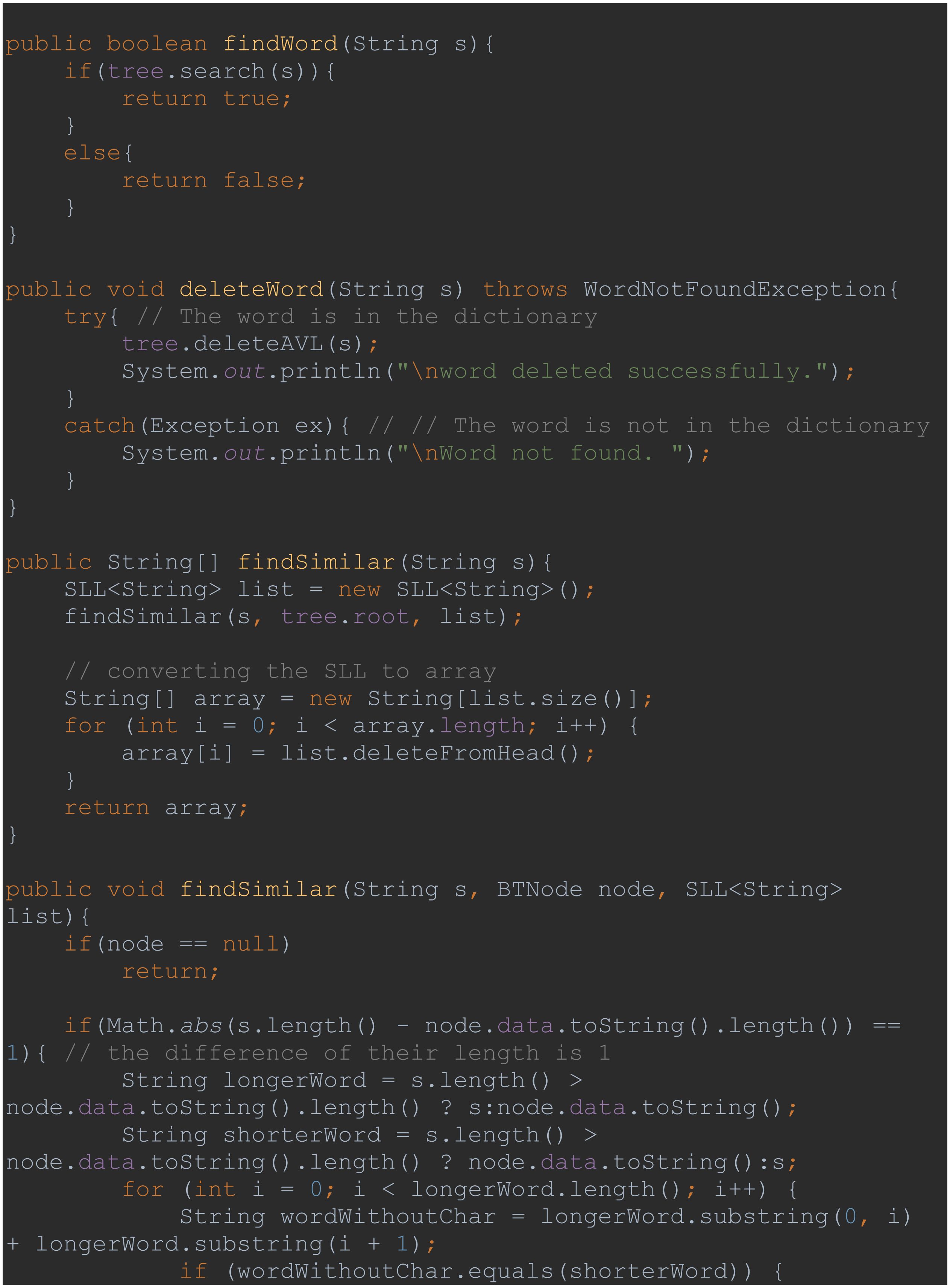
Why AVL?

This project is about a dictionary that allows users to do different operations such as adding, deleting, and searching. A suitable data structure that has been used is AVL tree because it balances itself after inserting, deleting, or searching for a node. Since BST, SLL, DLL, and CLL have high complexity, AVL Tree was the best choice. As a result, these operations' complexities are O(logn).

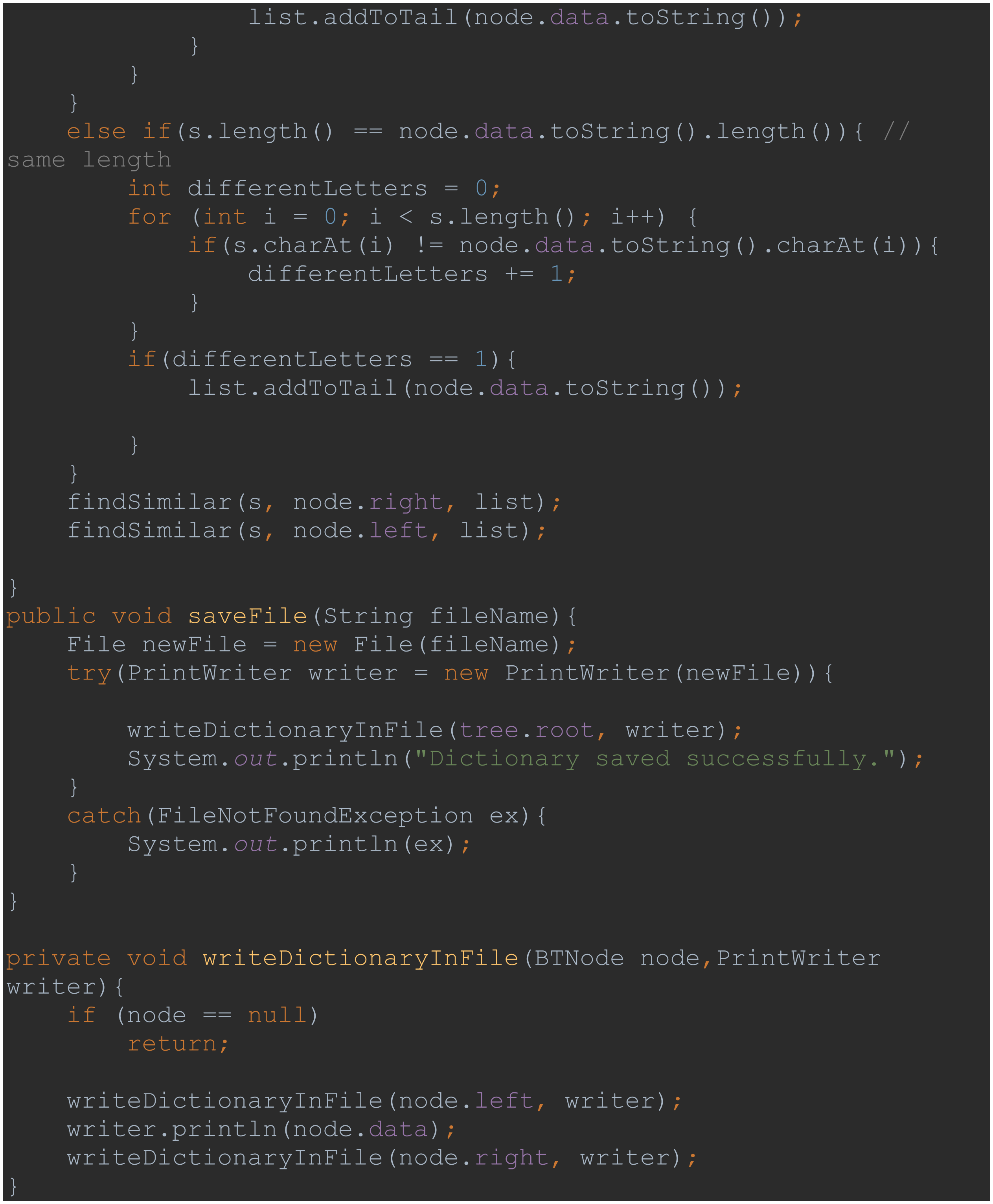
Code:

Class Dictionary:

```
private AVLTree<String> tree;

public Dictionary(){ // This constructor creates an empty AVL
tree.
    this.tree = new AVLTree<>();
}

public Dictionary(String string){ // This constructor creates an
AVL tree with a root containing the specified string.
    this.tree = new AVLTree<>();
    tree.insertAVL(string);
}

public Dictionary(File file) throws Exception{ // This
constructor creates an AVL tree with nodes containing the words
in a file.
    this.tree = new AVLTree<>();
    Scanner fileScanner = new Scanner(file);
    while(fileScanner.hasNext()){
        String word = fileScanner.next();
        try{
            tree.insertAVL(word);
        }
        catch(IllegalArgumentException ex){
        }
    }
    System.out.println("\ndictionary loaded successfully.");

}

public void addWord(String s) throws WordAlreadyExistsException{
    try{
        tree.insertAVL(s);
        System.out.println("\nword added successfully.");
    }
    catch(Exception ex){
        System.out.println("The word is already in the
dictionary.");
    }
}
```



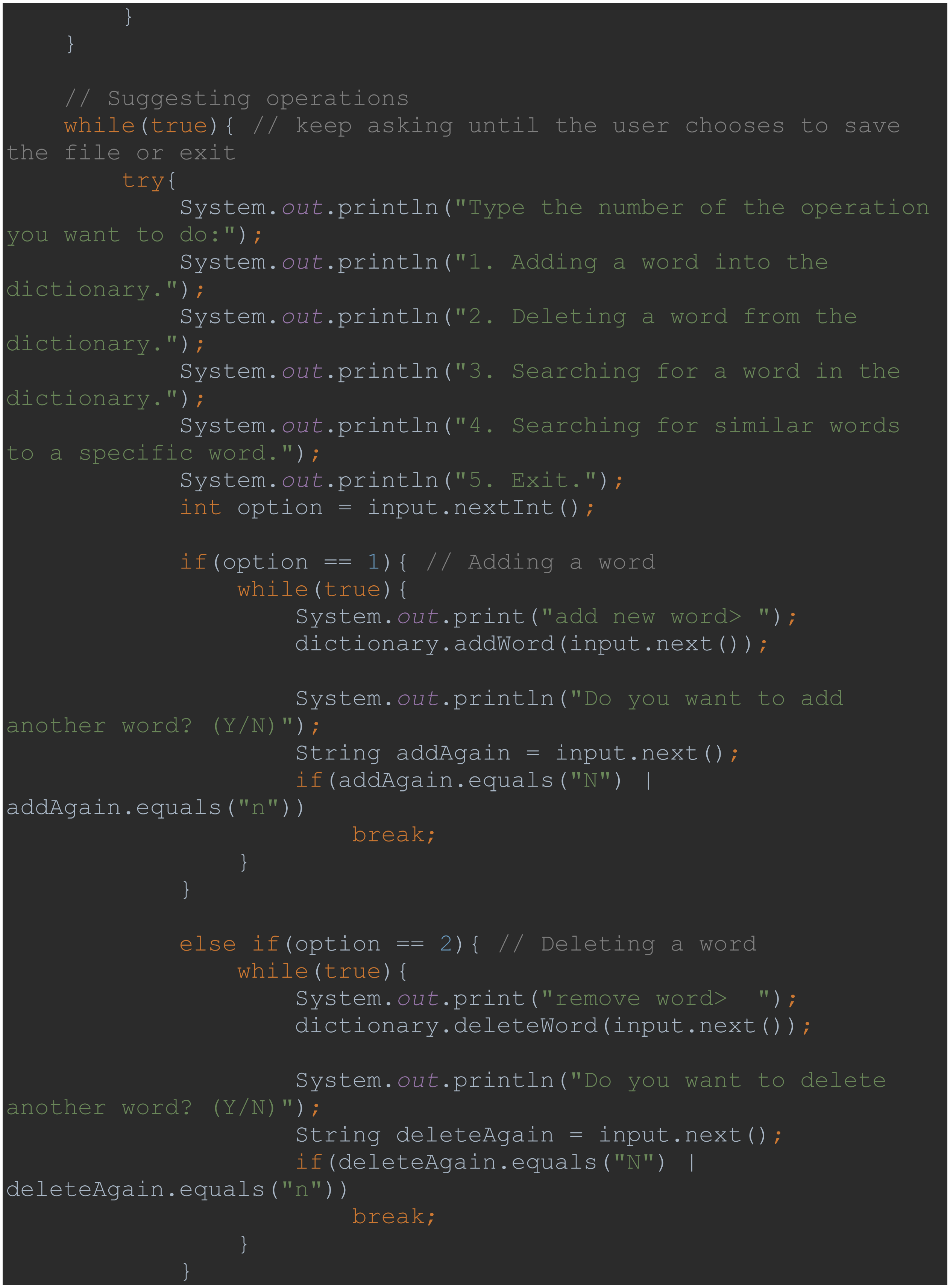
```
public boolean findWord(String s){
    if(tree.search(s)){
        return true;
    }
    else{
        return false;
    }
}

public void deleteWord(String s) throws WordNotFoundException{
    try{ // The word is in the dictionary
        tree.deleteAVL(s);
        System.out.println("\nword deleted successfully.");
    }
    catch(Exception ex){ // // The word is not in the dictionary
        System.out.println("\nWord not found. ");
    }
}

public String[] findSimilar(String s){
    SLL<String> list = new SLL<String>();
    findSimilar(s, tree.root, list);

    // converting the SLL to array
    String[] array = new String[list.size()];
    for (int i = 0; i < array.length; i++) {
        array[i] = list.deleteFromHead();
    }
    return array;
}

public void findSimilar(String s, BTNode node, SLL<String>
list){
    if(node == null)
        return;

    if(Math.abs(s.length() - node.data.toString().length()) ==
1){ // the difference of their length is 1
        String longerWord = s.length() >
node.data.toString().length() ? s:node.data.toString();
        String shorterWord = s.length() >
node.data.toString().length() ? node.data.toString():s;
        for (int i = 0; i < longerWord.length(); i++) {
            String wordWithoutChar = longerWord.substring(0, i)
+ longerWord.substring(i + 1);
            if (wordWithoutChar.equals(shorterWord)) {
```



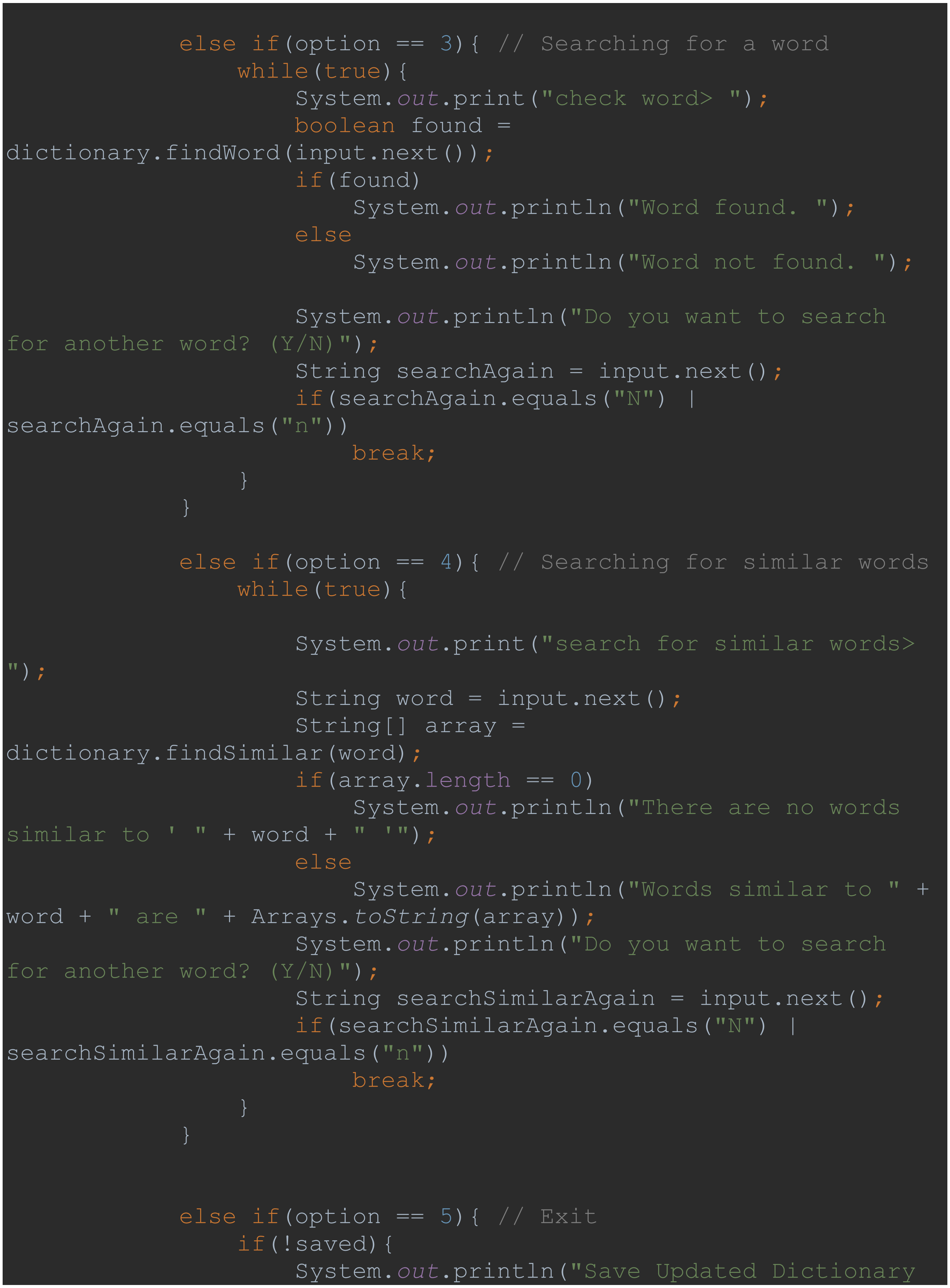
```
                list.addToTail(node.data.toString());
            }
        }
    }
    else if(s.length() == node.data.toString().length()){ //
same length
        int differentLetters = 0;
        for (int i = 0; i < s.length(); i++) {
            if(s.charAt(i) != node.data.toString().charAt(i)){
                differentLetters += 1;
            }
        }
        if(differentLetters == 1){
            list.addToTail(node.data.toString());

        }
    }
    findSimilar(s, node.right, list);
    findSimilar(s, node.left, list);

}
public void saveFile(String fileName){
    File newFile = new File(fileName);
    try(PrintWriter writer = new PrintWriter(newFile)){

        writeDictionaryInFile(tree.root, writer);
        System.out.println("Dictionary saved successfully.");
    }
    catch(FileNotFoundException ex){
        System.out.println(ex);
    }
}

private void writeDictionaryInFile(BTNode node,PrintWriter
writer){
    if (node == null)
        return;

    writeDictionaryInFile(node.left, writer);
    writer.println(node.data);
    writeDictionaryInFile(node.right, writer);
}
```

Testing Class:

```
public static void main(String[] args) throws Exception{
    // Asking for the filename
    Scanner input = new Scanner(System.in);

    // offering type of dictionary

    // Creating a dictionary
    Dictionary dictionary;
    boolean saved = false;
    while(true){
        try{

            System.out.println("Type the number of the
dictionary you want: ");
            System.out.println("1. A dictionary with no words.
");
            System.out.println("2. A dictionary with single
word. ");
            System.out.println("3. A dictionary from a file>
");
            int type = input.nextInt();

            if(type == 1){
                dictionary = new Dictionary();
                break;

            }
            else if(type == 2){
                System.out.println("Add a word: ");
                dictionary = new Dictionary(input.next());
                break;
            }
            else if(type == 3){
                System.out.print("Enter filename> ");
                String fileName = input.next();
                dictionary = new Dictionary(new File(fileName));
                break;
            }
            else{
                System.out.println("Choose a valid option,
please.");
            }
        }
        catch(IOException ex){
            System.out.println("This file does not exist.\n");
```



```
        }
    }

    // Suggesting operations
    while(true){ // keep asking until the user chooses to save
the file or exit
        try{
            System.out.println("Type the number of the operation
you want to do:");
            System.out.println("1. Adding a word into the
dictionary.");
            System.out.println("2. Deleting a word from the
dictionary.");
            System.out.println("3. Searching for a word in the
dictionary.");
            System.out.println("4. Searching for similar words
to a specific word.");
            System.out.println("5. Exit.");
            int option = input.nextInt();

            if(option == 1){ // Adding a word
                while(true){
                    System.out.print("add new word> ");
                    dictionary.addWord(input.next());

                    System.out.println("Do you want to add
another word? (Y/N)");
                    String addAgain = input.next();
                    if(addAgain.equals("N") |
addAgain.equals("n"))
                        break;
                }
            }

            else if(option == 2){ // Deleting a word
                while(true){
                    System.out.print("remove word>  ");
                    dictionary.deleteWord(input.next());

                    System.out.println("Do you want to delete
another word? (Y/N)");
                    String deleteAgain = input.next();
                    if(deleteAgain.equals("N") |
deleteAgain.equals("n"))
                        break;
                }
            }
```



```
            else if(option == 3){ // Searching for a word
                while(true){
                    System.out.print("check word> ");
                    boolean found =
dictionary.findWord(input.next());
                    if(found)
                        System.out.println("Word found. ");
                    else
                        System.out.println("Word not found. ");

                    System.out.println("Do you want to search
for another word? (Y/N)");
                    String searchAgain = input.next();
                    if(searchAgain.equals("N") |
searchAgain.equals("n"))
                        break;
                }
            }

            else if(option == 4){ // Searching for similar words
                while(true){

                    System.out.print("search for similar words>
");
                    String word = input.next();
                    String[] array =
dictionary.findSimilar(word);
                    if(array.length == 0)
                        System.out.println("There are no words
similar to ' " + word + " '");
                    else
                        System.out.println("Words similar to " +
word + " are " + Arrays.toString(array));
                    System.out.println("Do you want to search
for another word? (Y/N)");
                    String searchSimilarAgain = input.next();
                    if(searchSimilarAgain.equals("N") |
searchSimilarAgain.equals("n"))
                        break;
                }
            }


            else if(option == 5){ // Exit
                if(!saved){
                    System.out.println("Save Updated Dictionary
(Y/N)> ");
                    String choice = input.next();
                    if(choice.equals("Y") | choice.equals("y")){
                        System.out.print("Enter filename> ");
                        dictionary.saveFile(input.next());
                    }
                    else if(choice.equals("N") |
choice.equals("n"))
                        break;
                    else{
                        System.out.println("Wrong input. Please,
choose a valid option.");
                    }
                }
                break;
            }

            else{ // wrong option
                System.out.println("Please choose a valid
option.");
            }

        }
        catch(Exception ex){

        }
    }
}
```

Errors Classes:

```
public class WordAlreadyExistsException extends Exception{
    public WordAlreadyExistsException(String s){
        super(s);
    }
}

public class WordNotFoundException extends Exception{
    public WordNotFoundException(String s){
        super(s);
    }
}
```

Efficiency

1. { Dictionary() } => Creating an empty dictionary costs O(1)

2. { Dictionary(String s) } => Creating a dictionary containing single word is O(1)

3. { Dictionary(File f) } => Traversing on the words in the file and then use the insert avl algorithm costs O(nlogn)

4. { addWord(String s) } => O(log n) -> Insertion in AVL Tree is O(log n)

5. { deleteWord(String s) } => O(log n) -> Deletion in AVL Tree is O(log n)

6. { findWord(String s) } => O(log n) -> Searching in AVL Tree is O(log n) since AVL trees are always balanced

7. { findSimilar(String s) } => O(n) -> Since they go over all elements in the AVL Tree

8. { saveFile(String fileName) & writeDictionaryInFile(BTNode node,PrintWriter writer) } => O(n) ->
Since they go over all elements in the AVL Tree