Name:{Khairul Rizqi Bin Mohd Shariff}
 	
Tutorial Group ID: {W15.}	

Code

```
/**
 * This class serves as a textUI as well as storing the main function
 * @author Khairul Rizqi Bin Mohd Shariff
 */
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class TextBuddyPlusPlus {

	private final static String MESSAGE_WELCOME = "Welcome to TextBuddy.
	%1$s is ready for use";
	private static Scanner inputScanner = new Scanner(System.in);

	public static void main(String[] args) throws IOException {
		Logic logicComponent = new Logic(args[0]);
		printWelcomeMessage(args);
		executeUserInputs(logicComponent);
	}
	/**
	 * Forwards to Logic inputs from user
	 * @param logicComponent      The logicComponent object that is used to
	 *                            process commands
	 * @throws IOException        Happens if storage operations are unable
	 *                            to read/write to file
	 */
	private static void executeUserInputs(Logic logicComponent) throws
	IOException {
		while(true) {
		logicComponent.parseCommand(inputScanner.next(),
		inputScanner.nextLine());
		}
	}

	public static void printArrayListToScreen(ArrayList<String>
	outputToScreen) {
```

```
            for (int i = 0; i < outputToScreen.size();i++) {
                  printTextToScreen(outputToScreen.get(i));
            }
      }

      public static void printTextToScreen(String outputToScreen){
            System.out.println(outputToScreen);
      }
      private static void printWelcomeMessage(String[] args) {
            System.out.println(String.format(MESSAGE_WELCOME, args[0]));
      }

}

/**
 * This class serves as a parser and a logic component of TextBuddy++
 * @author Khairul Rizqi Bin Mohd Shariff
 */
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

public class Logic {
      private final static String COMMAND_ADD = "add";
      private final static String COMMAND_DISPLAY = "display";
      private final static String COMMAND_DELETE = "delete";
      private final static String COMMAND_CLEAR = "clear";
      private final static String COMMAND_EXIT = "exit";
      private final static String COMMAND_INVALID = "Invalid command! Please
      try again!";
      private static final String COMMAND_SORT = "sort";
      private static final String COMMAND_SEARCH = "search";
      private static final String BLANK_STRING = "";
      private static Storage storageComponent;

      public Logic(String filename) throws IOException {
            storageComponent = new Storage(filename);
      }

      /**
       * Acts as a parser to determine what command was given and then passes
       * the instruction and variables needed for the storage to carry out
       * the instruction
       *
       * @param command       Takes in the instruction for Storage to act
       *                      upon
       * @param variables     Takes in the variable needed for the Storage to
       *                      use
       * @throws IOException  Happens if the storage operations are unable to
       *                      read/write to a file
       */
```

```
public void parseCommand(String command,String variables) throws
IOException {
      if (command.equals(COMMAND_ADD)){
            addToTextFile(variables);
      } else if (command.equals(COMMAND_CLEAR)) {
            clearTextFile();
      } else if (command.equals(COMMAND_DISPLAY)) {
            displayTextFile();
      } else if (command.equals(COMMAND_EXIT)) {
            exitSystem();
      } else if (command.equals(COMMAND_DELETE)) {
            deleteTextEntryFromFile(variables);
      } else if (command.equals(COMMAND_SORT)) {
            sortTextFile();
      } else if (command.equals(COMMAND_SEARCH)) {
            searchTextFile(variables);
      } else {
            getTextMessage(COMMAND_INVALID);
      }
}

private void searchTextFile(String variables) {
      if (isVariableEmpty(variables)) {
            getTextMessage(COMMAND_INVALID);
      } else {
            storageComponent.search(variables);
      }
}

private void sortTextFile() throws IOException {
      storageComponent.sort();
}

private static void exitSystem() throws IOException {
      storageComponent.exit();
}

private void deleteTextEntryFromFile(String variables) throws
IOException {
      if (isVariableEmpty(variables)) {
            getTextMessage(COMMAND_INVALID);
      } else {
            storageComponent.delete(variables);
      }
}

private void displayTextFile() throws FileNotFoundException {
      storageComponent.display();
}

private void clearTextFile() throws IOException {
      storageComponent.clear();
}
```

```
	private void addToTextFile(String variables) throws IOException {
		if (isVariableEmpty(variables)) {
			getTextMessage(COMMAND_INVALID);
		} else {
			storageComponent.add(variables);
		}
	}

	private boolean isVariableEmpty(String variables) {
		return variables.equals(BLANK_STRING);
	}

	public static void getTextMessage(String message) {
		TextBuddyPlusPlus.printTextToScreen(message);
	}

	public static void getTextMessages(ArrayList<String> message) {
		TextBuddyPlusPlus.printArrayListToScreen(message);
	}

}

/**
 * This class handles all the memory storage and file writing operations
 * @author Khairul Rizqi Bin Mohd Shariff
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;

public class Storage {
	private static final String INVALID_INDEX = "Invalid index";
	private final static String DONE_EMPTY_COMMAND = " is empty";
	private final static String DONE_CLEAR_COMMAND = "all content deleted
	from ";
	private final static String DONE_DELETE_COMMAND = "deleted from %1$s:
	\"%2$s\"";
	private final static String DONE_ADD_COMMAND = "added to %1$s:
	\"%2$s\"";
	private final static String DONE_SORT_COMMAND = " is sorted";
	private static final String KEYWORD_NOT_FOUND = "keyword: \"%1$s\" is
	not found";
	private static String filename;
	private static PrintWriter fileWriter;
	private static BufferedWriter fileWriterBuffer;
	private static FileReader fileReader;
	private static BufferedReader textReader;
	private static ArrayList<String> textBuffer = new ArrayList<String>();
	private static int lineCounter = 1;
```

```java
public Storage(String file) throws IOException {
      filename = file;
      initaliseFile(file);
      readTextFile(file);
}

/**
 * Reads in the text file line by line and adds them into the
 * textBuffer for processing
 *
 * @param file           Name of text file
 * @throws IOException   Happen if function unable to read from file
 */
private void readTextFile(String file) throws IOException {
      initaliseFileReader();
      String line;

      while ((line = textReader.readLine()) != null) {
            textBuffer.add(line);
            lineCounter++;
      }
}

/**
 * Initializes the PrintWriter function. Will check for the file. If
 * have, will use the text file, otherwise create a new file
 *
 * @param filename       Name of text file
 * @throws IOException   Happens if function unable to create file
 */
private static void initaliseFile(String filename) throws IOException {
      fileWriter = new PrintWriter(new FileWriter(filename,true));
      fileWriterBuffer = new BufferedWriter(fileWriter);
}

/**
 * Add message input into textBuffer and directly to the text file
 *
 * @param textInput      Text message to be added to file
 * @throws IOException   Happens if function unable to write to file
 */
public void add(String textInput) throws IOException {
      String messageToBePrinted = lineCounter + "." + textInput;
      textBuffer.add(messageToBePrinted);

      /**to add the numbering in front of the text */
      fileWriter.println(messageToBePrinted);
      fileWriter.flush();
      lineCounter++;

      /**to remove the empty space before the textInput */
      Logic.getTextMessage(String.format(DONE_ADD_COMMAND,
      filename,textInput.substring(1)));
}
```

```
/**
 * Delete all texts in the text file.
 *
 * @throws IOException  Happens if unable to read/write to file
 */
public void clear() throws IOException {
      initaliseFile(filename);
      lineCounter = 1;
      textBuffer.clear();
      Logic.getTextMessage(DONE_CLEAR_COMMAND + filename);
}

/**
 * Displays all the texts from the text file to screen
 *
 * @throws FileNotFoundException if unable to find the file
 */
public void display() throws FileNotFoundException {
      if (textBuffer.size() == 0) {
            Logic.getTextMessage(filename + DONE_EMPTY_COMMAND);
      } else {
            Logic.getTextMessages(textBuffer);
      }
}

private void initaliseFileReader() throws IOException {
      fileReader = new FileReader(filename);
      textReader = new BufferedReader(fileReader);
}

/**
 * Deletes the line requested to be deleted from text file
 *
 * @param variables     Takes in the line number to be cleared from the
 *                      text file
 * @throws IOException  Happens if unable to write to file
 */
public void delete(String variables) throws IOException {
      int lineNumberToBeRemoved =
      Integer.parseInt(variables.substring(1))-1;
      if (lineNumberToBeRemoved < 0) {
            Logic.getTextMessage(INVALID_INDEX);
      } else {
            lineCounter = 1;
            String messageToBeDeleted =
            textBuffer.get(lineNumberToBeRemoved).substring(3);
            textBuffer.remove(lineNumberToBeRemoved);
            @SuppressWarnings("unchecked")
            ArrayList<String> temp = (ArrayList<String>)
            textBuffer.clone();
            textBuffer = new ArrayList<String>();

            for (int i = 0; i < temp.size(); i++) {
                  textBuffer.add(lineCounter+temp.get(i).substring(1));
```

```
				lineCounter++;
			}
			Logic.getTextMessage(String.format(DONE_DELETE_COMMAND,
			filename, messageToBeDeleted));
		}
		initaliseFile(filename);
		writeTextBufferToFile();
	}

	/**
	* Used to check for contents in textBuffer. Only for debugging
	* purposes
	*/
	@SuppressWarnings("unused")
	private void checkTextBuffer() {
		for (int i = 0; i < textBuffer.size(); i++) {
			System.out.println(textBuffer.get(i));
		}
	}

	/**
	 * Writes to text file from the textBuffer
	 *
	 * @throws FileNotFoundException Happens if unable to write to file.
	 */
	private static void writeTextBufferToFile() throws
	FileNotFoundException {
		fileWriter = new PrintWriter(filename);

		for (int i = 0; i < textBuffer.size(); i++) {
			fileWriter.println(textBuffer.get(i));
			fileWriter.flush();
		}
	}

	/**
	 * Closes all the streams leading to the text file
	 *
	 * @throws IOException  Happens if unable to close any streams to the
text file
	 */
	public static void closeFile() throws IOException {
		fileWriter.close();
		fileWriterBuffer.close();
		fileReader.close();
		textReader.close();
	}
```

```
/**
 * Writes any changes done in the textBuffer into the text file and
 * closes all streams
 *
 * @throws IOException  Happens if unable to write to file
 */
public void exit() throws IOException {
      writeTextBufferToFile();
      closeFile();
      System.exit(0);
}

/**
 * Sorts all the content in the textBuffer by alphabetical order.
 * It is also case insensitive which further ensures alphabetical order
 * Will write to file after sorting all the text inputs both in file
 * and textBuffer
 *
 * @throws IOException  Happens if unable to write to file
 */
public void sort() throws IOException {
      ArrayList<String> temp = new ArrayList<String>();

      for (int i = 0; i < textBuffer.size(); i++) {
            temp.add(textBuffer.get(i).substring(3));
      }

      textBuffer = new ArrayList<String>();
      Collections.sort(temp, String.CASE_INSENSITIVE_ORDER);
      lineCounter = 1;

      for (int i = 0; i < temp.size(); i++) {
            textBuffer.add(lineCounter+". "+temp.get(i));
            lineCounter++;
      }

      initaliseFile(filename);
      writeTextBufferToFile();
      Logic.getTextMessage(filename+DONE_SORT_COMMAND);
}
```

```
	/**
	 * Searches all entries in the textBuffer and returns all the entries
	 * that contains the keyword
	 */
		public void search(String keyword) {
		int searchResultsListing = 1;
		boolean isFound = false;
		ArrayList<String> searchResults = new ArrayList<String>();

		for (int i = 0; i < textBuffer.size(); i++) {
		if (textBuffer.get(i).toLowerCase().contains
			(keyword.toLowerCase())) {
			 searchResults.add(searchResultsListing+".
			 "+textBuffer.get(i).substring(3));
				searchResultsListing++;
				isFound = true;
			}
		}
		if (isFound) {
			Logic.getTextMessages(searchResults);
		} else {
			/**to remove the empty space in keyword */
			Logic.getTextMessage(String.format(KEYWORD_NOT_FOUND,
			keyword.substring(1)));
		}
	}
}
```

TestInput.txt

```
add Sherry
add is
add cute
add and
add tsun
display
delete 4
delete 4
display
clear
display
add Sherry
add uses
add moe moe attack
display
sort
```

```
display
clear
add Sherry
add have
add moe moe attack
add bubblebeam attack
add flamethrower attack
display
search attack
search attacks
exit
```

ExpectedOutput.txt

```
Welcome to TextBuddy. finalSherryUpgradedTest.txt is ready for use
added to finalSherryUpgradedTest.txt: "Sherry "
added to finalSherryUpgradedTest.txt: "is"
added to finalSherryUpgradedTest.txt: "cute"
added to finalSherryUpgradedTest.txt: "and"
added to finalSherryUpgradedTest.txt: "tsun"
1. Sherry
2. is
3. cute
4. and
5. tsun
deleted from finalSherryUpgradedTest.txt: "and"
deleted from finalSherryUpgradedTest.txt: "tsun"
1. Sherry
2. is
3. cute
all content deleted from finalSherryUpgradedTest.txt
finalSherryUpgradedTest.txt is empty
added to finalSherryUpgradedTest.txt: "Sherry"
added to finalSherryUpgradedTest.txt: "uses"
added to finalSherryUpgradedTest.txt: "moe moe attack"
1. Sherry
2. uses
3. moe moe attack
finalSherryUpgradedTest.txt is sorted
1. moe moe attack
2. Sherry
3. uses
```

```
all content deleted from finalSherryUpgradedTest.txt
added to finalSherryUpgradedTest.txt: "Sherry"
added to finalSherryUpgradedTest.txt: "have"
added to finalSherryUpgradedTest.txt: "moe moe attack"
added to finalSherryUpgradedTest.txt: "bubblebeam attack"
added to finalSherryUpgradedTest.txt: "flamethrower attack"
1. Sherry
2. have
3. moe moe attack
4. bubblebeam attack
5. flamethrower attack
1. moe moe attack
2. bubblebeam attack
3. flamethrower attack
keyword: "attacks" is not found
```