Name:Morgan James Howell

Tutorial Group ID: T09

Code

```
/**
 * TextBuddy - Morgan J. Howell
 * A1 Submission for CS2103
 *
 *
 * This class is used to write, store, and retrieve one line notes that are organized into
 * a numerical list. Please find details of commands, assumptions, and making the executable
 * with a test suite below.
 *
 *
 * Commands of TextBuddy
 *--------------------------
 *  1. add ANY_ STRING: Add items to your list.
 *  2. delete N : Deletes the Nth line item of your list.
 *  3. display : Shows your current list to be saved into persistent memory.
 *  4. clear : Deletes all items in the list.
 *  5. help: Displays a list of all available commands.
 *  6. exit: Exits the program.
 *
 * Assumptions
 *------------
 *          1. Each operation will trigger a save into persistent memory, because the program is
 * extremely light weight I experimented with different forms and found no increased latency.
 *          2. Not all file extensions can be written to, I keep a static string of "approved extensions"
 * (i.e. txt, md, and rtf) that are matched via regex on program startup.
 *          3. All given commands can be normalized in a {command, payload} format.
 *  4. Exceptions, such as deleting an item that doesn't exist and issuing unsupported commands,
 * will not crash the program, rather only trigger print statements explaining the error.
 *          5. The user will be writing a file in the current directory that the program sits, the program
 * does not support absolute path parameters, only params of type "\w+\.(rtf|md|txt)".
 *  6. I assumed that testing could just be carried out via the make file (i.e. "make test").
 *                                      -"make test" pipes in expected input and prints diff with actual output and expected output,
 *  7. I made a series of much smaller arbitrary assumptions such as commands having to follow a linear format
 *  and always being lowercase. Also assumed that commands such as "add " would just write an empty line.
 *  8. Last major assumption was that we can neglect whitespace, for instance "delete     9" just means "delete 9"
 *  and "add hey   there    example" was intended as the format "add hey there example".
 *
 * ****Building and Testing TextBuddy****
 *------------------
 * - I created a makefile to streamline this process. Simply issue the following commands:
 *   1. make clean (cleans all executables and testers)
 *   2. make  (compiles the java program into a class file using javac)
 *   3. make test (pipes the input file in and diffs it against an expected output file)
 *
 * @author Morgan Howell
 */

import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.io.FileReader;
import java.io.BufferedReader;

public class TextBuddy{
	//Standard console response strings
	private static final String MESSAGE_WELCOME_SCREEN = "\n<Welcome to TextBuddy by Morgan Howell!>\n"
					+ "          ---------------------          \n"
					+ "Changes will be saved to \"%1$s\"\n"
					+ "Type \"help\" for a list of available commands\n";
	private static final String MESSAGE_HELP_GUIDE = "\n   ---------------HELP GUIDE---------------\n"
					+ "  |                                      |\n"
					+ "  | add ANY_TEXT_CAN_FOLLOW: adds text    |\n"
					+ "  | delete LINE_NUMBER: removes that entry |\n"
					+ "  | clear: removes all entries           |\n"
					+ "  | display: shows all entries           |\n"
					+ "  | exit: terminates the program         |\n"
					+ "   ---------------------------------------\n";
	private static final String MESSAGE_DISPLAY_TEMPLATE = "\n------------LIST FOR \"%1$s\"------------\n"
					+ "%2$s"
					+ "-----------------------------------------------------\n";
	private static final String MESSAGE_IO_EXCEPTION = "\nWe encountered an IOException while attempting to open the given file.\n";
	private static final String MESSAGE_UNSUPPORTED_COMMAND = "\nYou've attempted an unsupported command. Issue command 'help' for
details.\n";
	private static final String MESSAGE_ABSENT_COMMAND = "\nPlease provide a command and press enter. Issue command 'help' for a list of valid
commands.\n";
	private static final String MESSAGE_WRONG_FILEOUT = "\nMalformed Request: Supplied parameter does not follow expected format of
(*.%1$s).\n";
	private static final String MESSAGE_WRONG_NUM_PARAMS = "\nMalformed Request: Please provide the correct number of parameters.\n";
	private static final String MESSAGE_SENTENCE_ADDED = "\nAdded to %1$s: \"%2$s\"\n";
	private static final String MESSAGE_BLANK_LINE_ATTEMPT = "\nA blank line was added to %1$s. Please consider adding something more
meaningful next time.\n";
	private static final String MESSAGE_DELETE_TYPE_ERROR = "\nPlease provide a valid line number to be deleted.\n";
	private static final String MESSAGE_SENTENCE_DELETED = "\nDeleted from %1$s: \"%2$s\"\n";
	private static final String MESSAGE_MEMORY_CLEARED = "\nAll content cleared from %1$s.\n";
	private static final String MESSAGE_FILE_EMPTY = "\n%1$s is empty.\n";
	private static final String MESSAGE_EXIT = "\nThank you for using TextBuddy by Morgan Howell!\n"
                                                                                                                + "Your additions were
saved to %1$s\n";
	private static final String MESSAGE_COMMAND_PROMPT = "%1$s>$ ";
	//Add supported extensions for output file below (regex tested).
	private static final String SUPPORTED_EXTENSIONS = "txt|md|rtf";
	private static final String REGEX_EXTENSION_TEST = "\\w+\\.(" + SUPPORTED_EXTENSIONS + ")";

	public enum CommandIssue {
		HELP, ADD_ITEM, DELETE_ITEM, CLEAR, DISPLAY, EMPTY, UNSUPPORTED, EXIT
		}

	private List<String> items;
	private File fileOut;
	private String fileName;
	private int itemCount;
	private Scanner systemIn;

	public static void main(String[] args){
	String sanitizedFileName = sanitizeArgs(args);
	TextBuddy helper = TextBuddy.generateBuddyHelper(sanitizedFileName);
		if(helper != null) {
			helper.welcomeUser();
			CommandIssue command;

			do {
				//User input is normalized to an array always containing two elements (command + payload)
				String[] parsedUserInput = helper.promptUser();
				command = mapUserInputToCommand(parsedUserInput[0]);
				helper.executeCommand(command, parsedUserInput[1]);
			} while(command != CommandIssue.EXIT);

		}
	}

	//Factory design pattern style object generator for instantiation and object setup
	public static TextBuddy generateBuddyHelper(String fileName) {
		try {
			TextBuddy helper = new TextBuddy(fileName);
			helper.restoreInMemory();
			return helper;
		} catch (UnsupportedOperationException err) {
			System.out.println(err.getMessage());
		} catch (IOException err) {
			System.out.println(MESSAGE_IO_EXCEPTION);
		}
		return null;
	}

	//Throws error upon malformed parameter types
	public static String sanitizeArgs(String[] args) throws UnsupportedOperationException {
		if (args.length == 1) {
			//Validate given parameter as expected text file format
			String unsanitizedFile = args[0];
```

```
                        Pattern pattern = Pattern.compile(REGEX_EXTENSION_TEST);
                        Matcher match = pattern.matcher(unsanitizedFile);
                        boolean isValidDataSource = match.matches();
                        if(isValidDataSource) {
                                return unsanitizedFile;
                        } else {
                                throw new UnsupportedOperationException(String.format(MESSAGE_WRONG_FILEOUT, SUPPORTED_EXTENSIONS));
                        }
                } else {
                        throw new UnsupportedOperationException(MESSAGE_WRONG_NUM_PARAMS);
                }
        }

        public static CommandIssue mapUserInputToCommand(String userInput) {
                CommandIssue command;
                switch (userInput) {
                        case "add":
                                command = CommandIssue.ADD_ITEM;
                                break;

                        case "display":
                                command = CommandIssue.DISPLAY;
                                break;

                        case "delete":
                                command = CommandIssue.DELETE_ITEM;
                                break;

                        case "clear":
                                command = CommandIssue.CLEAR;
                                break;

                        case "help":
                                command = CommandIssue.HELP;
                                break;

                        case "exit":
                                command = CommandIssue.EXIT;
                                break;

                        case "":
                                command = CommandIssue.EMPTY;
                                break;

                        default:
                                command = CommandIssue.UNSUPPORTED;
                                break;
                }

                return command;
        }

        public TextBuddy(String fileName) {
                this.fileName = fileName;
                fileOut = new File(fileName);
                items = new ArrayList<String>();
                systemIn = new Scanner(System.in);
        }

        public void welcomeUser() {
                System.out.println(String.format(MESSAGE_WELCOME_SCREEN, fileName));
        }

        public String[] promptUser() {
                System.out.print(String.format(MESSAGE_COMMAND_PROMPT, fileName));
                String nextLine = systemIn.nextLine();
                String[] userInput = nextLine.split("\\s+");
                return sanitizeUserCommand(userInput);
        }

        public void restoreInMemory() throws IOException {
                if(fileOut.exists()) {
                        try(BufferedReader br = new BufferedReader(new FileReader(fileOut))) {
                                for(String line; (line = br.readLine()) != null;) {
                                        items.add(line);
                                        itemCount++;
                                }
                        }
                } else {
                        fileOut.createNewFile();
                }
        }

        public void executeCommand(CommandIssue command, String payload) {
                switch(command) {
                        case HELP:
                                showHelpGuider();
                                break;

                        case ADD_ITEM:
                                addText(payload);
                                break;
```

```
                case DELETE_ITEM:
                        deleteLine(payload);
                        break;

                case DISPLAY:
                        getText();
                        break;

                case CLEAR:
                        clearText();
                        break;

                case EMPTY:
                        notifyUnsupported(false);
                        break;

                case UNSUPPORTED:
                        notifyUnsupported(true);
                        break;

                case EXIT:
                        showExit();
                        break;
        }
        //save user changes persistently after every command in case of interrupt
        saveAndPersist();
}

//Normalizes all given user commands to an array of size two (e.g. [command, payload])
private String[] sanitizeUserCommand(String[] userInput) {
        String[] normalizedUserCommand = new String[2];
        if(userInput.length>1) {
                String[] payloadPortions = Arrays.copyOfRange(userInput, 1, userInput.length);
                String payloadParsed = String.join(" ", payloadPortions);
                normalizedUserCommand[0] = userInput[0];
                normalizedUserCommand[1] = payloadParsed;
        } else if(userInput.length == 1) {
                normalizedUserCommand[0] = userInput[0];
                normalizedUserCommand[1] = "";
        } else {
                normalizedUserCommand[0] = "";
                normalizedUserCommand[1] = "";
        }
        return normalizedUserCommand;
}

private void notifyUnsupported(Boolean unsupportedCommand) {
        if(unsupportedCommand) {
                System.out.println(MESSAGE_UNSUPPORTED_COMMAND);
        } else {
                System.out.println(MESSAGE_ABSENT_COMMAND);
        }
}

private void showHelpGuider() {
        System.out.println(MESSAGE_HELP_GUIDE);
}

private void addText(String sentence) {
        if(sentence == "") {
                items.add("");
                System.out.println(String.format(MESSAGE_BLANK_LINE_ATTEMPT, fileName));
        } else {
                items.add(sentence);
                System.out.println(String.format(MESSAGE_SENTENCE_ADDED, fileName, sentence));
        }
        itemCount++;
}

private void deleteLine(String lineNumber) {
        try {
                int targetIndex = Integer.parseInt(lineNumber);
                if(targetIndex > 0 && itemCount >= targetIndex) {
                        String targetLine = items.remove(targetIndex-1);
                        System.out.println(String.format(MESSAGE_SENTENCE_DELETED, fileName, targetLine));
                        itemCount--;
                } else {
                        throw new NumberFormatException();
                }
        } catch(NumberFormatException err) {
                System.out.println(MESSAGE_DELETE_TYPE_ERROR);
        }
}

private void clearText() {
        if(itemCount>0) {
                items.clear();
                itemCount = 0;
                System.out.println(String.format(MESSAGE_MEMORY_CLEARED, fileName));
        } else {
                System.out.println(String.format(MESSAGE_FILE_EMPTY, fileName));
        }
}
```

```
	private void getText() {
		if(!items.isEmpty()) {
			String totalList = "";
			for(int i = 0; i < items.size(); i++)
			{
				totalList += "\n" + (i+1) + ". " + items.get(i) + "\n";
			}
			System.out.println(String.format(MESSAGE_DISPLAY_TEMPLATE, fileName, totalList));
		} else {
			System.out.println(String.format(MESSAGE_FILE_EMPTY, fileName));
		}
	}

	private void showExit() {
		systemIn.close();
		System.out.println(String.format(MESSAGE_EXIT, fileName));
	}

	private void saveAndPersist() {
		if(itemCount>0) {
			try(PrintWriter writer = new PrintWriter(fileName, "UTF-8")) {
				for(String item : items) {
					writer.println(item);
				}
			} catch(IOException err) {
				System.out.println(MESSAGE_IO_EXCEPTION);
			}
		}
	}


}
```

TestInput.txt

help

add 12345 56789

add testing here, I am testing

add take me to the grocery store

add This is a cool test with a cool make file!

display

delete 1

delete 2

display

delete 1

add testing is soo cool, I love it.

add Time to go for some corner cases soon.

display

clear

clear

display

add first one to be added

delete 1

delete 1

delete 1

delete 0

delete -1

add

delete

delete 5

add now let's think of some clever edge cases

-1234123

null

0

delete HEYYYY

delete null

delete "test"

add null

add TEST0

clear 650

help 500

add wow, hope we survived that. We can run make test to run our diffs to test that.

add We're going to exit and reload this to make sure persistent memory works!

exit

TestInput2.txt (testing persistent memory)

display

add since we reloaded this file using the same file previously, we should have access to our old items

clear display

delete 1

display

exit

ExpectedOutput.txt

<Welcome to TextBuddy by Morgan Howell!>

Changes will be saved to "memory.txt"

Type "help" for a list of available commands

memory.txt>$

----------HELP GUIDE----------

| |

| add ANY_TEXT_CAN_FOLLOW: adds text |

| delete LINE_NUMBER: removes that entry |

| clear: removes all entries |

| display: shows all entries |

| exit: terminates the program |

---

memory.txt>$

Added to memory.txt: "12345 56789"

memory.txt>$

Added to memory.txt: "testing here, I am testing"

memory.txt>$

Added to memory.txt: "take me to the grocery store"

memory.txt>$

Added to memory.txt: "This is a cool test with a cool make file!"

memory.txt>$

----------LIST FOR "memory.txt"----------

1. 12345 56789

2. testing here, I am testing

3. take me to the grocery store

4. This is a cool test with a cool make file!

---

memory.txt>$

Deleted from memory.txt: "12345 56789"

memory.txt>$

Deleted from memory.txt: "take me to the grocery store"

memory.txt>$

----------LIST FOR "memory.txt"----------

1. testing here, I am testing

2. This is a cool test with a cool make file!

---

memory.txt>$

Deleted from memory.txt: "testing here, I am testing"

memory.txt>$

Added to memory.txt: "testing is soo cool, I love it."

memory.txt>$

Added to memory.txt: "Time to go for some corner cases soon."

memory.txt>$

----------LIST FOR "memory.txt"----------

1. This is a cool test with a cool make file!

2. testing is soo cool, I love it.

3. Time to go for some corner cases soon.

---

memory.txt>$

All content cleared from memory.txt.

memory.txt>$

memory.txt is empty.

memory.txt>$

memory.txt is empty.

memory.txt>$

Added to memory.txt: "first one to be added"

memory.txt>$

Deleted from memory.txt: "first one to be added"

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

A blank line was added to memory.txt. Please consider adding something more meaningful next time.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Added to memory.txt: "now let's think of some clever edge cases"

memory.txt>$

You've attempted an unsupported command. Issue command 'help' for details.

memory.txt>$

You've attempted an unsupported command. Issue command 'help' for details.

memory.txt>$

You've attempted an unsupported command. Issue command 'help' for details.

memory.txt>$

You've attempted an unsupported command. Issue command 'help' for details.

memory.txt>$

You've attempted an unsupported command. Issue command 'help' for details.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

Added to memory.txt: "null"

memory.txt>$

Added to memory.txt: "TEST0"

```
memory.txt>$

All content cleared from memory.txt.


memory.txt>$

  ----------HELP GUIDE----------

 |                              |

 | add ANY_TEXT_CAN_FOLLOW: adds text    |

 | delete LINE_NUMBER: removes that entry |

 | clear: removes all entries           |

 | display: shows all entries           |

 | exit: terminates the program         |

  ----------------------------------------


memory.txt>$

You've attempted an unsupported command. Issue command 'help' for details.


memory.txt>$

Added to memory.txt: "wow, hope we survived that. We can run make test to run our diffs to test that."


memory.txt>$

Added to memory.txt: "We're going to exit and reload this to make sure persistent memory works!"


memory.txt>$

Thank you for using TextBuddy by Morgan Howell!

Your additions were saved to memory.txt
```

ExpectedOutput2.txt

<Welcome to TextBuddy by Morgan Howell!>

Changes will be saved to "memory.txt"

Type "help" for a list of available commands

memory.txt>$

----------LIST FOR "memory.txt"----------

1. wow, hope we survived that. We can run make test to run our diffs to test that.

2. We're going to exit and reload this to make sure persistent memory works!

---

memory.txt>$

Added to memory.txt: "since we reloaded this file using the same file previously, we should have access to our old items"

memory.txt>$

All content cleared from memory.txt.

memory.txt>$

Please provide a valid line number to be deleted.

memory.txt>$

memory.txt is empty.

memory.txt>$

Thank you for using TextBuddy by Morgan Howell!

Your additions were saved to memory.txt

makefile

```
JFLAGS = -g

JC = javac

.SUFFIXES: .java .class

.java.class:

	$(JC) $(JFLAGS) $*.java

CLASSES = TextBuddy.java

default: classes

classes: $(CLASSES:.java=.class)


clean:

	$(RM) *.class

	$(RM) actual*

	$(RM) memory.txt


test:

	java TextBuddy memory.txt < testinput.txt > actualoutput1.txt

	java TextBuddy memory.txt < testinput2.txt > actualoutput2.txt

	DIFF1="$(diff actualoutput1.txt expected.txt)"

	DIFF2="$(diff actualoutput2.txt expected2.txt)"

	$(echo "difference of actual1 and expected1:\n")

	$(echo DIFF1)

	$(echo "difference of actual2 and expected2:\n")

	$(echo DIFF2)
```