Name: Ang Kah Min, Kelvin

Tutorial Group ID: T17

Code

TextBuddy.java

```
package ce1;

/**
 * This program instantiates and run TextBuddy Text UI.
 *
 * The program requires a valid NTFS file name to be
 * supplied as the argument.
 *
 * @author Ang Kah Min, Kelvin
 */
public class TextBuddy {

	public static void main(String[] args) {
		TextBuddyTUI tb = new TextBuddyTUI(args);
		tb.systemRun();
	}
}
```

TextBuddyTUI.java

```
package ce1;

import java.util.List;

/**
 * This program is implements the Text UI for the TextBuddy system.
 *
 * TextBuddyTUI handles user interaction and comprehensive input checking.
 *
 * The program makes use of TextBuddyLogic to handle low level tasks.
 *
 * It has been updated to support alternate shorthand commands using String dictionaries.
 * (Just an array of String for the time being, not the Dictionary data structure).
 *
 * Methods are sorted according to abstraction level from the highest to the lowest level.
 *
 * @author Ang Kah Min, Kelvin
 */
public class TextBuddyTUI {

	// Command Type Enumeration

	private static enum CommandType {
		ADD, DELETE, DISPLAY, CLEAR, EXIT, INVALID
	};

	// Dictionaries

	private static final String[] DICTIONARY_ADD = { "add", "-a", "/a" };
	private static final String[] DICTIONARY_CLEAR = { "clear", "-c", "/c" };
	private static final String[] DICTIONARY_DELETE = { "delete", "-de", "/de" };
	private static final String[] DICTIONARY_DISPLAY = { "display", "-di", "/di" };
```

```
private static final String[] DICTIONARY_EXIT = { "exit", "-e", "/e" };
private static final String[] DICTIONARY_NTFS_INVALID = { "<", ">", ":",
                                                          "\"", "/", "\\",
                                                          "|", "?", "*" };

// Error Strings

private static final String ERROR_ARGUMENT_COUNT = "Wrong argument count supplied.";
private static final String ERROR_EXTENSION = "Invalid file extension.";
private static final String ERROR_FILE_NAME_LENGTH = "File name should be at least 1 character long.";
private static final String ERROR_FILE_RW = "There has been a filesystem error.";
private static final String ERROR_INVALID_CHARACTERS = "File name contains invalid characters.";

private static final String ERROR_ADD = "Add command has failed.";
private static final String ERROR_CLEAR = "Clear command has failed.";
private static final String ERROR_DELETE = "Delete command failed.";
private static final String ERROR_FATAL = "A fatal error has been encountered.";
private static final String ERROR_INVALID_COMMAND = "Command is invalid.";

// Error Codes

private static final int INVALID_INTEGER = -1;

// Printed Strings

private static final String FORMAT_ADD = "added to %1$s: \"%2$s\"";
private static final String FORMAT_CLEAR = "all content deleted from %1$s";
private static final String FORMAT_DELETE = "deleted from %1$s: \"%2$s\"";
private static final String FORMAT_DISPLAY = "%1$d. %2$s\n";
private static final String FORMAT_EMPTY = "%1$s is empty";
private static final String FORMAT_PROMPT = "command: ";
private static final String FORMAT_WELCOME = "Welcome to TextBuddy. %1$s is ready for use\n";

// Other Constants

private static final String VALID_EXTENSION = ".txt";

// Objects Declaration

TextBuddyLogic logic;
Scanner scanner;

// High Level System Implementation

// Constructor expects a single argument as file name.
public TextBuddyTUI(String[] args) {
        systemAssertArguments(args);
        systemInitialize(args);
        systemAssertReady();
}

private void systemAssertArguments(String[] args) {
        exitIfInvalidArgumentCount(args);
        exitIfInvalidFileName(args);
}

private void systemInitialize(String[] args) {
        initializeLogic(args);
        initializeTUI();
}

private void systemAssertReady() {
        exitIfFilesystemError();
}

public void systemRun() {
        printWelcomeMessage();
        interactiveUntilUserExits();
}

private void interactiveUntilUserExits() {
        while (true) {
                String userCommand = getUserCommand();
                String feedback = processUserCommand(userCommand);
                systemSave();
                systemPrint(feedback);
        }
}

// Basic Operations

private String processUserCommand(String userCommand) {
        String firstWord = getFirstWord(userCommand);
        CommandType commandType = getCommandType(firstWord);
        switch (commandType) {
        case ADD :
                return add(userCommand);
        case CLEAR :
                return clear();
```

```
		case DELETE :
			return delete(userCommand);
		case DISPLAY :
			return display();
		case EXIT :
			exit();
		case INVALID :
			return ERROR_INVALID_COMMAND;
		default :
			throw new Error(ERROR_FATAL);
		}
	}

	private CommandType getCommandType(String command) {
		String commandLowerCase = command.toLowerCase();
		if (isDictionaryContains(DICTIONARY_ADD, commandLowerCase)) {
			return CommandType.ADD;
		} else if (isDictionaryContains(DICTIONARY_CLEAR, commandLowerCase)) {
			return CommandType.CLEAR;
		} else if (isDictionaryContains(DICTIONARY_DELETE, commandLowerCase)) {
			return CommandType.DELETE;
		} else if (isDictionaryContains(DICTIONARY_DISPLAY, commandLowerCase)) {
			return CommandType.DISPLAY;
		} else if (isDictionaryContains(DICTIONARY_EXIT, commandLowerCase)) {
			return CommandType.EXIT;
		} else {
			return CommandType.INVALID;
		}
	}

	private String add(String userCommand) {
		String lineToAdd = removeFirstWord(userCommand);

		boolean isAddSuccess = logic.add(lineToAdd);
		if (isAddSuccess) {
			return String.format(FORMAT_ADD, logic.getFileName(), lineToAdd);
		} else {
			return ERROR_ADD;
		}
	}

	private String delete(String userCommand) {
		String lineNumberString = removeFirstWord(userCommand);
		int lineNumber = parseInt(lineNumberString);
		int actualLineNumber = lineNumber - 1;
		String deletedLine = logic.delete(actualLineNumber);

		boolean isDeleteSuccess = deletedLine != null;
		if (isDeleteSuccess) {
			return String.format(FORMAT_DELETE, logic.getFileName(),
					deletedLine);
		} else {
			return ERROR_DELETE;
		}
	}

	private String display() {
		List<String> list = logic.getList();

		boolean isListEmpty = list.isEmpty();
		if (isListEmpty) {
			return String.format(FORMAT_EMPTY, logic.getFileName());
		} else {
			return listToNumberedString(list);
		}
	}

	private String clear() {
		boolean isClearSuccess = logic.clear();
		if (isClearSuccess) {
			return String.format(FORMAT_CLEAR, logic.getFileName());
		} else {
			return ERROR_CLEAR;
		}
	}

	private void exit() {
		systemCleanUp();
		systemExit();
	}

	// Input Methods

	private String getUserCommand() {
		System.out.print(FORMAT_PROMPT);
		String userCommand = scanner.nextLine();
		return userCommand;
	}
```

```
private String getFirstWord(String userCommand) {
        String oneOrMoreSpaces = "\\s+";
        String[] splitUserCommand = userCommand.split(oneOrMoreSpaces);
        String firstWord = splitUserCommand[0];
        return firstWord;
}

private String removeFirstWord(String userCommand) {
        String blank = "";
        String firstWord = getFirstWord(userCommand);
        String removedFirstWord = userCommand.replaceFirst(firstWord, blank);
        String removedFirstWordTrimmed = removedFirstWord.trim();
        return removedFirstWordTrimmed;
}

private int parseInt(String intString) {
        try {
                int value = Integer.parseInt(intString);
                return value;
        } catch (Exception e) {
                return INVALID_INTEGER;
        }
}

private boolean isDictionaryContains(String[] dictionary, String command) {
        boolean isFound = false;
        for (int i = 0; i < dictionary.length; i++) {
                if (dictionary[i].equals(command)) {
                        isFound = true;
                        break;
                }
        }
        return isFound;
}

// Output Methods

private void printErrorAndExit(String errorMessage) {
        systemPrint(errorMessage);
        System.exit(1);
}

private void printWelcomeMessage() {
        System.out.printf(FORMAT_WELCOME, logic.getFileName());
}

private void systemPrint(String feedback) {
        System.out.println(feedback);
}

private String listToNumberedString(List<String> list) {
        String consolidateString = "";
        int lineNumber = 1;
        for (String content : list) {
                consolidateString += String.format(FORMAT_DISPLAY, lineNumber, content);
                lineNumber++;
        }
        String trimmedString = consolidateString.trim();
        return trimmedString;
}

// Initialization and Clean Up

private void initializeLogic(String[] args) {
        logic = new TextBuddyLogic(args[0]);
}

private void initializeTUI() {
        scanner = new Scanner(System.in);
}

private void systemSave() {
        logic.save();
}

private void systemCleanUp() {
        scanner.close();
}

private void systemExit() {
        System.exit(0);
}

// System Assertions

private void exitIfInvalidFileName(String[] args) {
        String fileName = args[0];
        exitIfIncorrectExtension(fileName);
        exitIfIncorrectLength(fileName);
```

```
                exitIfContainsInvalidCharacters(fileName);
        }

        private void exitIfFilesystemError() {
                exitIfFileNotReadableWritable();
        }

        private void exitIfFileNotReadableWritable() {
                boolean isFileReadableWritable = logic.isFileReadableWritable();
                if (!isFileReadableWritable) {
                        printErrorAndExit(ERROR_FILE_RW);
                }
        }

        private void exitIfInvalidArgumentCount(String[] args) {
                boolean isOnlyOneArgument = (args.length == 1);
                if (!isOnlyOneArgument) {
                        printErrorAndExit(ERROR_ARGUMENT_COUNT);
                }
        }

        private void exitIfIncorrectExtension(String fileName) {
                boolean isCorrectExtension = fileName.endsWith(VALID_EXTENSION);
                if (!isCorrectExtension) {
                        printErrorAndExit(ERROR_EXTENSION);
                }
        }

        private void exitIfIncorrectLength(String fileName) {
                String fileNameWithoutExtension = fileName.replaceAll(VALID_EXTENSION, "");
                boolean isLengthZero = (fileNameWithoutExtension.trim().length() == 0);
                if (isLengthZero) {
                        printErrorAndExit(ERROR_FILE_NAME_LENGTH);
                }
        }

        private void exitIfContainsInvalidCharacters(String fileName) {
                for (int i = 0; i < fileName.length(); i++) {
                        String currentCharacter = fileName.substring(i, i + 1);
                        boolean isInvalidCharacter = isDictionaryContains(
                                        DICTIONARY_NTFS_INVALID, currentCharacter);
                        if (isInvalidCharacter) {
                                printErrorAndExit(ERROR_INVALID_CHARACTERS);
                        }
                }
        }
}
```

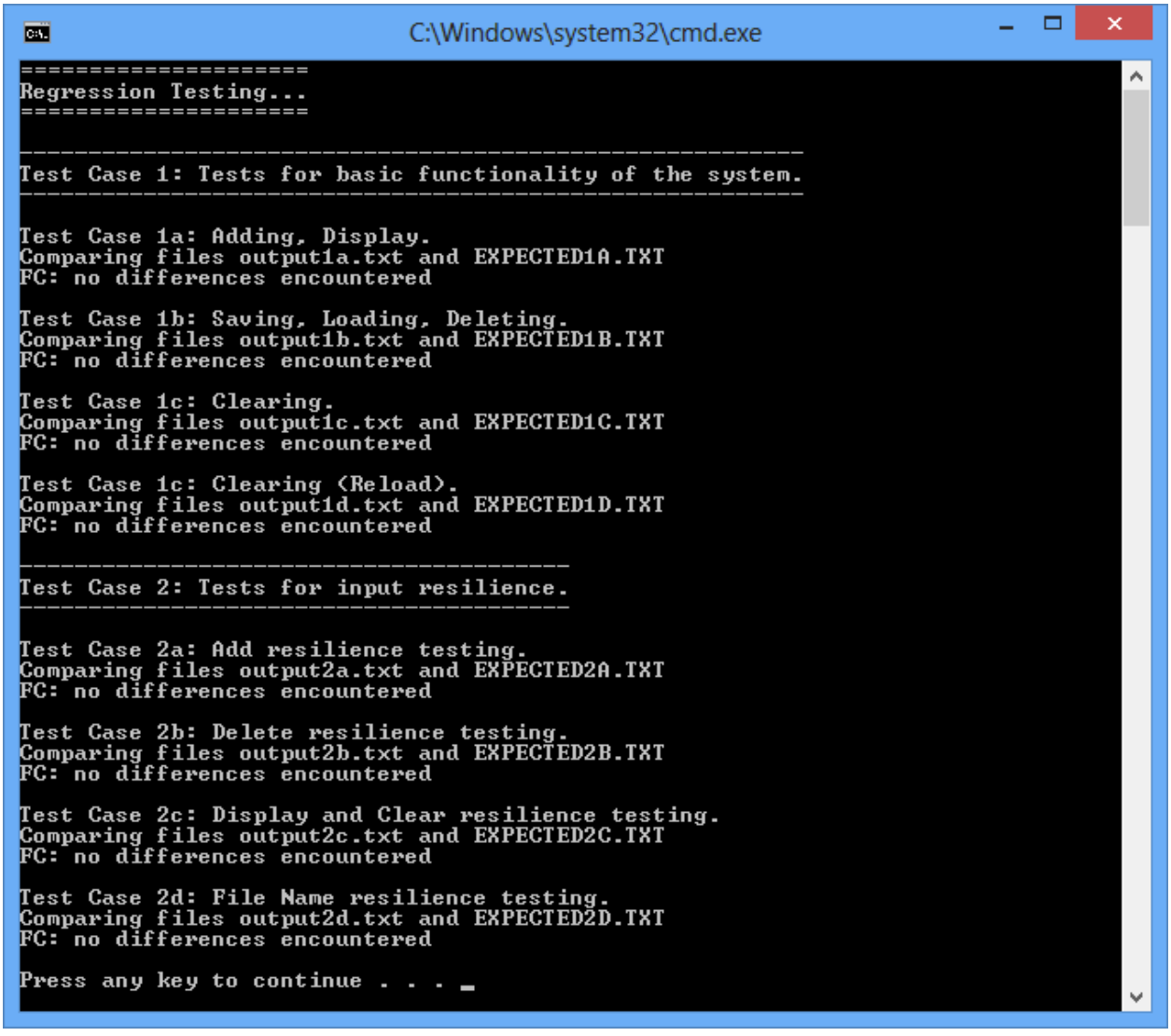
TextBuddyLogic.java

```
package ce1;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.List;

/**
 * This program handles the logic for the TextBuddy system.
 *
 * This implementation uses an ArrayList<String> as the data structure and is
 * able to perform add, delete, clear and save functionalities.
 *
 * The text file is only used as data storage and is written to disk by
 * invoking the save() method.
 *
 * @author Ang Kah Min, Kelvin
 */
public class TextBuddyLogic {

        // Warning Strings

        private static final String WARNING_READ = "Warning: Unable to read from file.";
        private static final String WARNING_WRITE = "Warning: Unable to create new file.";

        // Objects Declaration

        Path path;
        List<String> list;

        // Constructors

        public TextBuddyLogic(String fileName) {
```

```
        initializePath(fileName);
        initializeFile();
        initializeList();
}

// Initialization Methods

private void initializePath(String fileName) {
        this.path = Paths.get(fileName);
}

private void initializeFile() {
        boolean isExistingFile = Files.exists(path);
        if (!isExistingFile) {
                try {
                        // This would write a blank file if it does not exist.
                        Files.write(path, new byte[0], StandardOpenOption.CREATE,
                                        StandardOpenOption.TRUNCATE_EXISTING);
                } catch (IOException e) {
                        System.out.println(WARNING_WRITE);
                }
        }
}

private void initializeList() {
        try {
                list = Files.readAllLines(path);
        } catch (Exception e) {
                System.out.println(WARNING_READ);
        }
}

// Accessor Methods

public String getFilePath() {
        return this.path.toString();
}

public String getFileName() {
        return this.path.getFileName().toString();
}

public List<String> getList() {
        return this.list;
}

public int getListSize() {
        return this.list.size();
}

// Verification Methods

public boolean isFileReadableWritable() {
        return Files.isReadable(path) && Files.isWritable(path);
}

// Operation Methods

public boolean add(String content) {
        boolean success;
        try {
                boolean isNull = content == null;
                boolean isZeroLength = content.length() == 0;
                if (isNull || isZeroLength) {
                        throw new NullPointerException();
                }

                list.add(content);
                success = true;
        } catch (Exception e) {
                success = false;
        }
        return success;
}

public String delete(int lineNumber) {
        String deletedLine;
        try {
                deletedLine = list.remove(lineNumber);
        } catch (Exception e) {
                deletedLine = null;
        }
        return deletedLine;
}

public boolean clear() {
        boolean success;
        try {
                list.clear();
```

```
                        success = true;
                } catch (Exception e) {
                        success = false;
                }
                return success;
        }

        public boolean save() {
                boolean success;
                try {
                        Files.write(path, list, StandardOpenOption.CREATE,
                                        StandardOpenOption.TRUNCATE_EXISTING);
                        success = true;
                } catch (Exception e) {
                        success = false;
                }
                return success;
        }
}
```

Regression Testing.bat




```
@echo off
```

```
echo ====================
echo Regression Testing...
echo ====================
echo.
echo ------------------------------------------------------------
echo Test Case 1: Tests for basic functionality of the system.
echo ------------------------------------------------------------
echo.
echo Test Case 1a: Adding, Display.
java ce1.TextBuddy test.txt < input1a.txt > output1a.txt
fc output1a.txt expected1a.txt

echo Test Case 1b: Saving, Loading, Deleting.
java ce1.TextBuddy test.txt < input1b.txt > output1b.txt
fc output1b.txt expected1b.txt

echo Test Case 1c: Clearing.
java ce1.TextBuddy test.txt < input1c.txt > output1c.txt
fc output1c.txt expected1c.txt

echo Test Case 1c: Clearing (Reload).
java ce1.TextBuddy test.txt < input1d.txt > output1d.txt
fc output1d.txt expected1d.txt

echo ----------------------------------------
echo Test Case 2: Tests for input resilience.
echo ----------------------------------------
echo.
echo Test Case 2a: Add resilience testing.
java ce1.TextBuddy test.txt < input2a.txt > output2a.txt
fc output2a.txt expected2a.txt

echo Test Case 2b: Delete resilience testing.
java ce1.TextBuddy test.txt < input2b.txt > output2b.txt
fc output2b.txt expected2b.txt

echo Test Case 2c: Display and Clear resilience testing.
java ce1.TextBuddy test.txt < input2c.txt > output2c.txt
fc output2c.txt expected2c.txt

echo Test Case 2d: File Name resilience testing.
java ce1.TextBuddy > output2d.txt
java ce1.TextBuddy . >> output2d.txt
java ce1.TextBuddy .txt >> output2d.txt
java ce1.TextBuddy .txt.txt >> output2d.txt
java ce1.TextBuddy he:llo.txt >> output2d.txt
java ce1.TextBuddy he/llo.txt >> output2d.txt
java ce1.TextBuddy he?llo.txt >> output2d.txt
java ce1.TextBuddy he*llo.txt >> output2d.txt
fc output2d.txt expected2d.txt

pause
```

input1a.txt – Adding, Display

```
add A common mistake that people make when trying to design something
completely foolproof was to underestimate the ingenuity of complete
fools.
display
add Programming today is a race between software engineers striving to
build bigger and better idiot-proof programs, and the Universe trying
to produce bigger and better idiots. So far, the Universe is winning.
display
add The first 90 percent of the code accounts for the first 90 percent
of the development time...The remaining 10 percent of the code
accounts for the other 90 percent of the development time.
display
exit
```

input1b.txt – Saving, Loading, Deleting

```
display
delete 3
display
delete 1
display
exit
```

input1c.txt - Clearing

```
clear
display
exit
```

input1d.txt – Clearing (Reload)

```
display
exit
```

input2a.txt – Add resilience testing.

```
add
add
add
add add
add -a
display
clear
```

```
exit
```

input2b.txt – Delete resilience testing.

```
add line for testing
delete
delete
delete
delete delete
display
delete -d
display
delete -5
display
delete -1
display
delete 0
display
delete 1
display
delete 5
display
exit
```

input2c.txt – Display and Clear resilience testing.

```
add line for testing
display
display
display
display -di
clear   a
display
clear
display
clear
display
exit
```

expected1a.txt – Adding, Display

```
Welcome to TextBuddy. test.txt is ready for use
command: added to test.txt: "A common mistake that people make when
trying to design something completely foolproof was to underestimate
the ingenuity of complete fools."
command: 1. A common mistake that people make when trying to design
something completely foolproof was to underestimate the ingenuity of
complete fools.
command: added to test.txt: "Programming today is a race between
software engineers striving to build bigger and better idiot-proof
```

```
programs, and the Universe trying to produce bigger and better idiots.
So far, the Universe is winning."
command: 1. A common mistake that people make when trying to design
something completely foolproof was to underestimate the ingenuity of
complete fools.
2. Programming today is a race between software engineers striving to
build bigger and better idiot-proof programs, and the Universe trying
to produce bigger and better idiots. So far, the Universe is winning.
command: added to test.txt: "The first 90 percent of the code accounts
for the first 90 percent of the development time...The remaining 10
percent of the code accounts for the other 90 percent of the
development time."
command: 1. A common mistake that people make when trying to design
something completely foolproof was to underestimate the ingenuity of
complete fools.
2. Programming today is a race between software engineers striving to
build bigger and better idiot-proof programs, and the Universe trying
to produce bigger and better idiots. So far, the Universe is winning.
3. The first 90 percent of the code accounts for the first 90 percent
of the development time...The remaining 10 percent of the code
accounts for the other 90 percent of the development time.
command:
```

expected1b.txt – Saving, Loading, Deleting

```
Welcome to TextBuddy. test.txt is ready for use
command: 1. A common mistake that people make when trying to design
something completely foolproof was to underestimate the ingenuity of
complete fools.
2. Programming today is a race between software engineers striving to
build bigger and better idiot-proof programs, and the Universe trying
to produce bigger and better idiots. So far, the Universe is winning.
3. The first 90 percent of the code accounts for the first 90 percent
of the development time...The remaining 10 percent of the code
accounts for the other 90 percent of the development time.
command: deleted from test.txt: "The first 90 percent of the code
accounts for the first 90 percent of the development time...The
remaining 10 percent of the code accounts for the other 90 percent of
the development time."
command: 1. A common mistake that people make when trying to design
something completely foolproof was to underestimate the ingenuity of
complete fools.
2. Programming today is a race between software engineers striving to
build bigger and better idiot-proof programs, and the Universe trying
to produce bigger and better idiots. So far, the Universe is winning.
command: deleted from test.txt: "A common mistake that people make
when trying to design something completely foolproof was to
underestimate the ingenuity of complete fools."
command: 1. Programming today is a race between software engineers
striving to build bigger and better idiot-proof programs, and the
Universe trying to produce bigger and better idiots. So far, the
Universe is winning.
command:
```

expected1c.txt - Clearing

```
Welcome to TextBuddy. test.txt is ready for use
command: all content deleted from test.txt
command: test.txt is empty
command:
```

expected1d.txt – Clearing (Reload)

```
Welcome to TextBuddy. test.txt is ready for use
command: test.txt is empty
command:
```

expected2a.txt – Add resilience testing.

```
Welcome to TextBuddy. test.txt is ready for use
command: Add command has failed.
command: Add command has failed.
command: Add command has failed.
command: added to test.txt: "add"
command: added to test.txt: "-a"
command: 1. add
2. -a
command: all content deleted from test.txt
command:
```

expected2b.txt – Delete resilience testing.

```
Welcome to TextBuddy. test.txt is ready for use
command: added to test.txt: "line for testing"
command: Delete command failed.
command: Delete command failed.
command: Delete command failed.
command: Delete command failed.
command: 1. line for testing
command: Delete command failed.
command: 1. line for testing
command: Delete command failed.
command: 1. line for testing
command: Delete command failed.
command: 1. line for testing
command: Delete command failed.
command: 1. line for testing
command: deleted from test.txt: "line for testing"
command: test.txt is empty
command: Delete command failed.
command: test.txt is empty
command:
```

expected2c.txt – Display and Clear resilience testing.

```
Welcome to TextBuddy. test.txt is ready for use
command: added to test.txt: "line for testing"
command: 1. line for testing
command: 1. line for testing
command: 1. line for testing
command: 1. line for testing
command: all content deleted from test.txt
command: test.txt is empty
command: all content deleted from test.txt
command: test.txt is empty
command: all content deleted from test.txt
command: test.txt is empty
command:
```

expected2d.txt – File Name resilience testing.

```
Wrong argument count supplied.
Invalid file extension.
File name should be at least 1 character long.
File name should be at least 1 character long.
File name contains invalid characters.
File name contains invalid characters.
File name contains invalid characters.
File name contains invalid characters.
```