CIS 163-02

Project 4: Text Editor

Michael Hartung & Matthew Armand

Submitted: 15 April 2014

Submitted to: Professor Jerry Scripps

We, those named above as submitters of this project, pledge that all the source code contained herein is of our own design and work.

```java
import java.util.Scanner;
/**********************************************************************
 * Main class to run Ted Editor
 * @author Michael Hartung
 */
public class EditorMain {
    /**********************************************************************
     * Runs Ted Editor program
     * @param args
     */
    public static void main(String[] args) {
        String text = "";
        Editor ted = new Editor();
        Scanner sc = new Scanner(System.in);
        System.out.println("Welcome to the Ted Editor! If you need help, type
the letter h.");
        while(ted.getActive()) {
            text = sc.nextLine();
            ted.processCommand(text);
        }
    }
}
```

```java
/**********************************************************************
 * Class for overall Editor object in Ted Editor; processes user input
 * @author Michael Hartung, Matthew Armand
 */
public class Editor implements IEditor {

    //Process and "running" boolean instance variables
    private CmdProcess process;
    private boolean active = true;
    //End the program by setting this boolean to false,
    //use while loop to keep program running in a main method

    /**********************************************************************
     * Constructs new Editor object, instantiates command processing
     */
    public Editor() {
        process = new CmdProcess();
    }

    /**********************************************************************
     * Switch statement to process user input commands
     */
    public void processCommand(String command) {
        String[] token = command.split(" ");
        try {
        switch (token[0].trim().toLowerCase()) {

            case "b":   process.insertBefore(command.substring(2));
                        break;

            case "e":   process.insertLast(command.substring(2));
                        break;

            case "i":   process.insertAfter(command.substring(2));
                        break;

            case "m":   if (token.length == 2) {
                            process.down(Integer.parseInt(token[1].trim()));
                        }
                        else {
                            process.downOnePos();
                        }
                        break;

            case "u":   if (token.length == 2) {
                            process.up(Integer.parseInt(token[1].trim()));
                        }
                        else {
```

```
                process.upOnePos();
            }
            break;

        case "r":   if (token.length == 2) {
                process.remove(Integer.parseInt(token[1].trim()));
            }
            else {
                process.removeCurrentLine();
            }
            break;

        case "d":   if (token.length == 3) {
                int x = Integer.parseInt(token[1].trim());
                int y = Integer.parseInt(token[2].trim());
                process.display(x, y);
            } else if (token.length == 1) {
                process.displayFile();
            }
            break;

        case "c":   if (process.isSaved())
                process.clearFile();
            else {
                System.out.print("File not saved! ");
                System.out.println("Save (s) or force-clear (!c)");
            }
            break;

        case "!c":  process.clearFile();
            break;

        case "s":   process.saveFile(token[1].trim());
            break;

        case "l":   if (process.isSaved())
                process.loadFile(token[1].trim());
            else {
                System.out.print("File not saved! ");
                System.out.println("Save (s) or force-load (!l)");
            }
            break;

        case "!l":  process.loadFile(token[1].trim());
            break;

        case "h":   System.out.println(process.showHelp());
            break;
```

```
            case "x":   if (process.isSaved()) {
                            System.out.println("Closing Ted Editor. Goodbye!");
                            active = false;
                        } else {
                            System.out.print("File not saved! ");
                            System.out.println("Save (s) or force-quit (!x)");
                        }
                        break;

            case "!x":  System.out.println("Closing Ted Editor. Goodbye!");
                        active = false;
                        break;

            case "cut": process.cutSelection(Integer.parseInt(token[1].trim()),
                                             Integer.parseInt(token[2].trim()),
                                             Integer.parseInt(token[3].trim()));
                        break;

            case "pas": process.pasteClipboard(Integer.parseInt(token[1].trim
()));
                        break;

            default:    System.out.println("Invalid Command!");
                        break;
        }
        } catch(NumberFormatException e) {
            System.out.println("Invalid Command! Numeric parameters only.");
        } catch (IndexOutOfBoundsException f) {
            System.out.println("Invalid Command! Enter a string to use.");
        } catch (NullPointerException n) {
            System.out.println("Object Not Found! Please create things before
attmpting to use them.");
        }
    }

    /**********************************************************************
     * Gets line of lineNumber matching the input parameter
     * @param lineNbr number of line to be fetched
     * @return line matching input line number
     */
    public String getLine(int lineNbr) {
        Line l = process.getList().getHead();
        for (int i=1; i<=lineNbr; i++)
            l = l.getNext();
        return l.toString();
    }
```

```
    /*************************************************************************
     * Gets line currently labeled as current
     * @return current Line object
     */
    public String getCurrentLine() {
        return process.getList().getCurrent().toString();
    }

    /*************************************************************************
     * Gets running status of program
     * @return true if still running, false if program has been exited
     */
    public boolean getActive() {
        return active;
    }

    /*************************************************************************
     * Gets CmdProcess object (used here for JUnit testing
     * @return CmdProcess object
     */
    public CmdProcess getProcess() {
        return process;
    }
}
```

```java
import java.io.*;
/**************************************************************************
 * CmdProcess: class for processing commands from text editor
 * @author Michael Hartung, Matthew Armand
 */
public class CmdProcess {
    //LineList of input lines, boolean saved instance variables
    private LineList tedList;
    private boolean saved;
    private CutPaste clipboard;

    /**************************************************************************
     * Standard constructor, instantiates new process object
     */
    public CmdProcess() {
        tedList = new LineList();
        clipboard = new CutPaste();
        saved = false;
    }

    /**************************************************************************
     * Checks whether the file has been saved since last alteration
     * @return true if unchanged since last saved, false if altered
     */
    public boolean isSaved() {
        return saved;
    }

    /**************************************************************************
     * Gets linelist object within Process class
     * @return LineList object (tedList)
     */
    public LineList getList () {
        return tedList;
    }

    /**************************************************************************
     * Inserts a new line before the current line in the list
     * @param newLine String of text to comprise new line
     */
    public void insertBefore(String newLine) {
        tedList.insertBefore(newLine);
        saved = false;
    }

    /**************************************************************************
     * Inserts a new line after the current line in the list
     * @param newLine String of text to comprise new line
```

```
 */
public void insertAfter(String newLine) {
    tedList.insertAfter(newLine);
    saved = false;
}

/**********************************************************************
 * Inserts a new line at the end of the file
 * @param newLine String of text to comprise new line
 */
public void insertLast(String newLine) {
    tedList.insertLast(newLine);
    saved = false;
}

/**********************************************************************
 * Moves current line indicator down a certain number of positions
 * @param numDown Number of lines to move down
 */
public void down(int numDown) {
    while(numDown > 0) {
        downOnePos();
        numDown--;
    }
}

/**********************************************************************
 * Moves the current line indicator down one position
 */
public void downOnePos() {
    tedList.down();
}

/**********************************************************************
 * Moves the current line indicator up a certain number of positions
 * @param numUp Number of lines to move up
 */
public void up(int numUp) {
    while(numUp > 0) {
        upOnePos();
        numUp--;
    }
}

/**********************************************************************
 * Moves the current line indicator up one position
 */
public void upOnePos() {
```

```
    tedList.up();
}

/*****************************************************************
 * Removes the current line from the list
 */
public void removeCurrentLine() {
    tedList.remove();
    saved = false;
}

/*****************************************************************
 * Removes a number of Lines from the list
 */
public void remove(int numLines) {
    for(int i = 0; i < numLines; i++) {
        removeCurrentLine();
    }
}
/*****************************************************************
 * Displays all the lines in the list in formatted order
 */
public void displayFile() {
    System.out.println(tedList.toString());
}

/*****************************************************************
 * Displays lines x to y in the list in formatted order
 * @param x starting line to display
 * @param y ending line to display
 */
public void display (int x, int y) {
    if (x<1 || y<x) {
        System.out.println("Invalid Command!");
    }
    else {
        System.out.println(tedList.display(x,y));
    }
}

/*****************************************************************
 * Clears file and removes all existing lines
 */
public void clearFile() {
    tedList = new LineList();
    saved = false;
}
```

```java
/*****************************************************************
 * Saves contents of file to a file in directory structure
 * @param fileName Name of the file to be saved
 */
public void saveFile(String fileName) {
    PrintWriter p = null;
    try {
        p=new PrintWriter(new BufferedWriter(new FileWriter(fileName)));
        Line tmp = tedList.getHead();
        while (tmp!=null){
            p.print(tmp);
            tmp=tmp.getNext();
        }
        saved = true;
        p.close();
    } catch (IOException e) {
        System.out.println("Error while writing file!");
    }
}

/*****************************************************************
 * Loads contents of the file into current buffer
 * @param fileName Name of the file to be loaded
 */
public void loadFile(String fileName) {
    try {
        Scanner sc = new Scanner (new File(fileName));
        clearFile();
        while (sc.hasNextLine()) {
            tedList.insertAfter(sc.nextLine());
        }
        tedList.setCurrent(tedList.getHead());
        saved = true;
        sc.close();
    } catch (FileNotFoundException e) {
        System.out.println("File not found");
    }
    saved = false;
}

/*****************************************************************
 * Displays the help dialog with list of commands and functions
 */
public String showHelp() {
    String out = "";
    out += ("Welcome to Text Editor Help!\n");
    out += ("Command:      Function:\n");
    out += ("b 'sentence'  Insert sentence before current");
```

```
        out += (" line, make inserted line current\n");
        out += ("i 'sentence'  Insert sentence after current");
        out += (" line, make inserted line current\n");
        out += ("e 'sentence'  Insert sentence after last line, make inserted
line current\n");
        out += ("m             Move cursor down 1 position\n");
        out += ("m #           Move cursor down # positions\n");
        out += ("u             Move cursor up 1 position\n");
        out += ("u #           Move cursor down # positions\n");
        out += ("r             Remove current line. Next line becomes ");
        out += ("current, unless no next \n             line, then previous
becomes ");
        out += ("current\n");
        out += ("r #           Remove # lines, starting at current\n");
        out += ("d             Display all lines with line numbers\n");
        out += ("d # *         Display lines # to * with line numbers\n");
        out += ("cut # $ *     Cut lines # to $ to clipboard *\n");
        out += ("pas *         Paste clipboard * before current position\n");
        out += ("c             Clear all lines in the file\n");
        out += ("!c            Force clear all lines in the file\n");
        out += ("s 'filename'  Save contents to specified text file\n");
        out += ("l 'filename'  Load contents of file into current buffer\n");
        out += ("!l 'filename' Force load contents of file into current buffer
\n");
        out += ("h             Display this help page\n");
        out += ("x             Exit the editor\n");
        out += ("!x            Force exit the editor");
        return out;
    }

    /*************************************************************************
     * Cuts selection onto a clipboard to be pasted
     */
    public void cutSelection(int startLine, int endLine, int clipboardNum) {
        LineList temp = new LineList();
        tedList.setCurrent(tedList.getHead());
            // Finding Starting Position
            for(int i = 1; i < startLine; i++) {
                if(tedList.getCurrent().getNext() != null) {
                    tedList.setCurrent(tedList.getCurrent().getNext());
                }
                else {
                    System.out.println("Invalid Starting Point!");
                    i = endLine;
                }
            }
            // After Start has been found, copy lines into temp line list and
remove lines
```

```
        for(int i = startLine; i <= endLine; i++) {
            if(tedList.getCurrent() != null) {
                temp.insertAfter(tedList.getCurrent().toString());
                removeCurrentLine();
            }
            else {
                System.out.println("Invalid Ending Point!");
                i = endLine;
            }
        }
    if(temp.getHead() != null) {
        clipboard.add(temp, clipboardNum);
    }
    else {
        System.out.println("Cut failed!");
    }
    saved = false;
}

/**********************************************************************
 * Pastes selection from clipboard into the file
 */
public void pasteClipboard(int clipboardNum) {
    LineList temp = clipboard.getBoard(clipboardNum);
    String mod;
    temp.setCurrent(temp.getHead());
    if(tedList.getHead() == null) {
        while(temp.getCurrent() != null) {
            mod = temp.getCurrent().toString();
            mod = mod.substring(0,mod.length()-2);
            tedList.insertAfter(mod);
            temp.setCurrent(temp.getCurrent().getNext());
        }
    }
    else if(tedList.getCurrent().getPrev() != null) {
        upOnePos();
        while(temp.getCurrent() != null) {
            mod = temp.getCurrent().toString();
            mod = mod.substring(0,mod.length()-2);
            tedList.insertAfter(mod);
            temp.setCurrent(temp.getCurrent().getNext());
        }
    }
    else {
        mod = temp.getCurrent().toString();
        mod = mod.substring(0,mod.length()-2);
        tedList.insertBefore(mod);
        temp.setCurrent(temp.getCurrent().getNext());
```

```
            while(temp.getCurrent() != null) {
                mod = temp.getCurrent().toString();
                mod = mod.substring(0,mod.length()-2);
                tedList.insertAfter(mod);
                temp.setCurrent(temp.getCurrent().getNext());
            }
        }
        if (tedList.getCurrent().getNext() != null)
            downOnePos();
        saved = false;
    }
}
```

```java
public class CutPaste {
    private LineList head, current;

    public CutPaste() {
        head = null;
        current = null;
    }

    public void insertBefore(LineList newList) {
        if(head == null) {
            head = newList;
            current = newList;
        }
        else if(current.getPrev() == null) {
            head = newList;
            current.setPrev(newList);
            newList.setNext(current);
            current = newList;
        }
        else {
            current.getPrev().setNext(newList);
            newList.setPrev(current.getPrev());
            current.setPrev(newList);
            newList.setNext(current);
            current = newList;
        }
    }

    public void insertAfter(LineList newList) {
        if(head == null) {
            head = newList;
            current = newList;
        }
        else if(current.getNext() == null) {
            current.setNext(newList);
            newList.setPrev(current);
            current = newList;
        }
        else {
            current.getNext().setPrev(newList);
            newList.setNext(current.getNext());
            current.setNext(newList);
            newList.setPrev(current);
            current = newList;
        }
    }

    public void add(LineList newBoard, int newBoardNum) {
```

```
    newBoard.setClipNum(newBoardNum);
    if(head == null) {
        insertAfter(newBoard);
    }
    else if(current.getNext() == null) {
        insertAfter(newBoard);
    }
    else if(current.getClipNum() > newBoardNum) {
        insertBefore(newBoard);
    }
    else if(current.getClipNum() < newBoardNum) {
        current = current.getNext();
        add(newBoard, newBoardNum);
    }
}

private void removeCurrent() {
    if (current == null) {
        System.out.println("Nothing to Cut!");
    }
    else if(current.getNext() != null) {
        if(current.getPrev() != null) {
            current.getPrev().setNext(current.getNext());
            current.getNext().setPrev(current.getPrev());
            current = current.getNext();
        }
        else {
            current.getNext().setPrev(null);
            head = current.getNext();
            current = current.getNext();
        }
    }
    else if(current.getPrev() != null) {
        current.getPrev().setNext(null);
        current = current.getPrev();
    }
    else {
        head = current = null;
    }
}

public LineList getBoard(int clipBoardNum) {
    current = head;
    LineList retrieve = null;
    while(current != null && retrieve == null) {
        if(current.getClipNum() == clipBoardNum) {
            retrieve = current;
            removeCurrent();
```

```
            }
            else {
                current = current.getNext();
            }
        }
        return retrieve;
    }
}
```

```java
/**********************************************************************
 * Class for LineList, providing overall list of Linked List structure
 * @author Michael Hartung, Matthew Armand
 */
public class LineList {
    //Current and head object references for line objects
    private Line head, current;
    private LineList next, prev;
    private int clipNum;
    /**********************************************************************
     * Constructs a new, empty LineList object
     */
    public LineList() {
        head = null;
        current = null;
        next = null;
        prev = null;
    }

    /**********************************************************************
     * Inserts a new line before the current line
     * @param newLine content to be set as the text of new line
     */
    public void insertBefore(String newLine) {
        Line addLine = new Line(newLine);
        if (current == null) {
            current = head = addLine;
        }
        else if (current.getPrev() != null) {
            addLine.setNext(current);
            addLine.setPrev(current.getPrev());
            current.getPrev().setNext(addLine);
            current.setPrev(addLine);
            current = addLine;
        }
        else {
            addLine.setNext(head);
            head.setPrev(addLine);
            current = head = addLine;
        }
    }

    /**********************************************************************
     * Inserts a new line at the end of the list
     * @param newLine content to be set as the text of new line
     */
    public void insertLast(String newLine) {
        while(current.getNext() != null) {
```

```
            down();
        }
        insertAfter(newLine);
    }

    /**************************************************************************
     * Inserts a new line after the current line
     * @param newLine content to be set as the text of new line
     */
    public void insertAfter(String newLine) {
        Line addLine = new Line(newLine);
        if (current == null) {
            current = head = addLine;
        }
        else if (current.getNext() != null) {
            addLine.setNext(current.getNext());
            addLine.setPrev(current);
            current.getNext().setPrev(addLine);
            current.setNext(addLine);
            current = addLine;
        }
        else {
            addLine.setPrev(current);
            current.setNext(addLine);
            current = addLine;
        }
    }

    /**************************************************************************
     * Moves current line indicator down one position
     */
    public void down() {
        if(current.getNext() != null) {
            current = current.getNext();
        }
        else {
            System.out.println("Bottom Line Reached!");
        }
    }

    /**************************************************************************
     * Moves current line indicator up one position
     */
    public void up() {
        if(current.getPrev() != null) {
            current = current.getPrev();
        }
        else {
```

```
        System.out.println("Top Line Reached!");
    }
}

/**************************************************************************
 * Removes current line from the list
 */
public void remove() {
    if (current == null) {
        System.out.println("Nothing to remove!");
    }
    else if(current.getNext() != null) {
        if(current.getPrev() != null) {
            current.getPrev().setNext(current.getNext());
            current.getNext().setPrev(current.getPrev());
            current = current.getNext();
        }
        else {
            current.getNext().setPrev(null);
            head = current.getNext();
            current = current.getNext();
        }
    }
    else if(current.getPrev() != null) {
        current.getPrev().setNext(null);
        current = current.getPrev();
    }
    else {
        head = current = null;
    }
}

/**************************************************************************
 * Sets the input line as the current line indicator in the list
 * @param l Line to be set as current
 */
public void setCurrent (Line l) {
    current = l;
}

/**************************************************************************
 * Gets the line currently set as current within the list
 * @return current Line in list
 */
public Line getCurrent () {
    return current;
}
```

```java
/**************************************************************************
 * Gets the line at the head of the list
 * @return object reference to head of the list
 */
public Line getHead () {
    return head;
}

/**************************************************************************
 * Displays certain lines from x to y in the list
 * @param x starting point to display
 * @param y ending point to display
 * @return string representation of desired lines in list
 */
public String display (int x, int y) {
    String result = "";
    Line pass = head;
    int lineNumber = 1;
    boolean done = false;
    while (pass != null && !done) {
        if (lineNumber == x) {
            while (pass != null && !done) {
                if (pass == current) {
                    result += "--> ";
                }
                else {
                    result += "    ";
                }
                result += lineNumber + ": ";
                result += pass.toString();
                pass = pass.getNext();
                lineNumber++;
                if (lineNumber == y+1) {
                    done = true;
                }
            }
        }
        lineNumber++;
        try {
            pass = pass.getNext();
        } catch (NullPointerException e) {
            done = true;
        }
    }
    return result;
}

/**************************************************************************
```

```
     * Returns a string representation of the list object
     */
    public String toString() {
        String result = "";
        Line pass = head;
        int lineNumber = 1;
        while(pass != null) {
            if(pass == current) {
                result += "--> ";
            }
            else {
                result += "    ";
            }
            result += lineNumber + ": ";
            result += pass.toString();
            pass = pass.getNext();
            lineNumber++;
        }
        return result;
    }

    public void setPrev(LineList pPrev) {
        prev = pPrev;
    }

    public void setNext(LineList pNext) {
        next = pNext;
    }

    public LineList getPrev() {
        return prev;
    }

    public LineList getNext() {
        return next;
    }

    public int getClipNum() {
        return clipNum;
    }

    public void setClipNum(int pClipNum) {
        clipNum = pClipNum;
    }
}
```

```java
/*****************************************************************************
 * Class for Line object within Ted Editor Program as the base object in
 * the Linked List structure
 * @author Michael Hartung, Matthew Armand
 */
public class Line {
    //Content string, next and previous reference objects
    private String content;
    private Line next;
    private Line prev;

    /*************************************************************************
     * Constructs a new blank line object
     */
    public Line() {
        content = "\n";
    }

    /*************************************************************************
     * Constructs a new line object with parameter text as content
     * @param newLine String to be set as line content
     */
    public Line(String newLine) {
        content = newLine + "\n";
    }

    /*************************************************************************
     * Sets the next object reference to the parameter line
     * @param l Line to be set as next in the Linked List
     */
    public void setNext(Line l) {
        next = l;
    }

    /*************************************************************************
     * Gets the next object reference in the Linked List structure
     * @return Next Line in the Linked List
     */
    public Line getNext() {
        return next;
    }

    /*************************************************************************
     * Sets the previous object reference in the Linked List structure
     * @param l Line to be set as previous in the Linked List
     */
    public void setPrev(Line l) {
        prev = l;
```

```
    }

    /**********************************************************************
     * Gets the previous object reference in the Linked List structure
     * @return Previous Line in the Linked List
     */
    public Line getPrev() {
        return prev;
    }

    /**********************************************************************
     * Returns string representation of the Line object
     */
    public String toString() {
        return content;
    }
}
```

```java
*   Editor Test Class
import static org.junit.Assert.*;

public class EditorTest {

    @Test
    public void testInsertAfter() {
        Editor e = new Editor();
        e.processCommand("i Test 1");
        e.processCommand("i Test 2");
        assertEquals("    1: Test 1\n--> 2: Test 2\n",
                e.getProcess().getList().toString());
    }

    @Test
    public void testInsertBefore() {
        Editor e = new Editor();
        e.processCommand("b Test 1");
        e.processCommand("b Test 2");
        assertEquals("--> 1: Test 2\n    2: Test 1\n",
                e.getProcess().getList().toString());
    }

    @Test
    public void testInsertLast() {
        Editor e = new Editor();
        e.processCommand("b Test 1");
        e.processCommand("b Test 2");
        e.processCommand("e Test 3");
        assertEquals("    1: Test 2\n    2: Test 1\n--> 3: Test 3\n",
                e.getProcess().getList().toString());
    }

    @Test
    public void testMoveUp() {
        Editor e = new Editor();
        e.processCommand("b Test 1");
        e.processCommand("b Test 2");
        e.processCommand("e Test 3");
        e.processCommand("u");
        assertEquals("Test 1\n", e.getCurrentLine());
    }

    @Test
    public void testMoveUpMult() {
        Editor e = new Editor();
        e.processCommand("b Test 1");
        e.processCommand("b Test 2");
```

```
    e.processCommand("e Test 3");
    e.processCommand("u 2");
    assertEquals("Test 2\n", e.getCurrentLine());
}

@Test
public void testMoveDown() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("b Test 2");
    e.processCommand("b Test 3");
    e.processCommand("b Test 4");
    e.processCommand("m");
    assertEquals("Test 3\n", e.getCurrentLine());
}

@Test
public void testMoveDownMult() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("b Test 2");
    e.processCommand("b Test 3");
    e.processCommand("b Test 4");
    e.processCommand("m 2");
    assertEquals("Test 2\n", e.getCurrentLine());
}

@Test
public void testRemove() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("i Test 2");
    e.processCommand("i Test 3");
    e.processCommand("r");
    assertEquals("    1: Test 1\n--> 2: Test 2\n",
            e.getProcess().getList().toString());
}

@Test
public void testRemoveMult() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("i Test 2");
    e.processCommand("i Test 3");
    e.processCommand("r 2");
    assertEquals("--> 1: Test 1\n",
            e.getProcess().getList().toString());
}
```

```java
@Test
public void testDisplay() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("i Test 2");
    e.processCommand("i Test 3");
    e.processCommand("d");
    assertEquals("    1: Test 1\n    2: Test 2\n--> 3: Test 3\n",
            e.getProcess().getList().toString());
}

@Test
public void testDisplaySpecific() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("i Test 2");
    e.processCommand("i Test 3");
    String t1 = e.getProcess().getList().display(2,3);
    assertEquals("    2: Test 2\n--> 3: Test 3\n", t1);
}

@Test
public void testClear() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("i Test 2");
    e.processCommand("i Test 3");
    e.processCommand("!c");
    assertEquals("", e.getProcess().getList().toString());
}

@Test
public void testSaveLoad() {
    Editor e = new Editor();
    e.processCommand("i Test 1");
    e.processCommand("i Test 2");
    e.processCommand("i Test 3");
    e.processCommand("s file1");
    Editor f = new Editor();
    f.processCommand("!l file1");
    f.processCommand("m 2");
    assertEquals(e.getProcess().getList().toString(),
            f.getProcess().getList().toString());
}

@Test
public void testExit() {
```

```
        Editor e = new Editor();
        e.processCommand("!x");
        assertFalse(e.getActive());
    }

    @Test
    public void testCut() {
        Editor e = new Editor();
        e.processCommand("i Test 1");
        e.processCommand("cut 1 1 1");
        assertEquals("", e.getProcess().getList().toString());
    }

    @Test
    public void testPaste() {
        Editor e = new Editor();
        e.processCommand("i Test 1");
        String t1 = e.getProcess().getList().toString();
        e.processCommand("cut 1 1 1");
        e.processCommand("pas 1");
        assertEquals(t1, e.getProcess().getList().toString());
    }

    @Test
    public void testHelp() {
        Editor e = new Editor();
        String t1 = e.getProcess().showHelp();
        String t2 = "";
        t2 += ("Welcome to Text Editor Help!\n");
        t2 += ("Command:      Function:\n");
        t2 += ("b 'sentence'  Insert sentence before current");
        t2 += (" line, make inserted line current\n");
        t2 += ("i 'sentence'  Insert sentence after current");
        t2 += (" line, make inserted line current\n");
        t2 += ("e 'sentence'  Insert sentence after last line, make inserted
line current\n");
        t2 += ("m             Move cursor down 1 position\n");
        t2 += ("m #           Move cursor down # positions\n");
        t2 += ("u             Move cursor up 1 position\n");
        t2 += ("u #           Move cursor down # positions\n");
        t2 += ("r             Remove current line. Next line becomes ");
        t2 += ("current, unless no next \n             line, then previous
becomes ");
        t2 += ("current\n");
        t2 += ("r #           Remove # lines, starting at current\n");
        t2 += ("d             Display all lines with line numbers\n");
        t2 += ("d # *         Display lines # to * with line numbers\n");
        t2 += ("cut # $ *     Cut lines # to $ to clipboard *\n");
```

```
        t2 += ("pas *         Paste clipboard * before current position\n");
        t2 += ("c             Clear all lines in the file\n");
        t2 += ("!c            Force clear all lines in the file\n");
        t2 += ("s 'filename'  Save contents to specified text file\n");
        t2 += ("l 'filename'  Load contents of file into current buffer\n");
        t2 += ("!l 'filename' Force load contents of file into current buffer
\n");
        t2 += ("h             Display this help page\n");
        t2 += ("x             Exit the editor\n");
        t2 += ("!x            Force exit the editor");
        assertEquals(t1,t2);
    }
}
```