```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;
```

```java
            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;
    }
}
```