CIS 163 – Computer Science II, Fall 2014

Project 4: "TED" Editor (Due: December 1, 2014)

88/100

Student Name Aaron Teague	
Date Submitted, Days Late, and Late Penalty	

Graded Item	Points	Points Secured	Comments
Javadoc and coding style/technique (http://www.cis.gvsu.edu/studentsupport/javaguide) • Code Indentation (auto format in IDE) • Naming Conventions (see Java style guide) • Proper access modifiers for fields/methods • Use of helper (private) methods	10		-4
Step 1 Basic Functionality • commands: b, i • commands: m, u • command: r • commands: l, s • commands: d, c • commands: h, x • JUnit tests	55		CRASH DOES NOT WORK SOME ISSUES
Step 2 Functionality • command: e • commands: m n and u n • command: r n • command: d n1 n2 • JUnit tests	20		
Step 3 Functionality • command: rev • commands: cut paste • JUnit tests	15		
Extra Credit Functionality • command: ud • JUnit tests	☺		**Pair programming required Step 3**
Max Possible Points	**100**		

Additional Comments: BAD Error Checking ISSUES

```java
package package1;
import java.util.Scanner;
/**
 * EditorCLI class provides a text-based user interface to the editor
 * supporting a read-process-print loop.
 *
 * @author YOUR NAME(S)
 *
 */
public class EditorCLI {

    public static void main(String[] args) {
        // COMPLETE THIS METHOD TO IMPLEMENT A
        // COMMAND-LINE USER INTERFACE (CLI) TO THE EDITOR
        Editor ed = new Editor();
        String x;
        Scanner scan = new Scanner(System.in);

        while(true){
            try {

                System.out.print("ted> ");
                x = scan.nextLine();
                ed.processCommand(x);

            }
            catch (EditorException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }
    }

}

```

```java
package package1;
import java.util.*;

/**
 * Editor class implements a simple line-oriented editor.
 *
 * @author YOUR NAME(S)
 *
 */
public class Editor implements IEditor {

    private MyLinkedList list = new MyLinkedList();
    private String command;
    private ArrayList al = new ArrayList();


    // add other fields as needed

    @Override
    public void processCommand(String command) throws EditorException {
        Scanner scan = new Scanner(System.in);

        if(command.equals("rev")){
            this.rev();
            return;
        }

        if(command.equals("pas")){
            this.paste();
            return;
        }


        char operation = command.charAt(0);
        int x;

        switch (operation)
        {
        case 'e':
            //adds the line to back of list
            String element = command.substring(2);
            list.add(element);
            break;
        case 'b':
            //finds the index before current
            x = this.getCurrentLineNbr();
            if(x < 0)
                x = 0;
            list.add(x, command.substring(2));
            break;

        case 'i':
            //finds the index after current
            x = this.getCurrentLineNbr() + 1;
            if(list.isEmpty()){
```

```
58              list.add(command.substring(2));
59              break;
60          }
61          list.add(x, command.substring(2));
62          break;
63
64      case 'm':
65          //moves the current line down
66          if(command.length() == 1){
67              if(!list.isEmpty()){
68                  x = list.getCurrentNbr() + 1;
69                  list.setCurrent(x);
70                  break;
71              }
72              else{
73                  throw new EditorException("list is empty.");
74              }
75          }
76
77          else if((this.getCurrentLineNbr() +
78                  Integer.parseInt(command.substring(2))) >=
79                  (list.size())
80                  || list.isEmpty()){
81              throw new EditorException("Move not possible.");
82          }
83
84
85          else{
86              x = list.getCurrentNbr() +
87                      Integer.parseInt(command.substring(2));
88              list.setCurrent(x);
89              break;
90          }
91
92      case 'u':
93          //moves the current line down
94          if(command.length() == 1) {
95              if(!list.isEmpty()){
96                  x = list.getCurrentNbr() - 1;
97                  list.setCurrent(x);
98                  break;
99              }
100             else{
101
102                 throw new EditorException("list is empty.");
103             }
104         }
105
106
107         else if((this.getCurrentLineNbr() -
108                 Integer.parseInt(command.substring(2)) < 0))
109
110             throw new EditorException("Move not possible.");
111
112         else{
```

```
113                 x = list.getCurrentNbr() -
114                         Integer.parseInt(command.substring(2));
115                 list.setCurrent(x);
116                 break;
117             }
118
119         case 'r':
120
121             if(this.nbrLines() == 0)
122                 //throws exception if the list is empty
123                 throw new EditorException("The list is empty.");
124             else{
125
126                 //list.remove(list.getCurrentNbr());
127                 this.removeLines(command);
128                 break;
129             }
130
131         case 'd':
132             //displays the list in a nice format
133             this.display(command);
134             break;
135
136         case 'c':
137             //clears the buffer of all items
138             if(command.length() > 1){
139
140                 String y = command.substring(command.length()-2);
141                 int a = Integer.parseInt(command.substring(4, 6)) - 1;
142                 int b = Integer.parseInt(y);
143                 this.cut(a, b);
144                 break;
145             }
146
147             System.out.print("You are about to clear the editor " +
148                     "are you sure you want to procceed: ");
149             String choice = scan.nextLine();
150
151             if(choice.equals("y")){
152                 this.clear();
153                 break;
154             }
155
156             else if(choice.equals("n"))
157                 break;
158
159             else{
160                 System.out.println("Invalid Choice.");
161                 break;
162             }
163
164         case 's':
165             try{
166                 this.save(command.substring(2));
167                 break;
```

```
168             }
169             catch(StringIndexOutOfBoundsException e){
170                 System.out.println("No File.");
171                 break;
172             }
173 
174 
175         case 'l':
176             //clears buffer then loads new items from file
177             if(command.length() == 1){
178                 System.out.println("No File.");
179                 break;
180             }
181 
182             else{
183                 System.out.print("Are you sure you want to load " +
184                         "this file: ");
185                 choice = scan.nextLine();
186                 this.load(choice, command.substring(2));
187                 break;
188             }
189 
190         case 'h':
191             //displays a description of commands
192             this.help();
193             break;
194 
195         case 'x':
196             //exits the program
197             System.out.print("Are you sure you want to exit: ");
198             choice = scan.nextLine();
199             this.exit(choice);
200             break;
201         default:
202             System.out.println("Error");
203         }
204 
205     }
206 
207     /**
208      * Gets the number lines.
209      *
210      * @return the number of lines.
211      */
212     @Override
213     public int nbrLines() {
214 
215         return list.size();
216     }
217 
218     /**
219      * Gets the specified node.
220      *
221      * @param lineNbr the index number of the node.
222      * @return the node by specified line index.
```

```
223      */
224     @Override
225     public String getLine(int lineNbr) throws EditorException {
226         if(lineNbr < 0 || lineNbr > list.size()){
227             throw new EditorException("Line number is invalid.");
228         }
229 
230         return list.get(lineNbr);
231     }
232 
233     /**
234      * Gets the contents current node.
235      *
236      * @return the contents current node.
237      */
238     @Override
239     public String getCurrentLine() {
240         // REPLACE THE LINE BELOW WITH YOUR CODE
241         return list.getCurrent().getData();
242     }
243 
244     /**
245      * Gets the index of the current node.
246      *
247      * @return index of current node.
248      */
249     @Override
250     public int getCurrentLineNbr() {
251         // REPLACE THE LINE BELOW WITH YOUR CODE
252         return list.getCurrentNbr();
253     }
254 
255     /**
256      * Displays the buffer in a formatted form.
257      *
258      */
259     public void display(String command){
260 
261         int i;
262         int x;
263 
264         //prints the entire list
265         if(command.length() == 1){
266             System.out.println("Start of file.");
267             i = 0;
268             //loops through and gets the info from the list
269             while(i <= (this.nbrLines()-1)){
270                 if(list.getNode(i) == list.getCurrent())
271                     System.out.println("*" + (i+1) + ": " +
272                             list.get(i).toString());
273                 else
274                     System.out.println(" " + (i+1) + ": " +
275                             list.get(i).toString());
276                 i++;
277             }
```

```
278             System.out.println("End of file.");
279         }
280
281         else{
282
283             //Gets the number lines to print
284             i = Integer.parseInt(command.substring(2, 4));
285             x = Integer.parseInt(command.substring(5));
286
287             System.out.println("Start of file.");
288
289             //loops to get the info from the list
290             while((i-1) <= (x-1)){
291
292                 System.out.println(" " + (i) + ": " +
293                         list.get(i-1).toString());
294                 i++;
295             }
296
297             System.out.println("End of file.");
298
299         }
300
301     }
302
303
304     /**
305      * Clears the buffer
306      *
307      */
308     public void clear(){
309         list.clear();
310     }
311
312
313     /**
314      * Saves the buffer to a file.
315      *
316      */
317     public void save(String fileName){
318
319         int i = 1;
320
321         try{
322
323             File linkInfo = new File(fileName);
324             //creates the file to save to
325             PrintWriter infoToWrite = new PrintWriter(
326                     new BufferedWriter(new FileWriter(linkInfo)));
327             //prints the list to the file
328             while(i <= this.nbrLines()){
329
330                 infoToWrite.println(list.get(i-1));
331                 i++;
332             }
```

```
333
334             infoToWrite.close();
335         }
336
337         catch(IOException e){
338
339             System.out.println("An I/O Error Occured.");
340             System.exit(0);
341         }
342
343     }
344
345     /**
346      * Clears the buffer and the loads a file into it.
347      *
348      */
349     public void load(String choice, String x){
350
351         if(choice.equals("y")){
352             this.clear();
353
354             File linkedInfo = new File(x);
355
356             try{
357                 //sets up the file to load from
358                 BufferedReader getInfo = new BufferedReader(
359                         new FileReader(linkedInfo));
360
361                 String info = getInfo.readLine();
362                 list.add(info);
363
364                 //reads and adds the lines to the list
365                 while(info != null){
366                     info = getInfo.readLine();
367
368                     if(info == null)
369                         break;
370
371                     list.add(info);
372                 }
373
374                 list.setCurrent(0);
375                 getInfo.close();
376
377             }
378
379             catch(FileNotFoundException e){
380                 System.out.println("Couldn't Find File.");
381                 System.exit(0);
382             }
383
384             catch(IOException e){
385                 System.out.print("An I/O Exception has occured.");
386                 System.exit(0);
387             }
```

```
388         }
389
390         else if(choice.equals("n"))
391             return;
392
393         else{
394             System.out.println("Invalid Choice.");
395             return;
396         }
397
398
399     }
400
401     /**
402      * Exits the program
403      *
404      */
405     public void exit(String choice){
406
407         //Asks if you are sure you want to quit
408         if(choice.equals("y")){
409             System.out.println("Bye!");
410             System.exit(0);
411         }
412
413         else if(choice.equals("n"))
414             return;
415
416         else{
417             System.out.println("Invalid Choice.");
418             return;
419         }
420
421
422     }
423
424     /**
425      * Displays a help section that describes the functions
426      * of the commands.
427      *
428      */
429     public void help(){
430         System.out.println("b <sentence>:  Insert sentence before " +
431                 "the current; makes the inserted line the " +
432                 "current line.");
433         System.out.println();
434         System.out.println("i <sentence>: Insert sentence after " +
435                 "the current; makes the inserted line the " +
436                 "current line.");
437         System.out.println();
438         System.out.println("m   Move current line " +
439                 "indicator down 1 position.");
440         System.out.println();
441         System.out.println("u   Move current line " +
442                 "indicator up 1 position.");
```

```
443         System.out.println();
444         System.out.println("r  Remove the current line; make " +
445                 "the next line the current line, if there " +
446                 "is a line; otherwise \n   make the previous " +
447                 "line the current line, if there is a previous " +
448                 "line. Throws EditorException if \n   remove is not" +
449                 " possible.");
450         System.out.println();
451         System.out.println("c   Removes all lines in the buffer. ");
452         System.out.println();
453         System.out.println("s <filename>   Save the contents " +
454                 "to the specified file.");
455         System.out.println();
456         System.out.println("l <filename> Load the contents of the " +
457                 "specified file into editor buffer replacing " +
458                 "the current contents;  \n   the first line becomes " +
459                 "the current line.");
460         System.out.println();
461         System.out.println("x   Exit the editor");
462         System.out.println();
463         System.out.println("e <sentence>   Insert line after the" +
464                 " last line; make the inserted line the current line.");
465         System.out.println();
466         System.out.println("m n  Move current line indicator " +
467                 "down n positions.");
468         System.out.println();
469         System.out.println("u n  Move the current line indicator " +
470                 "up n positions.");
471         System.out.println();
472         System.out.println("r n   Remove n lines at the " +
473                 "current line; makes the next line the current " +
474                 "line, \n      if there is a next line; otherwise " +
475                 "make the previous line the current line.");
476         System.out.println();
477         System.out.println("d n1 n2  Display lines from line " +
478                 "n1 to line n2.");
479         System.out.println();
480         System.out.println("rev   Reverses the lines in the buffer.");
481         System.out.println();
482         System.out.println("cut n1 n2  Cut the lines from the " +
483                 "file from line1 n1 to line n2 (inclusive), and" +
484                 " copy to \n    an internal clipboard.");
485         System.out.println();
486         System.out.println("pas  Paste the clipboard contents " +
487                 "before the current line position.\n     Make the " +
488                 "first line of the lines pasted the current line.");
489
490     }
491
492     /**
493      * Removes lines from the list.
494      *
495      */
496
497     public void removeLines(String command) throws EditorException {
```

```
498         //checks if I'm removing 1 line
499         if((command.length() == 1) ||
500                 (Integer.parseInt(command.substring(2)) == 1))
501             list.remove(list.getCurrentNbr());
502 
503         else{
504 
505             int x = Integer.parseInt(command.substring(2));
506             //checks to see if the remove goes out of bounds
507             if((list.getCurrentNbr() + x) > (list.size()))
508                 throw new EditorException("Removal Invalid");
509 
510 
511             for(int i = 0; i < x; i++){
512 
513                 list.remove(list.getCurrentNbr());
514 
515             }
516         }
517 
518 
519     }
520 
521     /**
522      * Reverses the order of the lines
523      *
524      */
525     public void rev(){
526         Stack st = new Stack();
527         int i = 0;
528         int x = 0;
529 
530         //loops through list and pops the strings into st
531         while(i <= (this.nbrLines()-1)){
532             st.push(list.get(i));
533             i++;
534         }
535 
536         //clears list
537         list.clear();
538 
539         //loops through st and adds the elements back to list
540         while(!st.empty()){
541             list.add((String) st.pop());
542         }
543 
544         return;
545     }
546 
547     /**
548      * Saves and removes the indicated elements
549      *
550      */
551     public void cut(int a, int b) throws EditorException{
552         if(list.size() < (b))
```

```
553             throw new EditorException("Invalid Cut");
554
555         //I'll save position a to x because I'll need to know
556         //where to remove elements from the list
557         int x = a;
558
559         while(a < b){
560
561             al.add(list.get(a));
562             a++;
563         }
564
565         list.remove(x);
566         x++;
567         while(x < b){
568
569             list.remove(list.getCurrentNbr());
570             x++;
571         }
572
573     }
574
575     /**
576      * Pastes the lines saved
577      *
578      */
579     public void paste(){
580         //pastes the lines saved in ascending order
581         //makes the first pasted the new current
582         for(int i = (al.size()-1); i >= 0; i--){
583             list.add(list.getCurrentNbr(), (String)al.get(i));
584         }
585     }
586
587 }
588
```

```java
package package1;
/**
 * The MyLinkedList class implements a simple linked list.
 *
 * @author YOUR NAME(S)
 *
 */
public class MyLinkedList implements ILinkedList {

    private Node front;
    private Node rear;
    private Node current;
    private int size;

    /**
     * Initializes a newly constructed MyLinkedList object.
     */
    public MyLinkedList() {
        front = null;
        rear = null;
        current = null;
        size = 0;
    }

    /**
     * Adds the specified element to the end of the list.
     *
     * @param element the element to add to the list
     *
     * @throws NullPointerException if the specified element is null
     */
    @Override
    public void add(String element)throws NullPointerException {
        // COMPLETE THIS METHOD
        if(element.equals(null)){
            throw new NullPointerException();
        }



        //if the list is empty assign the new element to first
        if(front == null){
            front = rear = new Node(element);
            current = rear;
            size++;

        }
        else{
            rear.setNext(new Node(element));
            rear = rear.getNext();
            current = rear;
            size++;
        }


```

```
56      }
57
58      /**
59       * Inserts the given element at the specified position in the list.
60       *
61       * @param index index at which the specified element is to be
62       * inserted.
63       * @param element element to be inserted
64       * @throws IndexOutOfBoundsException if the index is out of range
65       * (index < 0 || index > size()).
66       * @throws NullPointerException if the specified element is null
67       */
68      @Override
69      public void add(int index, String element) throws
70      IndexOutOfBoundsException, NullPointerException{
71
72          if(index < 0 || index > size()){
73              throw new IndexOutOfBoundsException();
74          }
75
76          if(element.equals(null)){
77              throw new NullPointerException();
78          }
79
80          Node x = front;
81          //creates a node with element
82          Node temp = new Node(element);
83          int i = 0;
84
85          //what to do if no list
86          if(isEmpty()){
87              front = rear = new Node(element);
88              front.setNext(null);
89              size++;
90              current = front;
91              return;
92          }
93
94
95          //what to do if index is the first element
96          if(index == 0){
97              temp.setNext(front);
98              front = temp;
99              current = front;
100             size++;
101             return;
102         }
103
104
105         //finds the node before the position you want to insert
106         while(i != (index-1)){
107             x = x.getNext();
108             i++;
109
110         }
```

```
111         if(x.getNext() == null)
112         {
113             rear.setNext(temp);
114             rear = rear.getNext();
115             current = rear;
116             size++;
117             return;
118         }
119         temp.setNext(x.getNext());
120         x.setNext(temp);
121         current = temp;
122         size++;
123 
124     }
125 
126     /**
127      * Removes the element at the specified position in this list.
128      *
129      * @param index the index of the element to be removed
130      *
131      * @return the element previously at the specified position
132      *
133      * @throws IndexOutOfBoundsException if the index is out of range
134      * (index < 0 || index >= size())
135      */
136     @Override
137     public String remove(int index) throws IndexOutOfBoundsException{
138 
139         //set a Node to the front
140         Node x = front;
141         //a is used to save the data of the deleted element
142         String a;
143         //i is an iterator
144         int i = 0;
145 
146         //throws an exception if index is out of bounds
147         if((index < 0 || index >= size()))
148             throw new IndexOutOfBoundsException();
149 
150         //Do this if removing the first
151         if(index == 0){
152             //save the deleted string
153             a = front.getData();
154             front = front.getNext();
155             current = front;
156             size--;
157             //return the string
158             return a;
159         }
160 
161 
162         //This is done if the element is in the middle or last of list
163         while(i != (index-1)){
164             x = x.getNext();
165             i++;
```

```
166         }
167
168         a = x.getNext().getData();
169         x.setNext(x.getNext().getNext());
170         current = x;
171         size--;
172
173         //Sets the new current line
174         if(x.getNext() != null)
175             current = x.getNext();
176         else
177             current = x;
178
179         if(a.equals(null))
180             return null;
181         else
182             return a;
183     }
184
185     /**
186      * Returns the element at the specified position in this list.
187      * Return null if the index is out of range
188      * (index < 0 || index >= size())
189      *
190      * @param index index of the element to return
191      *
192      * @return the element at the specified position in this list
193      *
194      * @throws IndexOutOfBoundsException if the index is out of range
195      * (index < 0 || index >= size())
196      */
197     @Override
198     public String get(int index) throws  IndexOutOfBoundsException{
199         if(index < 0 || index >= size())
200             throw new IndexOutOfBoundsException();
201         int i = 0;
202         Node x = front;
203         while(i != index){
204             x = x.getNext();
205             i++;
206
207         }
208
209         return x.getData();
210
211     }
212
213     /**
214      * Returns true if this list contains no elements.
215      *
216      * @return true if this list contains no elements and false
217      * otherwise
218      */
219     @Override
220     public boolean isEmpty() {
```

```
221         // ALREADY DONE FOR YOU :-)
222         return size == 0;
223     }
224 
225     /**
226      * Returns the number of elements in this list.
227      *
228      * @return the number of elements in this list
229      */
230     @Override
231     public int size() {
232         // ALREADY DONE FOR YOU :-)
233         return size;
234     }
235 
236     /**
237      * Removes all of the elements from this list. The list will be
238      * empty after this call returns.
239      */
240     @Override
241     public void clear() {
242         // ALREADY DONE FOR YOU :-)
243         front = null;
244         rear = null;
245         current = null;
246         size = 0;
247     }
248 
249     /**
250      * Sets the current to a new index.
251      *
252      * @param i is the new of Current
253      *
254      */
255 
256     public void setCurrent(int i){
257 
258         if(i == 0){
259             current = front;
260             return;
261         }
262 
263         //creates a temporary node
264         Node temp = front;
265         int x = 0;
266         //loops to where i is
267         while(!(x == i))
268         {
269             temp = temp.getNext();
270             x++;
271         }
272         //sets current to temp
273         current = temp;
274     }
275 
```

```
276
277     /**
278      * Gets the current node.
279      *
280      * @return the current node.
281      */
282     public Node getCurrent(){
283         return current;
284     }
285
286
287     /**
288      * Gets the index of the current node.
289      *
290      * @return the current node index.
291      */
292     public int getCurrentNbr(){
293         Node temp = front;
294         int i = 0;
295         while(current != temp){
296             temp = temp.getNext();
297             i++;
298         }
299         return i;
300     }
301
302     /**
303      * Gets the node from the index passed in.
304      *
305      * @param the index of the node you want to get.
306      * @return the node.
307      */
308
309     public Node getNode(int index) throws IndexOutOfBoundsException{
310         if(index < 0 || index >= size())
311             throw new IndexOutOfBoundsException();
312         int i = 0;
313         Node x = front;
314         while(i != index){
315             x = x.getNext();
316             i++;
317
318         }
319
320         return x;
321
322     }
323
324
325 }
326
```

```java
package package1;
import static org.junit.Assert.*;

/**
 * EditorTest class contains unit tests for Editor class.
 *
 * @author YOUR NAME(S)
 *
 * ADD MORE UNIT TESTS TO THIS CLASS
 *
 */
public class EditorTest {

	@Test
	public void testInsertCommand() throws EditorException {
		Editor ed = new Editor();
		ed.processCommand("i Java is an OO language.");
		ed.processCommand("i Ruby is another OO language.");
		assertEquals("Ruby is another OO " +
				"language.", ed.getCurrentLine());
		assertEquals("Java is an OO language.", ed.getLine(0));
	}

	@Test
	public void testBCommand() throws EditorException {
		Editor ed = new Editor();
		ed.processCommand("i Java is an OO language.");
		ed.processCommand("i Ruby is another OO language.");
		ed.processCommand("b Hello World");
		assertEquals("Hello World", ed.getLine(1));
		assertEquals("Hello World", ed.getCurrentLine());

	}

	@Test
	public void testECommand() throws EditorException {
		Editor ed = new Editor();
		ed.processCommand("i Line1.");
		ed.processCommand("i Line2.");
		ed.processCommand("i Line3.");
		ed.processCommand("i Line4.");
		ed.processCommand("i Line5.");
		ed.processCommand("e Line6.");
		assertEquals("Line6.", ed.getLine(5));
		assertEquals("Line6.", ed.getCurrentLine());
		ed.processCommand("e Line7.");
		assertEquals("Line7.", ed.getLine(6));
		assertEquals("Line7.", ed.getCurrentLine());
	}
```

```
52
53          @
54          Test
55          public void testUNCommand() throws EditorException {
56                  Editor ed = new Editor();
57                  ed.processCommand("i Line1.");
58                  ed.processCommand("i Line2.");
59                  ed.processCommand("i Line3.");
60                  ed.processCommand("i Line4.");
61                  ed.processCommand("i Line5.");
62                  ed.processCommand("i Line6.");
63                  ed.processCommand("i Line7.");
64                  ed.processCommand("i Line8.");
65                  ed.processCommand("i Line9.");
66                  ed.processCommand("i Line10.");
67                  ed.processCommand("u 5");
68                  assertEquals("Line5.", ed.getCurrentLine());
69          }
70
71          @
72          Test
73          public void testUMCommand() throws EditorException {
74                  Editor ed = new Editor();
75                  ed.processCommand("b Line1.");
76                  ed.processCommand("b Line2.");
77                  ed.processCommand("b Line3.");
78                  ed.processCommand("b Line4.");
79                  ed.processCommand("b Line5.");
80                  ed.processCommand("b Line6.");
81                  ed.processCommand("b Line7.");
82                  ed.processCommand("b Line8.");
83                  ed.processCommand("b Line9.");
84                  ed.processCommand("b Line10.");
85                  ed.processCommand("m 5");
86                  assertEquals("Line5.", ed.getCurrentLine());
87          }
88
89
90          @Test(expected=EditorException.class)
91          public void testUNCommandError() throws EditorException {
92                  Editor ed = new Editor();
93                  ed.processCommand("i Line1.");
94                  ed.processCommand("i Line2.");
95                  ed.processCommand("i Line3.");
96                  ed.processCommand("i Line4.");
97                  ed.processCommand("i Line5.");
98                  ed.processCommand("i Line6.");
99                  ed.processCommand("i Line7.");
100                 ed.processCommand("i Line8.");
```

```
101             ed.processCommand("i Line9.");
102             ed.processCommand("i Line10.");
103             ed.processCommand("u 12");
104         }
105
106         @Test(expected=EditorException.class)
107         public void testUMCommandError() throws EditorException {
108             Editor ed = new Editor();
109             ed.processCommand("b Line1.");
110             ed.processCommand("b Line2.");
111             ed.processCommand("b Line3.");
112             ed.processCommand("m 3");
113         }
114
115
116         @Test
117         public void testbInsertCommand() throws EditorException {
118             Editor ed = new Editor();
119             ed.processCommand("b Java is an OO language.");
120             ed.processCommand("b All Java applications have a " +
121                               "main method.");
122             ed.processCommand("b Ruby is another OO language.");
123             assertEquals("Ruby is another OO language.", ed.getLine(0));
124             assertEquals("Ruby is another OO " +
125                               "language.", ed.getCurrentLine());
126         }
127
128         @Test
129         public void testUCommand() throws EditorException {
130             Editor ed  = new Editor();
131             ed.processCommand("i Line1");
132             ed.processCommand("i Line2");
133             ed.processCommand("i Line3");
134             ed.processCommand("i Line4");
135             ed.processCommand("u");
136             ed.processCommand("u");
137             assertEquals("Line2", ed.getCurrentLine());
138
139         }
140
141         @Test
142         public void testMCommand() throws EditorException {
143             Editor ed  = new Editor();
144             ed.processCommand("i Line1");
145             ed.processCommand("i Line2");
146             ed.processCommand("i Line3");
147             ed.processCommand("i Line4");
148             ed.processCommand("u");
149             ed.processCommand("u");
```

```
150                 assertEquals("Line2", ed.getCurrentLine());
151                 ed.processCommand("m");
152                 ed.processCommand("m");
153                 assertEquals("Line4", ed.getCurrentLine());
154
155         }
156
157         @Test
158         public void testRCommand() throws EditorException {
159                 Editor ed  = new Editor();
160                 ed.processCommand("i Line1");
161                 ed.processCommand("i Line2");
162                 ed.processCommand("i Line3");
163                 ed.processCommand("i Line4");
164                 ed.processCommand("u");
165                 ed.processCommand("u");
166                 ed.processCommand("r");
167                 assertEquals("Line3", ed.getCurrentLine());
168                 assertEquals(3, ed.nbrLines());
169         }
170
171         @Test
172         public void testClearCommand()throws EditorException {
173                 Editor ed = new Editor();
174                 ed.processCommand("i Java is cool.");
175                 ed.processCommand("i Python is cooler.");
176                 ed.processCommand("i Ruby is hot.");
177                 ed.processCommand("i COBOL is old.");
178                 ed.processCommand("c");
179                 assertEquals(0, ed.nbrLines());
180
181         }
182
183         @Test
184         public void testRemoveCommand()throws EditorException {
185                 Editor ed = new Editor();
186                 ed.processCommand("i Java is cool.");
187                 ed.processCommand("i Python is cooler.");
188                 ed.processCommand("i Ruby is hot.");
189                 ed.processCommand("i COBOL is old.");
190                 assertEquals(4, ed.nbrLines());
191                 ed.processCommand("u 2");
192                 ed.processCommand("r 2");
193                 assertEquals(2, ed.nbrLines());
194                 assertEquals("COBOL is old.", ed.getCurrentLine());
195         }
196
197         @Test(expected=EditorException.class)
198         public void testRemoveError()throws EditorException {
```

```
199             Editor ed = new Editor();
200             ed.processCommand("i Java is cool.");
201             ed.processCommand("i Python is cooler.");
202             ed.processCommand("i Ruby is hot.");
203             ed.processCommand("i COBOL is old.");
204             assertEquals(4, ed.nbrLines());
205             ed.processCommand("u 2");
206             ed.processCommand("r 2");
207             assertEquals(2, ed.nbrLines());
208             assertEquals("COBOL is old.", ed.getCurrentLine());
209             ed.processCommand("r 2");//should cause an error
210     }
211
212     @Test
213     public void testReverseCommand()throws EditorException {
214             Editor ed = new Editor();
215             ed.processCommand("i Java is cool.");
216             ed.processCommand("i Python is cooler.");
217             ed.processCommand("i Ruby is hot.");
218             ed.processCommand("i COBOL is old.");
219             assertEquals(4, ed.nbrLines());
220             ed.processCommand("rev");
221             assertEquals(4, ed.nbrLines());
222             assertEquals("COBOL is old.", ed.getLine(0));
223             assertEquals("Ruby is hot.", ed.getLine(1));
224             assertEquals("Python is cooler.", ed.getLine(2));
225             assertEquals("Java is cool.", ed.getLine(3));
226     }
227
228
229     @Test
230     public void testRevCommand()throws EditorException {
231             Editor ed = new Editor();
232             ed.processCommand("i Line1");
233             ed.processCommand("i Line2");
234             ed.processCommand("i Line3");
235             ed.processCommand("i Line4");
236             ed.processCommand("i Line5");
237             ed.processCommand("i Line6");
238             ed.processCommand("i Line7");
239             assertEquals(7, ed.nbrLines());
240             assertEquals(6, ed.getCurrentLineNbr());
241             ed.processCommand("rev");
242             assertEquals(7, ed.nbrLines());
243             assertEquals("Line7", ed.getLine(0));
244             assertEquals("Line6", ed.getLine(1));
245             assertEquals(6, ed.getCurrentLineNbr());
246
247     }
```

```
248
249         @Test(expected=EditorException.class)
250         public void testRemoveException()throws EditorException {
251                 Editor ed = new Editor();
252                 ed.processCommand("i Java is cool.");
253                 ed.processCommand("i Python is cooler.");
254                 ed.processCommand("i Ruby is hot.");
255                 assertEquals(3, ed.nbrLines());
256                 ed.processCommand("u 1");
257                 ed.processCommand("r 3"); // this call should throw
    EditorException
258         }
259
260         @Test
261         public void testCut()throws EditorException {
262                 Editor ed = new Editor();
263                 ed.processCommand("i line1.");
264                 ed.processCommand("i line2.");
265                 ed.processCommand("i line3.");
266                 ed.processCommand("i line4.");
267                 ed.processCommand("i line5.");
268                 ed.processCommand("i line6.");
269                 ed.processCommand("i line7.");
270                 ed.processCommand("i line8.");
271                 ed.processCommand("cut 02 08");
272                 assertEquals(1, ed.nbrLines());
273                 assertEquals("line1.", ed.getCurrentLine());
274         }
275
276         @Test
277         public void testPaste()throws EditorException {
278                 Editor ed = new Editor();
279                 ed.processCommand("i line1.");
280                 ed.processCommand("i line2.");
281                 ed.processCommand("i line3.");
282                 ed.processCommand("i line4.");
283                 ed.processCommand("i line5.");
284                 ed.processCommand("i line6.");
285                 ed.processCommand("i line7.");
286                 ed.processCommand("i line8.");
287                 ed.processCommand("cut 02 08");
288                 ed.processCommand("pas");
289                 assertEquals("line2.", ed.getCurrentLine());
290         }
291
292
293         @Test(expected=EditorException.class)
294         public void testCutException()throws EditorException {
295                 Editor ed = new Editor();
```

```
296                 ed.processCommand("i line1.");
297                 ed.processCommand("i line2.");
298                 ed.processCommand("i line3.");
299                 ed.processCommand("i line4.");
300                 ed.processCommand("i line5.");
301                 ed.processCommand("i line6.");
302                 ed.processCommand("i line7.");
303                 ed.processCommand("i line8.");
304                 ed.processCommand("cut 02 09");
305
306         }
307
308 }
```