```
 1: // $Id: jroff.java,v 1.3 2014-01-09 17:06:42-08 - - $
 2:
 3: import java.io.*;
 4: import java.util.Scanner;
 5: import static java.lang.System.*;
 6:
 7: class jroff{
 8:    static final String STDIN_NAME = "-";
 9:
10:    static void scanfile (String filename, Scanner infile) {
11:       out.printf ("STUB: filename = %s%n", filename);
12:       for (int linenr = 1; infile.hasNextLine(); ++linenr) {
13:          String line = infile.nextLine();
14:          out.printf ("STUB: %s: %4d: [%s]%n", filename, linenr, line);
15:          String[] words = line.split ("\\s+");
16:          if (words.length > 0 && words[0].startsWith (".")) {
17:             try {
18:                commands.do_command (words);
19:             }catch (IllegalArgumentException error) {
20:                auxlib.warn (filename, linenr, words[0],
21:                             "invalid command");
22:             }
23:          }else {
24:             for (String word: words) out.printf ("[%s]", word);
25:             out.printf ("%n");
26:          }
27:       }
28:    }
29:
30:    public static void main (String[] args) {
31:       linkedqueue <String> words = new linkedqueue <String> ();
32:       if (args.length == 0) {
33:          scanfile (STDIN_NAME, new Scanner (in));
34:       }else {
35:          for (String filename : args) {
36:             if (filename.equals (STDIN_NAME)) {
37:                scanfile (STDIN_NAME, new Scanner (in));
38:             }else {
39:                try {
40:                   Scanner scan = new Scanner (new File (filename));
41:                   scanfile (filename, scan);
42:                   scan.close();
43:                }catch (IOException error) {
44:                   auxlib.warn (error.getMessage());
45:                }
46:             }
47:          }
48:       }
49:    }
50:
51: }
```

```
 1: // $Id: commands.java,v 1.7 2014-01-17 17:57:56-08 - - $
 2:
 3: import java.util.HashMap;
 4: import static java.lang.System.*;
 5:
 6: class commands {
 7:
 8:    private static void command_00 (String[] words) {
 9:       // Executing a comment does nothing.
10:    }
11:
12:    private static void command_bp (String[] words) {
13:       STUB (words);
14:    }
15:
16:    private static void command_br (String[] words) {
17:       STUB (words);
18:    }
19:
20:    private static void command_cc (String[] words) {
21:       STUB (words);
22:    }
23:
24:    private static void command_in (String[] words) {
25:       STUB (words);
26:    }
27:
28:    private static void command_ll (String[] words) {
29:       STUB (words);
30:    }
31:
32:    private static void command_mt (String[] words) {
33:       STUB (words);
34:    }
35:
36:    private static void command_pl (String[] words) {
37:       STUB (words);
38:    }
39:
40:    private static void command_po (String[] words) {
41:       STUB (words);
42:    }
43:
44:    private static void command_sp (String[] words) {
45:       STUB (words);
46:    }
```

```
47:
48:    private static void STUB (String[] words) {
49:       out.printf ("%s: STUB: %s",
50:                   auxlib.PROGNAME, auxlib.caller());
51:       for (String word: words) out.printf (" %s", word);
52:       out.printf ("%n");
53:    }
54:
55:    public static void do_command (String[] words) {
56:       switch (words[0]) {
57:          case ".\\\"": command_00 (words); break;
58:          case ".bp"  : command_bp (words); break;
59:          case ".br"  : command_br (words); break;
60:          case ".cc"  : command_cc (words); break;
61:          case ".in"  : command_in (words); break;
62:          case ".ll"  : command_ll (words); break;
63:          case ".mt"  : command_mt (words); break;
64:          case ".pl"  : command_pl (words); break;
65:          case ".po"  : command_po (words); break;
66:          case ".sp"  : command_sp (words); break;
67:          default     : throw new IllegalArgumentException (words[0]);
68:       }
69:    }
70:
71: }
72:
```

```
 1: // $Id: linkedqueue.java,v 1.1 2011-01-20 21:05:43-08 - - $
 2:
 3: import java.util.NoSuchElementException;
 4:
 5: class linkedqueue <item_t> {
 6:
 7:    private class node{
 8:       item_t item;
 9:       node link;
10:    }
11:
12:    //
13:    // INVARIANT:  front == null && rear == null
14:    //          || front != null && rear != null
15:    // In the latter case, following links from the node pointed
16:    // at by front will lead to the node pointed at by rear.
17:    //
18:    private node front = null;
19:    private node rear = null;
20:
21:    public boolean empty (){
22:       return front == null;
23:    }
24:
25:    public void insert (item_t any) {
26:       // STUB: Add code here to insert an item_t into the queue.
27:    }
28:
29:    public item_t remove (){
30:       if (empty ()) throw new NoSuchElementException ();
31:       // STUB: Add code for remove here.
32:       return null; // STUB: Delete this return statement.
33:    }
34:
35: }
```

```
 1: // $Id: auxlib.java,v 1.4 2014-01-17 17:43:33-08 - - $
 2: //
 3: // NAME
 4: //    auxlib - Auxiliary miscellanea for handling system interaction.
 5: //
 6: // DESCRIPTION
 7: //    Auxlib has system access functions that can be used by other
 8: //    classes to print appropriate messages and keep track of
 9: //    the program name and exit codes.  It assumes it is being run
10: //    from a jar and gets the name of the program from the classpath.
11: //    Can not be instantiated.
12: //
13:
14: import static java.lang.System.*;
15: import static java.lang.Integer.*;
16:
17: public final class auxlib{
18:    public static final String PROGNAME =
19:                   basename (getProperty ("java.class.path"));
20:    public static final int EXIT_SUCCESS = 0;
21:    public static final int EXIT_FAILURE = 1;
22:    public static int exitvalue = EXIT_SUCCESS;
23:
24:    //
25:    // private ctor - prevents class from new instantiation.
26:    //
27:    private auxlib () {
28:       throw new UnsupportedOperationException ();
29:    }
30:
31:    //
32:    // basename - strips the dirname and returns only the basename.
33:    //            See:  man -s 3c basename
34:    //
35:    public static String basename (String pathname) {
36:       if (pathname == null || pathname.length () == 0) return ".";
37:       String[] paths = pathname.split ("/");
38:       for (int index = paths.length - 1; index >= 0; --index) {
39:          if (paths[index].length () > 0) return paths[index];
40:       }
41:       return "/";
42:    }
```

```
43:
44:    //
45:    // Functions:
46:    //    whine   - prints a message with a given exit code.
47:    //    warn    - prints a stderr message and sets the exit code.
48:    //    die     - calls warn then exits.
49:    // Combinations of arguments:
50:    //    objname - name of the object to be printed (optional)
51:    //    message - message to be printed after the objname,
52:    //              either a Throwable or a String.
53:    //
54:    public static void whine (int exitval, Object... args) {
55:       exitvalue = exitval;
56:       err.printf ("%s", PROGNAME);
57:       for (Object argi : args) err.printf (": %s", argi);
58:       err.printf ("%n");
59:    }
60:    public static void warn (Object... args) {
61:       whine (EXIT_FAILURE, args);
62:    }
63:    public static void die (Object... args) {
64:       warn (args);
65:       exit ();
66:    }
67:
68:    //
69:    // usage_exit - prints a usage message and exits.
70:    //
71:    public static void usage_exit (String optsargs) {
72:       exitvalue = EXIT_FAILURE;
73:       err.printf ("Usage: %s %s%n", PROGNAME, optsargs);
74:       exit ();
75:    }
76:
77:    //
78:    // exit - calls exit with the appropriate code.
79:    //        This function should be called instead of returning
80:    //        from the main function.
81:    //
82:    public static void exit () {
83:       System.exit (exitvalue);
84:    }
```

```
 85:
 86:    //
 87:    // identity - returns the default Object.toString value
 88:    //            Useful for debugging.
 89:    //
 90:    public static String identity (Object object) {
 91:       return object == null ? "(null)"
 92:            : object.getClass().getName() + "@"
 93:            + toHexString (identityHashCode (object));
 94:    }
 95:
 96:    //
 97:    // caller - return information about the caller of the
 98:    //          function that called this function in the form
 99:    //          filename[linenumber] functionname
100:    //
101:    public static String caller() {
102:       StackTraceElement caller
103:            = Thread.currentThread().getStackTrace()[2];
104:       return String.format ("%s[%d] %s", caller.getFileName(),
105:              caller.getLineNumber(), caller.getMethodName());
106:    }
107:
108: }
```

```
 1: # $Id: Makefile,v 1.6 2014-01-17 17:40:59-08 - - $
 2:
 3: JAVASRC    = jroff.java commands.java linkedqueue.java auxlib.java
 4: SOURCES    = ${JAVASRC} Makefile README
 5: MAINCLASS  = jroff
 6: CLASSES    = jroff.class commands.class linkedqueue.class auxlib.class
 7: JARCLASSES = ${CLASSES} linkedqueue\$$node.class
 8: JARFILE    = jroff
 9: LISTING    = Listing.ps
10: SUBMITDIR  = cmps012b-wm.w14 asg2
11:
12: all : ${JARFILE}
13:
14: ${JARFILE} : ${CLASSES} Makefile
15:         echo Main-class: ${MAINCLASS} >Manifest
16:         jar cvfm ${JARFILE} Manifest ${JARCLASSES}
17:         - rm Manifest
18:         chmod +x ${JARFILE}
19:
20: %.class : %.java
21:         - checksource $<
22:         - cid + $<
23:         javac $<
24:
25: clean :
26:         - rm ${JARCLASSES} Manifest
27:
28: spotless : clean
29:         - rm ${JARFILE}
30:
31: ci : ${SOURCES}
32:         - checksource ${SOURCES}
33:         cid + ${SOURCES}
34:
35: lis : ${SOURCES}
36:         mkpspdf ${LISTING} ${SOURCES}
37:
38: submit : ${SOURCES}
39:         submit ${SUBMITDIR} ${SOURCES}
40:         testsubmit ${SUBMITDIR} ${SOURCES}
41:
42: again:
43:         gmake --no-print-directory spotless ci all lis
44:
```

```
1: $Id: README,v 1.1 2011-01-20 21:05:43-08 - - $
```