

```

1  package Assign_2;
2
3  import BasicIO.*;
4
5  /** This class creates a BasicForm-driven application that keeps print jobs in a
    linked structure.
6
7     * @author S. Fenwick
8     *
9     * student # 6005011
10
11    * @version 1.0 (Feb. 2017)
12
13    public class Main{
14
15        private Node que; //list of print jobs
16        private ASCIIIDisplayer display;
17        public BasicForm form;
18        private int jobsLeft;
19
20        /* This constructor creates the BasicForm form and ASCIIIDisplayer display, and
        waits for user interaction. */
21
22        public Main(){
23
24            int button; //button pressed
25
26            display = new ASCIIIDisplayer();
27            form = new BasicForm("Add Job" , "Print Next" , "Check" , "Quit");
28            que = null;
29
30            setUpForm();
31
32            for ( ; ; ) {
33                form.clearAll();
34                button = form.accept();
35                if ( button == 3 ) break; // Quit
36                switch ( button ) {
37                    case 0: { // AddJob
38                        addJob(jobsLeft);
39                        break;
40                    }
41                    case 1: { // PrintNext
42                        printNext(jobsLeft);
43                        break;
44                    }
45                    case 2: { // Check
46                        check();
47                        break;
48                    }
49                };
50                form.accept("OK");
51            };
52
53            form.close();
54            display.close();
55
56        } //constructor
57
58        /* This method checks the priority of the current job being added, and if it is a
        priority 0 sends the job through
59        * the addStudent() method (sorted insertion), if it is priority 1 sends the job
        through the addStaff()
60        * method (sorted insertion), and if it is priority 2 it is sent through the
        addFaculty() method
61        * (insertion at end of list). It updates the counter whenever a job is added.
62        *
63        * @param counter keeps count of the jobs being added or removed from the spooler
64        */

```

```

65 private void addJob(int counter){
66
67     int priority; //student, staff, faculty
68     Job aJob;     //job being added
69
70     priority = form.readInt("priority");
71     aJob = new Job(form);
72
73     if(priority == 2){
74         addFaculty(aJob);
75     }
76     if(priority == 1){
77         addStaff(aJob);
78     }
79     if(priority == 0){
80         addStudent(aJob);
81     }
82
83     jobsLeft = counter + 1;
84
85 } //addJob
86
87 /* This method adds a job to the back of the list. Since it is a Faculty job
88 (priority 2) and nothing has a larger
89 * priority it does not require a sorted search, and can just be added to the
90 very back of the entire list.
91 *
92 * @param aJob the current job (node) being added to the spooler (list). */
93
94 private void addFaculty(Job aJob){
95
96     Node p;
97     Node q;
98
99     q = null;
100    p = que;
101
102    while ( p != null ) {
103        q = p;
104        p = p.next;
105    };
106    if ( q == null ) {
107        que = new Node(aJob,null);
108    }
109    else {
110        q.next = new Node(aJob,null);
111    };
112
113    form.writeString("status", "Job Added.");
114
115 } //addFaculty
116
117 /* This method adds a node of priority 1 (staff) to the list. It sorts through
118 the list checking the priority of
119 * the other nodes in the list and adds it to the back of the "staff section", i.
120 e it becomes the first node before
121 * any node with a priority of 2.
122 *
123 * @param aJob the current job (node) being added to the spooler (list). */
124
125 private void addStaff(Job aJob){
126
127     Node p;
128     Node q;
129
130     q = null;
131     p = que;
132
133     while ( p != null && p.item.getPriority() <= aJob.getPriority()){
134         q = p;

```

```

131     p = p.next;
132 };
133
134     if(q == null){
135         que = new Node(aJob, p);
136     }
137     else{
138         q.next = new Node(aJob, p);
139     }
140 }
141
142     form.writeString("status", "Job Added.");
143
144 } //addStaff
145
146 /* This method adds a node of priority 0 (student) to the list. It sorts through
147 the list checking the priority of
148 * the other nodes in the list and adds it to the back of the "student section",
149 i.e it becomes the first node before
150 * any node with a priority of 1.
151 *
152 * @param aJob the current job (node) being added to the spooler (list). */
153
154 private void addStudent(Job aJob){
155     Node p;
156     Node q;
157
158     q = null;
159     p = que;
160
161     while ( p != null && p.item.getPriority() <= aJob.getPriority()){
162         q = p;
163         p = p.next;
164     };
165
166     if(q == null){
167         que = new Node(aJob, p);
168     }
169     else{
170         q.next = new Node(aJob, p);
171     }
172
173     form.writeString("status", "Job Added.");
174
175 } //addStudent
176
177
178 /* This method deletes the first node of the list.
179 *
180 * @param counter keeps count of the jobs being added or removed from the spooler
181 */
182
183 private void printNext(int counter){
184     Job item;
185
186     if(que == null){
187         form.writeString("status", "There are no print jobs remaining.");
188         jobsLeft = 0;
189     }
190     else{
191         item = que.item;
192         que = que.next;
193         jobsLeft = counter - 1;
194         form.writeString("status", "Job Printed. " + jobsLeft + " Job(s) Remain.");
195         writeToDisplay(item);
196     }
197 }

```

```

198
199 } //printNext
200
201 /* This method writes to the ASCIIIDisplayer the information of the particular job
being printed.
202  *
203  * @param aJob the current job (node) being printed (deleted). */
204
205 private void writeToDisplay(Job aJob){
206
207     String desc;
208     String name;
209     int pages;
210
211     desc = aJob.getDescription();
212     name = aJob.getName();
213     pages = aJob.getPages();
214
215     display.writeString("(" + name + " prints " + desc + ": " + pages + " pages.)"
);
216     display.newLine();
217
218 } //writeToDisplay
219
220 /* This method checks how many pages are remaining in the spooler (que). */
221
222 private void check(){
223
224     Node p;
225     int pageCount = 0;
226
227     p = que;
228
229     while( p != null){
230         pageCount = pageCount + p.item.getPages();
231         p = p.next;
232     }
233
234     form.writeString("status", "There are " + pageCount + " pages in the que.");
235
236 } //check
237
238 /* This method sets up the BasicForm (form). */
239
240 private void setUpForm(){
241
242     form.setTitle("Print Spooler");
243
244     form.addTextField("description","Description:", 15, 10, 10);
245
246     form.addTextField("send","Sender:", 10, 10, 40);
247
248     form.addTextField("pages","# Pages:", 6, 10, 70);
249
250     form.addRadioButtons("priority","Priority",true, 248, 8, Job.PRIORITY);
251
252     form.addTextField("status","Status:", 39, 10, 100);
253     form.setEditable("status", false);
254
255 } //setUpForm
256
257 public static void main (String args[]) {Main m = new Main();}
258
259 } //Main

```