```
printf \\n\\n\\n            #print three blank lines



cat p3_Bonus.sh               #display the shell script file for the program
#!/bin/bash

set -v                        #turn on echo
printf \\n\\n\\n            #print three blank lines
cat p3_Bonus.sh               #display the shell script file for the program
printf \\f                    #issue a form feed (top of a new page)
cat -b p3_Bonus.java           #display the source file with line numbers
:                              #null command
:
:
javac p3_Bonus.java           #compile the java file
java p3_Bonus             #execute the file from the current directory
:
:
:
date                          #print the date
printf \\f                    #issue a form feed (top of a new page)

cat -b p3_Bonus.java           #display the source file with line numbers
     1    /*
     2    PROGRAM NAME: Lab 3 Bonus, Read Information from File and add to circular linked list
     3    PROGRAMMER:   Samuel Jentsch
     4    CLASS:        CSC 241.001, Fall 2013
     5    INSTRUCTOR:   Dr. D. Dunn
     6    DATE STARTED: October 22, 2013
     7    DUE DATE:     October 23, 2013
     8    REFERENCES:
     9                Dr. Dunn: assignment information sheet

    10    PROGRAM PURPOSE:
    11    a. This program reads a Job and attributes associated with the job from a file.
    12    b. The program parses commands, calling methods accordingly to add jobs
    13         to a circular linked list and manipulate the list as necessary.

    14    VARIABLE DICTIONARY:
    15       list - Node, pointer to the circular linked list.
    16       currentPointer- Node, pointer to the current job (node).
    17
    18    ADTs:
    19       none

    20    FILES USED:
    21       p3.dat - a file containing jobs and commands.
    22    ---------------------------------------------------------------------
    23    */
    24    import java.io.*;
    25    import java.util.*;

    26    public class p3_Bonus {
    27         static Node list;
    28         static Node currentPointer;
    29
    30         public static void main(String[] args) {
    31              File f = new File("../instr/p3.dat");
    32
    33              handleInputForFile(f);
    34
    35              System.out.println("Exiting...");
    36              printList();
```

```
37          }
38
39          public static void handleInputForFile(File file) {
40               //-------------------------------------------------
41               //Handle the input for the file object passed
42               //as a parameter. Continue reading input until
43               //true (quit) is returned by parseInput. Pass
44               //lines to parseInput to be interpreted and
45               //have the appropriate method executed.
46               //Precondition: File object referencing a file
47               //containing jobs and commands.
48               //Postcondition: Commands are executed line by
49               //line using parseInput until the end of the file
50               //or the Q command is reached.
51               //-------------------------------------------------

52               try {
53                    Scanner fileReader = new Scanner(file);
54                    boolean quit = false;
55                    while (fileReader.hasNext() && !quit) {
56                         quit = parseInput(fileReader.nextLine());
57                    }
58
59               } catch (FileNotFoundException e) {
60                    // TODO Auto-generated catch block
61                    e.printStackTrace();
62               }
63
64          }
65
66          public static boolean parseInput(String input) {
67               //-------------------------------------------------
68               //Handle a string passed to the method. Call the
69               //appropriate command or add a job to the list
70               //based on the string passed.
71               //Precondition: String object containing a command
72               //or job attributes.
73               //Postcondition: appropriate command executed or
74               //job added to the list.
75               //-------------------------------------------------
76
77               String[] commands = input.split(" ");
78
79               char firstChar = commands[0].toCharArray()[0];
80               if(!Character.isDigit(firstChar)) {
81                    //If the character at the beginning of the first command string
82                    //is not a digit, look to see if it signifies a command. Otherwise,
83                    //Check if it can be parsed and added as a job to the list.
84                    switch(firstChar) {
85                    case 'T':
86                         tCommand();
87                         break;
88                    case 'I':
89                         iCommand(commands);
90                         break;
91                    case 'D':
92                         dCommand();
93                         break;
94                    case 'Q':
95                         return true;
96                    default:
97                         System.out.println("Error. Unsupported Command: " + firstChar);
98                    }
99               } else {
```

```
100                  //Check if string is correct job format. If yes,
101                  //add to the linked list.
102                  if(commands.length == 3) {
103                       //String has correct number of parts.
104                       Node newNode = verifyAndReturnNodeWithAttributes(commands);
105                       addJob(newNode);
106                       if(currentPointer == null)
107                            currentPointer = list.next;
108                  } else {
109                       System.out.println("Invalid format to add new job. Jobs should"
110                                 + " be in format: JobID TIME NAME");
111                  }
112             }
113
114             return false;
115        }
116
117        public static Node verifyAndReturnNodeWithAttributes(String[] attributes) {
118             //-----------------------------------------------
119             //Verify that the attributes array passed to the
120             //method meets the conditions for initializing
121             //a node object. If it does, create and return
122             //the node. If not, print an error and return
123             //null.
124             //Precondition: String array attributes passed
125             //as parameter containing potential attributes
126             //for a new Node object.
127             //Postcondition: If the attributes meet the
128             //conditions, a new Node is created and returned.
129             //-----------------------------------------------
130             Node newNode = null;
131             int jobID;
132             int time;
133             String name = attributes[2];

134             try {
135                  jobID = Integer.parseInt(attributes[0]);
136                  time = Integer.parseInt(attributes[1]);
137             } catch(Exception ex) {
138                  System.out.println("Invalid Job ID or Time.");
139                  return null;
140             }
141
142             newNode = new Node(name, jobID, time, null);
143
144             return newNode;
145        }
146
147        public static void tCommand() {
148             //-----------------------------------------------
149             //If the list isn't empty, decrement the time
150             //of the Node currently referenced by
151             //currentPointer. If the time reaches 0, remove the
152             //node from the list. Print the amount of time left
153             //in the node and the list.
154             //Precondition: List and currentPointer class
155             //variables.
156             //Postcondition: time attribute of the node
157             //referenced by the currentPointer if the time is
158             //0 after decrement, node is removed from list.
159             //-----------------------------------------------
160
161             if(list != null) {
162                  Node n = currentPointer;
```

```
163                 n.time -= 1;
164                 currentPointer = currentPointer.next;
165                 if(n.time == 0)
166                      deleteNodeWithID(n.jobID);
167                 System.out.print(n.jobID + " has " + n.time + " Ticks. ");
168                 printList();
169                 System.out.println();
170            } else {
171                 System.out.println("Error in T command. Job list empty.");
172            }
173       }
174
175       public static void iCommand(String[] strings) {
176            //-----------------------------------------------
177            //Verify node attributes are correct and add the
178            //node to the list.
179            //Precondition: String[] describing node attributes.
180            //Postcondition: attributes are verified and added
181            //to the list if acceptable.
182            //-----------------------------------------------
183            if(strings.length == 4) {
184                 String[] job = {strings[1], strings[2], strings[3]};
185                 Node newNode = verifyAndReturnNodeWithAttributes(job);
186                 if(newNode != null)
187                      addJob(newNode);
188            } else {
189                 System.out.println("Unsupported I command. Use: " +
190            "I JobID TIME JobName");
191            }
192       }
193
194       public static void dCommand() {
195            //-----------------------------------------------
196            //Print the list.
197            //Precondition: printList() method defined in class.
198            //Postcondition: list is printed.
199            //-----------------------------------------------
200            printList();
201            System.out.println();
202       }
203
204       /***Circular list methods***/
205       public static void addJob(Node newNode) {
206            //-----------------------------------------------
207            //Adds a job (node) to the end of the list
208            //referenced by the list class variable.
209            //Precondition: newNode passed as parameter. Class
210            //variable list referencing list.
211            //Postcondition: The node is added to the end of
212            //the list.
213            //-----------------------------------------------
214            if(list == null) {
215                 list = newNode;
216                 list.next = list;
217            }
218            else {
219                 Node first = list.next;
220                 list.next = newNode;
221                 newNode.next = first;
222                 list = newNode;
223            }
224       }
225
226       public static void deleteNodeWithID(int ID) {
```

```
227             //-------------------------------------------------
228             //Handle the input for the file object passed
229             //as a parameter. Continue reading input until
230             //true (quit) is returned by parseInput. Pass
231             //lines to parseInput to be interpreted and
232             //have the appropriate method executed.
233             //Precondition: File object referencing a file
234             //containing jobs and commands.
235             //Postcondition: Commands are executed line by
236             //line using parseInput until the end of the file
237             //or the Q command is reached.
238             //-------------------------------------------------
239             if(list != null) {
240                  if(list.next == list) {
241                       list = null;
242                       return;
243                  }
244
245                  Node curr = list;
246                  Node prev = curr;
247                  do {
248                       prev = curr;
249                       curr = curr.next;
250                       if(curr.jobID == ID) {
251                            prev.next = curr.next;
252                            if(curr == list)
253                                 list = prev.next;
254                       }
255                  } while(curr != list);
256             }
257        }
258
259        public static void printList() {
260             //-------------------------------------------------
261             //Prints each method present in the list
262             //referenced by the class variable list. Stops
263             //when the curr reference has traversed the entire
264             //list.
265             //Precondition: Class variable list.
266             //Postcondition: The list is traversed and each
267             //node's description is printed, along with an *
268             //signifying the current pointer.
269             //-------------------------------------------------
270             if(list != null) {
271                  Node curr = list;
272                  do {
273                       curr = curr.next;
274
275                       System.out.print(curr.getJobDescription());
276                       if(curr == currentPointer)
277                            System.out.print("*");
278                       if(curr != list)
279                            System.out.print("=>");
280
281                  } while(curr != list);
282             }
283        }

284    }//end list

285    class Node {
286         /*-------------------------------------------------------
287         CLASS NAME: Node
288         PROGRAMMER:   Samuel Jentsch
```

```
   289
   290          VARIABLE DICTIONARY:
   291             next - Node, pointer to next node in the list.
   292             name - String, the name of the job.
   293             jobID - int, the ID for the job.
   294             time - int, the time associated with the job.
   295          -----------------------------------------------------*/
   296          Node next;
   297          String name;
   298          int jobID;
   299          int time;
   300
   301          public Node() {
   302               //Initialize data fields to default values
   303               next = null;
   304               name = "";
   305               jobID = 0;
   306               time = 0;
   307          }
   308
   309          public Node(String name, int ID, int time, Node next) {
   310               //-------------------------------------------------
   311               //Initializes datafields to values passed as
   312               //parameters.
   313               //Precondition: parameters name, ID, time, and next
   314               //Postcondition: data fields are set to the values
   315               //set as parameters.
   316               //-------------------------------------------------
   317
   318               this.name = name;
   319               this.next = next;
   320               this.jobID = ID;
   321               this.time = time;
   322          }
   323
   324          public String getJobDescription() {
   325               //-------------------------------------------------
   326               //Return the job description.
   327               //Precondition: data fields.
   328               //Postcondition: A description containing the node
   329               //attributes is returned.
   330               //-------------------------------------------------
   331               return jobID + "/" + time + "/" + name;
   332          }
   333     }

:                              #null command
:
:
javac p3_Bonus.java           #compile the java file
java p3_Bonus             #execute the file from the current directory
1/4/Payroll*=>2/1/Student1=>3/5/Faculty1=>4/2/Registration
1 has 3 Ticks. 1/3/Payroll=>2/1/Student1*=>3/5/Faculty1=>4/2/Registration
2 has 0 Ticks. 1/3/Payroll=>3/5/Faculty1*=>4/2/Registration
1/3/Payroll=>3/5/Faculty1*=>4/2/Registration=>5/1/Student2
3 has 4 Ticks. 1/3/Payroll=>3/4/Faculty1=>4/2/Registration*=>5/1/Student2
4 has 1 Ticks. 1/3/Payroll=>3/4/Faculty1=>4/1/Registration=>5/1/Student2*
5 has 0 Ticks. 3/4/Faculty1=>4/1/Registration=>1/3/Payroll*
1 has 2 Ticks. 3/4/Faculty1*=>4/1/Registration=>1/2/Payroll
3 has 3 Ticks. 3/3/Faculty1=>4/1/Registration*=>1/2/Payroll=>6/2/Faculty2
4 has 0 Ticks. 3/3/Faculty1=>1/2/Payroll*=>6/2/Faculty2
1 has 1 Ticks. 3/3/Faculty1=>1/1/Payroll=>6/2/Faculty2*
6 has 1 Ticks. 3/3/Faculty1*=>1/1/Payroll=>6/1/Faculty2
3 has 2 Ticks. 3/2/Faculty1=>1/1/Payroll*=>6/1/Faculty2
```

```
1 has 0 Ticks. 3/2/Faculty1=>6/1/Faculty2*
3/2/Faculty1=>6/1/Faculty2*
Exiting...
3/2/Faculty1=>6/1/Faculty2*:
:
:
date                        #print the date
Wed Oct 23 09:01:21 CDT 2013
[cs241114@cs ~]$ exit
exit
```