

```

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

cat p5.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 p5.sh                #display the shell script file for the program
printf \\f                #issue a form feed (top of a new page)
cat -b p5.java            #display the source file with line numbers
:                          #null command
:
:
javac p5.java             #compile the java file
java p5                   #execute the file from the current directory
:
:
:
date                     #print the date
printf \\f                #issue a form feed (top of a new page)

cat -b p5.java            #display the source file with line numbers
1      /*
2      -----
3      PROGRAM NAME: Program 5, Queues and Inheritance
4      PROGRAMMER:   Samuel Jentsch
5      CLASS:        CSC 241.001, Fall 2013
6      INSTRUCTOR:   Dr. D. Dunn
7      DATE STARTED: October 27, 2013
8      DUE DATE:     October 28, 2013
9      REFERENCES:
10     Dr. Dunn: assignment information sheet

11     PROGRAM PURPOSE:
12     a. This program reads a series of commands from a file.
13     b. The program parses these commands and manipulates the Priority
14         Queue as dictated by the commands parsed. (Adding, printing,
15         deleting, inserting, etc).

16     VARIABLE DICTIONARY:
17         pQueue - PriorityQueue, queue for handling items read in from
18                 data file.
19
20     ADTs:
21         Queue

22     FILES USED:
23         p5.dat - a file containing commands for manipulating the Priority
24         Queue.
25     -----

```

```

26  */
27  import java.io.*;
28  import java.util.*;
29  public class p5 {
30      static PriorityQueue pQueue;
31
32      public static void main(String[] args) {
33
34          File file = new File("../instr/p5.dat");
35
36          pQueue = new PriorityQueue();
37
38          handleInput(file);
39
40          //Release memory
41          System.out.println("Releasing queue, System Shutdown!");
42          pQueue.dequeueAll();
43          pQueue = null;
44      }
45
46      public static void handleInput(File inputFile) {
47          //-----
48          //Handles input from file passed as a parameter.
49          //Passes lines read from file to parseCommand()
50          //and continues until end of file or parseCommand()
51          //returns true (exit).
52          //Preconditions: File inputFile passed as parameter.
53          //Method parseCommand defined in class.
54          //Postconditions: parseCommand() is called with a
55          //line of text and the correct methods are called.
56          //-----
57
58          try {
59              boolean exit = false;
60              Scanner fileInput = new Scanner(inputFile);
61              while(fileInput.hasNext() && !exit)
62                  exit = parseCommand(fileInput.nextLine());
63
64          } catch (FileNotFoundException e) {
65              // TODO Auto-generated catch block
66              e.printStackTrace();
67          }
68
69      }
70
71      public static boolean parseCommand(String command) {
72          //-----
73          //Takes a string command as a parameter and calls
74          //the corresponding method in the class. Returns true
75          //if the program should exit.
76          //Preconditions: String command passed as parameter.

```

```

75      //Postconditions: Corresponding command is called
76      //based on the string parameter passed. A message
77      //is printed if an unsupported command is read.
78      //Returns false if program should continue, false if
79      //not.
80      //-----
81
82      boolean exit = false;
83
84      char commandChar = command.charAt(0);
85
86      switch (commandChar) {
87      case 'I':
88          iCommand(command.substring(1));
89          break;
90      case 'R':
91          rCommand();
92          break;
93      case 'P':
94          pCommand();
95          break;
96      case 'Q':
97          exit = true;
98          break;
99      default:
100         System.out.println("Unsupported command.");
101     }
102
103     return exit;
104 }
105
106 public static void iCommand(String nodeString) {
107     //-----
108     //Insert node based on string passed as parameter.
109     //Creates the node and passes to the PriorityQueue
110     //enqueue method.
111     //Preconditions: nodeString passed as parameter.
112     //Postconditions: New node is created and passed
113     //to the PriorityQueue enqueue method to be
114     //inserted.
115     //-----
116     String nodeName = nodeString.substring(0, 1);
117     int priority = Integer.parseInt(nodeString.substring(1));
118     System.out.println("Inserting " + nodeName + " with priority " + priority);
119     Node newNode = new Node(nodeName, priority, null);
120     pQueue.enqueue(newNode);
121 }
122
123 public static void rCommand() {
124     //-----
125     //Removes first entry from queue (Dequeues).
126     //Preconditions: pQueue class variable.
127     //Postconditions: Node is dequeued from the queue.

```

```

128         //A message is printed if the queue was already
129         //empty.
130         //-----
131         if(!pQueue.isEmpty()) {
132             Node n = pQueue.dequeue();
133             System.out.println(n.item + " removed.");
134         } else {
135             System.out.println("Cannot remove from empty queue");
136         }
137     }
138
139     public static void pCommand() {
140         //-----
141         //Prints items in the queue by calling printQueue().
142         //Preconditions: printQueue() method defined in
143         //class.
144         //Postconditions: The queue pQueue is printed.
145         //-----
146
147         printQueue();
148     }
149
150     public static void printQueue() {
151         //-----
152         //Prints queue pQueue.
153         //Preconditions: class variable pQueue.
154         //Postconditions: the queue pQueue is traversed
155         //and its items are stored in an auxiliary queue.
156         //the items are printed as they are stored in the
157         //aux queue and then added back to the pQueue.
158         //-----
159
160         if(pQueue.isEmpty()) {
161             System.out.println("Queue is empty.");
162             return;
163         }
164
165         Queue auxQueue = new Queue();
166
167         while(!pQueue.isEmpty()) {
168             Node n = pQueue.dequeue();
169             System.out.print(n.item + "/" + n.priority);
170             if(!pQueue.isEmpty())
171                 System.out.print("=>");
172             auxQueue.enqueue(n);
173         }
174
175         while(!auxQueue.isEmpty()) {
176             pQueue.enqueue(auxQueue.dequeue());
177         }
178
179         System.out.println();
180     }

```

```

180
181 }//end p5

182 class Queue {
183     /*
184     -----
185     CLASS NAME: Queue
186
187     DESCRIPTION:
188         Implements ADT Queue with a reference based implementation. Uses
189         list as a reference to the reference based list used as the data
190         structure.
191
192     VARIABLE DICTIONARY:
193         list - Node, pointer to the last item in the reference based list.
194     -----
195     */
196
197     protected Node list;
198
199     public Queue() {
200         //-----
201         //Initialize default Queue.
202         //Preconditions: Class variable list.
203         //Postconditions: this.list initialized to null.
204         //-----
205         this.list = null;
206     }
207
208     public void dequeueAll() {
209         //-----
210         //Set list to null (remove all items)
211         //Preconditions: Class variable list.
212         //Postconditions: list is set to null.
213         //-----
214
215         this.list = null;
216     }
217
218     public boolean isEmpty() {
219         //-----
220         //Returns of list is equal to null or not.
221         //Preconditions: Class variable list.
222         //Postconditions:
223         //-----
224
225         return this.list == null;
226     }
227
228     public void enqueue(Node newNode) {
229         //-----
230         //Insert newNode passed as parameter into the end
231         //of the queue.

```

```

232         //Preconditions: Class variable list. Parameter
233         //newNode.
234         //Postconditions: newNode is inserted into the end
235         //of the queue.
236         //-----
237
238         newNode.next = null;
239
240         if(list == null) {
241             list = newNode;
242             list.next = list;
243         } else {
244             Node first = list.next;
245             list.next = newNode;
246             newNode.next = first;
247             list = newNode;
248         }
249     }
250
251     public Node dequeue() {
252         //-----
253         //Removes and returns first item in the queue.
254         //Preconditions: Class variable list.
255         //Postconditions: First item is removed from queue
256         //and returned. An error message is printed if the
257         //queue was already empty.
258         //-----
259
260         if(!isEmpty()) {
261             Node first = list.next;
262
263             if(list.next == list)
264                 list = null;
265             else {
266                 list.next = first.next;
267             }
268             return first;
269         } else {
270             System.out.println("Error: cannot dequeue from empty queue.");
271             return null;
272         }
273     }
274
275     public String peek() {
276         //-----
277         //Returns first item in the queue.
278         //Preconditions: Class variable list.
279         //Postconditions: First item in queue is returned.
280         //An error message is printed if the queue is
281         //empty.
282         //-----
283
284         if(!isEmpty()) {

```

```

284         String item = list.next.item;
285         return item;
286     } else {
287         System.out.println("Error: cannot peek from empty queue.");
288         return null;
289     }
290 }
291
292 }//end Queue

```

```

293 class PriorityQueue extends Queue {
294     /*
295     -----
296     CLASS NAME: PriorityQueue, extends Queue
297
298     DESCRIPTION:
299         Overrides the enqueue method of the class Queue to implement an
300         enqueue based on the priority level of the node being inserted.
301     -----
302     */
303
304     @Override
305     public void enqueue(Node newNode) {
306         //-----
307         //Inserts node into queue based on the priority
308         //level in the newNode passed as parameter.
309         //Preconditions: newNode passed as parameter.
310         //Class variable list.
311         //Postconditions: Inserts node into list based on
312         //node priority.
313         //-----
314
315         newNode.next = null;
316         //Use priority queue implementation.
317
318         if(super.list == null) {
319             list = newNode;
320             list.next = list;
321         } else {
322             boolean inserted = false;
323             Node cur = list;
324             Node prev = cur;
325             do {
326                 prev = cur;
327                 cur = cur.next;
328
329                 if(newNode.priority < cur.priority) {
330                     prev.next = newNode;
331                     newNode.next = cur;
332                     inserted = true;
333                 }
334

```

```

335         } while(cur != list && !inserted);
336
337         if(!inserted) {
338             //insert at end of list
339             Node first = list.next;
340             list.next = newNode;
341             newNode.next = first;
342             list = newNode;
343         }
344     }
345
346     }//end priority enqueue
347
348 }//end PriorityQueue
349
350 class Node {
351     /*
352     -----
353     CLASS NAME: Node
354
355     VARIABLE DICTIONARY:
356         next - Node, pointer to the the next node.
357         item - String, contains the string description of the node.
358         priority - int, contains the node's priority level.
359     -----
360     */
361
362     Node next;
363     String item;
364     int priority;
365
366     public Node() {
367         //Creates a default node
368         //Initializes data fields to default values.
369         this.item = "";
370         this.next = null;
371         priority = 0;
372     }
373
374     public Node(String item, Node next) {
375         //Creates a node, initializing item and next
376         //to values passed as parameters.
377         this.item = item;
378         this.next = next;
379         this.priority = 0;
380     }
381
382     public Node(String item, int priority, Node next) {
383         //Creates a node, initializing all data fields
384         //to values passed as parameters.
385         this.item = item;
386         this.next = next;

```



```

387         this.priority = priority;
388     }
389
390 }//end Node
:
:
:
:
javac p5.java          #compile the java file
java p5                #execute the file from the current directory
Inserting c with priority 10
Inserting a with priority 12
Inserting i with priority 11
c/10=>i/11=>a/12
c removed.
Inserting d with priority 15
i removed.
a/12=>d/15
a removed.
d removed.
Queue is empty.
Inserting b with priority 2
b/2
Releasing queue, System Shutdown!
:
:
:
:
date                  #print the date
Mon Oct 28 09:07:48 CDT 2013

```