```
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	---------------------------------------------------------------------
```

```
*/

import java.io.*;
import java.util.*;

public class p5 {
     static PriorityQueue pQueue;

     public static void main(String[] args) {

          File file = new File("../instr/p5.dat");

          pQueue = new PriorityQueue();

          handleInput(file);

          //Release memory
          System.out.println("Releasing queue, System Shutdown!");
          pQueue.dequeueAll();
          pQueue = null;
     }

     public static void handleInput(File inputFile) {
          //-------------------------------------------------
          //Handles input from file passed as a parameter.
          //Passes lines read from file to parseCommand()
          //and continues until end of file or parseCommand()
          //returns true (exit).
          //Preconditions: File inputFile passed as parameter.
          //Method parseCommand defined in class.
          //Postconditions: parseCommand() is called with a
          //line of text and the correct methods are called.
          //-------------------------------------------------

          try {
               boolean exit = false;
               Scanner fileInput = new Scanner(inputFile);
               while(fileInput.hasNext() && !exit)
                    exit = parseCommand(fileInput.nextLine());

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

     }

     public static boolean parseCommand(String command) {
          //-------------------------------------------------
          //Takes a string command as a parameter and calls
          //the corresponding method in the class. Returns true
          //if the program should exit.
          //Preconditions: String command passed as parameter.
          //Postconditions: Corresponding command is called
          //based on the string parameter passed. A message
          //is printed if an unsupported command is read.
          //Returns false if program should continue, false if
          //not.
          //----------------------------------------------------

          boolean exit = false;

          char commandChar = command.charAt(0);

          switch (commandChar) {
          case 'I':
               iCommand(command.substring(1));
               break;
          case 'R':
               rCommand();
               break;
          case 'P':
               pCommand();
               break;
          case 'Q':
               exit = true;
               break;
          default:
               System.out.println("Unsupported command.");
          }

          return exit;
     }

     public static void iCommand(String nodeString) {
          //--------------------------------------------------
          //Insert node based on string passed as parameter.
          //Creates the node and passes to the PriorityQueue
          //enqueue method.
          //Preconditions: nodeString passed as parameter.
          //Postconditions: New node is created and passed
          //to the PriorityQueue enqueue method to be
          //inserted.
          //--------------------------------------------------
          String nodeName = nodeString.substring(0, 1);
          int priority = Integer.parseInt(nodeString.substring(1));
          System.out.println("Inserting " + nodeName + " with priority " + priority);
          Node newNode = new Node(nodeName, priority, null);
          pQueue.enqueue(newNode);
     }

     public static void rCommand() {
          //--------------------------------------------------
          //Removes first entry from queue (Dequeues).
          //Preconditions: pQueue class variable.
          //Postconditions: Node is dequeued from the queue.
```

```java
          //A message is printed if the queue was already
          //empty.
          //----------------------------------------------
          if(!pQueue.isEmpty()) {
               Node n = pQueue.dequeue();
               System.out.println(n.item + " removed.");
          } else {
               System.out.println("Cannot remove from empty queue");
          }
     }

     public static void pCommand() {
          //-----------------------------------------------
          //Prints items in the queue by calling printQueue().
          //Preconditions: printQueue() method defined in
          //class.
          //Postconditions: The queue pQueue is printed.
          //-----------------------------------------------

          printQueue();
          }

     public static void printQueue() {
          //----------------------------------------------
          //Prints queue pQueue.
          //Preconditions: class variable pQueue.
          //Postconditions: the queue pQueue is traversed
          //and its items are stored in an auxiliary queue.
          //the items are printed as they are stored in the
          //aux queue and then added back to the pQueue.
          //----------------------------------------------

          if(pQueue.isEmpty()) {
               System.out.println("Queue is empty.");
               return;
          }

          Queue auxQueue = new Queue();

          while(!pQueue.isEmpty()) {
               Node n = pQueue.dequeue();
               System.out.print(n.item + "/" + n.priority);
               if(!pQueue.isEmpty())
                    System.out.print("=>");
               auxQueue.enqueue(n);
          }

          while(!auxQueue.isEmpty()) {
               pQueue.enqueue(auxQueue.dequeue());
          }

          System.out.println();
     }
```

```java

}//end p5

class Queue {
	/*
	----------------------------------------------------------------------
	CLASS NAME: Queue

	DESCRIPTION:
		Implements ADT Queue with a reference based implementation. Uses
		list as a reference to the reference based list used as the data
		structure.

	VARIABLE DICTIONARY:
		list - Node, pointer to the last item in the reference based list.
	----------------------------------------------------------------------
	*/

	protected Node list;

	public Queue() {
		//-----------------------------------------------
		//Initialize default Queue.
		//Preconditions: Class variable list.
		//Postconditions: this.list initialized to null.
		//-----------------------------------------------
		this.list = null;
	}

	public void dequeueAll() {
		//-----------------------------------------------
		//Set list to null (remove all items)
		//Preconditions: Class variable list.
		//Postconditions: list is set to null.
		//-----------------------------------------------

		this.list = null;
	}

	public boolean isEmpty() {
		//-----------------------------------------------
		//Returns of list is equal to null or not.
		//Preconditions: Class variable list.
		//Postconditions:
		//-----------------------------------------------

		return this.list == null;
	}

	public void enqueue(Node newNode) {
		//-----------------------------------------------
		//Insert newNode passed as parameter into the end
		//of the queue.
		//Preconditions: Class variable list. Parameter
		//newNode.
		//Postconditions: newNode is inserted into the end
		//of the queue.
		//----------------------------------------------

		newNode.next = null;

		if(list == null) {
			list = newNode;
			list.next = list;
		} else {
			Node first = list.next;
			list.next = newNode;
			newNode.next = first;
			list = newNode;
		}
	}

	public Node dequeue() {
		//-----------------------------------------------
		//Removes and returns first item in the queue.
		//Preconditions: Class variable list.
		//Postconditions: First item is removed from queue
		//and returned. An error message is printed if the
		//queue was already empty.
		//-----------------------------------------------

		if(!isEmpty()) {
			Node first = list.next;

			if(list.next == list)
				list = null;
			else {
				list.next = first.next;
			}
			return first;
		} else{
			System.out.println("Error: cannot dequeue from empty queue.");
			return null;
		}
	}

	public String peek() {
		//----------------------------------------------
		//Returns first item in the queue.
		//Preconditions: Class variable list.
		//Postconditions: First item in queue is returned.
		//An error message is printed if the queue is
		//empty.
		//----------------------------------------------

		if(!isEmpty()) {
```

```
                String item = list.next.item;
                return item;
            } else {
                System.out.println("Error: cannot peek from empty queue.");
                return null;
            }
        }

}//end Queue


class PriorityQueue extends Queue {
        /*
        ---------------------------------------------------------------------
        CLASS NAME: PriorityQueue, extends Queue

        DESCRIPTION:
            Overrides the enqueue method of the class Queue to implement an
            enqueue based on the priority level of the node being inserted.
        ---------------------------------------------------------------------
        */

        @Override
        public void enqueue(Node newNode) {
            //-------------------------------------------------
            //Inserts node into queue based on the priority
            //level in the newNode passed as parameter.
            //Preconditions: newNode passed as parameter.
            //Class variable list.
            //Postconditions: Inserts node into list based on
            //node priority.
            //-------------------------------------------------

            newNode.next = null;
            //Use priority queue implementation.

            if(super.list == null) {
                list = newNode;
                list.next = list;
            } else {
                boolean inserted = false;
                Node cur = list;
                Node prev = cur;
                do {
                    prev = cur;
                    cur = cur.next;

                    if(newNode.priority < cur.priority) {
                        prev.next = newNode;
                        newNode.next = cur;
                        inserted = true;
                    }

                } while(cur != list && !inserted);

                if(!inserted) {
                    //insert at end of list
                    Node first = list.next;
                    list.next = newNode;
                    newNode.next = first;
                    list = newNode;
                }
            }

        }//end priority enqueue

    }//end PriorityQueue

    class Node {
        /*
        ----------------------------------------------------------------------
        CLASS NAME: Node

        VARIABLE DICTIONARY:
            next - Node, pointer to the the next node.
            item - String, contains the string description of the node.
            priority - int, contains the node's priority level.
        ----------------------------------------------------------------------
        */

        Node next;
        String item;
        int priority;


        public Node() {
            //Creates a default node
            //Initializes data fields to default values.
            this.item = "";
            this.next = null;
            priority = 0;
        }

        public Node(String item, Node next) {
            //Creates a node, initializing item and next
            //to values passed as parameters.
            this.item = item;
            this.next = next;
            this.priority = 0;
        }

        public Node(String item, int priority, Node next) {
            //Creates a node, initializing all data fields
            //to values passed as parameters.
            this.item = item;
            this.next = next;
            this.priority = priority;
        }

    }//end Node
:                               #null command
:
:
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
```