```
Script started on Wed 02 Oct 2013 03:30:27 AM CDT
[cs241114@cs ~]$ p3.sh
printf \\n\\n\\n          #print three blank lines



cat p3.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.sh                #display the shell script file for the program
printf \\f               #issue a form feed (top of a new page)
cat -b p3.java           #display the source file with line numbers
:                        #null command
:
:
javac p3.java            #compile the java file
java p3                  #execute the file from the current directory
:
:
:
date                     #print the date
printf \\f               #issue a form feed (top of a new page)
^Lcat -b p3.java         #display the source file with line numbers
     1  /*
     2  PROGRAM NAME: Lab 3
     3  PROGRAMMER:   Samuel Jentsch
     4  CLASS:        CSC 241.001, Fall 2013
     5  INSTRUCTOR:   Dr. D. Dunn
     6  DATE STARTED: September 25, 2013
     7  DUE DATE:     October 2, 2013
     8  REFERENCES:   Computer Science
     9                Introduction to Java Programming
    10                Y. Daniel Liang
    11             Dr. Dunn: assignment information sheet

    12  PROGRAM PURPOSE:
    13  a. This program reads a list of company names from a file
    14           and creates a list of CompanyNodes from the company names.
    15  b. The program then reads in a list of commands from a commands file,
    16           parses the commands, and calls methods corresponding to the commands.
    17  c. Based on the commands, the program manipulates reference based linked
    18           lists by adding employees, changing employees, firing employees, etc.
```

```
19  VARIABLE DICTIONARY:
20    companyHead - CompanyNode, the first CompanyNode in the list.
21                                  The first node corresponds to the Unemployed company
22
23  ADTs:
24    List

25  FILES USED:
26    p3company.dat - a file containing company names
27    p3commands.dat - a file containing command to manipulate employees
28
29  Verification:
30          Turned in Excel spreadsheet.

31  ----------------------------------------------------------------
32  */

33  import java.util.*;
34  import java.io.*;

35  public class p3 {
36
37          //pointer to the first node in the company list.
38          static CompanyNode companyHead;
39
40          public static void main(String[] args) {
41                  //Read in the companies and sort the into a list in
42                  //alphabetical order.
43                  populateCompanyList(new File("../instr/p3company.dat"));
44
45                  //Add a CompanyNode to the beginning of the list of
46                  //companies to be used as a placeholder for unemployed
47                  //persons.
48                  addUnemployedCompany();
49
50                  runCommandsFromFile(new File("../instr/p3command.dat"));
51          }//end main
52
53          /***Input Methods***/
54          public static void runCommandsFromFile(File file) {
55                  //-----------------------------------------------------
56                  //Reads commands as strings from the file passed. Passes
57                  //the command to parseCommand() to be parsed as the
58                  //commands are read.
59                  //Preconditions: File object passed as parameter.
60                  //Postconditions: Commands are read as lines from file and
```

```
61              //passed to parseCommand() until the entire file is read
62              //or parseCommand returns true (signals end).
63              //-------------------------------------------------------
64
65              try {
66                      Scanner fileReader = new Scanner(file);
67                      boolean stopReading = false;
68
69                      while(fileReader.hasNextLine() && stopReading == false) {
70                              stopReading = parseCommand(fileReader.nextLine());
71                      }//end while
72              } catch (FileNotFoundException e) {
73                      e.printStackTrace();
74                      System.out.println("Issue ocurred with commands file.");
75              }//end catch
76      }//end runCommandsFromFile
77
78      public static boolean parseCommand(String commandLine) {
79              //-------------------------------------------------------------
80              //This method takes a string parameter commandLine, splits
81              //it into an array commands[], and calls a method corresponding
82              //to commands[0]. If the command called requires parameters,
83              //the other indices of commands (the parameters of the command)
84              //are passed.
85              //Preconditions: commandLine is a nonempty string.
86              //Postconditions: a method corresponding to the first word in
87              //commandLine is called with appropriate parameters. If the
88              //command is not found, the program notifies the user.
89              //-------------------------------------------------------------
90
91              boolean shouldEnd = false;
92
93              String[] commands = commandLine.split(" ");
94
95
96              String command = commands[0];
97              command = command.toUpperCase();
98              if(command.matches("JOIN")) {
99                      joinCommand(commands[1], commands[2]);
100             } else if(command.matches("QUIT")) {
101                     quitCommand(commands[1]);
102             } else if(command.matches("CHANGE")) {
103                     changeCommand(commands[1], commands[2]);
104             } else if(command.matches("PROMOTE")) {
105                     promoteCommand(commands[1]);
106             } else if(command.matches("DEMOTE")) {
```

```
107                     demoteCommand(commands[1]);
108             } else if(command.matches("PAYDAY")) {
109                     paydayCommand();
110             } else if(command.matches("EMPLOYEES")) {
111                     System.out.println("*******************************************");
112                     employeesCommand(commands[1]);
113                     System.out.println("*******************************************");
114             } else if(command.matches("UNEMPLOYED")) {
115                     System.out.println("*******************************************");
116                     unemployedCommand();
117                     System.out.println("*******************************************");
118             } else if(command.matches("DUMP")) {
119                     System.out.println("*******************************************");
120                     dumpCommand();
121                     System.out.println("*******************************************");
122             } else if(command.matches("END")) {
123                     shouldEnd = true;
124             } else {
125                     System.out.println("Unrecognized command.");
126             }
127
128             return shouldEnd;
129     }//end parse command
130
131     public static void joinCommand(String employeeName, String companyName) {
132             //-------------------------------------------------------
133             //Takes two strings, employeeName and companyName, as
134             //parameters. Uses the methods findCompanyWithName() to get
135             //a reference to the CompanyNode with a name matching
136             //companyName. If a matching company is found, Unemployed
137             //is checked to see if the employee is already created or
138             //if a new EmployeeNode should be created with the
139             //employeeName passed. The EmployeeNode is then inserted
140             //into the company using insertEmployeeInCompany().
141             //Preconditions: findCompanyWithName(String companyName),
142             //searchUnemployedForEmployeeWithName(String employeeName),
143             // and insertEmployeeInCompany(CompanyNode company,
144             // EmployeeNode employee) methods defined in class.
145             //Postconditions: If a matching company is found, an
146             //EmployeeNode is either found in the currently
147             //unemployed employees, or created using the name passed.
148             //The EmployeeNode is then inserted into the list of
149             //employees contained in the company.
150             //-------------------------------------------------------
151
152             EmployeeNode newEmployee = null;
```

```
153             CompanyNode company = findCompanyWithName(companyName);
154             if(company != null) {
155                     //Check to see if employee is currently unemployed or a new employee
156                     newEmployee = searchUnemployedForEmployeeWithName(employeeName);
157                     if(newEmployee != null)
158                             newEmployee.next = null;
159                     else
160                             newEmployee = new EmployeeNode(employeeName, null);
161
162                     insertEmployeeInCompany(company, newEmployee);
163             }
164             else
165                     System.out.println("Company not found " + companyName + ".");
166     }
167
168     public static void quitCommand(String employeeName) {
169             //---------------------------------------------------------
170             //This method removes the EmployeeNode matching the string
171             //employeeName passed as a parameter from the current
172             //company containing it and adds it to the unemployed
173             //company.
174             //Preconditions: companyHead class variable referencing
175             //list of companies. Method deleteAndReturnEmployeeWithName().
176             //Postconditions: The EmployeeNode matching employee name is
177             //removed from the current company containing it and added
178             //to the unemployed company node (companyHead).
179             //---------------------------------------------------------
180
181             EmployeeNode employee = deleteAndReturnEmployeeWithName(employeeName);
182
183             if(employee != null)
184                     addToUnemployed(employee);
185     }//quit
186
187     public static void changeCommand(String employeeName, String companyName) {
188             //---------------------------------------------------------
189             //Takes two strings, employeeName and companyName, as
190             //parameters. Uses the method deleteAndReturnEmployeeWithName()
191             //to remove the EmployeeNode with a name matching employeeName
192             //from its current list and return it. Then uses
193             //the method findCompanyWithName() to get a reference to the
194             //CompanyNode with a name matching companyName. Finally
195             //the method insertEmployeeInCompany() is used to insert
196             //the employee in the company.
197             //Postconditions: Strings employeeName and companyName passed
198             //as parameters. The methods described above must be defined.
```

```
199             //Preconditions: The EmployeeNode with a name matching
200             //employeeName is removed from its current list and inserted
201             //into the CompanyNode employee list for the company whose name
202             //matches companyName.
203             //-------------------------------------------------------
204
205             EmployeeNode employee = deleteAndReturnEmployeeWithName(employeeName);
206             CompanyNode company = findCompanyWithName(companyName);
207
208             employee.next = null;
209             insertEmployeeInCompany(company, employee);
210         }//change
211
212         public static void promoteCommand(String employeeName) {
213             //-------------------------------------------------------
214             //Finds an EmployeeNode matching the string employeeName
215             //and moves the node up in the list if it is not already
216             //the last node.
217             //Preconditions: Class variable companyHead. String
218             //employeeName passed as string.
219             //Postconditions: The EmployeeNode matching employeeName is
220             //moved up in the list and the references of the nodes in
221             //the list are altered accordingly.If the EmployeeNode was
222             //already last, no change occurs.
223             //-------------------------------------------------------
224
225             CompanyNode currentCompany = companyHead;
226
227             boolean found = false;
228
229             while(currentCompany != null && found == false) {
230
231                 EmployeeNode currentEmployee = currentCompany.employeeListHead;
232                 EmployeeNode prev = currentEmployee;
233
234                 if(currentEmployee != null && currentEmployee.name.matches(employeeName)
235                         && currentEmployee.next != null) {
236                     currentCompany.employeeListHead = currentEmployee.next;
237                     currentEmployee.next = currentCompany.employeeListHead.next;
238                     currentCompany.employeeListHead.next = currentEmployee;
239                     found = true;
240                 }
241                 while(currentEmployee != null && currentEmployee.next != null
242                         && found == false) {
243                     if(currentEmployee.name.matches(employeeName)) {
244                         EmployeeNode curNext = currentEmployee.next;
```

```
245                                 currentEmployee.next = curNext.next;
246                                 prev.next = curNext;
247                                 curNext.next = currentEmployee;
248                                 found = true;
249                             }
250                             prev = currentEmployee;
251                             currentEmployee = currentEmployee.next;
252
253                     }//end while
254
255                     currentCompany = currentCompany.next;
256             }//end while
257
258             //if(!found)
259                     //System.out.println("Could not find employee with name " + employeeName + ".");
260     }//promote
261
262     public static void demoteCommand(String employeeName) {
263             //-------------------------------------------------------
264             //Finds an EmployeeNode matching the string employeeName
265             //and moves the node down in the list if it is not already
266             //the first node.
267             //Preconditions: Class variable companyHead. String
268             //employeeName passed as string.
269             //Postconditions: The EmployeeNode matching employeeName is
270             //moved down in the list and the references of the nodes in
271             //the list are altered accordingly.If the EmployeeNode was
272             //already first, no change occurs.
273             //-------------------------------------------------------
274
275             CompanyNode currentCompany = companyHead;
276
277             boolean found = false;
278
279             while(currentCompany != null && found == false) {
280
281                     EmployeeNode currentEmployee = currentCompany.employeeListHead;
282                     EmployeeNode prev = currentEmployee;
283                     EmployeeNode prevPrev = null;
284
285                     if(currentEmployee != null && currentEmployee.name.matches(employeeName)) {
286                             found = true;
287                     }
288                     while(currentEmployee != null && found == false) {
289                             if(currentEmployee.name.matches(employeeName)) {
290                                     if(prevPrev != null)
```

```
291                                         prevPrev.next = currentEmployee;
292
293                                 prev.next = currentEmployee.next;
294
295                                 currentEmployee.next = prev;
296
297                                 if(prevPrev == null)
298                                         currentCompany.employeeListHead = currentEmployee;
299
300                                 found = true;
301                         }
302
303                         if(prev != currentEmployee) {
304                                 prevPrev = prev;
305                         }
306                         prev = currentEmployee;
307                         currentEmployee = currentEmployee.next;
308                 }//end while
309
310                 currentCompany = currentCompany.next;
311         }//end while
312
313         //if(!found)
314                 //System.out.println("Could not find employee with name " + employeeName + ".");
315 }//demote
316
317 public static void paydayCommand() {
318         //-------------------------------------------------------
319         //The companyList and the lists of employees are traversed.
320         //As each employee list is traversed, a rank variable is
321         //set to  1 and incremented as the new list is traversed.
322         //If the company being traversed is the Unemployed company,
323         //the amountEarned field of the EmployeeNode is incremented
324         //by 50. If the company being traversed is not the Unemployed
325         //company, the EmployeeNode amountEarned field is incremented
326         //by (1000 * rank).
327         //Preconditions: a class variable companyHead referencing
328         //a CompanyNode.
329         //Postconditions: The amountEarned field of every employee
330         //is incremented by 50 if unemployed, or (rank * 1000) if
331         //employed.
332         //-------------------------------------------------------
333
334         CompanyNode currentCompany = companyHead;
335
336         while(currentCompany != null) {
```

```
337                     EmployeeNode currentEmployee = currentCompany.employeeListHead;
338
339                     double pay = 0;
340                     int rank = 1;
341                     while(currentEmployee != null) {
342
343                             //change pay rate based on if the employee
344                             //is unemployed or employed companyHead is the
345                             //unemployed company
346                             if(currentCompany == companyHead)
347                                     pay = 50;
348                             else
349                                     pay = 1000 * rank;
350
351                             currentEmployee.amountEarned += pay;
352                             rank++;
353                             currentEmployee = currentEmployee.next;
354                     }//end while
355
356                     currentCompany = currentCompany.next;
357             }//end while
358
359     }//payday
360
361     public static void employeesCommand(String companyName) {
362             //-------------------------------------------------------
363             //This method takes a string parameter companyName, uses the
364             //method findCompanyWithName() to return a reference to the
365             //company with a name matching companyName, and prints the
366             //employees in the company returned using the
367             //printEmployeesInCompany() method if the company returned
368             //is not equal to null.
369             //Preconditions: A string companyName passed as a parameter.
370             //methods findCompanyWithName() and printEmployeesInCompany()
371             //defined in the class.
372             //Postconditions: The employees present in the CompanyNode
373             //with the name companyName are printed to the console. If
374             //the company is not found, nothing is displayed.
375             //-------------------------------------------------------
376
377             CompanyNode company = findCompanyWithName(companyName);
378             if(company != null)
379                     printEmployeesInCompany(company);
380     }//employees
381
382     public static void unemployedCommand() {
```

```
383             //-------------------------------------------------------
384             //This method prints out all of the employees present
385             //in the Unemployed company (companyHead).
386             //Preconditions: Class variable companyHead referring to
387             //the unemployed company.
388             //Postconditions: The employees present in the Unemployed
389             //company are displayed to the console. If there are no
390             //employees, a message is displayed to the console.
391             //-------------------------------------------------------
392
393             if(companyHead != null && companyHead.employeeListHead != null) {
394                     printEmployeesInCompany(companyHead);
395             } else {
396                     System.out.println("There are currently no unemployed workers.");
397             }
398     }//unemployed
399
400     public static EmployeeNode searchUnemployedForEmployeeWithName(String employeeName) {
401             //-------------------------------------------------------------
402             //Searches for an EmployeeNode in the Unemployed company
403             //with a name matching the employeeName string passed as a
404             //parameter. If a matching employee is found, the object is
405             //removed from the Unemployed company employee list and
406             //returned.
407             //Preconditions: Class variable companyHead referencing the
408             //Unemployed company. String employeeName passed as parameter.
409             //Postconditions: The EmployeeNode list referenced in the
410             //Unemployed company is traversed. If an EmployeeNode with a
411             //name matching employeeName is found, the EmployeeNode is
412             //returned. If a match is not found, null is returned.
413             //-------------------------------------------------------------
414
415             CompanyNode currentCompany = companyHead;
416             boolean found = false;
417                     EmployeeNode currentEmployee = currentCompany.employeeListHead;
418                     EmployeeNode prev = currentEmployee;
419
420                     if(currentEmployee != null && currentEmployee.name.matches(employeeName)) {
421                             //Delete first employee if it matches or the list is empty.
422                             currentCompany.employeeListHead =
currentCompany.employeeListHead.next;
423
424                             //Set to null for insertion into unemployed
425                             currentEmployee.next = null;
426                             found = true;
```

```
427                         return currentEmployee;
428                     }
429                     while(currentEmployee != null && found == false) {
430                         if(currentEmployee.name.matches(employeeName)) {
431                             prev.next = currentEmployee.next;
432
433                             found = true;
434                             return currentEmployee;
435                         }
436                         prev = currentEmployee;
437                         currentEmployee = currentEmployee.next;
438                     }//end while
439
440             return null;
441             }//end searchUnemployedForEmployeeWithName
442
443         public static void dumpCommand() {
444             //-------------------------------------------------------
445             //This method prints all the employees present in each
446             //CompanyNode present in the list referenced by companyHead.
447             //The employees in the unemployed company are printed last.
448             //Preconditions: A list of CompanyNodes referenced by the
449             //class variable companyHead.
450             //Postconditions: The employees in each company are printed
451             //to the console (employee name and amountEarned). The
452             //employees in unemployed are printed last.
453             //-------------------------------------------------------
454
455             CompanyNode cur = companyHead.next;
456             while(cur != null) {
457                 printEmployeesInCompany(cur);
458                 System.out.println();
459                 cur = cur.next;
460             }
461
462             System.out.println();
463             printEmployeesInCompany(companyHead);
464         }//dump
465
466         public static void endCommand() {
467             //-------------------------------------------------------
468             //Postconditions: The program is running.
469             //Preconditions: The program is exited with status code 0.
470             //-------------------------------------------------------
471
472             System.exit(0);
```

```
473         }//end
474
475         /********Node Methods********/
476         public static void populateCompanyList(File companyFile) {
477                 //-------------------------------------------------------
478                 //Read in the company names from the source file and
479                 //create new CompanyNode objects based on the names
480                 //read. Add them to the companyList referenced by
481                 //companyHead using the insertCompanyInOrder() method.
482                 //Postconditions: File object passed to method containing
483                 //strings separated by lines.
484                 //Preconditions: CompanyNodes are created based on the
485                 //strings read from the file and inserted into the company
486                 //list.
487                 //-------------------------------------------------------
488
489                 try {
490                         Scanner fileReader = new Scanner(companyFile);

491                         while(fileReader.hasNextLine()) {
492                                 CompanyNode newCompanyNode = new
CompanyNode(fileReader.nextLine(), null);
493                                 insertCompanyInOrder(newCompanyNode);
494                         }

495                 } catch (FileNotFoundException e) {
496                         e.printStackTrace();
497                         System.out.println("Issue ocurred with company file.");
498                 }//end catch
499         }//end populateCompanyList

500         public static void insertCompanyInOrder(CompanyNode newNode) {
501                 //-------------------------------------------------------
502                 //Takes a CompanyNode in as parameter and inserts it in the
503                 //list of CompanyNodes referenced by the class variable
504                 //in alphabetical order (A-Z) based on the name data field
505                 //in CompanyNode.
506                 //Preconditions: A list of CompanyNodes referenced by the
507                 //class data field companyHead. A CompanyNode newNode
508                 //passed as a parameter that is initialized with a value
509                 //for name (newNode != null && newNode.name != null).
510                 //Postconditions: The CompanyNode newNode is inserted into
511                 //the list referenced by companyHead in alphabetical order.
512                 //-------------------------------------------------------
513
514                 CompanyNode cur = companyHead;
```

```
515             CompanyNode prev = cur;

516             if(companyHead == null || newNode.name.compareTo(companyHead.name) < 0) {
517                     //insert at beginning
518                     newNode.next = cur;
519                     companyHead = newNode;
520             } else {//insert in list
521                     boolean inserted = false;
522                     while(cur != null && inserted == false) {
523                             //insert at position
524                             if(newNode.name.compareTo(cur.name) < 0) {
525                                     //insert in list
526                                     prev.next = newNode;
527                                     newNode.next = cur;
528                                     inserted = true;
529                             }
530                             prev = cur;
531                             cur = cur.next;
532                     }//end while
533                     if(inserted == false) {
534                             //insert at end of list
535                             prev.next = newNode;
536                     }//end if
537
538             }//end else
539
540     }//end insertInOrder
541
542     public static CompanyNode findCompanyWithName(String companyName) {
543             //-------------------------------------------------------
544             //Traverse the list of companies and search for one with a
545             // name matching companyName. If a match is found, return
546             //the reference to the CompanyNode with the matching name.
547             //If a match is not found, the method returns null.
548             //Postconditions: A class variable called companyHead,
549             //referencing the first node CompanyNode in the list.
550             //Preconditions: A CompanyNode reference with a name
551             //matching companyName is returned if a match is found.
552             //Null is returned if a match is not found.
553             //-------------------------------------------------------
554
555             CompanyNode cur = companyHead;
556             while(cur != null) {
557                     if(cur.name.matches(companyName))
558                             return cur;
559                     cur = cur.next;
```

```
560             }
561
562             System.out.println("Could not find company with name " + companyName + ".");
563             return null;
564         }//findCompanyWithName
565
566         public static void printCompanyList() {
567             //-----------------------------------------------------
568             //Prints the list of CompanyNodes referenced by companyHead.
569             //Specifically prints the name data field of CompanyNode.
570             //Preconditions: A list of CompanyNodes referenced by the
571             //class variable companyHead.
572             //Postconditions: The name data field of each CompanyNode
573             //present in the list is printed to the console.
574             //-----------------------------------------------------
575
576             CompanyNode cur = companyHead;
577             while(cur != null) {
578                 System.out.println(cur.name);
579                 cur = cur.next;
580             }
581         }//end printCompanyList
582
583         public static void addUnemployedCompany() {
584             //-----------------------------------------------------
585             //This method initializes and adds a CompanyNode initialized
586             //with the name Unemployed to act as a placeholder for
587             //unemployed persons.
588             //Postconditions: Class variable companyHead.
589             //Preconditions: A CompanyNode is created to hold the
590             //unemployed employees and added to the beginning of the
591             //list referenced by companyHead.
592             //-----------------------------------------------------
593
594             CompanyNode unemployed = new CompanyNode("Unemployed", null);
595             unemployed.next = companyHead;
596             companyHead = unemployed;
597         }//end addUnemployedCompany
598
599         public static void addToUnemployed(EmployeeNode employee) {
600             //-----------------------------------------------------
601             //This method takes an EmployeeNode employee passed as a
602             //parameter and adds it to the unemployed company.
603             //Note that the unemployed company is represented by
604             //companyHead since the unemployed company is always at the
605             //beginning of the CompanyNode list. The method sets
```

```
606             //employee.next to null to prevent issues when the employee
607             //is inserted into the unemployed CompanyNode.
608             //Postconditions:
609             //Preconditions:
610             //-------------------------------------------------------
611
612             employee.next = null;
613             insertEmployeeInCompany(companyHead, employee);
614         }//end addToUnemployed
615
616         public static void insertEmployeeInCompany(CompanyNode company, EmployeeNode employee)
{
617             //-------------------------------------------------------
618             //This method takes a CompanyNode and an EmployeeNode as a
619             //parameter. It inserts the EmployeeNode at the beginning
620              //of the EmployeeNode list contained in the CompanyNode.
621             //Preconditions: CompanyNode and EmployeeNode passed as
622             //parameters. Both must be initialized.
623             //Postconditions: The EmployeeNode passed is inserted into
624             //the beginning of the employee list contained in the
625             //CompanyNode.
626             //-------------------------------------------------------
627
628             //Add the employee to the beginning of the list to easily
629             //maintain a seniority system
630             if(company.employeeListHead == null)
631                 company.employeeListHead = employee;
632             else {
633                 employee.next = company.employeeListHead;
634                 company.employeeListHead = employee;
635             }
636         }//end insertEmployeeInCompany
637
638         public static EmployeeNode deleteAndReturnEmployeeWithName(String employeeName) {
639             //-------------------------------------------------------
640             //This method searches each list within each CompanyNode
641             //contained in the list referenced by companyHead. If an
642             //EmployeeNode with a name matching the string employeeName
643             //(passed to this method as a parameter) is found, the
644             //matching EmployeeNode is removed from the list it is
645             //contained in and returned to the caller.
646             //Preconditions: Class variable companyHead referencing a
647             //list of CompanyNodes.
648             //Postconditions: The EmployeeNode with a name matching
649             //employee name is removed from its current list and
650             //a reference to the matching node is returned. If a
```

```
651             //matching EmployeeNode is not found, null is returned.
652             //-----------------------------------------------------
653
654             CompanyNode currentCompany = companyHead;
655
656             boolean found = false;
657
658             while(currentCompany != null && found == false) {
659                 EmployeeNode currentEmployee = currentCompany.employeeListHead;
660                 EmployeeNode prev = currentEmployee;
661
662                 if(currentEmployee != null && currentEmployee.name.matches(employeeName)) {
663                     //Delete first employee if it matches or the list is empty.
664                     currentCompany.employeeListHead =
currentCompany.employeeListHead.next;
665
666                     //Set to null for insertion into unemployed
667                     currentEmployee.next = null;
668                     found = true;

669                     return currentEmployee;
670                 }
671                 while(currentEmployee != null && found == false) {
672                     if(currentEmployee.name.matches(employeeName)) {
673                         prev.next = currentEmployee.next;
674
675                         found = true;
676                         return currentEmployee;
677                     }
678                     prev = currentEmployee;
679                     currentEmployee = currentEmployee.next;
680                 }//end while
681
682                 currentCompany = currentCompany.next;
683             }//end while
684
685             System.out.println("Could not find employee with name " + employeeName + ".");
686             return null;
687         }//deleteAndReturnEmployeeWithName
688
689         public static void printEmployeesInCompany(CompanyNode company) {
690             //------------------------------------------------------------
691             //This method prints the EmployeeNode objects present in the
692             //in the EmployeeNode list contained in the CompanyNode company
693             //parameter passed. The each  Employee's name and amount earned
694             //is displayed below the name of the company.
```

```
695                 //Preconditions: CompanyNode passed by calling method.
696                 //Postconditions: The company name followed by each employee's
697                 //-------------------------------------------------------
698                 EmployeeNode currentEmployee = company.employeeListHead;
699
700                 if(currentEmployee != null)
701                         System.out.println("Employees in " + company.name + ": ");
702                 else
703                         System.out.println("There aren't any employees in " + company.name +
"."");
704                 while(currentEmployee != null) {
705                         System.out.println("\tEmployee: " + currentEmployee.name);
706                         System.out.printf("\t\tAmount Earned: $%.2f\n", currentEmployee.amountEarned);
707                         currentEmployee = currentEmployee.next;
708                 }
709         }//end printEmployeesInCompany
710
711 }//end p3


712 class CompanyNode {
713         String name;
714         CompanyNode next;
715
716         EmployeeNode employeeListHead;

717         public CompanyNode() {
718                 //Sets CompanyNode data fields to default values.
719                 this.name = "";
720                 this.next = null;
721         }
722
723         public CompanyNode(String name, CompanyNode next) {
724                 //Sets data fields to values passed as parameters.
725                 this.name = name;
726                 this.next = next;
727         }
728
729 }//end CompanyNode

730 class EmployeeNode {
731         String name;
732         EmployeeNode next;
733         double amountEarned;
734
735         public EmployeeNode() {
```

```
   736                  //Sets EmployeeNode data fields to default values.
   737                  this.name = "";
   738                  this.next = null;
   739                  amountEarned = 0.0;
   740          }
   741
   742          public EmployeeNode(String name, EmployeeNode next) {
   743                  //Sets data fields to values passed as parameters.
   744                  this.name = name;
   745                  this.next = next;
   746                  amountEarned = 0.0;
   747          }
   748  }//end EmployeeNode


:                       #null command
:
:
javac p3.java           #compile the java file
java p3                 #execute the file from the current directory
*********************************************
Employees in Digital:
        Employee: Harvey
                Amount Earned: $1000.00
        Employee: John
                Amount Earned: $2000.00
*********************************************
*********************************************
Employees in IBM:
        Employee: Susan
                Amount Earned: $1000.00
        Employee: Phil
                Amount Earned: $3000.00
*********************************************
*********************************************
Employees in Digital:
        Employee: Joshua
                Amount Earned: $1000.00
        Employee: Sam
                Amount Earned: $2000.00
        Employee: Harvey
                Amount Earned: $4000.00
        Employee: John
                Amount Earned: $6000.00
*********************************************
*********************************************
```

Employees in NEC:
Employee: Max
Amount Earned: $1000.00
Employee: George
Amount Earned: $2000.00
Employee: Fred
Amount Earned: $4000.00
****
****
Employees in Unemployed:
Employee: David
Amount Earned: $6050.00
Employee: Mario
Amount Earned: $3050.00
Employee: John
Amount Earned: $6100.00
****
****
Employees in IBM:
Employee: Tim
Amount Earned: $3000.00
Employee: Susan
Amount Earned: $6000.00
Employee: Marge
Amount Earned: $6000.00
Employee: Phil
Amount Earned: $11000.00
****
****
There aren't any employees in XEROX.
****
****
Employees in Borland:
Employee: Bob
Amount Earned: $3000.00
Employee: Miriam
Amount Earned: $8000.00

Employees in Compaq:
Employee: Phil
Amount Earned: $11000.00
Employee: Ralph
Amount Earned: $0.00
Employee: John
Amount Earned: $6100.00

Employees in Digital:
    Employee: Joshua
        Amount Earned: $3000.00
    Employee: Harvey
        Amount Earned: $8000.00
    Employee: Sam
        Amount Earned: $8000.00

Employees in IBM:
    Employee: Tim
        Amount Earned: $3000.00
    Employee: Susan
        Amount Earned: $6000.00
    Employee: Marge
        Amount Earned: $6000.00

Employees in Microsoft:
    Employee: Lesley
        Amount Earned: $3000.00
    Employee: Sharon
        Amount Earned: $7000.00

Employees in NEC:
    Employee: George
        Amount Earned: $4000.00
    Employee: Max
        Amount Earned: $5000.00
    Employee: Fred
        Amount Earned: $10000.00

There aren't any employees in XEROX.

Employees in Unemployed:
    Employee: David
        Amount Earned: $6050.00
    Employee: Mario
        Amount Earned: $3050.00
****
****
Employees in Compaq:
    Employee: Susan
        Amount Earned: $7000.00
    Employee: Fred
        Amount Earned: $12000.00
    Employee: Miriam

Amount Earned: $11000.00
Employee: Marge
Amount Earned: $10000.00
Employee: Phil
Amount Earned: $16000.00
Employee: Ralph
Amount Earned: $6000.00
Employee: John
Amount Earned: $13100.00
****
****
Employees in XEROX:
Employee: David
Amount Earned: $6100.00
Employee: Mario
Amount Earned: $3100.00
****
****
There are currently no unemployed workers.
****
****
Employees in Borland:
Employee: Bob
Amount Earned: $6000.00

Employees in Compaq:
Employee: Joshua
Amount Earned: $6000.00
Employee: Susan
Amount Earned: $11000.00
Employee: Miriam
Amount Earned: $17000.00
Employee: Tim
Amount Earned: $11050.00
Employee: Fred
Amount Earned: $22000.00
Employee: Marge
Amount Earned: $22000.00
Employee: Phil
Amount Earned: $30000.00
Employee: John
Amount Earned: $30100.00
Employee: Ralph
Amount Earned: $23000.00

Employees in Digital:

```
        Employee: Laszlo
                Amount Earned: $2000.00
        Employee: Harvey
                Amount Earned: $14000.00
        Employee: Sam
                Amount Earned: $17000.00

There aren't any employees in IBM.

Employees in Microsoft:
        Employee: Lesley
                Amount Earned: $6000.00
        Employee: Sharon
                Amount Earned: $13000.00

Employees in NEC:
        Employee: George
                Amount Earned: $7000.00
        Employee: Max
                Amount Earned: $11000.00

Employees in XEROX:
        Employee: David
                Amount Earned: $8100.00
        Employee: Mario
                Amount Earned: $7100.00


There aren't any employees in Unemployed.
**********************************************
:
:
:
date                    #print the date
Wed Oct  2 03:30:33 CDT 2013
[cs241114@cs ~]$ exit
exit

Script done on Wed 02 Oct 2013 03:30:37 AM CDT
```