```
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.
```

```
VARIABLE DICTIONARY:
  companyHead - CompanyNode, the first CompanyNode in the list.
                        The first node corresponds to the Unemployed company

ADTs:
  List

FILES USED:
  p3company.dat - a file containing company names
  p3commands.dat - a file containing command to manipulate employees

Verification:
        Turned in Excel spreadsheet.

------------------------------------------------------------------
*/

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

public class p3 {

        //pointer to the first node in the company list.
        static CompanyNode companyHead;

        public static void main(String[] args) {
                //Read in the companies and sort the into a list in
                //alphabetical order.
                populateCompanyList(new File("../instr/p3company.dat"));

                //Add a CompanyNode to the beginning of the list of
                //companies to be used as a placeholder for unemployed
                //persons.
                addUnemployedCompany();

                runCommandsFromFile(new File("../instr/p3command.dat"));
        }//end main

        /***Input Methods***/
        public static void runCommandsFromFile(File file) {
                //-----------------------------------------------------
                //Reads commands as strings from the file passed. Passes
                //the command to parseCommand() to be parsed as the
                //commands are read.
                //Preconditions: File object passed as parameter.
                //Postconditions: Commands are read as lines from file and
```

```java
		//passed to parseCommand() until the entire file is read
		//or parseCommand returns true (signals end).
		//--------------------------------------------------------

		try {
			Scanner fileReader = new Scanner(file);
			boolean stopReading = false;

			while(fileReader.hasNextLine() && stopReading == false) {
				stopReading = parseCommand(fileReader.nextLine());
			}//end while
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			System.out.println("Issue ocurred with commands file.");
		}//end catch
	}//end runCommandsFromFile

	public static boolean parseCommand(String commandLine) {
		//------------------------------------------------------------
		//This method takes a string parameter commandLine, splits
		//it into an array commands[], and calls a method corresponding
		//to commands[0]. If the command called requires parameters,
		//the other indices of commands (the parameters of the command)
		//are passed.
		//Preconditions: commandLine is a nonempty string.
		//Postconditions: a method corresponding to the first word in
		//commandLine is called with appropriate parameters. If the
		//command is not found, the program notifies the user.
		//------------------------------------------------------------

		boolean shouldEnd = false;

		String[] commands = commandLine.split(" ");


		String command = commands[0];
		command = command.toUpperCase();
		if(command.matches("JOIN")) {
			joinCommand(commands[1], commands[2]);
		} else if(command.matches("QUIT")) {
			quitCommand(commands[1]);
		} else if(command.matches("CHANGE")) {
			changeCommand(commands[1], commands[2]);
		} else if(command.matches("PROMOTE")) {
			promoteCommand(commands[1]);
		} else if(command.matches("DEMOTE")) {
```

```
            demoteCommand(commands[1]);
        } else if(command.matches("PAYDAY")) {
            paydayCommand();
        } else if(command.matches("EMPLOYEES")) {
            System.out.println("*********************************************");
            employeesCommand(commands[1]);
            System.out.println("*********************************************");
        } else if(command.matches("UNEMPLOYED")) {
            System.out.println("*********************************************");
            unemployedCommand();
            System.out.println("*********************************************");
        } else if(command.matches("DUMP")) {
            System.out.println("*********************************************");
            dumpCommand();
            System.out.println("*********************************************");
        } else if(command.matches("END")) {
            shouldEnd = true;
        } else {
            System.out.println("Unrecognized command.");
        }

        return shouldEnd;
    }//end parse command

    public static void joinCommand(String employeeName, String companyName) {
        //-------------------------------------------------------
        //Takes two strings, employeeName and companyName, as
        //parameters. Uses the methods findCompanyWithName() to get
        //a reference to the CompanyNode with a name matching
        //companyName. If a matching company is found, Unemployed
        //is checked to see if the employee is already created or
        //if a new EmployeeNode should be created with the
        //employeeName passed. The EmployeeNode is then inserted
        //into the company using insertEmployeeInCompany().
        //Preconditions: findCompanyWithName(String companyName),
        //searchUnemployedForEmployeeWithName(String employeeName),
        // and insertEmployeeInCompany(CompanyNode company,
        // EmployeeNode employee) methods defined in class.
        //Postconditions: If a matching company is found, an
        //EmployeeNode is either found in the currently
        //unemployed employees, or created using the name passed.
        //The EmployeeNode is then inserted into the list of
        //employees contained in the company.
        //-------------------------------------------------------

        EmployeeNode newEmployee = null;
```

```
		CompanyNode company = findCompanyWithName(companyName);
		if(company != null) {
			//Check to see if employee is currently unemployed or a new employee
			newEmployee = searchUnemployedForEmployeeWithName(employeeName);
			if(newEmployee != null)
				newEmployee.next = null;
			else
				newEmployee = new EmployeeNode(employeeName, null);

			insertEmployeeInCompany(company, newEmployee);
		}
		else
			System.out.println("Company not found " + companyName + ".");
	}

	public static void quitCommand(String employeeName) {
		//-----------------------------------------------------------
		//This method removes the EmployeeNode matching the string
		//employeeName passed as a parameter from the current
		//company containing it and adds it to the unemployed
		//company.
		//Preconditions: companyHead class variable referencing
		//list of companies. Method deleteAndReturnEmployeeWithName().
		//Postconditions: The EmployeeNode matching employee name is
		//removed from the current company containing it and added
		//to the unemployed company node (companyHead).
		//-----------------------------------------------------------

		EmployeeNode employee = deleteAndReturnEmployeeWithName(employeeName);

		if(employee != null)
			addToUnemployed(employee);
	}//quit

	public static void changeCommand(String employeeName, String companyName) {
		//-----------------------------------------------------------
		//Takes two strings, employeeName and companyName, as
		//parameters. Uses the method deleteAndReturnEmployeeWithName()
		//to remove the EmployeeNode with a name matching employeeName
		//from its current list and return it. Then uses
		//the method findCompanyWithName() to get a reference to the
		//CompanyNode with a name matching companyName. Finally
		//the method insertEmployeeInCompany() is used to insert
		//the employee in the company.
		//Postconditions: Strings employeeName and companyName passed
		//as parameters. The methods described above must be defined.
```

```
        //Preconditions: The EmployeeNode with a name matching
        //employeeName is removed from its current list and inserted
        //into the CompanyNode employee list for the company whose name
        //matches companyName.
        //---------------------------------------------------------

        EmployeeNode employee = deleteAndReturnEmployeeWithName(employeeName);
        CompanyNode company = findCompanyWithName(companyName);

        employee.next = null;
        insertEmployeeInCompany(company, employee);
    }//change

    public static void promoteCommand(String employeeName) {
        //-------------------------------------------------------
        //Finds an EmployeeNode matching the string employeeName
        //and moves the node up in the list if it is not already
        //the last node.
        //Preconditions: Class variable companyHead. String
        //employeeName passed as string.
        //Postconditions: The EmployeeNode matching employeeName is
        //moved up in the list and the references of the nodes in
        //the list are altered accordingly.If the EmployeeNode was
        //already last, no change occurs.
        //-------------------------------------------------------

        CompanyNode currentCompany = companyHead;

        boolean found = false;

        while(currentCompany != null && found == false) {

            EmployeeNode currentEmployee = currentCompany.employeeListHead;
            EmployeeNode prev = currentEmployee;

            if(currentEmployee != null && currentEmployee.name.matches(employeeName)
                    && currentEmployee.next != null) {
                currentCompany.employeeListHead = currentEmployee.next;
                currentEmployee.next = currentCompany.employeeListHead.next;
                currentCompany.employeeListHead.next = currentEmployee;
                found = true;
            }
            while(currentEmployee != null && currentEmployee.next != null
                    && found == false) {
                if(currentEmployee.name.matches(employeeName)) {
                    EmployeeNode curNext = currentEmployee.next;
```

```
					currentEmployee.next = curNext.next;
					prev.next = curNext;
					curNext.next = currentEmployee;
					found = true;
				}
				prev = currentEmployee;
				currentEmployee = currentEmployee.next;

			}//end while

			currentCompany = currentCompany.next;
		}//end while

		//if(!found)
			//System.out.println("Could not find employee with name " + employeeName + ".");
	}//promote

	public static void demoteCommand(String employeeName) {
		//-------------------------------------------------------
		//Finds an EmployeeNode matching the string employeeName
		//and moves the node down in the list if it is not already
		//the first node.
		//Preconditions: Class variable companyHead. String
		//employeeName passed as string.
		//Postconditions: The EmployeeNode matching employeeName is
		//moved down in the list and the references of the nodes in
		//the list are altered accordingly.If the EmployeeNode was
		//already first, no change occurs.
		//-------------------------------------------------------

		CompanyNode currentCompany = companyHead;

		boolean found = false;

		while(currentCompany != null && found == false) {

			EmployeeNode currentEmployee = currentCompany.employeeListHead;
			EmployeeNode prev = currentEmployee;
			EmployeeNode prevPrev = null;

			if(currentEmployee != null && currentEmployee.name.matches(employeeName)) {
				found = true;
			}
			while(currentEmployee != null && found == false) {
				if(currentEmployee.name.matches(employeeName)) {
					if(prevPrev != null)
```

```
						prevPrev.next = currentEmployee;

					prev.next = currentEmployee.next;

					currentEmployee.next = prev;

					if(prevPrev == null)
						currentCompany.employeeListHead = currentEmployee;

					found = true;
				}

				if(prev != currentEmployee) {
					prevPrev = prev;
				}
				prev = currentEmployee;
				currentEmployee = currentEmployee.next;
			}//end while

			currentCompany = currentCompany.next;
		}//end while

		//if(!found)
			//System.out.println("Could not find employee with name " + employeeName + ".");
	}//demote

	public static void paydayCommand() {
		//----------------------------------------------------------
		//The companyList and the lists of employees are traversed.
		//As each employee list is traversed, a rank variable is
		//set to  1 and incremented as the new list is traversed.
		//If the company being traversed is the Unemployed company,
		//the amountEarned field of the EmployeeNode is incremented
		//by 50. If the company being traversed is not the Unemployed
		//company, the EmployeeNode amountEarned field is incremented
		//by (1000 * rank).
		//Preconditions: a class variable companyHead referencing
		//a CompanyNode.
		//Postconditions: The amountEarned field of every employee
		//is incremented by 50 if unemployed, or (rank * 1000) if
		//employed.
		//----------------------------------------------------------

		CompanyNode currentCompany = companyHead;

		while(currentCompany != null) {
				EmployeeNode currentEmployee = currentCompany.employeeListHead;

				double pay = 0;
				int rank = 1;
				while(currentEmployee != null) {

					//change pay rate based on if the employee
					//is unemployed or employed companyHead is the
					//unemployed company
					if(currentCompany == companyHead)
						pay = 50;
					else
						pay = 1000 * rank;

					currentEmployee.amountEarned += pay;
					rank++;
					currentEmployee = currentEmployee.next;
				}//end while

				currentCompany = currentCompany.next;
			}//end while

		}//payday

		public static void employeesCommand(String companyName) {
			//---------------------------------------------------------
			//This method takes a string parameter companyName, uses the
			//method findCompanyWithName() to return a reference to the
			//company with a name matching companyName, and prints the
			//employees in the company returned using the
			//printEmployeesInCompany() method if the company returned
			//is not equal to null.
			//Preconditions: A string companyName passed as a parameter.
			//methods findCompanyWithName() and printEmployeesInCompany()
			//defined in the class.
			//Postconditions: The employees present in the CompanyNode
			//with the name companyName are printed to the console. If
			//the company is not found, nothing is displayed.
			//---------------------------------------------------------

			CompanyNode company = findCompanyWithName(companyName);
			if(company != null)
				printEmployeesInCompany(company);
		}//employees

		public static void unemployedCommand() {
```

```
		//------------------------------------------------------
		//This method prints out all of the employees present
		//in the Unemployed company (companyHead).
		//Preconditions: Class variable companyHead referring to
		//the unemployed company.
		//Postconditions: The employees present in the Unemployed
		//company are displayed to the console. If there are no
		//employees, a message is displayed to the console.
		//------------------------------------------------------

		if(companyHead != null && companyHead.employeeListHead != null) {
			printEmployeesInCompany(companyHead);
		} else {
			System.out.println("There are currently no unemployed workers.");
		}
	}//unemployed

	public static EmployeeNode searchUnemployedForEmployeeWithName(String employeeName) {
		//---------------------------------------------------------
		//Searches for an EmployeeNode in the Unemployed company
		//with a name matching the employeeName string passed as a
		//parameter. If a matching employee is found, the object is
		//removed from the Unemployed company employee list and
		//returned.
		//Preconditions: Class variable companyHead referencing the
		//Unemployed company. String employeeName passed as parameter.
		//Postconditions: The EmployeeNode list referenced in the
		//Unemployed company is traversed. If an EmployeeNode with a
		//name matching employeeName is found, the EmployeeNode is
		//returned. If a match is not found, null is returned.
		//---------------------------------------------------------

		CompanyNode currentCompany = companyHead;
		boolean found = false;
			EmployeeNode currentEmployee = currentCompany.employeeListHead;
			EmployeeNode prev = currentEmployee;

			if(currentEmployee != null && currentEmployee.name.matches(employeeName)) {
				//Delete first employee if it matches or the list is empty.
				currentCompany.employeeListHead = currentCompany.employeeListHead.next;

				//Set to null for insertion into unemployed
				currentEmployee.next = null;
				found = true;
```

```
                return currentEmployee;
            }
            while(currentEmployee != null && found == false) {
                if(currentEmployee.name.matches(employeeName)) {
                    prev.next = currentEmployee.next;

                    found = true;
                    return currentEmployee;
                }
                prev = currentEmployee;
                currentEmployee = currentEmployee.next;
            }//end while

        return null;
        }//end searchUnemployedForEmployeeWithName

    public static void dumpCommand() {
        //-------------------------------------------------------
        //This method prints all the employees present in each
        //CompanyNode present in the list referenced by companyHead.
        //The employees in the unemployed company are printed last.
        //Preconditions: A list of CompanyNodes referenced by the
        //class variable companyHead.
        //Postconditions: The employees in each company are printed
        //to the console (employee name and amountEarned). The
        //employees in unemployed are printed last.
        //-------------------------------------------------------

        CompanyNode cur = companyHead.next;
        while(cur != null) {
            printEmployeesInCompany(cur);
            System.out.println();
            cur = cur.next;
        }

        System.out.println();
        printEmployeesInCompany(companyHead);
    }//dump

    public static void endCommand() {
        //------------------------------------------------------
        //Postconditions: The program is running.
        //Preconditions: The program is exited with status code 0.
        //------------------------------------------------------

        System.exit(0);
```

```
	}//end

	/********Node Methods********/
	public static void populateCompanyList(File companyFile) {
		//-------------------------------------------------------
		//Read in the company names from the source file and
		//create new CompanyNode objects based on the names
		//read. Add them to the companyList referenced by
		//companyHead using the insertCompanyInOrder() method.
		//Postconditions: File object passed to method containing
		//strings separated by lines.
		//Preconditions: CompanyNodes are created based on the
		//strings read from the file and inserted into the company
		//list.
		//-------------------------------------------------------

		try {
			Scanner fileReader = new Scanner(companyFile);

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

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

	public static void insertCompanyInOrder(CompanyNode newNode) {
		//--------------------------------------------------------
		//Takes a CompanyNode in as parameter and inserts it in the
		//list of CompanyNodes referenced by the class variable
		//in alphabetical order (A-Z) based on the name data field
		//in CompanyNode.
		//Preconditions: A list of CompanyNodes referenced by the
		//class data field companyHead. A CompanyNode newNode
		//passed as a parameter that is initialized with a value
		//for name (newNode != null && newNode.name != null).
		//Postconditions: The CompanyNode newNode is inserted into
		//the list referenced by companyHead in alphabetical order.
		//--------------------------------------------------------

		CompanyNode cur = companyHead;
```

```
        CompanyNode prev = cur;

        if(companyHead == null || newNode.name.compareTo(companyHead.name) < 0) {
            //insert at beginning
            newNode.next = cur;
            companyHead = newNode;
        } else {//insert in list
            boolean inserted = false;
            while(cur != null && inserted == false) {
                //insert at position
                if(newNode.name.compareTo(cur.name) < 0) {
                    //insert in list
                    prev.next = newNode;
                    newNode.next = cur;
                    inserted = true;
                }
                prev = cur;
                cur = cur.next;
            }//end while
            if(inserted == false) {
                //insert at end of list
                prev.next = newNode;
            }//end if

        }//end else

    }//end insertInOrder

    public static CompanyNode findCompanyWithName(String companyName) {
        //-------------------------------------------------------
        //Traverse the list of companies and search for one with a
        // name matching companyName. If a match is found, return
        //the reference to the CompanyNode with the matching name.
        //If a match is not found, the method returns null.
        //Postconditions: A class variable called companyHead,
        //referencing the first node CompanyNode in the list.
        //Preconditions: A CompanyNode reference with a name
        //matching companyName is returned if a match is found.
        //Null is returned if a match is not found.
        //-------------------------------------------------------

        CompanyNode cur = companyHead;
        while(cur != null) {
            if(cur.name.matches(companyName))
                return cur;
            cur = cur.next;
```

```
		}

		System.out.println("Could not find company with name " + companyName + ".");
		return null;
	}//findCompanyWithName

	public static void printCompanyList() {
		//-------------------------------------------------------
		//Prints the list of CompanyNodes referenced by companyHead.
		//Specifically prints the name data field of CompanyNode.
		//Preconditions: A list of CompanyNodes referenced by the
		//class variable companyHead.
		//Postconditions: The name data field of each CompanyNode
		//present in the list is printed to the console.
		//-------------------------------------------------------

		CompanyNode cur = companyHead;
		while(cur != null) {
			System.out.println(cur.name);
			cur = cur.next;
		}
	}//end printCompanyList

	public static void addUnemployedCompany() {
		//-------------------------------------------------------
		//This method initializes and adds a CompanyNode initialized
		//with the name Unemployed to act as a placeholder for
		//unemployed persons.
		//Postconditions: Class variable companyHead.
		//Preconditions: A CompanyNode is created to hold the
		//unemployed employees and added to the beginning of the
		//list referenced by companyHead.
		//-------------------------------------------------------

		CompanyNode unemployed = new CompanyNode("Unemployed", null);
		unemployed.next = companyHead;
		companyHead = unemployed;
	}//end addUnemployedCompany

	public static void addToUnemployed(EmployeeNode employee) {
		//------------------------------------------------------
		//This method takes an EmployeeNode employee passed as a
		//parameter and adds it to the unemployed company.
		//Note that the unemployed company is represented by
		//companyHead since the unemployed company is always at the
		//beginning of the CompanyNode list. The method sets
		//employee.next to null to prevent issues when the employee
		//is inserted into the unemployed CompanyNode.
		//Postconditions:
		//Preconditions:
		//-------------------------------------------------------

		employee.next = null;
		insertEmployeeInCompany(companyHead, employee);
	}//end addToUnemployed

	public static void insertEmployeeInCompany(CompanyNode company, EmployeeNode employee)
{
		//-------------------------------------------------------
		//This method takes a CompanyNode and an EmployeeNode as a
		//parameter. It inserts the EmployeeNode at the beginning
		 //of the EmployeeNode list contained in the CompanyNode.
		//Preconditions: CompanyNode and EmployeeNode passed as
		//parameters. Both must be initialized.
		//Postconditions: The EmployeeNode passed is inserted into
		//the beginning of the employee list contained in the
		//CompanyNode.
		//-------------------------------------------------------

		//Add the employee to the beginning of the list to easily
		//maintain a seniority system
		if(company.employeeListHead == null)
			company.employeeListHead = employee;
		else {
			employee.next = company.employeeListHead;
			company.employeeListHead = employee;
		}
	}//end insertEmployeeInCompany

	public static EmployeeNode deleteAndReturnEmployeeWithName(String employeeName) {
		//-------------------------------------------------------
		//This method searches each list within each CompanyNode
		//contained in the list referenced by companyHead. If an
		//EmployeeNode with a name matching the string employeeName
		//(passed to this method as a parameter) is found, the
		//matching EmployeeNode is removed from the list it is
		//contained in and returned to the caller.
		//Preconditions: Class variable companyHead referencing a
		//list of CompanyNodes.
		//Postconditions: The EmployeeNode with a name matching
		//employee name is removed from its current list and
		//a reference to the matching node is returned. If a
		//matching EmployeeNode is not found, null is returned.
		//-------------------------------------------------------

		CompanyNode currentCompany = companyHead;

		boolean found = false;

		while(currentCompany != null && found == false) {
			EmployeeNode currentEmployee = currentCompany.employeeListHead;
			EmployeeNode prev = currentEmployee;

			if(currentEmployee != null && currentEmployee.name.matches(employeeName)) {
				//Delete first employee if it matches or the list is empty.
				currentCompany.employeeListHead = 
currentCompany.employeeListHead.next;

				//Set to null for insertion into unemployed
				currentEmployee.next = null;
				found = true;

				return currentEmployee;
			}
			while(currentEmployee != null && found == false) {
				if(currentEmployee.name.matches(employeeName)) {
					prev.next = currentEmployee.next;

					found = true;
					return currentEmployee;
				}
				prev = currentEmployee;
				currentEmployee = currentEmployee.next;
			}//end while

			currentCompany = currentCompany.next;
		}//end while

		System.out.println("Could not find employee with name " + employeeName + ".");
		return null;
	}//deleteAndReturnEmployeeWithName

	public static void printEmployeesInCompany(CompanyNode company) {
		//----------------------------------------------------------
		//This method prints the EmployeeNode objects present in the
		//in the EmployeeNode list contained in the CompanyNode company
		//parameter passed. The each  Employee's name and amount earned
		//is displayed below the name of the company.
		//Preconditions: CompanyNode passed by calling method.
		//Postconditions: The company name followed by each employee's
		//----------------------------------------------------------
		EmployeeNode currentEmployee = company.employeeListHead;

		if(currentEmployee != null)
			System.out.println("Employees in " + company.name + ": ");
		else
			System.out.println("There aren't any employees in " + company.name + ".");
		while(currentEmployee != null) {
			System.out.println("\tEmployee: " + currentEmployee.name);
			System.out.printf("\t\tAmount Earned: $%.2f\n", currentEmployee.amountEarned);
			currentEmployee = currentEmployee.next;
		}
	}//end printEmployeesInCompany

}//end p3


class CompanyNode {
	String name;
	CompanyNode next;

	EmployeeNode employeeListHead;

	public CompanyNode() {
		//Sets CompanyNode data fields to default values.
		this.name = "";
		this.next = null;
	}

	public CompanyNode(String name, CompanyNode next) {
		//Sets data fields to values passed as parameters.
		this.name = name;
		this.next = next;
	}

}//end CompanyNode

class EmployeeNode {
	String name;
	EmployeeNode next;
	double amountEarned;

	public EmployeeNode() {
```

```
                //Sets EmployeeNode data fields to default values.
                this.name = "";
                this.next = null;
                amountEarned = 0.0;
        }

        public EmployeeNode(String name, EmployeeNode next) {
                //Sets data fields to values passed as parameters.
                this.name = name;
                this.next = next;
                amountEarned = 0.0;
        }
}//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
```