**Assignment 6**

Josue Ponce

Montgomery College

4/20/2018

Author’s Note

This project report was prepared for CMSC 203 CRN #30672, taught by professor Ahmed Tarek

**Table of Contents**

Design Reflection…………………………………………………………………………………2

Hand Calculated Test Cases Wacky Widgets Company Application…………………………….4

GUI Test for Wacky Widgets Company Application…………………………………………….5

Junit Test for Wacky Widgets COMPANY Application………………………………………..13

CompanyTest.Java Copy of Code Used for Junit Test…………………………………….……14

Copy of Company.Java Code……………………………………………………………………19

Copy of Employee.Java Code……………………………………………………………………27

Copy of Sales.Java Code…………………………………………………………………………31

Copy of Design.Java Code……………………………………………………………………….33

Copy of Position.Java Code…………………………………………………...…………………36

Copy of Manufacturing.Java Code…………………………………………………………...….37

Copy of Manager.Java Code…………………………………………………………………….39

**Design Reflection**

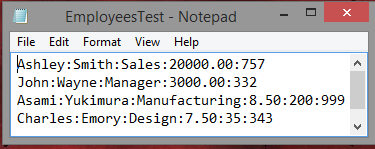
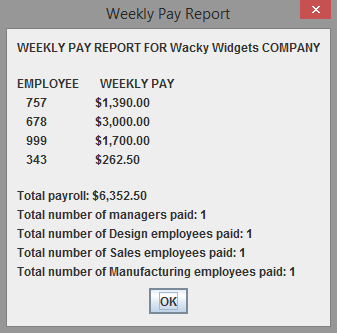
The initial design of the application had to be modified because the application during run-time displayed a lot of logical errors. Additionally, the add employee button in the application was not working as intended, so the application needed to be modified a bit to meet the requirements as well as getting everything to work correctly. In my initial design, there were missing methods that should have been added originally to the data management class (Company.java). When the add employee, button did not work on the application during run-time, I assumed there was something missing in company.java that needed to be implemented. I approached the issue by adding static methods such as int getNumCompanies(), public static void resetCompanyCount, and public static void updateCompanyNumber(). Once I implemented those methods into the application I noticed that the add employee button worked as intended. Additionally, there was no remove button in the application, so I analyzed the GUI code provided by the instructor and noticed that no remove button was implemented. I approached the issue by simply implementing a remove button in the GUI provided. Implementation of the remove button was essential to ensure that the removeEmployee() method worked as intended. After implementing the remove button, the application was tested again and after typed an employee’s name into the application. Then I hit the remove employee button and the application successfully deleted the employee from the array list. The button was tested several times to ensure that the application removed employees correctly. All the tests conducted on the remove employee button succeeded. Furthermore, logical errors were found in the application. The logical errors consisted of wrong currency formatting in total payroll and decimal numbers not being rounded up. I fixed the errors by simply going through all of the subclasses of employee.java and adding String.*format*("%,.2f", calculateWeeklyPay()) in the toString method. In addition, I initiated a variable to hold number formatting which was fmt = NumberFormat.getCurrencyInstance(Locale.US). Upon implementing number formating in both the calculateWeeklyPay method and generateWeeklyReport method the application started to display the desired results. Those were the only modifications added to the final design of the application. Everything else in the application worked as intended and met the requirements for the assignment, so no further modifications were needed.

**Hand Calculated Test Cases Wacky Widgets Company Application**

Hand calculated test cases were created to ensure that the application was free of logical errors and mathematically correct. A visual representation of all the hand calculated results can be found down below (Table 1). The table set up is like the application’s GUI. After running the application, the results matched the hand calculated results. A custom text file was also used for this test. A screenshot of the data from the custom text file (Figure 1) used can be found down below along with screenshots of the application’s output (Figure 2).

|  |  |
| --- | --- |
| Employee ID | Weekly pay |
| 757 | $1,390.00 |
| 698 | $3,000.00 |
| 999 | $1,700.00 |
| 343 | $262.50 |
| Total Payroll | $6,352.50 |
| Total number of Managers paid | 1 |
| Total number of Design employees paid | 1 |
| Total number of Sales employees paid | 1 |
| Total number of Manufacturing employees paid | 1 |

*Table 1.* The values under weekly pay were all mathematically hand calculated and the total payroll was rounded up by two decimal points. In addition, the number of employees for all positions were counted by hand.

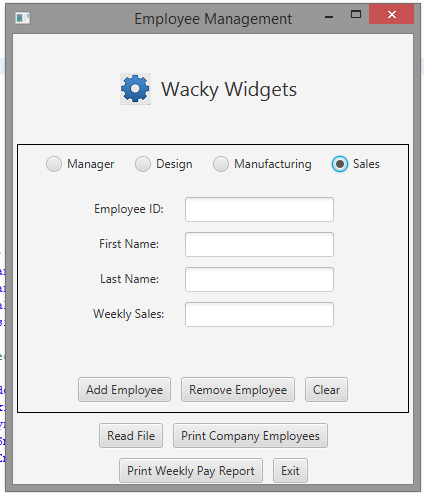
*Figure 1.* Custom text file used to test application. *Figure 2.* Sample Output of the program

**GUI Test for Wacky Widgets Company Application**

The application’s GUI was tested to ensure that the GUI for the application worked as intended and free of glitches or screen tearing. The tests conducted also ensured that the application was free of logical errors and displayed the correct results to the user during run-time. The tests made sure that the correct currency format was being displayed to the user. Additionally, the test ensured that the remove employee button that was implemented into the GUI worked as intended and removed an employee successfully without error. The test conducted on the application’s GUI passed because the remove employee button worked as intended without error. The GUI also displayed the results to the user with no logical or formatting errors. Screenshots of the application’s GUI test runs have been attached down below.



*Note.* GUI displaying correct labels and TextFields when manager is selected.



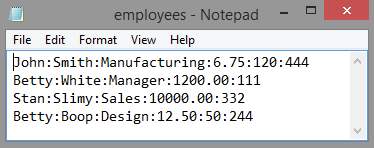
*Note.* GUI displaying correct labels and TextFields when sales is selected.



*Note.* GUI displaying correct labels and TextFields when Manufacturing is selected.



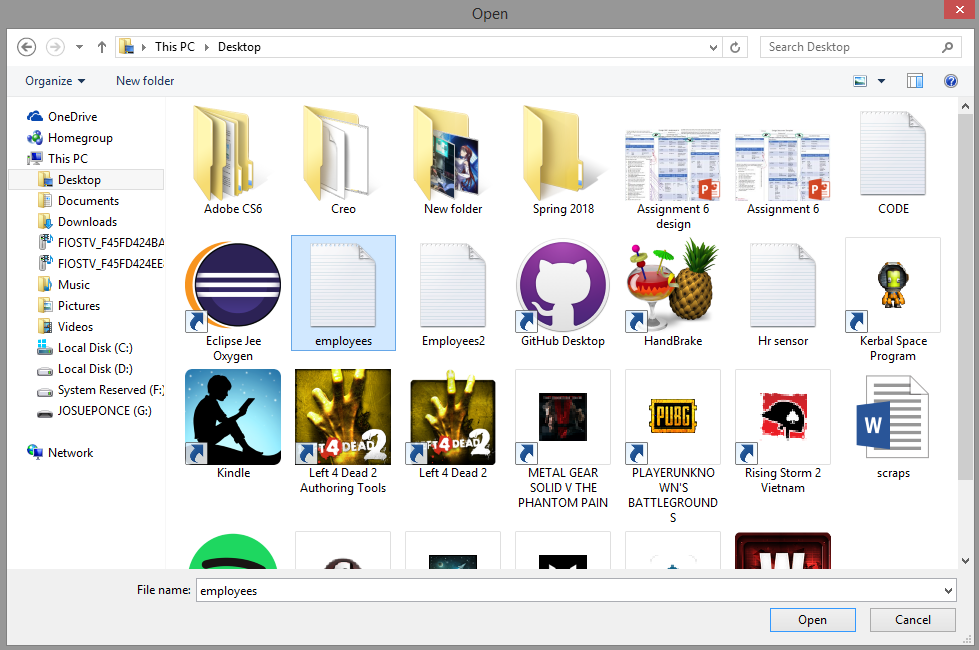
*Note.* GUI displaying correct labels and TextFields when design is selected.



*Note.* Original input file for test case.



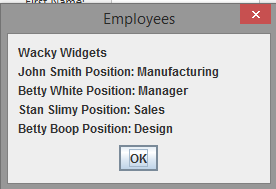
*Note.* Pressing the read file button.



*Note.* Inputting file into application



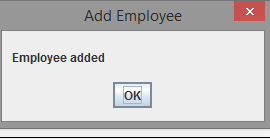
*Note.* GUI displaying a summary of weekly pay report.



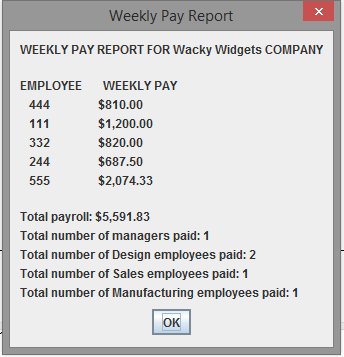
*Note.* GUI displaying employees.



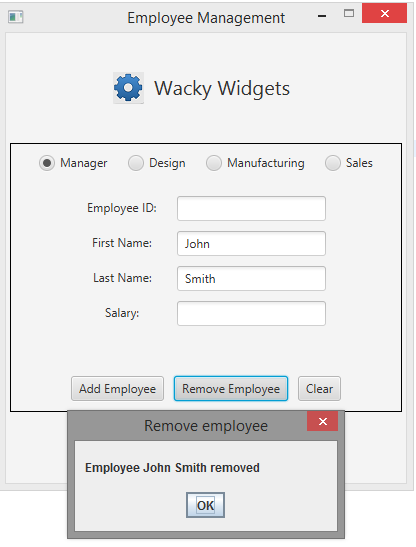
*Note.* Adding new employee through GUI.



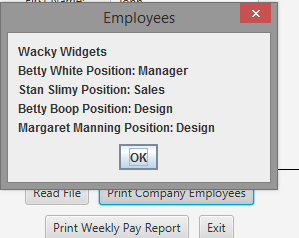
*Note.* Employee being successfully added through the GUI.



*Note.* GUI displaying summary pay report with new employee.



*Note.* Employee being removed correctly by entering the name into GUI.

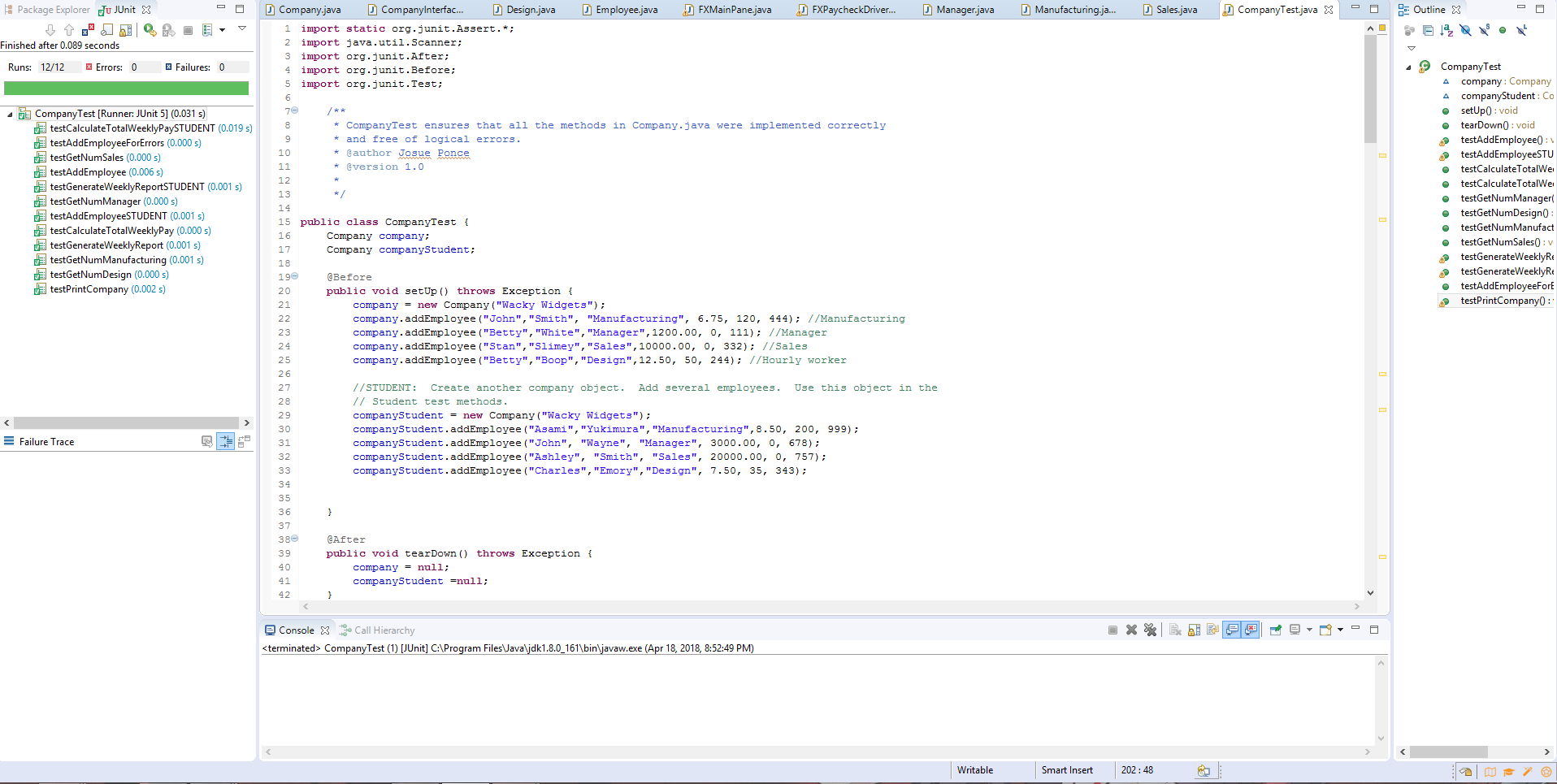


*Note.* Results after removing an employee correctly.

**Junit Test for Wacky Widgets COMPANY Application**

Junit tests were created to check the application’s functionality. This test ensures that all the methods implemented into the Company.Java work as intended. The test also ensures that the application is free of logical errors. Assertion methods were used to test the following methods in Company.Java, AddEmployee(), CalculateTotalWeeklyPay(), GetNumManager(), GetNumDesign(), NumManufacturing(), GetNumSales(), GenerateWeeklyReport(), AddEmployeeForErrors(Checks if the application recognizes that there is a max number of employees in a specific job position. Ex more than 2 designers should return error.), and PrintCompany(). After running the Junit tests, all the results returned green which means that the application is functioning as intended. The screenshot of the results from the Junit tests are down below (Figure 1).

---------------------------------------------------------------------------------------------------------------------



*Figure 1.* Junit tests returning positive results which indicates that the application is working as intended.

**CompanyTest.Java Copy of Code Used for Junit Test**

**import** **static** org.junit.Assert.\*;

**import** **java.util.Scanner**;

**import** **org.junit.After**;

**import** **org.junit.Before**;

**import** **org.junit.Test**;

/\*\*

\* CompanyTest ensures that all the methods in Company.java were implemented correctly

\* and free of logical errors.

\* @author Josue Ponce

\* @version 1.0

\*

\*/

**public** **class** **CompanyTest** {

Company company;

Company companyStudent;

**@Before**

**public** **void** **setUp**() **throws** Exception {

company = **new** Company("Wacky Widgets");

company.addEmployee("John","Smith", "Manufacturing", **6.75**, **120**, **444**); //Manufacturing

company.addEmployee("Betty","White","Manager",**1200.00**, **0**, **111**); //Manager

company.addEmployee("Stan","Slimey","Sales",**10000.00**, **0**, **332**); //Sales

company.addEmployee("Betty","Boop","Design",**12.50**, **50**, **244**); //Hourly worker

//STUDENT: Create another company object. Add several employees. Use this object in the

// Student test methods.

companyStudent = **new** Company("Wacky Widgets");

companyStudent.addEmployee("Asami","Yukimura","Manufacturing",**8.50**, **200**, **999**);

companyStudent.addEmployee("John", "Wayne", "Manager", **3000.00**, **0**, **678**);

companyStudent.addEmployee("Ashley", "Smith", "Sales", **20000.00**, **0**, **757**);

companyStudent.addEmployee("Charles","Emory","Design", **7.50**, **35**, **343**);

}

**@After**

**public** **void** **tearDown**() **throws** Exception {

company = **null**;

companyStudent =**null**;

}

**@Test**

**public** **void** **testAddEmployee**() {

String result = company.generateWeeklyReport();

Scanner report = **new** Scanner(result);

report.nextLine(); //WEEKLY PAY REPORT FOR WIDGET COMPANY

report.nextLine(); //empty line

report.nextLine(); //EMPLOYEE WEEKLY PAY

report.nextLine(); //employee 444

report.nextLine(); //employee 111

report.nextLine(); //employee 332

assertEquals("244",report.next()); //employee 244

company.addEmployee("Charles","Emory","Design", **7.50**, **35**, **343**); //Design employee

result = company.generateWeeklyReport();

report = **new** Scanner(result);

report.nextLine(); //WEEKLY PAY REPORT FOR Wacky Widget COMPANY

report.nextLine(); //empty line

report.nextLine(); //EMPLOYEE WEEKLY PAY

report.nextLine(); //employee 444

report.nextLine(); //employee 111

report.nextLine(); //employee 332

report.nextLine(); //employee 244

assertEquals("343",report.next()); //employee 343

assertEquals(**2**,company.getNumDesign());

}

**@Test**

**public** **void** **testAddEmployeeSTUDENT**() {

String result = companyStudent.generateWeeklyReport();

Scanner report = **new** Scanner(result);

report.nextLine(); //WEEKLY PAY REPORT FOR WIDGET COMPANY

report.nextLine(); //empty line

report.nextLine(); //EMPLOYEE WEEKLY PAY

report.nextLine(); //employee 999

report.nextLine(); //employee 678

report.nextLine(); //employee 757

assertEquals("343",report.next());

companyStudent.addEmployee("Conejo", "Slays", "Design", **8.50**, **45**, **256**);

result = companyStudent.generateWeeklyReport();

report = **new** Scanner(result);

report.nextLine(); //WEEKLY PAY REPORT FOR Wacky Widget COMPANY

report.nextLine(); //empty line

report.nextLine(); //EMPLOYEE WEEKLY PAY

report.nextLine(); //employee 999

report.nextLine(); //employee 678

report.nextLine(); //employee 757

report.nextLine(); //employee 343

assertEquals("256",report.next()); //employee 256

assertEquals(**2**,companyStudent.getNumDesign());

}

**@Test**

**public** **void** **testCalculateTotalWeeklyPay**() {

assertEquals(**3517.50**, company.calculateTotalWeeklyPay(), .**001**);

company.addEmployee("Charles","Emory","Design", **7.50**, **35**, **343**); //Design employee

assertEquals(**3780.00**, company.calculateTotalWeeklyPay(), .**001**);

}

**@Test**

**public** **void** **testCalculateTotalWeeklyPaySTUDENT**()

{

assertEquals(**6352.50**, companyStudent.calculateTotalWeeklyPay(), .**001**);

companyStudent.addEmployee("Conejo", "Slays", "Design", **8.50**, **45**, **256**);

assertEquals(**6756.25**, companyStudent.calculateTotalWeeklyPay(), .**001**);

}

**@Test**

**public** **void** **testGetNumManager**() {

assertEquals(**1**, company.getNumManager());

}

**@Test**

**public** **void** **testGetNumDesign**() {

assertEquals(**1**, company.getNumDesign());

}

**@Test**

**public** **void** **testGetNumManufacturing**() {

assertEquals(**1**, company.getNumManufacturing());

}

**@Test**

**public** **void** **testGetNumSales**() {

assertEquals(**1**, company.getNumSales());

}

**@Test**

**public** **void** **testGenerateWeeklyReport**() {

String result = company.generateWeeklyReport();

Scanner report = **new** Scanner(result);

report.nextLine(); //WEEKLY PAY REPORT FOR Wacky Widget COMPANY

report.nextLine(); //empty line

report.nextLine(); //EMPLOYEE WEEKLY PAY

assertEquals("444",report.next()); //444

assertEquals("$810.00",report.next()); //$1,200.00

report.nextLine();//extracts newline at end of line

assertEquals("111",report.next()); //22222

}

**@Test**

**public** **void** **testGenerateWeeklyReportSTUDENT**() {

String result = companyStudent.generateWeeklyReport();

Scanner report = **new** Scanner(result);

report.nextLine(); //WEEKLY PAY REPORT FOR Wacky Widget COMPANY

report.nextLine();

report.nextLine(); //EMPLOYEE WEEKLY PAY

assertEquals("999",report.next()); //757

assertEquals("$1,700.00",report.next()); //$1,200.00

report.nextLine();//extracts newline at end of line

assertEquals("678",report.next()); //22222

}

**@Test**

/\*\*

\* Test to

\* 1. add 3 new Employees as a manufacturing person

\* check if recognizes there are already 4 manufacturing persons

\* 2. add a new employee as a sales person

\* check if recognizes there is already a a sales person

\* 3. add 2 new employee as a design person

\* check if recognizes there is already 2 design persons

\*/

**public** **void** **testAddEmployeeForErrors**() {

String result;

//add another designer - No problem, should return null

result = company.addEmployee("Bobby", "Match", "Design",**200**,**35**,**333**);

assertEquals(**null**, result);

//add another designer - already 2 designers - return error message

result = company.addEmployee("Albert","Pine", "Design",**200**,**35**,**777**);

assertEquals("There are already two design persons\nEmployee not added", result);

//add another sales person - already 1 sales person - return error message

result = company.addEmployee("Susie", "Smith", "Sales",**4000**,**0**,**888**);

assertEquals("There is already a sales person\nEmployee not added", result);

//add another manufacturer - No problem, should return null

result = company.addEmployee("Benedict", "Cumberbatch", "Manufacturing",.**35**,**2500**,**999**);

assertEquals(**null**, result);

//add another manufacturer - No problem, should return null

result = company.addEmployee("Martin", "Freeman", "Manufacturing",.**37**,**1500**,**543**);

assertEquals(**null**, result);

//add another manufacturer - No problem, should return null

result = company.addEmployee("Fred", "Flintstone", "Manufacturing",.**37**,**1500**,**389**);

assertEquals(**null**, result);

//add another manufacturer - already 4 manufacturers - return error message

result = company.addEmployee("Andrew", "Scott", "Manufacturing",.**52**,**1000**,**765**);

assertEquals("There are already four manufacturing persons\nEmployee not added", result);

}

**@Test**

/\*\*

\* Test to:

\* 1. Check if the company name is on the first line

\* 2. Check if John is on the second line

\* 3. Check if Betty is on the third line

\* 4. Check if Stan is on the fourth line

\* 5. Check if Betty is on the fifth line

\*/

**public** **void** **testPrintCompany**() {

String result = company.printCompany();

Scanner company = **new** Scanner(result);

assertEquals("Wacky Widgets",company.nextLine());

//extract three Employees

assertEquals("John", company.next());

company.nextLine(); //Smith Position manufacturing (rest of line)

assertEquals("Betty", company.next());

company.nextLine(); //White Position manager (rest of line)

assertEquals("Stan",company.next());

company.nextLine(); //Slimey Position Sales (rest of line);

assertEquals("Betty",company.next());

}

}

**Copy of Company.Java Code**

**import** **java.util.ArrayList**;

**import** **java.text.NumberFormat**;

**import** **java.util.\***;

/\*\*

\* Company class implements CompanyInterface. Contains arraylist which holds references to

\* Employee data element class objects to be inherited from employee.

\* @author Josue Ponce

\* @version 1.0

\*

\*/

**public** **class** **Company** **implements** CompanyInterface {

/\*\* Attribute companyName holds the name of the company.\*/

**private** String companyName;

/\*\* Attribute employees holds objects from the data element class.\*/

**private** ArrayList < Employee > employees;

/\*\* Attribute numberOfCompanies holds the total number of companies set to 0.\*/

**private** **static** **int** numberOfCompanies = **0**;

/\*\* Attribute numDesign holds the total number of designers.\*/

**private** **int** numDesign;

/\*\* Attribute numEmployees holds the total number of employees.\*/

**private** **int** numEmployees;

/\*\* Attribute numManufacturing holds the total number of employees working in manufacturing.\*/

**private** **int** numManufacturing;

/\*\* Attribute numSales holds the total number of employees who work in sales.\*/

**private** **int** numSales;

/\*\* Attribute numManager holds the total number of managers who manage the company.\*/

**private** **int** numManager;

/\*\* Attribute fmt holds proper currency formatting representing U.S dollars.\*/

**private** NumberFormat fmt;

/\*\*

\* Parameterized class constructor that initiates arg.

\* @param arg - Represents the company name.

\*/

**public** **Company**(String arg )

{

// Instantiate fields

**this**.companyName = arg;

**this**.employees = **new** ArrayList < Employee > ();

**this**.numSales = **0**;

**this**.numEmployees = **0**;

**this**.numDesign = **0**;

**this**.numManufacturing = **0**;

**this**.numManager = **0**;

updateCompanyNumber();

**this**.fmt = NumberFormat.getCurrencyInstance(Locale.US);

}

/\*\*

\* Adds an employee to the ArrayList.

\* @param type - Employee's position: Manager, Design, Sales, Manufacturing

\* @param fname - Employee's first name.

\* @param lname - Employee's last name.

\* @param firstParam - Manager-salary, Design-hourly rate, Sales-weekly sales, Manufacturing-rate per piece

\* @param secondParam - Manager-not needed (0), Design-number of hours, Sales-not needed (0), Manufacturing-number of pieces

\* @param empNum - Employee's ID number

\* @return null if successful add. Returns a string that describes the reason

\* for not adding the employee.

\*/

**public** String **addEmployee**(String fname, String lname, String type, **double** firstParam, **int** secondParam, **int** empNum)

{

**if** (numSales == **1** && type.equals("Sales")) {

**return** "There is already a sales person\nEmployee not added";

} **else** **if** (numDesign == **2** && type == "Design")

**return** "There are already two design persons\nEmployee not added";

**else** **if** (numManufacturing == **4** && type.equals("Manufacturing"))

**return** "There are already four manufacturing persons\nEmployee not added";

**else** **if** (numManager == **1** && type.equals("Manager"))

**return** "There is already a manager\nEmployee not added";

**else** **if** (**this**.getNumEmployees() == **8**)

**return** "There are already 8 employees\nEmployee not added";

**else** **if** (type.equals("Sales") == **false** && type.equals("Design") == **false** && type.equals("Manufacturing") == **false**

&&

type.equals("Manager") == **false**)

**return** "That position does not exist";

**else** {

**if** (type.equals("Sales")) {

Sales s = **new** Sales(fname, lname, empNum, firstParam);

employees.add(s);

numSales++;

} **else** **if** (type.equals("Design")) {

Design d = **new** Design(fname, lname, empNum, firstParam, secondParam);

employees.add(d);

numDesign++;

} **else** **if** (type.equals("Manufacturing")) {

Manufacturing m = **new** Manufacturing(fname, lname, secondParam, firstParam, empNum);

employees.add(m);

numManufacturing++;

} **else** {

Manager m = **new** Manager(fname, lname, empNum, firstParam);

employees.add(m);

numManager++;

}

**return** **null**;

}

}

/\*\*

\* Returns the total number of companies.

\* @return numberOfCompanies - Returns the total number of companies.

\*/

**public** **static** **int** **getNumCompanies**()

{

**return** numberOfCompanies;

}

/\*\*

\* Updates the Company number and increments the number of companies by 1.

\*/

**private** **static** **void** **updateCompanyNumber**()

{

numberOfCompanies += **1**;

}

/\*\*

\* Resets the company count.

\*/

**public** **static** **void** **resetCompanyCount**()

{

numberOfCompanies = **0**;

}

/\*\*

\* Returns the number of employees

\* @return numEmployees - The total number of employee's working at the company.

\*/

**public** **int** **getNumEmployees**()

{

**return** numEmployees;

}

/\*\*

\* Returns the total number of managers.

\* @return numManager - The number of managers

\*/

**public** **int** **getNumManager**() {

**return** numManager;

}

/\*\*

\* Returns the total number of designers.

\* @return numDesign - Total number of designers.

\*/

**public** **int** **getNumDesign**() {

**return** numDesign;

}

/\*\*

\* Returns the total number of sales employees.

\* @return numSales - Total number of sales employees.

\*/

**public** **int** **getNumSales**() {

**return** numSales;

}

/\*\*

\* Returns the number of piece workers in the ArrayList

\* @return numManufacturing - Total number of piece workers.

\*/

**public** **int** **getNumManufacturing**() {

**return** numManufacturing;

}

/\*\*

\* Calculate the total weekly pay for all employees in the ArrayList

\* @return total - The total weekly pay for all employees

\*/

**public** **double** **calculateTotalWeeklyPay**() {

**double** total = **0**;

**for** (Employee **e:** employees) {

total += e.calculateWeeklyPay();

fmt.format(total);

}

**return** total;

}

/\*\*

\* Creates a string representation of the object which represents first name, last name, and the employee's position.

\* @return printEmp - String that represents all the objects in the ArrayList from employee.

\*/

**public** String **printCompany**()

{

String printEmp = companyName;

printEmp = **this**.companyName + "\n";

**for** (Employee **e:** employees) {

printEmp += e.getFirstName() + " " + e.getLastName() + " Position: " + e.getPos() + "\n";

}

**return** printEmp;

}

/\*\*

\* Generates a weekly report of employee weekly pay

\* @return report - String that contains the Weekly Report

\*/

**public** String **generateWeeklyReport**() {

String report = "WEEKLY PAY REPORT FOR " + companyName +

" COMPANY\n\nEMPLOYEE WEEKLY PAY\n" +

toString() + "\nTotal payroll: " + fmt.format(calculateTotalWeeklyPay()) +

"\nTotal number of managers paid: " + String.valueOf(numManager) +

"\nTotal number of Design employees paid: " + String.valueOf(numDesign) +

"\nTotal number of Sales employees paid: " + String.valueOf(numSales) +

"\nTotal number of Manufacturing employees paid: "

+

String.valueOf(numManufacturing);

**return** report;

}

/\*\*

\* Remove a specified employee from the ArrayList.

\* @param firstName First name of the employee to remove from the payroll

\* @param lastName Last name of the employee to remove from the payroll

\* @return true if the employee was successfully removed, false otherwise

\*/

**public** **boolean** **removeEmployee**(String firstName, String lastName) {

// Iterate over employee list

**for** (Employee **e:** employees) {

// IF first name and last name matches, remove employee

// return true

**if** (firstName.equals(e.getFirstName()) && lastName.equals(e.getLastName())) {

employees.remove(e);

**return** **true**;

}

}

**return** **false**;

}

/\*\*

\* Creates a string representation of the object

\* @return s - String that represents all the objects in the ArrayList from employee.

\*/

**@Override**

**public** String **toString**()

{

String s = "";

**for** (Employee **e:** employees) {

s += e.toString() + "\n";

}

**return** s;

}

}

**Copy of Employee.Java Code**

/\*\*

\* This program demonstrates use of inheritance and polymorphism.

\* Employee is an abstract based class defined with

\* 4 subclasses which are, manager, sales, design, and manufacturing.

\* @author Josue Ponce

\* @version 1.0

\*/

**public** **abstract** **class** **Employee**

{

/\*\*Attribute fName holds employee's first name \*/

**private** String fName;

/\*\*Attribute lName holds employee's last name \*/

**private** String lName;

/\*\*Attribute empNum holds employee's ID number \*/

**private** **int** empNum;

/\*\*Attribute p holds predefined constant based on the employee's position. \*/

**private** Position p;

/\*\*

\* Default Class constructor.

\*/

**public** **Employee**() {

**this**.fName = "";

**this**.lName = "";

**this**.empNum = -**1**;

**this**.p = **null**;

}

/\*\*

\* Parameterized Class constructor that initiates fName, lName, empNum, and p

\* @param fName - Holds the employee's first name.

\* @param lName - Holds the employee's last name.

\* @param empNum - Holds the employee's ID number.

\* @param p - Holds the employee's current position at the company.

\*/

**public** **Employee**(String fName, String lName, **int** empNum, Position p)

{

**this**.fName = fName;

**this**.lName = lName;

**this**.empNum = empNum;

**this**.p = p;

}

/\*\*

\* Setter method sets the employee's current position.

\* @param pos - Holds the employee's current position.

\*/

**public** **void** **setPos**(Position pos) {

**this**.p = pos;

}

/\*\*

\* Setter method sets the employee's first name.

\* @param fName - Holds the employee's first name.

\*/

**public** **void** **setFirstName**(String fName) {

**this**.fName = fName;

}

/\*\*

\* Setter method sets the employee's ID number.

\* @param empNum - Holds the employee's ID number.

\*/

**public** **void** **setEmpNum**(**int** empNum) {

**this**.empNum = empNum;

}

/\*\*

\* Setter method sets the employee's last name.

\* @param lName - Holds the employee's last name.

\*/

**public** **void** **setLastName**(String lName) {

**this**.lName = lName;

}

/\*\*

\* Returns the employee's current position.

\* @return p - current employee's position.

\*/

**public** Position **getPos**() {

**return** p;

}

/\*\*

\* Returns the employee's first name.

\* @return fName - employee's first name

\*/

**public** String **getFirstName**() {

**return** fName;

}

/\*\*

\* Returns the employee's last name.

\* @return lName - The employee's last name.

\*/

**public** String **getLastName**() {

**return** lName;

}

/\*\*

\* Returns the employee's ID number.

\* @return empNum - The employee's ID number.

\*/

**public** **int** **getEmpNum**() {

**return** empNum;

}

/\*\*

\* Derived class that defines the calculated weekly pay for

\* each position an employee is working in for the company.

\* @return derived calculated weekly pay from all 4 sub classes.

\*/

**public** **abstract** **double** **calculateWeeklyPay**();

/\*\*

\* Returns the employee's ID number if calculateWeeklyPay != null.

\* @return empNum - Otherwise return null.

\*/

**@Override**

**public** String **toString**() {

// String.format will print pay to 2 decimal values

**if** (String.valueOf(calculateWeeklyPay()) != **null**) {

**return** " "+ String.valueOf(empNum);

}

**return** **null**;

}

}

**Copy of Sales.Java Code**

/\*\*

\* Inherits accessible members from employee and represents employees who work in sales for the company.

\* @author Josue Ponce

\* @version 1.0

\*/

**public** **class** **Sales** **extends** Employee{

/\*\*Attribute BONUS holds the bonus earned for each sale the employee makes. \*/

**private** **final** **static** **int** BONUS = **250**;

/\*\*Attribute P holds the predefined constant for employees in sales. \*/

**private** **final** **static** Position p = Position.SALES;

/\*\* Attribute weeklySales holds the amount of sales for the week. \*/

**private** **double** weeklySales;

/\*\*

\* Default class constructor.

\*/

**public** **Sales**()

{

**super**();

weeklySales = **0**;

}

/\*\*

\* Parameterized constructor that initiates fName, lName, empNum,and weeklySales.

\* @param fName - Represents the sales employee's first name.

\* @param lName - Represents the sales employee's last name.

\* @param empNum - Represents the sales employee's ID number.

\* @param weeklySales - Represents the total amount of weekly sales.

\*/

**public** **Sales**(String fName, String lName, **int** empNum, **double** weeklySales)

{

**super**(fName, lName, empNum, p);

**this**.weeklySales = weeklySales;

}

/\*\*

\* Setter method sets the total amount of weekly sales made by employees.

\* @param weeklySales - Holds the total amount of sales the employees have made for the week.

\*/

**public** **void** **setWeeklySales**(**double** weeklySales)

{

**this**.weeklySales = weeklySales;

}

/\*\*

\* Returns weekly Sales.

\* @return weeklySales - Weekly sales made by employees.

\*/

**public** **double** **getWeeklySales**()

{

**return** weeklySales;

}

/\*\*

\* Calculates the weekly pay for sales employees based on the total amount of weekly sales made

\* multiplied by 0.057 plus the bonus per sale.

\* @return weeklySales - Calculated weeklySales multiplied by 0.057 plus the bonus.

\*/

**public** **double** **calculateWeeklyPay**() {

**return** (weeklySales \* .**057**) + BONUS;

}

/\*\*

\* Super.toString() calls Object class method toSting() and also calculates weekly pay in proper currency format for this sub class.

\* @return information from the designer's subclass and calculated weekly pay in proper format.

\*/

**@Override**

**public** String **toString**(){

**return** **super**.toString()+" $"+String.format("%,.2f", calculateWeeklyPay());

}

}

**Copy of Design.Java Code**

/\*\*

\* Inherits accessible members from employee and represents employees who are designers for the company.

\* @author Josue Ponce

\* @version 1.0

\*/

**public** **class** **Design** **extends** Employee {

/\*\*Attribute p holds predefined constant for employees who are designers \*/

**private** **final** **static** Position p = Position.DESIGN;

/\*\*Attribute payRate holds the total pay rate per hour for designers. \*/

**private** **double** payRate;

/\*\* Attribute hours holds the total amount of hours for designers \*/

**private** **double** hours;

/\*\*

\* Default class constructor.

\*/

**public** **Design**() {

**super**();

payRate = **0**;

hours = **0**;

}

/\*\*

\* Parameterized constructor that initiates fName, lName, empNum,payRate and hours.

\* @param fName - Represents the designer's first name.

\* @param lName - Represents the designer's last name.

\* @param empNum - Represents the designer's employee ID.

\* @param payRate - Represents the designer's payRate.

\* @param hours - Represents the designer's total amount of hours worked.

\*/

**public** **Design**(String fName, String lName, **int** empNum, **double** payRate, **double** hours) {

**super**(fName, lName, empNum, p);

setPayRate(payRate);

setHours(hours);

}

/\*\*

\* Setter method sets the hourly pay rate for designers.

\* @param payRate - Holds hourly pay rate.

\*/

**public** **void** **setPayRate**(**double** payRate) {

**this**.payRate = payRate;

}

/\*\*

\* Setter method sets the total amount of hours worked.

\* @param hours - Holds the total amount of hours worked.

\*/

**public** **void** **setHours**(**double** hours) {

**this**.hours = hours;

}

/\*\*

\* Returns the pay rate for designers.

\* @return payRate - Total pay per hour.

\*/

**public** **double** **getPayRate**() {

**return** payRate;

}

/\*\*

\* Returns the total amount of hours worked.

\* @return hours - Total amount of hours worked.

\*/

**public** **double** **getHours**() {

**return** hours;

}

/\*\*

\* Returns the calculated weekly pay for designers.

\* @return total - calculated weekly pay for designers.

\*/

**public** **double** **calculateWeeklyPay**() {

**if** (hours <= **40**)

**return** **getHours**() \* getPayRate();

**else** **if** (hours < **0** || payRate < **0**)

**return** -**1**;

**else** {

**double** total;

total = **40** \* payRate;

total += (hours - **40**) \* (payRate \* **1.5**);

**return** total;

}

}

/\*\*

\* Super.toString() calls Object class method toSting() and also calculates weekly pay in proper currency format for this sub class.

\* @return information from the designer's subclass and calculated weekly pay in proper format.

\*/

**@Override**

**public** String **toString**(){

**return** **super**.toString()+" $"+String.format("%,.2f", calculateWeeklyPay());

}

}

**Copy of Position.Java Code**

/\*\*

\* Enmun type Position sets Manager, Sales, Design and Manufacturing as a fixed set of predefined constants.

\* @author Josue Ponce

\* @version 1.0

\*

\*/

**public** **enum** Position {

/\*\*Attribute MANAGER holds the name of the position for employees who are managers. \*/

MANAGER("Manager"),

/\*\*Attribute SALES holds the name of the position for employees who work in sales. \*/

SALES("Sales"),

/\*\*Attribute DESIGN holds the name of the position for employees who are designers. \*/

DESIGN("Design"),

/\*\*Attribute MANUFACTURING holds the name of the position for employees who work in manufacturing. \*/

MANUFACTURING("Manufacturing");

/\*\*Attribute name holds the name of the positions. \*/

**private** String name;

/\*\*

\* Parameterized constructor initiates name.

\* @param name - Represents the name of the employee's position.

\*/

Position(String name) {

**this**.name = name;

}

/\*\*

\* Returns the name of the employee's position.

\* @return name - The name of the employee's position.

\*/

**@Override**

**public** String **toString**() {

**return** name;

}

}

**Copy of Manufacturing.Java Code**

/\*\*

\* Inherits accessible members from employee and represents employees who work in manufacturing for the company.

\* @author Josue Ponce

\* @version 1.0

\*/

**public** **class** **Manufacturing** **extends** Employee

{

/\*\*Attribute p holds predefined constant for employees in manufacturing. \*/

**private** **final** **static** Position p = Position.MANUFACTURING;

/\*\*Attribute pieceAmount holds the total amount of pieces manufactured by the employee\*/

**private** **double** pieceAmount;

/\*\* Attribute pricePerPiece holds the price per piece manufactured by the employee.\*/

**private** **double** pricePerPiece;

/\*\*

\* Default constructor.

\*/

**public** **Manufacturing**()

{

**super**();

setPieceAmount(**0**);

setPricePerPiece(**0**);

}

/\*\*

\* Parameterized Constructor that initializes fName,lName, pieceAmount, PricePerPice, and int empNum.

\* @param fName - Represents the manufacturer's first name.

\* @param lName - Represents the manufacturer's last name.

\* @param pieceAmount - Represents the total amount of pieces manufactured.

\* @param pricePerPiece - Represents the total cost per piece.

\* @param empNum - Represents the manufacturer's employees ID number.

\*/

**public** **Manufacturing**(String fName, String lName, **double** pieceAmount, **double** pricePerPiece, **int** empNum)

{

// call super

**super**(fName, lName, empNum, p);

setPieceAmount(pieceAmount);

setPricePerPiece(pricePerPiece);

}

/\*\*

\* Setter method sets the price for each piece manufactured.

\* @param pricePerPiece - Holds the price for each piece manufactured.

\*/

**public** **void** **setPricePerPiece**(**double** pricePerPiece) {

**this**.pricePerPiece = pricePerPiece;

}

/\*\*

\* Setter method sets the total piece amount.

\* @param pieceAmount - Holds the total cost for each piece.

\*/

**public** **void** **setPieceAmount**(**double** pieceAmount) {

**this**.pieceAmount = pieceAmount;

}

/\*\*

\* Returns the calculated weekly pay for employees in manufacturing

\* @return getPieceAmount() \* getPricePerPiece();

\*/

**public** **double** **calculateWeeklyPay**() {

**return** **getPieceAmount**() \* getPricePerPiece();

}

/\*\*

\* Returns the total amount for each piece manufactured.

\* @return pieceAmount - Total amount each piece is worth.

\*/

**public** **double** **getPieceAmount**() {

**return** pieceAmount;

}

/\*\*

\*

\* @return pricePerPiece - Price per each item manufactured.

\*/

**public** **double** **getPricePerPiece**() {

**return** pricePerPiece;

}

/\*\*

\* Super.toString() calls Object class method toSting() and also calculates weekly pay in proper currency format for this sub class.

\* @return information from the manufacturing subclass and calculated weekly pay in proper format.

\*/

**@Override**

**public** String **toString**(){

**return** **super**.toString()+" $"+String.format("%,.2f", calculateWeeklyPay());

}

}

**Copy of Manager.Java Code**

/\*\*

\* Inherits accessible members from employee and represents the managers for the company.

\* @author Josue Ponce

\* @version 1.0

\*

\*/

**public** **class** **Manager** **extends** Employee {

/\*\*Attribute P holds the predefined constant for employees who are managers. \*/

**private** **final** **static** Position p = Position.MANAGER;

/\*\*Attribute salary holds the salary earned by managers. \*/

**private** **double** salary;

/\*\*

\* Default class constructor.

\*/

**public** **Manager**() {

**super**();

salary = **0**;

}

/\*\*

\* Parameterized constructor that initiates fName, lName, empNum, and salary.

\* @param fName - Represents the manager's first name.

\* @param lName - Represents the manager's last name.

\* @param empNum - Represents the manager's employee ID

\* @param salary - Represents the manager's salary.

\*/

**public** **Manager**(String fName, String lName, **int** empNum, **double** salary) {

**super**(fName, lName, empNum, p);

**this**.salary = salary;

}

/\*\*

\* Returns the calculated weekly pay for managers.

\* @return salary - Total weekly salary for managers.

\*/

**public** **double** **calculateWeeklyPay**() {

**return** salary;

}

/\*\*

\* Gets the employee's salary.

\* @return Salary - Returns the manager's salary.

\*/

**public** **double** **getSalary**() {

**return** salary;

}

/\*\*

\* Setter method to set the manager's salary.

\* @param salary - Holds the salary for managers.

\*/

**public** **void** **setSalary**(**double** salary) {

**this**.salary = salary;

}

/\*\*

\*Super.toString() calls Object class method toSting() and also calculates weekly pay in proper currency format for this sub class.

\*@return Information from the manager's subclass and calculated weekly pay in proper format.

\*/

**@Override**

**public** String **toString**(){

**return** **super**.toString()+" $"+String.format("%,.2f", calculateWeeklyPay());

}

}