Assignment 7

CMSC 103

Kartchner / Alexander

Concepts tested by this program:

Layouts with JavaFX

Reading from a file

Comparable Interface

New concepts tested by this program

ArrayList

For each loop

Inheritance

Polymorphism

Abstract classes

Overriding methods

super()

Sort ArrayList

Writing to a file

WEEKLY PAY REPORT FOR PETE’S PETS

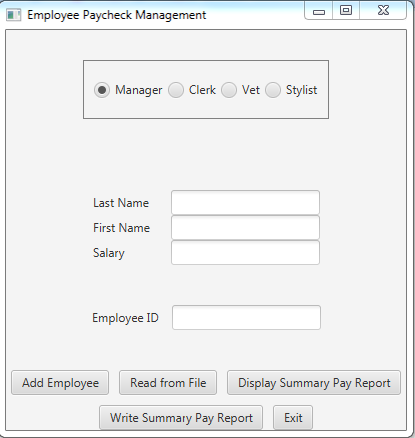
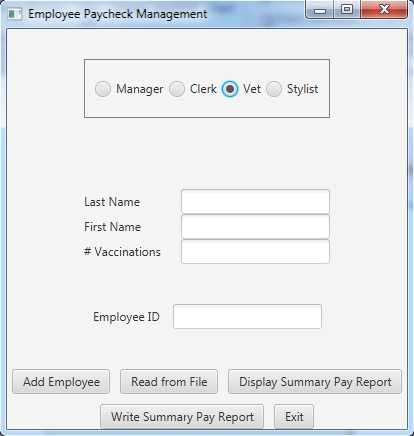
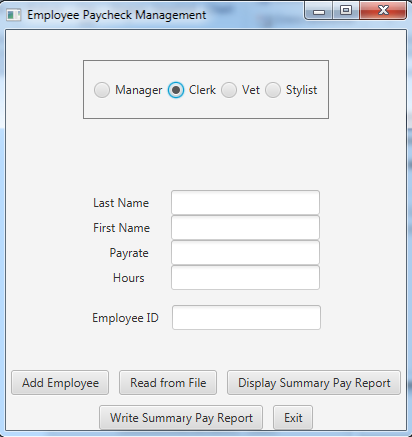
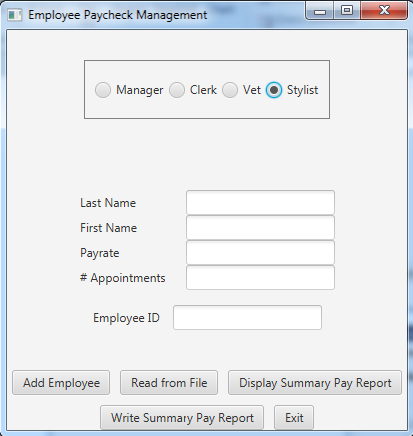
The Pete’s Pets has four types of employees. Managers (who receive a fixed weekly salary), Clerks (who receive a fixed hourly wage for up to the first 40 hours they work and "time-and-a-half," i.e., 1.5 times their hourly wage, for overtime hours worked), Vets (who receive $800 plus $3.75 for each vaccine they administer), and Stylists (who receive a fixed amount of money per appointment).

Write a Java program to compute the weekly pay for each employee and the total payroll. You do not know the number of employees in advance. Each type of employee has its own pay code: Managers have paycode 1, Clerks have code 2, Vets have code 3 and Stylists have code 4. Initial input will come from a sequential file that has a line for each employee. The first item in the line will always be the paycode. The second and third item will be the first and last name. For a paycode of 1, the salary will be the next item. If the pay code is 2, hourly salary is the next item followed by hours worked. If the pay code is 3, the next item will be the number of vaccines administered. If the pay code is 4, the next item will be the payrate per appointment followed by the number of appointments. The last item on the line will be the worker's ID number. Additional input can come from the keyboard. The user will select the type of employee, then fill in the fields for that employee and then select the Add Employee button.

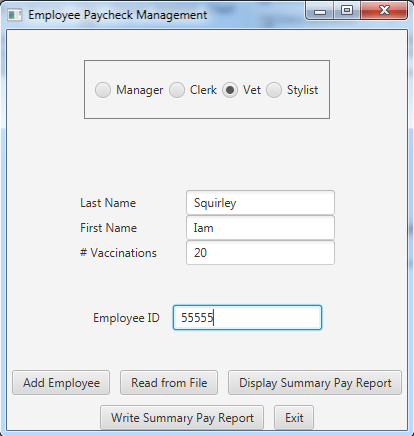
When the weekly pay is calculated, a report can be generated and sent to the screen and written to a text file. Each employee weekly wage should be printed before the summary items. The summary should consist of the total payroll and the number of workers in each category. Use the FileChooser dialog box to select the input and output file names at run time. Print all dollar amounts with currency formatting. At any time the user can select the “Read from File” button and choose a file to read in employees.

You can assume that the input files are in the correct format.

The user can add new employees by selecting the correct radio button. The appropriate labels and text boxes should appear.

When the screen displays text fields for the correct employee type, user can enter the information, and can then select the “Add Employee” button.



When the user selects the “Display Summary Pay Report”, a JOptionPane should display information in the following format:



“Write Summary Pay Report” uses FileChooser to find or name a file to which to write. This will write the same summary as above to a file. Be sure to select a different file name than the input file, or you will overwrite it with output.

## **Specifications**

**Data Element classes** – Employee, Manager, Clerk, Vet, Sylist

* Create an abstract class called Employee with one abstract method called calculateWeeklyPay which returns a double. This class will have instance variables for the first name, last name and employee id, a toString method and a constructor. This class will implement the Comparable interface and will provide a compareTo method that compares the employee id. Employee ids will be compared based on their numeric value.
* Create the following inherited classes for the 4 types of employees: Manager, Clerk, Vet, and Stylist. They will contain instance variables appropriate for the type of employee. They will extend from Employee and define the calculateWeeklyPay method. There should also be getter and setters, and any other methods that are needed for your design. The toString method and constructors for these classes will use the super reference to take advantage of the Employee constructor and toString methods. Use finals to represent any constants.

**Data Manager class** - Employees

* Create an Employees class that implements EmployeesInterface. It will contain an Arraylist which holds Employee/Comparable objects (Manager, Clerk, Vet, Stylist all extending Employee). It will have methods required by the EmployeesInterface, e.g., addEmployee, calculateWeeklyTotal, generateWeeklyReport, sort, retrieve number of managers, clerks, etc., and any other methods needed for your design. The toString method for Employees will return a String with the employee number and weekly pay for all employees. Use a for each loop anytime you need to process all the objects in the Arraylist.

**Gui Driver class**

* Create Stage/Pane classes which implement the GUI. They will have methods to read from the file, print the report to the screen, print the report to the file, and any other methods needed for your design. Use the FileChooser dialog box to select the input and output file names at run time. You can assume all information in the input file is correct. Use Radio buttons to select the type of employee. The fields change as the type of employee changes.

**Utility class** - SortingArrayList

* I have provided a SortingArray class that will need to be modified to work with an ArrayList.

**Other specifications**

* Create Javadoc for your Employee, Manager, Clerk, Vet, Stylist and Employees classes.
* I have provided a JUnit Test class that tests the methods of the Employees class to add an employee, calculate weekly total, generate the weekly pay report, retrieve number of managers, clerks, etc.
* Create a Junit test for your Vet class.
* Insure that your solution passes the JUnit tests, implements the interface, and operates the GUI as described above. Turn in your all the java code needed to run your program, including the EmployeeInterface.java, EmployeesTest.java, and SortingArrayList.java.

Deliverables:

Java files - The src folder with your driver (javafx application), utility class, data elements, data manager and Junit Test (.java) files

Javadoc files - The doc folder with your javadoc for student generated files

UML Class Diagram (an image, not the proprietary format, must be a .jpg)

Deliverable format: The above deliverables will be packaged as follows. Two compressed files in the following formats:

LastNameFirstName\_AssignmentX\_Complete.zip [a compressed file containing the following]

                        UML.jpg

Assignment 7 Checklist (filled in with YES or NO or ?)

                        doc [a directory] *please include the entire doc folder with the javadoc for student*

*generated files*

                                    file1.html (example)

                                    file2.html (example)

                        src [a directory] *contains your driver (javafx application), utility class, data*

*elements, data manager and Junit Test (.java) files*

                                    File1.java (example)

                                    File2.java (example)

                                    File\_Test.java (example)

LastNameFirstName\_AssignmentX\_Moss.zip [a compressed file containing only the following]

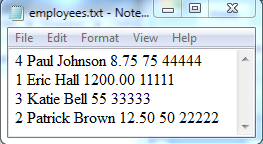
*contains .java file which includes the driver (javafx application), utility*

*class, data elements, data manager and Junit Test (.java) files – NO FOLDERS!!*

                        File1.java (example)

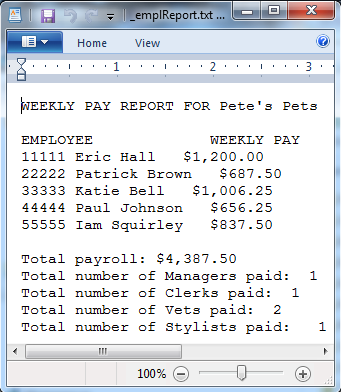
                        File2.java (example)

**Input file for test case:**



When you create your own test files, make sure there are not any empty lines at the end of your data

**Output file from the test case:**



If your output file doesn’t display correctly, try opening it in WordPad instead of NotePad.

**Program Grade Sheet**

Assignment #7

Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DOCUMENTATION**  (30 pts)

# CheckList for Assignment 7 is included and completed 1 pt \_\_\_\_\_\_

**Javadoc** for the following classes: 10 pts \_\_\_\_\_

Employee, Manager, Clerk, Vet, Stylist

Employees, GUI classes

Test Cases 10 pts \_\_\_\_\_

**JUnit Test Class**

Implement STUDENT Tests for methods of Employees

Create a Junit test for your Vet class

**UML Diagram** 4 pts \_\_\_\_\_

**Lessons Learned** 5 pts \_\_\_\_\_\_

In 3+ paragraphs, highlight your lessons learned and learning experience from working on this project. How did you do? What have you learned? What did you struggle with? How will you approach your next project differently?

**PROGRAMMING** (70 pts)

Internal class documentation (within source code) using Javadoc 6 pts \_\_\_\_\_

Description of what class does

Author’s Name, @author

Methods commented properly using Javadoc

Description

@param, @return

Compiles and Runs without runtime errors or warnings 10 pts \_\_\_\_\_

Accuracy

Public tests - JUnit tests given you 6 pts \_\_\_\_\_

Your Junit Tests – STUDENT test for Employees 4 pts \_\_\_\_\_

Private tests 10 pts \_\_\_\_\_

Program Details

Employee Class 4 pts \_\_\_\_\_

1. Create an abstract Employee class with abstract method

calculateWeeklyPay

1. Implements Comparable
2. Contains a compareTo method
3. Instance variables common to all Employee classes
4. Getters and setters for those instance variables

Inherited Classes 10 pts \_\_\_\_\_

1. Create Manager, Clerk, Vet and Stylist classes that inherit

from Employee

1. Define the calculateWeeklyPay method
2. Additional instance variables if needed
3. Includes getters and setters for instance variables
4. Uses finals for constant numbers
5. Uses super within the constructor and toString methods

Employees Class 10 pts \_\_\_\_\_

1. Implements EmployeesInterface
2. Contains an arraylist of Employee objects
3. Uses for each loops
4. Contains methods for adding, calculating total weekly pay,

generating a weekly pay report, toString, getters and setters

GUI Classes 10 pts \_\_\_\_\_

1. Gets the file names and paths from a FileChooser dialog box
2. Prints a the report to the screen
3. Prints the report to a file
4. Prints a program title on the screen and output file
5. Weekly pay formatted to two decimal places
6. Uses an Employees object
7. Program user interface

Clear to user how data is to be entered

Output is easy to understand

Total

100 pts \_\_\_\_\_