Computer Science 205

Project #3

The Personnel Database

50 Points

Due Date : Wednesday, November 8th, 11:59 PM

Objective

The purpose of this assignment is to extend students' familiarity with inheritance, polymorphism, and abstract classes. The program also provides the first introduction to working with objects on external files.

Assignment Summary

Write a program named `Personnel` that maintains wage information for the employees of a company. The following example shows what the user will see during a session with the program.

The program will repeatedly prompt the user to enter commands, which it then executes. The program will not terminate until the user enters the command `q`. Note that the list of commands is re-displayed after each command has been executed.

Sample Run

```
---------------------------------
|Commands: n - New employee      |
|          c - Compute paychecks |
|          r - Raise wages       |
|          p - Print records     |
|          d - Download data     |
|          u - Upload data       |
|          q - Quit              |
---------------------------------
Enter command:  n
Enter the name of new employee:  Plumber, Phil
Hourly (h) or salaried (s):  h
Enter hourly wage:  40.00

---------------------------------
|Commands: n - New employee      |
|          c - Compute paychecks |
```

```
|          r - Raise wages       |
|          p - Print records     |
|          d - Download data     |
|          u - Upload data       |
|          q - Quit              |
---------------------------------
Enter command:  n
Enter name of new employee:  Coder, Carol
Hourly (h) or salaried (s):  s
Enter annual salary:  80000

---------------------------------
|Commands: n - New employee      |
|          c - Compute paychecks |
|          r - Raise wages       |
|          p - Print records     |
|          d - Download data     |
|          u - Upload data       |
|          q - Quit              |
---------------------------------
Enter command:  c
Enter number of hours worked by Plumber, Phil:  50
Pay:  $2200.00
Enter number of hours worked by Coder, Carol:  50
Pay:  $1538.46

---------------------------------
|Commands: n - New employee      |
|          c - Compute paychecks |
|          r - Raise wages       |
|          p - Print records     |
|          d - Download data     |
|          u - Upload data       |
|          q - Quit              |
---------------------------------
Enter command:  r
Enter percentage increase:  4.5

New Wages
---------
Plumber, Phil               $41.80/hour
Coder, Carol              $83600.00/year

---------------------------------
|Commands: n - New employee      |
|          c - Compute paychecks |
|          r - Raise wages       |
|          p - Print records     |
|          d - Download data     |
```

```
|          u - Upload data       |
|          q - Quit              |
----------------------------------
Enter command:  q
```

Class Specifications

Write three classes that store information about an employee:

- `Employee` (abstract). Instances of this class will store an employee name (a string) and the employee's hourly wage (a `double` value). Methods will include getters and setters for the name and hourly wage, a method that increases the hourly wage by a given percentage, and an abstract method named `computePay` that computes the weekly pay for the employee when given the number of hours worked.

- `HourlyEmployee`, which extends `Employee`. The constructor will take a name and hourly wage as its parameters. Methods will include `computePay` and `toString`. To determine the employee's pay, `computePay` multiplies the first 40 hours (or fewer) by the employee's hourly wage. Hours worked beyond 40 are paid at time-and-a-half (1.5 times the hourly wage). `toString` returns a string containing the employee's name and hourly wage, formatted as shown in the example of the `r` command. Note that spaces are added between the name and wage so that the entire string is 40 characters long.

- `SalariedEmployee`, which extends `Employee`. The constructor will take a name and annual salary as its parameters. Methods will include a getter and a setter for the annual salary, along with `computePay` and `toString`. (Note that the salary will need to be converted to an hourly wage, because that's what the `Employee` class requires. To do this conversion, assume that a salaried employee works 40 hours a week for 52 weeks.) `computePay` always returns 1/52 of the annual salary, regardless of the number of hours worked. `toString` returns a string containing the employee's name and annual salary, formatted as shown in the example of the `r` command.

Data Structure

Use an array to store the employee records. Each element of the array will store a reference to an `HourlyEmployee` object or a `SalariedEmployee` object. The array used to store employee objects must contain only one element initially. When the array becomes full, it must be doubled in size. To do this, you can use traditional arrays that you resize or arraylists. To resize a traditional array, use a coding segment similar to the following. We're just creating a new temporary array object that is twice as large, and then giving our old array the address of this array. Remember to be careful when passing by reference.

```
if (numEmployees == employees.length) {
  Employee[] tempArray =
    new Employee[employees.length * 2];
  for (int i = 0; i < employees.length; i++)
```

```
      tempArray[i] = employees[i];
    employees = tempArray;
  }
```

Additional Requirements

Here are a few other requirements for the program:

- The user should be able to print the current set of active records, download all the active records to be stored on disc, and upload new records. All external data should be stored in a file named `employee.dat`.

  See your class handout and class demo files for information on reading objects from a file, and writing objects out to a file. Note that you will need to make all classes `Serializable`. For example,

  ```
  class HourlyEmployee extends Employee implements Serializable
  ```

  If any records are currently active when you do a file upload, add all new records to the end of your array.

- Dollar amounts must be displayed correctly, with two digits after the decimal point. For example, make sure that your program displays $100.00, not $100.0.

- All user input may be preceded or followed by spaces. Commands may be entered as upper-case or lowercase letters. If the user enters a command other than `n`, `c`, `r`, `p`, `d`, `u`, or `q`, the program must display following message:

  ```
  Command was not recognized;  please try again.
  ```

- When asked if the employee is hourly or salaried, the user must enter either `h` or `s` (case doesn't matter). If the user enters any other input, the program must display the following message:

  ```
  Input was not h or s;  please try again.
  ```

Analysis & Design

You are required to create a one page typed write-up of the problem description, the program specification, and your algorithm for the driver program. This should be stored in the file `README` in your project directory.

Briefly explain all of the different classes used, their attributes, and their operations.

The program specification should describe exactly what type of data is being input and output as well as the format of the input and output files.

Extra Credit and Revision Policy

You will receive at least five extra credit points added to your score out of 50 if you complete this assignment 24 hours before the due date (Tuesday, November 7th). You may also receive extra credit points for adding significant additional features beyond the minimum requirements.

All assignments handed in on or before the due date that you do not receive full credit for in either implementation or documentation are eligible for revision.

Submission

In your typescript, include a listing of your main program followed by all supporting classes. Include a sample run which illustrates all menu options.