

Programming Project 3

Note: When you turn in an assignment to be graded in this class, you are making the claim that you neither gave nor received assistance on the work you turned in (except, of course, assistance from the instructor).

Write a program named **Project3** that maintains wage information for the employees of a company. The company maintains a text file of all of the current employees called **EmployeesIn.dat**. Each line of the file contains the employee's name (last name, first name), a character code indicating their wage status (**s** for salary, **h** for hourly, **w** for student worker, or **i** for intern), and their wage (hourly rate for hourly employees, student workers, and interns; total annual salary for salary employees). The name, wage status and wage are separated by varying amounts of white space (at minimum, there are two spaces). For example,

```
Harris, Joan      h      15.00
Baird, Joey       h      17.50
Kinsey, Paul      h      18.00
Brinkley, Bobby   i     10.00
Stevens, Sam      w       8.00
Olson, Margaret   s    25000
Campbell, Peter   s    30000
Draper, Donald    s    60000
Sterling, Roger   s    75000
Cooper, Bertram   s    90000
```

Once a week (generally on Friday), updates to the file (**EmployeesIn.dat**) are made and the weekly paycheck amount is calculated for each employee based on the hours worked that week. The update information is read from a second text file called **Updates.dat**. Updates can be any of the following:

- adding a new employee (**n**),
- raising the rate of pay for all employees except new hires, interns, and student workers by a given percentage (**r**), or
- dismissal of an existing employee (**d**).

For example, the file could contain the following lines:

```
n      Pryce, Lane      s      40000
r      5
d      Kinsey
```

After performing the updates, the revised information is written to a text file named **EmployeesOut.dat** a summary report is *displayed on the console* in the following format,

New Employee added: Pryce, Lane

New Wages:

Harris, Joan	\$15.25/hour
Baird, Joey	\$18.38/hour
Brinkley, Bobby	\$10.00/hour
Stevens, Sam	\$ 8.00/hour
Olson, Margaret	\$26250.00/year
Campbell, Peter	\$31500.00/year
Draper, Donald	\$63000.00/year
Sterling, Roger	\$78750.00/year
Cooper, Bertram	\$94500.00/year
Pryce, Lane	\$40000.00/year

Deleted Employee: Kinsey, Paul

Paycheck amounts are calculated based on each employee's wage before any raises have been applied and number of hours worked for hourly employees. This information is included in the file **HoursWorked.dat**. This file simply lists each employee's last name and the number of hours worked that week, separated by white space. For example,

```
Harris    65
Baird     40
Olson     70
Campbell  40
Draper    60
Sterling  40
Cooper    35
Pryce     44
Brinkley  10
```

After calculating each employee's pay for the week, a report is written to an output file called **WeeklyPayroll.txt** in the following format,

```
Paycheck amount:
      Harris, Joan          $406.88
      Baird, Joey          $315.00
      Olson, Margaret      $302.88
      Campbell, Peter      $403.85
      Draper, Donald       $807.69
      Sterling, Roger      $908.65
      Cooper, Bertram      $1009.62
      Pryce, Lane          $807.69
      Cooper, Bertram      $100.00
      -----
      Total                $19962.26
```

(Please note that all the examples shown are for formatting purposes only; do not expect that the numeric values reflect correct calculations in any way.)

The abstract class `Employee.java` is provided. Do not make any changes to this class. Write the classes that store information about the three categories of employee:

- **HourlyEmployee**, which extends the abstract class **Employee**. The constructor will take an employee's name (first name and last name are two separate data fields) 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 output of the **r** command. Note that spaces are added between the **names** and **wage** so that the entire string is 40 characters long.
- **StudentWorker**, which extends the class **HourlyEmployee** and has the 'w' designation in the input file. The constructor will take an employee's name (first name and last name are two separate data fields) and hourly wage as its parameters. There is a constraint on the wage for student workers that they cannot earn more than \$8.00 / hour. The overridden **computePay** method restricts the student worker to 10 hours of week pay, or less – even if the recorded hours are greater than 10 in a given week.
- **Intern**, also extends the class **HourlyEmployee** and has the 'i' designation in the input file. The constructor will take an employee's name (first name and last name are two separate data fields) and hourly wage as its

parameters. The only other method is the overridden **computePay** method which restricts the intern to 40 hours of week pay, or less – even if the recorded hours are greater than 40 in a given week.

- **SalariedEmployee**, which extends the abstract class **Employee**. The constructor will take the employee's name (first name and last name are two separate data fields) 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 annual salary must 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 output for the **r** command. (*Do not add a separate instance variable for the annual salary. Follow the program specifications in order to receive full credit for this project.*)

The program **MUST** use an **array** to store employee records of data type **Employee**. The array should be managed in a class called **HumanResourceDept**. Each element of the array will store a reference to an **Employee** 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 by calling a method written for this purpose in the **HumanResourceDept** class.

Your main method is located in the **Project3** class, which is responsible for calling its own methods to handle reading and writing the data files. It instantiates a **HumanResourceDept** object in order to process the employee updates and generate the weekly payroll.

Here are a few other requirements for the program:

- After updating the employee records and calculating the weekly payroll, the revised employee information is to be stored in the same format as the original **EmployeesIn.dat** file. However, name this output file **EmployeesOut.dat**.
- 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 input may be preceded or followed by spaces, tabs, or other whitespace characters. Character command codes may be uppercase or lowercase letters. If the user enters a character code other than n, d, h, i, s, w, or r; the program must display the following message along with the line containing the invalid character:
Command was not recognized; <line containing the error>

Write this program in JAVA and compile it in JDK 1.7 or better. Follow all commenting conventions discussed in class, including a comment block at the top of each file with your name, date, the course number and section, program purpose, input, and output. It is expected that your program will be well documented and you are required to include a private helper method called **printHeading** to the **Project3** class that outputs the following information to the console in an easy-to-read format: your name, the project number, the course identifier, and the current semester. You will call this method as the first statement in your **main** method.

You will submit all the project files except **Employee.java** in a zipped folder named **Project3** that you are to upload to Blackboard.