COMP 2503

ASSIGNMENT# 1

Date Given: September 19, 2022

Date Due: September 26, 2022

Dept. of Mathematics & Computing

Mount Royal University

Objectives:

- to develop classses using the Best Software Practices guidelines
- to implement and test the design

Description:

Payroll is a major key function of any company. If the company fails to pay its employees on time, it gets into trouble not only with its employees but also with government and regulatory institutions. Two important information that is generated by the payroll system are the payroll information of each employee and the information required by the government, particularly, Revenue Canada.

Before we can start processing the payroll, we need to create the **Employee** class, which describe the different types of employees and how their pay is calculated.

The Employee Class:

The Employee class describes the different employees that a particular organization has. For Fly By Night Consulting Company, there are three types of employees: salary, hourly, and consultants.

The following provides a description of the Employee class: (Note: the description below only identifies relevant information for our assignment. A real Employee class would contain a lot more information)

1. **Instance Variables**

   The Employee class should have the following instance variables:

   a. **empNo** – an Integer field that uniquely identifies a particular employee
   b. **empName** – a String field that stores the name of the employee
   c. **department** – a String field that stores department to which the employee belongs
   d. **type**– a character field that stores the type of employee. ‘S’ for salary, ‘H’ for hourly, ‘C’ for Consultant
   e. **payRate** – the pay rate for each type of employee. This is defined as follows: for salary employees, this value is the yearly salary; for hourly employees, this value is the hourly wage; for Consultants, this is the hourly rate.
   f. **maxHours** – the maximum number of hours that an employee can work per week. Any hours worked beyond this number is not paid.

2. **Constructors**

   The Employee class will have 3 overloaded constructors.

   a. A default (no-arg) constructor that sets all numerical values to 0 and strings to “null”

   b. A constructor that accepts as parameters, values for the fields.

   c. A copy constructor that will take in the values of another employee except for the employee number, employee name, and department. The other employee whose values will be copied will be passed as an argument. The values of the employee number, employee name, and department will also be passed as arguments. (Note: This would be a way to easily create a new employee that have the same initial values as another employee.).

3. **Accessors**

   There should be accessor methods (getters) for each one of the instance variables. These methods should follow the naming standards for accessors: **getEmpNo**, **getEmpName**, etc.

4. **Mutators**

   There should be mutator methods (setters) for each one of the instance variables. These methods should follow the naming standards for mutators: **setEmpNo**, **setEmpName**, etc.

5. **Processing Methods**

   There should be processing methods to fulfill the following requirements: (the processing details are described in the next section)

   a. Calculate the gross weekly pay
   b. Calculates the deductions from the gross weekly pay which includes:
      1. Income Withholding Tax
      2. Canada Pension Plan (CPP) contribution
      3. Employment Insurance (EI) contribution
      4. Extended Health Benefit
      5. Union Dues
   c. Calculates Net Pay
   d. Comparing one employee to another
   e. Print Details of an employee

   Because we may use this class in other project, it is necessary to standardize the method names of these methods. These will be shown below. You will need to fill in the details of the method header. You are however free to add more methods as you see fit.

Processing Details

1. **Calculate Gross Weekly Pay [**calcGrossPay(hours worked)**]**

   a. **Salary Employees**

      Salaried employees received a set weekly gross pay. The amount is determined by their annual salary divided by 52, the number of weeks per year. Salaried employees are not paid overtime. The maximum number of hours that a salaried employee can work is limited by the number of hours in a 7 day period (7 * 24 = 168).

   b. **Hourly Employees**

      Hourly employees are paid by the hour using their negotiated pay rate per hour. They are also paid overtime. If they work more than 40 hours for the week, any time over 40 hours is paid at time and a half. However, according to union contract, hourly employees can work up to 60 hours a week. Any amount over 60 hours is not paid

   c. **Consultants**

      Consultants are also paid by the hour using the contracted hourly rate. However, consultants are contracted to work only up to a certain maximum number of hours per week determine at the time of the signing of the contract. Any time over this maximum is not paid. They are also not paid for overtime.

2. **Calculate Deductions**

   The following are the deductions from the gross pay. Not all deductions are appropriate for each type of employee. However, employees of the same type have the same deductions. Those type of employees affected by the deductions are identified at the bottom of each description. The percentages are applied to the gross pay. (Note: The government deductions below have been simplified for the purposes of this assignment. The actual requirements are much more complex.)

   1. Income Withholding Tax [calcWithhold (gross)]

      The following table shows both the Federal Income Tax withholding amounts.

Income Range	Federal Tax Deductions
Less than $ 1,000	7.5%
$1,000 to less than $2,000	12%
$2,000 and over	17%

      This deduction apply to all type of employees.

2. Canada Pension Plan (CPP) contribution [calcCPP(gross)]

   4.75% of gross weekly pay. This applies to all employees.

3. Employment Insurance (EI) contribution [calcEI(gross)]

   1.8% of gross weekly pay. This applies to all employees.

4. Extended Health Benefit [calcExtHealth(gross)]

   The premium charged to the participants is 1.3% of their gross weekly pay. Salaried and hourly employees participate in this program.

5. Union Dues [calcUnionDues(gross)]

   Union dues are charged at 1% of their gross weekly pay. Only hourly employees are members of the union.

3. Calculate Net Pay [calcNetPay (hours worked)**]**

Once the gross pay has been determined, the net pay is calculated by calculating the appropriate deductions as described above and subtracting from the gross pay.

4. Compare one employee to another [compareTo (other)**]**

We need to be able to compare one employee to another in order to be able to perform searches and sorting. For this case, the employee number is used for comparison. The other employee is passed as an argument to this **compareTo** method and the employee numbers are compared. The table below shows the results of the comparison:

condition	**returns**
current empNo < other empNo	-1
current empNo = other empNo	0
current empNo > other empNo	1

5. **Print Details**

This should print the data one line per data item with explanatory text. For example:

```
Employee Number:    20101
Employee Name:      Arnold
Employee Type:      Salary        (Note that this is not just a character)
  etc.
```

Design and Development

Although this class is not that complex, developing it properly requires some good programming practices. I recommend that you develop the class incrementally, one processing method at a time and then, checking the method so that it functions properly.

Coding and Documentation Requirements

- **You must use java doc for your headers on each file and on important methods**
  - **Look to the java doc template examples at the end of this document**
- Choose self-documenting identifiers (e.g. for variables, methods, etc.)
- explicit initialization of variables (where appropriate)
- consistent and correct use of white space, including:
  - proper indentation and use of white space to highlight the logical structure of an algorithm
- marking all methods as either public or private, as appropriate
- marking all instance data as private
- making variables local and avoid the use of global variables
- avoiding overly long/complex methods by instead delegating key subtasks to other methods (this includes avoidance of code duplication)
- using inline comments, sparingly, to clarity especially tricky or less-obvious segments of code
- avoiding hard-coded magic literals in favour of named constants
- Choose good classes and service methods based upon the practice you have had in class.

Testing the Employee Class Implementation

It is important to ensure that the class is implemented as expected. Therefore, create an application class that will test the processing methods. You can hard code the data so you can work on sufficiently testing your class. You may, if you prefer, input data from a file. For this purpose, a text file containing three employees is provided. The file name can be inputted or hardcoded.

Assignment Instructions

1. Use **only** Eclipse IDE.
2. The due date for this assignment is posted in D2L.

Submission

Rename the entire folder that contain your code as **<last_name> <first_name> Assign1** and submit as a zip file through the submission link in D2L. Assignments can be submitted late with a penalty:

- 10% deduction for 1 day delay

- 20% deduction for 2 days late
- After 2 days no submission is acceptable (100% deduction)

Javadoc Templates

File header

```
/**
 *  <include description of the class here>
 * @author <your name>
 * @version 1.0
 * Last Modified: <date> - <action> <who made the change>
 */
```

Method with no parameters and no return type

```
/**
 *  <a description of what the method does>
 */
```

Method with no parameters and has a return type

```
/**
 *  <a description of what the method does>
 *  @return <a description of what is returned, including if errors are
returned>
 */
```

Method with parameters and no return type

```
/**
 *  <a description of what the method does>
 *  @param <parame1name> <a one line description of the parameter>
 *  @param <parame2name> <a one line description of the parameter>
 *  < include all parameters in a similar way as above>
 */
```

Method with parameters and has return type

```
/**
 *  <a description of what the method does>
 *  @param <parame1name> <a one line description of the parameter>
 *  @param <parame2name> <a one line description of the parameter>
 *  < include all parameters in a similar way as above>
 * @return <a description of what is returned, including if errors are
returned>
 */
```

Marking Scheme

Description	**Max**
I. Employee Class	
- 6 instance variables	
– Each one properly declared………………………………………..	**3**
- 3 constructors	
– Default	
– Parametrized	
– Copy ……………………………………………………………….	**4**
– Accessors and Mutators	
– One for each instance variables ………………………………………..	**3**
– Processing Methods (Proper method headers)	
– Calculate Gross Pay	
– Salary ………………………………………………………………	**1**
– Hourly ……………………………………………………………..	**3**
– Consultant ………………………………………………………….	**2**
– Calculate Individual Deductions	
– Withholding Tax (tiered deductions) …………………………….	**3**
– CPP Contribution	**1**
– EI Contribuion	**1**
– Extended Health	**1**
– Union Dues ………………………………………………………..	**1**
– Calculate Net Pay (proper deductions depending on employee type)	
– Salary ………………………………………………………………..	**3**
– Hourly	**4**
– Consultant …………………………………………………………	**2**
– Compare To	**3**
– Returns proper values …………………………………………….	
– Print Details	
– Employee Type spelled out	**5**

	40
II. Programming Design and Documentation - Proper use of Java Doc …………………………………………………. - Self documenting identifiers for variables and methods - Use of proper indentation and white space - Declare methods as public or private; instance variables as private - No global variables - Simple methods (implements proper cohesion and coupling)…………….. - Avoidance of code duplication	**2** **1** **1** **2** **1** **2** **1** **10**
III. Errors - Compilation Errors - Runtime Errors	**- 10** **- 10** **50**
Total	