CET2012 - Programming Paradigms: Advanced Java - Practicum 01

Topics Covered: Object-Oriented Programming, Abstraction, Encapsulation, Inheritance

Learning Objectives:

- Apply object-oriented design concepts such as abstraction, encapsulation and inheritance in solving complex problems
- Understand the application of classes and objects
- Apply access modifiers according to the program specifications
- Create a class diagram
- Introduce and familiarize using Java Platform API.

Deliverables:

- Submit a single zip file called `CET2012_P01_<Your_Name>.zip` (e.g. `CET2012_P01_John_Doe.zip`) containing your Java code and an image file of your class diagram.

Overview

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or *properties*), and code, in the form of procedures (often known as *methods*). In this practicum you will be applying concepts of object-oriented programming , in particular abstraction, encapsulation and inheritance to solve a problem.

As an introduction to OOP, you will be writing a simple payroll system for the employees of DigiBank. The program also calculates the Central Digi-Fund (CDF) contributions, a mandatory saving scheme for all citizens in DigiWorld.

Program Requirements

There are 3 types of employees at DigiBank:

1. **Fixed Salary** - A fixed salary employee will have a fixed amount paid each month. This employee is subjected to mandatory CDF contributions. In addition, he/she is entitled to additional wage in the form of business expenses.
2. **Hourly Wage** - An hourly wage employee is paid a fixed hourly wage rate. This employee can work a maximum of 176 hours in a month. If the employee works for more than 176 hours, he/she is entitled to an additional wage component i.e. overtime pay for each additional hour or part thereof. The overtime hourly wage rate will be 1.5 of the fixed hourly wage rate. For example, if the employee is paid $10/hour, his/her overtime wage will be $15/hour. This employee is subjected to mandatory CDF contributions.
3. **Intern** - An intern is paid fixed allowances each month. Interns are **not** subjected to mandatory CDF contributions and is **not** entitled to any additional wage.

For the CDF contribution rates, please refer to the table below:

CDF Contribution Rate Table

Employee's Total Wage	Total CDF contributions	Employee's share of CDF contributions
≤ $50	Nil	Nil
> $50 to $500	13% (TW)	Nil
> $500 to $750	13% (TW) + 0.6 (TW - $500)	0.6 (TW - $500)
> $750	[33% (OW)]* + 33% (AW) * Max. of $1,980	[17% (OW)]* + 17% (AW) * Max. of $1,020

Notes:
OW: Ordinary Wages (capped at OW Ceiling of $6000)
AW: Additional Wages (including overtime pay & business expenses)
TW: Total Wages = OW + AW

Steps to compute CDF contributions

1. Compute the total CDF contribution (rounded to the nearest dollar). Cents should be dropped for an amount less than 50 cents. An amount of 50 cents and above should be treated as an additional dollar.
2. Compute the employee's share of CDF contribution (cents should be dropped).
3. Employer's share = Total contribution - Employee's share

Tasks

- As part of the practicum, you are to provide a class diagram of your program. You may use https://app.diagrams.net/ to create your class diagrams. **You should save your class diagram as `<Your_Name>.jpeg` or `<Your_Name>.png` file.**
- Using the following incomplete abstract class, write a Java program that will allow a user to generate an employee's payroll depending on the type of employee, wage, and, if applicable, number of hours worked.

```
1  import java.math.BigDecimal;
2
3  /**
4   * An abstract class used to create concrete employee classes.
5   * The abstract class provides the general implementation of the CDF
6   * calculations based off a Fixed Salaried employee. It also provides
7   * a getter method to return the current number of employees employed
8   * by the company.
9   * An abstract method is also provided for the concrete classes to setup its
10  * individual wage components before generating the payroll.
11  */
12
13 public abstract class Employee {
```

```java

    protected String name;       // variable for employee name
    private static int count = 0; // count the total number of employees
    Employee(String name) {
        this.name = name;
        count++;
    }

    /**
     * Abstract method for the concrete classes to implement their own
     * calculation of the different wage components
     */
    protected abstract void setupWageComponents();

    /**
     * Returns the payroll components as a BigDecimal array depending on
     * the type of employee, wage, and if applicable, number of hours
     * worked.
     * Payroll components include:
     * 1. gross pay i.e. OW + AW (before CDF deduction)
     * 2. employee's CDF
     * 3. employer's CDF
     * 4. net pay (after CPF deduction)
     * 5. total CPF contributions
     *
     * @return  payroll components
     */
    protected BigDecimal[] generatePayroll() {
        // your code here
    }

    /**
     * Returns the current number of employees employed by the company
     *
     * @return int Number of employees
     */
    protected static int getCount() {
        return count;
    }
}
```

Employee
Fixed
Hourly
Intern

Driver - not to be submitted. So try to contain all printout statements in driver.

take note end product has no print statements

BigDecimal usage means no more using of Float and Double

- The three concrete employee classes should be called
    1. `Fixed`
    2. `Hourly`
    3. `Intern`

concrete so should be relate

- As part of your practicum, you are expected to use the `BigDecimal` & `RoundingMode` classes from the `java.math` package to represent the employee's wages and aid in wages calculation. You may refer to the official documentation (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/math/package-summary.html) for more information.
- Document your code using the style specified for Javadoc (https://www.oracle.com/sg/technical-resources/articles/java/javadoc-tool.html). You are to ensure that the Javadoc files can be generated successfully using the Javadoc tool.
- Add any other methods and variables that you may require

For the maximum allocation of marks, refer to the table below.

Description	Marks (%)
Proper construction of class diagram	10
Implementation of the program as according to class diagram (does not include success of the program)	5
Successful implementation of program (passes instructor's test case, no bugs, does not crash)	65
Proper use of `BigDecimal` & `RoundingMode` classes from the `java.math` package	5
Correct use of access modifiers	10
Proper and generable documentation using Javadoc tool	5

class diagram and code should match

Once you have completed, zip your files and rename in the following format `CET2012_P01_<Your_Name>.zip` e.g. `CET2012_P01_John_Doe.zip`.