Object-Oriented Programming

Overview

In this lab, you will write an application that defines and uses an `Employee` class.

Source folders

Student folder : `C:\PythonDev\Student\06-OOP`

Solution folder: `C:\PythonDev\Solutions\06-OOP`

Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Defining a class and creating objects
2. Implementing "pay bonus" functionality
3. Adding class-wide members
4. Additional suggestions (if time permits)

Exercise 1: Defining a class and creating objects

Write a Python program that defines a simple `Employee` class and creates some instances.

Suggestions and requirements:

- The `Employee` class needs to hold the name and salary of the employee, and the date/time he/she joined the company. For the date/time, you can use the `datetime` class from the `datetime` module:

```
from datetime import datetime
…
currentDateAndTimeVariable = datetime.now()
```

- The class must honour the OO principle of encapsulation, so make sure the instance variables are private.
- The class needs to allow an employee to have a pay raise, so define a `payRaise()` method that takes the amount of the pay raise and adds it to the employee's current salary.
- The class should also have a `toString()` method that returns a textual representation of the employee's info. Note, to format the "date joined" value, you can call the `strftime("%c")` method defined in the `datetime` class.
- Write some client code, where you can create some `Employee` objects and invoke methods upon them.

Exercise 2: Implementing "pay bonus" functionality

In the `Employee` class, add a method named `payBonus()`. The method should be flexible enough to be called in three different ways (you'll need to declare some default argument values to support this behavior):

- The client can call the method with no parameters. In this case, add a fixed percentage of the employee's salary (e.g. a 1% bonus).
- The client can call the method with a single parameter, specifying the percentage of the bonus. For example, the client code might request a 10% pay bonus.
- The client can call the method with parameters specifying the percentage of the bonus, along with a minimum and maximum salary (such that the bonus only applies if the employee's salary is within that range).

Exercise 3: Adding class-wide members

Refactor your `Employee` class to make appropriate use of class-wide variables and methods.

Suggestions and requirements:

- In the `Employee` class, define a class-wide variable to hold the statutory minimum salary for all employees. Set it to 12000. Use this class variable in the constructor, to ensure the employee earns at least this much.
- Define class-wide methods to get / set the statutory minimum salary. Call these methods from your test code.

Exercise 4: Additional suggestions (if time permits)

In the `Employee` class, define an instance variable to hold the employee's ID number. Also add a class-wide variable to hold the next employee ID, which will be incremented each time a new employee is created.