Lab 7: Programming a Payroll System with OOP

Objectives

This lab is intended to demonstrate the major object-oriented programming features and possibility of collaboratively developing a complex information system with OOP.

General Guideline

This lab is a team project. Each team may freely choose up to 4 students from the class. It is expected that every member actively participates and makes fair contributions to the project. Each member must be responsible for completing at least two classes/files. Put your name in PHP comments in the files you complete. All members in the same team will receive the same score. However, if some members do not make fair contributions, scores may be adjusted among members in the same team.

Project Description

In this project, each team will create a PHP application for a company that wishes to perform several accounting operations in a single accounts payable application. The payroll system needs to be implemented with object-oriented programming approach. The application calculates the earning that must be paid to each employee. It also calculates the payment due on each of several invoices (i.e. bills for goods purchased).

You may test the application at http://www.iupui.edu/~i211. The lab is located in Unit 3.

The company pays its employees on a weekly basis. The employees are of four types: **salaried** employees are paid a fixed weekly salary regardless of the number of hours worked; **hourly** employees are paid by hours and receive overtime pay (150%) for all hours worked in excess of 40 hours; **commission** employees are paid a percentage of their sales; and **base plus commission** employees receive a base salary plus a percentage of their sales.

The application should contain nine files:

1. The *payable.class.php* file contains the interface, which is implemented by **Employee** and **Invoice** classes.
2. The *invoice*.class.php file contains the **Invoice** class, which implements the **Payable** interface. The class also contains a static attribute and method that keep track of number of Invoice objects created in the system.
3. The *employee.class.php* file contains an abstract, base class named **Employee**. One of the attributes of the class is an object of **Person**. This demonstrates one of the three relationship types among classes: composition. This class also contains a static attribute and method that keep track of number of Employee objects created in the system.
4. The *person.class.php* file contains the **Person** class.

5. The *hourly_employee.class.php* file contains the **HourlyEmployee** class. This class inherits from the **Employee** class.
6. The *salaried_employee.class.php* file contains the **SalariedEmployee** class, which inherits from the **Employee** class.
7. The *commission_employee.class.php* file contains the **CommissionEmployee** class, which also inherits from the **Employee** class.
8. The *base_plus_commission_employee.class.php* contains the class named **BasePlusCommissionEmployee**, which inherits from the **CommissionEmployee** class.
9. The *test_payable.php* contains a client program. The program tests all classes in the **Payable** class hierarchy. Make sure polymorphism is demonstrated in this file. The Web page displays string representation of each object type in the class hierarchy and the number of **Invoice** and **Employee** objects created in the system.

Required OOP Features

Following OOP features must be included in the application:

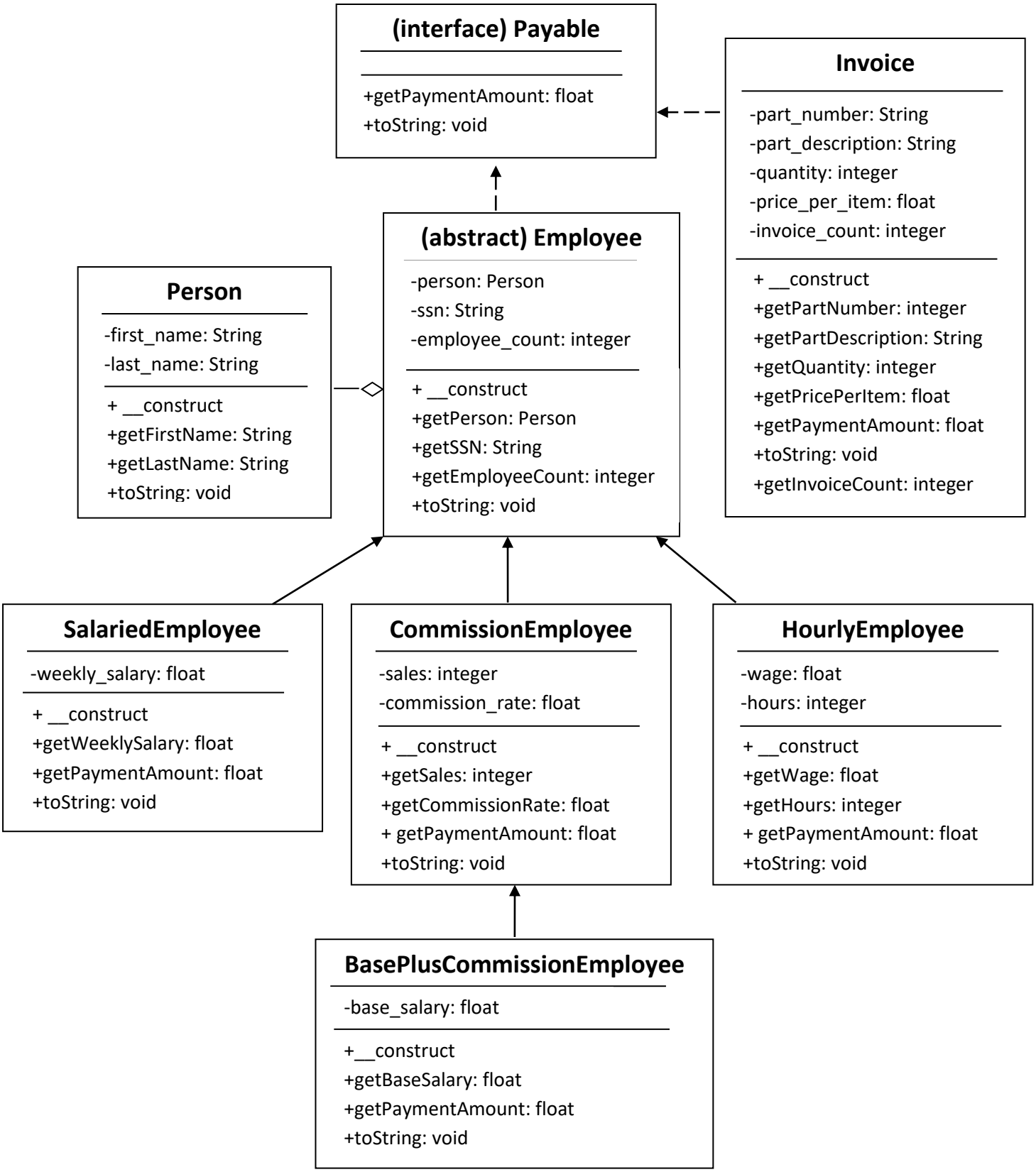
1. Encapsulation
2. Inheritance
3. Composition
4. Polymorphism
5. Interface
6. Abstract classes and methods
7. Static data members and methods
8. Accessibility modifiers

UML Diagrams

The UML diagrams of the **Payable** class hierarchy are shown on the next page.

Class Autoloading

To automatically load class files, you need to call the **spl_autoload_register** function to register the **camelCaseToUnderscore** function. The complete code is available in hands-on practices or labs you've completed. Copy and paste the code into the *autoloading*.php file and require the file in any script in which autoloading is desired.


(interface) Payable
+getPaymentAmount: float
+toString: void
Invoice
-part_number: String
-part_description: String
-quantity: integer
-price_per_item: float
-invoice_count: integer
+ __construct
+getPartNumber: integer
+getPartDescription: String
+getQuantity: integer
+getPricePerItem: float
+getPaymentAmount: float
+toString: void
+getInvoiceCount: integer
(abstract) Employee
-person: Person
-ssn: String
-employee_count: integer
+ __construct
+getPerson: Person
+getSSN: String
+getEmployeeCount: integer
+toString: void
Person
-first_name: String
-last_name: String
+ __construct
+getFirstName: String
+getLastName: String
+toString: void
SalariedEmployee
-weekly_salary: float
+ __construct
+getWeeklySalary: float
+getPaymentAmount: float
+toString: void
CommissionEmployee
-sales: integer
-commission_rate: float
+ __construct
+getSales: integer
+getCommissionRate: float
+ getPaymentAmount: float
+toString: void
HourlyEmployee
-wage: float
-hours: integer
+ __construct
+getWage: float
+getHours: integer
+ getPaymentAmount: float
+toString: void
BasePlusCommissionEmployee
-base_salary: float
+__construct
+getBaseSalary: float
+getPaymentAmount: float
+toString: void

Code style guidelines:

1. Code style is as important as the code itself. Code style includes enough comments, adequate space between code blocks, and proper indentation, etc. Read details and view sample code from http://pear.php.net/manual/en/standards.php.
2. Please provide sufficient comments in your source code. Source code is a language for people, not just for computers. Comment as you go and don't wait for later. Ask yourself: "How will the next person know that?" Commenting code shows your professionalism, but also helps your grader understand your code.
3. Every class file should contain a header in this format:

```
/*
* Author: your name
* Date: today's date
* Name: file name
* Description: short paragraph that explains what the class is for
*/
```

4. Indent your code. Leave enough space between code blocks. You can always use Reformat feature in PhpStorm to automatically format your code.

Turning in your lab

Each group needs to turn in one copy. Provide names of all members in the team in Canvas. Your work will be evaluated on completeness and correctness. Thoroughly test your code before you turn it in. It is your responsibility to ensure you turn in the correct files. You will NOT receive any credit if you turn in the wrong files whether or not you've completed the lab.

1. Zip the entire **Lab07** folder and save it as *Lab07.zip*.
2. Upload the *Lab07.zip* file in Canvas before the lab's deadline.

Grading rubric

Your TAs will assess your lab according to the following grading rubric. You should very closely follow the instructions in this handout when working on the lab. Small deviations may be fine, but you should avoid large deviations. You will not receive credits if your deviation does not satisfy an item of the grading rubric. Whether a deviation is small or large and whether it satisfies the requirements are at your TAs' discretion. For the convenience of grading, the total point of this lab is 60. To calculate each member's score, divide the project score by 3. Here is the breakdown of the scoring:

Creating classes and interfaces (50 points)

Activities	**Points**

The **Payable** interface	4
The **Invoice** class	6
The **Employee** class	6
The **Person** class	6
The **HourlyEmployee** class	4
The **SalariedEmployee** class	4
The **CommissionEmployee** class	4
The **BasePlusCommissionEmployee** class	4
The *test_payable.php* file	12

Programming style (10 points)

Activities	**Points**
Comment your code	6
Use white spaces to separate code sections	2
Indent and line up code	2