Lab Assignment #3

Due Date: **Midnight Sunday 13th October 2019** **Marks/Weightage: 30/7%**

Purpose: The purpose of this Lab assignment is to:

- Practice the use of Inheritance, Polymorphism and Exception Handling

References: Read the course's text "Java How to program, 11th edition Early Objects", **chapters 8 to 11** and the lecture notes/ppts. This material provides the necessary information that you need to complete the exercises.

Instructions: Be sure to read the following general instructions carefully:

This lab should be completed individually by all the students. You will have to demonstrate your solution in a scheduled lab session and submitting the assignment **through drop box link on e-Centennial**.

>> At the start, you must name your **Eclipse work space** according to the following rule:

FirstName_LastName_SectionNumber_COMP228_Labnumber

For Example: ***John_Smith_Sec006_COMP228_Lab03 (say if your section number is 006)***

>> And after that your **project name** should be as follows:

FirstName_LastName_SectionNumber _Labnumber

For Example: ***John_Smith_Sec006_Lab03***

>>Each exercise should be placed in a separate package named as firstname_last-name_*exercise1*, firstname_last-name_*exercise2* etc.

>> After you complete, exit eclipse and go to workspace folder, zip it up and you will get the following zip file.

FirstName_LastName_SectionNumber_COMP228_Labnumber.zip

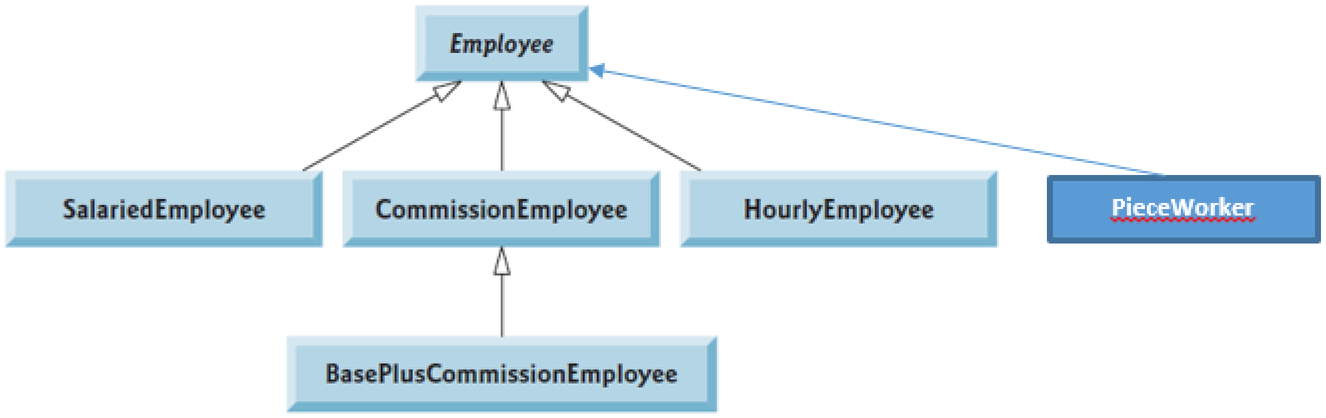
Example: **John_Smith_Sec006_COMP228_Lab02.zip** *(if your section is 006..)*

>> Apply the naming conventions for variables, methods, classes, and packages:

- *variable names* start with a *lowercase* character for the first word and uppercase for every other word
- *classes* start with an *uppercase* character of every word
- **packages** use only *lowercase* characters
- *methods* start with a *lowercase* character for the first word and uppercase for every other word

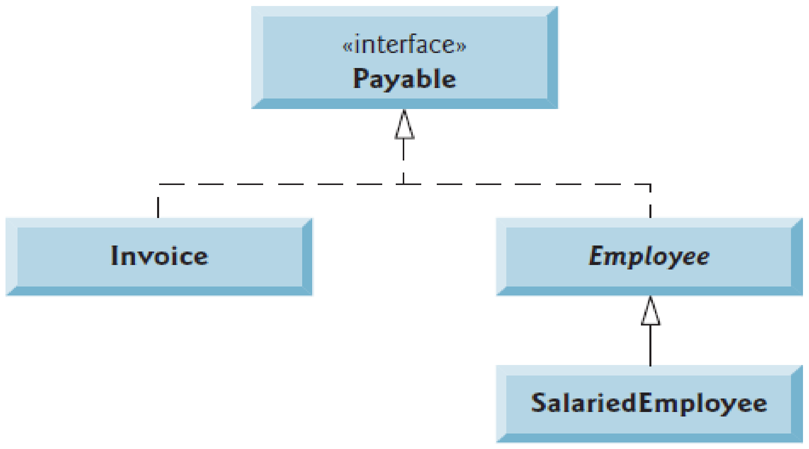
Note: Late submissions are accepted until up to three days past due date with 25% deductions. After that no submission will be considered.

Exercise #1: **(Payroll System Modification)** **[10 marks]**

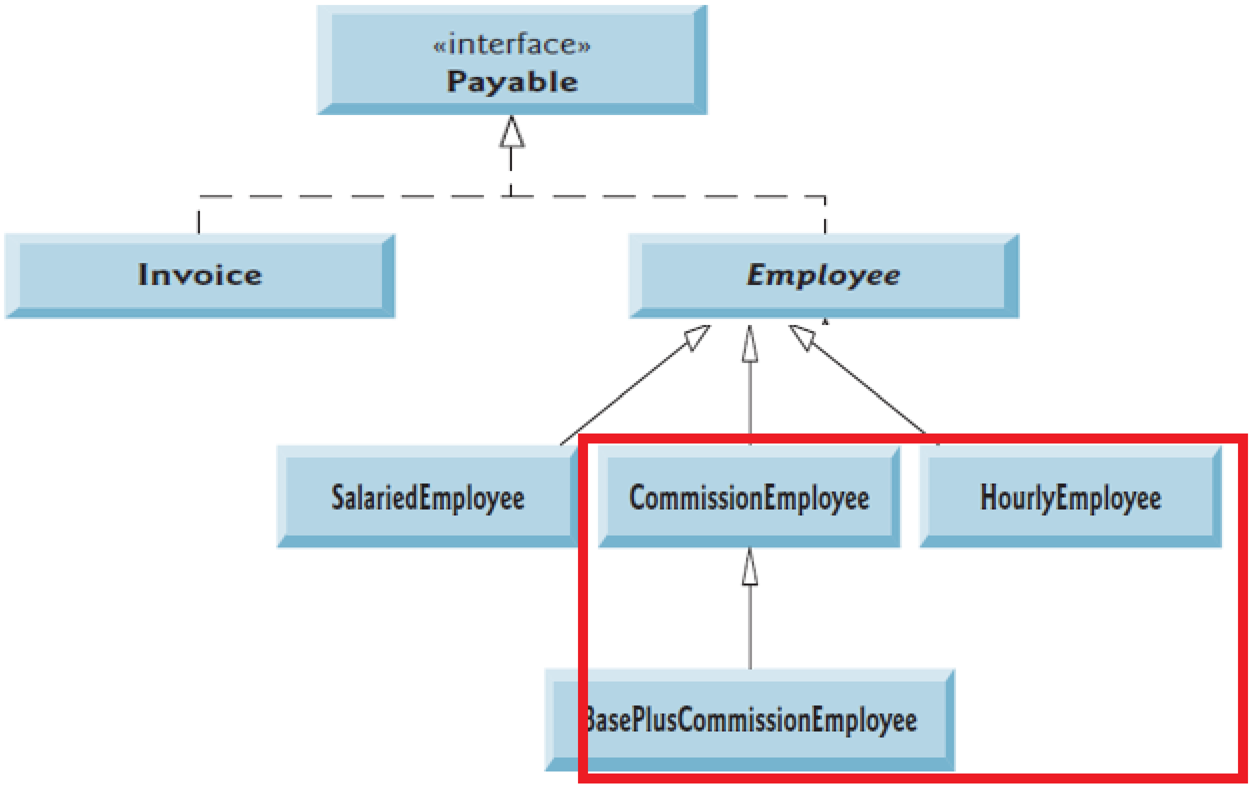


Modify the above payroll system which was implemented in the lab class, to include an additional Employee subclass **PieceWorker** that represents an employee whose pay is based on the number of pieces of merchandise produced.
Class PieceWorker should contain private instance variables wage (to store the employee's wage per piece) and pieces (to store the number of pieces produced).
Provide a concrete implementation of method **earning**s() in class PieceWorker that calculates the employee's earnings by multiplying the number of pieces produced by the wage per piece.
In the driver class (EmployeeTest.java), create an array of Employee variables to store references to objects of each concrete class in the new Employee hierarchy. For each Employee type, display its String representation and earnings.

Exercise #2: (**Accounts Payable System Modification**) **[10 marks]**
Following **Accounts Payable System** is implemented previously in the lab.



[Previously implemented]



[Three classes to be added as shown in the box and you can modify them so as to fit them in the above hierarchy]

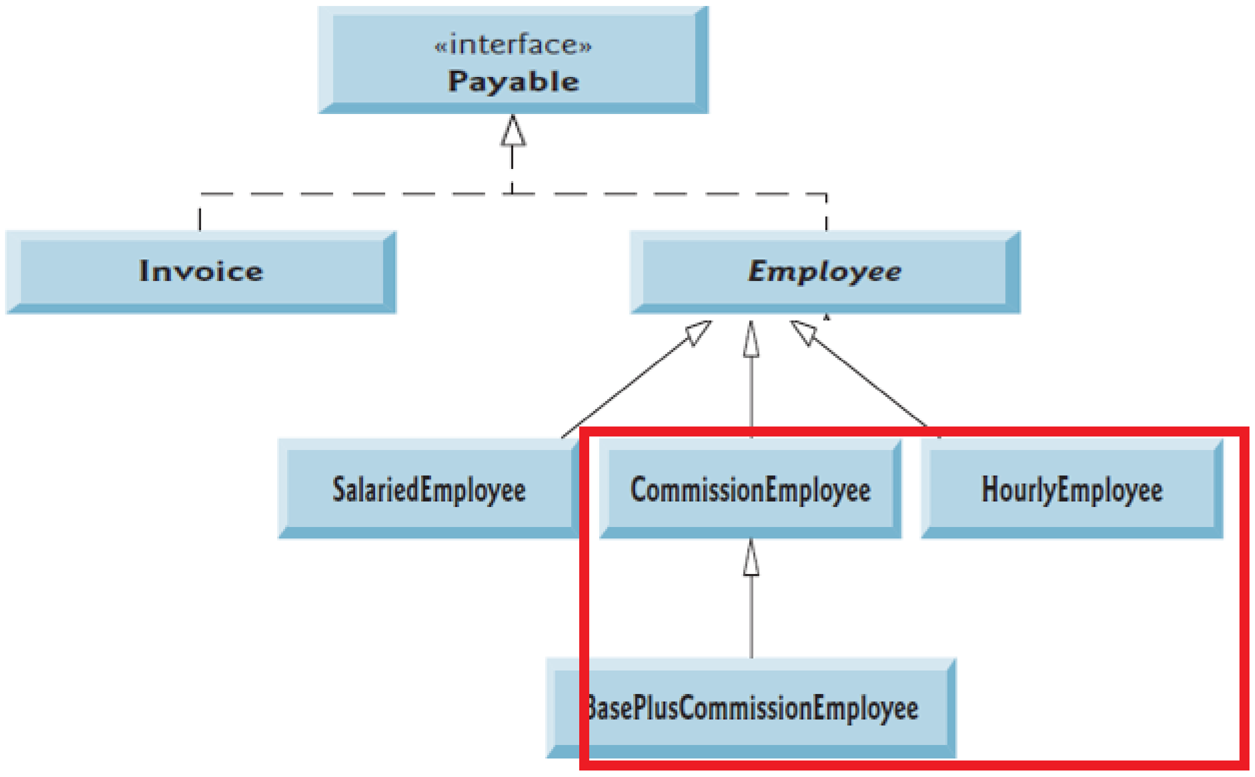
In this exercise, you need to modify the above **Accounts Payable System** application (covered in the class) to include the complete functionality of the payroll application
The application should still process two Invoice objects, but now it should process one object of each of the four Employee subclasses as shown above.
If the object currently being processed is a Base-PlusCommissionEmployee, the application should increase the BasePlusCommissionEmployee's base salary by 10%. Finally, the application should output the payment amount for each object.
Complete the following steps to create the new modified/updated application:
a) Modify classes HourlyEmployee and CommissionEmployee to place them in the Payable hierarchy as subclasses of the version of Employee that implements Payable. *[Hint: Change the name of method **earnings** to **getPaymentAmount** in each subclass so that the class satisfies its inherited contract with interface Payable.]*

b) Modify class BasePlusCommissionEmployee in such a way that it extends the version of class CommissionEmployee created in part (a).

c) Modify driver class PayableInterfaceTest to polymorphically process two Invoices, one SalariedEmployee, one HourlyEmployee, one CommissionEmployee and one Base-PlusCommissionEmployee.
First output/display a String representation of each Payable object.
Next, if an object is of the type BasePlusCommissionEmployee, increase its base salary by 10% and cal

If the object currently being processed is a HourlyEmployee, the application should increase the HourlyEmployee's hourly rate by 2.00 dollars. Finally, the application should output the payment amount for each object.

Exercise #3: (Accounts Payable System Modification)
In the **Accounts Payable System,** do the modification as explained below *[10 marks]*



[Three classes to be added as shown in the box and but you CAN NOT modify them. They need to be added as it as they are in Ex#1.0]

[In this exercise refer above figure, three classes - CommissionEmployee, BasePlusCommissionEmployee and HourlyEmployee need to be added without modifying them. They should be exactly as they are in Ex 1.0]

It's possible to include the functionality of the payroll application of **Exercise #2** in the accounts payable application without modifying Employee subclasses SalariedEmployee, HourlyEmployee, CommissionEmployee or BasePlusCommission-Emplyee.
*[Hint: To do so, you can modify class Employee to implement interface Payable and declare method getPaymentAmount to invoke method **earnings**. Method **getPaymentAmount** would then be inherited by the subclasses in the Employee hierarchy]*
When getPaymentAmount is called for a particular subclass object, it polymorphically invokes the appropriate earnings method for that subclass. Re-implement Exercise 2.0 using the original Employee hierarchy from the payroll application of Modify class Employee as described in this exercise, and do not modify any of class Employee's subclasses.