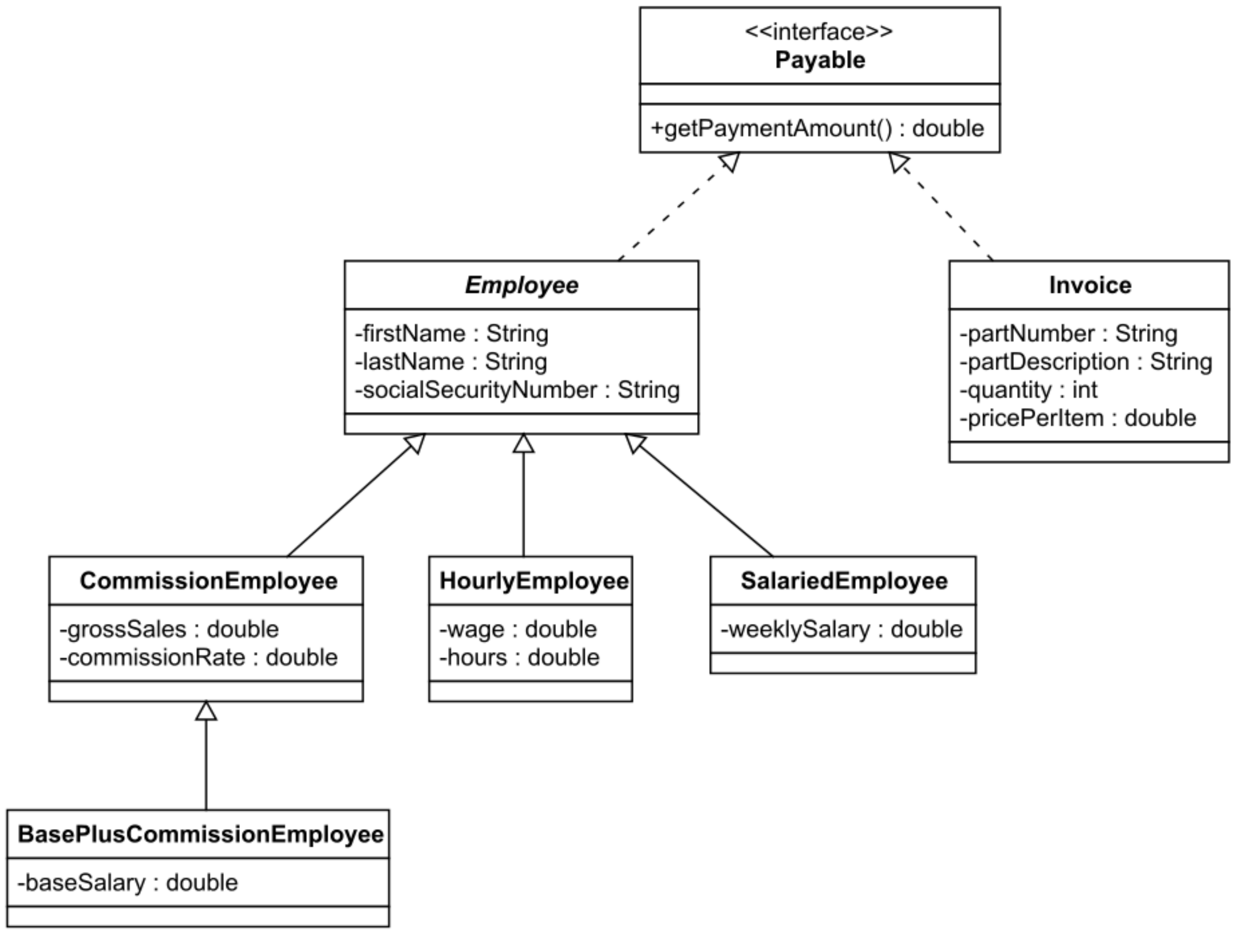
Seneca College
Applied Arts & Technology
SCHOOL OF COMPUTER STUDIES

JAC444 Submission date: June 21, 2022

Workshop 4

Description:
The following workshop lets you practice basic java coding techniques, creating classes, methods, using arrays, inheritance, polymorphism, Exceptional Handling.

Employee and Payable Case Study:



The Payable Interface

An *interface* declares one or more methods.

The Invoice and Employee Classes

Both the **Employee class** and **Invoice class** implements Payable interface. This allows **Employee** objects and **Invoice** objects to be processed polymorphically as Payable objects.

The Invoice class is a basic class with

- *instance variables*
- A *constructor*
- *getters* and *setters*
- A *toString* method – returns String representation of Invoice Object.
- An overridden *getPaymentAmount* method – returns the cost of the invoice

Note: *toString* and *getPaymentAmount* use the get methods rather than directly accessing the variables.

The Employee class

- Should be an *abstract class*. You cannot create an object whose class is Employee. For example:

```
Employee employee007 = new Employee("James", "Bond", "007-00-7007");
```

  would cause a runtime error.

- *instance variables*
- *getters* and *setters*
- A *constructor* that initialize the variables of the class
- A *toString* method – returns String representation of Employee Object.

<u>Subclasses of Employee</u>

Each of the subclasses have the following features:

- It extends the Employee class or a subclass of Employee.
- The parameters of its constructor include values to initialize all the instance variables of the class and its superclasses.
- The first line of code in the constructor calls the constructor of the immediate superclass.
- The toString method combines the result of the toString method of the superclass with additional information.
- The constructor, getPaymentAmount and toString methods use the getters and setters to get and set the instance variables.

- The setters should also include code to validate (or at least partially validate) the new values.

CommissionEmployee Class

Commission employees are paid a percentage of their sales

- *instance variabes*
    - grossSales – (throw an exception when <= 0.0)
    - commissionRate – (throw an exception when rate is not between 0.0 to 1.0)
- *getters* and *setters*
- A *constructor*
- A *toString* method – Overrides *toString* method in class Employee and returns String representation of CommissionEmployee Object.

HourlyEmployee Class

Hourly employees are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) for all hours worked in excess of 40 hours.

- *instance variabes*
    - wage – (throw an exception when <= 0.0)
    - hours – (throw an exception when hours is not between 0.0 to 168.0)
- *getters* and *setters*
- A *constructor*
- A *toString* method – Overrides *toString* method in class Employee and returns String representation of HourlyEmployee Object.

SalariedEmployee Class

Salaried employees are paid a fixed weekly salary regardless of the number of hours worked

- *instance variabes*
    - weeklysalary – (throw an exception when <= 0.0)
- *getters* and *setters*
- A *constructor*
- A *toString* method – Overrides *toString* method in class Employee and returns String representation of SalariedEmployee Object.

BasePlusCommissionEmployee Class

Base-salaried commission employees receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salaries

- *instance variabes*
    - baseSalary – (throw an exception when <= 0.0)
- *getters* and *setters*
- A *constructor*
- A *toString* method – Overrides *toString* method in class Employee and returns String representation of BasePlusCommissionEmployee Object.

Testing (main method):

Payroll System Test

- Polymorphically process
    - *Two Inovices*
    - *One SalariedEmployee*
    - *One HourlyEmployee*
    - *One CommissionEmployee*
    - *One BasePlusCommissionEmployee*

  First output a String representation of each Payable object. Next, if an object is a *BasePlusCommissionEmployee*, increase its base salary by 10%. Finally output the payment amount for each Payable object.

Continue to next page…

Workshop Header

```
/*********************************************
Workshop #
Course:<subject type> - Semester
Last Name:<student last name>
First Name:<student first name>
ID:<student ID>
Section:<section name>
This assignment represents my own work in accordance with Seneca Academic Policy.
Signature
Date:<submission date>
*********************************************/
```

Code Submission Criteria:

Please note that you should have:

- Appropriate indentation.
- Proper file structure
- Follow java naming convention
- Document all the classes properly using JavaDoc
- JavaDoc should be generated properly in the project
- Do Not have any debug/ useless code and/ or files in the assignment

Deliverables and Important Notes:

All these deliverables are supposed to be uploaded on the blackboard once done.

- You are supposed to create **video with voice**/ **detailed document** of your running solution. **(50%)**
  - Screen Video captured file should state your last name and id, like Ali_123456.mp4 (or whatever the extension of the file is)

    <u>OR</u>
  - Detailed document should include screen shots of your output, have your name and id on the top of the file and save the file with your last name and id, like Ali_123456.docx (or whatever the extension of the file is)
- A word/ text file which will reflect on learning of your concepts in this workshop. **(20%)**
  - Should state your Full name and Id on the top of the file and save the file with your last name and id, like Ali_123456.txt

- JavaDocs must be used for proper documentation of each task. **(15%)**
- Submission of working code. **(15%)**
    - Make sure your follow the “**Code Submission Criteria”** mentioned above.
    - You should zip your whole working project to a file named after your Last Name followed by the first 3 digits of your student ID. For example, **Ali123.**zip.
- Your marks will be deducted according to what is missing from the above-mentioned submission details.
- Late submissions would result in additional 10% penalties for each day or part of it.

Remember that you are encouraged to talk to each other, to the instructor, or to anyone else about any of the assignments, but the final solution may not be copied from the assignments, but the final solution may not be copied from any