# Seneca College

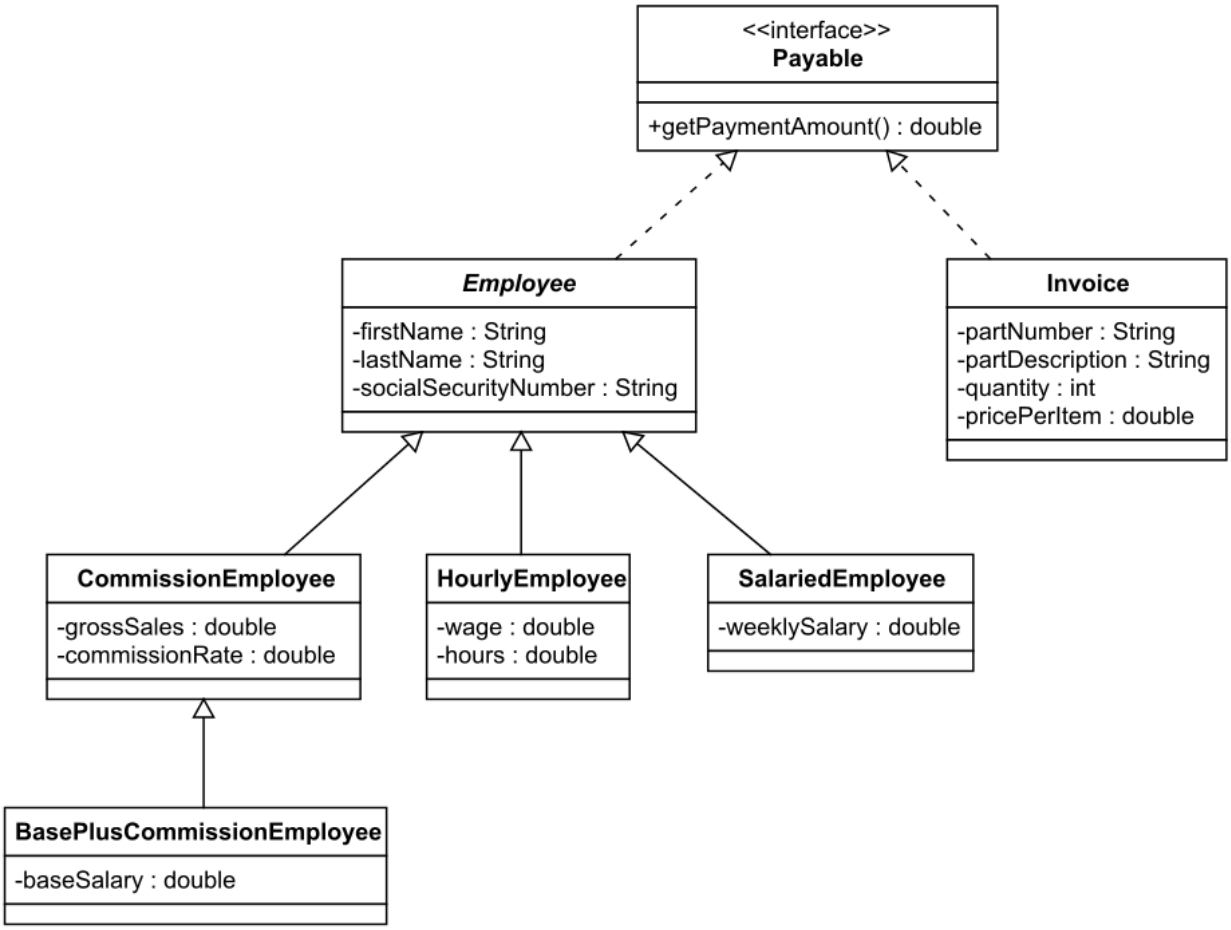
Applied Arts & Technology
SCHOOL OF COMPUTER STUDIES

**JAC444** **Submission date:** **Feb 14, 2020**

## Workshop 3

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

# Employee and Payable Case Study:



## The Payable Interface

An *interface* declares one or more methods but does not implement the methods. For example, the **Payable interface** should declare the *getPaymentAmount* method.

## The Invoice and Employee Classes

Both the **Employee class** and **Invoice class** implements Payable interface, which is a promise to implement the *getPaymentAmount* method. This allows **Employee** objects and **Invoice** objects to be processed polymorphically as Payable objects because they all implement the *getPaymentAmount* method.

## **The Invoice class** is a basic class with

- *instance variables*
  - part – the part number
  - description – the part description
  - count – how many
  - price – cost per item
- A *toString* method – returns String representation of Invoice Object.
- A *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*.
- *instance variables.*
  - `first` - first name of employee
  - `last` - last name of employee
  - `ssn` - SSN of employee
- A *toString* method – returns String representation of Employee Object.

## Subclasses of Employee

Each of the subclasses have the following features:

- 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. Note the use of the super keyword. This is good practice in inheritance hierarchies.
- The *getPaymentAmount* method is implemented, fulfilling the promise made by the Employee class.

- The toString method combines the result of the toString method of the superclass with additional information. Again, note the use of the super keyword.
- The constructor, getPaymentAmount and toString methods use the getters and setters to get and set the instance variables. One reason is that instance variables from the superclasses are private, so there is no other way to access and modify these variables. The other reason is that the setters 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 – gross sales of employee (should satisfy the condition <= 0.0, use exceptional handling)
    - commissionRate – commission rate of employee (rate should be between 0.0 to 1.0, use exceptional handling)
- 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 – hourly wage of employee (should satisfy the condition <= 0.0, use exceptional handling)
    - hours – number of hours worked by employee (hours should be between 0.0 to 168.0, use exceptional handling)
- 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 – hourly wage of employee (should satisfy the condition <= 0.0, use exceptional handling)
- 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 – base salary of employee (should satisfy the condition <= 0.0, use exceptional handling)
- A *toString* method – Overrides *toString* method in class Employee and returns String representation of BasePlusCommissionEmployee Object.

## Testing (main method):

The PayrollSystemTest Main Method:

- The main method for *PayrollSystemTest.java* should uses the **Employee class** and its subclasses.
- It should first create four objects, one for each subclass of Employee.

  **Note**: the constructors have different numbers of parameters.

  For example, a BasePlusCommissionEmployee object will need information for six instance variables (one in BasePlusCommissionEmployee class and five in superclasses). Also, the BasePlusCommissionEmployee constructor needs some way to initialize the private variables from the superclasses.

Sample of an object creation

SalariedEmployee salariedEmployee = **new** SalariedEmployee("John", "Smith","111-11-1111", 800.00);

- Next your test should print information about each object individually, using *toString*.

Sample

Employees processed individually:

Employee: Kyle Wright; SSN: 123-123-123; Salary {weekly salary:$500.00}

earned: $500.00

**Note:** You must use **System.out.printf** for this, Learn about Formatting with printf()

- Next your test program should print information about each object polymorphically. (You have four objects in the Array that you created when the test start so you have to go over each object and print)

  Sample output:

  Employee: Jack Barney; SSN: 123-123-123; Commission {Gross Sales : 25000.0, Commission Rate: 0.12 }

  earned: $3000.00

- The last for loop illustrates how to find out the specific class for each object.

  Sample:

  Employee: Jack Barney is of class ca.senecacollege.ict.CommissionEmployee

***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
- Do Not have any debug/ useless code and/ or files in the assignment
- Check your inputs if the user is not entering garbage inputs.
- Use exceptional handling or other methods to let the user know if the inputs are incorrect.

## Deliverables and Important Notes:

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

- You are supposed to create video/ record voice/ detailed document of your running solution. **(40%)**
  - Screen Video captured file should state your last name and id, like Ali_123456.mp4 (or whatever the extension of the file is)
  - Record voice clip should also include a separate word file with the screen shots of your program's output, state your last name and id, like Ali_123456.mp3 (or whatever the extension of the file is)
  - 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. (Also include the instructions on how to run your code. Which is only required if you have any special instructions for me on how to run your code.) **(30%)**
    - 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
- Submission of working code. **(30%)**
    - 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 any source.