## Lab-4- Inheritance and Interface(Must solve all the problems using Polymorphism)

**Reading:** Do the reading of Interview questions for Lesson-5 from Sakai\Resources\FPP Interview Questions.pdf

## Problem 1: Product Pricing System

**Classes to be implemented:**

- Product(Regular class)
- Electronics
- Furniture
- Clothing
- TestClass

**Details:**

- The Product class has fields for productName and price and methods to get and set the price.
- Electronics, Furniture, and Clothing classes inherit from Product and may have additional fields.
- Clothing class has additional fields brand and discount value(eg: 5 or 10 or 20)
- Furniture class has additional fields material(wood, plastic, steel) and Shipping cost.
- Electronics class has additional fields warranty (in months), warranty Cost.

**Override the getPrice() behavior in the sub classes by following procedure below,**

- **Clothing class:** Apply the discount percentage to the original price and return discounted price.
- **Electronics class:** Add the warranty cost to the original price and return the updated price.
- **Furniture class:** Add the shipping cost to the original price and return the updated price.

**Do the following in the TestClass with the main().**

- Create an array of type Product, Store 5 different objects.
- Loop through the objects and print the status of the objects(overriding toString()).
- Create a static method that takes the array of products and returns the sum of all the products. Inside this method deals the logic to avoid NPE.

  ```
  public static double sumProducts(Product[] col) {
  }
  ```

- Print the sum on the console.

**Problem 2: Smart Home Sensors - Interface**

1. Create a Sensor Interface with the following behaviors
    - getSensorType() – Return the name of the Sensor
    - getReading() – Return the sensor data in double
    - getLocation() – Return the Home location where sensor deployed. [ Garden, Kitchen, etc.,]
    - getLastUpdated() – Return the system current time.
    - String performAction(); - Return the action taken based on the Sensor alert
2. Create Classes LightSensor, SoundSensor, TemparatureSensor implements Sensor. Add the common attributes such as location and lastupdated in each class.
3. LightSensor class has additional field lightlevel.
4. SoundSensor class has additional field soundlevel.
5. TemparatureSensor has additional field temperature.
6. If the user invoke the getLastUpdated() method return the current time and update the instance field lastupdated with the current time.
7. Do the below logic in each subclass for the performAction()
    - In LightSensor, if the lightlevel reaches below 100 return “an alert to turn on the light”, else “ Light is sufficient”
    - In SoundSensor, if the sound level reaches above 70 return “an alert to turn on noise cancellation”, else “ Sound is within normal range”
    - In TemparatureSensor, if the temperature reaches above 30 return “an alert to turn on the AC”, if it reaches below 18 return “an alert to turn on the Heater” otherwise “ Temperature is in normal range”
8. Write a SensorTest class with the main() method.
    a. Create an array of type Sensor, Store 5 different objects.
    b. Loop through the objects and print the status of the objects. (Override toString())
    c. Print the getLastUpdated() output shows the time in HH:MM am/pm

**Sample output:**

Sensor Type: Temperature
Reading: 23.5
Location: Living Room
Last Updated: 03:55 PM
Action: Temperature is within the normal range.

Sensor Type: Light
Reading: 80.0

Location: Garden
Last Updated: 03:55 PM
Action: Light level is too low! Turning on the lights.

Sensor Type: Sound
Reading: 65.0
Location: Bedroom
Last Updated: 03:55 PM
Action: Sound level is within the normal range.

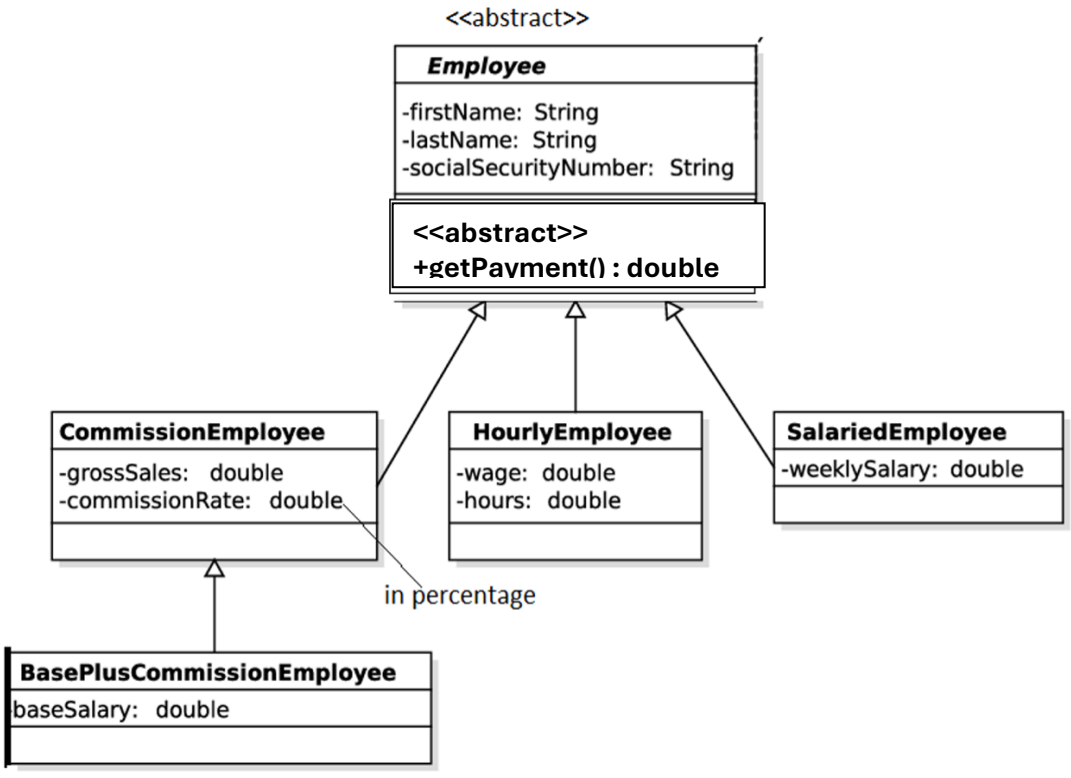
## Problem – 3 – Employee Salary Management – Abstract class

Write a Java code for the given UML Diagram.

1. Provide necessary getters and setters
2. Provide necessary constructors to initialize values in all the classes.
3. Override the toString() method to display the current status of the objects
4. Write a driver class to test by creating an array of five objects for various employee categories.
5. Create a static method that takes the array of Employees, and the salary. Return an array of Employees whose getPayment() < salary. Inside this method deal the logic to avoid NPE. Your result array does not contain null employees.

   **public static Employee[] findSalaryList (Employee[] col, double salary) {**
   **}**

6. Print the results on the console.



**Hints:** The getPayment() return  double values as mentioned below according to the specific class object.

1. CommissionEmployee : grossSales * CommisionRate
2. BasePlusCommisionEmployee : baseSalary + (grossSales * CommisionRate)
3. HourlyEmployee : wage * hours
4. SalariedEmployee : weeklySalary

## Problem 4: Understanding Non-OO code to OO Code

You have given code package prob4.nonoo as a compressed file on your Sakai assignments. The problem solved using Non-OO way. Your job is to convert the Non-OO code to OO code using Polymorphism by implementing the suitable interface with the suitable method declaration.

Refer Demo Packages: closedcurvebad and closedcurvegood.

OO – Object Oriented