Lab-4- Inheritance and Interface

Problem 1: Product Pricing System

Classes to be implemented:

- Product
- 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 percentage.
- Furniture class has additional fields material 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.

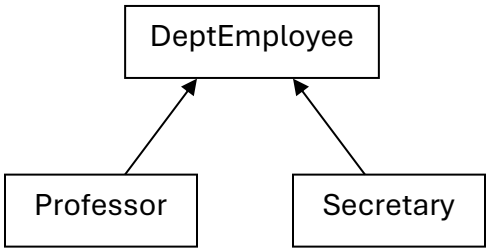
Problem 2: Employee Management

A university department consists of professors and secretaries. Each professor and each secretary has a name, a salary, and a hire date. Use inheritance and polymorphism to create an application that represents the department and its professors and secretaries as objects, and provides a test class that creates 3 professors and 2 secretaries, and then outputs the combined total of all of their salaries and then print the average salary on the console.

Start by creating classes

Professor
Secretary
DeptEmployee

having the following relationship:



Place instance fields and corresponding accessor/mutator methods in DeptEmployee to represent name and hire date (as a type of LocalDate)(do not create getters or setters for salary). Do not put these fields in either the Professor or Secretary class. Also place in the Professor class an int field numberOfPublications, with corresponding getter and setter methods. Place in the Secretary class a double field overtimeHours, also with corresponding getter and setter methods.

Place a computeSalary method in DeptEmployee which simply returns the value stored in salary. Override the computeSalary method in Secretary so that the return value is the sum of the salary value *plus* 12 times the number of overtime hours.

Then in the main method of a class named Main, create three instances of Professor and two instances of Secretary (you can invent the values to pass into the constructor). Finally, create an array of department employees:

```
DeptEmployee[] department = new DeptEmployee[5]
```

and then populate the array with the Professor and Secretary instances you have just created. Then ask the user if he wishes to see the sum of all Professor and Secretary salary in the department. If the user responds "Y", then loop through the department array and polymorphically read, and sum, all salaries, and output the result to the console. Also print the average salary of the department.

Problem 3: 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 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.

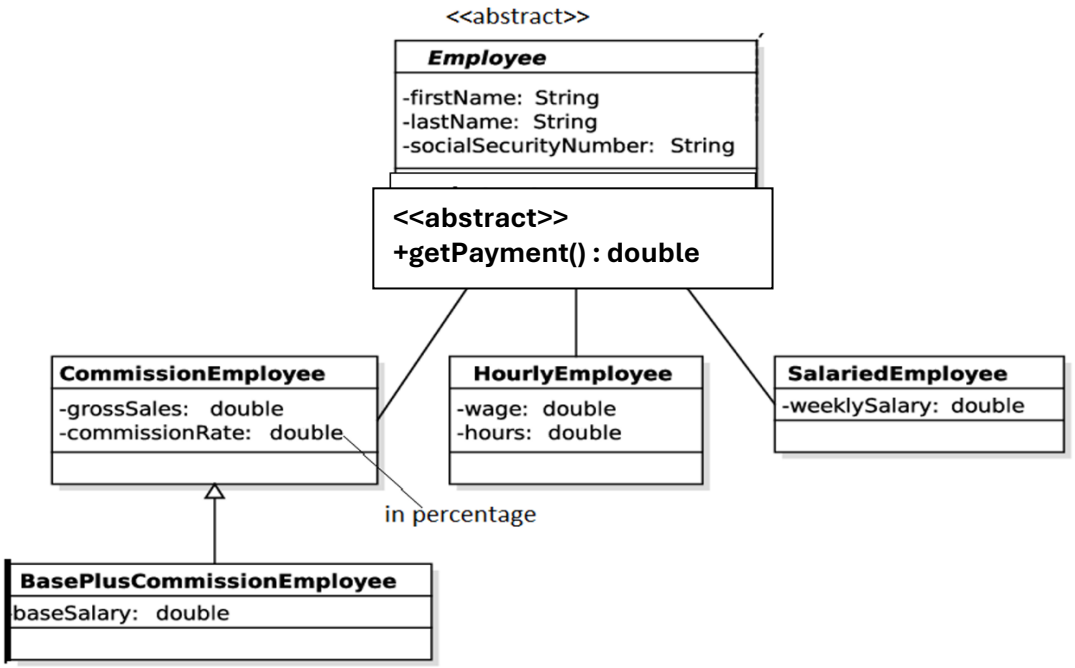
Problem – 4 – Employee Salary Management – Abstract class

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 return the who got the maximum salary. Inside this method deal the logic to avoid NPE.

   ```
   public static Employee findMaxSalary (Employee[] col) {
   }
   ```

6. Print the max salary employee detail 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

Note: Due to the Thanksgiving break, Problems 5 and 6 are optional. Submission is not required, but I encourage you to practice them if you find the time.

Problem 5: Understanding Non-OO code to OO Code

You have given code package prob5.nonoo. 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.

Problem 6: Understanding of overriding equals() with the Inheritance relationship.

To give a chance to bring your creativity, you have to come with the idea of Parent and a Child class. Similar to Person and PersonWithJob example.

Refer Slide 86-97. Lectures slides discussed with Instanceof Strategy, Same class strategy, Composition.

Refer Demo Package equals.equals, case1, case 2 and case3.

Apply three cases with problem you designed.