**OOP Lab 5**

|  |  |
| --- | --- |
| Due Date: | **Oct 4, 23 : 59** |

* **Submit your assignment using the following file format:**

LabNumber\_RoomNumber\_StudentName\_Student\_ID.zip

Example: Lab5\_328\_Hongkildong\_201620505.zip

* The zip file will contain **two types of** files, namely:

1. **report file** with file format **“Report\_Lab number**” (eg. report\_5) to answer theory questions and to write the screen shot that display the output of your program
2. Source code file that contains codes of classes to answer programming questions.
3. **Objectives**

1. Learning the concept of an **implicit parameter** called **“this”** reference keyword

2. Learning the concept of **Overloaded constructor** using “**this**”refrence keyword.

3. Learning and utilizing the concept of **composition (HAS-A relationship between two** classes).

4. Learning and utilizing the use of “**final**” member variable,

5. Learning the similarity and difference between inheritance and polymorphism, and how to utilize them.

6. Learning the similarity and difference between method overriding (dynamic binding) and method **overloading (static binding) in terms of polymorphism.**

**7. Learning similarity and difference between of concrete class and abstract class and how to utilize them**

**8. Learning the similarity and difference between abstract class and interface and how to use them**.

**EXERCISE 3’s problem numbers are scrambled so I reordered them. Please note.**

**Original ->A B Q(Question? It may be cause) R S T U**

**Changed -> A B C D E F**

**II. Exercises (31 points)**

1. Answer the following questions about “**ThisTest**” class and “**SimpleTime**” class (**Fig.8.4).**
   1. “**this.toUniversalString**()” and “**toUniversalString**()” within **buildString(**) method have the same results. Why they have the same result? Explain only the reason (**2 point**)

**-> It is used to call instance variables in itself.**

* 1. 개체이(가) 표시된 사진

     자동 생성된 설명Remove all “**this”** reference variables in **SimpleTime** class. Re-write the code to get the same result as shown in (A) (include your code and captured result screen)(**2 point**)

2. Answer the questions about **Account** class of **Fig 3.8** and **AccountTest class** of **Fig 3.9.**

* 1. Add the following **constructor** in the **Account class**. This constructor should call an **original** **constructor** of the Account class (include the code of the added constructor code)(**2pt**).

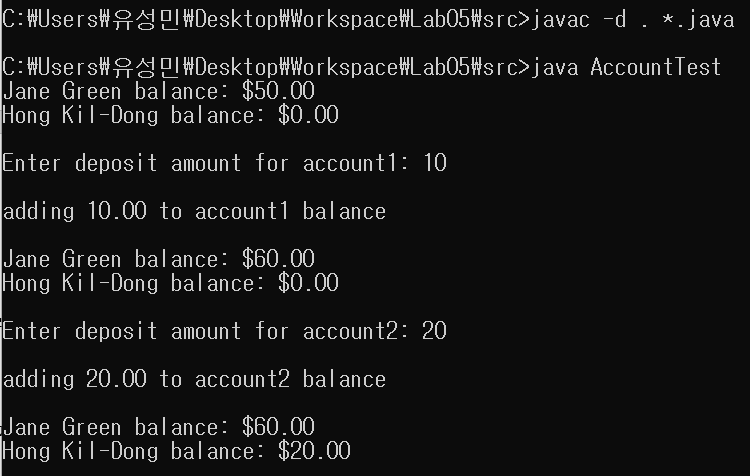
|  |
| --- |
| Account (String name); **//**  initializing **name** by the given String and balance by zero.  {  ………….  } |

스크린샷이(가) 표시된 사진

자동 생성된 설명

* 1. Using the code in (A), edit 10th line of **AccountTest class** as shown below, and then execute the program (including capture result screen) (**2 point**)

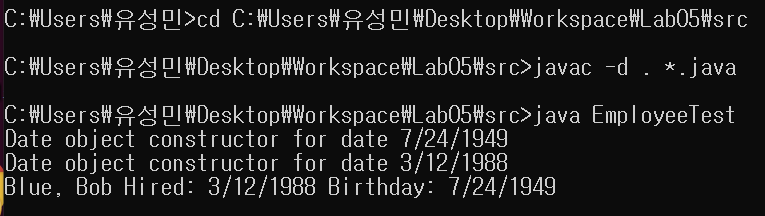
|  |
| --- |
| Account account2 = **new** Account("Hong Kil-Dong"); |

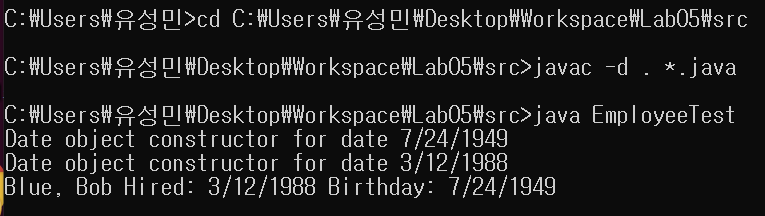


3. Answer the questions using **Date**, **Employee**, **EmployeeTest** classes in **Fig** **8.7 /8.8 / 8.9**.

A. Once “**birthDate**” and “**hireDate**” member fields of the Employeeclass of Fig 8.8 are initialized, you don’t need to re-initialize them again. Execute the program after adding “**final modifier**” on two member variables. **Check what results happens before and after**. Explain the reason (including result difference before and after, and reason) (**2 point**)

**-> The result does not change. Because the modifier “final” can’t affect to the value of variables(birthDate, hireDate). The if variables are initialized with “final” modifier, then we can’t reinitialize the values of variables.**

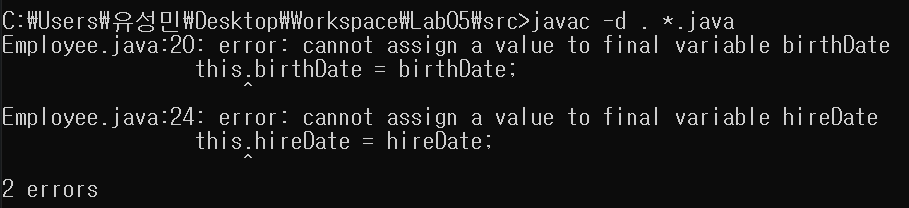
**<before>**

**<after>**

B. Once “birthDate” and “hireDate” member fields of the Employee class of Fig 8.8 are initialized, you don’t need to re-initialize them again. **In the Employee class**,

1) add “final modifier” on two member variables.

2) add “SetBirthDate() and SetHireDate() methods and try updates the initial values of “birtgDate” and “hireDate” fields by adding new values.

a) What happens when you try to implement the above two methods? (including your reason) (2 point)

**-> Variables with a final modifier are constants that cannot be changed. If you try to change this through a method, you will naturally get an error. However, it is possible to specify different values ​​for each instance through the constructor.**

|  |
| --- |
| **Public Employee**() { } |

b) After running the previous program in (A), add the following constructor in the ***Employee*** class. What result you get? Explain your reason). (2 point)

**-> As I said at question (A). we used the “final” modifier so we can’t reinitialize them.**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

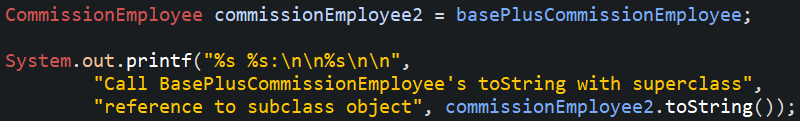
**C. Answer the following questions using CommissionEmployee (Fig 9.10), BasePlusCommissionEmployee (Fig 9.11) and PolymorphismTest (Fig 10.1).**

**a.** Find all places where polymorphism is used in Fig 10.1. (2 points).

**=>**

**1) CommissionEmployee commissionEmployee2 = basePlusCommissionEmployee;**

**‘commissionEmployee2’ is variable of superclass(CommissionEmployee Class), but it referernce the subclass(basePlusCommissionEmployee).**

**2) ‘comissionEmployee2.toString()’ => commissionEmployee2 is set like the screenshot , so it call the toString() method which is defined at basePlusCommissionEmployee class**

b. After line 34 in Fig 10.1, insert the following code and run the program. What happens? Why does that happen? (2 points)

|  |
| --- |
| BasePlusCommissionEmploye **bpce** = commissionEmployee2;  System.out.println("Salary = " + **bpce.getBaseSalary**()); |

**=> There are errors. Because the commissionEmployee2 is instance of superclass,**

**but bpce is instance of subclass. So it needs down-casting.**

c. Replace the code in B by the following code and run the program. What is different between program in A and program B ? (2 points)

|  |
| --- |
| BasePlusCommissionEmployee **bpce** = (BasePlusCommissionEmployee) commissionEmployee2;  System.out.println("Salary = " + **bpce.getBaseSalary**()); |

**=> It is down-casting. So the type of the object is changed to subclass. So is does not make error.**

d. Replace the code in C by the following code and run the program. What happens? Why does it happen? (2 points)

|  |
| --- |
| BasePlusCommissionEmployee **bpce** = (BasePlusCommissionEmployee) commissionEmployee;  System.out.println("Salary = " + **bpce.getBaseSalary**()); |

**스크린샷이(가) 표시된 사진

자동 생성된 설명=> The super class can’t be instance of subclass, so there is error.**

**D. Answer the questions after running Payroll system in Section 10.5 (Fig 10.4 ~ 10.9, Employee, SalariedEmployee, HourlyEmployee, CommissionEmployee, BasePlusCommissionEmploye, PayrollSystemTest).**

a. The **output** of the program has **two** different parts. Explain the **output** of each part (**2 points**)

**<Employee Class>**

**This class is an abstract class, so classes that extend this class should complete it. And variables are initialized with ‘private’ modifier. This class shows ‘firstName’, ‘lastName’, and ‘socialSecurityNumer’.**

**<salariedEmployee Class>**

**This class extends Employee Class. At the code, we set the person(John Smith, 111-11-1111) who gets weekly earnings. This class overrides the earnings method and this returns getWeeklySalary. So salariedEmployee Class shows ‘John’, ‘Smith’, ‘111-11-1111’ and John Smith’s weekly salary.**

**<HourlyEmployee Class>**

**This class also extends Employee Class. This class is for the person who works part-time job, so there is member variables(wage, hours). Hourly part-timer get money of wage\*hours, so the HourlyEmployee Class overrides earnings method to get the total earns of the person. So this class shows the person we set, wage, hours the person worked, and the total earned.**

**<CommissionEmployee Class>**

**This class also extends Employee Class and the class shows the salary of the person who get commissions of gross sales. So this class added its member variables(commissionRate, grossSales). it overrides the earnings method and the method calculate the commission rate \* gross sales. This class shows the person’s name and the person’s salary(commission rate \* gross sales).**

**<BasePlusCommissionEmployee Class>**

**This class extends CommissionEmployee Class. It’s function is to show the person’s commission of gross sales \* base salary, so the member variable baseSalary is added. And the class overrides the earnings method to return commission of gross sales(commission employee’s earnings) \* base salary. The class shows the person’s name, earnings(base salary \* super class’s earnings), and super class’s values’s of variables(commission rate, gross sales).**

**<PayrollSystemTest Class>**

**This class shows all of the results of other classes. At the code we set the each employee and made array to handle each class by saving class in a array space.**

**At result, this class shows all of the employee’s information. As you can see the result of the code, it shows the result of 10% increased base salary and earned money.**

b. Change the code in **Employee** class as follows,

* deleting “**abstract**” keyword at line **4:** **public class Employee**
* Change line **46** by the code**: public double earnings() { return 0.0; }**

After running the program, compare the results with previous (unchanged) program (**1 points**)

**=>**

**There is no changes at result. If Employee class is abstract class and earnings method was empty, then we should complete the earnings method at every classes which extend Employee class. But after changing the code, every classes that extend ‘Employee Class’ override the earnings method, so the results of before and after are same.**

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명problem D a) result.. b) result is same, so don’t attach b) result.**

**c. Before changing** and **after** changing in (B), add the following code to the **main** function and run it.

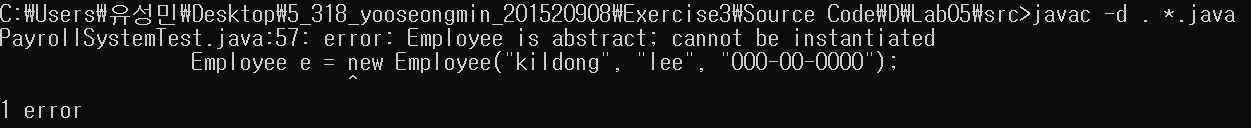
|  |
| --- |
| Employee e = **new** Employee("Kildong", "Lee", "000-00-0000"); |

What is the difference between the two? Explain the reason (2 points)

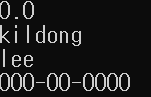
**<original code>**

**=>**

**There is error. It should override the earning method because Employee Class is abstract class and the earning method is not concrete.**

**<after deleting abstract keyword>**

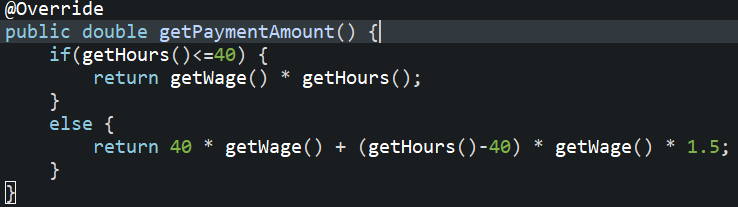
**= >**

**There is no error. We made Employee instance well and printed well.**

**E. Answer the questions after running Payroll system in Section 10.9 (Fig 10.11 ~ 10.15, Payable, Invoice, Employee, SalariedEmployee, PayableInterfaceTest).**

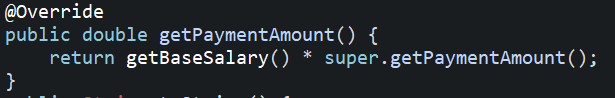
a. Modify the program by adding the following **subclasses** of the Employee class (see Fig.10.2 and Fig.10.10) (**2pt)**.

* **CommissionEmployee**
* **HourlyEmployee**
* **BasePlusCommissionEmployee**( Subclass of Commission Employee)

< **HourlyEmployee>**

사진이(가) 표시된 사진

자동 생성된 설명< **CommissionEmployee** >

< **BasePlusCommissionEmployee** >

**b. Modify the code** of the new classes in A **by** replacing their **earnings (**) methods using **getPaymentAmount()** method similar to the **SalariedEmployee class**. Modify also **PayableInterfaceTest** class by **expanding the array**, so that the class can create one object of each subclass and calculate the payment (**2 points**)

**텍스트이(가) 표시된 사진

자동 생성된 설명<array>**

**텍스트이(가) 표시된 사진

자동 생성된 설명<result>**

**F. Dog, Cat , and Sheep are all animals with their own unique cries. Answer the following questions.**

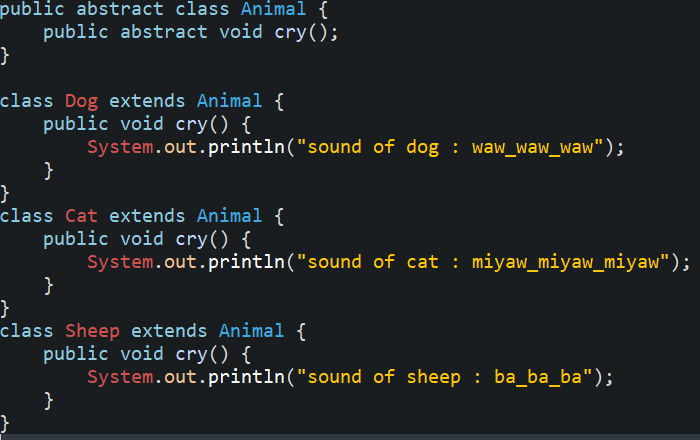
a. Define a class of each animal including at least **one method** as follows.

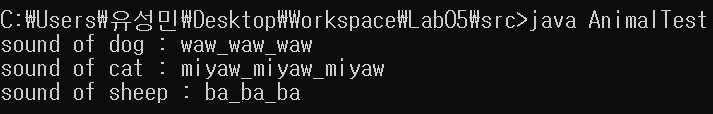
|  |
| --- |
| void **cry**()  {  ...  } |

However, the crying operation is replaced by printing a string corresponding to the crying sound (2point).

**Example:**  sound of dog: “waw waw waw

**Example:** sound of Cat: “ Miyaw Miyaw Miyaw”

**Example:** sound of sheep: “Ba Ba Ba ”

b. Run **AnimalTest** class that tests the crying of these animals using both non-polymorphism and **polymorphism approaches** (2 points)