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**

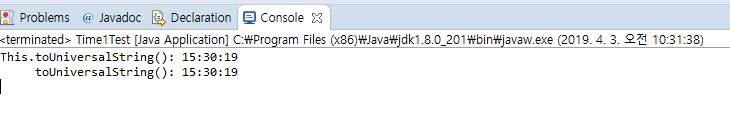
**8. Learning the similarity and difference between abstract class and interface and how to use them**.

**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**)

* toUniversalString() 메소드가 SimpleTime 클래스에 하나만 있기 때문에 this를 사용해서 지정해주어도 클래스내에 toUniversalString()이 하나 밖에 없기 때문에 차이점이 없다.
  1. Remove 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**)

스크린샷 첨부



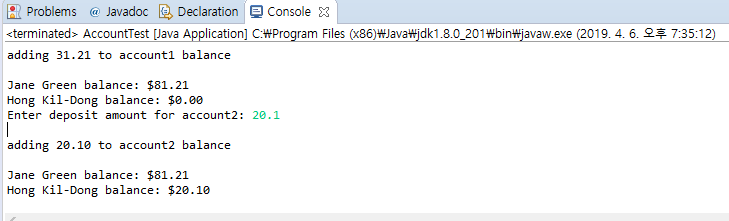
1. Answer the questions about **Account** class of **Fig 3.8** and **AccountTest class** of **Fig 3.9.**
   1. Add the following **constructor** in the **Account class**. This constructor should call an **original** **constructor** of the Account class (include the code of the added constructor code)(**2pt**).

|  |
| --- |
| Public Account(String name){  this.name = name;  this.balance = 0;  } |

* 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"); |

스크린샷 첨부 코드는 ex2에 첨부하였습니다.



1. Answer the questions using **Date**, **Employee**, **EmployeeTest** classes in **Fig** **8.7 /8.8 / 8.9**.
   1. 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**)

* 출력된 내용은 차이 없음 final을 붙인 변수는 그 생성자에 의해 한 번 초기화된 이후에는 수정될 수 없으므로 바꿀 수 없는 상태로 고정이된다.
  1. 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**,
* add “final modifier” on two member variables.
* Add “SetBirthDate() and SetHireDate() methods and try updates the initial values of “birtgDate” and “hireDate” fields by adding new values.

1. What happens when you try to implement the above two methods?

(including your reason) (2 point)

* 1. After running the previous program in (A), add the following constructor in the ***Employee*** class. What result you get? Explain your reason). (2 point)
* A의 값과 똑 같은 결과가 나온다. 하지만 만약에 EmployeeTest 클래스에서 인자를 받지 않는 생성자 Employee()를 사용한다면 birthDate와 hireDate의 값을 초기화하지 못할 것이다.

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

1. **Answer the following questions using CommissionEmployee (Fig 9.10), BasePlusCommissionEmployee (Fig 9.11) and PolymorphismTest (Fig 10.1).**
   1. Find all places where polymorphism is used in Fig 10.1. (2 points).

* Line 30~34이다.
  1. After line 34 in Fig 10.1, insert the following code and run the program. What happens? Why does that happen? (2 points)
* 컴파일이 정상적으로 되지 않아 실행이 되지 않는다. 이유는 BasePlusCommissionEmployee 클래스는 결국 commissionEmployee 클래스를 상속하기 때문이다. 이 경우는 참조변수가 자식클래스인데 인스턴스 타입이 부모 클래스이기 때문에 문법적으로 오류가 생겨서 컴파일이 되지 않았다.

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

* 1. 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)
* 문제 A의 경우는 참조변수가 자식클래스이지만 인스턴스 타입이 부모클래서여서 오류 였지만 C의 경우는 자식클래스의 참조타입으로 캐스팅이 되었기에 정상적으로 작동한다.

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

* 1. Replace the code in C by the following code and run the program. What happens? Why does it happen? (2 points)
* BasePlusCommissionEmployee bpce =

(BasePluseCommissionEmployee)commissionEmployee;

이 부분에서 오류가 떠서 컴파일이 불가능하다. 이유는 인스턴스 부모 타입에서는 자식 참조 타입으로는 캐스팅이 불가능하다.

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

1. **Answer the questions after running Payroll system in Section 10.5 (Fig 10.4 ~ 10.9, Employee, SalariedEmployee, HourlyEmployee, CommissionEmployee, BasePlusCommissionEmploye, PayrollSystemTest).**
   1. The **output** of the program has **two** different parts. Explain the **output** of each part (**2 points**)

* Employees processed individually Part : employee를 상속하는 S ~E, H ~ E, C ~ E 그리고 CommissionEmployee 클래스를 상속하는 B-S C M가 각각 자신의 참조변수와 인스턴스 타입을 동일하게 사용하여 출력하는 것이다.
* Employees processed polymorphically Part: Employee 클래스를 상속하는 salaried employee, hourty employee, commission employee와 commissionemployee 클래스를 상속하는 Base – Salaried Commssion Employee 총 4가지 타입 모두 Employee 생성자를 통해서 Employee 타입으로 객체가 생성되었다.

* 1. 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**)

* 출력값에 바뀐 점이 없다.
  1. **Before changing** and **after** changing in (B), add the following code to the **main** function and run it.
* **정상적으로 컴파일 되지 않는다. 이유는 abstract class는 인스턴스화가 불가능하기 때문이다. 추상 클래스를 이용하여 인스턴스화를 하려면 반드시 추상 클래스와 그 자식클래스가 존재해야만 하고 자식클래스를 통해서만 인스턴스화를 해야한다.**

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

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

1. **Answer the questions after running Payroll system in Section 10.9 (Fig 10.11 ~ 10.15, Payable, Invoice, Employee, SalariedEmployee, PayableInterfaceTest).**
   1. 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)
  1. **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**)

1. **Dog, Cat , and Sheep are all animals with their own unique cries. Answer the following questions.**
   1. 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 ”

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