객체지향프로그래밍 - 9주차 실습 답안지

성명: 조우연 학과: 수학과 학번: 201921195

※ 본 실습활동지에 대한 보고서나 코드를 작성함에 있어 교재나 강의노트를 제외한 다른 자료로부터 일부 또는 전체를 복사하였습니까? 예( ) 아니오(O)

1. Employee.java와 EmployeeSortTest.java를 수행해보고 다음 물음에 답하시오.

(가) Employee 객체를 이름순으로 정렬하도록 Employee class를 수정해보시오. EmployeeSortTest class는 그대로 이용한다.

- 수정된 코드

```
36    public int compareTo(Employee other)
37    {
38       return this.name.compareTo(other.name);
39    }
```

- 수행결과

```
<terminated> EmployeeSortTest [Java Applic
name=Carl Cracker,salary=75000.0
name=Harry Hacker,salary=35000.0
name=Tony Tester,salary=38000.0
```

(나) Employee.java를 수정하지 않고 Comparator<Employee> 인터페이스를 이용하여 Employee 객체를 이름순으로 정렬하고자 한다. 단, EmployeeSortTest 클래스는 정렬부분을 실습문제지에 주어진 대로 수정한다.

- 코드

```
class NameComparator implements Comparator<Employee>
{
    public int compare(Employee e1, Employee e2)
    {
        return e1.getName().compareTo(e2.getName());
    }
}
```

- 수행결과

```
name=Carl Cracker,salary=75000.0
name=Harry Hacker,salary=35000.0
name=Tony Tester,salary=38000.0
```

2. 실습문제지에 주어진 지문을 읽고 실습지에 주어진 코드를 실행해본 후 아래 물음에 답하시오.

(가) Spider class를 동물의 subclass로 추가하고자 한다. 그런데 Spider는 일반적으로 소리를 내지 않는다고 가정하자. 어떤 문제가 있는가?

Animal은 abstract 클래스이므로, Animal의 subclass는 cry()메소드를 꼭 구현해야만 한다. 하지만, Spider와 같은 소리를 내지 않는 동물들도 존재한다. 따라서 동물이지만, 울음소리를 내지 않는 동물들은 Animal class의 subclass로 추가할 수 없다는 문제가 생긴다.

(나) (가)번의 문제를 해결하기 위하여 '소리를 낼 수 있는 것'들을 AbleToCry interface로 정의하고 Dog, Car, Duck 클래스들을 Animal class를 상속을 받으면서 AbleToCry interface를 구현하여 다시 정의하고자 한다. AbleToCry interface와 각 클래스를 정의하여 주어진 AbleToCryTest.java를 이용해 실행해 보시오. Dog, Cat, Duck 외에도 Siren 클래스도 subtype으로 추가해보시오.

- 코드
-> Dog,Cat,Duck : Animal class를 상속받으면서 AbleToCry interface를 구현,
-> Siren : Animal class를 상속받지 않으면서 AbleToCry interface를 구현

다음페이지에 코드있음!

```java
package lab_B_9_201921195;

public abstract class Animal
{
    public abstract void cry();
}

class Dog extends Animal implements AbleToCry
{
    public void cry() {
        System.out.println("Bow Wow!");
    }
}

class Cat extends Animal implements AbleToCry
{
    public void cry() {
        System.out.println("Meow!");
    }
}

class Duck extends Animal implements AbleToCry
{
    public void cry() {
        System.out.println("Quack Quack!");
    }
}

class Siren implements AbleToCry
{
    public void cry() {
        System.out.println("wee-oww wee-oww!");
    }
}
```

- 코드

-> AbleToCry 인터페이스

```java
package lab_B_9_201921195;

public interface AbleToCry {
    void cry();
}
```

- 수행결과

```
Bow Wow!
Meow!
Quack Quack!
wee-oww wee-oww!
```

3. 강의노트 Chapter5 part3(Abstract Class)에서 사용한 Employee class계층에 대한 ObjectClassTest 클래스를 실행하여 Employee class 및 SalariedEmployee class의 equals() 및 toString() method의 용도 및 의미를 파악한 후 다음 물음에 답하시오.

(가) Employee class의 각 subclass에 toString() method를 추가한 후 ObjectClassTest class를 수정하여 이를 테스트하시오.

- 코드

```
@Override
public String toString() {
    return super.toString() + "[grossSales=" + grossSales + ",commissionRate=" + commissionRate +"]";
}
```

-> CommissionEmployee class에 toString() 추가

```
@Override
public String toString() {
    return super.toString() + "[wage=" + wage + ",hours=" + hours +"]";
}
```

-> HourlyEmployee class에 toString() 추가

```
@Override
public String toString()
{
    return super.toString();
}
```

-> Manager class에 toString() 추가

```
public class ObjectClassTest
{
   public static void main(String[] args)
   {
      Employee alice1 = new SalariedEmployee("Alice Adams", "11111", 1000.0);
      //Employee alice2 = alice1;
      //Employee alice3 = new SalariedEmployee("Alice Adams", "11111", 1000.0);
      Employee bob = new CommissionEmployee("Bob", "11112", 1000.0, 0.2);
      Employee david = new HourlyEmployee("David", "11113", 5000.0, 5.0);
      Employee frank = new Manager("Frank", "11114", 2000.0);


      System.out.println("alice1.toString(): " + alice1);
      System.out.println("bob.toString(): " + bob);
      System.out.println("david.toString(): " + david);
      System.out.println("frank.toString(): " + frank);
   }
}
```

-> Test를 위해 수정한 ObjectClassTest class 코드

- 수행결과

```
alice1.toString(): lab_B_9_201921195.SalariedEmployee[name=Alice Adams,ssn=11111][weeklySalary=1000.0]
bob.toString(): lab_B_9_201921195.CommissionEmployee[name=Bob,ssn=11112][grossSales=1000.0,commissionRate=0.2]
david.toString(): lab_B_9_201921195.HourlyEmployee[name=David,ssn=11113][wage=5000.0,hours=5.0]
frank.toString(): lab_B_9_201921195.Manager[name=Frank,ssn=11114][weeklySalary=2000.0]
```

(나) Employee class의 각 subclass에 equals() method를 추가한 후 ObjectClassTest class를 수정하여 이를 테스트하시오.

- 코드

```
@Override
public boolean equals(Object otherObject)
{   // super.equals checks  that "this" and other belong to the same class
     if ( ! super.equals(otherObject) )
           return false;
     CommissionEmployee other = (CommissionEmployee) otherObject;
     if (this.grossSales!=other.grossSales)
         return false;
     return this.commissionRate == other.commissionRate; // compare fields
}
```

-> CommissionEmployee class에 equals() 추가

```
@Override
public boolean equals(Object otherObject)
{   // super.equals checks  that "this" and other belong to the same class
     if ( ! super.equals(otherObject) )
           return false;
     HourlyEmployee other = (HourlyEmployee) otherObject;
     if(this.wage!=other.wage)
         return false;
     return this.hours == other.hours; // compare fields
}
```

-> HourlyEmployee class에 equals() 추가

```
@Override
public boolean equals(Object otherObject)
{   // super.equals checks  that "this" and other belong to the same class
     if ( ! super.equals(otherObject) )
           return false;
     return true;
}
```

-> Manager class에 equals() 추가

```
public class ObjectClassTest
{
   public static void main(String[] args)
   {
      Employee alice1 = new SalariedEmployee("Alice Adams", "11111", 1000.0);
      Employee alice2 = alice1;
      Employee alice3 = new SalariedEmployee("Alice Adams", "11111", 1000.0);

      Employee bob1 = new CommissionEmployee("Bob", "11112", 1000.0, 0.2);
      Employee bob2 = bob1;
      Employee bob3 = new CommissionEmployee("Bob", "11112", 1000.0, 0.2);

      Employee david1 = new HourlyEmployee("David", "11113", 5000.0, 5.0);
      Employee david2 = david1;
      Employee david3 = new HourlyEmployee("David", "11113", 5000.0, 5.0);

      Employee frank1 = new Manager("Frank", "11114", 2000.0);
      Employee frank2 = frank1;
      Employee frank3 = new Manager("Frank", "11114", 2000.0);

      System.out.println("alice1 == alice2: " + (alice1 == alice2));
      System.out.println("alice1 == alice3: " + (alice1 == alice3));
      System.out.println("alice1.equals(alice3): " + alice1.equals(alice3));

      System.out.println("bob1 == bob2: " + (bob1 == bob2));
      System.out.println("bob1 == bob3: " + (bob1 == bob3));
      System.out.println("bob1.equals(bob3): " + bob1.equals(bob3));

      System.out.println("david1 == david2: " + (david1 == david2));
      System.out.println("david1 == david3: " + (david1 == david3));
      System.out.println("david1.equals(david3): " + david1.equals(david3));

      System.out.println("frank1 == frank2: " + (frank1 == frank2));
      System.out.println("frank1 == frank3: " + (frank1 == frank3));
      System.out.println("frank1.equals(frank3): " + frank1.equals(frank3));
   }
}
```

-> Test를 위해 수정한 ObjectClassTest class 코드

- 수행결과

```
alice1 == alice2: true
alice1 == alice3: false
alice1.equals(alice3): true
bob1 == bob2: true
bob1 == bob3: false
bob1.equals(bob3): true
david1 == david2: true
david1 == david3: false
david1.equals(david3): true
frank1 == frank2: true
frank1 == frank3: false
frank1.equals(frank3): true
```