CHOO TIAN LONG 24068668

WIX1002 Fundamentals of Programming
Tutorial 10 Polymorphism

1. Create an abstract class DiscountPolicy. It has a single abstract method computeDiscount that return the discount for the purchase of a given number of item. The method has two parameters count and itemCost. Derived a class BulkDiscount from DiscountPolicy. It has a constructor that has two parameters minimum and discount rate. It has a method computeDiscount that compute the discount base on discount rate if the number of item more than minimum. Otherwise, no discount given. Derived a class OtherDiscount that compute the discount base on the table below

N (number of Item)	1 – 2	3 – 5	6 – 8	>8
Discount	0	10%	20%	30%

Derived a class combineDiscount from DiscountPolicy. It has two parameters of type DiscountPolicy It has a method computeDiscount that return the maximum value returned by the computeDiscount for the two discount policies. Create a Tester class to test the program.

DiscountPolicy abstract class

```
public abstract class DiscountPolicy {
        public abstract double computeDiscount(int count, double itemCost);
}
```

BulkDiscount derived class

```
public class BulkDiscount extends DiscountPolicy{
        protected int minimum;
        protected double discountRate;

        public BulkDiscount(int m, double d){
           this.minimum = m;
           this.discountRate = d;
        }

        public double computeDiscount(int count, double itemCost){
           if (count > minimum) {
              return (itemCost * count * discountRate);
           }else{
              return 0; // no discount is given
```

```
            }
        }
}
```

OtherDiscount derived class

```
public class OtherDiscount extends DiscountPolicy{

        public double computeDiscount(int count, double itemCost){
            if (count >= 1 && count <= 2) {
                return 0;
            } else if (count >= 3 && count <= 5){
                return (0.10 * count * itemCost);
            } else if (count >= 6 && count <= 8){
                return (0.20 * count * itemCost);
            } else {
                return (0.30 * count * itemCost);
            }
        }
}
```

combineDiscount derived class

```
public class combineDiscount extends DiscountPolicy{
        protected DiscountPolicy policy1, policy2;

        public combineDiscount(DiscountPolicy p1, DiscountPolicy p2){
            this.policy1 = p1;
            this.policy2 = p2;
        }

        public double computeDiscount(int count, double itemCost){
            return Math.max(this.policy1.computeDiscount(count, itemCost),
    this.policy2.computeDiscount(count, itemCost));
        }
}
```

Tester class

```
public class Tester {
        public static void main(String[] args) {
            BulkDiscount bd = new BulkDiscount(400, 0.5);
            OtherDiscount od = new OtherDiscount();
            combineDiscount cd = new combineDiscount(bd,od);

            System.out.println("Bulk Discount: " + bd.computeDiscount(500, 2.0));
            System.out.println("Other Discount: " + od.computeDiscount(500, 2.0));
            System.out.println("Combine Discount: " + cd.computeDiscount(500, 2.0));
        }
}
```

2. Create an interface Interest that has a single method computeInterest that return the monthly interest based on the balance in the account. Create the SavingAccount that implement the interface, the class has an instance variable called balance. Define the method to compute interest. The interest rate for saving account is 0.5% per year. Create the FixedAccount that implement the interface. The class has an instance variable called balance. Define the method to compute interest. The interest rate for saving account is 3% per year. Create a Tester class to test the program.

Interest interface class

```
public interface Interest {
        public double computeInterest();
}
```

SavingAccount class

```
public class SavingAccount implements Interest{
        protected double balance;

        public SavingAccount(double b){
           this.balance = b;
        }

        public double computeInterest(){
           return (0.005*balance);
        }

}
```

FixedAccount class

```
public class FixedAccount implements Interest{
        protected double balance;

        public FixedAccount(double b){
           this.balance = b;
        }
```

```
        public double computeInterest(){
            return (0.03 * balance);
        }
}
```

Tester class

```
public class Tester {
        public static void main(String[] args) {
            SavingAccount sa = new SavingAccount(1000);
            FixedAccount fa = new FixedAccount(1000);

            System.out.println("Saving Account Interest: " + sa.computeInterest());
            System.out.println("Fixed Account Interest: " + fa.computeInterest());
        }
}
```

3. Create a class Person that implements the comparable interface. The class has an instance variable name. The class has the constructor that initializes the name. The class also has the accessor method and a display method to display the name. Create an array for multiple Person objects. Sort the person in ascending order. Create a Tester class to test the program.

Person class

```
public class Person implements Comparable<Person>{
        protected String name;

        // Constructor
        public Person(String n){
            this.name = n;
        }

        // Accessor
        public String getName(){
            return this.name;
        }

        public void displayName(){
            System.out.println("Name: " + getName());
        }

        public int compareTo(Person other){
            return (this.name.compareToIgnoreCase(other.getName()));
        }
}
```

Tester class

```
public class Tester {
    public static void main(String[] args) {

        Person[] people = {new Person("John"), new Person("Ali"), new Person("Bob"), new Person("Mutu"), new Person("Siti")};
        Person temp;

        System.out.println("List of People: ");
        for(Person person : people){
            person.displayName();
        }

        // Sort the names in ascending order using Bubble Sort
        for (int i = 0; i < people.length - 1; i++) {
            for (int j = 0; j < people.length - 1 - i; j++) {
                if(people[i].compareTo(people[i+1]) > 0){
                    temp = people[i];
                    people[i] = people[i+1];
                    people[i+1] = temp;
                }
            }
        }



        System.out.println("\nList of People in ascending order: ");
        for(Person person : people){
            person.displayName();
        }
    }
}
```