OOP Class

Documentation on assignment3

Computer Science
2014004411 Si Wan KIM

1. PizzaOrTopping Class

PizzaOrTopping class is abstract component and it should contain data that is common to both the Component and the decorator.

```java
package pizza;

public abstract class PizzaOrTopping {

        protected String description;
        private boolean isFinished;
        private int orderNum;

        public PizzaOrTopping(int orderNum){
                this.orderNum =orderNum;
                isFinished = false;
                description = "Unknown Pizza";
        }

        public String getDescription(){
                return description;
        }

        public int getOrderNum(){
                return orderNum;
        }

        public boolean getIsFinished(){
                return isFinished;
        }

        public void finish(){
                isFinished = true;
        }

        public String toString(){
                return description;
        }

        public abstract long getCookingTime();

        public abstract double cost();
}
```

There's no cookingtime and cost on PizzaOrTopping class but we need it for the derived class so it defined as a abstract method.

2. OriginalPizza Class

OriginalPizza class is concrete component and it extend the PizzaOrTopping class.

```java
package pizza;

public class OriginalPizza extends PizzaOrTopping{

        private double cost;
        private long cookingTime;

        public OriginalPizza(int orderNum){
                super(orderNum);
                description = "OriginalPizza\n";
                cost = 3.75;
                cookingTime=10000;
        }

        public long getCookingTime(){
                return cookingTime;
        }

        public double cost(){
                return cost;
        }
}
```

The DeepPanPizza which inherit the PizzaOrTopping Class has the same constitution with OriginalPizza except only cost and cooking time.
Each concrete class that inherit the PizzaOrTopping class has the different characteristic value on cooking time and cost.

The abstract method on baseclass is fully defined in a descendent class, and it each descendent class has a different cookingtime, and cost.

3. Topping Class

```java
package pizza;

public abstract class Topping extends PizzaOrTopping{

        public Topping(int orderNum){
                super(orderNum);
        }

        public abstract String getDescription();

}
```

Topping Class is abstract decorator extend the abstract component.

It contains abstract method to get the common data, and it's body will be determinded by concrete decorator.

4. Bacon Class

Bacon Class is concrete decorator that extend the abstract decorator(Topping class).

```java
package pizza;
public class Bacon extends Topping{

        private PizzaOrTopping pizzaOrTopping;
        private double cost;
        private int cookingTime;

        public Bacon(PizzaOrTopping pizzaOrTopping){
                super(pizzaOrTopping.getOrderNum());
                this.pizzaOrTopping = pizzaOrTopping;
                cost = 0.75;
                cookingTime = 2000;
        }

        public String toString(){
                return "\tBacon\n";
        }

        public String getDescription(){
                return pizzaOrTopping.getDescription() + toString();
        }

        public long getCookingTime(){
                return pizzaOrTopping.getCookingTime() + cookingTime;
        }

        public double cost(){
                return pizzaOrTopping.cost() + cost;
        }

}
```

Constructor accept PizzaOrTopping object as a parameter. And the PizzaOrTopping object is stored on instance variable.

Pineapple, Pepporoni, Mushroom and Cheese Class have a same constitution with this class, and it also concrete decorator that extend the abstract decorator.
It only differnet it's description, cost and cooking time.

Accessor method returns not only it's value, but also pizzaOrTopping object's value.
Instance vairable's value is appended on the object(PizzaOrTopping)'s value. (When the method is invoked, it return all value that is appended)

5. Subject Interface

The oven class implements the Subeject interface.

```java
package pizza;

public interface Subject {

        public void registerObserver(Observer O);
        public void removeObserver(Observer O);
        public void notifyObservers();
}
```

6. Observer Interface

The order class implements the Observer interface.

```java
package pizza;

public interface Observer {

        public void update(PizzaOrTopping pizza);
}
```

7. Oven Class (1/2)

We have to import to use these each class.

```
1  package pizza;
2  import java.util.ArrayList;
3  import java.util.Timer;
4  import java.util.TimerTask;
5
6  public class Oven implements Subject{
7
8          private PizzaOrTopping finishedPizza;
9          private ArrayList<PizzaOrTopping> pizzas;
10         private ArrayList<Observer> observers;
11
12         public Oven(){
13                 pizzas = new ArrayList<PizzaOrTopping>();
14                 observers = new ArrayList<Observer>();
15         }
16
17         public void registerObserver(Observer O){
18                 observers.add(O);
19         }
20
21         public void removeObserver(Observer O){
22                 observers.remove(O);
23         }
24
25         public void notifyObservers(){
26                 for(int i =0;i<observers.size();i++){
27                         observers.get(i).update(finishedPizza);
28                 }
29         }
30
31
```

The oven class implements Subject interface, and it sends a notification message to observers(Order object) when the states is change(when the pizza making is finished).

The Arraylist have a type parameter. And it is possible to store only the type which is in the angular brackets.

ArrayList type Observer store the order object.

These method add or delete the Order object on the ArrayList.

When the state is change(when the pizza making is finish), invoke this method then it calls update method on all observer object(Order object) in Observer type ArrayList(observers).

7. Oven Class (2/2)

```java
public String toString(){
        String returns = "";
        for(PizzaOrTopping e : pizzas){
                returns += e.getDescription();
        }
        return returns;
}

public void removePizza(PizzaOrTopping pizza){
        pizzas.remove(pizza);
}

public void addPizza(PizzaOrTopping pizza){
        pizzas.add(pizza);

        Timer pizzaTimer = new Timer();
        pizzaTimer.schedule(new TimerTask(){
                public void run(){
                        pizza.finish();
                        finishedPizza = pizza;
                        removePizza(pizza);
                        notifyObservers();
                }
        },pizza.getCookingTime());

}
}
```

It returns description of all pizzaOrTopping Object in pizzas ArrayList.
For – each loop is used to call the all of the object in ArrayList.

This method is responsible for cooking the pizza. This creates a thread for each pizza based on the pizzas cooking time therefore each pizza cooks at a different time. And when the pizza making is finished notifyObservers method is called to notify the information to Observers.

8. Order Class

```java
package pizza;

public class Order implements Observer{

        private int orderNum;
        private boolean collected;
        private Subject pizzaOven;

        public Order(int orderNum, Oven pizzaOven){
                this.orderNum = orderNum;
                this.pizzaOven = pizzaOven;
                collected = false;
        }

        public void update(PizzaOrTopping pizza){
                if(this.orderNum == pizza.getOrderNum()){
                        collected = true;
                        pizzaOven.removeObserver(this);
                }
        }

}
```

When update method is invoked, it accept the object which have a information from a parameter. And use a getmethod to get the information from object.

When the finished pizza's ordernumber is same with the orderNum remove the this Observer(Order obejct) from Oven's observer ArrayList.

9. PizzaShop Class (1/2)

```
1   package pizza;
2   import java.util.ArrayList;
3
4   public class PizzaShop {
5
6           private volatile static PizzaShop singlePizzaShop;
7           private int orderCounter;
8           private ArrayList<Order> orders;
9           private Oven pizzaOven;
10
11          private PizzaShop(){
12                  orders = new ArrayList<Order>();
13                  pizzaOven = new Oven();
14                  orderCounter =1;
15          }
16
17          public static PizzaShop getInstance(){
18                  if(singlePizzaShop==null){
19                          synchronized(PizzaShop.class){
20                                  if(singlePizzaShop==null){
21                                          singlePizzaShop = new PizzaShop();
22                                  }
23                          }
24                  }
25                  return singlePizzaShop;
26          }
27
28
```

We want to know right information from the object, so we use the volatile.
It write or read the data from the main memory not cache memory. So it ensure that a write will always complete before a read.

We need only one pizzashop, so we don't have to make more than one object. When the modifier is not private it is too easy to make multiple.

We make constructor modifier private, so we make an object in other class by using this method. It using synchronized and double-checked locking so it protect multiple thread's access at the same time. And only when the instance variable is null it makes new object.

9. PizzaShop Class (2/2)

```
29          public void placeOrder(PizzaOrTopping pizza){
30                  Order order = new Order(orderCounter,pizzaOven);
31                  orderCounter++;
32                  orders.add(order);
33                  pizzaOven.registerObserver(order);
34                  pizzaOven.addPizza(pizza);
35          }
36
37          public int getOrderCounter(){
38                  return orderCounter;
39          }
40
41          public Oven getPizzaOven(){
42                  return pizzaOven;
43          }
44
45          public String toString(){
46                  return pizzaOven.toString();
47
48          }
    }
```

When placeOrder method is invoked adds the order(Observer) and pizza object in each ArrayList.

Screenshot of Outputs(Simple Launcher)

```
*******Pizzashop Management System*******
(1) Order a Pizza:
(2) View Pizzas
(3) Exit the program
****************************************
1
*******Pizzashop Management System*******
Please choose the type of pizza
(1) Original Pizza:
(2) DeepPan Pizza:
(3) Cancel Order
****************************************
1
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
1
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
2
```

```
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
6
Pizza Ordered: OriginalPizza
        Pepperoni
        Bacon

*******Pizzashop Management System*******
(1) Order a Pizza:
(2) View Pizzas
(3) Exit the program
****************************************
2
Pizzas in the Oven...
OriginalPizza
        Pepperoni
        Bacon

*******Pizzashop Management System*******
(1) Order a Pizza:
(2) View Pizzas
(3) Exit the program
****************************************
Pizza is finished OriginalPizza
        Pepperoni
        Bacon
```

Screenshot of Outputs(Simple Launcher)

```
*******Pizzashop Management System*******
(1) Order a Pizza:
(2) View Pizzas
(3) Exit the program
****************************************
1
*******Pizzashop Management System*******
Please choose the type of pizza
(1) Original Pizza:
(2) DeepPan Pizza:
(3) Cancel Order
****************************************
2
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
1
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
3
```

```
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
5
*******Pizzashop Management System*******
Please choose a topping
(1) Pepperoni
(2) Bacon
(3) Pineapples
(4) Mushrooms
(5) Extra Cheese
(6) Confirm Order
(7) Cancel Order
****************************************
6
Pizza Ordered: Deep Pan
        Pepperoni
        Pineapples
        Extra Cheese
```

```
Pizza Ordered: Deep Pan
        Pepperoni
        Pineapples
        Extra Cheese

*******Pizzashop Management System*******
(1) Order a Pizza:
(2) View Pizzas
(3) Exit the program
****************************************
2
Pizzas in the Oven...
Deep Pan
        Pepperoni
        Pineapples
        Extra Cheese

*******Pizzashop Management System*******
(1) Order a Pizza:
(2) View Pizzas
(3) Exit the program
****************************************
Pizza is finished Deep Pan
        Pepperoni
        Pineapples
        Extra Cheese

3
Exiting Program... Goodbye.
```

Screenshot of Outputs(GUI Launcher)