ANADOLU ÜNİVERSİTESİ

Design pattern Project

Hana Hadžo Mulalić

987 156 287 27

prof. Dr. Öğr. ÜyesiAlper BİLGE

Res.Asst. Gökhan ÇIPLAK

Project definition

Solve a software development problem using one (or possibly more) of the design patterns that we cover in class.

To-Do

- **Statement of Work**: Write your problem definition in detail using 100-200 words.
- **Design Pattern(s)**: Select design pattern(s) that are useful in solving the problem you defined. Explain why and how you employ such design pattern(s) in your Project.
- **UML**: Draw a detailed class diagram of your proposed solution using some UML Diagram Drawing Tool.
- **Implementation**: : Implement your proposed solution based on the UML Class Diagram using the base Maven Project that you are supplied. The Maven Project produces a executable **.jar** file. Your project should not expect any input from user, and it should produce decent outputs when compiled by the following command: java -jar target\project1.jar You are allowed to change pom.xml to add or remove any dependency or plugin.
- You need to upload the **.pdf** file and necessary parts of your code. (Do not upload target folder and IDE-specific files)

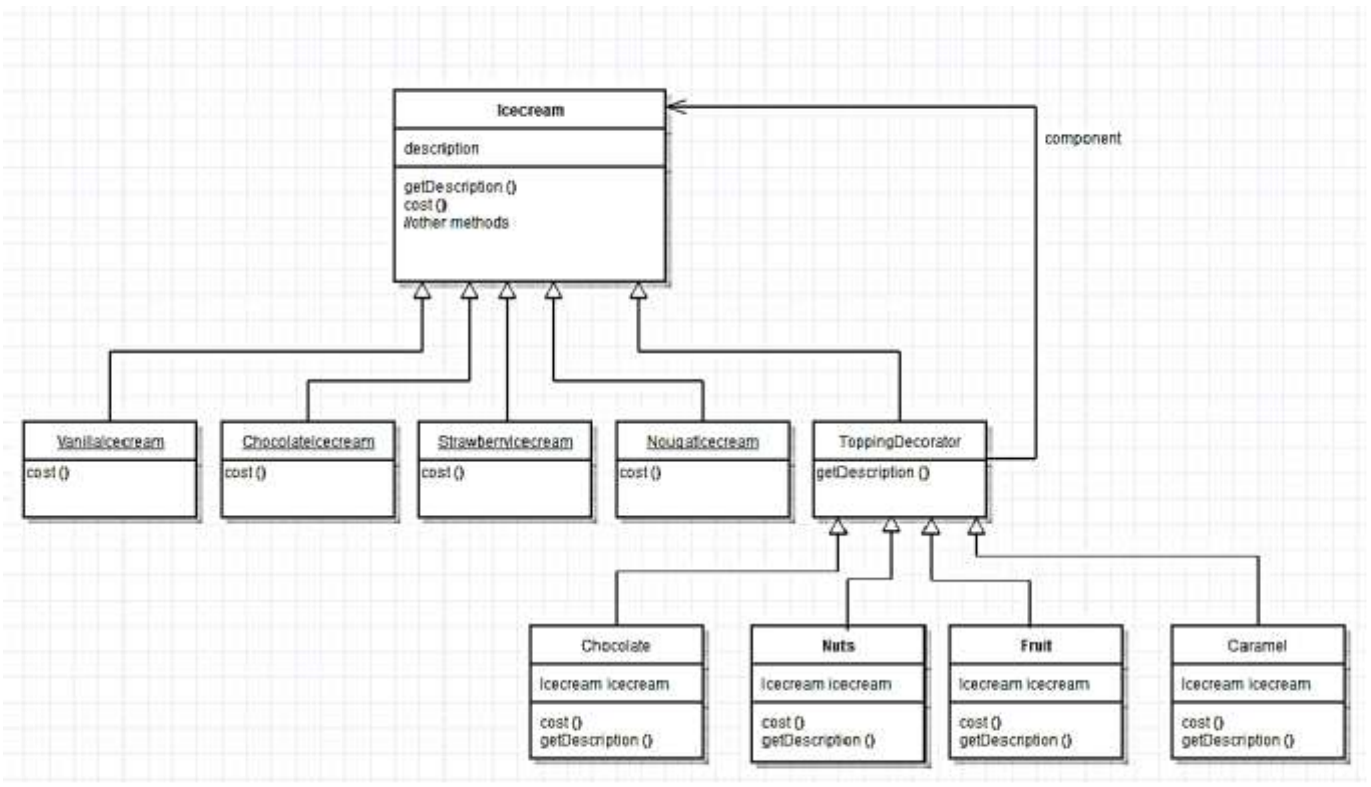
Statement of Work

Let' s first demonstrate a real-life situation. Icecream shop offers a different flavours of icecream: vanilla icecream, chocolate icecream, strawberry icecream, etc. However, the icecream buyers can also recquire a different kinds of icecream toppings such are: chocolate, fruit, nuts, caramel, etc. Different kinds of icecream have their own costs and each icecream topping adds its own cost. So, the total cost, the buyer needs to pay is equal to cost of specific kind of the icecream plus one or more kinds of icecream toppings. It is needed to write a java program that manages the icecreams available for the purchase as a icecream shop. The class IcecreamShop should be a driver class for testing other classes. The abstradct class Icecream should contain methods getDescription() (returns the kind of icecream) and cost() (abstract method which specifies the cost of the kind of icecream). Also, implement the class for icecream toppings. Specific kinds of toppings should returns its own description and have its own costs.There should be printed on the screen the kind of the icecream, its toppings and the total cost.

Design Pattern - Decorator Pattern

If we define the class which represents combinations of all kinds ice-cream with all combinations of ice-cream toppings we would get the problem of 'class explosion'. We could implement a base Icecream class and add instance variables to represent weather or not each ice-cream has nuts, chocolate... Method cost() would be implemented in Icecream class to calculate the cost regarded to icecream toppings for particular icecream instance. Subclasses would override cost(), but also, they would calculate the total cost of basic icecream plus the costs of toppings. So, each cost() method would compute the cost of the icecream and then add in the topping by calling the superclass implementation of cost(). However, such design could be destroyed by several changes that may happen in future: 'What would happen if the price of the icecream or its toppings change, or if new kinds of icecream or topping be added, or, if the buyer would want the double quantity of toppings. In program implementation, it would violate two design principles: ‘Encapsulate what varies' and 'Favor composition over inheritance'. We can see that an icecream plus topping cost does not work well with the inheritance. We get class explosion and rigid design. The solution is to extend an object’s behavior through composition dynamically at runtime. It will give us the flexibility of adding new functionalities to objects without code modifications (changes). As a result, the bugs will be reduced. Design principle says: **“Classes should be open for extension, but closed for modification”.** We can take an icecream of specific flavor object and then, decorate it with topping object, then if buyer required decorated with another topping object and so on. An object – icecream can be wrapped in any number of decorators. The decorators (caramel, nuts…) should be of the same type as the object they decorate (icecream). **The decorator adds its own behavior either before or /and after delegation to the object it decorates to do rest of the job.** In the end, we call the cost() method through delegation to add the topping costs.

UML Diagram



Implementation

IcecreamShop.java

```
package icecreamshop;
/**
 *
 * @author Hana
 */
public class IcecreamShop {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    Icecream icecream=new VanillaIcecream();
    icecream=new Fruit (icecream);
    System.out.println(icecream.getDescription()+ " $" + icecream.cost());
```

```
   Icecream icecream2=new ChocolateIcecream();
   icecream2=new Chocolate (icecream2);
   icecream2=new Caramel (icecream2);
   System.out.println(icecream2.getDescription()+ " $" + icecream2.cost());
   Icecream icecream3=new StrawberryIcecream();
   System.out.println(icecream3.getDescription()+ " $" + icecream3.cost());

   Icecream icecream4=new NougatIcecream();
   icecream4=new Nuts(icecream4);
   icecream4=new Caramel (icecream4);
   System.out.println(icecream4.getDescription()+ " $" + icecream4.cost());
}
}
```

Icecream.java

```
package icecreamshop;
public abstract class Icecream {
String description="Unknown Icecream";

public String getDescription(){
   return description;
}
public abstract double cost();
}
```

NougatIcecream.java

```
package icecreamshop;
public class NougatIcecream extends Icecream {
   public NougatIcecream(){
description="Icecream with nougat flavour ";
}
public double cost(){
return 2.3;
}
}
```

ChocolateIcecream.java

```
package icecreamshop;
public class ChocolateIcecream extends Icecream {
  public ChocolateIcecream() {
    description="Icecream with chocolate flavour ";
}
public double cost() {
return 2.5;
}
}
```

StrawberryIcecream.java

```
package icecreamshop;
public class StrawberryIcecream extends Icecream {
public StrawberryIcecream(){
description="Icecream with strawberry flavour";
}
public double cost() {
return 2.3;
}
}
```

VanillaIcecream.java

```
package icecreamshop;
public class VanillaIcecream extends Icecream {
 public VanillaIcecream(){
   description="Icecream with vanilla flavour ";
}
public double cost() {
return 2.0;
}
}
```

ToppingDecorator.java

```
package icecreamshop;
public abstract class ToppingDecorator extends Icecream {
   public abstract String getDescription();
}
```

Caramel.java

```
package icecreamshop;
public class Caramel extends ToppingDecorator {
Icecream icecream;
public Caramel (Icecream icecream){
this.icecream=icecream;
}
public String getDescription() {
return icecream.getDescription()+ ", Caramel Topping";
}
public double cost(){
return 0.6 + icecream.cost();
}
}
```

Chocolate.java

```
package icecreamshop;
public class Chocolate extends ToppingDecorator {
Icecream icecream;
public Chocolate(Icecream icecream){
this.icecream=icecream;
}
public String getDescription() {
return icecream.getDescription()+ ", Chocolate Topping";
}
public double cost(){
return 0.5 + icecream.cost();
}
}
```

Fruit.java

```
package icecreamshop;
public class Fruit extends ToppingDecorator {
Icecream icecream;
public Fruit (Icecream icecream){
this.icecream=icecream;
}
public String getDescription() {
return icecream.getDescription()+ ", Fruit Topping";
}
public double cost(){
return 0.5 + icecream.cost();
}
}
```

Nuts.java

```
package icecreamshop;
public class Nuts extends ToppingDecorator {
Icecream icecream;
public Nuts (Icecream icecream){
this.icecream=icecream;
}
public String getDescription() {
return icecream.getDescription()+ ", Nuts Topping";
}
public double cost(){
return 0.7 + icecream.cost();
}
}
```

Output:

```
run:
Icecream with vanilla flavour , Fruit Topping $2.5
Icecream with chocolate flavour , Chocolate Topping, Caramel Topping $3.6
Icecream with strawberry flavour $2.3
Icecream with nougat flavour , Nuts Topping, Caramel Topping $3.6
BUILD SUCCESSFUL (total time: 6 seconds)
```