

## Homework:

- 1、 试用 Bridge 模式完成下列事情：饮料的杯子有大、中、小；行为有：加奶，加糖，啥都不加。

答：使用桥接模式（Bridge Pattern）可以分离抽象和实现，使得两者可以独立地变化。Drink 类充当抽象部分。它包含一个 Additive 类型的引用，这个引用是一个接口，表明饮料可以添加的成分。Drink 类自身是抽象的，定义了 serve() 方法的框架，但不实现具体的加料逻辑。

```
23 abstract class Drink {
24     4 个用法
25     protected Additive additive;
26     3 个用法
27     public Drink(Additive additive) {
28         this.additive = additive;
29     }
30     3 个用法 3 个实现
31     abstract void serve();
32 }
33
34 1 个用法
35 class SmallDrink extends Drink {
36     1 个用法
37     public SmallDrink(Additive additive) {
38         super(additive);
39     }
40
41     3 个用法
42     public void serve() {
43         System.out.print("饮料-小杯-");
44         additive.add();
45     }
46 }
47
48 1 个用法
49 class MediumDrink extends Drink {
50     1 个用法
51     public MediumDrink(Additive additive) {
52         super(additive);
53     }
54
55     3 个用法
56     public void serve() {
57         System.out.print("饮料-中杯-");
58         additive.add();
59     }
60 }
61
62 1 个用法
63 class LargeDrink extends Drink {
64     1 个用法
65     public LargeDrink(Additive additive) {
66         super(additive);
67     }
68
69     3 个用法
70     public void serve() {
71         System.out.print("饮料-大杯-");
72         additive.add();
73     }
74 }
```

Additive 接口及其实现类（Milk, Sugar, NoAdditive）充当实现部分。这些类实现了 Additive 接口，具体定义了可以添加哪些配料。

```
1 interface Additive {  
    3 个用法 3 个实现  
2     void add();  
3 }  
4  
5 class Milk implements Additive {  
    3 个用法  
6     public void add() {  
7         System.out.println("加奶");  
8     }  
9 }  
10  
11 class Sugar implements Additive {  
    3 个用法  
12     public void add() {  
13         System.out.println("加糖");  
14     }  
15 }  
16  
17 class NoAdditive implements Additive {  
    3 个用法  
18     public void add() {  
19         System.out.println("无额外添加");  
20     }  
21 }
```

在 main 方法中使用并测试桥接模型：

```
64 public class Main {  
    0 个用法  
65     public static void main(String[] args) {  
66         Drink smallWithSugar = new SmallDrink(new Sugar());  
67         smallWithSugar.serve();  
68         Drink mediumWithMilk = new MediumDrink(new Milk());  
69         mediumWithMilk.serve();  
70         Drink largeNoAdd = new LargeDrink(new NoAdditive());  
71         largeNoAdd.serve();  
72     }  
73 }
```

运行结果如下：

```
运行: Main x  
F:\Java\bin\java.exe "-javaagent:F:\IntelliJ  
饮料-小杯-加糖  
饮料-中杯-加奶  
饮料-大杯-无额外添加  
进程已结束,退出代码0
```

可以发现，程序中饮料的大小和加料的种类可以独立进行修改和扩展，而不会相互影响。例如，如果需要添加一种新的饮料大小（比如超大杯），只需扩展 Drink 类。同样，若需添加新的配料（如加蜂蜜），只需实现 Additive 接口。在运行时，可以灵活地组合任意大小的饮料与任意类型的添加剂，这是通过在 Drink 的构造函数中传递不同的 Additive 实例实现的。这种组合是动态的，提供了极高的灵活性。

## 2、 附录

### 1) Additive

```
interface Additive {  
    void add();  
}
```

### 2) Milk

```
class Milk implements Additive {  
    public void add() {  
        System.out.println("加奶");  
    }  
}
```

### 3) Sugar

```
class Sugar implements Additive {  
    public void add() {  
        System.out.println("加糖");  
    }  
}
```

### 4) NoAdditive

```
class NoAdditive implements Additive {  
    public void add() {  
        System.out.println("无额外添加");  
    }  
}
```

### 5) Drink

```
abstract class Drink {  
    protected Additive additive;  
    public Drink(Additive additive) {  
        this.additive = additive;  
    }  
    abstract void serve();  
}
```

#### 6) SmallDrink

```
class SmallDrink extends Drink {  
    public SmallDrink(Additive additive) {  
        super(additive);  
    }  
  
    public void serve() {  
        System.out.print("饮料-小杯-");  
        additive.add();  
    }  
}
```

#### 7) MediumDrink

```
class MediumDrink extends Drink {  
    public MediumDrink(Additive additive) {  
        super(additive);  
    }  
  
    public void serve() {  
        System.out.print("饮料-中杯-");  
        additive.add();  
    }  
}
```

#### 8) LargeDrink

```
class LargeDrink extends Drink {  
    public LargeDrink(Additive additive) {  
        super(additive);  
    }  
  
    public void serve() {  
        System.out.print("饮料-大杯-");  
        additive.add();  
    }  
}
```

#### 9) Main

```
public class Main {  
    public static void main(String[] args) {  
        Drink smallWithSugar = new SmallDrink(new Sugar());  
        smallWithSugar.serve();  
        Drink mediumWithMilk = new MediumDrink(new Milk());  
    }  
}
```

```
        mediumWithMilk.serve();  
        Drink largeNoAdd = new LargeDrink(new NoAdditive());  
        largeNoAdd.serve();  
    }  
}
```