

## Homework:

### 1、 阅读 Abstract Factory 的例子代码，举例说明使用 Abstract Factory 模式的其他应用。

答：抽象工厂模式是一种创建型设计模式，它使得一个类的实例化可以在运行时通过子类来决定。抽象工厂模式通常涉及以下几个组成部分：

- 抽象工厂：定义用于创建不同产品的方法。
- 具体工厂：实现抽象工厂中的方法，用于生产具体的产品。
- 抽象产品：定义产品的接口。
- 具体产品：实现抽象产品接口的实体类。

根据抽象工厂设计模式，我们可以设计一个车辆工厂，这个工厂可以生产不同类型的车辆，比如电动车和燃油车。我们将定义抽象工厂来生产这些不同类型的车辆，并提供相应的具体工厂来实现它们。

首先，定义抽象工厂，一个工厂生产经济型和豪华型两种汽车：

```
2  ↓ interface VehicleFactory {  
    2 个用法  2 个实现  
3  1↓ Car createLuxuryCar();  
    2 个用法  2 个实现  
4  1↓ Car createEconomyCar();  
5  }
```

接着，定义抽象产品，也就是车辆：

```
2  ↓ interface Car {  
    4 个用法  4 个实现  
3  1↓ void drive();  
    4 个用法  4 个实现  
4  1↓ void refuel();  
5  }
```

随后，实现具体产品，即不同类型的车辆：

```
8  ↓ class LuxuryElectricCar implements Car {  
    4 个用法  
9  1↑ public void drive() {  
10     System.out.println("驾驶豪华型电动车。");  
11 }  
    4 个用法  
12 1↑ public void refuel() {  
13     System.out.println("给豪华型电动车充电。");  
14 }  
15 }
```

```

17 class EconomyElectricCar implements Car {
    4 个用法
18 public void drive() {
19     System.out.println("驾驶经济型电动车。");
20 }
    4 个用法
21 public void refuel() {
22     System.out.println("给经济型电动车充电。");
23 }
24 }
25
    1 个用法
26 class LuxuryGasolineCar implements Car {
    4 个用法
27 public void drive() {
28     System.out.println("驾驶豪华型燃油车。");
29 }
    4 个用法
30 public void refuel() {
31     System.out.println("给豪华型燃油车加油。");
32 }
33 }
34
    1 个用法
35 class EconomyGasolineCar implements Car {
    4 个用法
36 public void drive() {
37     System.out.println("驾驶经济型燃油车。");
38 }
    4 个用法
39 public void refuel() {
40     System.out.println("给经济型燃油车加油。");
41 }
42 }

```

然后，实现具体的工厂类：

```

7 class BMWFactory implements VehicleFactory {
    2 个用法
8 public Car createLuxuryCar() {
9     return new LuxuryElectricCar();
10 }
    2 个用法
11 public Car createEconomyCar() {
12     return new EconomyGasolineCar();
13 }
14 }
15
    1 个用法
16 class BenzFactory implements VehicleFactory {
    2 个用法
17 public Car createLuxuryCar() {
18     return new LuxuryGasolineCar();
19 }
    2 个用法
20 public Car createEconomyCar() {
21     return new EconomyElectricCar();
22 }
23 }

```

最后，使用工厂：

```
20 ▶ public class Main {  
    0 个用法  
21 ▶ public static void main(String[] args) {  
22     System.out.println("使用BMW工厂生产: ");  
23     VehicleFactory BMW = new BMWFactory();  
24     Car luxuryElectric = BMW.createLuxuryCar();  
25     luxuryElectric.drive();  
26     luxuryElectric.refuel();  
27     Car economyGasoline = BMW.createEconomyCar();  
28     economyGasoline.drive();  
29     economyGasoline.refuel();  
30     System.out.println("使用Benz工厂生产: ");  
31     VehicleFactory Benz = new BenzFactory();  
32     Car luxuryGasoline = Benz.createLuxuryCar();  
33     luxuryGasoline.drive();  
34     luxuryGasoline.refuel();  
35     Car economyElectric = Benz.createEconomyCar();  
36     economyElectric.drive();  
37     economyElectric.refuel();  
38 }  
39 }
```

查看运行结果：



```
运行: Main x  
F:\Java\bin\java.exe "-javaagent:F:  
使用BMW工厂生产:  
驾驶豪华型电动车。  
给豪华型电动车充电。  
驾驶经济型燃油车。  
给经济型燃油车加油。  
使用Benz工厂生产:  
驾驶豪华型燃油车。  
给豪华型燃油车加油。  
驾驶经济型电动车。  
给经济型电动车充电。  
进程已结束,退出代码0
```

## 2、 附录

### 1) Car

```
interface Car {  
    void drive();  
    void refuel();  
}
```

### 2) EconomyElectricCar

```
class EconomyElectricCar implements Car {  
    public void drive() {  
        System.out.println("驾驶经济型电动车。");  
    }  
}
```

```

    }
    public void refuel() {
        System.out.println("给经济型电动车充电。");
    }
}

```

### 3) LuxuryGasolineCar

```

class LuxuryGasolineCar implements Car {
    public void drive() {
        System.out.println("驾驶豪华型燃油车。");
    }
    public void refuel() {
        System.out.println("给豪华型燃油车加油。");
    }
}

```

### 4) LuxuryElectricCar

```

class LuxuryElectricCar implements Car {
    public void drive() {
        System.out.println("驾驶豪华型电动车。");
    }
    public void refuel() {
        System.out.println("给豪华型电动车充电。");
    }
}

```

### 5) EconomyGasolineCar

```

class EconomyGasolineCar implements Car {
    public void drive() {
        System.out.println("驾驶经济型燃油车。");
    }
    public void refuel() {
        System.out.println("给经济型燃油车加油。");
    }
}

```

### 6) VehicleFactory

```

interface VehicleFactory {
    Car createLuxuryCar();
    Car createEconomyCar();
}

```

### 7) BMWFactory

```
class BMWFactory implements VehicleFactory {  
    public Car createLuxuryCar() {  
        return new LuxuryElectricCar();  
    }  
    public Car createEconomyCar() {  
        return new EconomyGasolineCar();  
    }  
}
```

### 8) BenzFactory

```
class BenzFactory implements VehicleFactory {  
    public Car createLuxuryCar() {  
        return new LuxuryGasolineCar();  
    }  
    public Car createEconomyCar() {  
        return new EconomyElectricCar();  
    }  
}
```

### 9) Main

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("使用 BMW 工厂生产: ");  
        VehicleFactory BMW = new BMWFactory();  
        Car luxuryElectric = BMW.createLuxuryCar();  
        luxuryElectric.drive();  
        luxuryElectric.refuel();  
        Car economyGasoline = BMW.createEconomyCar();  
        economyGasoline.drive();  
        economyGasoline.refuel();  
        System.out.println("使用 Benz 工厂生产: ");  
        VehicleFactory Benz = new BenzFactory();  
        Car luxuryGasoline = Benz.createLuxuryCar();  
        luxuryGasoline.drive();  
        luxuryGasoline.refuel();  
        Car economyElectric = Benz.createEconomyCar();  
        economyElectric.drive();  
        economyElectric.refuel();  
    }  
}
```