Homework：

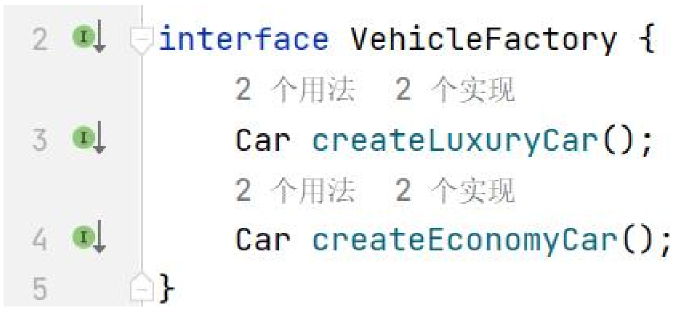
1、 **阅读 Abstract Factory 的例子的代码，举例说明使用 Abstract Factory 模式的其他应用。**

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。

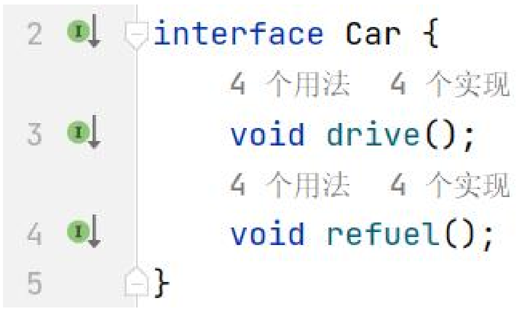
根据抽象工厂设计模式，我们可以设计一个车辆工厂，这个工厂可以生产不同类型的车辆，比如电动车和燃油车。我们将定义抽象工厂来生产这些不同类型的车辆，并提供相应的具体工厂来实现它们。

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：



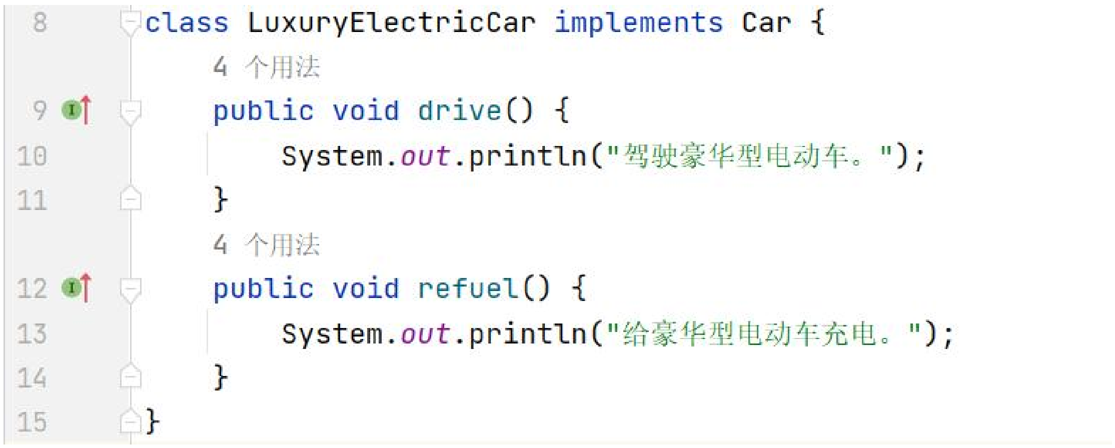
```
interface VehicleFactory {
        2 个用法  2 个实现
        Car createLuxuryCar();
        2 个用法  2 个实现
        Car createEconomyCar();
}
```

接着，定义抽象产品，也就是车辆：

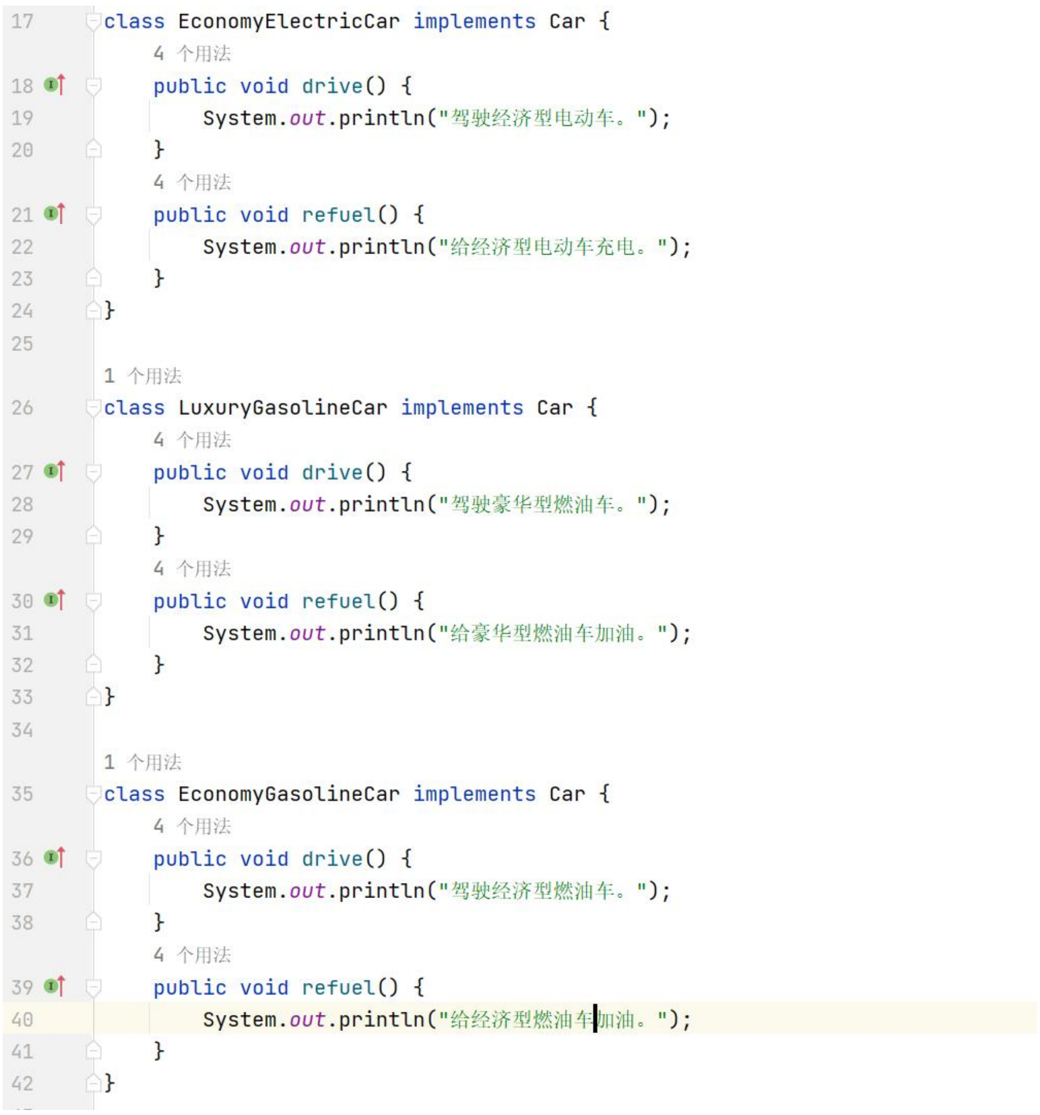


```
interface Car {
        4 个用法  4 个实现
        void drive();
        4 个用法  4 个实现
        void refuel();
}
```

随后，实现具体产品，即不同类型的车辆：

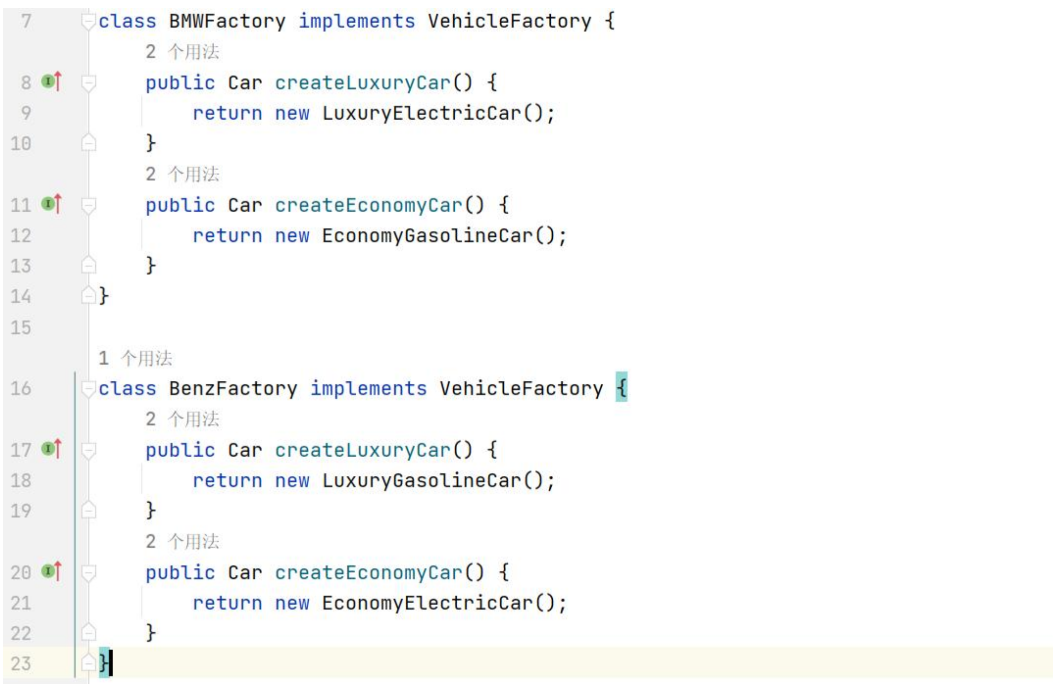


```
class LuxuryElectricCar implements Car {
        4 个用法
        public void drive() {
            System.out.println("驾驶豪华型电动车。");
        }
        4 个用法
        public void refuel() {
            System.out.println("给豪华型电动车充电。");
        }
}
```



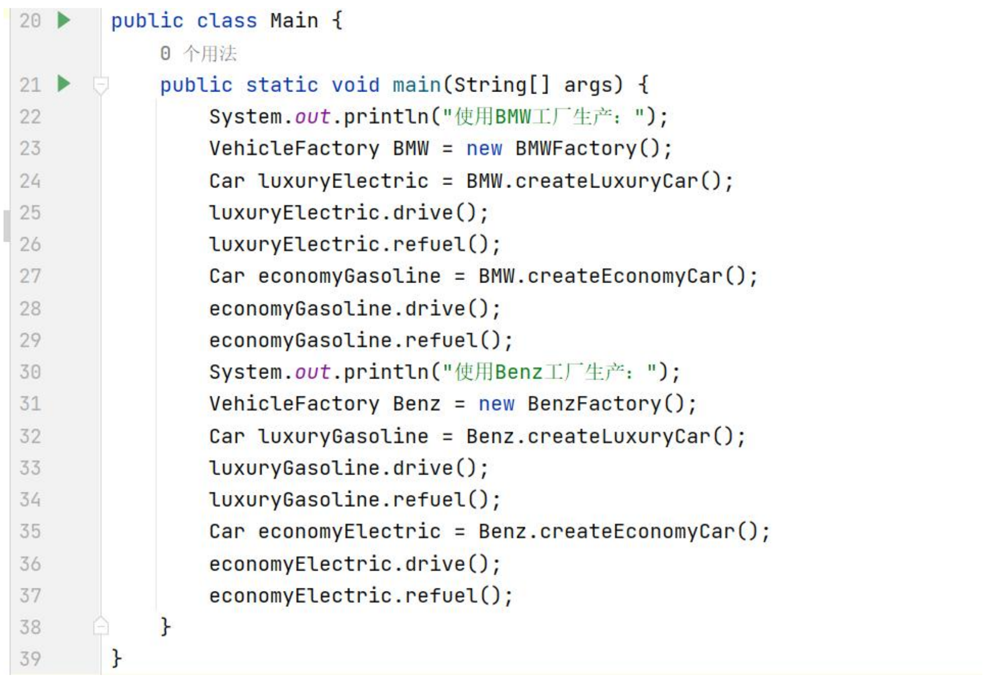
```java
class EconomyElectricCar implements Car {
    4 个用法
    public void drive() {
        System.out.println("驾驶经济型电动车。");
    }
    4 个用法
    public void refuel() {
        System.out.println("给经济型电动车充电。");
    }
}

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

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

然后，实现具体的工厂类：



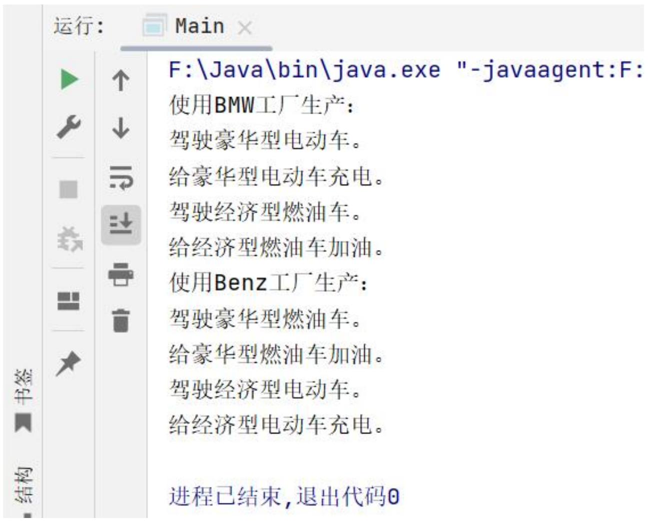
```java
class BMWFactory implements VehicleFactory {
    2 个用法
    public Car createLuxuryCar() {
        return new LuxuryElectricCar();
    }
    2 个用法
    public Car createEconomyCar() {
        return new EconomyGasolineCar();
    }
}

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

最后，使用工厂：



```
public class Main {
    0 个用法
    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();
    }
}
```

查看运行结果：



```
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

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

8) BenzFactory

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

9) Main

```java
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();
    }
}
```