

# SA第十次作业

## 1、改写本例，用于添加另一个具体工厂和具体产品。

添加一个具体的产品Car即车辆，有使用use()和获得其主人getOwner()的方法。

```
public class Car extends Product {  
    private String owner;  
  
    Car(String owner) {  
        System.out.println("生产" + owner + "的车");  
        this.owner = owner;  
    }  
  
    public void use() {  
        System.out.println("使用" + owner + "的车");  
    }  
  
    public String getOwner() {  
        return owner;  
    }  
}
```

添加一个具体的工厂CarFactory用于生产车辆。

使用一个HashMap记录所有车辆及其主人，新增getCar方法可以找到属于某owner的Car。

```
public class CarFactory extends Factory {  
  
    private Vector<String> owners = new Vector<>();  
    private Map<String, Car> cars = new HashMap();  
  
    protected Product createProduct(String owner) {  
        Car car = new Car(owner);  
        cars.put(owner, car);  
        return car;  
    }  
  
    protected void registerProduct(Product product) {  
        owners.add(((Car) product).getOwner());  
    }  
  
    public Vector<String> getOwners() {  
        return owners;  
    }  
  
    public Product getCar(String owner) {  
        return cars.get(owner);  
    }  
}
```

```
}
```

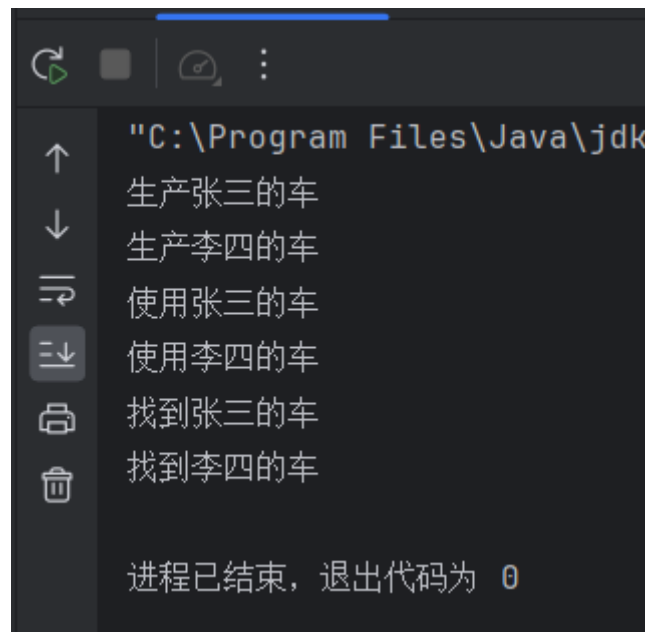
主函数Main

使用抽象类Factory实例化一个具体的CarFactory。

并通过这个CarFactory来生产使用和管理车辆。

```
public class Main {  
    public static void main(String[] args) {  
        Factory factory = new CarFactory();  
        Product car1 = factory.create("张三");  
        Product car2 = factory.create("李四");  
        car1.use();  
        car2.use();  
        if(car1 == ((CarFactory) factory).getCar("张三"))System.out.println("找到张三的车");  
        if(car2 == ((CarFactory) factory).getCar("李四"))System.out.println("找到李四的车");  
    }  
}
```

运行结果：



```
"C:\Program Files\Java\jdk  
↑ 生产张三的车  
↓ 生产李四的车  
⇌ 使用张三的车  
⇌ 使用李四的车  
📄 找到张三的车  
📄 找到李四的车  
进程已结束，退出代码为 0
```

## 2、请举例说明其他的工厂模式的应用。

答：Spring IoC容器的核心接口BeanFactory就使用了工厂模式来隐藏具体的对象实例化过程，它负责创建和管理应用程序中的对象。通过配置文件或注解，开发人员可以告诉 Spring 如何创建对象以及对象之间的依赖关系，从而实现了工厂模式的功能。，客户端只需要通过接口获取Bean对象，而不需要关心具体的实例化细节。