

Lecture 1

The Factory Method Pattern (工厂方法模式)

(Creational)

- **创建型设计模式是解决对象创建机制的设计模式。**
- **该类设计模式试图根据具体的情况，以适当方式创建对象。**

Professor:
Yushan (Michael) Sun
Fall 2020

Contents of this lecture

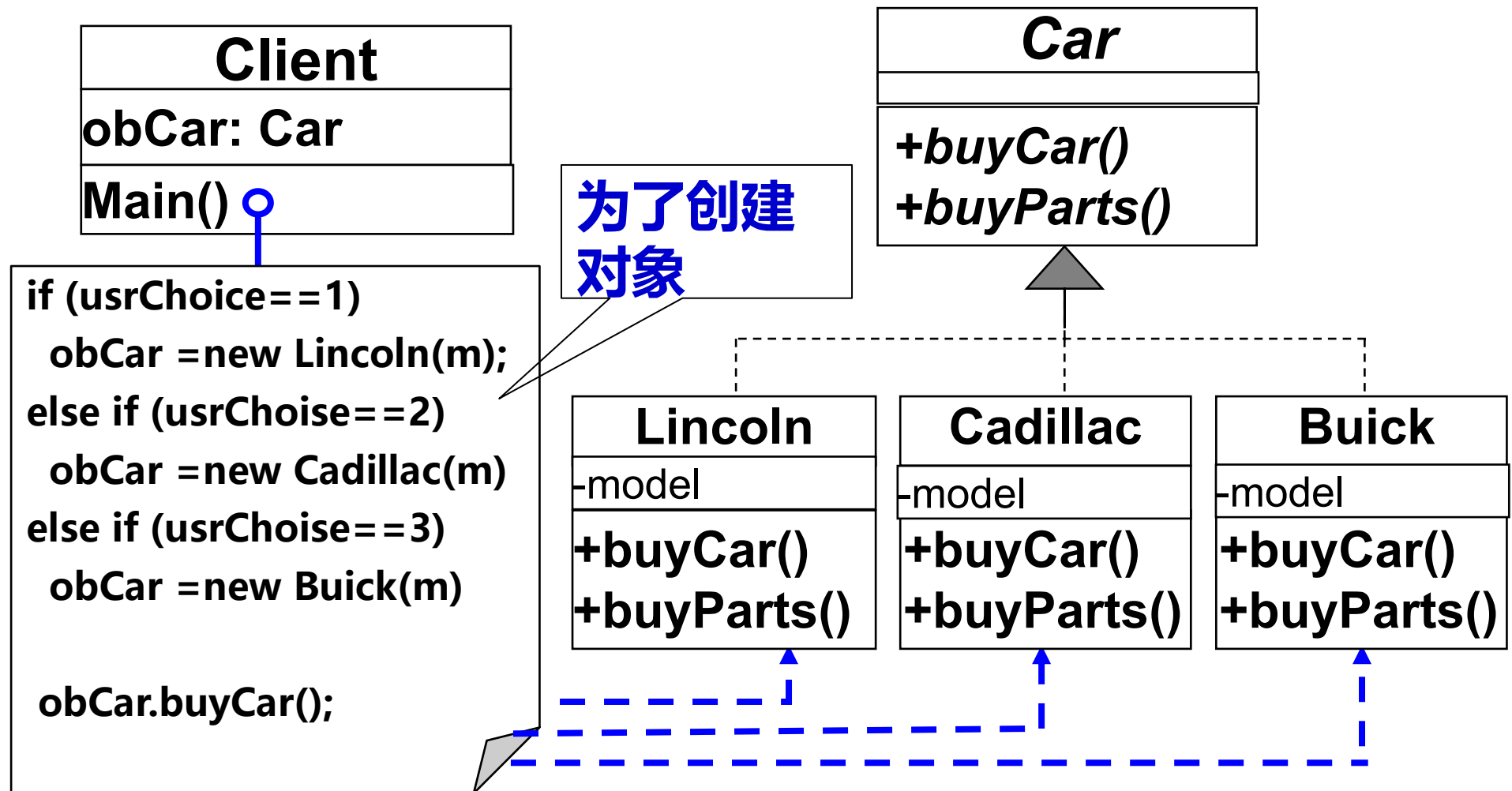
1. Introduction of the factory method pattern
2. Theory of the simple factory method Pattern and design examples
3. Theory of the factory method pattern and design examples
4. Further discussion of the factory method pattern

Introduction of the Factory Method Pattern

Introduction to Factory Method Pattern

Question: How to invoke a method in a class hierarchy that contains many subclasses and you don't know exactly which one to select?

Introduction to Factory Method Pattern



Client Object Directly Accessing a Class Hierarchy

Introduction to Factory Method Pattern

该设计的缺点:

This type of design has the following disadvantages:

a) High coupling (高耦合).

Client class has a **high degree of coupling** with the service provider class hierarchy.

b) Inelegant statement (难看的条件语句).

The Client class may contain **inelegant** conditional statements.

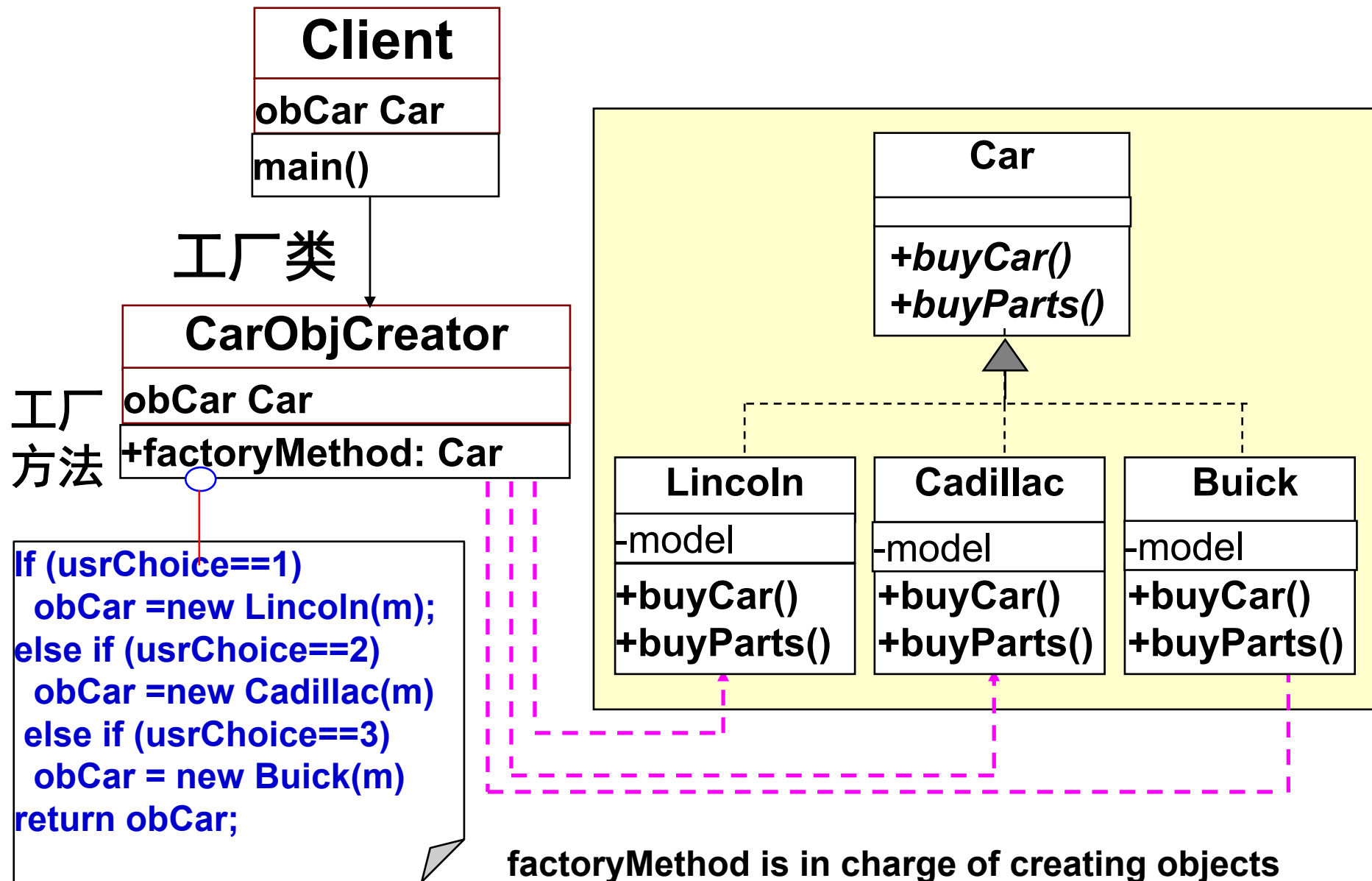
c) Client should fully know the class (客户程序需要知道服务类的全部细节) .

The Client object must know the existence and the functionality offered by each class in the service provider class hierarchy.

Introduction to Factory Method Pattern

- 怎样改善以上设计?
- In such cases, use the **Factory Method Pattern**, which recommends
 - encapsulating the functionality required to select and instantiate an appropriate class inside a designated method called a *factory method* (将选择与初始化一个合适的类的功能封装在一个专门的类的一个专门的方法中-委托) .

Introduction to Factory Method Pattern



Introduction to Factory Method Pattern

Functionalities of the factory method (工厂方法的功能):

- Thus, a factory method can be defined as a method in a class that:
 - **选择一个类**: Selects an appropriate class from a class hierarchy based on the application context and other influencing factors
 - **创建该类的对象**: Instantiates the selected class and
 - **以超类的类型返回该对象**: returns it as an instance of the parent class type
 - **不负责调用**

Introduction to Factory Method Pattern

Advantages of the factory method 工厂方法的优点

1. Clean Client program (清洗客户程序)

Application objects can **make use of** the factory method to get access to the appropriate class instance. This eliminates the need for an application object to deal with the varying class selection criteria.

Introduction to Factory Method Pattern

2. Hide details in instantiating an object. (隐藏初始对象的繁杂的细节)

The factory method also implements any special mechanisms required to instantiate the selected class. (工厂方法以特殊的方式实现初始化某类)

- 不同的子类以不同的方式初始化。 E.g., when different classes in the hierarchy need to be instantiated in different ways.
- 工厂方法将初始化的细节隐藏起来。 The factory method hides these details from the Client object and eliminates the need for them to deal with these intricacies.

Introduction to Factory Method Pattern

3. **Client doesn't need to know which concrete class has been instantiated (客户类只知道哪种类型的对象被创建了，而不必知道哪个具体的子类被初始化了；客户类只知道父类类型).**

Because the factory method returns the selected class instance as an object of the parent class type, a client program **does not have to be aware of the existence of the classes in the hierarchy.**

以下给出两种不同的工厂方法模式

- 简单工厂方法模式**
- 工厂方法模式**



Theory of the Simple Factory Method Pattern and Design Examples

简单工厂方法模式理论与设计例子

The Simple Factory Method Pattern

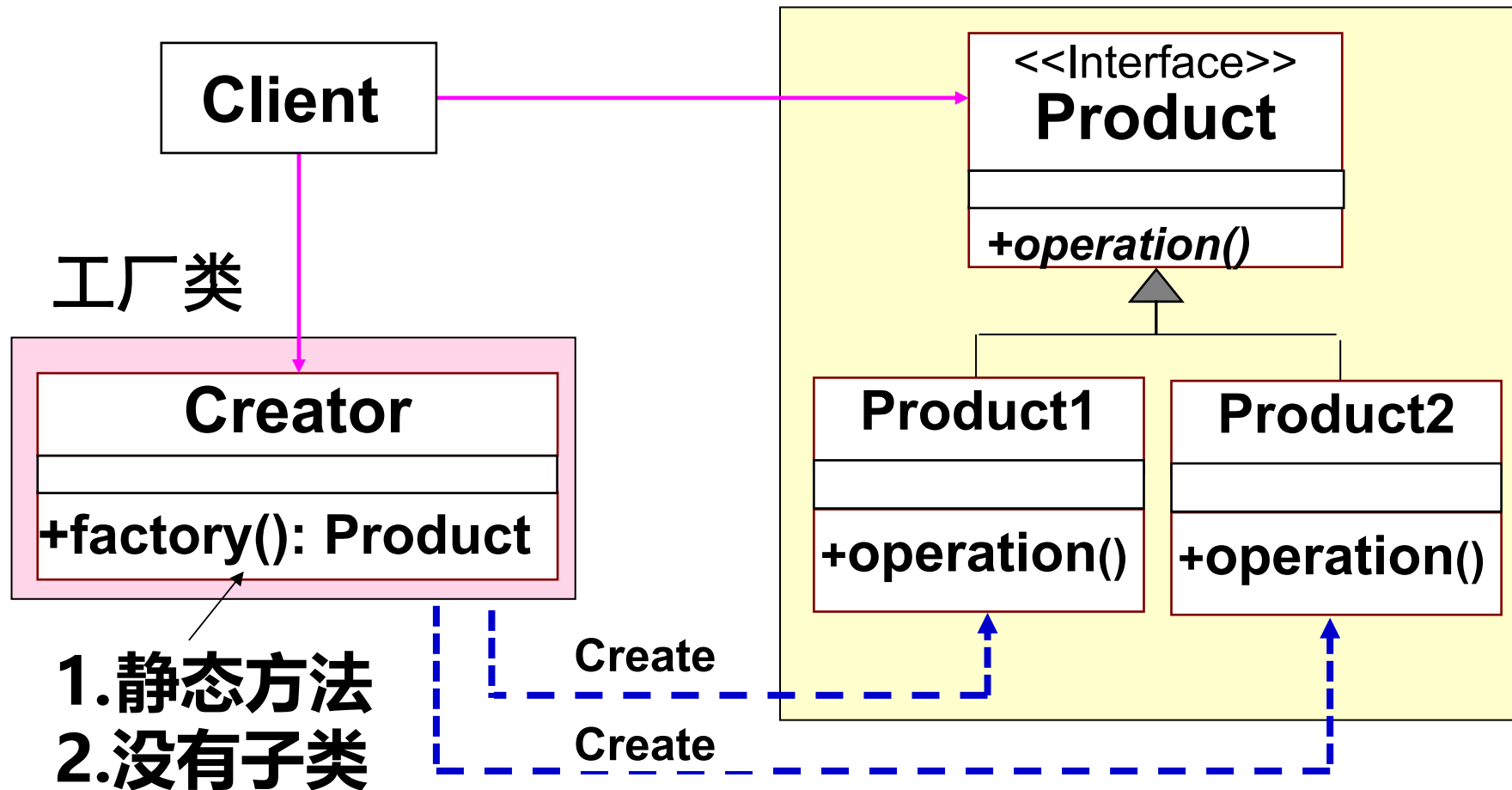


Diagram of Simple factory pattern

The Simple Factory Method Pattern

Creator (工厂类):

- **中心**. the center of the factory pattern
- **包含业务逻辑**. Contains business logic that application needs
- **创建产品类对象**. Factory method creates an object of Product type when called by the client program, and return this object to the caller object

The Simple Factory Method Pattern

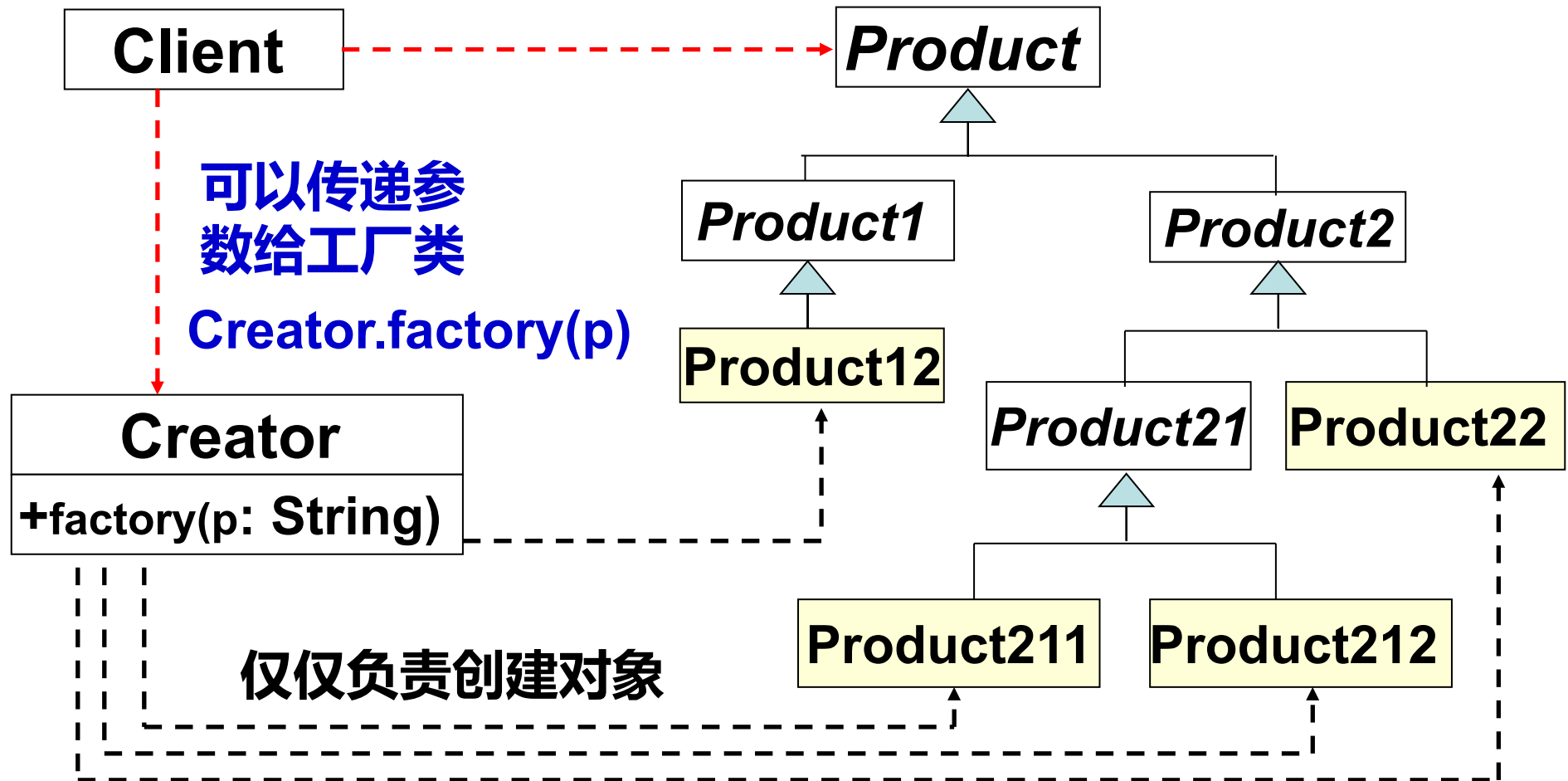
Product:

- Java interface or abstract java class
- Common interface or super class for the concrete subclasses

Concrete product:

- Concrete Java classes to implement interface Product or to extend abstract class Product

The Simple Factory Method Pattern



Simple Factory Pattern Implementation

The Simple Factory Method Pattern

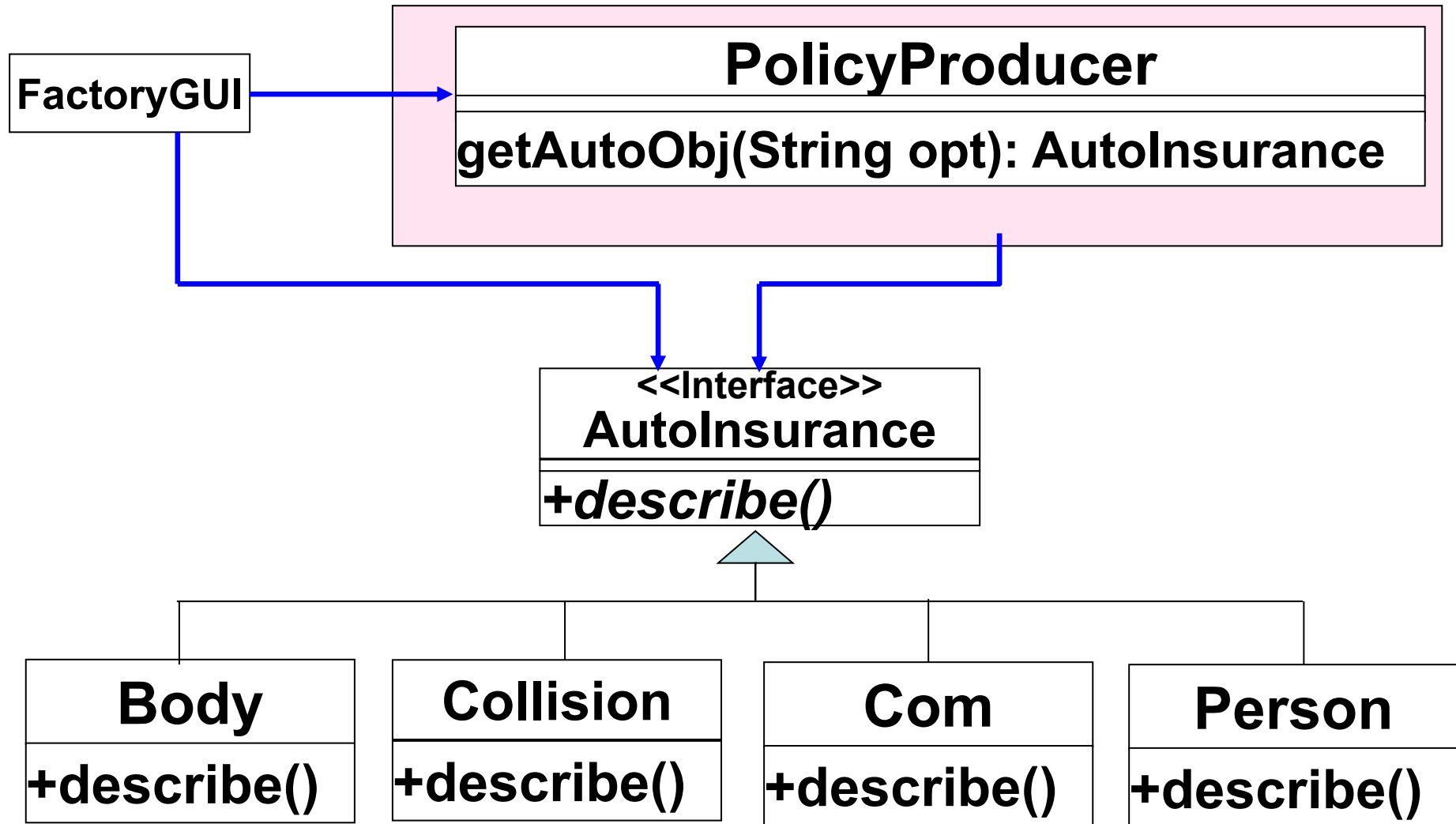
Example of Auto Insurance Policies



【例1】汽车保险介绍程序-用简单工厂方法模式设计

- The following class diagram represents a design
 - using simple factory method pattern to query about the features of different types of auto insurances.

The Simple Factory Method Pattern



使用简单工厂方法模式设计的汽车保险类

The Simple Factory Method Pattern

- **AutoInsurance**: interface to claim various insurance policies (here describe())
- Four subclasses:
 - **Body**: Body Injur Liability (人身保险, 你撞了别人)
 - **Collision**: Collision Coverage (车碰撞险, 你的车损坏了)
 - **Com**: Comprehensive Coverage (综合险)
 - **Person**: Person Injury Protection (驾驶员保险)

implement the interface *AutoInsurance*

The Simple Factory Method Pattern

- **PolicyProducer** is a factory class with a factory method:
getAutoObj(String option)
which will create and return an object of **AutoInsurance** type.
- Actually, it will create and return an object of one of the implementing classes according to the chosen parameter “opt” entered.

The Simple Factory Method Pattern

Java源代码

```
public interface AutoInsurance {  
    abstract String describe();  
}  
  
public class Body implements AutoInsurance{  
    private String description;  
    public String describe(){  
        description = " Body Injur Liability";  
        return description;  
    }  
}
```

The Simple Factory Method Pattern

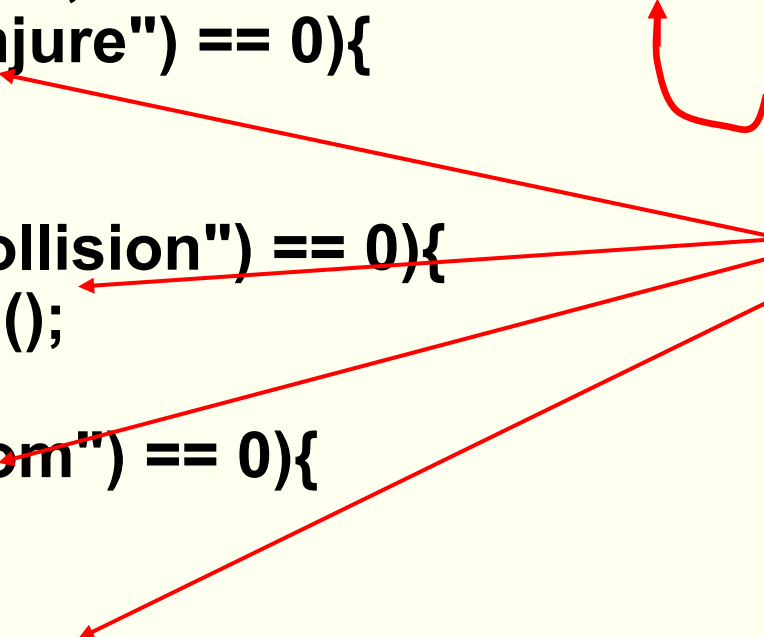
```
public class Collision implements
    AutoInsurance {
    private String description;

    public String describe() {
        description = "Collision Coverage";
        return description;
    }
} //其余2个子类的代码类似，故此省略掉
```


The Simple Factory Method Pattern

```
public class PolicyProducer{  
    public static AutoInsurance getAutoObj(String opt){  
        AutoInsurance policy = null;  
        if(opt.compareTo("bodyInjure") == 0){  
            policy = new Body();  
        }  
        else if(opt.compareTo("collision") == 0){  
            policy = new Collision();  
        }  
        else if(opt.compareTo("com") == 0){  
            policy = new Com();  
        }  
        else if(opt.compareTo("personInjure") == 0){  
            policy = new Person();  
        }  
        return policy; // 以超类的类型返回  
    }  
}
```

根据参
数不同
创建不
同对象



The Simple Factory Method Pattern

用户图形界面的监听器代码

```
class ButtonListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        if (ae.getActionCommand().equals(SHOW)) {  
            String type = (String) getSelectedItem();  
            String option="";
```

```
            if (type.equals(BODYINJURE)) {  
                option="bodyInjure";
```

```
            }  
            else if (type.equals(COLLISION)) {  
                option="collision";  
            }
```

```
            ...  
            AutoInsurance ai = PolicyProducer.getAutoObj(option);  
            String desc = ai.describe();  
            txtForInfo.setText(desc);  
        }
```

```
    }
```

创建对象的责任就
交给
PolicyProducer了



The Simple Factory Method Pattern

Advantages of the Simple Factory method Pattern (简单工厂方法模式的优点)

- a) **一些逻辑被放在了工厂类里面 (Some logics are put into the factory class)**

The factory class contains necessary logic, which can decide which instances of the product class to create and when to create

- b) **客户类不用自己创建对象 (Client is free from creating objects)**

The client class doesn't have to create object of the product classes.

- c) **责任分离 (Responsibility separation)**

The simple factory pattern have realized the **responsibility separation**.

The Simple Factory Method Pattern

Disadvantages of the Simple Factory method Pattern (简单工厂方法模式的缺点)

- Adding a new concrete subclass to Product class hierarchy is difficult (添加Product子类比较困难).
 - adding new sub classes in the Product class hierarchy will need to modify the source code of the factory class
 - 即，需要在工厂方法里面的代码再增加一个条件语句



[Back](#)

Theory of the Factory Method Pattern and Design Examples

工厂方法模式理论与设计例子

Factory Method Pattern

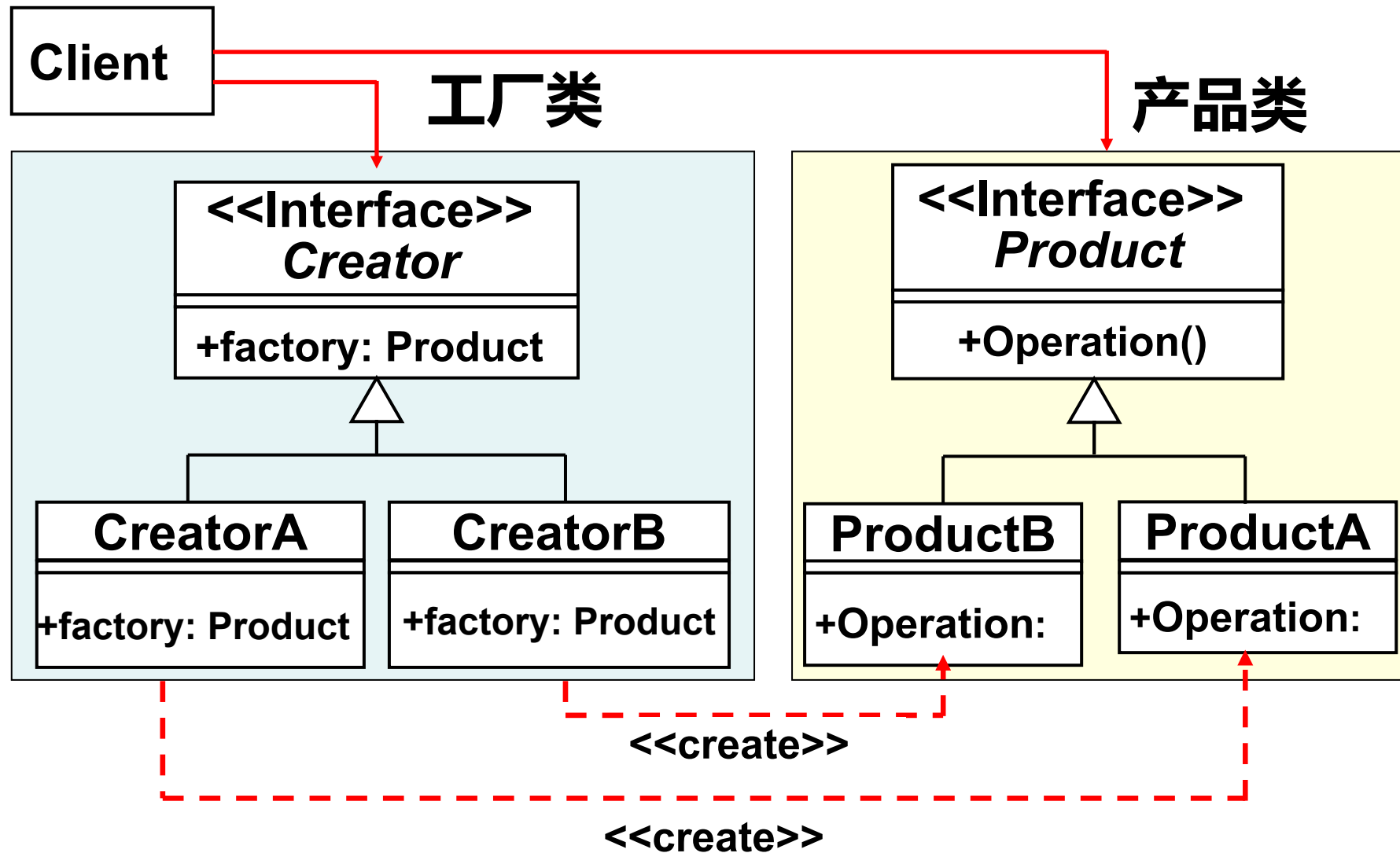


Diagram of Factory method pattern

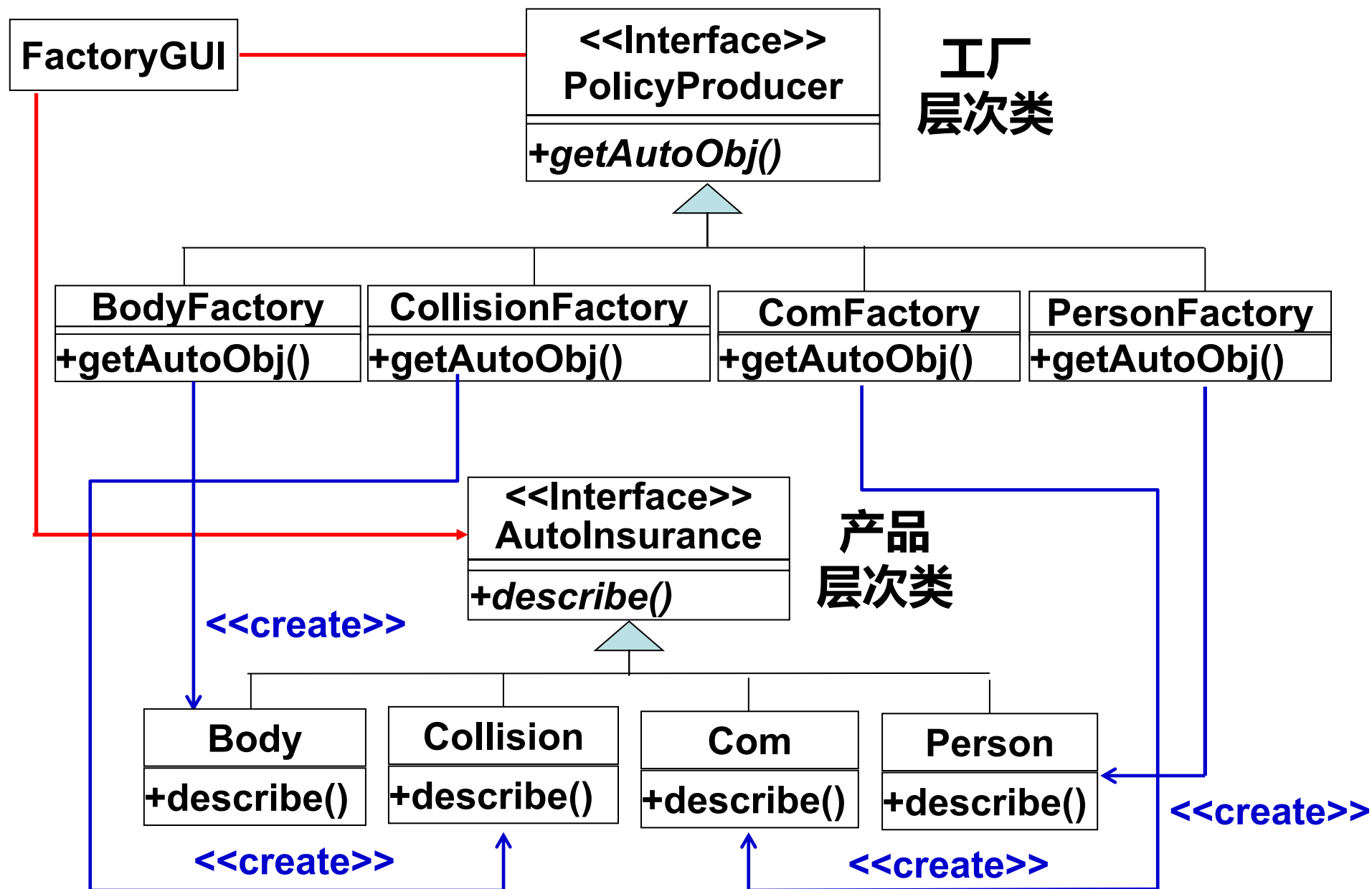
Example of Auto Insurance Policies 汽车保险的例子



【例2】汽车保险介绍程序-用工厂方法模式设计

- The following class diagram represents a design in factory method pattern to query about the features of different types of auto insurances.

工厂方法模式-汽车保险的例子



Example of Auto Insurance Policies

汽车保险的例子

工厂层次类

- 接口类 **PolicyProducer**.
 - 实现类 **BodyFactory**
 - 实现类 **CollisionFactory**
 - 实现类 **ComFactory**
 - 实现类 **PersonFactory**

Example of Auto Insurance Policies

汽车保险的例子

Source Code

```
public interface AutoInsurance {  
    abstract String describe();  
}
```

```
public class Body implements AutoInsurance {  
    private String description;
```

```
    public String describe() {  
        description = " Body Injur Liability:";  
        return description;  
    }  
}
```

其余的产品子类的实现类似，故此处省略。

Example of Auto Insurance Policies

汽车保险的例子

```
public interface PolicyProducer {  
    public AutoInsurance getAutoObj();  
}
```

```
public class BodyFactory implements  
    PolicyProducer {  
    public AutoInsurance getAutoObj() {  
        return new Body();  
    }  
}
```

工厂方法无条件语句

Example of Auto Insurance Policies

汽车保险的例子

```
public class CollisionFactory  
implements PolicyProducer {  
    public AutoInsurance getAutoObj() {  
        return new Collision();  
    }  
}
```

**工厂方法无条
件语句**

其它两个工厂子类代码类似，故此处省略

Example of Auto Insurance Policies

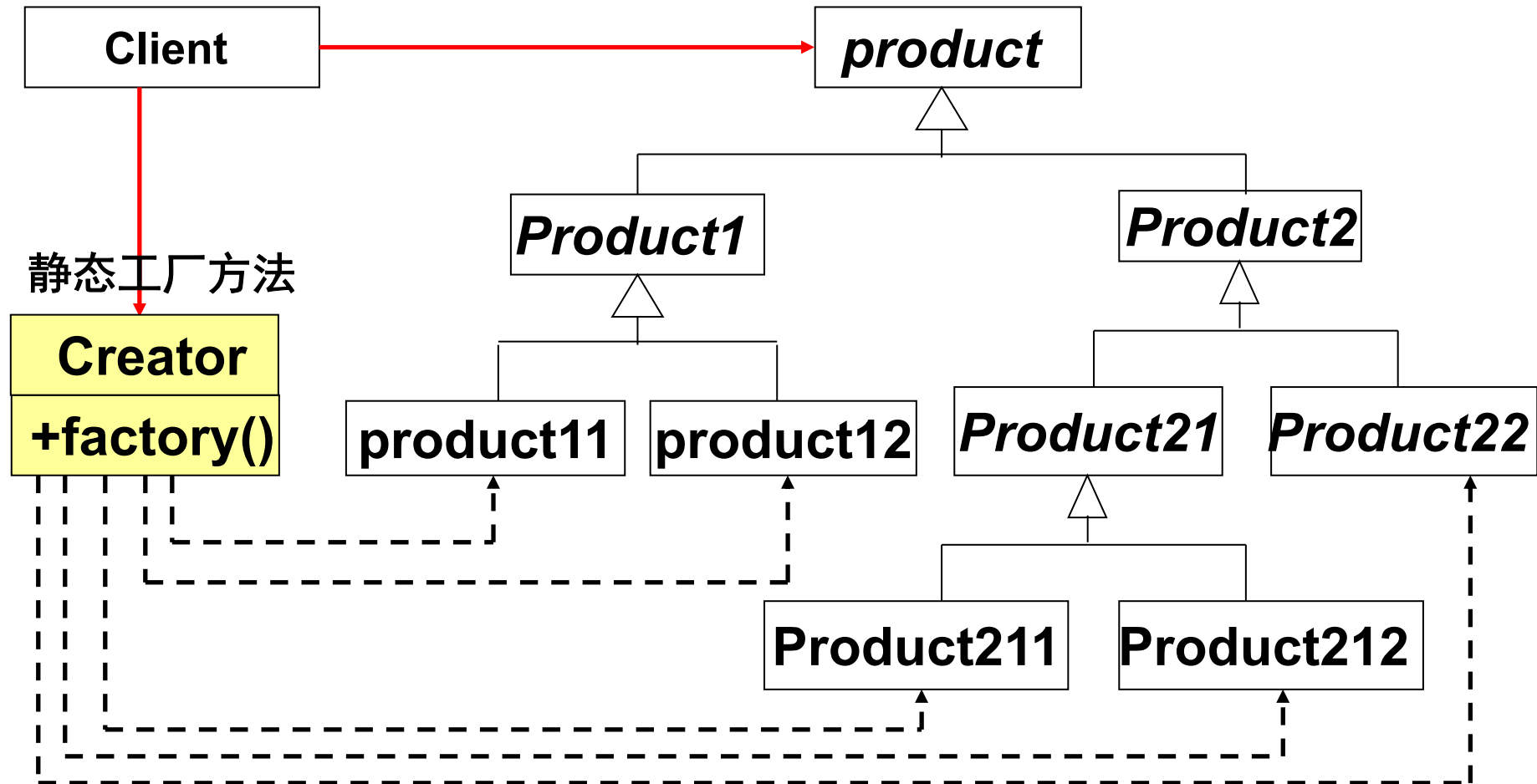
```
public class FactoryMethodGUI extends JFrame {  
    class ButtonListener implements ActionListener {  
        public void actionPerformed(ActionEvent ae) {  
  
            if (ae.getActionCommand().equals(SHOW)) {  
                String t = (String) cmbInsuranceType.getSelectedItem();  
                PolicyProducer pp=null;  
  
                //创建工厂子类对象  
                if (t.equals(BODYINJURE)) { pp = new BodyFactory(); }  
                if (t.equals(COLLISION)) { pp = new CollisionFactory (); }  
                if (t.equals(PERSONINJURE)) { pp = new PersonFactory (); }  
                if (t.equals(COMP)) { pp = new ComFactory (); }  
  
                AutoInsurance ai = pp.getAutoObj(); //获得了产品子类对象  
                String desc = ai.describe();  
            }  
        }  
    }  
}
```



[Back](#)

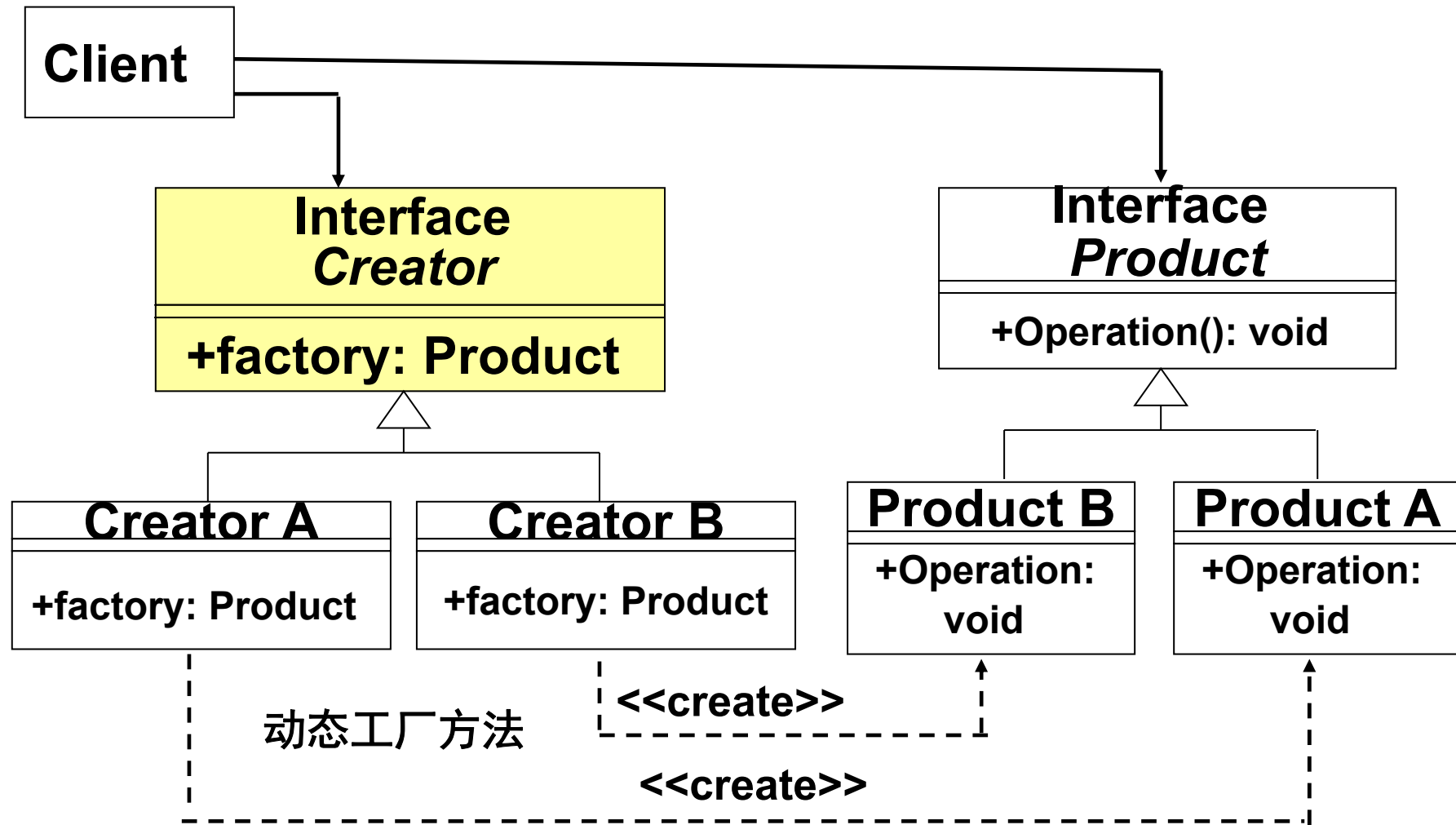
Further Discussion of the Factory Method Pattern

Comparison of Simple Factory Method Pattern and Factory Method Pattern



Simple Factory method pattern

Comparison of Simple Factory Method Pattern and Factory Method Pattern



Factory method pattern

Comparison of Simple Factory Method Pattern and Factory Method Pattern

a) 中心不同 Different Centers

- the **center** of the simple factory pattern is a **concrete factory class**.
- The **center** of a factory pattern is the **abstract factory class** (or interface),

Comparison of Simple Factory Method Pattern and Factory Method Pattern

b) 工厂方法不同，一个静态，一个动态。

- In the **simple factory method pattern**, the factory method is **static** and
- in the **factory method pattern**, the factory methods are **dynamic** and are distributed in the concrete factory classes

Comparison of Simple Factory Method Pattern and Factory Method Pattern

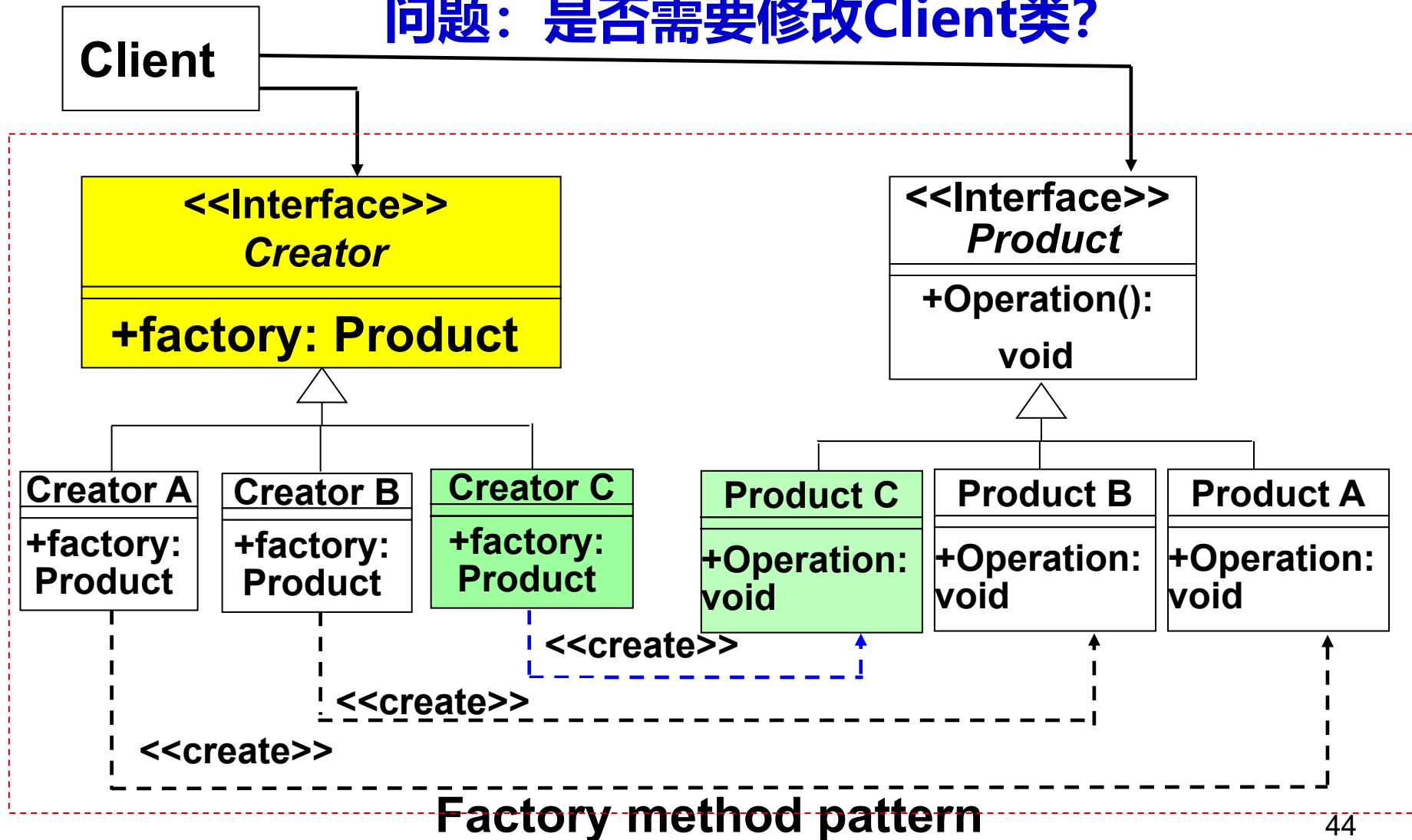
c) 工厂方法模式的可扩展性 (Extensibilities)

If a new product is added to the product hierarchy, then we only need to add a sub concrete class to the product hierarchy and a sub class to the factory hierarchy.

- **it is not necessary to modify the abstract factory and the existing subclasses of the concrete factory method (工厂类的已有源代码不需要改变)**

Comparison of Simple Factory Method Pattern and a Factory Method Pattern (比较)

问题：是否需要修改Client类？



Comparison of Simple Factory Method Pattern and Factory Method Pattern

d) 两个工厂方法返回类型都是超类类型 (Return type: super class type)

The factory methods, in the simple factory method pattern and the factory method pattern, return objects of **type Product**, not an object of the concrete product class.

- The client doesn't have to know the type of the real type (这样, client类就不知道到底哪个产品子类的对象被创建了).

Comparison of Simple Factory Method Pattern and Factory Method Pattern

- e) **工厂方法模式支持开闭原则，但是简单工厂方法模式不支持开闭原则。 Factory Method Pattern supports the open-closed principle, but simple factory method doesn't support the open-closed principle.**
 - **Open for extension and closed for modification**

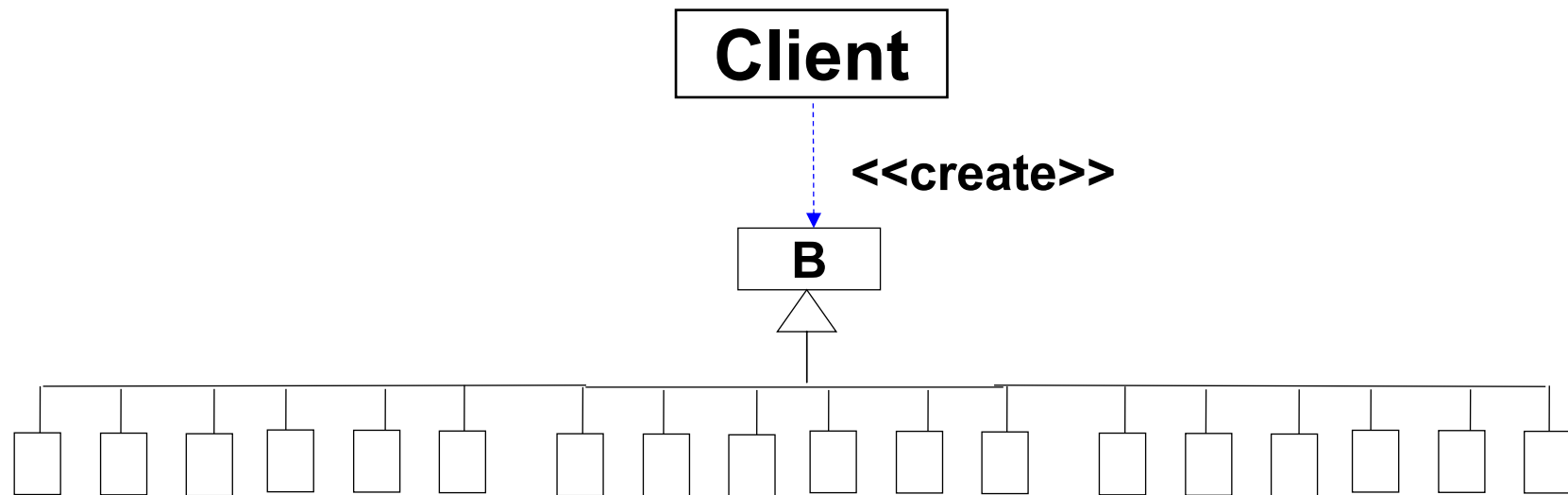


When to Use a Factory Pattern

何时使用工厂方法模式？

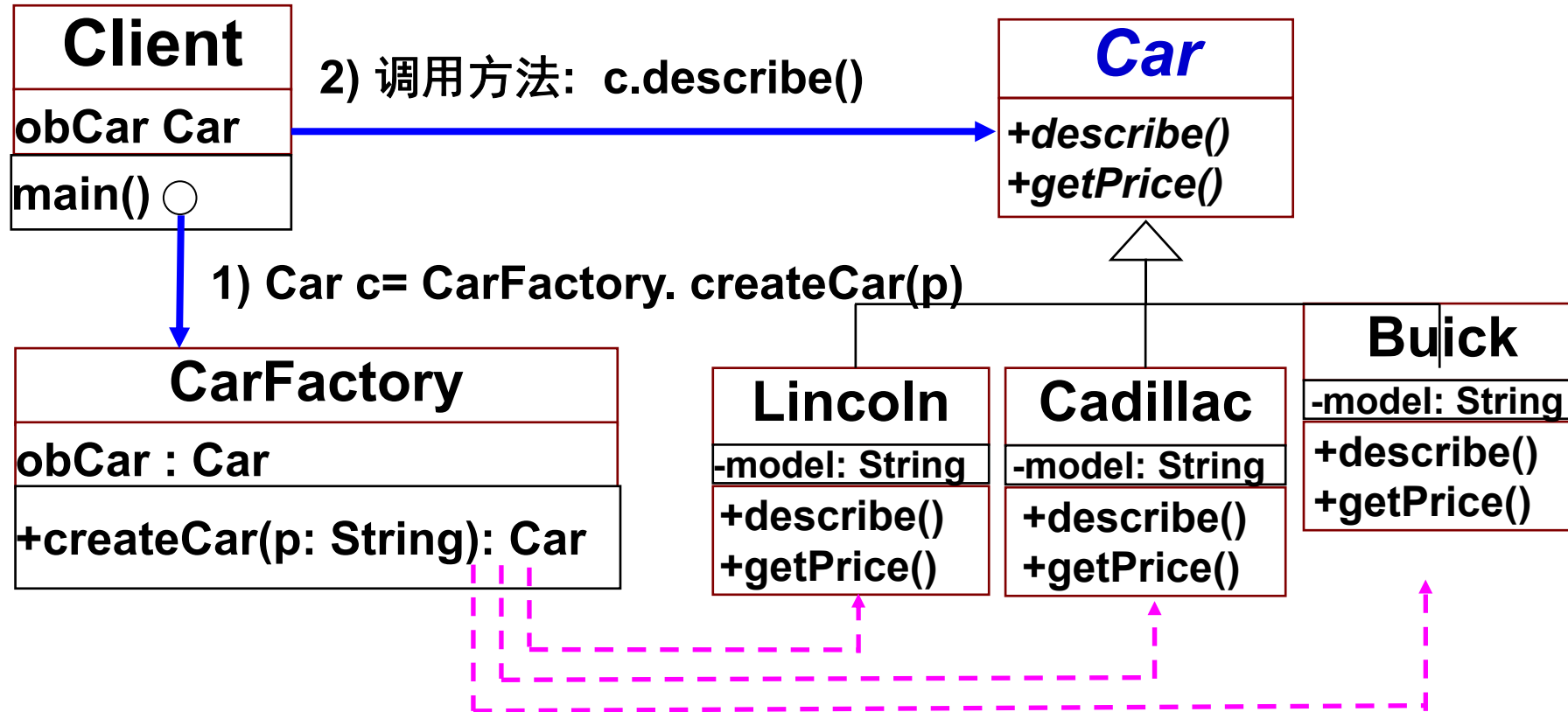
You should consider using a Factory method pattern when

- A class contains a lot of conditional statements for creating objects (大量条件语句) .
- A class uses its subclasses to specify which objects it creates (即, 创建对象的任务分散在客户类的子类中, 比较乱; 可使用工厂方法模式将创建产品类的子类对象的责任分离出来) .
- You want to **localize** the knowledge of which class gets created.



1. 简单工厂方法模式的实现

学生自学



将参数p传递给工厂类，工厂方法根据此参数创建合适的类的对象，并且以Car类型返回给调用者。

2. 工厂方法模式的一种实现方法

学生自学

