Object Oriented Analysis & Design
面向对象分析与设计

Lecture_10 GOF设计模式（二）

1）策略模式　　2）工厂

主讲: 姜宁康 博士

4、工厂模式 Factory Design Pattern...

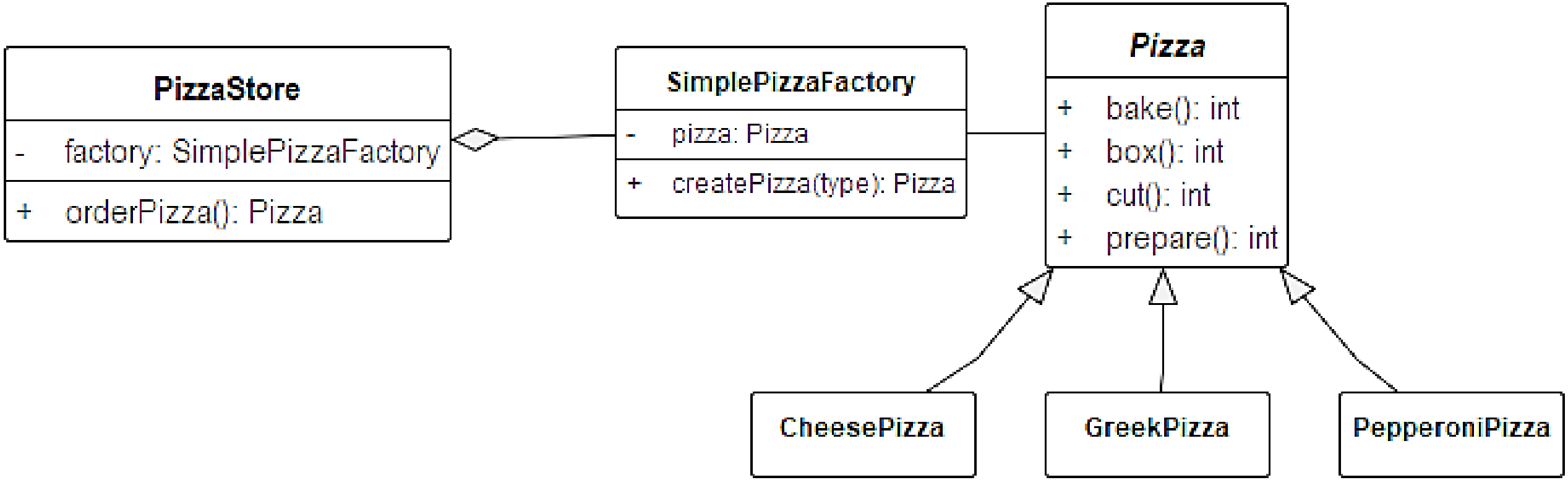
- 如何设计一个方法，去实例化不同的对象，也可以实例化今后新开发的类，这样做不需要修改该方法的代码

4.1 案例背景

- 背景
  - 在高科技的“对象村”拥有一家pizza店，负责为附近的居民提供比萨
  - 设计一个仿真的pizza店软件系统
- 需求继续变化
  - “对象村”业务发展，广开专卖店、连锁店 Pizza Branch
  - 在不同的城市开店 Branches in different cities
    - 必须保证pizza的质量标准 Must ensure quality of pizza
    - 必须适应当地口味 Must account for regional differences (NY, Chicago..)
  - 要求分店充分发挥店主 PizzaStore 的优点
    - 规定：以标准的方式准备披萨(prepare pizzas)，但是
      - 在NY有一个工厂，专门生产 NY 风味的 pizza
      - 在芝加哥Chicago有一个工厂，专门生产芝加哥风味的 pizza

4.1 案例背景

- 复习：简单工厂模式



4.1 案例背景

```
NYPizzaFactory nyFactory = new NYPizzaFactory ( );
PizzaStore nyStore = new PizzaStore (nyFactory);
nystore.order ("Veggie");

ChicagoPizzaFactory CFactory = new ChicagoPizzaFactory ( );
PizzaStore CStore = new PizzaStore (CFactory);
Cstore.order ("Veggie");
```

创建工厂生产NY风味的披萨pizza

在NY开一家 PizzaStore，把NY工厂的联系方式给它

…这样，当生产Pizza时，就得到了 NY 风味的披萨

使用简单工厂模式的问题:

-- 分店使用总店的工厂生产Pizza，但使用自己的流程进行烘烤等

-- 另外，每个分店“需要一些自我改进的空间”

总店不想知道每个分店往Pizza里面加了些啥（具体细节不需要告诉总店，分店自己知道即可）

但是，总店又需要对分店有一些控制（质量控制！）

结论：真正需要的是一套机制、框架 a framework，把PizzaStore与pizza的制作联系在一起，又允许一定的灵活性。

4.2 框架设计 A Framework

- 需要一种机制，为PizzaStore“本地化”制作披萨的所有活动，同时又使分店能自由加入本地风味

```
public abstract class PizzaStore {
    public Pizza orderPizza (String type) {
        Pizza pizza;

        pizza =  createPizza (type);

        pizza.prepare ( );
        pizza.bake ( );
        pizza.cut ( );
        pizza.box ( );

        return pizza;
    }

    abstract createPizza (String type);
}
```

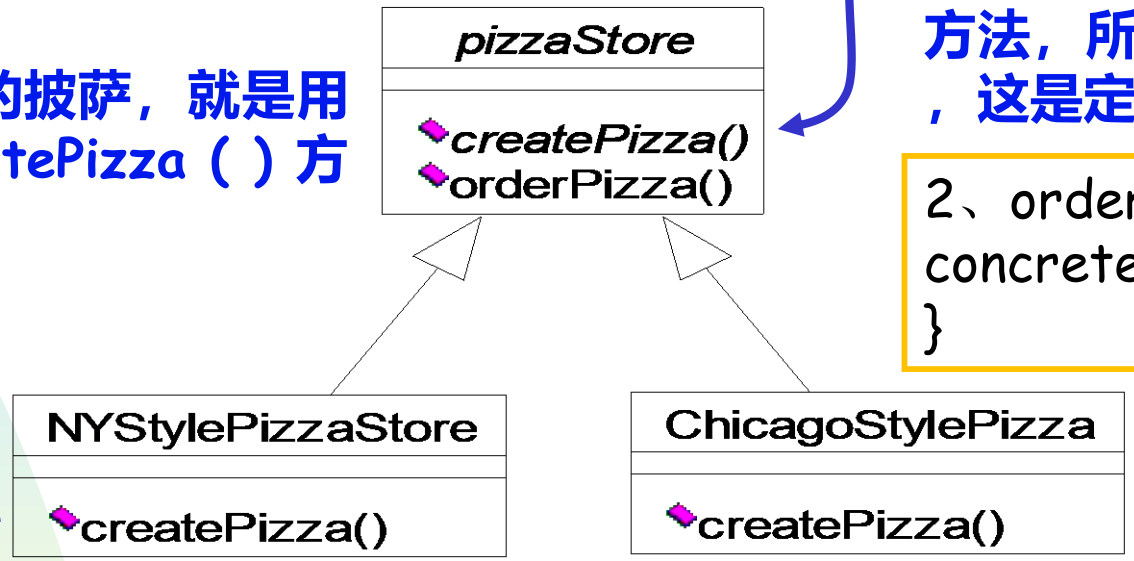
createPizza()又回到 PizzaStore了，这次不是一个 Factory 对象， 但createPizza()是一个抽象函数

这些功能仍然不变

这个方法，又称为工厂方法，位于PizzaStore

每个子类决定自己需要用到哪个工厂 Factory，所有子类都必须实现 createPizza（）方法

4.2 框架设计 Allowing the subclasses to decide...



1、每个子类覆盖实现 createPizza ( ) 方法，所有的子类使用 orderPizza ( )，这是定义在 PizzaStore的

```
2、orderPizza () {
concreteStore.createPizza()
}
```

3、如果分店想要NY风味的披萨，就是用NY子类，它有自己的 createPizza ( ) 方法，创建 NY风味的比萨

```
public Pizza createPizza (type) {
   if (type.equals("cheese"))
       pizza = new NYStyleCheesePizza ( );
   else if (type.equals ("pepperoni"))
       pizza = new NYStylePepperoniPizza ( );
 }
```

4、createPizza () 返回一个比萨，子类全权负责实例化哪个具体的Pizza类

5、创建具体类的对象

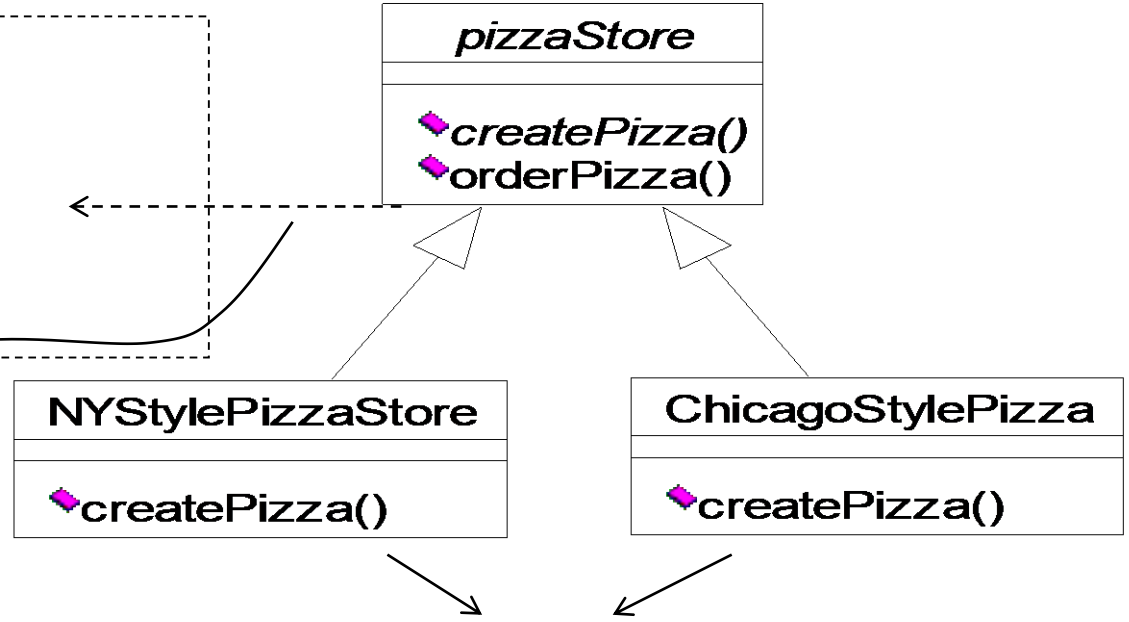
6、注意：在父类的 orderPizza ( )方法中，完全不知道将会创建一个什么样的比萨。它只知道 prepare，bake，cut and box 比萨

4.2 框架设计：子类如何决定呢?

我不明白，PizzaStore的子类只是子类而已，它如何被决定?
我在NYStylePizzaStore中没有看见任何逻辑决策的代码……

1/ orderPizza()方法是在抽象类PizzaStore中而不是在子类中定义的，因此子类在实际运行代码和下pizza订单时由这个方法来统一执行

```
pizza=createPizza();
Pizza->prepare();
Pizza->bake();
Pizza->cut();
Pizza->box();
```



2/ orderPizza() 调用createPizza()来获取一个pizza对象，但是它获取什么类型的pizza呢?
orderPizza() 并不能决定，它也不知道怎么做，那么由谁来做决定呢?

3/ 由下订单对应的PizzaStore来决定生产哪种类型的pizza：createPizza ()

4.3 工厂方法

- “工厂方法”把自己封装在一个子类中，负责对象创建，解耦父类相关的客户代码与子类中的创建对象对码 A “Factory” method handles object creation and encapsulates it in a subclass. This decouples the client code in the superclass from the object creation code in the subclass

工厂方法是抽象的，使得子类负责对象创建职责

工厂方法既可以带参数也可以不带参数，参数用于选择不同的产品（对象）

abstract Product factoryMethod (String type)

工厂方法返回一个产品（对象），供定义在父类中的一个函数使用

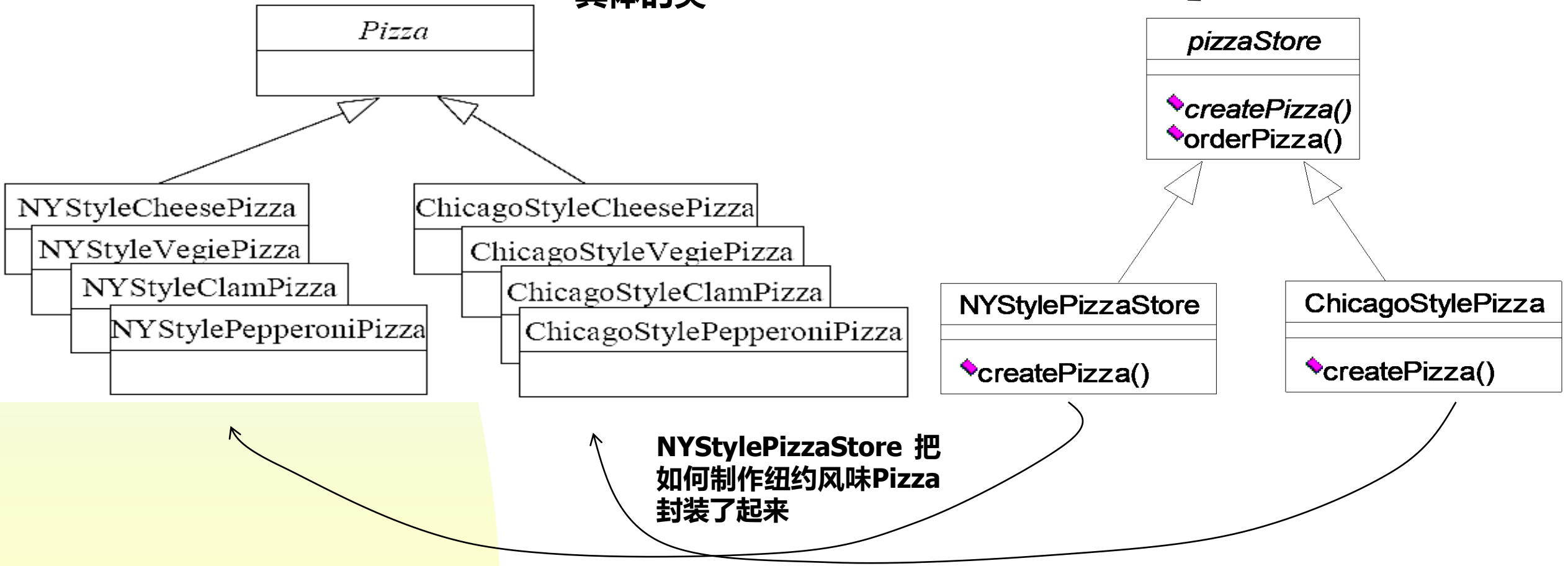
工厂方法隔离了客户（如，父类中的orderPizza ( )）了解创建对象（或者产品）的具体细节

4.4 工厂模式

从平行的角度来看看类的层次：
两个都是抽象类，均可以派生出具体的类

The Product Classes

The Creator Classes

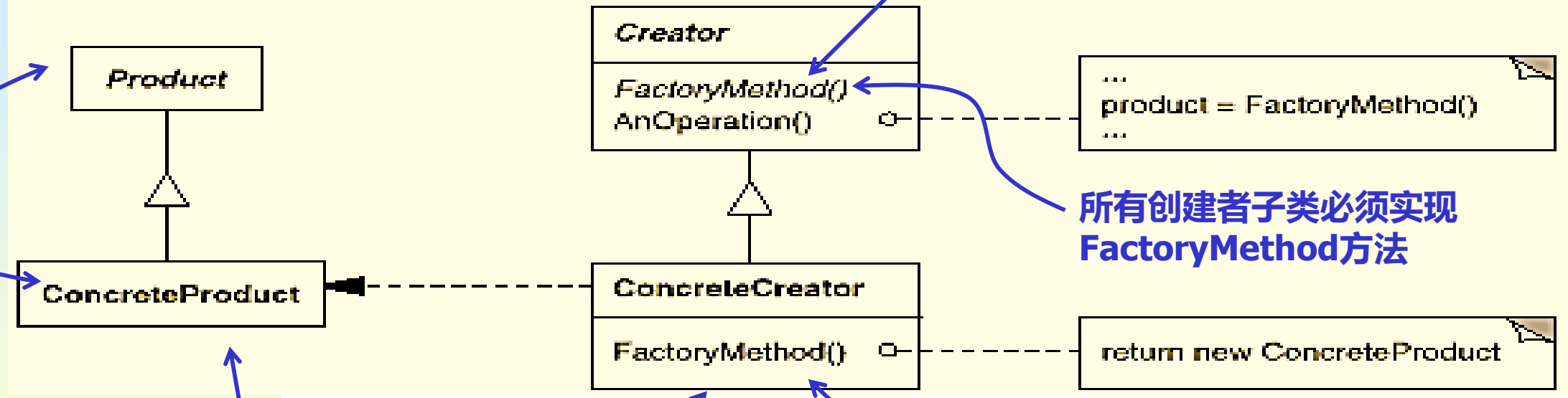


NYStylePizzaStore 把如何制作纽约风味Pizza封装了起来

工厂方法是封装“知识”的关键

4.4 工厂模式

定义：工厂模式定义了一个创建产品对象的工厂接口，将实际创建工作推迟到子类中



所有的产品必须实现同一个接口，这样，使用产品的类可以指向该接口而不是指向具体的类

除了工厂方法是抽象的外，Creator类包含了操作产品的所有方法的实现

所有创建者子类必须实现FactoryMethod方法

ConcreteCreator负责一个或者多个具体产品的创建，只有它才知道如何创建产品

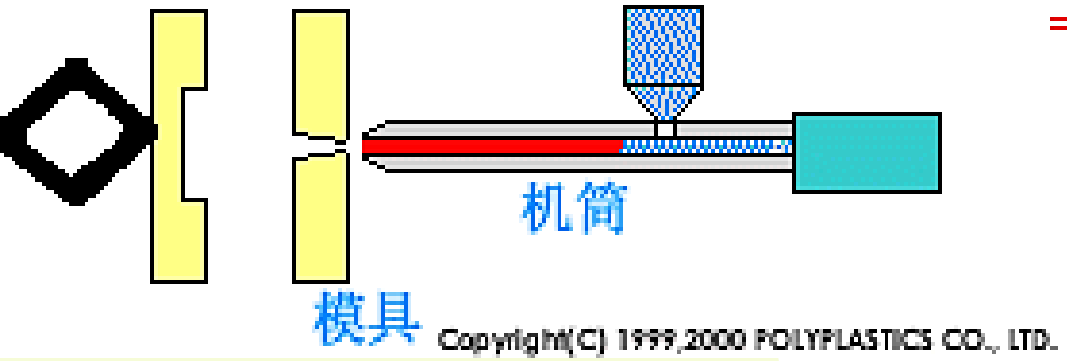
ConcreteCreator 必须实现FactoryMethod，这个方法完成实际的产品创建

4.4 工厂模式要点

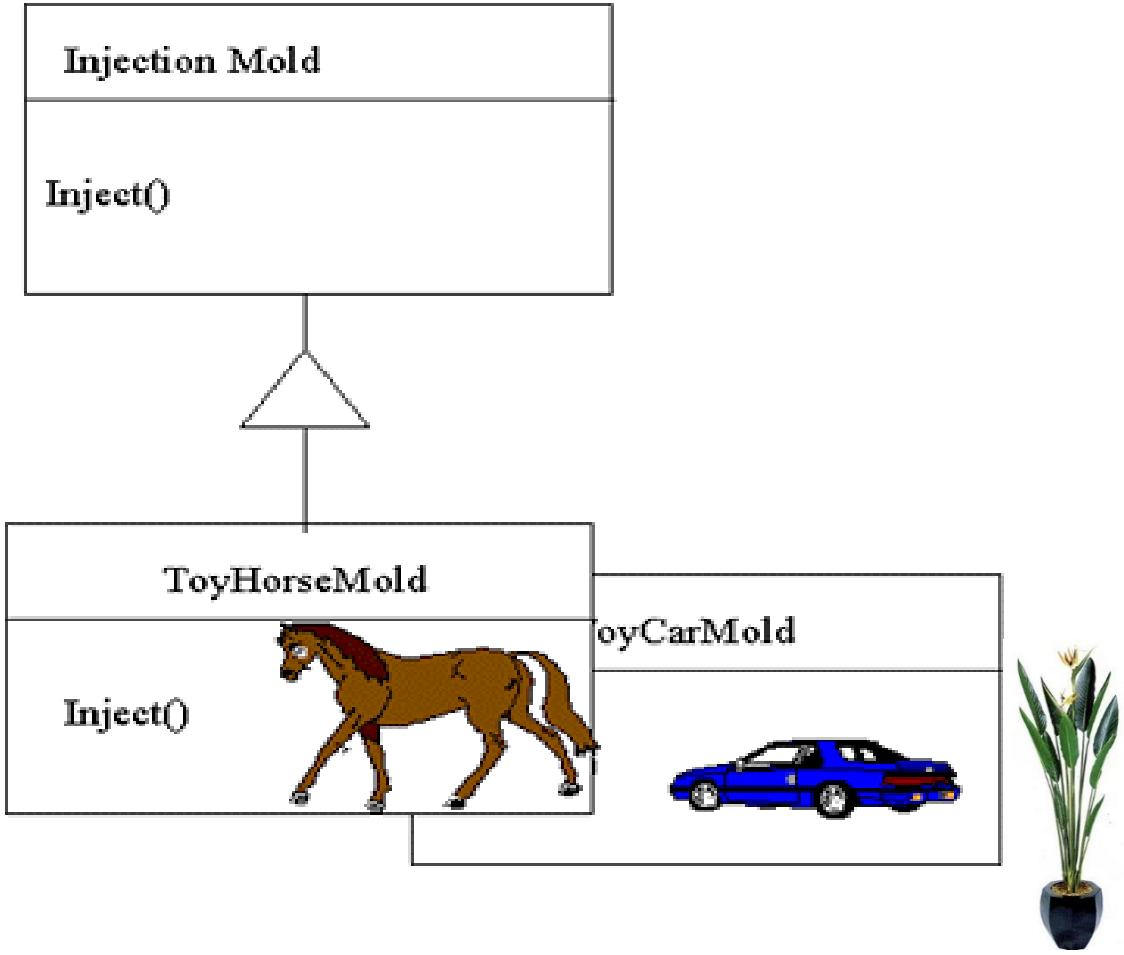
Intent	**Define an interface for creating an object, but let subclasses decide which class to instantiate. Defer instantiation to subclasses. 定义一个创建对象的接口，但是子类决定具体创建哪个类的对象，实例化决策推迟到子类去进行**
Problem	**A class needs to instantiate a derivation of another class, but doesn't know which one. Factory Method allows a derived class to make this decision**
Solution	**A derived class makes the decision on which class to instantiate and how to instantiate it**
Participants and collaborators	**Product is the interface for the type of object that the Factory Method creates. Creator is the interface that defines the Factory Method**
Consequences	**Clients will need to subclass the Creator class to make a particular ConcreteProduct**
Implementation	**Use a method in the abstract class that is abstract (pure virtual in C++). The abstract class' code refers to this method when it needs to instantiate a contained object but does not know which particular object it needs**

4.5 例：工厂模式在生活中

- 塑料玩具制造
  - 各自形状的磨具
  - 注塑系统
  - 冲床



⇒合模
注射
保压
冷却
开模
取出制品



- 本讲结束