**抽象工厂模式：**定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类。

**举个例子：开Pizza店**

**为了满足面向对象村村民对比萨的向往，你决定开一家Pizza店。**

**你的代码可能这样子写：**

[复制代码](javascript:void(0);)

Pizza orderPizza(){

Pizza pizza = new Pizza();

pizza.prepare();//比萨制作的过程

pizza.bake();

pizza.cut();

pizza.box();

return pizza;

}

[复制代码](javascript:void(0);)

**但是你的比萨店肯定不止一种比萨吧。所以你有修改了代码：**

[复制代码](javascript:void(0);)

Pizza orderPizza(String type) {

Pizza pizza;

if(type.equals("cheese")) {//下面一共有三种比萨

pizza = new CheesePizza();

}else if(type.equals("greek")) {

pizza = new GreekPizza();

}else if(type.equals("pepperoni")) {

pizza = new PepperoniPizza();

}

pizza.prepare();

pizza.bake();

pizza.cut();

pizza.box();

return pizza;

}

[复制代码](javascript:void(0);)

**但是过了几个月你发现面向对象村的村民希望加入新的口味，于是你又一次修改了代码：**

[复制代码](javascript:void(0);)

Pizza orderPizza(String type) {

Pizza pizza;

if(type.equals("cheese")) {//你有添加了Clam（蛤蜊比萨） Veggie(素食比萨)

pizza = new CheesePizza();

}else if(type.equals("greek")) {

pizza = new GreekPizza();

}else if(type.equals("pepperoni")) {

pizza = new PepperoniPizza();

}else if(type.equals("clam")) {

pizza = new ClamPizza();

}else if(type.equals("veggie")) {

pizza = new VeggiePizza();

}

}

[复制代码](javascript:void(0);)

**但是这样做明显违背了“开放封闭原则”，那么我们不如把创建比萨的代码移到另一个对象中，这个类专门来创建比萨。**

**我们称它为简单工厂：**

[复制代码](javascript:void(0);)

public class SimplePizzaFactory {

public Pizza createPizza(String type) {//通过传入的type来创建不同的比萨

Pizza pizza = null;

}

if(type.equals("cheese")){

pizza = new CheesePizza();

}else if(type.equals("pepperoni")) {

pizza = new PepperoniPizza();

}else if(type.equals("clam")){

pizza = new ClamPizza();

}else if(type.equals("veggie")){

pizza = new VeggiePizza();

}

return pizza;

}

[复制代码](javascript:void(0);)

**现在重现开店咯：**

[复制代码](javascript:void(0);)

public class PizzaStore(){

SimplePizzaFactory factory;//实例化一个比萨制造的简单工厂

public PizzaStore(SimplePizzaFactory factory){//构造函数

this.factory = factory;

}

public Pizza orderPizza(String type){

Pizza pizza;

pizza =factory.createPizza(type);//创建不同的比萨

pizza.prepare();//比萨制作过程

pizza.bake();

pizza.cut();

pizza.box();

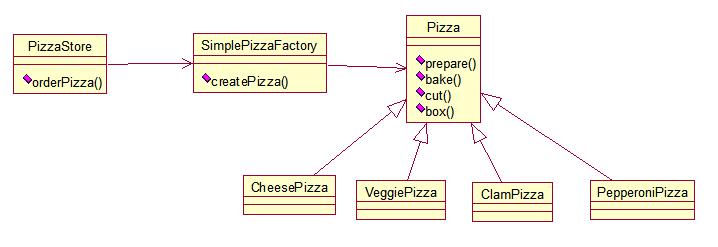
return pizza;

}

}

[复制代码](javascript:void(0);)

**简单模式其实也不是一个设计模式，反而比较像是一种编程习惯。但由于经常被使用，所以我们给它一个“OO Pattern荣誉奖”。不要以为简单工厂不是一个“真正的”模式，就忽略它的用法。让我们来看看新的比萨店类图。**



**由于经营有成，击败了各种竞争者，现在大家希望面向对象的比萨店能够在附近有加盟店。但是由于区域的差异，每家加盟店可能提供的口味不相同，比如说（纽约、芝加哥、加州）。因此为了满足不同区域的消费者的要求，现在需要重新安排我们的Pizza工厂。**

**一个想法就是利用SimplePizzaFactory，写出三个不同的工厂，分别为NYPizzaFactory、ChicagoPizzaFactory、CaliforniaPizzaFactory。**

[复制代码](javascript:void(0);)

NYPizzaFactory nyFactory = new NYPizzaFactory();

PizzaStore nyStore = new PizzaStore(nyFactory);

nyStore.orderPizza("Veggie");

ChicagoPizzaFactory chicagoFactory = new ChicagoPizzaFactory();

PizzaStore chicagoStore = new PizzaStore(chicagoFactory);

chicagoStore.orderPizza("Veggie");

[复制代码](javascript:void(0);)

**但是问题来了，每个加盟店都开始使用属于自己的一套制作pizza流程，因此我们需要给比萨店构造一个合适的框架：**

[复制代码](javascript:void(0);)

public abstract class PizzaStore() {//现在PizzaStore是抽象的

public Pizza orderPizza(String type){

Pizza pizza;

pizza = createPizza(type);

pizza.prepare();

pizza.bake();

pizza.cut();

pizza.box();

return pizza;

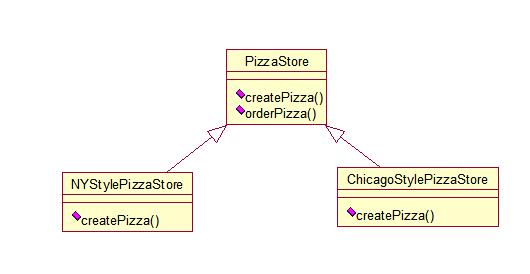
}

abstract Pizza createPizza(String type);//“工厂方法”也是抽象的

}

[复制代码](javascript:void(0);)

**我们需要让每一个加盟店自己决定自己的pizza制作流程，依旧是让子类来作出决定：**



[复制代码](javascript:void(0);)

Public Pizza createPizza(type) {//纽约的pizza店创造属于自己的独特口味

if(type.equals("cheese")) {

pizza = new NYStyleCheesePizza();

}else if(type.equals("greek")) {

pizza = new NYStyleGreekPizza();

}else if(type.equals("pepperoni")) {

pizza = new NYStylePepperoniPizza();

}else if(type.equals("clam")) {

pizza = new NYStyleClamPizza();

}else if(type.equals("veggie")) {

pizza = new NYStyleVeggiePizza();

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

Public Pizza createPizza(type) {//芝加哥的pizza店创造属于自己的独特口味

if(type.equals("cheese")) {

pizza = new ChicagoStyleCheesePizza();

}else if(type.equals("greek")) {

pizza = new ChicagoStyleGreekPizza();

}else if(type.equals("pepperoni")) {

pizza = new ChicagoStylePepperoniPizza();

}else if(type.equals("clam")) {

pizza = new ChicagoStyleClamPizza();

}else if(type.equals("veggie")) {

pizza = new ChicagoStyleVeggiePizza();

}

}

[复制代码](javascript:void(0);)

**现在让我们开始加盟店:**

[复制代码](javascript:void(0);)

public class NYPizzaStore extends PizzaStore {//纽约的pizza点

Pizza createPizza(String item) {

if(item.equals("cheese")){

return new NYStyleCheesePizza();

}else if(item.equals("veggie")){

return new NYStyleVeggiePizza();

}else if(item.equals("clas")){

return new NYStyleClamPizza();

}else if(item.equals("pepperoni")){

return new NYStylePepperoniPizza();

}else return null;

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

public class ChicagoPizzaStore extends PizzaStore {//芝加哥的pizza店

Pizza createPizza(String item) {

if(item.equals("cheese")){

return new ChicagoStyleCheesePizza();

}else if(item.equals("veggie")){

return new ChicagoStyleVeggiePizza();

}else if(item.equals("clas")){

return new ChicagoStyleClamPizza();

}else if(item.equals("pepperoni")){

return new ChicagoStylePepperoniPizza();

}else return null;

}

}

[复制代码](javascript:void(0);)

**声明一个工厂方法：**

[复制代码](javascript:void(0);)

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 Pizza createPizza(String type);

//其他方法

}

[复制代码](javascript:void(0);)

**我们忽略了很重要的一件事：比萨本身！**

[复制代码](javascript:void(0);)

public abstract class Pizza {

String name;//比萨名

String dough;//比萨面团类型

String sauce;//比萨酱料类型

ArrayList toppings = new ArrayList();//比萨一套佐料

void prepare() {

System.out.println("Preparing "+name);

System.out.println("Tossing dough...");

System.out.println("Adding sauce...");

System.out.println("Adding toppings: ");

for(int i = 0 ; i < toppings.size(); i++ ){

System.out.println(" "+toppings.get(i));

}

}

void baek(){

System.out.println("Bake for 25 minutes at 350");

}

void cut(){

System.out.println("Cutting the pizza into diagonal slices");

}

void box(){

System.out.println("Place pizza in offical PizzaStore box");

}

public String getName(){

return name;

}

}

[复制代码](javascript:void(0);)

**现在来定义纽约和芝加哥风味的比萨：**

[复制代码](javascript:void(0);)

public class NYStyleCheesePizza extends Pizza {

public NYStyleCheesePizza(){

name = "NY Style Sauce and Cheese Pizza";

dough= "Thin Crust Dough";

sauce = "Marinara Sauce";

toppings.add("Grated Reggiano Cheese");

}

}

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

public class ChicagoStyleCheesePizza extends Pizza {

public ChicagoStyleCheesePizza(){

name = "Chicago Style Deep Dish and Cheese Pizza";

dough= "Extra Thick Crust Dough";

sauce = "Plum Tomato Sauce";

toppings.add("Shredded Mozzarella Cheese");

}

void cut(){

System.out.println("Cutting the pizza into square slices");

}

}

[复制代码](javascript:void(0);)

**现在我们可以测试一下加盟店的运营情况了：**

[复制代码](javascript:void(0);)

public class PizzaTestDrive(){

public static void main(String[] args) {

PizzaStore nyStore = new NYPizzaStore();//纽约pizza店

PizzaStore chicagoStore = new ChicagoPizzaStore();//芝加哥pizza店

Pizza pizza = nyStore.orderPizza("cheese");

System.out.println("I ordered a "+pizza.getName()+"\n");

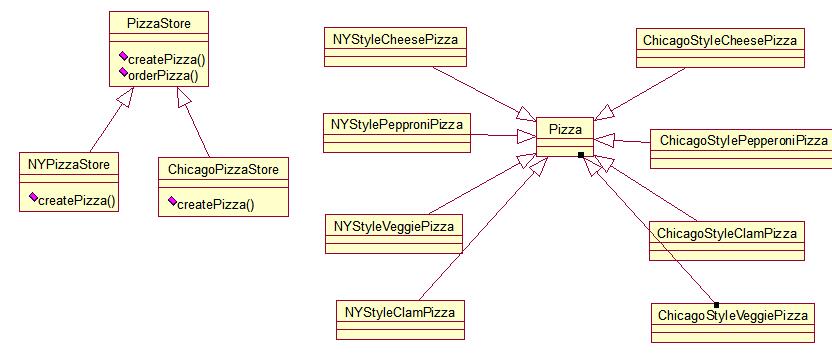
pizza = new chicagoStore.orderPizza("cheese");

System.out.println("You ordered a "+pizza.getName()+"\n");

}

}

[复制代码](javascript:void(0);)



**上面便是我们的工厂模式用途之一；现在我们正式介绍我们强大的工厂模式：**

