PADRÕES DE PROJETO

FACTORY

FACTORY

Padrões Factory

Remover código de criação (aquele com a palavra reservada **new**) de classes de negócio.

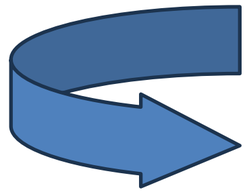
FACTORY

New : instâncias de classes concretas

Quando você tem um conjunto inteiro de classes concretas relacionadas, geralmente é forçado a escrever o código assim:

```
Duck duck;

if (picnic) {
    duck = new
MallardDuck();
} else if (hunting) {
    duck = new
DecoyDuck();
} else if (inBathTub) {
    duck = new
RubberDuck();
}
```

Temos várias classes diferentes de patos e não sabemos até o tempo de execução qual precisa ser instanciado.



- Código com “IFÃO”
- Atualização difícil
- Propenso a erros ao extender

FACTORY

Padrões Factory

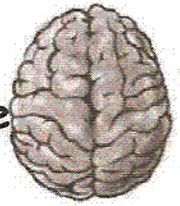
Princípio que é percebido com maior facilidade é o: "*Princípio da inversão de dependência.*“

Que basicamente diz para dependermos de abstração e não de classes concretas.

FACTORY

Muitas classes concretas: código precisa ser alterado caso novas classes concretas sejam adicionadas. Código terá que ser reaberto. (quebra OCP – fechado para modificação e aberto para extensão).

_ Poder da Mente



Como você pode pegar todas as partes de seu aplicativo que instanciam as classes concretas e separá-las ou encapsulá-las do resto do aplicativo?

SIMPLE FACTORY

```
Pizza pizza = new Pizza();

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

E se precisarmos ter vários tipos de pizza?

SIMPLE FACTORY

Este código NÃO está fechado para modificações. Se a pizzaria mudar suas ofertas de pizza, precisamos entrar neste código e modificá-lo.

```
Pizza orderPizza(String type) {
    Pizza pizza;

    if (type.equals("cheese")) {
        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 eggiePizza();
    }

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```

Isto é o que varia. Como a seleção de pizzas muda com o tempo, este código terá que ser modificado muitas vezes.

Isto é o que esperamos que permaneça igual. Para a maioria, preparar, assar e encaixotar uma pizza é a mesma coisa há anos. Então, não esperamos que esse código mude, apenas as pizzas nas quais ele opera.

SIMPLE FACTORY

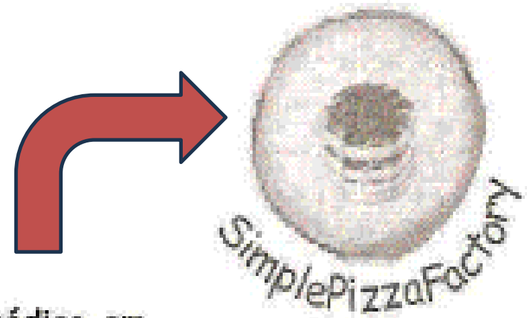
```
if (type.equals("cheese")) {
        pizza = new CheesePizza();
    } else if (type.equals("pepperoni") {
        pizza = new PepperoniPizza();
    } else if (type.equals("clam") {
        pizza = new ClamPizza();
    } else if (type.equals("veggie") {
        pizza = new VeggiePizza();
    }
```

```
Pizza orderPizza(String type) {
    Pizza pizza;

    pizza.prepare();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```

Primeiro, tiramos o código de criação de objetos do Método orderPizza.

O que ficará aqui?

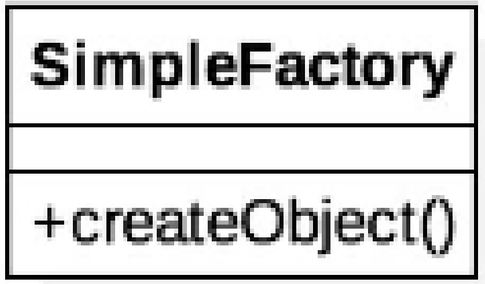
Depois, colocamos esse código em um objeto que só vai se preocupar em como criar pizzas. Se qualquer outro objeto precisar criar uma pizza, deverá recorrer a este objeto.



SIMPLE FACTORY

- Fábricas cuidam dos detalhes da criação de objetos.

- Criar uma classe que encapsule a criação de objetos.



Esta é nossa nova classe, a SimplePizzaFactory. Ela tem uma tarefa na vida: criar pizzas para seus clientes.

Primeiro, definimos um método createPizza() na fábrica. Este é o método que todos os clientes irão usar para instanciar novas objetos.

```
public class SimplePizzaFactory {
    public Pizza createPizza(String type) {
        Pizza pizza = null;
```

SIMPLE FACTORY

```
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;
}
```

Este é o código que tiramos do método orderPizza().

Este código ainda é parametrizado pelo tipo de pizza, assim como nosso método orderPizza() original era.

SIMPLE FACTORY

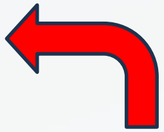
```
public class PizzaFactory {

    public Pizza criarPizza( String tipo ){

        Pizza pizza = null;

        if( tipo.equals( "queijo" ) ){

            pizza = new PizzaQuatroQueijos();
        }
        else if( tipo.equals( "portuguesa" ) ){

            pizza = new PizzaPortuguesa();
        }
        else if( tipo.equals( "calabresa" ) ){

            pizza = new PizzaCalabresa();
        }
        else if( tipo.equals( "camarao" ) ){

            pizza = new PizzaCamarao();
        }

        return pizza;
    }
}
```

```
public class Pizzaria {

    private Pizza pizza;

    public void criarPizza( String tipo ){

        PizzaFactory pizzaFactory = new PizzaFactory();
        pizza = pizzaFactory.criarPizza( tipo );
    }
    ...
}
```

FACTORY METHOD

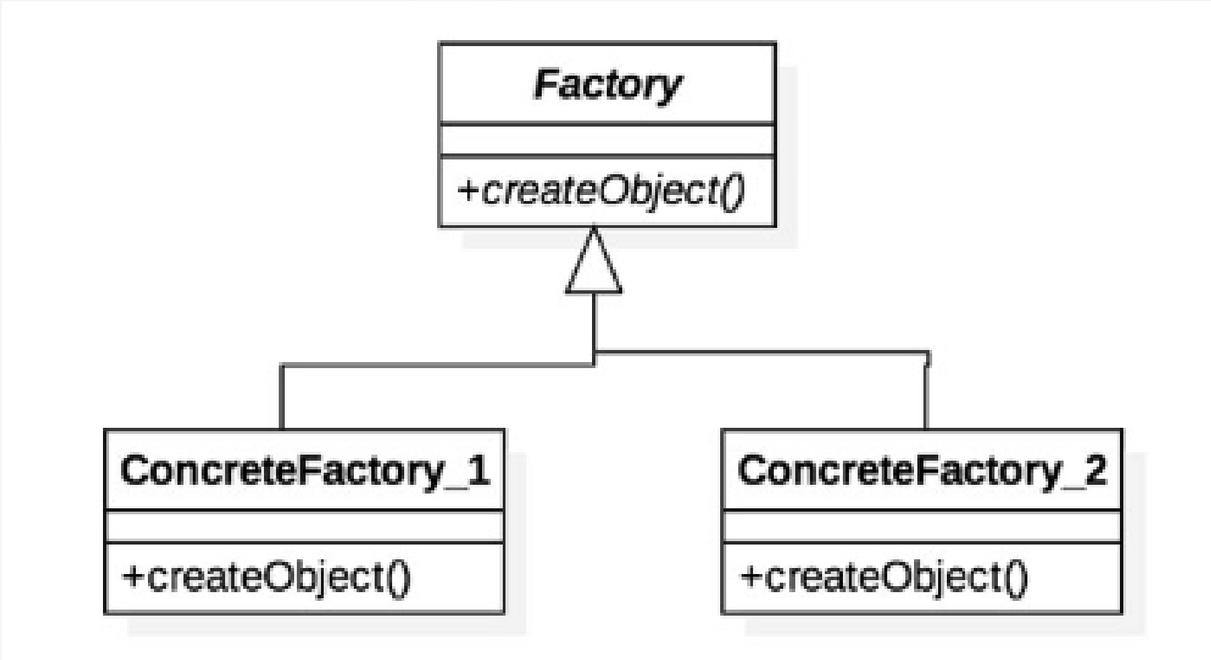
```
public class Pizzaria {

    private Pizza pizza;

    public void criarPizza(
        String cidade,
        String tipo ){

        if( cidade.equals("sao-paulo") ){

            if( tipo.equals( "queijo" ) ){

                pizza = new SPPizzaQuatroQueijos();
            }
            else if( tipo.equals( "portuguesa" ) ){

                pizza = new SPPizzaPortuguesa();
            }
            else if( tipo.equals( "calabresa" ) ){

                pizza = new SPPizzaCalabresa();
            }
        }
        else if( cidade.equals( "rio-de-janeiro" ) ){

            if( tipo.equals( "queijo" ) ){

                pizza = new RJPizzaQuatroQueijos();
            }
            else if( tipo.equals( "portuguesa" ) ){

                pizza = new RJPizzaPortuguesa();
            }
```

E se a mesma pizza tiver preparos diferentes em outras cidades/regiões?



Projeto fortemente acoplado: diminui a eficiência de evolução do software.

FACTORY METHOD



Conjunto de subclasses que assumem a responsabilidade por instanciação.

FACTORY METHOD

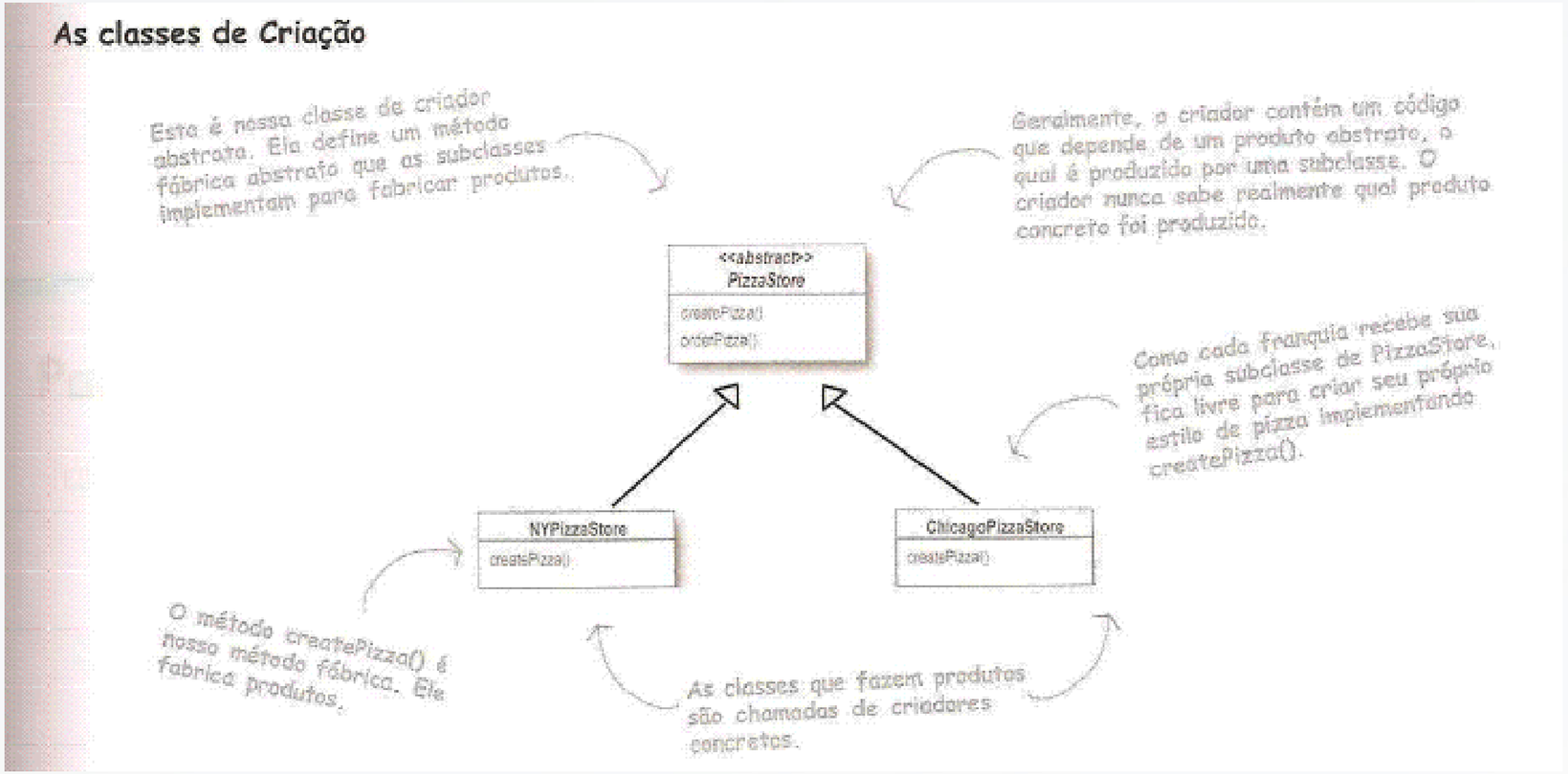
```
public abstract class PizzaFactory {

    protected Pizza pizza;

    public abstract void criarPizza( String tipo );

    public Pizza delivery(){

        return pizza;
    }
}
```

```
public class PizzaFactorySaoPaulo extends PizzaFactory {

    @Override
    public void criarPizza( String tipo ){

        if( tipo.equals( "queijo" ) ){

            pizza = new SPPizzaQuatroQueijos();
        }
        else if( tipo.equals( "portuguesa" ) ){

            pizza = new SPPizzaPortuguesa();
        }
        else if( tipo.equals( "calabresa" ) ){

            pizza = new SPPizzaCalabresa();
        }
    }
}
```

A implementação do método de criação foi delegada para as fábricas concretas.

FACTORY METHOD

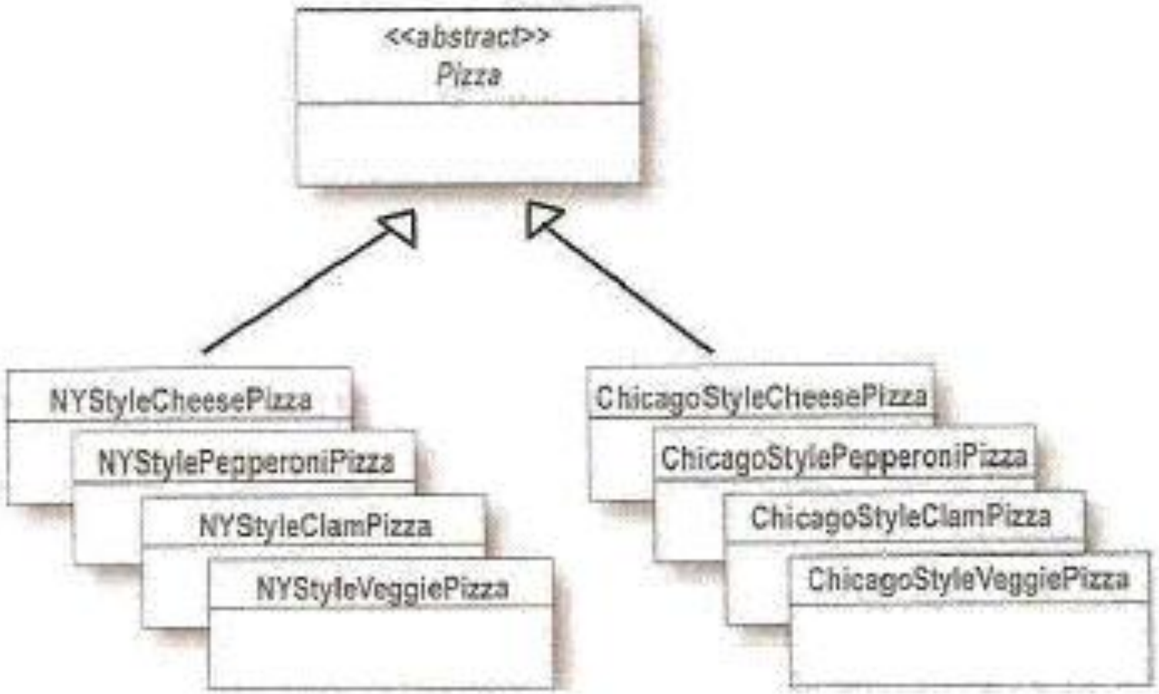
O padrão Factory Method encapsula criação de objetos deixando as subclasses decidirem quais objetos criar.



FACTORY METHOD

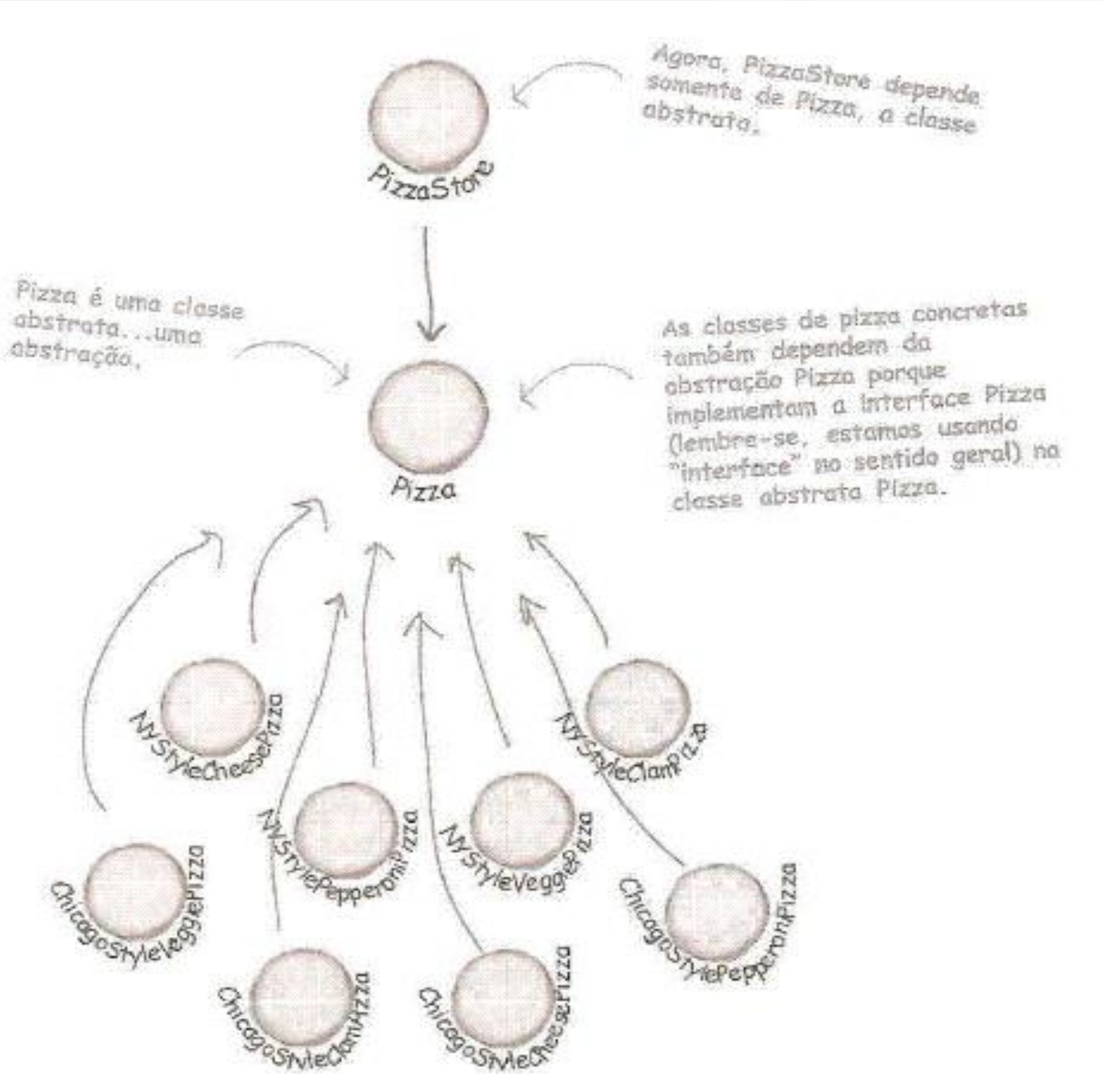
As classes de Produtos

As fábricas fazem produtos e, na pizzaria, nosso produto é a Pizza.



Estes são os produtos concretos - todas as pizzas que são produzidas por nossas lojas.

FACTORY METHOD



Princípio da Inversão de Dependência

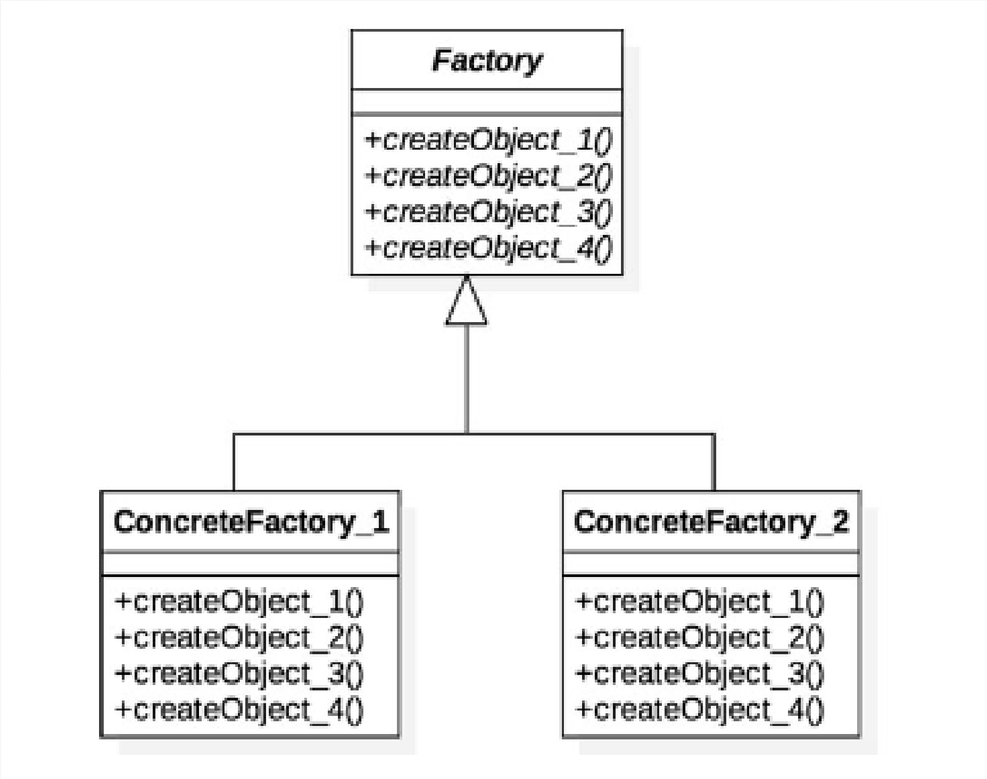
ABSTRACT FACTORY

```
public class PizzaQuatroQueijos extends Pizza {

    private String cidade;
    private Massa massa;
    private Queijo queijo;
    private Molho molho;
    private Salsa salsa;

    public PizzaQuatroQueijos( String cidade ){
        this.cidade = cidade;
    }

    public void prepara(){

        if( cidade.equals( "sao-paulo" ) ){

            massa = new MassaGrossa();
            queijo = new QueijoMinas();
            molho = new MolhoMarroquino();
            salsa = new SalsaNobre();
        }
        else{

            massa = new MassaFina();
            queijo = new QueijoParmesao();
            molho = new MolhoDaRoca();
            salsa = new SalsaNobreApimentada();
        }
    }
    ...
}
```

E se a mesma pizza tiver ingredientes diferentes em outras cidades/regiões, mas o mesmo preparado?

Temos uma família de objetos quando é uma pizza paulista e uma outra família quando é uma carioca.

ABSTRACT FACTORY



Uma família de métodos de criação em Abstract Factory

ABSTRACT FACTORY

Uma família de métodos de criação em Abstract Factory

```
public abstract class PizzaIngredientesFactory {

    public abstract Massa criarMassa();

    public abstract Queijo criarQueijo();

    public abstract Molho criarMolho();

    public abstract Salsa criarSalsa();
}
```

Temos uma família de objetos quando é uma pizza paulista e uma outra família quando é uma carioca

ABSTRACT FACTORY

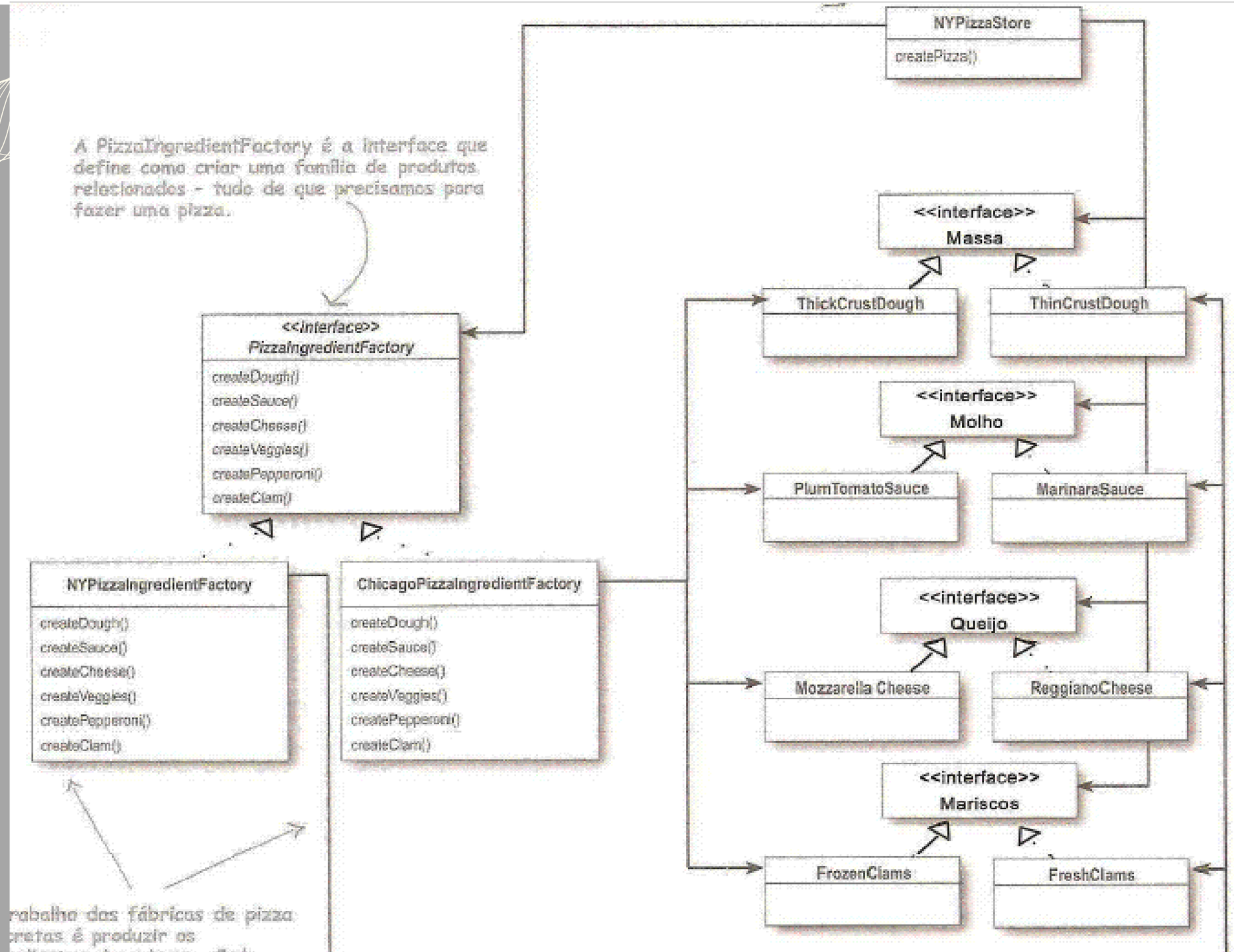
```
public class SPPizzaIngredientesFactory extends PizzaIngredientesFactory {

    @Override
    public Massa criarMassa(){

        return new MassaGrossa();
    }

    @Override
    public Queijo criarQueijo(){

        return new QueijoMinas();
    }

    @Override
    public Molho criarMolho(){

        return new MolhoMarroquino();
    }

    @Override
    public Salsa criarSalsa(){

        return new SalsaNobre();
    }
}
```

Como fábricas concretas teremos uma factory para São Paulo e outra para Rio de Janeiro.

ABSTRACT FACTORY

```
public class PizzaQuatroQueijos extends Pizza {

    private PizzaIngredientesFactory ingredientes;
    private Massa massa;
    private Queijo queijo;
    private Molho molho;
    private Salsa salsa;

    public PizzaQuatroQueijos( PizzaIngredientesFactory ingredientes ){

        this.ingredientes = ingredientes;
    }

    public void prepara(){

        massa = ingredientes.criarMassa();
        queijo = ingredientes.criarQueijo();
        molho = ingredientes.criarMolho();
        salsa = ingredientes.criarSalsa();
    }
}
```

Agora as fábricas já podem ser utilizadas nas subclasses de **Pizza**.


NYPizzaStore
createPizza()
A PizzaIngredientFactory é a interface que define como criar uma família de produtos relacionados - tudo de que precisamos para fazer uma pizza.
<<Interface>>
PizzaIngredientFactory
createDough()
createSauce()
createCheese()
createVeggies()
createPepperoni()
createClam()
<<interface>>
Massa
ThickCrustDough
ThinCrustDough
<<interface>>
Molho
PlumTomatoSauce
MarinaraSauce
<<interface>>
Queijo
Mozzarella Cheese
ReggianoCheese
<<interface>>
Mariscos
FrozenClams
FreshClams
NYPizzaIngredientFactory
createDough()
createSauce()
createCheese()
createVeggies()
createPepperoni()
createClam()
ChicagoPizzaIngredientFactory
createDough()
createSauce()
createCheese()
createVeggies()
createPepperoni()
createClam()
rabalho das fábricas de pizza
cretas é produzir os

ABSTRACT FACTORY

Quando usar fábricas?

Quando precisamos criar objetos que seguem várias regras específicas, resultando em variações semelhantes entre si, recorremos à abstração.

REFERÊNCIAS BIBLIOGRÁFICAS

E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design Patterns - Elements of Reusable Object-Oriented Software. AddisonWesley, 2000.

FREEMAN, E. Use a cabeça! - padrões de projeto (design patterns). 2. ed. Rio de Janeiro: Alta Books, 2007.

https://www.thiengo.com.br/use-a-cabeca-padroes-de-projetos