

TC3003: Diseño y Arquitectura de Software

Dr. Juan Manuel González Calleros

Email: jmgonzale@itesm.mx

Twitter: [@Juan__Gonzalez](https://twitter.com/Juan__Gonzalez)

Facebook: **Juan Glez Calleros**

Reuniones pedir cita



Patrones Creacionales

PATRÓN FABRICA

Patrón Fabrica

- Hasta ahora hemos aprendido a programar interfaces para hacer flexible nuestro código
 - Estrategia
 - Observador
 - Decorador

Patrón Fabrica

- Sin embargo, a final de cuentas para hacer uso de nuestro patrón siempre acabamos haciendo programación en la implementación
 - Constructores new()

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

Patrón Fabrica

- Sin embargo, a final de cuentas para hacer uso de nuestro patrón siempre acabamos haciendo programación en la implementación
 - Constructores new()

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

Problemas de new()

- Ninguno en particular
- Sin embargo, hay mucho que hacer para mantener el código en todos los lugares donde aparece un constructor
 - → Difícil mantenimiento
 - → Origen de errores
 - → Código abierto a modificaciones y sus posibles consecuencias

Problemas de new()

- Ninguno en particular
- Sin embargo, hay mucho que hacer para mantener el código en todos los lugares donde aparece un constructor
 - → Difícil mantenimiento
 - → Origen de errores
 - → Código abierto a modificaciones y sus posibles consecuencias

**RECUERDEN HAY QUE
SEPARAR LAS COSAS QUE
VARÍAN**

Identificar qué varia

- Escenario, una aplicación de una pizzería

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

Identificar qué varia

- Escenario, una aplicación de una pizzería

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

Identificar qué varia

- Escenario, una aplicación de una pizzería

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

Identificar qué varia

- La administración puede cambiar lo que vendemos

```
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();  
    }
```

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

```
}
```



```
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;
```

```
}
```

Problema

- El código no esta cerrado.
- ¿Qué hacemos en este caso?

Problema

- El código no esta cerrado.
- ¿Qué hacemos en este caso?
 - Sacando la creación de objetos fuera del método

Problema

- Sacando la creación de objetos fuera del método

```
Pizza orderPizza(String type) {  
    Pizza pizza;  
  
    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.prepare();  
    pizza.bake();  
    pizza.cut();  
    pizza.box();  
    return pizza;  
}
```

Problema

- Sacando la creación de objetos fuera del método

```
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;  
}
```


Problema

- Creamos un objeto encargado exclusivamente de la creación

```
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;  
}
```



Problema

- Creamos un objeto encargado exclusivamente de la creación (**Fabrica**)

```
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;  
}
```

Fabrica

- Las fabricas se encargan de los detalles de la creación de objetos
 - SimplePizzaFactory para nuestro ejemplo
 - ¿Cómo sería el código?

Fabrica

- Las fabricas SimplePizzaFactory

```
public class SimplePizzaFactory {  
    public Pizza createPizza(String type) {  
        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;  
    }  
}
```

Fabrica


- ¿Ventajas de la Fabrica?

Fabrica

- Ventajas de la Fabrica
 - Podemos reusar la fabrica con otras funciones.
 - Menús del restaurante
 - Ordenes para llevar
 - Ya solo tenemos un lugar donde modificar

Fabrica

- ¿Y cómo queda el código del cliente de a Fabrica?


```
public class PizzaStore {  
    Pizza orderPizza(String type) {  
        Pizza pizza;  
  
          
  
        pizza.prepare();  
        pizza.bake();  
        pizza.cut();  
        pizza.box();  
        return pizza;  
    }  
    // other methods here  
}
```

Fabrica

- El código del cliente

```
public class PizzaStore {  
    SimplePizzaFactory factory;  
  
    public PizzaStore(SimplePizzaFactory factory) {  
        this.factory = factory;  
    }  
  
    public Pizza orderPizza(String type) {  
        Pizza pizza;  
  
        pizza = factory.createPizza(type);  
  
        pizza.prepare();  
        pizza.bake();  
        pizza.cut();  
        pizza.box();  
        return pizza;  
    }  
  
    // other methods here  
}
```

La tienda usa la
Fabrica que le
pasamos como
parámetro a su
constructor



La Fabrica simple

- Esto es más que un patrón una estrategia de programación, muy usada y recomendada.

```
public class SimplePizzaFactory {  
    public Pizza createPizza(String type) {  
        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;  
    }  
}
```

```
public class PizzaStore {  
    SimplePizzaFactory factory;  
  
    public PizzaStore(SimplePizzaFactory factory) {  
        this.factory = factory;  
    }  
  
    public Pizza orderPizza(String type) {  
        Pizza pizza;  
  
        pizza = factory.createPizza(type);  
  
        pizza.prepare();  
        pizza.bake();  
        pizza.cut();  
        pizza.box();  
        return pizza;  
    }  
  
    // other methods here
```

Actividad- Parejas

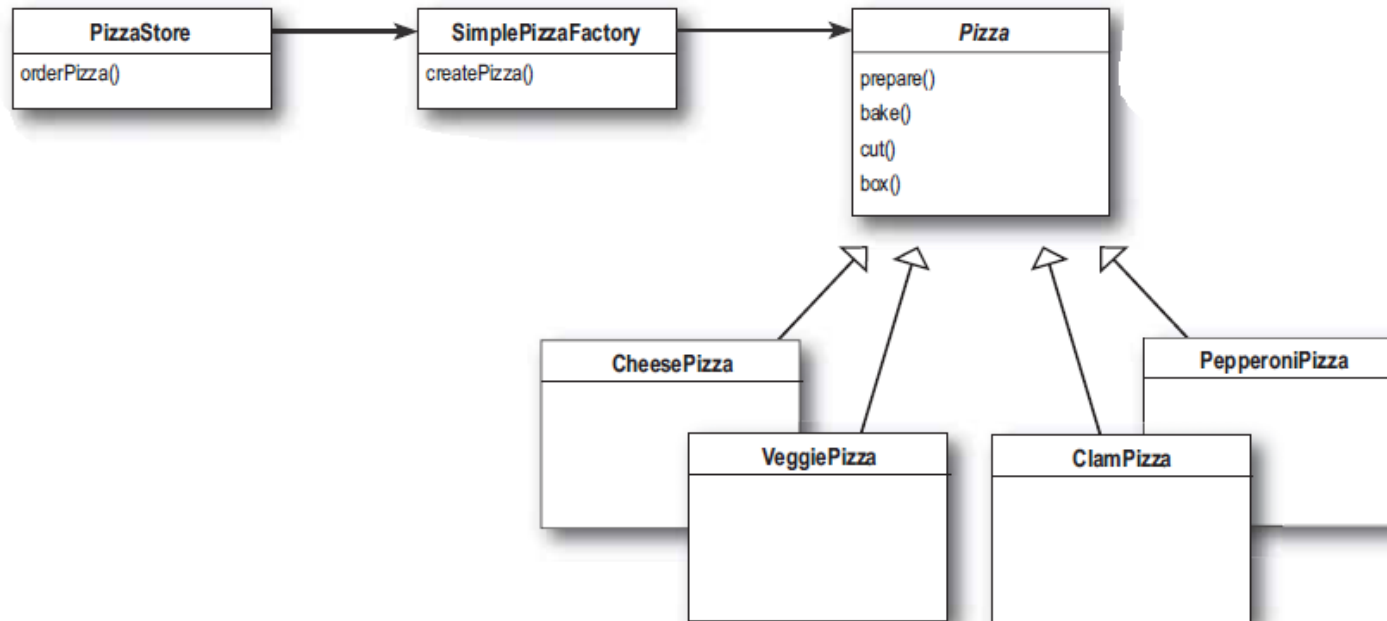
- ¿Cuál es el diagrama de clases de este código?

```
public class SimplePizzaFactory {  
    public Pizza createPizza(String type) {  
        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;  
    }  
}
```

```
public class PizzaStore {  
    SimplePizzaFactory factory;  
  
    public PizzaStore(SimplePizzaFactory factory) {  
        this.factory = factory;  
    }  
  
    public Pizza orderPizza(String type) {  
        Pizza pizza;  
  
        pizza = factory.createPizza(type);  
  
        pizza.prepare();  
        pizza.bake();  
        pizza.cut();  
        pizza.box();  
        return pizza;  
    }  
  
    // other methods here
```

Actividad- Parejas

- Diagrama de clases de la Pizzería

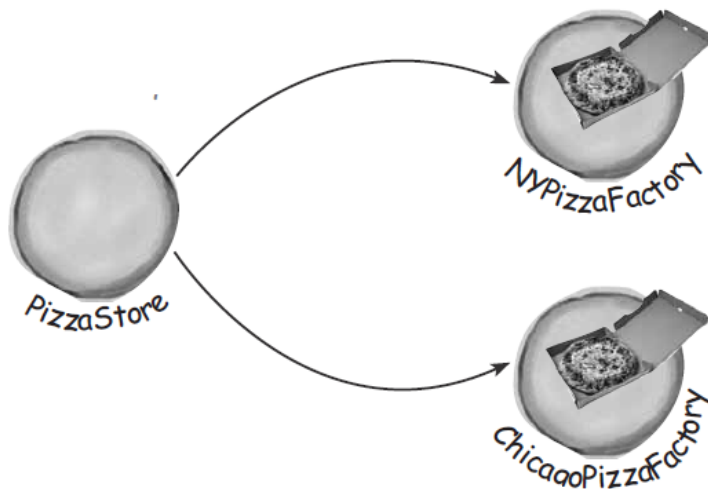


El negocio crece

FABRICA DE CONSTRUCCIÓN PARA MÚLTIPLES TIENDAS

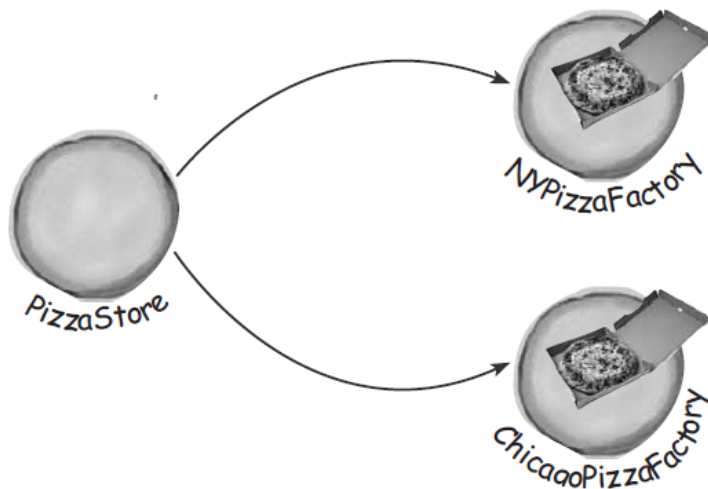
Fabrica

- El negocio crece y queremos que muchas tiendas adopten nuestro sistema pero considerando diferencias regionales.
 - Necesito fabricas de pizzas para cada tienda



Fabrica

- El negocio crece y queremos que muchas tiendas adopten nuestro sistema pero considerando diferencias regionales.
 - Necesito fabricas de pizzas para cada tienda



Fabrica

- Necesito fabricas de pizzas para cada tienda que respete sabores diferentes incluso para las mismas pizzas

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

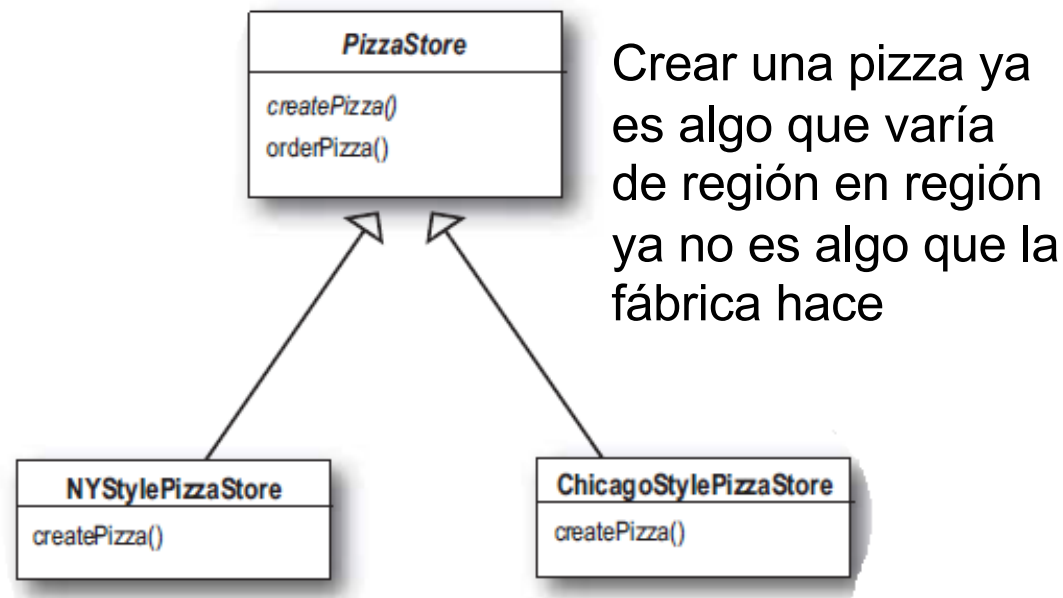
```
ChicagoPizzaFactory chicagoFactory = new ChicagoPizzaFactory();  
PizzaStore chicagoStore = new PizzaStore(chicagoFactory);  
chicagoStore.order("Veggie");
```

Fabrica

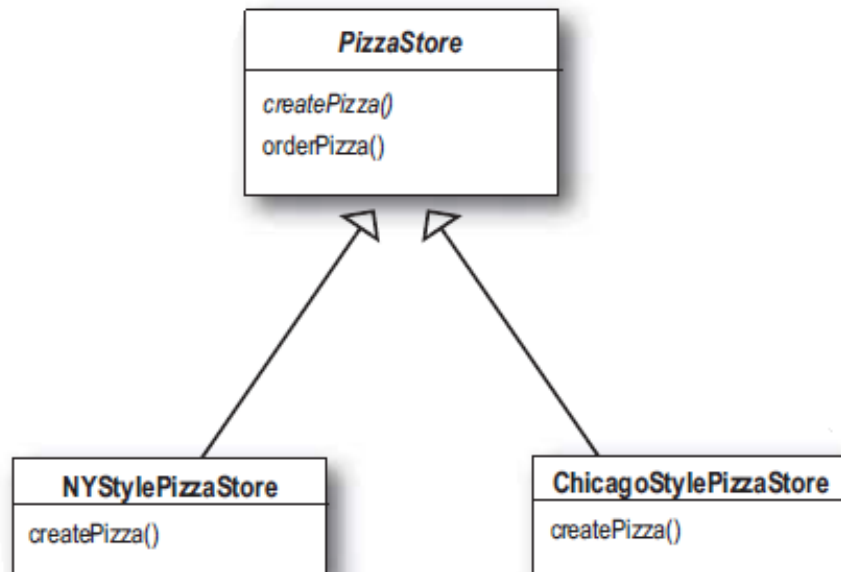
- La realidad es que manteniendo las diferencias regionales tenemos que asegurarnos que, la pizza se cocina y prepare siguiendo tu estándar.
 - Necesitamos un sistema que centralice la creación de pizza y sea flexible para las franquicias.

Fabrica

- Cada franquicia será una implementación de la tienda, por lo que la tienda se convierte en una clase abstracta



Fabrica



```
public abstract class PizzaStore {

    public Pizza orderPizza(String type) {
        Pizza pizza;

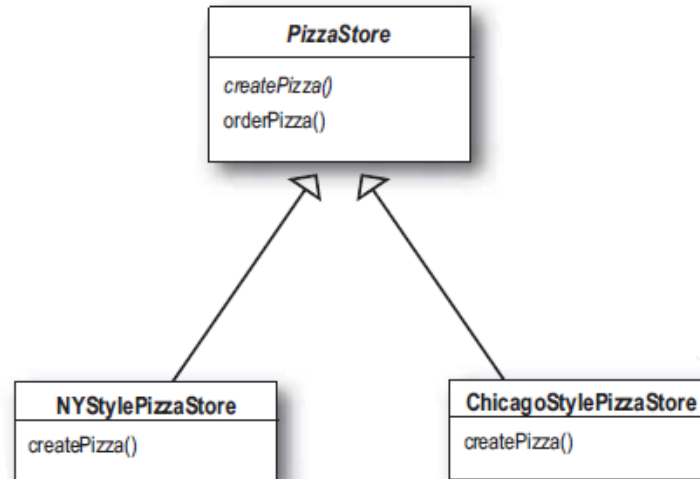
        pizza = createPizza(type);

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

        return pizza;
    }

    abstract createPizza(String type);
}
```

Fabrica



```
public abstract class PizzaStore {

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = createPizza(type);

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

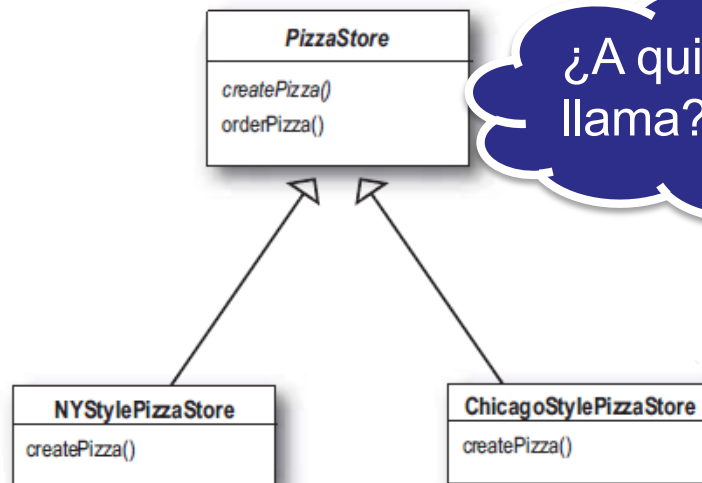
        return pizza;
    }

    abstract createPizza(String type);
}
```

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

```
public Pizza createPizza(type) {
    if (type.equals("cheese")) {
        pizza = new ChicagoStyleCheesePizza();
    } else if (type.equals("pepperoni")) {
        pizza = new ChicagoStylePepperoniPizza();
    } else if (type.equals("clam")) {
        pizza = new ChicagoStyleClamPizza();
    } else if (type.equals("veggie")) {
        pizza = new ChicagoStyleVeggiePizza();
    }
}
```

Fab



¿A quién llama?

```
public abstract class PizzaStore {

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = createPizza(type);

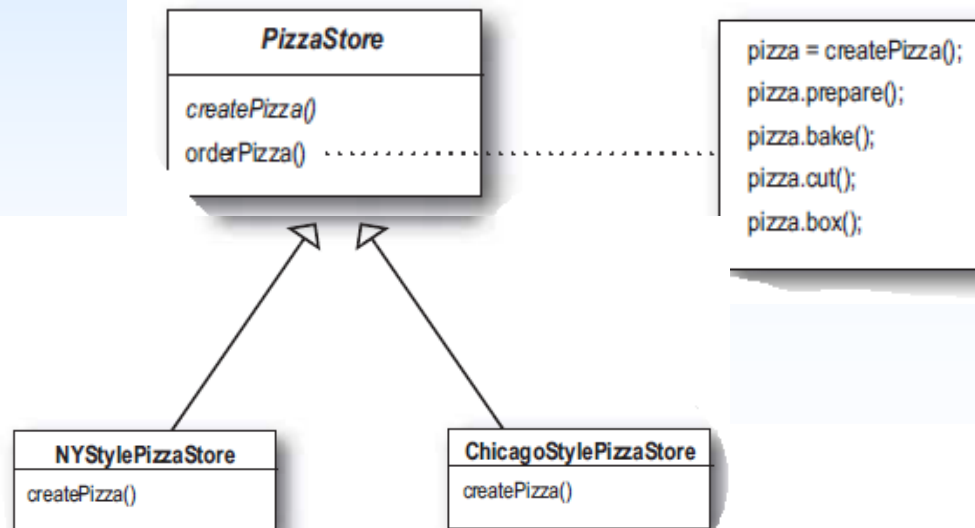
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();

        return pizza;
    }

    abstract createPizza(String type);
}
```

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

```
public Pizza createPizza(type) {
    if (type.equals("cheese")) {
        pizza = new ChicagoStyleCheesePizza();
    } else if (type.equals("pepperoni")) {
        pizza = new ChicagoStylePepperoniPizza();
    } else if (type.equals("clam")) {
        pizza = new ChicagoStyleClamPizza();
    } else if (type.equals("veggie")) {
        pizza = new ChicagoStyleVeggiePizza();
    }
}
```



```

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

```

public abstract class PizzaStore {

    public Pizza orderPizza(String type) {
        Pizza pizza;

        pizza = createPizza(type);

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

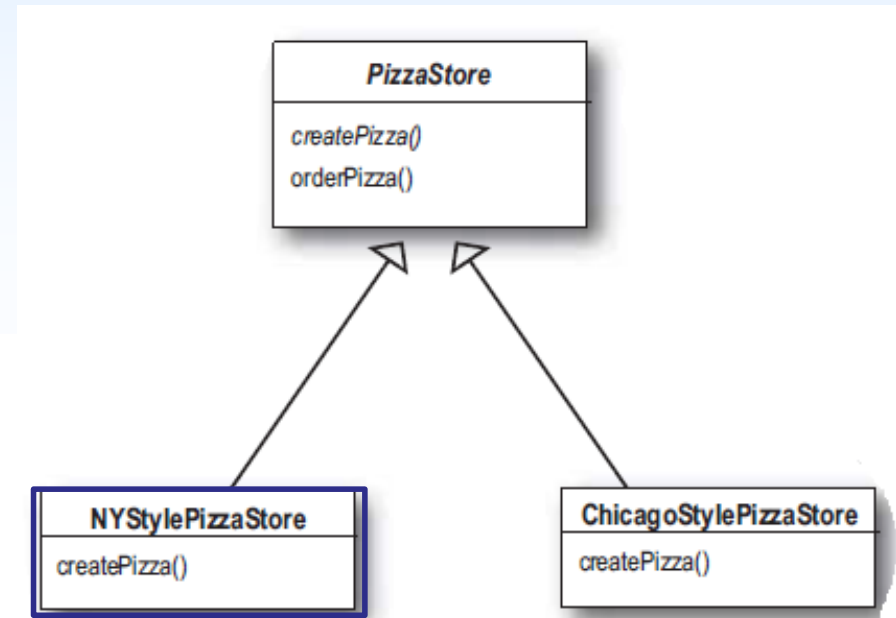
        return pizza;
    }

    abstract createPizza(String type);
}
  
```

- orderPizza que es común a todos va a usar el método de la ciudad y nos asegura que cocinará todo siguiendo nuestro estándar.

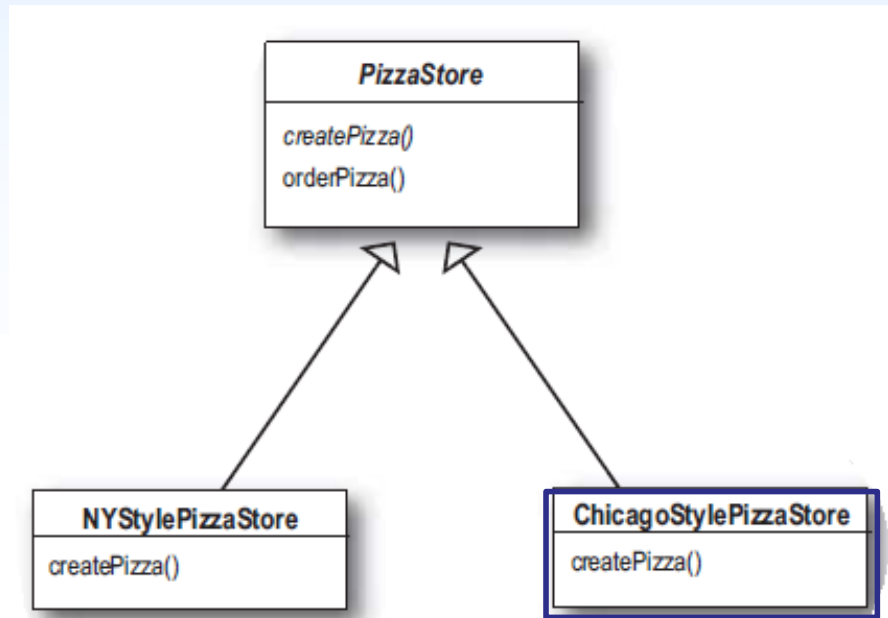
Vamos a hacer el código de las franquicias – Equipos

Vamos a hacer el código de las franquicias



```
public class NYPizzaStore extends PizzaStore {
    Pizza createPizza(String item) {
        if (item.equals("cheese")) {
            return new NYStyleCheesePizza();
        } else if (item.equals("veggie")) {
            return new NYStyleVeggiePizza();
        } else if (item.equals("clam")) {
            return new NYStyleClamPizza();
        } else if (item.equals("pepperoni")) {
            return new NYStylePepperoniPizza();
        } else return null;
    }
}
```

Vamos a hacer el código de las franquicias



```
public class ChicagoPizzaStore extends PizzaStore {

    Pizza createPizza(String item) {
        if (item.equals("cheese")) {
            return new ChicagoStyleCheesePizza();
        } else if (item.equals("veggie")) {
            return new ChicagoStyleVeggiePizza();
        } else if (item.equals("clam")) {
            return new ChicagoStyleClamPizza();
        } else if (item.equals("pepperoni")) {
            return new ChicagoStylePepperoniPizza();
        } else return null;
    }
}
```


El método fábrica

- El método actúa como fábrica
 - Encapsula su funcionalidad en las subclases
 - La única forma de crear instancias de algún producto
 - Otros métodos lo usan

```
public Pizza orderPizza(String type) {  
    Pizza pizza;  
  
    pizza = createPizza(type);  
  
    pizza.prepare();  
    pizza.bake();  
    pizza.cut();  
    pizza.box();  
  
    return pizza;  
}  
  
protected abstract Pizza createPizza(String type);  
  
// other methods here  
}
```

El método fábrica – En uso

- ¿Cómo van a ordenar?



El método fábrica – En uso

- ¿Cómo van a ordenar?
 1. Necesitan instancias de la tienda, NYC y Chicago

```
PizzaStore nyPizzaStore = new NYPizzaStore();
```



El método fábrica – En uso

- ¿Cómo van a ordenar?
 1. Necesitan instancias de la tienda, NYC y Chicago
 2. Ya con las instancia de tienda cada quien invoca a orderPizza(str)



```
nyPizzaStore.orderPizza("cheese");
```

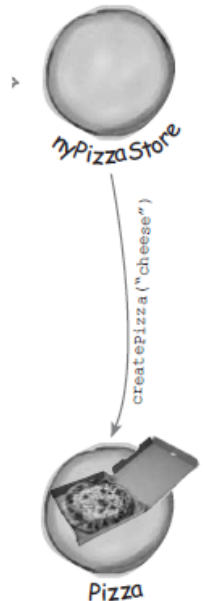
El método fábrica – En uso

- ¿Cómo van a ordenar?
 1. Necesitan instancias de la tienda, NYC y Chicago
 2. Ya con las instancia de tienda cada quien invoca a orderPizza(str)
 3. Se invoca el metodo createPizza()
 - Método Fábrica
 - Llama al método de la subclase que lo llamo

```
PizzaStore nyPizzaStore = new NYPizzaStore();
```

```
nyPizzaStore.orderPizza("cheese");  
... Order() ...
```

```
Pizza pizza = createPizza("cheese");
```



Necesitamos ahora Pizza

- Necesitamos definir algunos tipos de Pizza
- Y sus métodos
 - prepare()
 - bake()
 - cut()
 - box()
 - getName()
- Y sus atributos
 - name : String
 - Dough : String
 - Sauce : String
 - toppings : List

Necesitamos ahora Pizza

```
public abstract class Pizza {
    String name;
    String dough;
    String sauce;
    ArrayList toppings = new ArrayList();

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

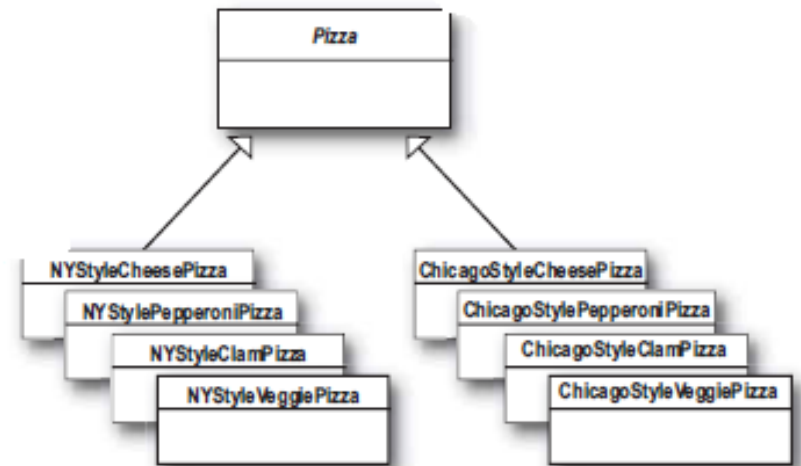
    void cut() {
        System.out.println("Cutting the pizza into diagonal slices");
    }

    void box() {
        System.out.println("Place pizza in official PizzaStore box");
    }

    public String getName() {
        return name;
    }
}
```

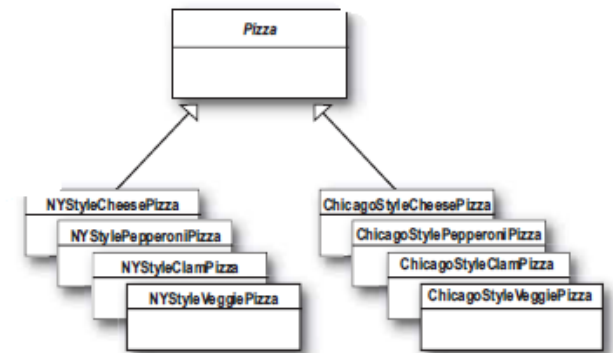
Necesitamos ahora variedades

- NYStyleCheesePizza
 - Name= “NY Style Sauce and Cheese Pizza”
 - Dough=“Thin crust dough”
 - Sauce=“Marinara sauce”
 - Toppings.add(“Grated Reggiano Cheese”)



Necesitamos ahora variedades

- ChicagoStyleCheesePizza
 - Name= “Chicago Style Deep Dish Cheese Pizza”
 - Dough=“Extra thick crust dough”
 - Sauce=“Plum Tomato sauce”
 - Toppings.add(“Shredded Mozzarella Cheese”)
 - Cut method to say
 - “cutting pizza into square slices”



Despliega esta salida

```
File Edit Window Help YouWantMooitzOnThatPizza?
%java PizzaTestDrive

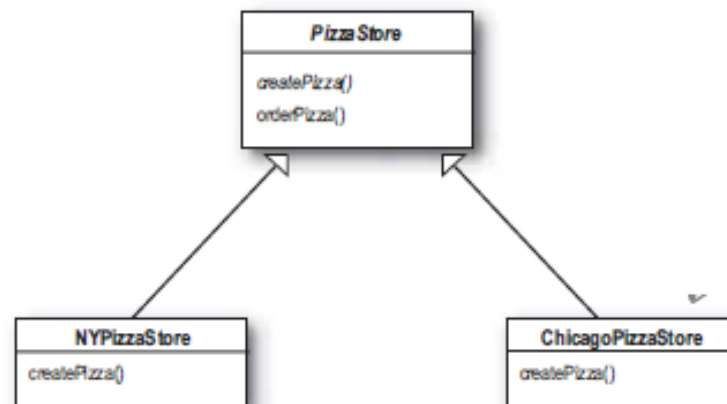
Preparing NY Style Sauce and Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
    Grated Regiano cheese
Bake for 25 minutes at 350
Cutting the pizza into diagonal slices
Place pizza in official PizzaStore box
Ethan ordered a NY Style Sauce and Cheese Pizza

Preparing Chicago Style Deep Dish Cheese Pizza
Tossing dough...
Adding sauce...
Adding toppings:
    Shredded Mozzarella Cheese
Bake for 25 minutes at 350
Cutting the pizza into square slices
Place pizza in official PizzaStore box
Joel ordered a Chicago Style Deep Dish Cheese Pizza
```

PATRÓN MÉTODO DE FÁBRICA

Patrón Método de Fábrica

- Todo patrón de Fábrica encapsula la creación de objetos
- Las subclases deciden que objetos van a crear
- Conocida como las clases creadoras



```

public abstract class PizzaStore {

    abstract Pizza createPizza(String item);

    public Pizza orderPizza(String type) {
        Pizza pizza = createPizza(type);
        System.out.println("--- Making a " + pizza.getName() + " ---");
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
        return pizza;
    }
}

```

```

public class NYPizzaStore extends PizzaStore {

    Pizza createPizza(String item) {
        if (item.equals("cheese")) {
            return new NYStyleCheesePizza();
        } else if (item.equals("veggie")) {
            return new NYStyleVeggiePizza();
        } else if (item.equals("clam")) {
            return new NYStyleClamPizza();
        } else if (item.equals("pepperoni")) {
            return new NYStylePepperoniPizza();
        } else return null;
    }
}

```

```

public class ChicagoPizzaStore extends PizzaStore {

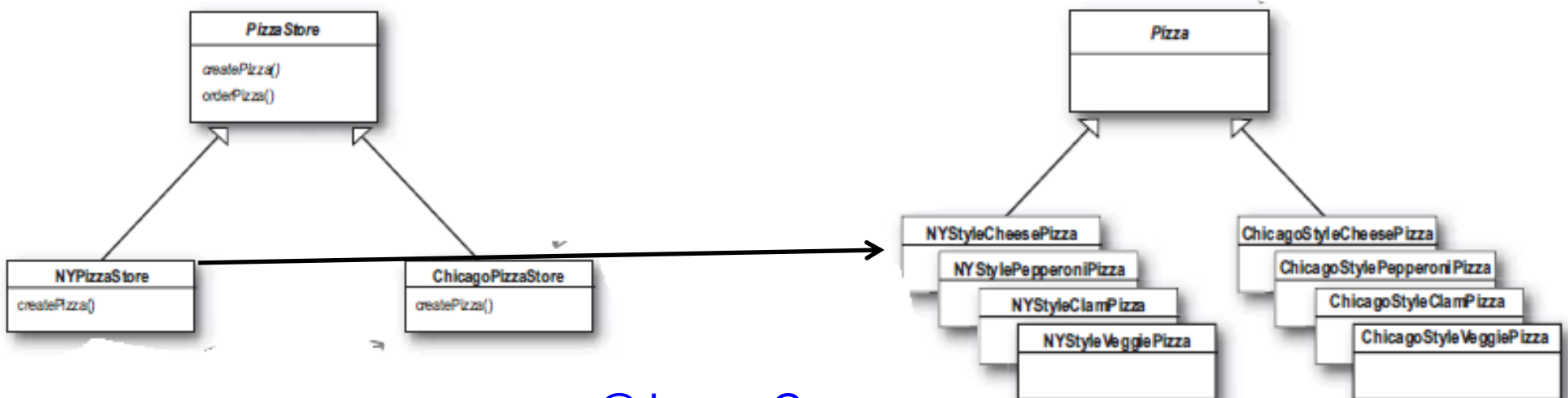
    Pizza createPizza(String item) {
        if (item.equals("cheese")) {
            return new ChicagoStyleCheesePizza();
        } else if (item.equals("veggie")) {
            return new ChicagoStyleVeggiePizza();
        } else if (item.equals("clam")) {
            return new ChicagoStyleClamPizza();
        } else if (item.equals("pepperoni")) {
            return new ChicagoStylePepperoniPizza();
        } else return null;
    }
}

```

@Juan_Gonzalez

Patrón Método de Fábrica

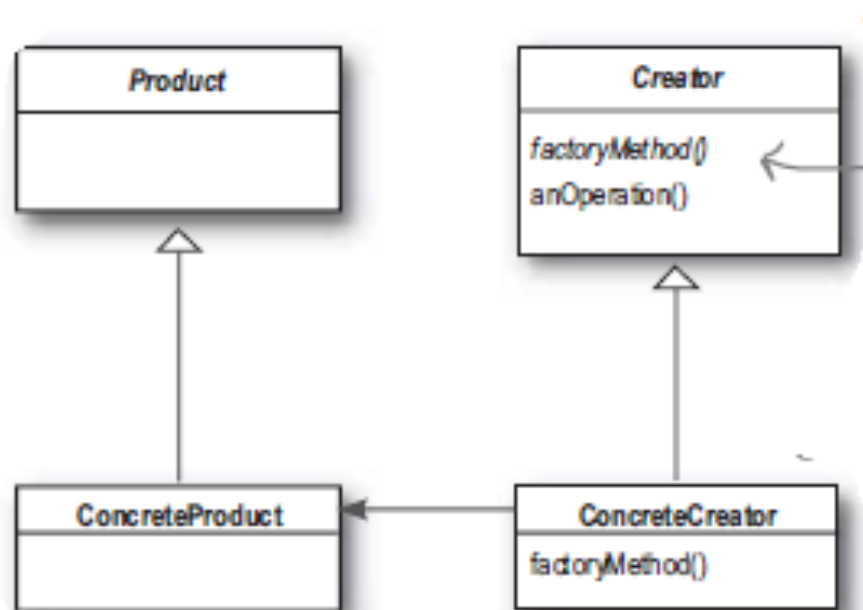
- Esta fabricas producen productos
 - Pizzas en nuestro caso



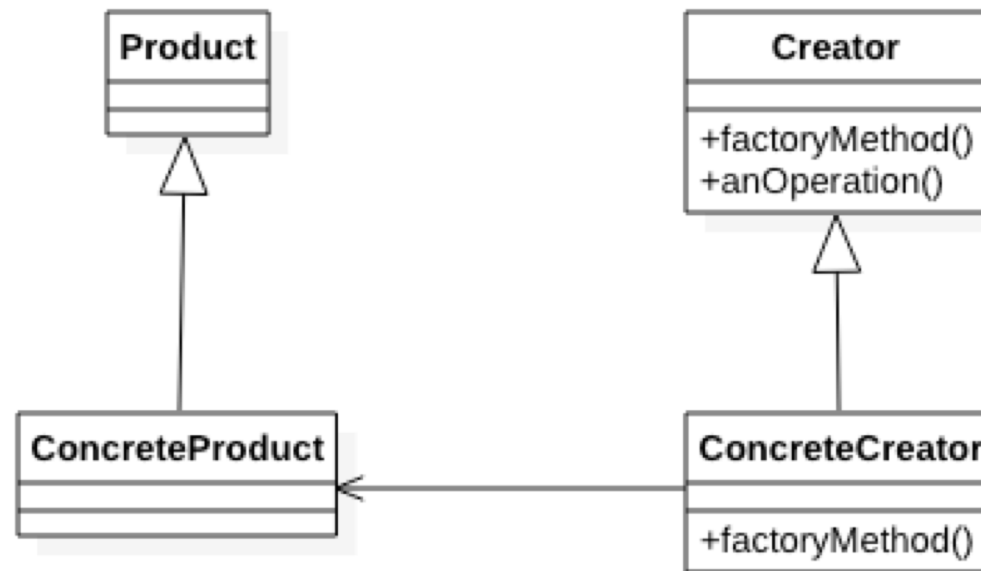
Patrón Método de Fábrica

- ¿Cómo lo defines?

Patrón Método de Fábrica



Patrón Método de Fábrica



Tarea Próximo Lunes

- Identificar los patrones de creación necesarios para tu proyecto