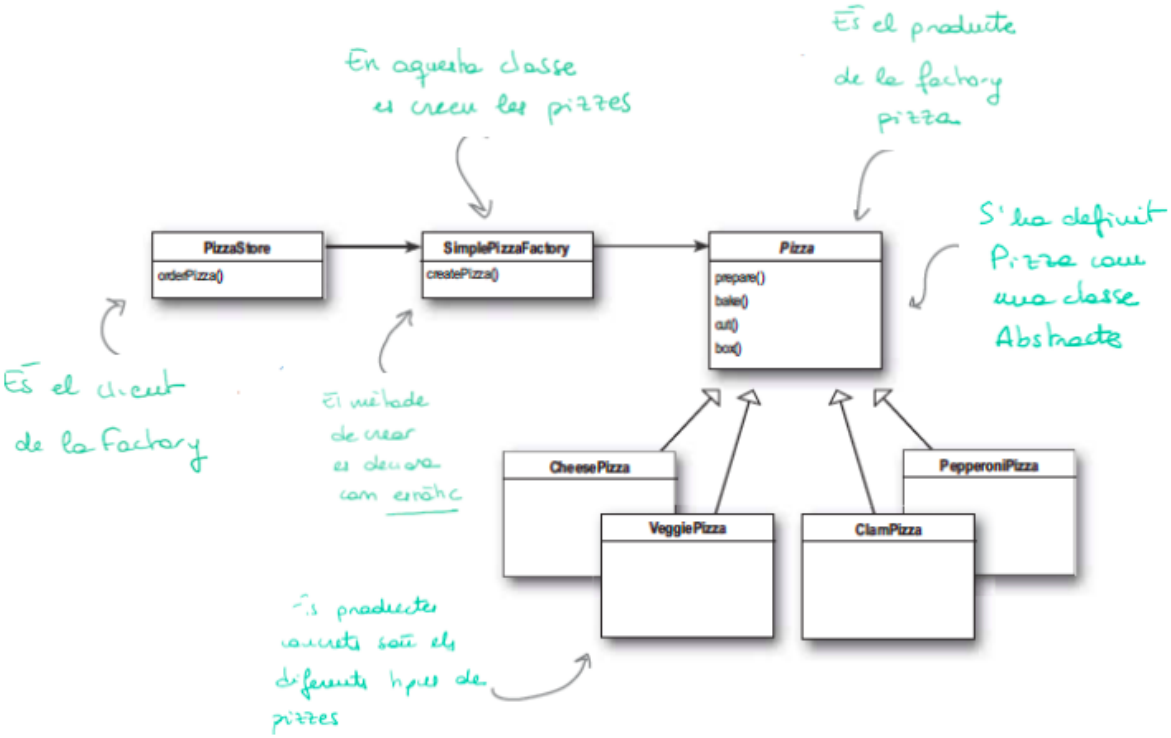
Exercicis de Patrons: Factory (Factory method i Abstract Factory)

Es vol modelar una pizzeria que ofereixen pizzes de formatge, de peperoni i vegetals. Cada pizza es crea, segons el seu tipus, es prepara, es cou, es talla, es posa en una caixa i es lliura la pizza. Un dissenyador novell ha fet el projecte PizzaStore (https://campusvirtual.ub.edu/mod/resource/view.php?id=1558021) ha dissenyat una classe Pizza de la qual hereten les classes les diferents classes de pizzes, Totes elles comparteixen els mètodes que estan implementats en la classe pare, però tenen atributs diferents.

a) **PROJECTE INICIAL:** Quin principi S.O.L.I.D. vulnera aquest codi? Mira el mètode orderPizza() de la classe Pizza. Si hi hagués alguna classe més que volgués crear pizza, seria reutilitzable? Com solucionaries aquest problema? Quin patró de Factories usaries? Fes una carpeta nova anomenada **Solucio1 i es el refactor del projecte inicial.**

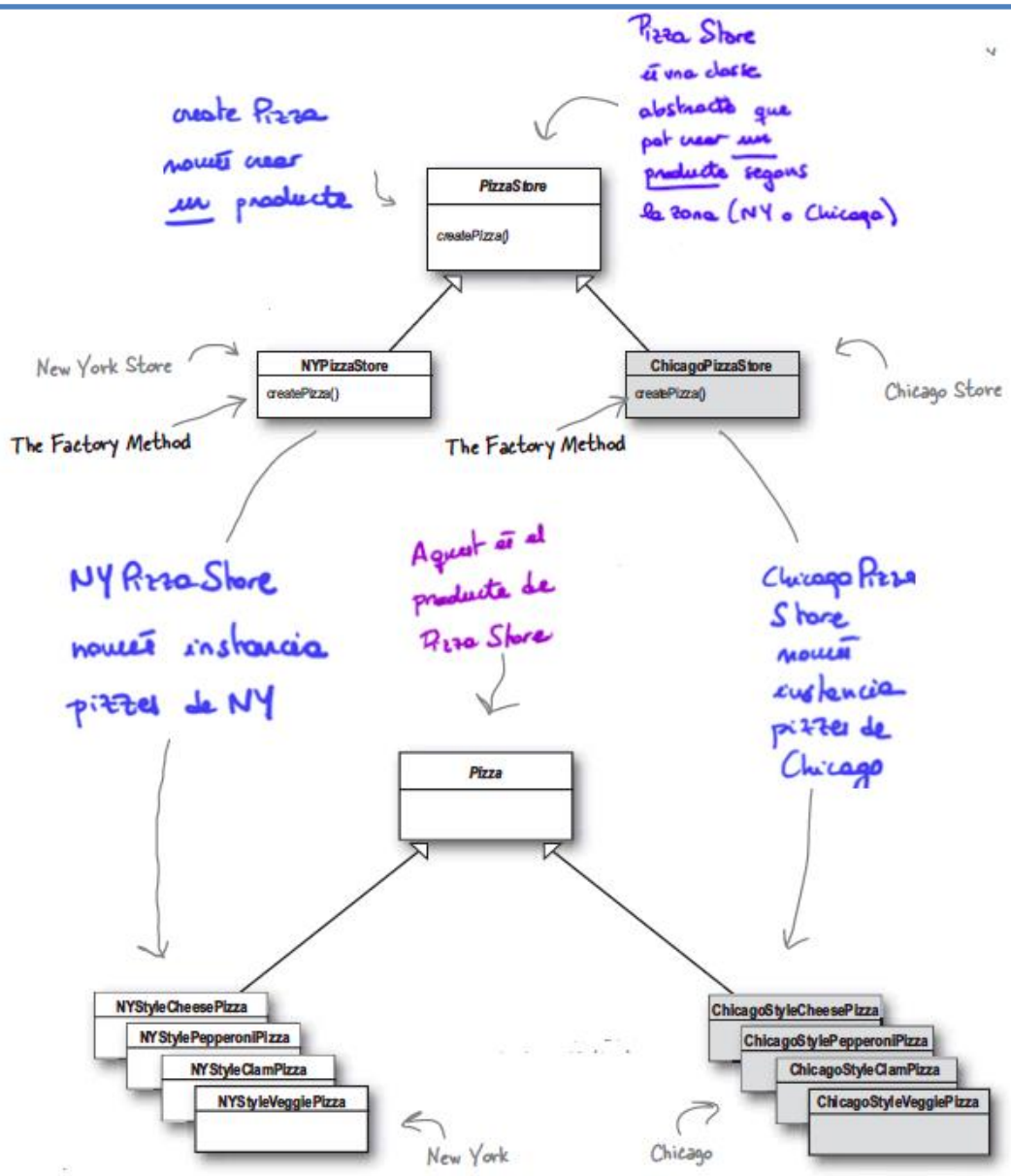
Es vulnera el principi Single Responsability en el mètode orderPizza, també es vulnera el Open-Closed Principle. A part, no s'aplica bé el patró Expert ja que el PizzaStore no té per què saber com es fan les pizzes. Una primera solució és aplicar el patró Template Factory (1a proximació).

Analitza la solució proposada en el paquet pizzas del projecte IntelliJ factorySolucions:



b) **PROJECTE FRANQUÍCIES**: En el cas que es volguessin fer vàries franquícies de la Pizzeria, on totes les franquícies tenen el mateix tipus de pizzes però cuinades de formes diferents, com s'hauria de modificar el codi? Imagina que tens franquícies a New York i a Chicago. Quins principis S.O.L.I.D. estaries vulnerant? **Fes una carpeta nova anomenada Solucio2 aplicant el patró de Factories adient per a aquest projecte de franquícies**.

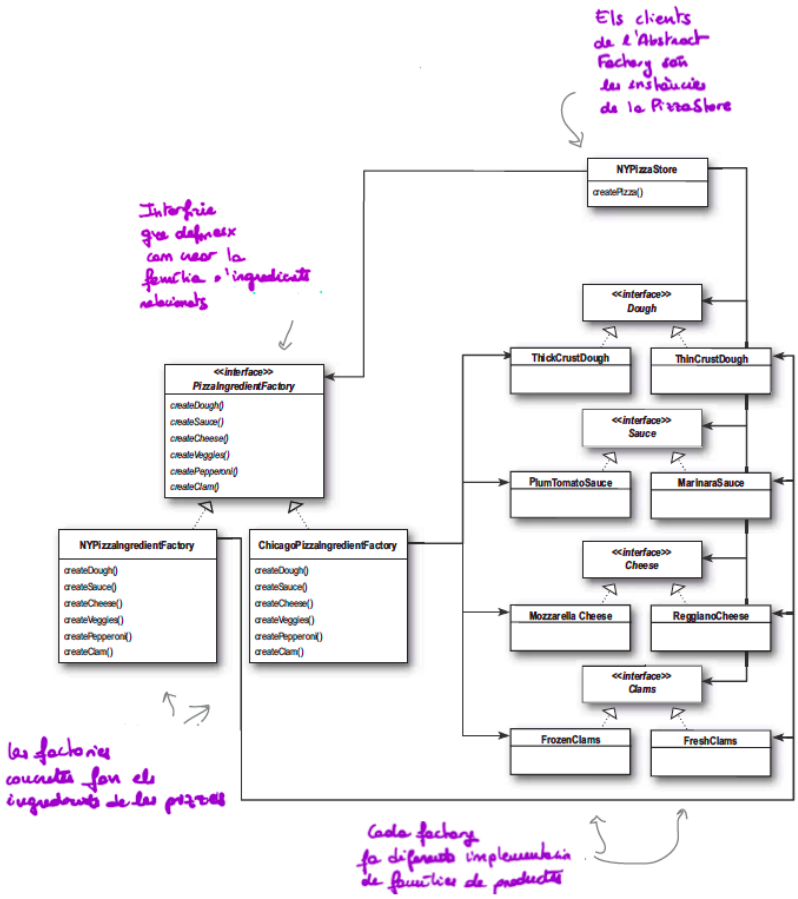
Bàsicament es vulnera el principi Dependency Inversion, ja que la classe Factory Dependent depèn de les franquícies concretes. A més a més es vulnera el principi Open-Closed, ja que afegir una nova franquícia obligarà a canviar el codi del mètode createPizza. En aplicar el patró FactoryMethod (mira el projecte pizzafm del projecte factorySolucions), queda el següent diagrama de classes:



c) **PROJECTE INGREDIENTS:** En el cas que es volguessin afegir ingredients a les diferents pizzes, que depenen també del tipus de pizzes i de la franquícia, quin problema es tindria en crear les classes? Cada ingredient de la pizza (massa, salsa, formatge, etc.) es fa de forma diferent segons siguin pizzes de NY o de Chicago. S'ha utilitzat el patró d'**Abstract Factory** per a solucionar aquest problema. Mira la solució plantejada al projecte projecteIngredients del campus i intenta fer el seu diagrama de classes. Mira com es correspon al patró **Abstract Factory** de la teoria. Quines classes equivalen al Client, AbstractFactory, ConcreteFactory, Product i ConcreteProduct?

Caldria fer que la creació d'ingredients i la seva utilització variï segons si és la franquícia de NY o de Chicago. La solució proposada segueix el següent diagrama de

classes. Fixa't que ara la classe NYPizzaStore és la classe client del Abstract Factory. L'interfície Abstract Factory és la classe PizzaIngredientsFactory.




```java
public class PepperoniPizza extends Pizza {
    PizzaIngredientFactory ingredientFactory;

    public PepperoniPizza(PizzaIngredientFactory ingredientFactory) {
        this.ingredientFactory = ingredientFactory;
    }

    void prepare() {
        System.out.println("Preparing " + name);
        dough = ingredientFactory.createDough();
        sauce = ingredientFactory.createSauce();
        cheese = ingredientFactory.createCheese();
        veggies = ingredientFactory.createVeggies();
        pepperoni = ingredientFactory.createPepperoni();
    }
}
```