Factory Pattern

Lets take example of pizza

```
orderPizza() {
    Pizza pizza = new Pizza();
    pizza.bake();
    pizza.cut();
    pizza.box();
    return pizza;
}
```

But we need multiple types of Pizza

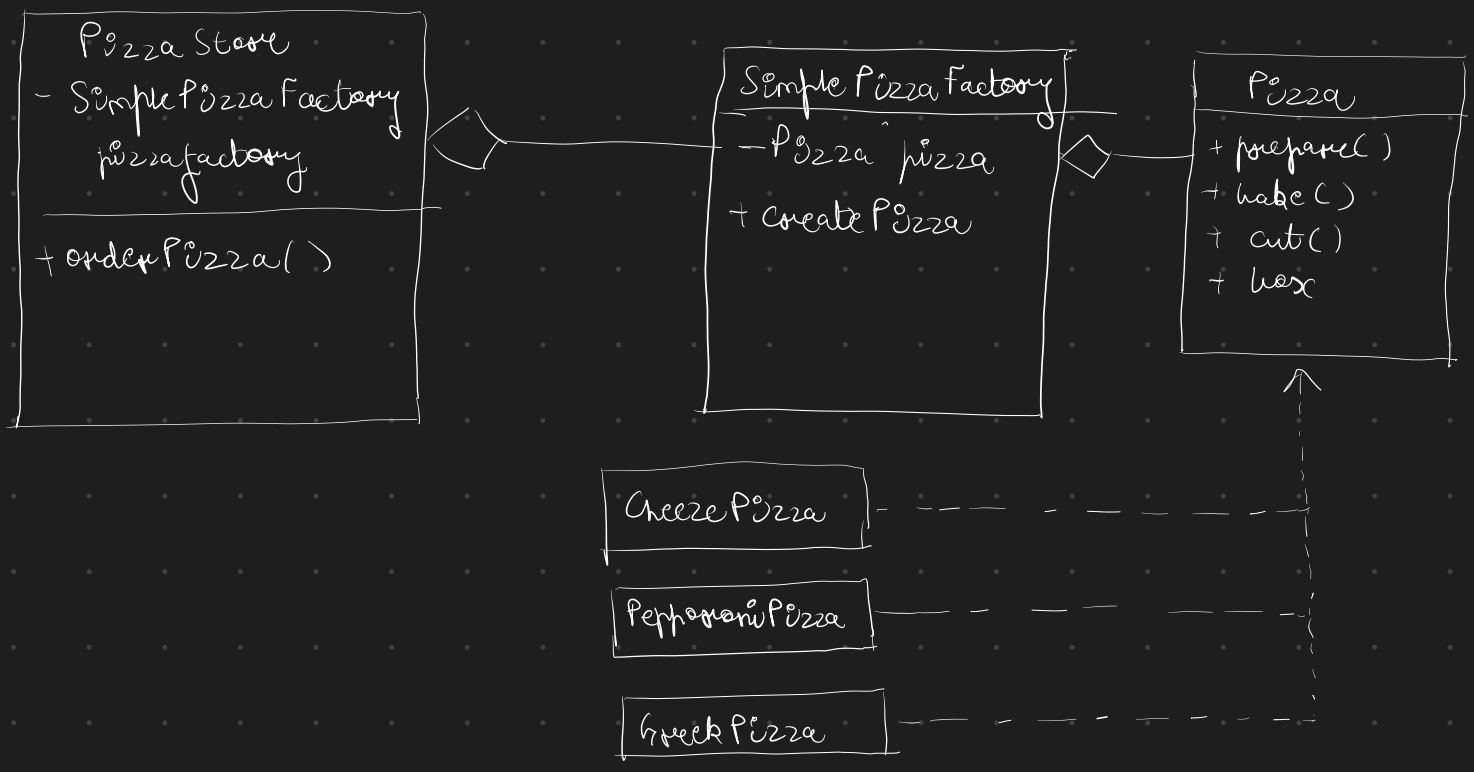
CheezePizza();
GreekPizza();
PepperoniPizza();

} This can vary with time

Q: How do we solve the problem of creating a new pizza without needing to add to orderPizza() method every time?

A: SHIFT the creation logic to another object known as SimplePizzaFactory

Current Diagram for our factory



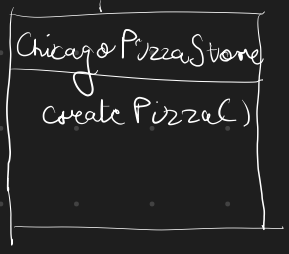
Improving Above Factory Design

Franchising

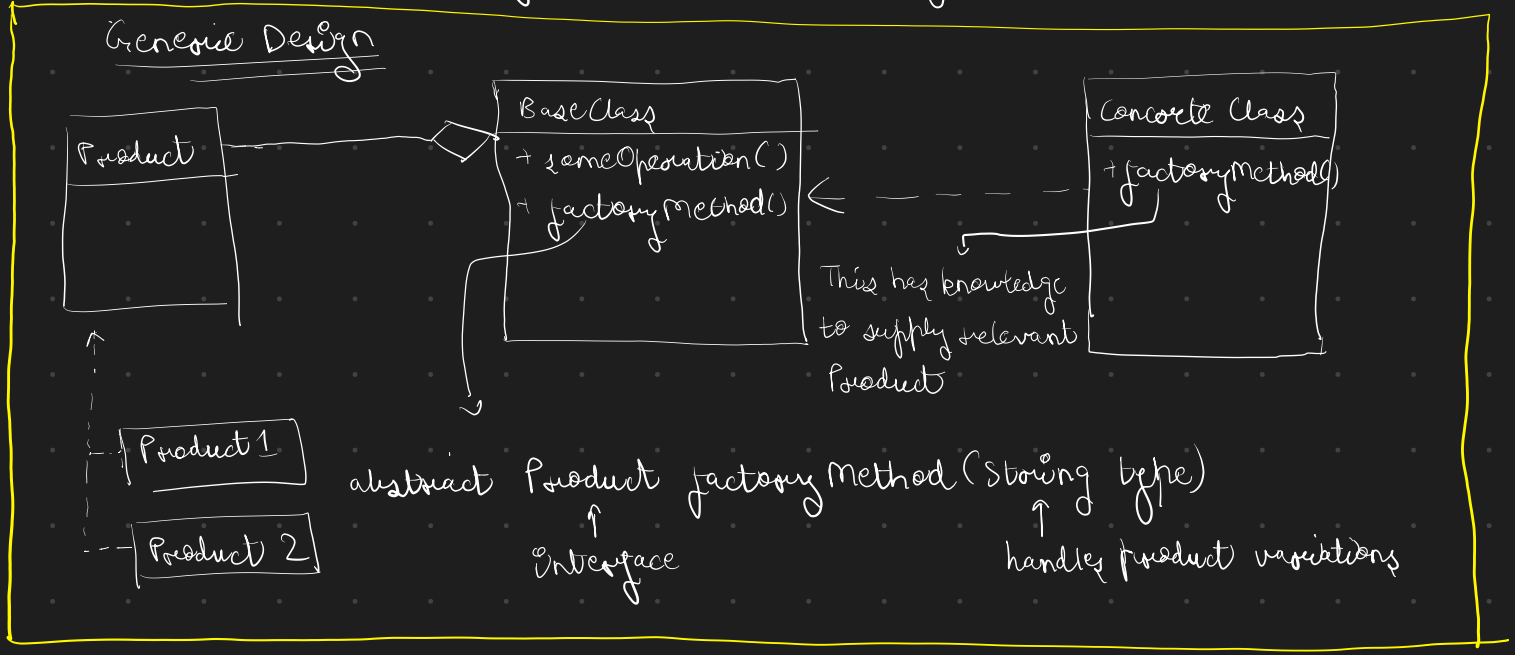
Lets suppose we want different regional pizzas. How do we achieve that?

A: We make createPizza method abstract in PizzaStore method.

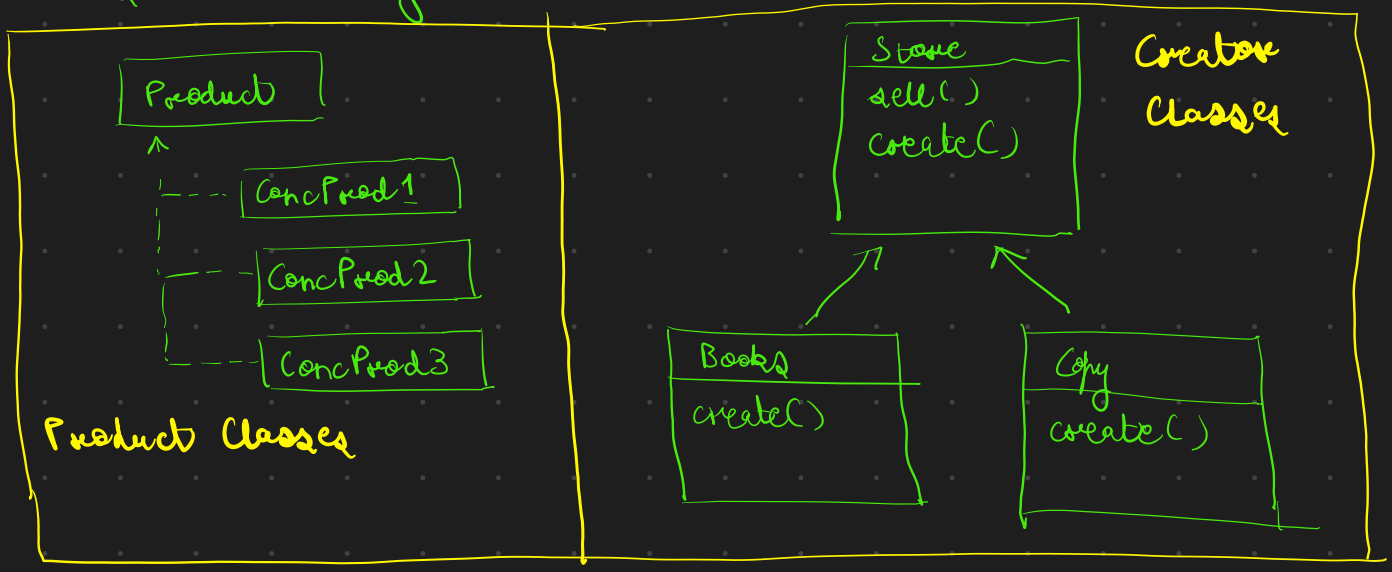
Design Now



With this design, we have decoupled createPizza() logic from the orderPizza() method.
orderPizza() doesn't need to know which pizza is being created. That logic is governed by the one implementing that class.



This pattern can also be seen as something which encapsulates Product and its creation logic.



Learning Dependency Inversion

Design Principle: (Dependency Inversion)

Depend on abstractions, don't depend on concrete classes.

Q: what does this design principle mean?

→ It means

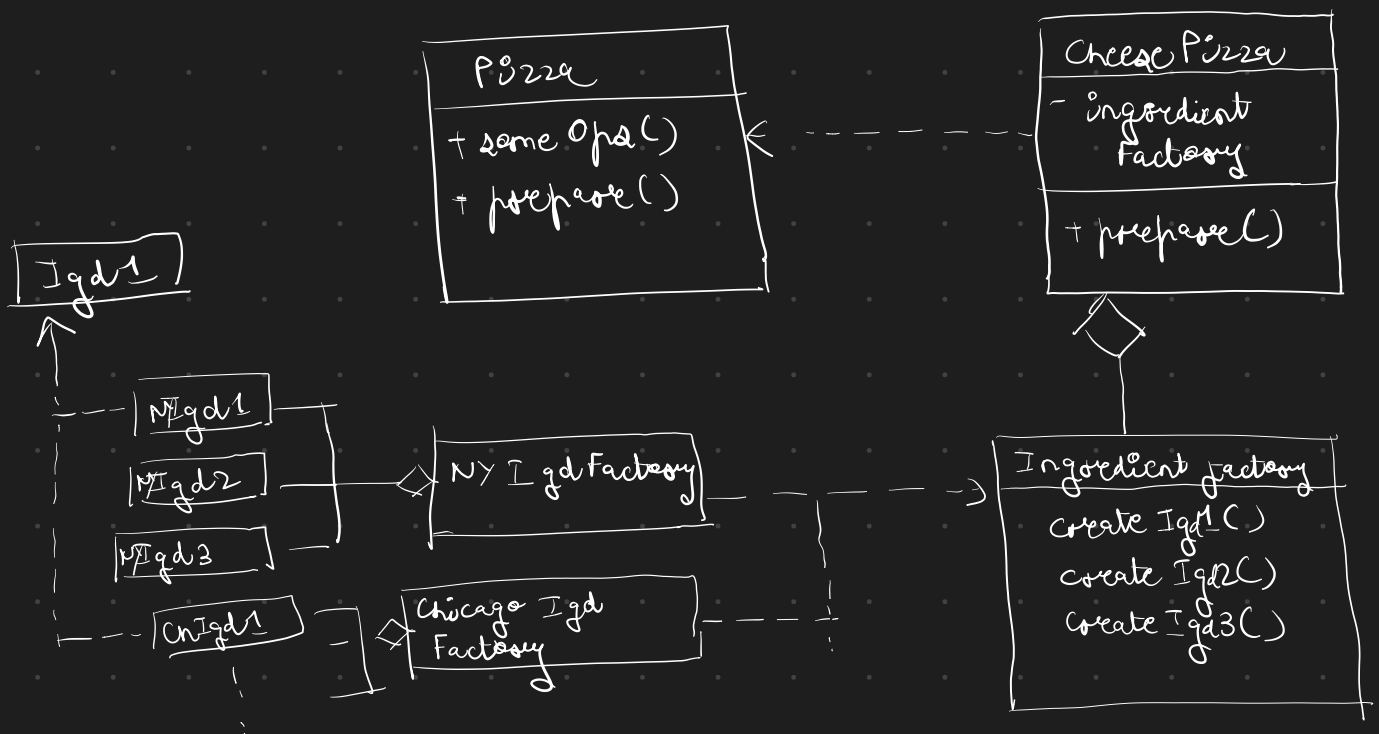
↳ High-level component ⟶ Low-level component ✗

↳ Low-level + High-level component ⟶ abstractions ✓

By applying factory pattern, we make sure that dependency inversion principle is adhered.

Another Eg

↳ Pizza with ingredients

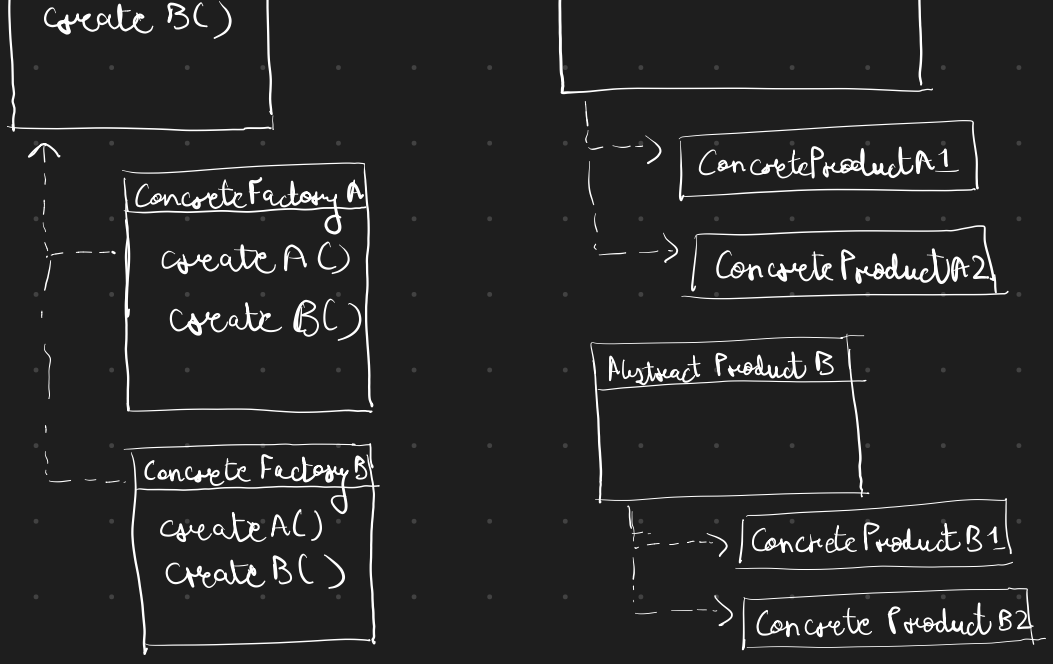


In above eg: abstract factory pattern is used

Abstract factory pattern

It allows us to create a family of products without specifying their concrete class.

General Design


create B()
ConcreteFactory A
create A()
create B()
ConcreteFactory B
create A()
Create B()
ConcreteProductA1
ConcreteProductA2
Abstract Product B
Concrete Product B1
Concrete Product B2