Factory Method パターン

❖ 目的

Factory Method パターンは、他のクラスのコンストラクタをサブクラスで上書き可能な自分のメソッドに置き換えることで、アプリケーションに特化したオブジェクトの生成をサブクラスに追い出し、クラスの再利用性を高めることを目的とする。

❖ 効果

実際に生成されるインスタンスに依存しない、インスタンスの生成方法を提供する。

❖ 背景

レストランでのキッチンは、工場のようなものです。それはレストランのプロダクションルームです。

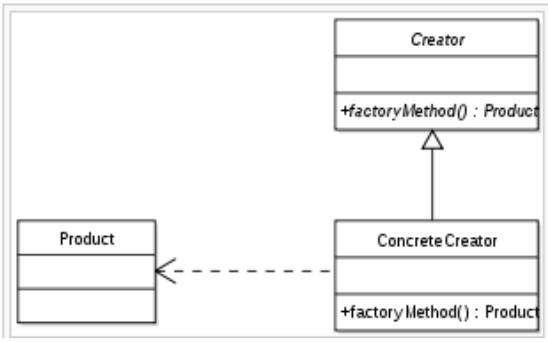
❖ **Factory Method パターン**の実際のコートと考え方

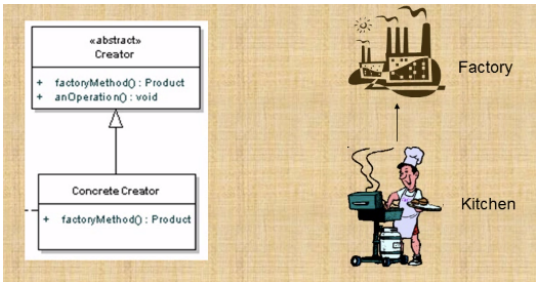
顧客のメニューからの注文とは、実際にそれを準備し取得する方法を知っている気にされていません。順番に現在の接続で、食べるための素晴らしい料理を取得します。

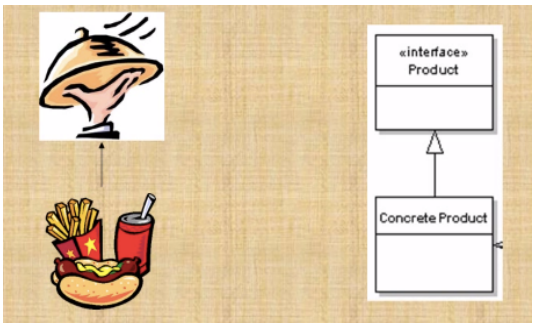
```
# Pizza (Product)
class Pizza
    def initialize(name)
        @name = name
    end

    def cook
        puts " pizza #{@name} は出来上がりました。"
    end
end

# Pasta (Product)
class Pasta
    def initialize(name)
        @name = name
    end

    def cook
        puts " pasta #{@name} は出来上がりました。"
    end
end

# Resturant 工場 (Creator)
class ResturantFactory
    def initialize(number_reciept)
        @reciept = []
        number_reciept.times do |i|
```

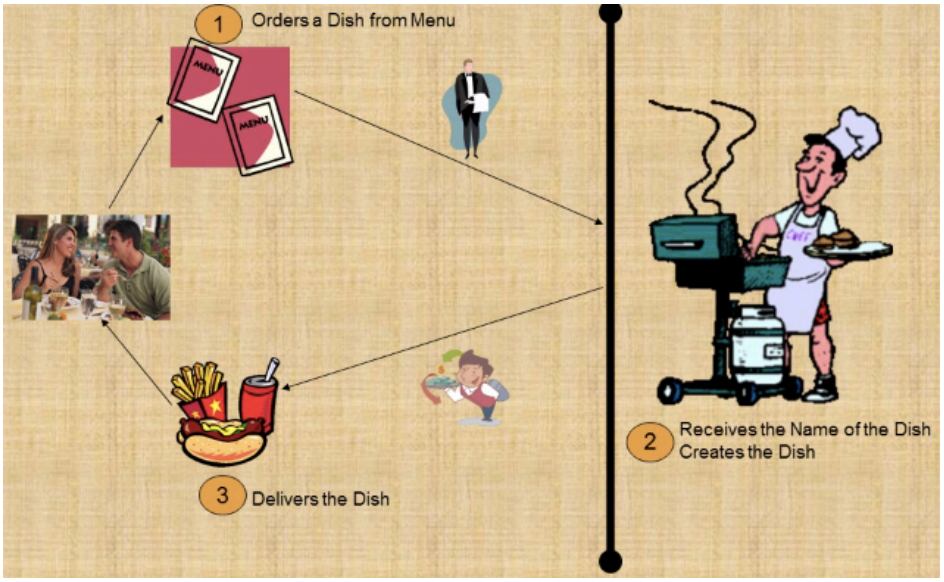
```ruby
            reciept = new_reciept("reciept #{i}")
            @reciept << reciept
        end
    end

    # 料理を作る
    def ship_out
      tmp = @reciept.dup
      @reciept = []
      tmp
    end
end

# PizzaFactory: Pizzaを生成する (ConcreteCreator)
class PizzaFactory < ResturantFactory
    def new_reciept(name)
        Pizza.new(name)
    end
end

# PastaFactory: Pastaを生成する (ConcreteCreator)
class PastaFactory < ResturantFactory
    def new_reciept(name)
        Pasta.new(name)
    end
end

# ============================================
factory = PizzaFactory.new(3)
pizza = factory.ship_out
pizza.each { |pizza| pizza.cook }
#=> pizza reciept 0 は出来上がりました。
#=> pizza reciept 1 は出来上がりました。
#=> pizza reciept 2 は出来上がりました。

factory = PastaFactory.new(2)
pasta = factory.ship_out
pasta.each { |pasta| pasta.cook }
#=> pasta reciept 0 は出来上がりました。
   #=> pasta reciept 1 は出来上がりました。
```

❖ **Factory Method パターン**のまとめ









１．注文する
２．料理を作る
３．お客さんに送る

❖ 総括
ファクトリパターンは、作成パターンである。それは、創造の詳細を公開することなく、オブジェクトの作成に役立ちます。