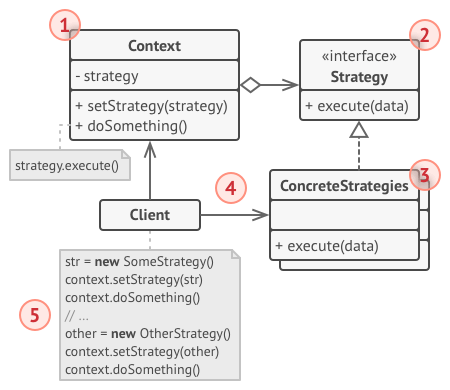
**Strategy Pattern:** es un patrón de diseño de comportamiento que le permite definir una familia de algoritmos, colocar cada uno de ellos en una clase separada y hacer que sus objetos sean intercambiables.



1. **Estructura**
   1. El **contexto** mantiene una referencia a una de las estrategias concretas y se comunica con este objeto solo a través de la interfaz de la estrategia.
   2. La interfaz de **estrategia** es común a todas las estrategias concretas. Declara un método que el contexto utiliza para ejecutar una estrategia.
   3. **Las estrategias concretas** implementan diferentes variaciones de un algoritmo que utiliza el contexto.
   4. El contexto llama al método de ejecución en el objeto de estrategia vinculada cada vez que necesita ejecutar el algoritmo. El contexto no sabe con qué tipo de estrategia funciona o cómo se ejecuta el algoritmo.
   5. El **Cliente** crea un objeto de estrategia específico y lo pasa al contexto. El contexto expone un setter que permite a los clientes reemplazar la estrategia asociada con el contexto en tiempo de ejecución.
2. **Pizza estilo paisa**

name = "Pizza estilo paisa";

        dough = "Masa extra gruesa";

        sauce = "Salsa de tomate para pizza";

        toppings.add("tocineta picada");

        toppings.add("chorizo");

        toppings.add("maíz tierno");

        toppings.add("queso mozzarella rayado");

        toppings.add("salsa de tomate");

        toppings.add("pimentón");

        toppings.add("cebolla cortada en medias rodajas");

        toppings.add("ajo");

1. **Builder**

es un patrón de diseño creativo que le permite construir objetos complejos paso a paso. El patrón le permite producir diferentes tipos y representaciones de un objeto utilizando el mismo código de construcción.