Chapter 4 : The Factory Pattern

- new - every time we use new, we program to an implementation
- when we have code that makes use of lots of concrete classes, code may have to be changed.
- Factories handle details of object creation.
- Advantage of factory pattern:
  By encapsulating pizza creating in one class, we have one place to make modifications when implementation changes. We also remove concrete instantiations.
- Static Factory
  - You don't need to instantiate object
  - But you also can't subclass & change behavior of the create method.
- A factory method handles object creation and encapsulates it in a subclass. This decouples the client code in superclass from object creation code in subclass.

  → Factory method returns a product

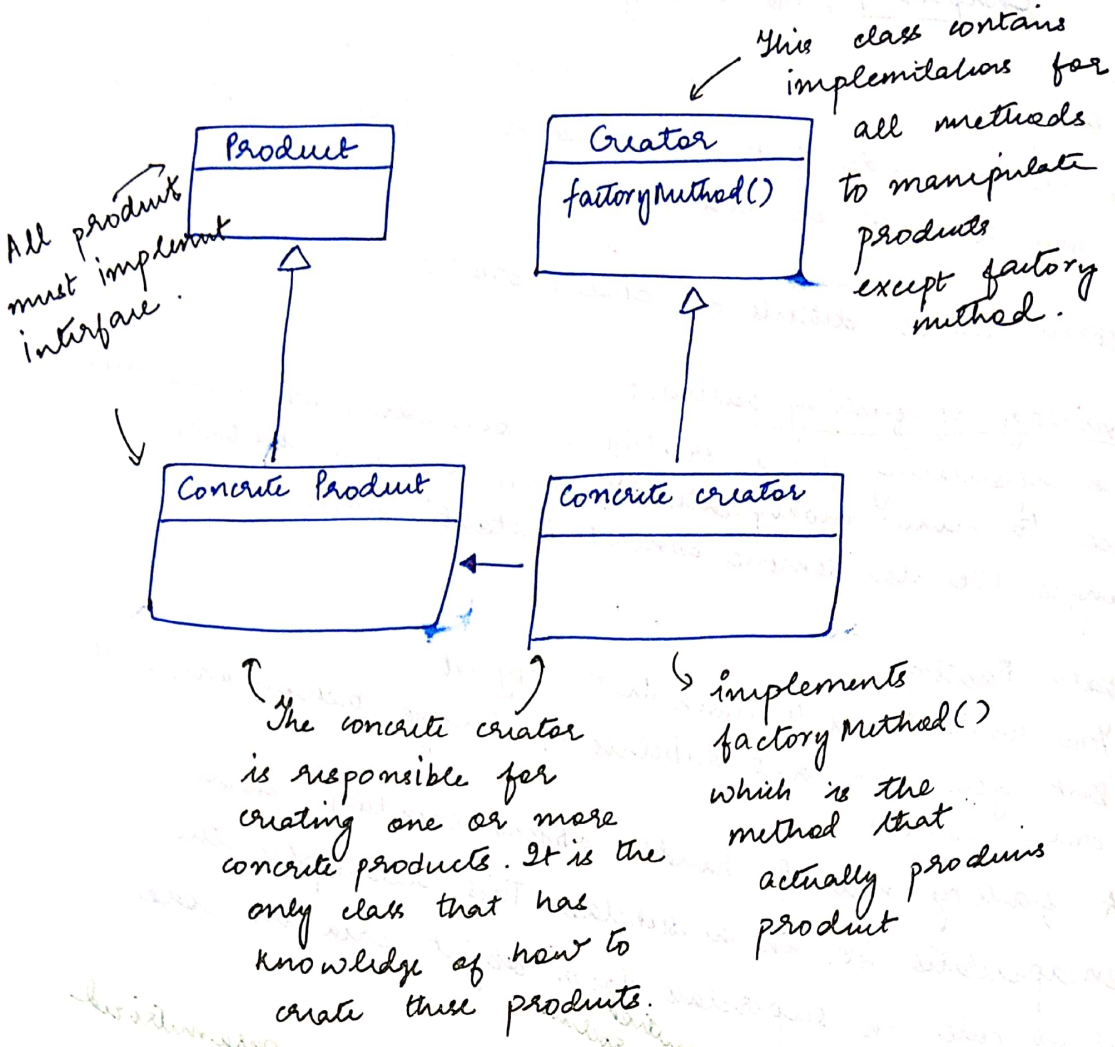
  abstract Product factoryMethod(String type) → parameterized

  abstract: so that subclass are counted on to handle object creation

  isolates client from knowing what kind of concrete Product is created
- Parallel class hierarchis → creator class, → Product class

DP : The Factory method pattern defines an interface for creating an object, but lets subclass decide which class to instantiate. It lets class defer instantiation to subclass.

- Factory method lets subclass decide which class to instantiate.



- No dumb questions ✓

- Advantage of Factory method pattern when you have one concrete creator, because we are decoupling the implementation of its product from use.

- Factory method and creator aren't abstract as we can define default factory method to produce some concrete product. Then we always have means of creating products even if there is no subclass of creator.

  - Parametrized factory method vs non-parametrized.
  - Use static const or enums to make parameters safe.
  - Simple factory - one shot deal
    Factory method - subclass decide which deal to be used.

• Master & student

• Factories allows to encapsulate this behaviour of instantiation

• Benefits of Factories:

1) Avoid duplication
2) provide maintanance
3) clients depend upon interfaces rather than concrete classes.
4) Flexible & extensible code.

★ Dependency Inversion Principle. Depend upon abstractions, not concrete classes.

★ high level component (class with behaviour defind in terms of other low-level components).

★ worse is the inversion in dependency inversion principle?
Both low & high level depend on abstractions.

★ Abstract Factory pattern: provide an interface for creating families of related or dependent objects without specifying their concrete classes.

★ often the methods abstract factory are implem as factory method

★ class digram.