Creational patterns

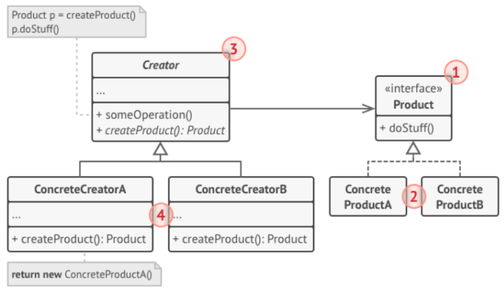
Factory Method Pattern

creating objects without having to specify the exact class of the object
创建实例 不用指定具体要创建的类别

定义一个创建对象的接口，让其子类自己决定实例化哪一个工厂类，
工厂模式使其创建过程延迟到子类进行

The object is created without exposing the creation logic to the client
创建对象时不会对客户端暴露创建逻辑

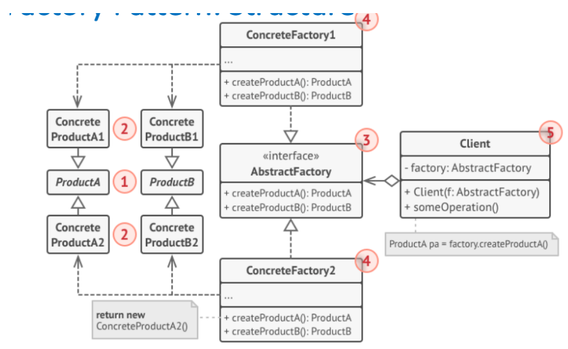
solution
- Define separate operation for create an object
- create object by calling a factory method
- can write subclasses for different way to creating object



- Product <<interface>> — common to all objects that can be produced by the creator and its subclasses 由创建者及其子类生成的所有对象都是通用的
- concrete product — different implementations of the product interface
- Creator class — declare factory method that returns new product objects
- concrete creator — override base factory method, returns a different type of product

Abstract Factory pattern

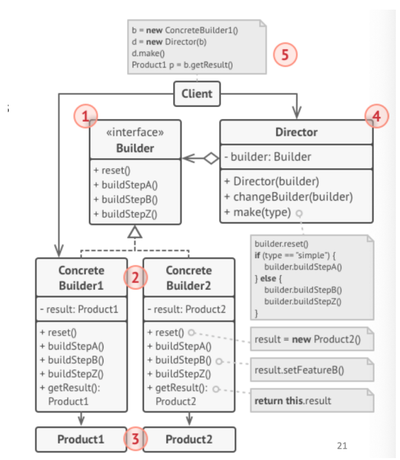
produce families of related objects without specifying their concrete classes



- Abstract product — declare interface for distinct but related products
- concrete products
  - various implementations of abstract products, grouped by variants 抽象产品的各种实现，按变体分组
  - abstract product must implemented in all given variants 每个抽象产品必须以所有给定的变体实现
- Abstract factory <<interface>> — declare a set of method for creating each of the abstract products
- concrete factories
  - implement creation methods of the abstract factory
  - each concrete factory corresponds to a specific variant of products and creates only those product variants 每个具体的工厂对应于产品的特定变体，并且只创建那些产品变体
- client
  - can work with any concrete factory/product variant
  - 只要 it communicates with their objects via abstract interfaces

Builder pattern

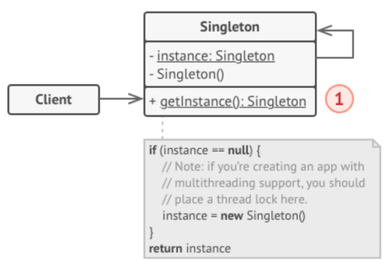
- construct complex objects step by step
- produce different types and representations of an object using same construction code
- extract the object construction code out of its own class and move it to separate objects called builders 将对象构造代码从它自己的类中提取出来，并将其移动到称为构建器的单独对象中
- director
  - define the order of execute building steps
  - builder provides the implementation for those steps



- builder <<interface>> — declares product construction steps that common to all types of builders
- concrete builders
  - provide different implementations of the construction steps
  - may not follow the common interface
- products
  - is resulting object
  - constructed by different builders may not belong to same class hierarchy or interface
- director — define the order in which to call construction steps
- client — must associate one of the builder objects with the director

Singleton pattern

- ensure a class has only one instance
- provide a global access point to instance



- singleton class — declare the static method getInstance() return the instance of the class
- singleton constructor — private — to prevent other object use
- calling getInstance() should be the only way to getting the object
- calling getInstance() always return the same object