## **工厂模式**

工厂模式可以分为简单工厂、工厂方法、抽象工厂三类

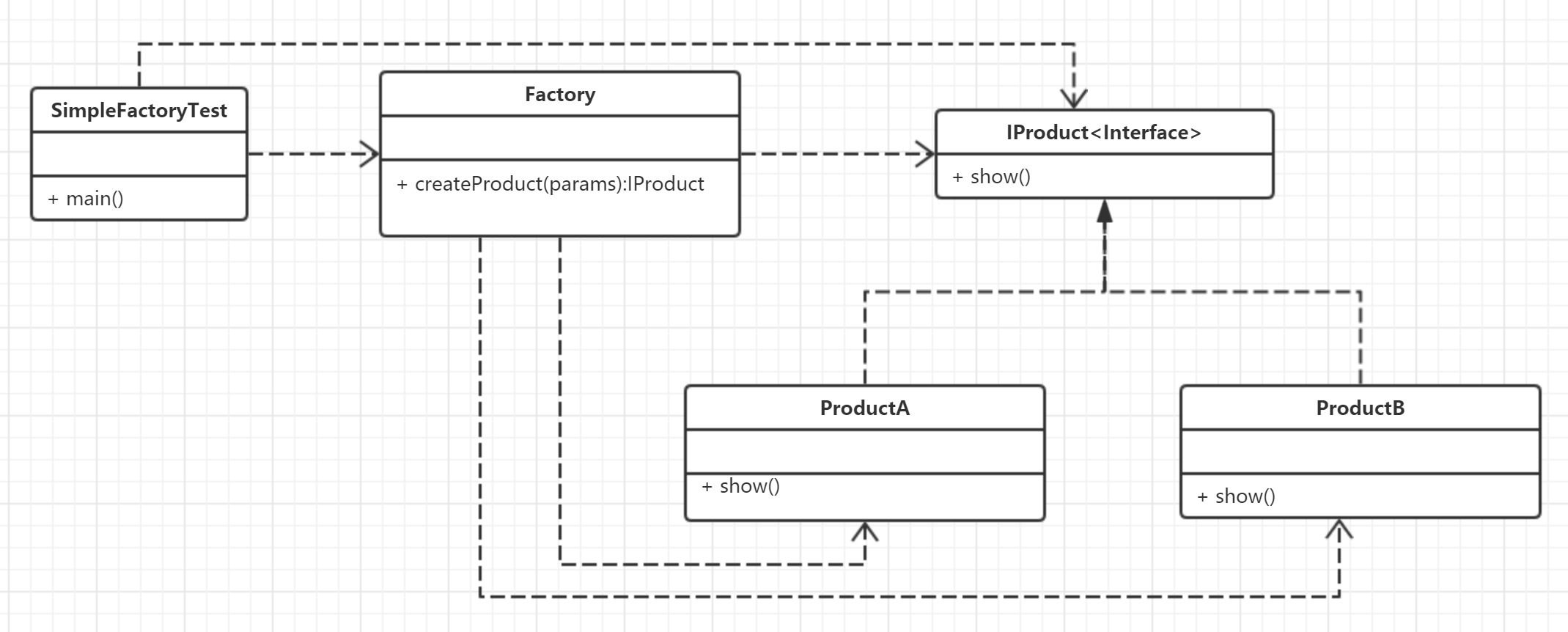
## **简单工厂**

把类的创建过程封装到工厂里面，降低和客户端的耦合

\*\*优点：\*\*简单，只需要传需要生产的类的必要信息，到工厂类中接口中即可，比如名字或者类对象

\*\*缺点：\*\*如果新增一个待创建类，那么就需要修改工厂类，没有遵循开闭原则，对修改没有关闭

UML:



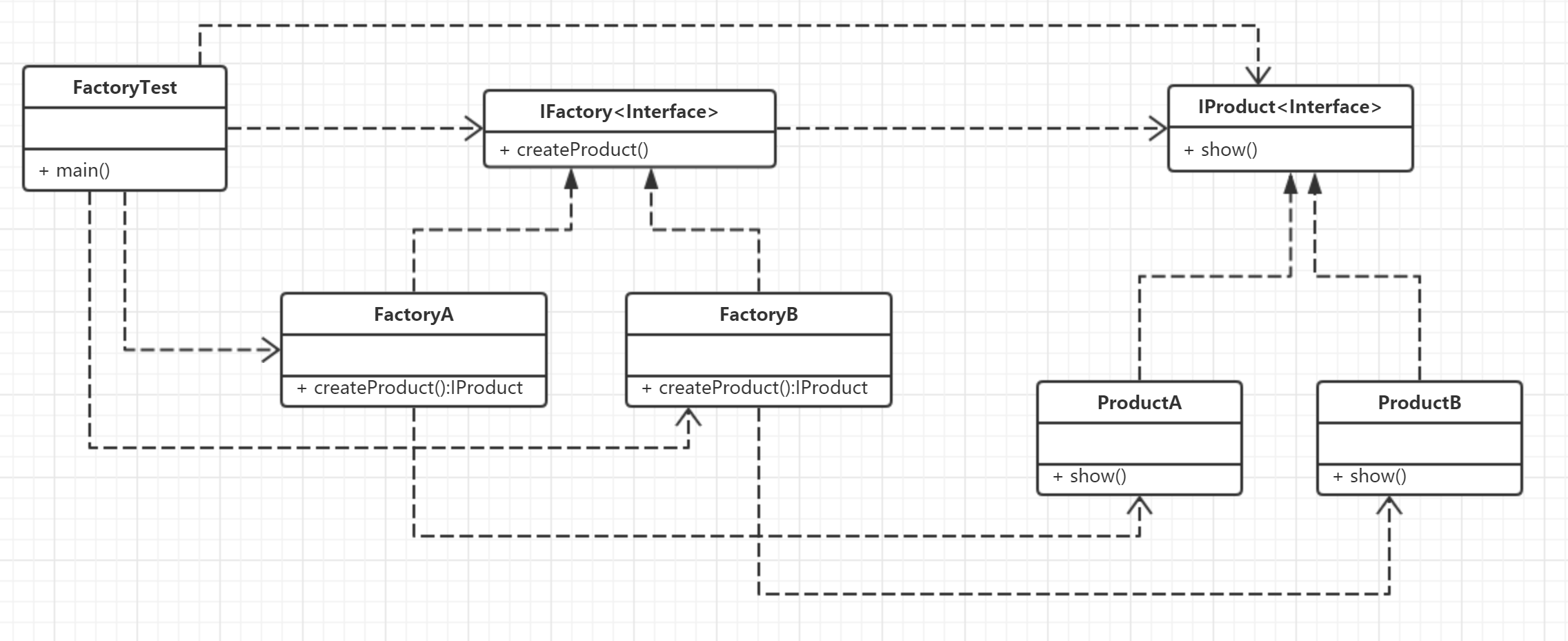
## **工厂方法**

在简单工厂的基础上，进一步改造，遵循开闭原则，改造过程为：设计工厂接口（或抽象类），每个产品可以设计一个对应的产品工厂实现工厂接口，这样每次新增一个产品，就不用修改原有的工厂代码，只需要新增一个工厂方法和产品类即可

\*\*优点：\*\*遵循开闭，维护性比简单工厂高

\*\*缺点：\*\*代码较多，接口稍微复杂，每次新增一个产品，都要建立产品工厂，而且工厂未考虑生产产品簇的问题，一个工厂可能需要生产产品簇（比如生产一台汽车，里面的内部零件组合即为产品簇）内的其他产品。

UML:



## **抽象工厂**

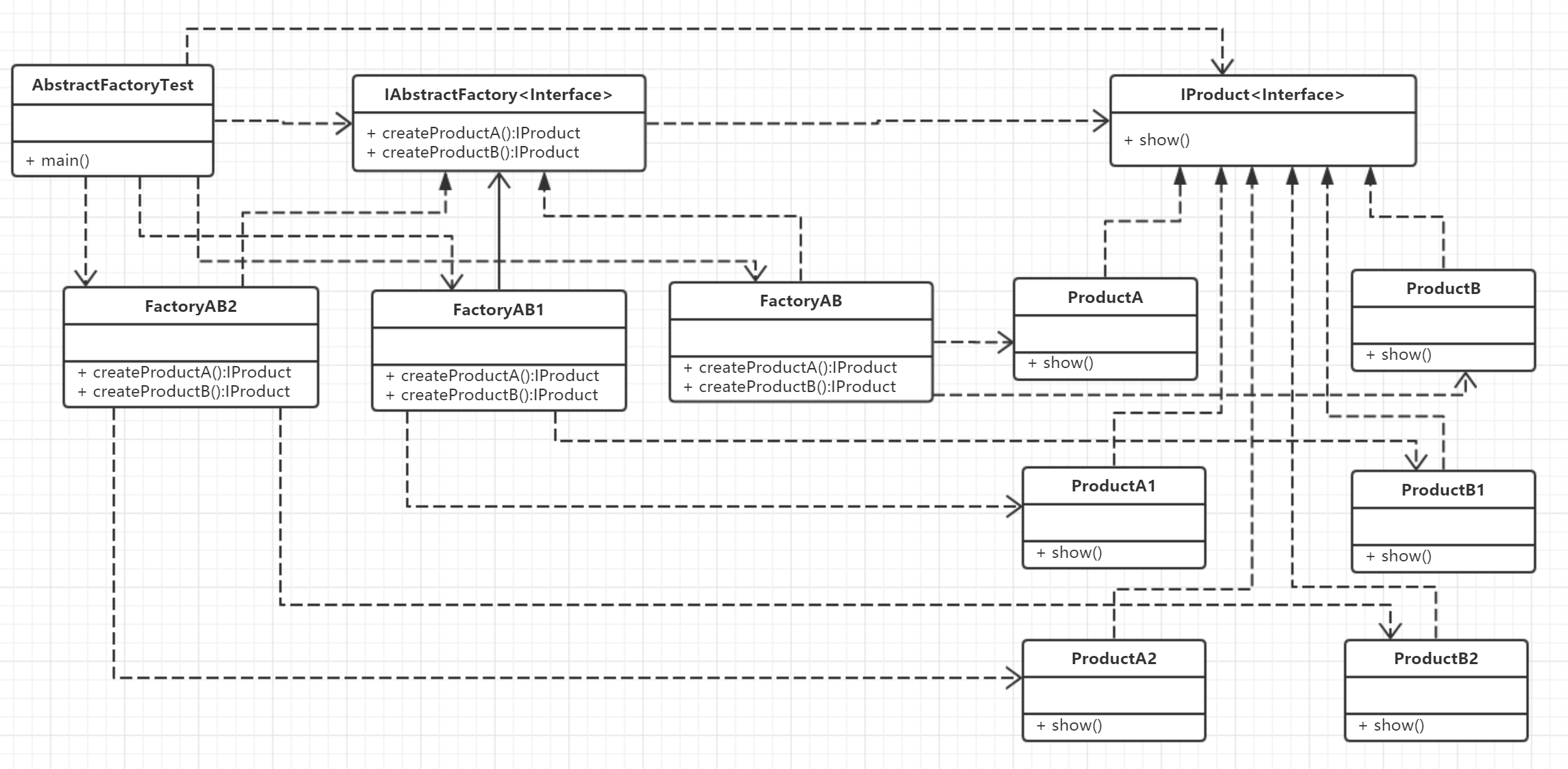
解决工厂生产产品簇的问题。需要对产品簇进行定义（有多少种产品），一类的相关产品创建封装到对应工厂中，不同工厂之间产品的关系可以理解为产品等级不同。打个比方华为手机有两种品牌：华为、荣耀，,这两个可理解为工厂，然后分别生产的手机型号为Mate10（中高端市场），荣耀10（中低端市场），针对不同市场而创建的工厂产品。Mate10和荣耀10的关系可理解为产品等级不同，如Mate10的摄像头为超清摄像头，荣耀10的摄像头为高清摄像头。

实现过程为：定义抽象产品簇工厂接口，然后每新增一种产品，就实现一个产品簇工厂和对应的产品部件类。

****优点：**** 更贴近现实，解决工厂生产的内部产品簇问题。

****缺点：**** 未遵循开闭原则，如果产品簇新增内部产品，那么就需要修改抽象产品簇接口。

UML:



## **总结**

​ 上面三种工厂，都是为了降低耦合，提高可维护性，在现实编码应用过程中，没有一定的优劣性，简单工厂未必就是不好，如果要创建的类比较少（如5个以内，不一定能是这个数），然后可以预见后面都不需要新增类或者新增类的概率很低了，那么简单工厂用在这种场景下是很好的选择，简单，有时候也代表维护性高，不需要那么多学习上手在里面。

​ 有些场景，工厂方法可以和抽象工厂混合使用，或者和其他模式混合使用，达到解耦也遵循7大原则也是可行的，形态随场景而变。