# 基本概念

它提供了一种创建对象的最佳方式，在工厂模式中，我们在创建对象时不会对客户端暴露创建逻辑，并且是通过使用一个共同的接口来指向新创建的对象。

# 意图

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。

# 主要解决

主要解决接口选择的问题。

# 何时使用

我们明确地计划不同条件下创建不同的实例。

# 如何解决

让其子类实现工厂接口，返回的也是一个抽象的产品。

# 应用实例

1、您需要一辆汽车，可以直接从工厂里面提货，而不用去管这辆汽车是怎么做出来的，以及这个汽车里面的具体实现。

2、Hibernate 换数据库只需换方言和驱动就可以。

# 优点

1、一个调用者想创建一个对象，只要知道其名称就可以了。

2、扩展性高，如果想增加一个产品，只要扩展一个工厂类就可以。

3、屏蔽产品的具体实现，调用者只关心产品的接口。

# 缺点

1、每次增加一个产品时，都需要增加一个具体类和对象实现工厂，使得系统中类的个数成倍增加，在一定程度上增加了系统的复杂度，同时也增加了系统具体类的依赖，这并不是什么好事。

# 使用场景

1、日志记录器：可能记录到本地硬盘、系统事件、远程服务器等，用户可以选择记录日志到什么地方。

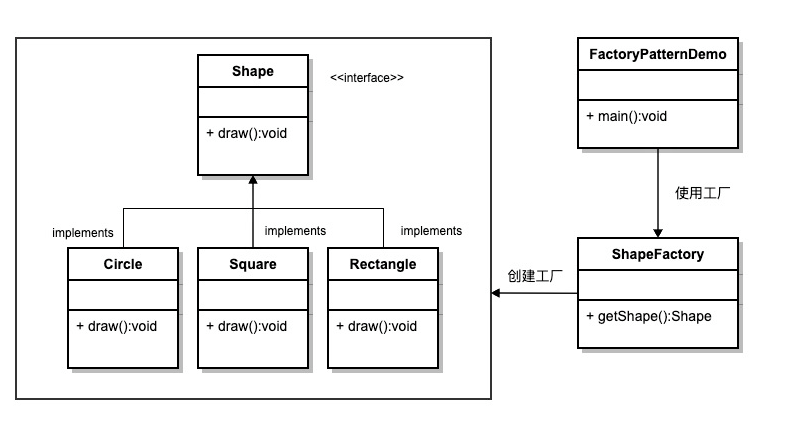
2、数据库访问，当用户不知道最后系统采用哪一类数据库，以及数据库可能有变化时。

# 注意事项

作为一种创建类模式，在任何需要生成复杂对象的地方，都可以使用工厂方法模式。有一点需要注意的地方就是复杂对象适合使用工厂模式，而简单对象，特别是只需要通过 new 就可以完成创建的对象，无需使用工厂模式。如果使用工厂模式，就需要引入一个工厂类，会增加系统的复杂度。

# 代码示例

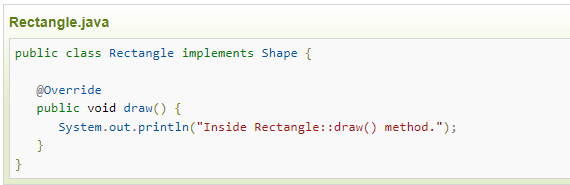
我们将创建一个 Shape 接口和实现 Shape 接口的实体类。下一步是定义工厂类 ShapeFactory，FactoryPatternDemo 类使用 ShapeFactory 来获取 Shape 对象。它将向 ShapeFactory 传递信息（CIRCLE / RECTANGLE / SQUARE），以便获取它所需对象的类型。

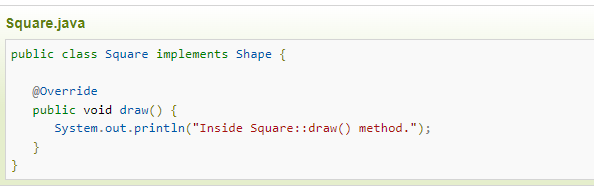


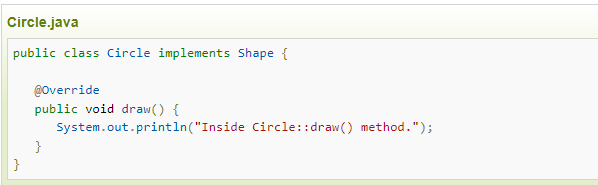
## 1、创建一个接口



## 2、创建实现接口的实体类







## 3、创建一个工厂，生成基于给定信息的实体类的对象。



## 4、使用该工厂，通过传递类型信息来获取实体类的对象。



## 5、执行程序，输出结果

