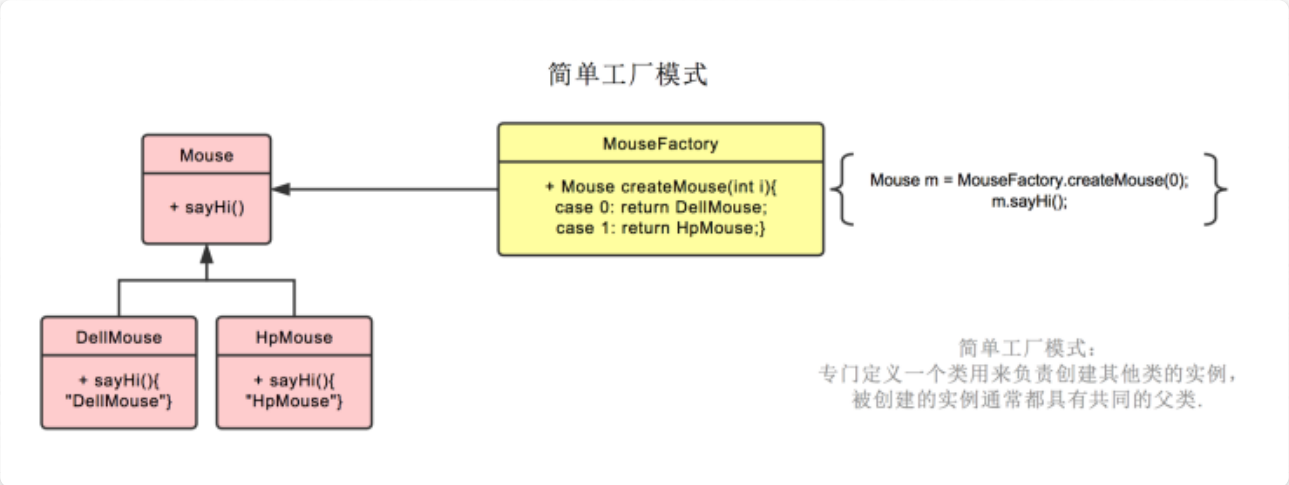
创建型

创建型设计模式包括
工厂方法模式、
抽象工厂模式、
生成器模式（别名：建造者模式）、
原型模式、
单件模式（别名：单例模式），
一共5种。

0.简单工厂模式

意图：专门定义一个类用来创建其他类的实例，被创建的实例通常具有共同的父类。简单工厂模式不是23种里的一种。

解释：简而言之，就是有一个专门生产某个产品的类。比如下图中的鼠标工厂，专业生产鼠标，给参数0，生产戴尔鼠标，给参数1，生产惠普鼠标。

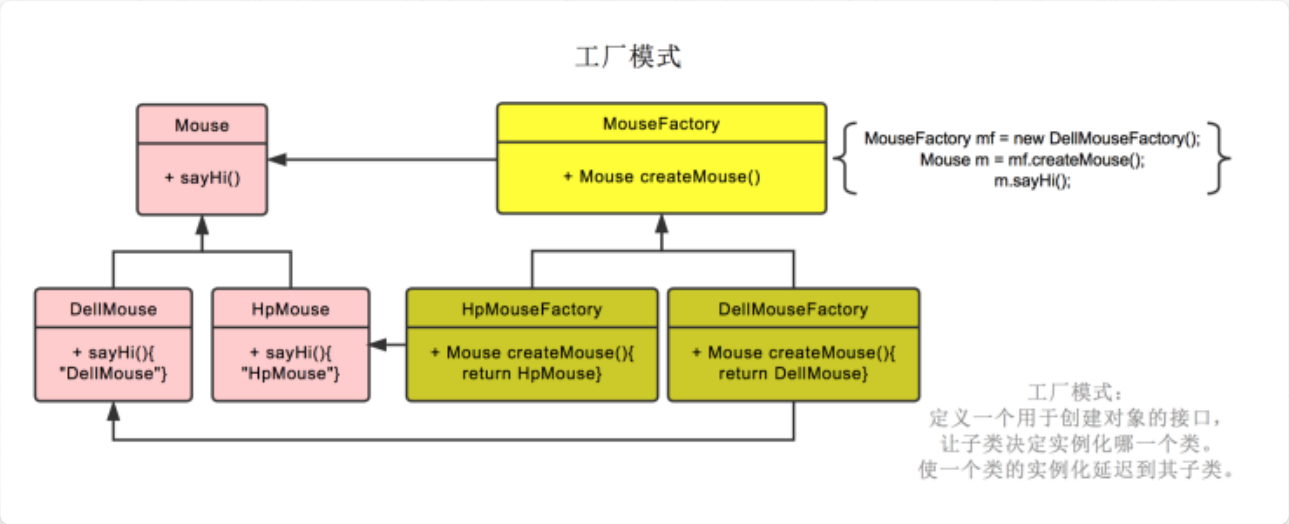


1.工厂方法模式

意图：定义一个用于创建对象的接口，让子类决定实例化哪一个类，使一个类的实例化延迟到起子类。

解释：工厂模式相比简单工厂模式，就是不再以传参数的形式来创建产品，而是通过创建具体子类来创建相应产品。举例来说，也就是鼠标工厂是个父类，有生产鼠标这个接口。 戴尔鼠标工厂，惠普鼠标工厂继承它，可以分别生产戴尔鼠标，惠普鼠

标。 生产哪种鼠标不再由参数决定，而是创建鼠标工厂时，由戴尔鼠标工厂创建。 后续直接调用鼠标工厂.生产鼠标()即可。



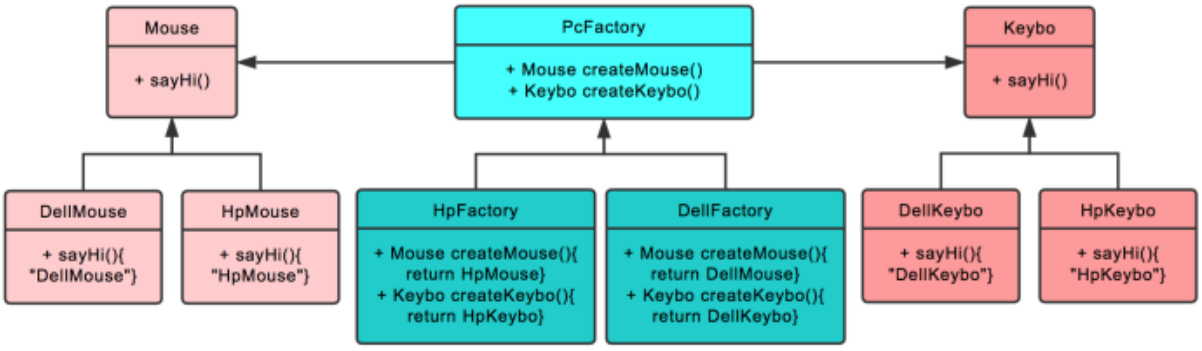
2.抽象工厂模式

意图：提供一个创建一系列或者相互依赖对象的接口，而无需指定它们的具体类。

解释：当工厂模式里的产品有多个种类时，比如添加键盘这一产品，一个具体的工厂就提供生产鼠标和提供生产键盘这两个功能了，这就变成了抽象工厂模式。具体的工厂提供生产鼠标和键盘的接口，这就是定义里说的“提供一个创建一系列或者相互依赖对象的接口”。举例来说，抽象工厂模式也就是不仅生产鼠标，同时生产键盘。 就是PC厂商是个父类，有生产鼠标，生产键盘两个接口。 戴尔工厂，惠普工厂继承它，可以分别生产戴尔鼠标+戴尔键盘，和惠普鼠标+惠普键盘。 创建工厂时，由戴尔工厂创建。 后续工厂.生产鼠标()则生产戴尔鼠标，工厂.生产键盘()则生产戴尔键盘。

iOS中的类簇：类簇是一种把一个公共的抽象超类下的一些私有的具体子类组合在一起的架构。抽象超类负责声明创建私有子类实例的方法，会根据被调用方法的不同分配恰当的具体子类，每个返回的对象都可能属于不同的私有子类，抽象超类负责生成一系列对象的接口，使用抽象工厂模式的思想。 Cocoa将类簇限制在数据存储可能因环境而变的对象生成上。Foundation框架为NSString、NSData、NSDictionary、NSSet、和NSArray对象定义了类簇。公共超类包括上述的不可变类和与其相互补充的可变类NSMutableString、NSMutableData、NSMutableDictionary、NSMutableSet、和NSMutableArray。

抽象工厂模式

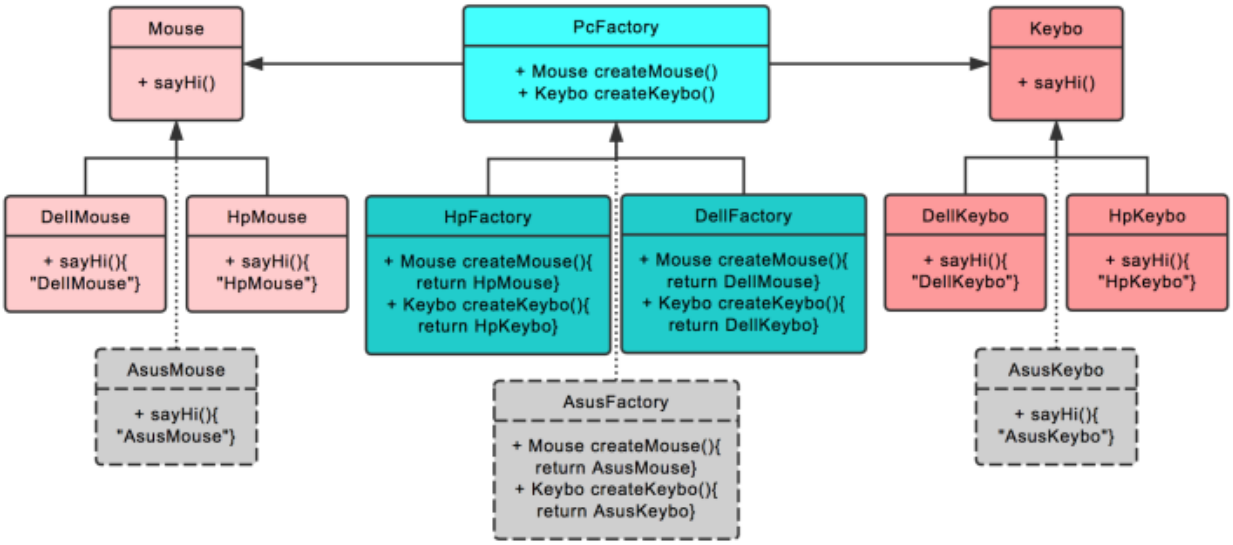


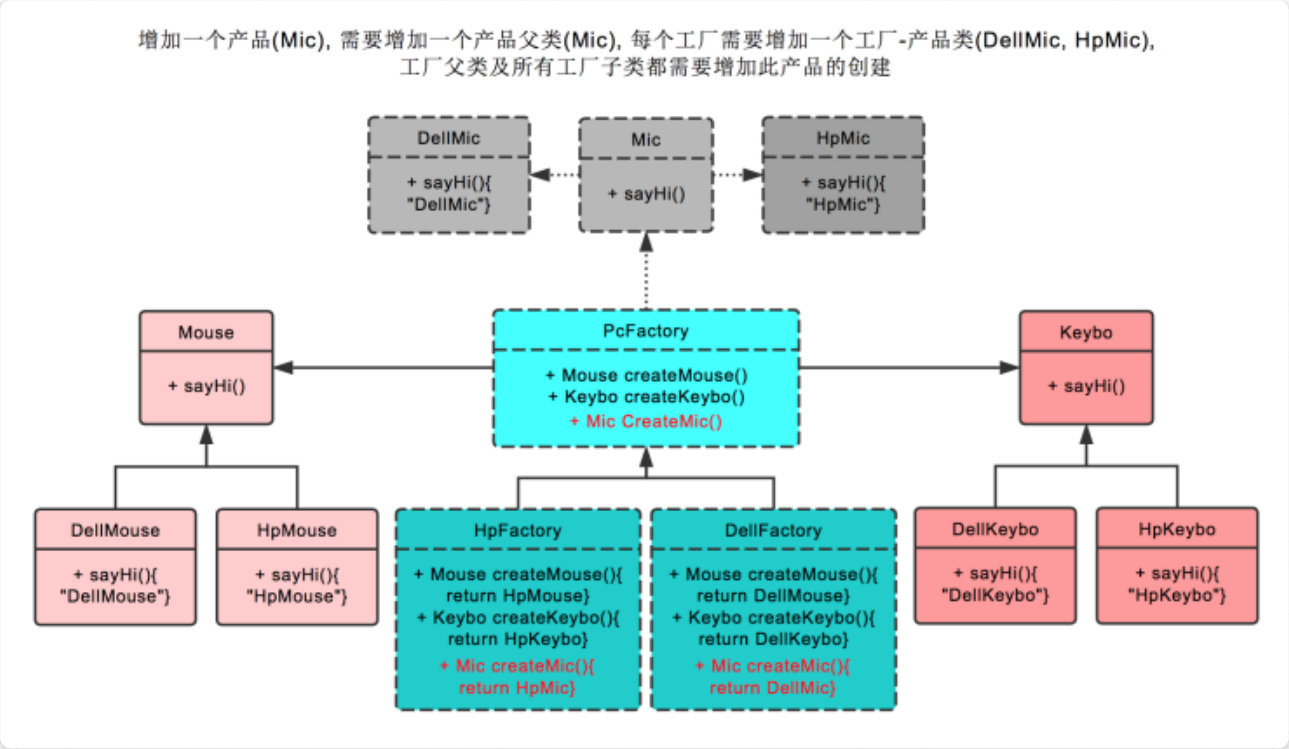

```
PcFactory pcFactory = new HpFactory;
Mouse mouse = pcFactory.createMouse();
Keybo keybo = pcFactory.createKeybo();
mouse.sayHi();
keybo.sayHi();
```

抽象工厂模式:
提供一个创建一系列相关
或者相互依赖对象的接口,
而无需指定他们具体的类

当产品只有一个的时候，抽象工厂模式即变成工厂模式
当工厂模式的产品变为多个时，工厂模式即变成抽象产品模式

增加一个工厂(Asus)，需要增加一个工厂类(AsusFactory)，每个产品需要增加一个工厂-产品类(AsusMouse, AsusKeybo)





抽象工厂模式与工厂方法模式的区别:
1.工厂方法模式:
一个抽象产品类，可以派生出多个具体产品类。
一个抽象工厂类，可以派生出多个具体工厂类。
每个具体工厂类只能创建一个具体产品类的实例。

2.抽象工厂模式:
多个抽象产品类，每个抽象产品类可以派生出多个具体产品类。
一个抽象工厂类，可以派生出多个具体工厂类。
每个具体工厂类可以创建多个具体产品类的实例。

3.区别:
工厂方法模式只有一个抽象产品类，而抽象工厂模式有多个。
工厂方法模式的具体工厂类只能创建一个具体产品类的实例，而抽象工厂模式可以创建多个。

4.联系
如果抽象工厂模式里的产品种类就只有一个的时候，抽象工厂模式也就变成了工厂方法模式

3.原型模式

意图：用原型实例指定创建对象的种类，并且通过拷贝这些原型创建新的对象。

解释：iOS中的 - copy 和 - mutableCopy，注意深复制与浅复制的区别。

4.单件模式

意图：保证一个类只有一个实例，并提供一个访问它的全局访问点。

解释：单件模式很简单，但需要注意多线程环境下的安全问题，需要注意通过其他渠道创建对象的方式，重写相关初始化方法，做到严谨化单例模式。

5.生成器模式

意图：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

解释：封装一个复杂对象的构建过程，并可以按步骤构造。建造者模式所创建的产品一般具有较多的共同点，其组成部分相似，如果产品之间的差异性很大，则不适合使用建造者模式。
举例来说，比如在玩“极品飞车”这款游戏，那么每一关的地图会千变万化，简单的来说，地图会有晴天和阴天之分，那么创建地图时就要根据晴天或者阴天来对地图上的场景，比如：天空，树，房子，和路面进行渲染，这个过程是一个固定的，每创建一个新地图都要执行这几个渲染，这是针对高级配置的电脑来说的。 现在拥有低配置电脑的人不在少数，那么他们就不能玩游戏了吗？完全可以！只要将地图中占用资源比较高的渲染去掉就可以，比如带反射光影的树，这时候需要创建不同的地图，但地图的创建过程却是固定的，建造者模式完全可以应对这样的情况。

生成器模式和抽象工厂模式的对比:

生 成 器	抽象工厂
构建复杂对象	构建简单或复杂对象
以多个步骤构建对象	以单一步骤构建对象
以多种方式构建对象	以单一方式构建对象
在构建过程的最后一步返回产品	立刻返回产品
专注一个特定产品	强调一套产品

参考材料

1.《Objective-C编程之道：iOS设计模式解析》 书籍

2.《设计模式_可复用面向对象软件的基础》 书籍
3. 知乎问答:抽象工厂模式和工厂模式的区别？ （需要指出的是，在这篇文章里，作者直接使用了抽象工厂类创建产品的做法是错误的。）
4. wuji3390 的CSDN 博客：建造者模式应用场景