**第四章 工厂方法模式**

本章知识点：

工厂方法模式的定义

工厂方法模式的结构

工厂方法模式的实现

工厂方法模式的应用

工厂方法模式的优/缺点

工厂方法模式的适用环境

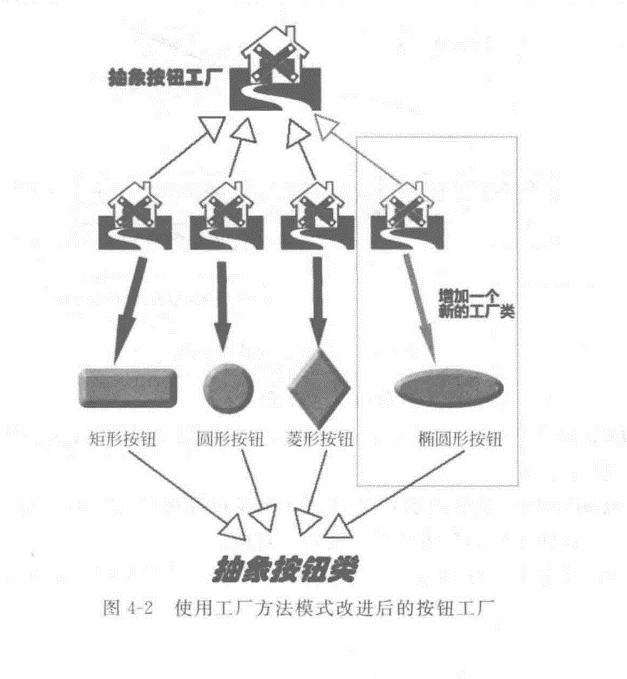
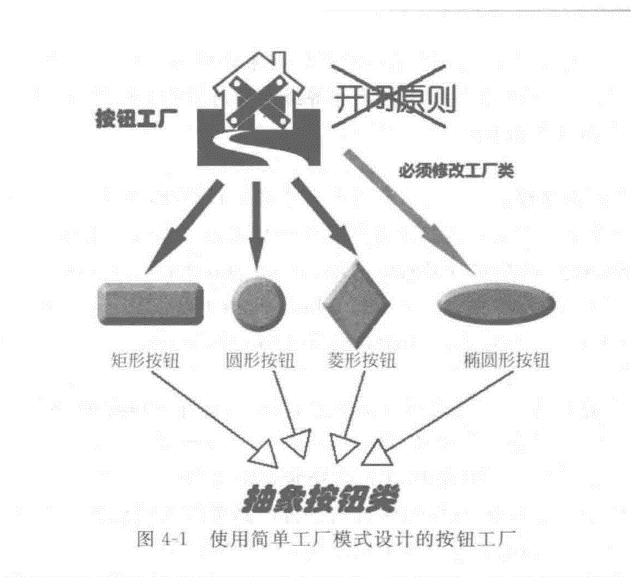
配置文件与反射

工厂方法的重载

工厂方法的隐藏

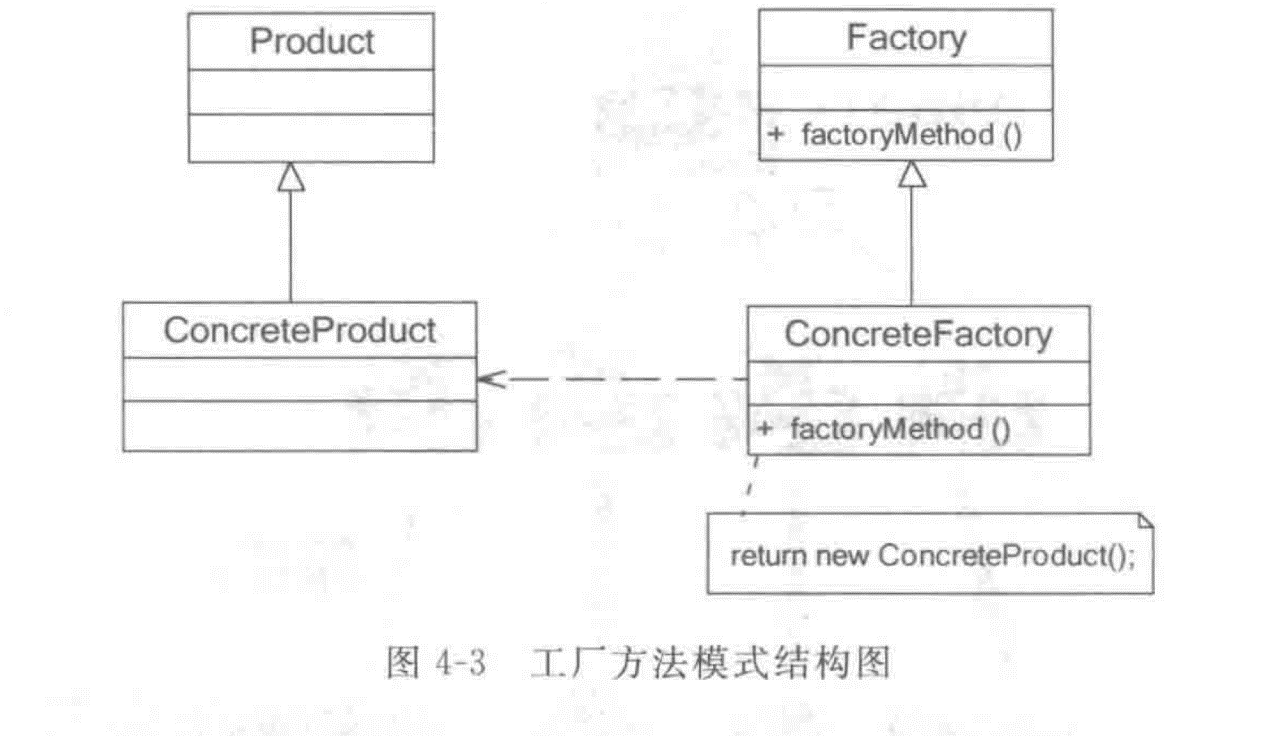
**工厂方法模式概述：**

工厂方法模式是对简单工厂模式的延伸，不再提供一个按钮工厂类来统一负责所有产品的创建,而是将具体按钮的创建过程交给专门的工厂子类去完成。先定义一个抽象的按钮工厂类,再定义具体的工厂类来生产圆形按钮、矩形按钮、菱形按钮等,它们实现了在抽象按钮工厂类中声明的方法。这种抽象化的结果是使得这种结构可以在不修改具体工厂类的情况下引进新的产品,如果出现新的按钮类型,只需要为这种新类型的按钮定义一个具体的工厂类就可以创建该新按钮的实例,这种改进的设计方案即为工厂方法模式。工厂方法模式通过引入抽象的工厂类,使得它具有超越简单工厂模式的优越性,让系统更加符合开闭原则。



其定义为：定义一个用于创建对象的接口，但是让子类决定将哪一个类实例化。工厂方法模式让一个类的实例化延迟到其子类。

**工厂方法模式结构：**

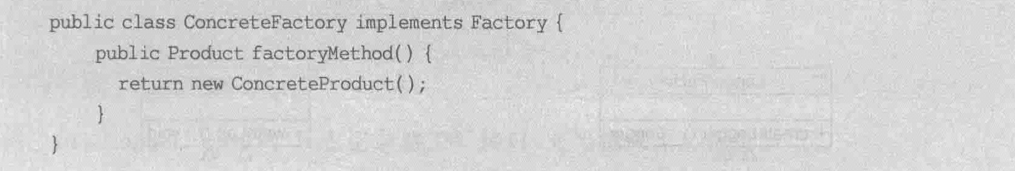


**工厂方法模式实现：**

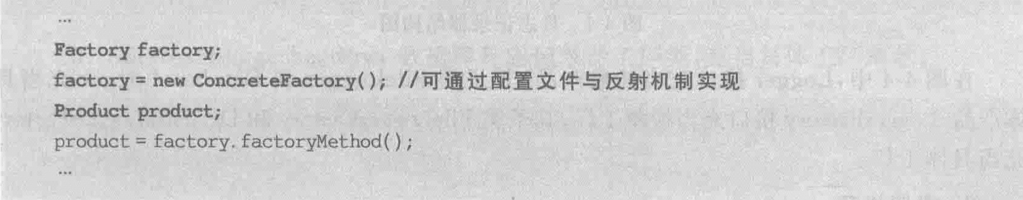
1. 引入抽象工厂角色(抽象工厂可以是接口，也可以是抽象类或者具体类)



1. 客户端针对抽象工厂编程，在运行时再指定具体工厂类，具体工厂类实现工厂方法，不同的具体工厂可以创建不同的具体产品



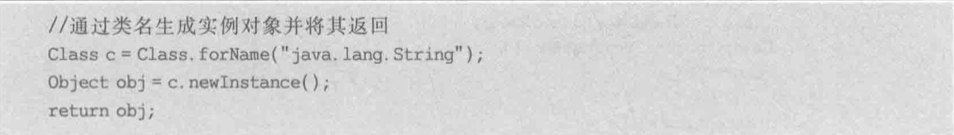
1. 具体工厂类在实现工厂方法时可以创建具体产品对象/负责产品对象初始化/资源环境配置工作



**反射机制与配置文件**

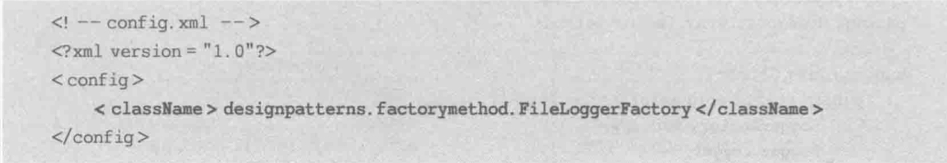
在实际开发中，在客户端代码中不直接使用new创建工厂类，而是通过java反射机制结合配置文件来生成具体工厂对象。

1. Java反射机制

Java反射(Java Reflection)是指在程序运行时获取已知名称的类或已有对象的相关信息的一种机制,包括类的方法、属性、父类等信息，还包括实例的创建和实例类型的判断等。  


1. 配置文件

软件系统的配置文件通常为XML文件，在软件开发中可以把类名存储到XML配置文件中，再读取配置文件获取类名字符串，然后通过Java反射机制来创建对象。



在引入XMLUtil类和XML配置文件之后，如果需要增加一种新类型的日志记录方式，只需要执行以下4个步骤:

(1)新的日志记录器需要继承抽象日志记录器Logger。

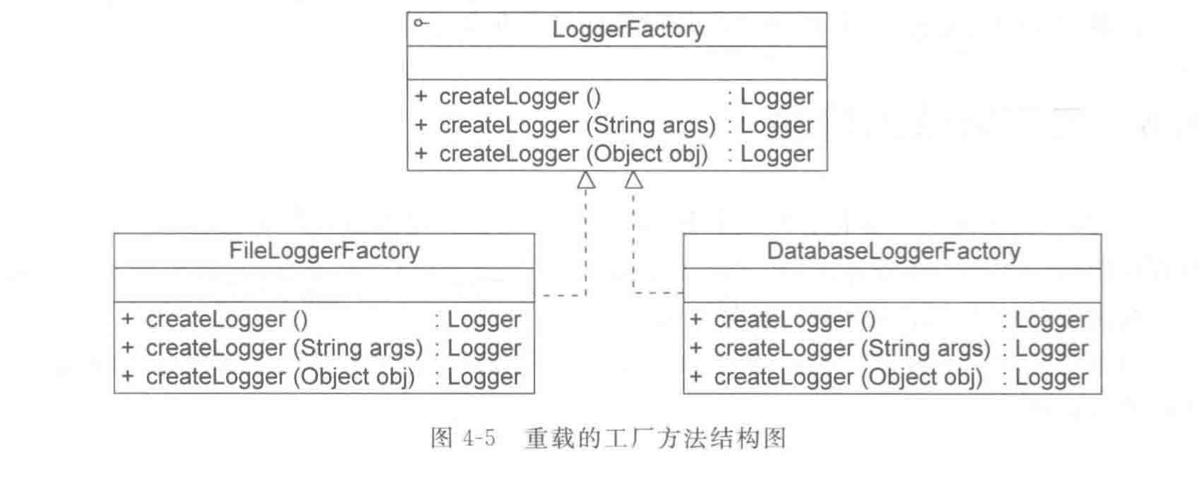
(2)对应增加一个新的具体日志记录器工厂,继承抽象日志记录器工厂LoggerFactory,并实现其中的工厂方法createLogger(),设置好初始化参数和环境变量,返回具体日志记录器

(3)修改配置文件config. xml,用新增的具体日志记录器工厂类的类名字符串替换原有工厂类的类名字符串。

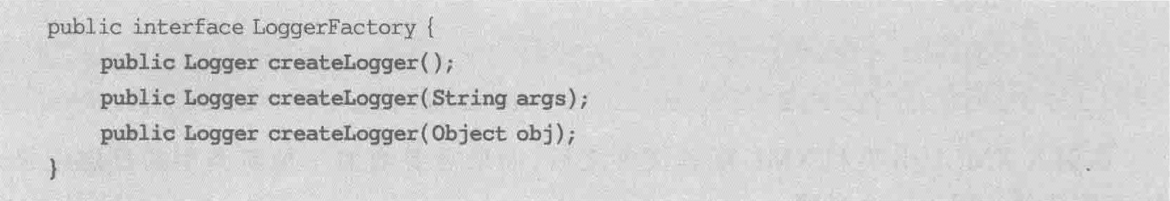
(4)编译新增的具体日志记录器类和具体日志记录器工厂类,运行客户端测试类即可使用新的日志记录方式，而原有类库代码无须做任何修改,完全符合开闭原则。

**工厂方法的重载**

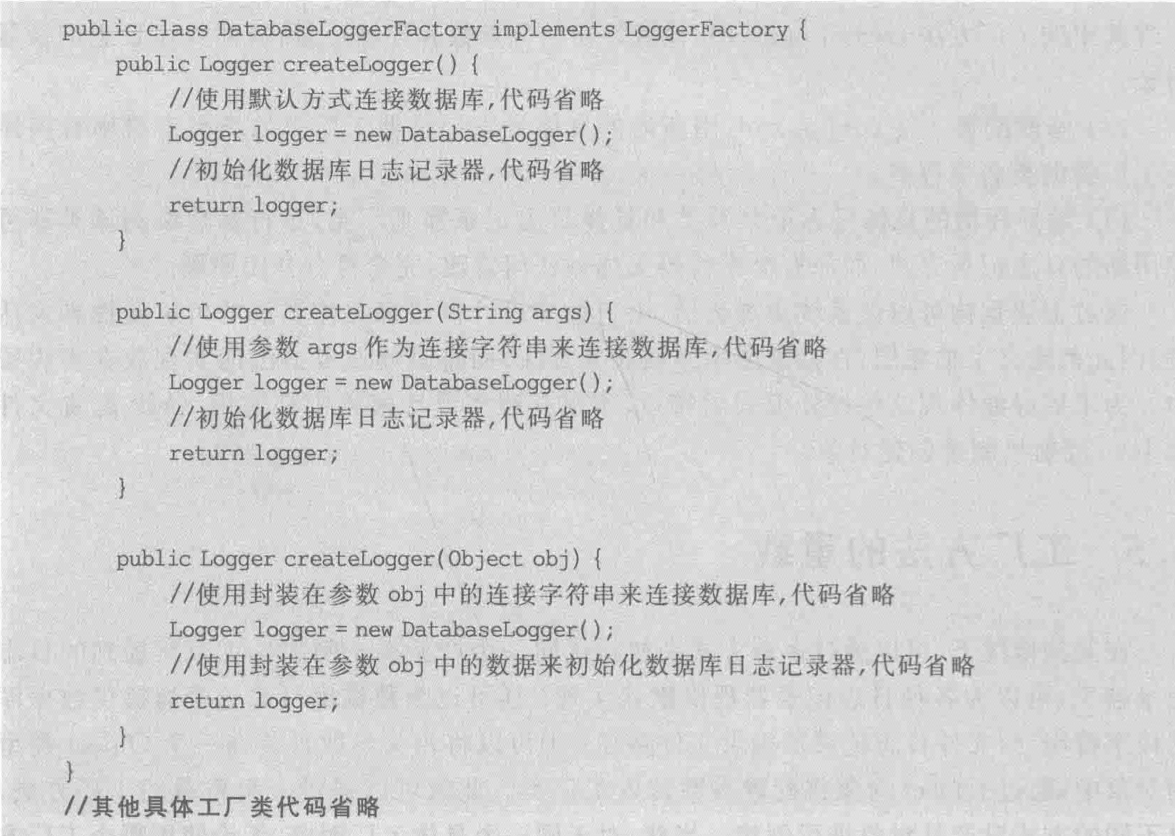
在某些情况下，可以通过多种方式来初始化同一产品类，此时可提供一组重载的工厂方法，以不同的方式对产品对象进行创建。



LoggerFactory代码修改如下：

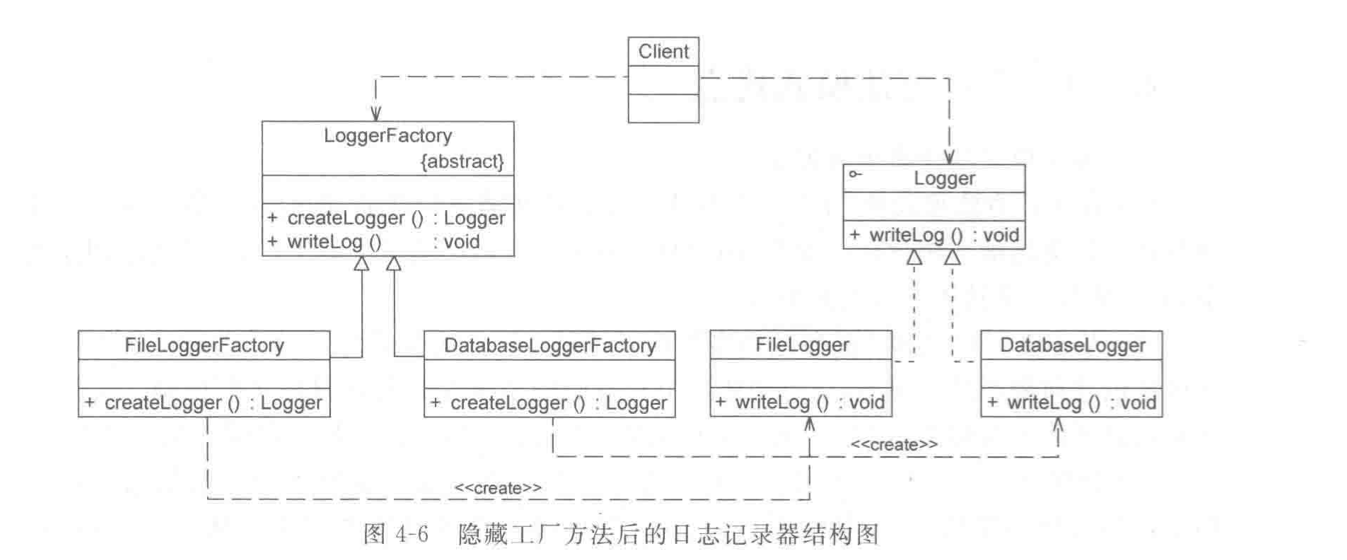


DatabaseLoggerFactory代码修改如下

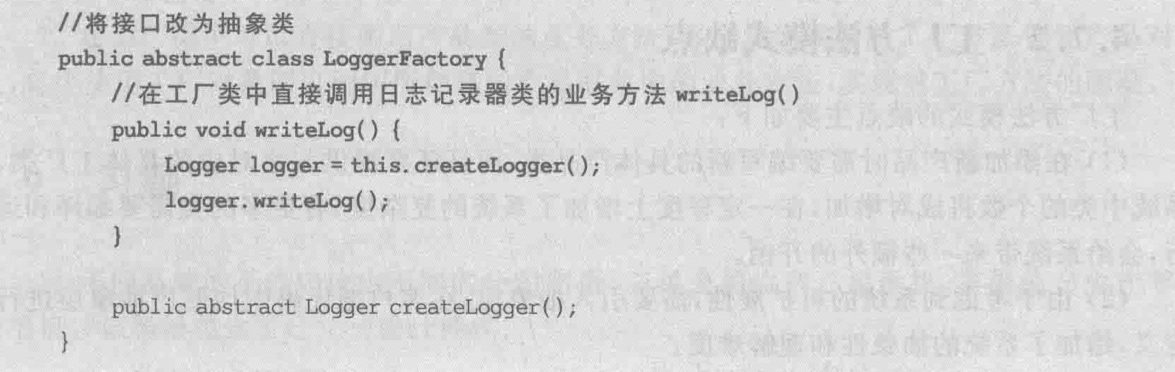


**工厂方法的隐藏**

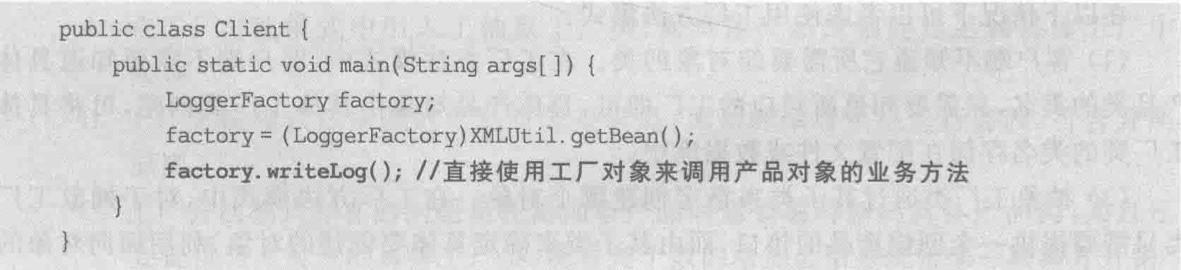
有时候,为了进一步简化客户端的使用,还可以对客户端隐藏工厂方法,此时在工厂类中直接调用产品类的业务方法,客户端无须调用工厂方法创建产品对象,直接使用工厂对象即可调用所创建的产品对象中的业务方法。



LoggerFactory代码修改



客户端代码修改如下



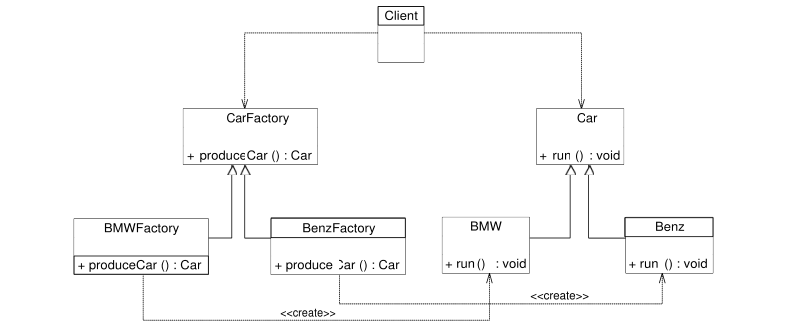
**工厂方法模式优缺点以及适用环境**

工厂方法模式的优点主要如下:  
(1)在工厂方法模式中,工厂方法用来创建客户所需要的产品,同时还向客户隐藏了哪种具体产品类将被实例化这一细节,用户只需要关心所需产品对应的工厂,无须关心创建细节,甚至无须知道具体产品类的类名。  
(2)基于工厂角色和产品角色的多态性设计是工厂方法模式的关键。它能够让工厂自主确定创建何种产品对象,而如何创建这个对象的细节完全封装在具体工厂内部。工厂方法模式之所以又被称为多态工厂模式,正是因为所有的具体工厂类都具有同一抽象父类。

(3)使用工厂方法模式的另一个优点是在系统中加人新产品时无须修改抽象工厂和抽象产品提供的接口,无须修改客户端,也无须修改其他的具体工厂和具体产品，而只要添加一个具体工厂和具体产品即可，这样系统的可扩展性也就变得非常好,完全符合开闭原则。  
  
工厂方法模式的缺点主要如下:  
(1)在添加新产品时需要编写新的具体产品类,而且还要提供与之对应的具体工厂类，系统中类的个数将，成对增加,在一定程度上增加了系统的复杂度,有更多的类需要编译和运行，会给系统带来一些额外的开销。  
(2)由于考虑到系统的可扩展性,需要引人抽象层,在客户端代码中均使用抽象层进行定义,增加了系统的抽象性和理解难度。  
  
在以下情况下可以考虑使用工厂方法模式:  
(1)客户端不知道它所需要的对象的类。在工厂方法模式中,客户端不需要知道具体产品类的类名,只需要知道所对应的工厂即可,具体产品对象由具体工厂类创建,可将具体工厂类的类名存储在配置文件或数据库中。  
(2)抽象工厂类通过其子类来指定创建哪个对象。在工厂方法模式中,对于抽象工厂类只需要提供一个创建产品的接口,而由其子类来确定具体要创建的对象,利用面向对象的多态性和里氏代换原则，在程序运行时子类对象将覆盖父类对象，从而使得系统更容易扩展。

**课后习题答案：**

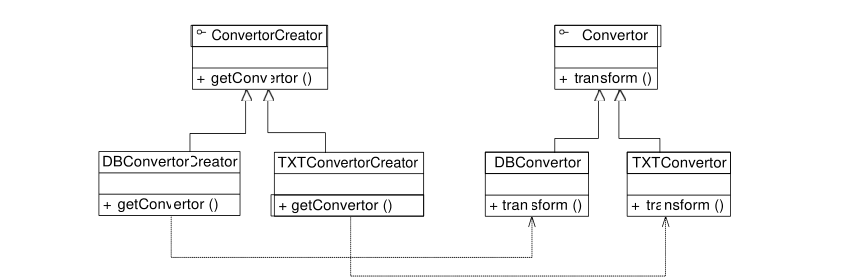
1. D
2. B
3. A B
4. 参考类图如下



其中，Car充当抽象产品，其子类BMW和Benz充当具体产品: CarFactory 充当抽象工厂，其子类BMWFactory和BenzFactory 充当具体工厂。

1. 抽象类/接口Chart 充当抽象产品，其子类LineChart 和BarChart 充当具体产品:抽象类/接口ChartFactory充当抽象工厂，其子类LineChartFactory 和BarChartFactory充当具体工厂。
2. 抽象类/接口Convertor 充当抽象产品， 其子类TXTConvertor、DBConvertor 和ExcelConvertor充当具体产品:抽象类/接口ConvertorCreator 充当抽象工厂，其 子类TXTConvertorCreator、DBConvertorCreator 和ExcelConvertorCreator充当具体工厂。

参考类图如下：



1. 参考类图如下:

