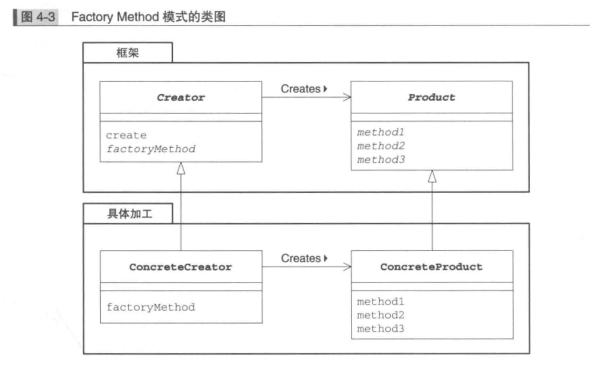
|4.3Factory Method模式中的登场角色

【将实例的生成交给子类】

在Factory Method模式中有以下登场角色。通过查看Factory Method模式的类图（图4-3），我们可以知道，父类（框架）这一方的Creator角色和Product角色的关系与子类（具体加工）这一方的ConcreteCreator 角色和ConcreteProduct角色的关系是平行的。



·Product（产品）Product角色属于框架这一方，是一个抽象类。它定义了在Factory Method模式中生成的那些实例所持有的接口（API），但具体的处理则由子类ConcreteProduct角色决定。在示例程序中，由Product类扮演此角色。◆Creator（创建者）Creator角色属于框架这一方，它是负责生成Product角色的抽象类。但具体的处理则由子类ConcreteCreator角色决定。在示例程序中，由Factory类扮演此角色。

Creator角色对于实际负责生成实例的ConcreteCreator角色一无所知，它唯一知道的就是，只要调用Product角色和生成实例的方法（图4-3中的factoryMethod方法），就可以生成Productde的实例。在示例程序中，createProduct方法是用于生成实例的方法。不用new关键字来生成实例，而是调用生成实例的专用方法来生成实例，这样就可以防止父类与其他具体类耦合。

◆ConcreteProduct（具体的产品）ConcreteProduct角色属于具体加工这一方，它决定了具体的产品。在示例程序中，由IDCard类扮演此角色。

◆ConcreteCreator（具体的创建者）ConcreteCreator角色属于具体加工这一方，它负责生成具体的产品。在示例程序中，由IDCardFactory类扮演此角色。

|4.4拓展思路的要点

|框架与具体加工

至此，我们分别学习了“框架”与“具体加工”这两方面的内容。它们分别被封装在framework包和idcard包中。

这里，让我们用相同的框架创建出其他的“产品”和“工厂”。例如，我们这次要创建表示电视机的类Televison和表示电视机工厂的类TelevisonFactory。这时，我们只需要引入（import）framework包就可以编写televison包。

请注意这里我们没有修改，也根本没有必要修改framework包中的任何内容，就可以创建出其他的“产品”和“工厂”。

请回忆一下，在framework包中我们并没有引入idcard包。在Product类和Factory类中，并没有出现IDCard和IDCardFactory等具体类的名字。因此，即使用已有的框架生成全新的类时，也完全不需要对framework进行修改，即不需要“将televison包引入到框架中”。关于这一点，我们称作是“framework包不依赖于idcard包”。

生成实例——方法的三种实现方式

在示例程序中，Factory类的createProduct方法是抽象方法，也就是说需要在子类中实现该方法。

createProduct方法的实现方式一般有以下3种。

·指定其为抽象方法

指定其为抽象方法。一旦将createProduct指定为抽象方法后，子类就必须实现该方法。

如果子类不实现该方法，编译器将会报告编译错误。这也是示例程序所采用的方式。

abstract class Factory{

public abstract Product createProduct（String name）；

……

}

◆为其实现默认处理

为其实现默认处理。实现默认处理后，如果子类没有实现该方法，将进行默认处理。

class Factory{

public Product createProduct（String name）{

return new Product（name）；

}

}

不过，这时是使用new关键字创建出实例的，因此不能将Product类定义为抽象类。

A·在其中抛出异常

在其中抛出异常的方法。createProduct方法的默认处理为抛出异常，这样一来，如果未在子类中实现该方法，程序就会在运行时出错（报错，告知开发人员没有实现createProduct方法）。

class Factory{

public Product createProduct（String name）{throw new FactoryMethodRuntimeException（）；

}

…

}

不过，需要另外编写FactoryMethodRuntimeException异常类。

使用模式与开发人员之间的沟通

不论是我们在第3章中学习的Template Method模式还是本章中学习的Factory Method模式，在实际工作中使用时，都会让我们感觉到比较困难。这是因为，如果仅阅读一个类的代码，是很难理解这个类的行为的。必须要理解父类中所定义的处理的框架和它里面所使用的抽象方法，然后阅读代码，了解这些抽象方法在子类中的实现才行。

通常，使用设计模式设计类时，必须要向维护这些类的开发人员正确地传达设计这些设计模式的意图。否则，维护人员在修改设计时可能会违背设计者最初的意图。

这时，我们建议在程序注释中和开发文档中记录所使用的设计模式的名称和意图。

|4.5相关的设计模式

◆Template Method模式（第3章）

Factory Method 模式是Template Method的典型应用。在示例程序中，create方法就是模板方法。

◆Singleton模式（第5章）在多数情况下我们都可以将Singleton模式用于扮演Creator角色（或是ConcreteCreator角色）的类。这是因为在程序中没有必要存在多个Creator角色（或是ConcreteCreator角色）的实例。不过在示例程序中，我们并没有使用Singleton模式。

◆Composite模式（第11章）有时可以将Composite 模式用于Product角色（或是ConcreteProduct角色）。

◆lterator模式（第l章）有时，在lterator模式中使用iterator方法生成Iterator的实例时会使用Factory Method模式。