面试经典问题，谈谈你对OOA,OOD,OOP的理解

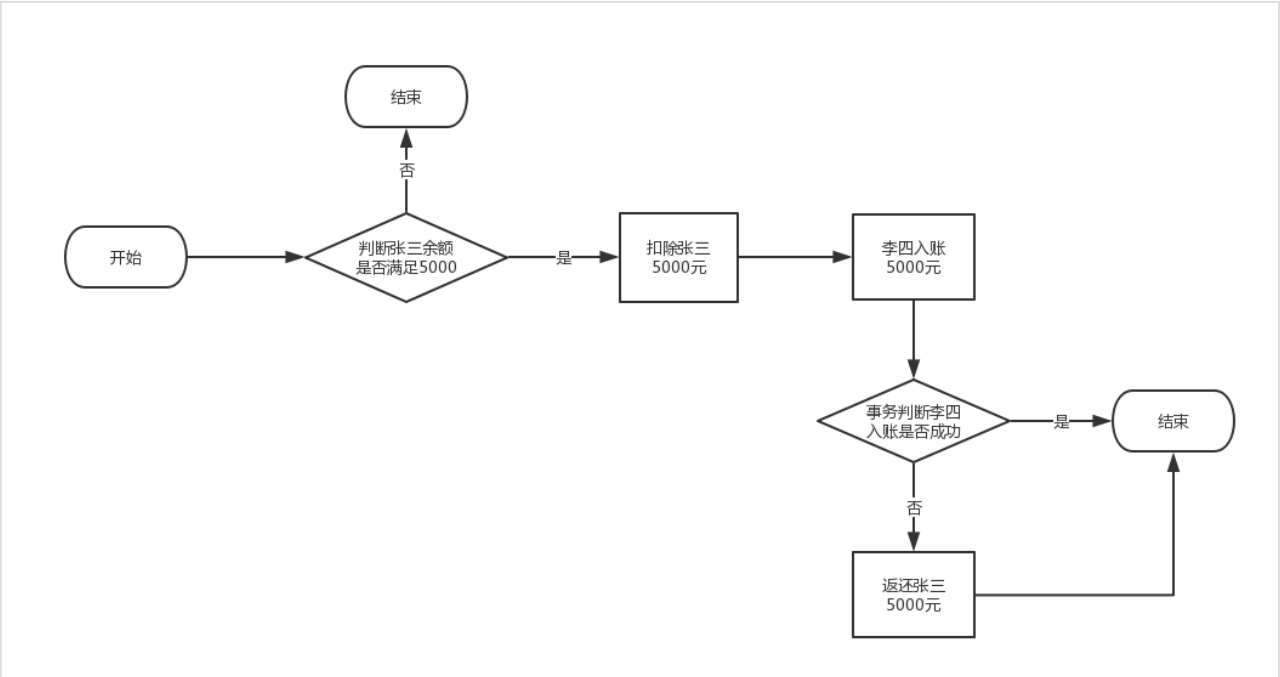
2017-06-19 | OOAD | 阅读量 次

OOA（面向对象分析）、OOD（面向对象设计）、OOP（面向对象编程），这3个概念，对于我们JAVA程序员来讲，或多或少应该都有所了解，或者说至少都听说过。但是要谈到对其理解，可能对于多数入行不深的从业者来说，确实不是那么容易做到。特别是对于绝大多数的3年以内的低中级软件工程师而言。因为他们的工作更多是需要按照项目经理分配的任务来编写功能代码，很少有多余的时间去阅读或者思考一些概念性的东西。说起这个问题，我也在网络上也搜索过很多的资料，大多摘录至书籍，比较官方化。让初学者无从理解。

为了广大的新从业者或者应聘者，在这里，我们以一种实例的方式来对这3个概念进行重新的阐述：

业务场景：

建行卡持有者，张三与李四两人，现在需要张三给李四转账人民币5000元整。按照业务分析后的流程图如下:



对于分析业务流程，常见的2种：面向过程分析（POA），面向对象分析（OOA）

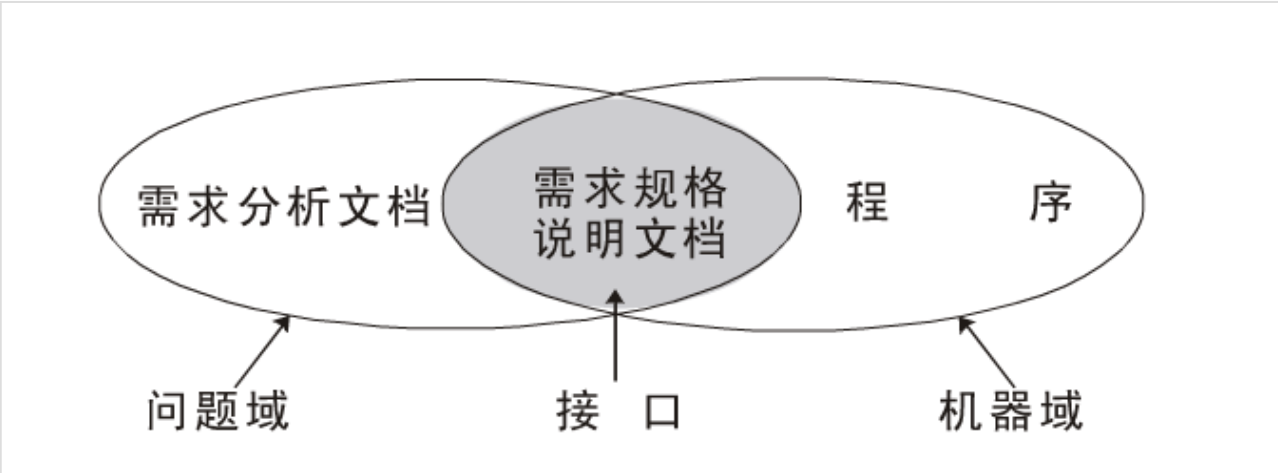
面向过程分析（Procedure Oriented Analysis）：是一种以过程为中心的编程思想，以数据流向为主要导向。为了解决问题，将解决问题的业务过程，按照一定的顺序划分成为一个又一个的事件，然后再封装成一个又一个的函数，最后由一个函数统一的按照顺序一步一步的调用即可。在面向过程分析中，顺序很重要，要实现功能只需要按照一定的顺序相互调用函数即可。

上述业务场景按照面向过程分析出来的结果就是：

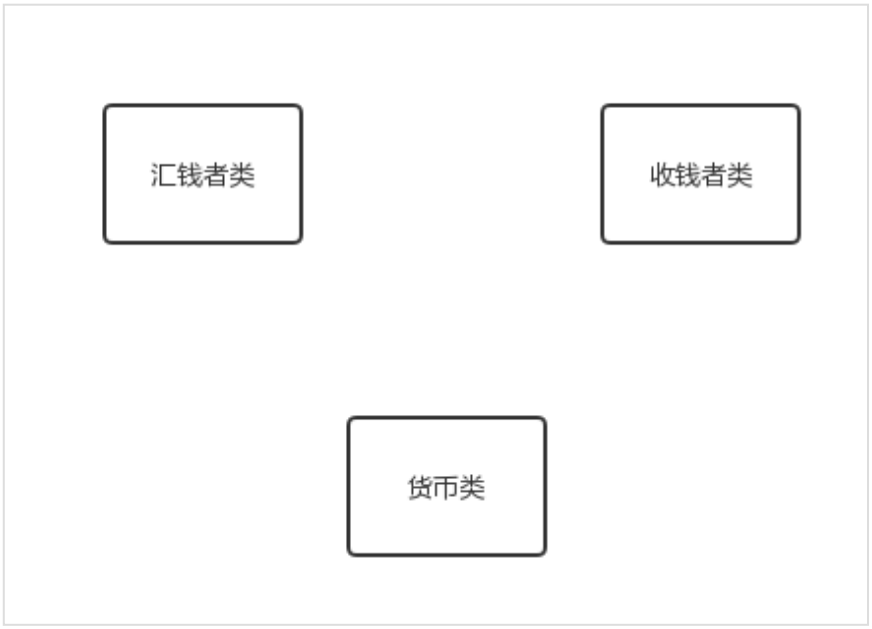
1. 程序检查张三卡中余额是否足够5000元人民币（事件1，满足则调用事件2）
2. 程序从张三卡中扣除5000元人民币（事件2）
3. 程序向李四卡中加入5000元人民币（事件3）
4. 程序检测李四卡中是否正常入账（事件4，满足则结束整个业务）
5. 程序向张三卡中加入5000元人民币（事件5）

在上述过程中，我们将这个业务过程，分成了5个步骤，也叫做5个事件，那么如果需要在程序中完成该转账业务的话，那么我们只需要按照1-2-3-4-5这样的顺序依次调用函数方法即可。在这个过程中，你会发现我们函数调用的顺序一定是不能变化的，变了就出问题了……

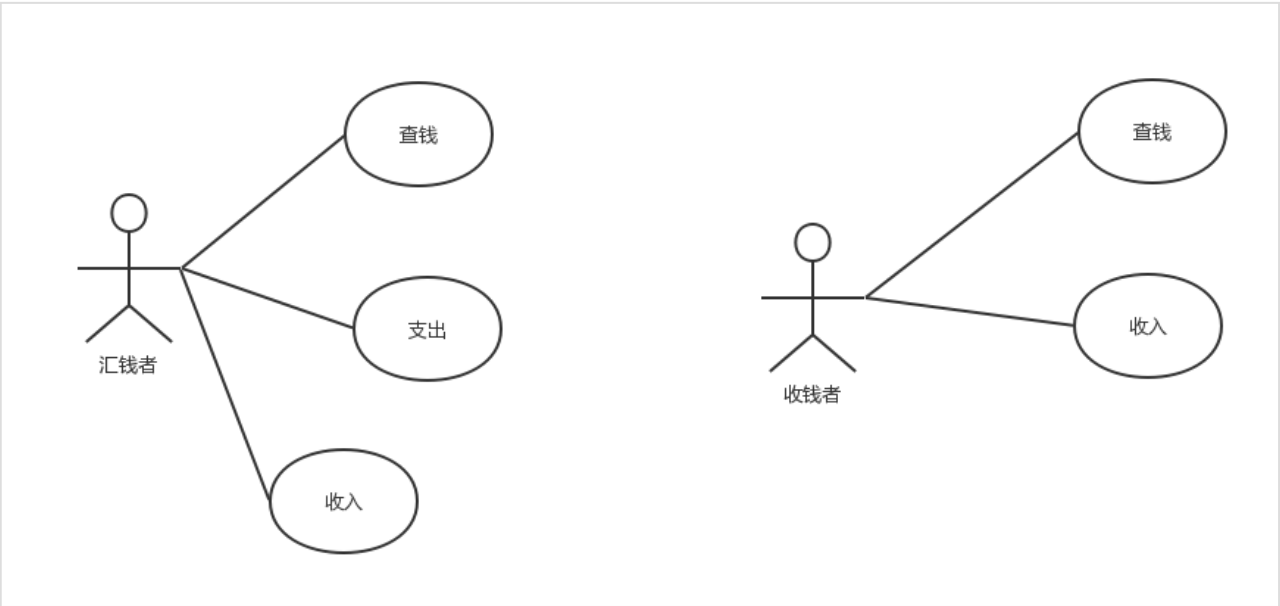
面向对象分析（Object Oriented Analysis）:是一种以对象为中心的编程思想。利用从问题域中的词汇表中找到类与对象。那么说到这里，很多人对问题域又不是很清楚了，问题域：说的简单直接一点，问题域就是客户告诉你的要求，他要干什么。问题域并不特指需求，很多时候，客户所提的需求很多地方都有关联，所以在很多场合，客户所提的需求，还需要需求分析师多角度的帮助客户理清与完善。那么在客户需求范围中，衍生出来的他想要告诉你做的事，就是问题域，包括后期需求发生的变化，同样隶属于问题域的范围。
用一个图来表示：需求分析文档，规格说明文档，以及程序之间的关系：



从上述业务场景的分析中，我们可以抽离出的类与对象有：



抽离出来的类型有：汇钱者类，收钱者类，货币类。而张三只能算是汇钱者类中的一个实例，也被称之为汇钱者对象，李四只能算是一个收钱者类中的一个实例，也被称之为收钱者对象。那么额度为5000的人民币也只是货币类的一个具体实例。通常汇钱者类，收钱者类，货币类，我们都统一称之为：领域模型类，张三，李四，5000元人民币我们都统一称之为：领域对象。作为面向对象分析来说，我们最为重要的就是要分析出领域对象的行为。领域对象的行为主要是为后期的**面向对象设计**（OOD）提供接口依据，而属性作为分析阶段不是我们的重点。



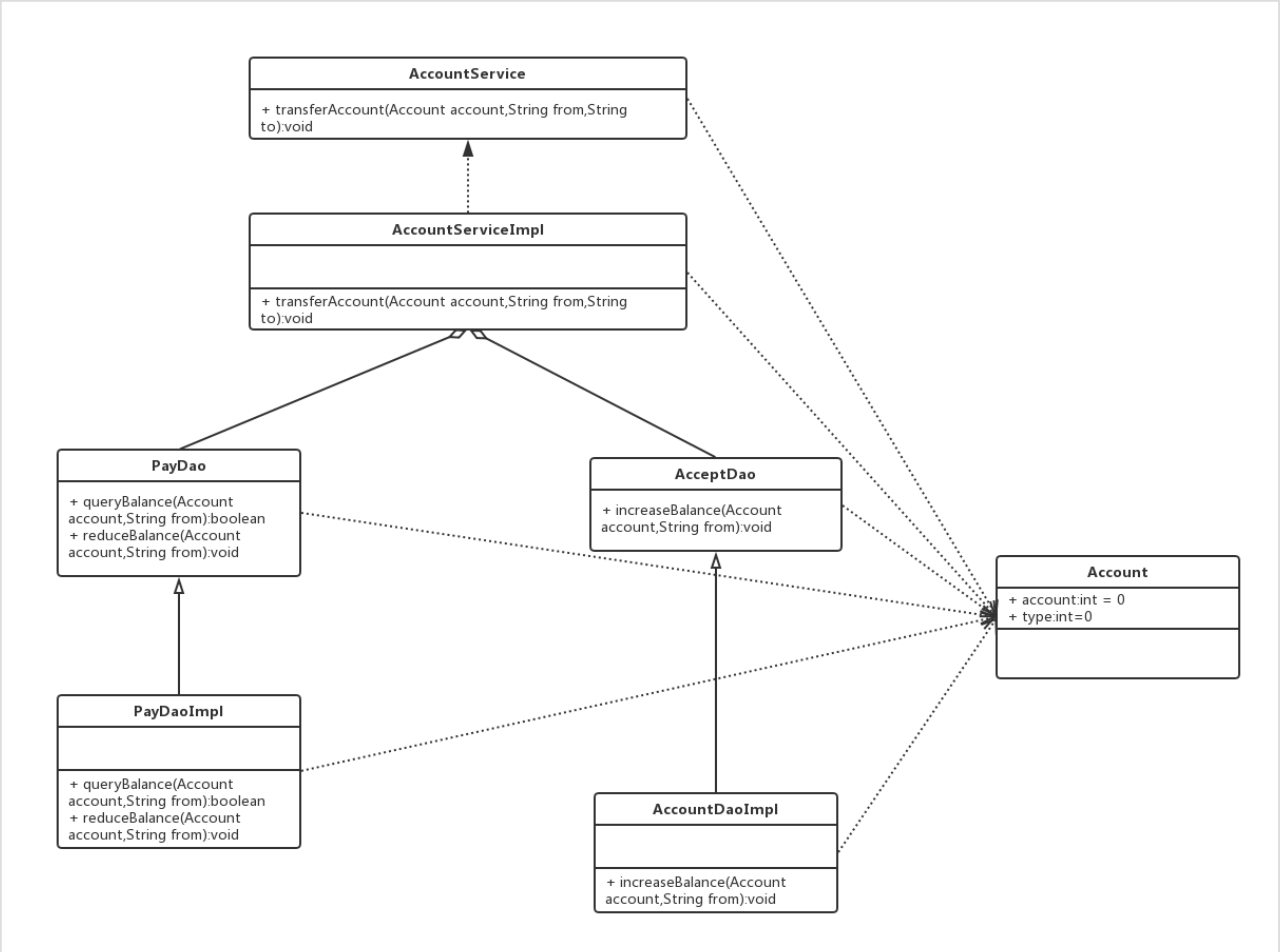
就上述3种领域对象而言，张三这个实例，可以查询自己卡中余额，可以从余额中取出5000元，也可以将外来的钱加入到自己的账户中。

李四这个实例，可以查询自己卡中余额，可以将外来的钱加入到自己的账户中。而5000元人民币只是数据的传输携带者，没有任何行为可言。

在JAVAEE项目中，组织业务逻辑的方式：事务脚本（面向过程设计 POD）、领域模型（面向对象设计 OOD）

事务脚本（面向过程设计 POD）： 它是纯面向过程的一种组织业务逻辑的方式，在JAVAEE项目，是一种非常常见的设计方式，做法很简单，将POA分析后的结果封装成JAVA对应的方法即可，然后在业务层中统一按照顺序调用接口。将领域对象（人民币）去掉行为，只保留属性即可，用于数据传输。以数据流向为主要导向。

按照上述的业务场景来设计：保留失去行为的领域对象（人民币），同时抽离出一个业务层的接口类，多个持久层的数据访问接口类，在业务层接口中提供一个统一的业务调用方法transferAccount(),在数据库发生关系的持久层，提供一系列事件方法：queryBalance(),reduceBalance(),increaseBalance();使用UML类图表示为：



当然还有一系列的表示手段，比如：包图，对象图，序列图，用例图..... 这里就不一一设计了。

领域模型（面向对象设计 OOD）（Object Oriented Design）：主要是正确有效的构造出复杂系统的抽象结构，将**面向对象分析**后的结果，作为面向对象设计的模型，通常用于展示被设计系统的逻辑模型（业务逻辑如何设计），物理模型（系统如何划分，层次如何划分，如何部署.....），静态模型（组成系统的元素：类，接口）和动态模型（对象的调用.....）。将OOA分析的结果作进一步的规范化整理，以便能够被OOP直接接受。

概念可能比较生硬，那么说的直白点，设计阶段的任务就是：

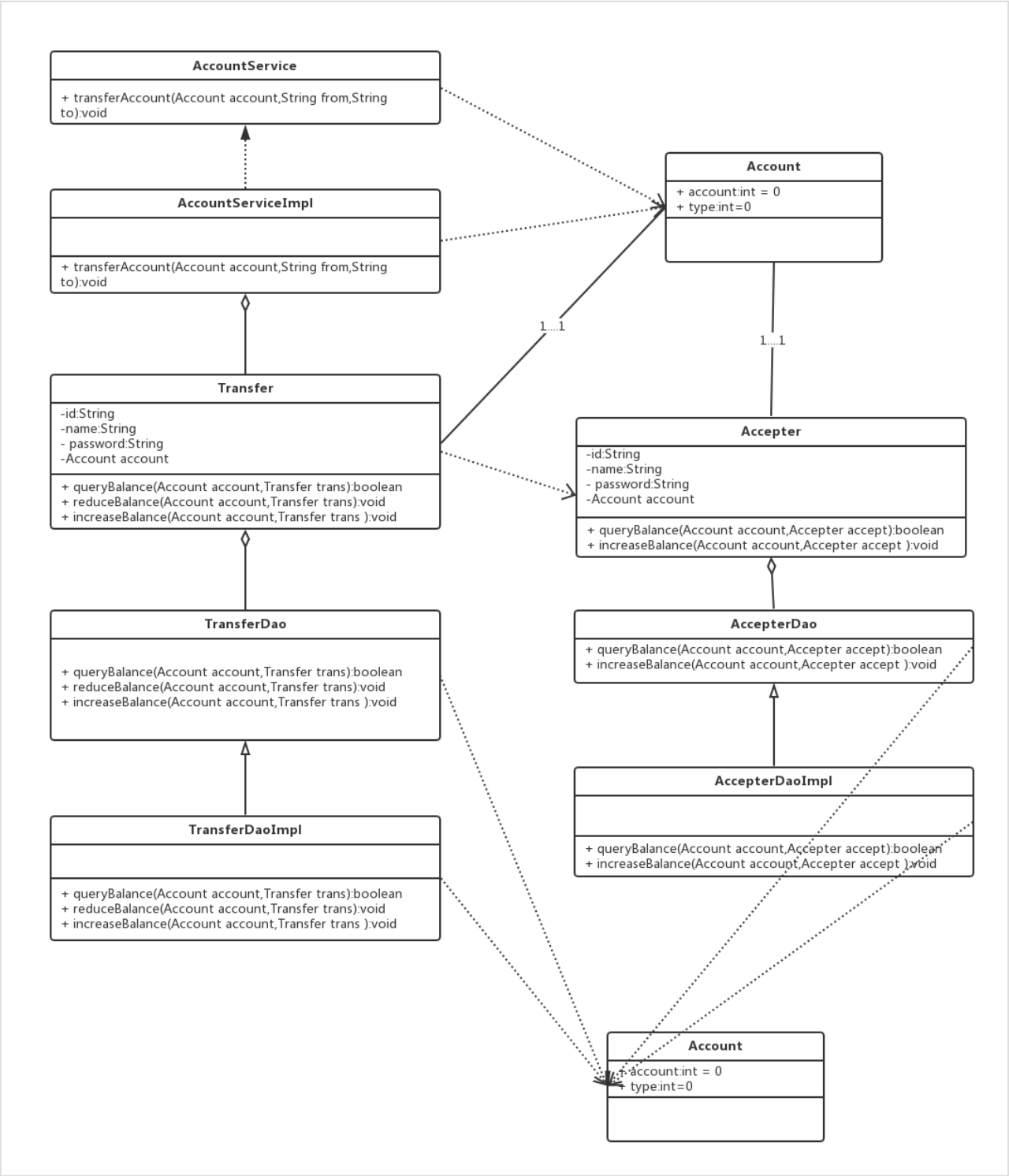
1. 按照面向对象的方式将系统分解成不同的模块化，或者系统再次划分为子系统，直到划分到可设计的最小粒度为止【垂直分块，水平分层】。
2. 使用不同的表示方法来展示被设计系统的逻辑模型（业务逻辑如何设计），物理模型（系统如何划分，层次如何划分，如何部署……），静态模型（组成系统的元素：类，接口）和动态模型（对象的调用……）【常用的表示法手段：UML各种图形】。

需要注意的是：层次结构并非就一定是三层结构，也可以是多层，具体分几层需要根据项目的复杂度来决定，如果没有过多的业务逻辑，只有简单的CRUD的话，那么三层结构足以应付一切。

按照上述的业务场景来设计：对于领域逻辑来说，三层结构就不能满足其业务逻辑复杂度了，可能这时整个系统可能是四层，五层，或者多层……，我们这个例子，暂时定为四层结构：控制层、业务层、领域逻辑层、数据访问层。在设计领域对象时，保留失去行为的领域对象（人民币），转钱对象，收钱对象，同时抽象出一个业务层的业务接口用于简单的调度业务逻辑，多个持久层的数据访问接口类。

在OOA阶段，我们分析出张三这个实例，可以查询自己卡中余额，可以从余额中取出5000元，也可以将外来的钱加入到自己的账户中。那么它就应该对应着对应的3种行为方法。
李四这个实例，可以查询自己卡中余额，可以将外来的钱加入到自己的账户中，那么它就应该对应着对应的2种行为方法。而5000元人民币只是数据的传输携带者，没有任何行为可言

如果采用UML中类图的表示法来看，图形应该如下：



（在同一个图中，相同名称的元素代表同一元素）

在这张图中，我们的层次就是4层结构，甚至我们还可以再分为5层，6层等，所以一般领域模型适用于业务需求较为复杂的情况。业务层仅仅是为了梳理业务流程，而真正的业务逻辑则交由领域层的领域对象之间相互调用对方的方法来实现业务逻辑。比如要转钱，就需要转钱者领域对象，减少自己的余额后，调用收钱者领域对象，添加自己的余额。当然真正的银行转钱复杂度，绝对远高于这个场景。 还需要使用到事件驱动模型，以及消息队列等一系列的技术，当然具体情况具体分析。

面向对象编程（OOP）

面向对象编程（Object Oriented programming）：将面向对象设计后的抽象，采用面向对象的方式来实现的方法，就叫做面向对象编程。在这种方法中，程序被组织成许多相互协作的对象，每个对象都是一个类的实例。利用对象构成业务逻辑的组成基本元素（组合模式的层次结构），而不是采用算法。组成程序的基本元素是一个又一个的基础组件（类），那么组件就是我们层次结构所对应的接口或接口的实现类，那么程序就是由一个又一个实现类的实例相互协作来完成业务逻辑的实现。

总体来说： 面向对象分析的结果可以作为面向对象设计的模型，而面向对象设计的结果则可以作为面向对象编程的蓝图，应用程序需要由编程才能实现。

‹ 你应该知道的Java创建对象的四种方式　　异常处理概述 ›