**案例编号**：UM001

**案例名称**：关于账户信息的类间泛化关系构建

**案例目标**：

通过此案例，让学生了解如何在建立类图时确定类与类之间的泛化关系

**案例描述**：

泛化关系是一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）的功能，并可以增加它自己的新功能的能力，继承是类与类或者接口与接口之间最常见的关系。本案例通过账户这一最基本的父类，一步一步把银行账户/信用卡账户/支票账户/储蓄账户这四个子类划分成不同的类别，进而识别出继承关系，并根据UML类图语法规则画出来。

**Keyword**：UML、类、泛化、父类、子类

**系统描述**：

该案例所属系统为账户信息管理系统。该系统为一个B/S结构的用户账户信息管理系统，具体包括：用户自服务管理和后台信息管理。 用户自服务管理包括登陆验证（即用户输入正确的用户名和密码后方可登陆系统），个人信息修改（其中不同的用户拥有的修改权限不一样）；后台信息管理功能包括对用户账户信息的增删改查操作并提供对指定信息的查询或模糊查询以及系统的相关维护。

其中，用户账户的信息包括：用户头像，用户ID（即员工的编号），用户姓名，身份证号，户籍所在地（包括省市，若户籍在国外则记录国家名称），出生日期，个人电话，住址，账户类型，账户编号，账户余额。其中，若为信用卡账户，还包括欠款金额，还款日期等信息。

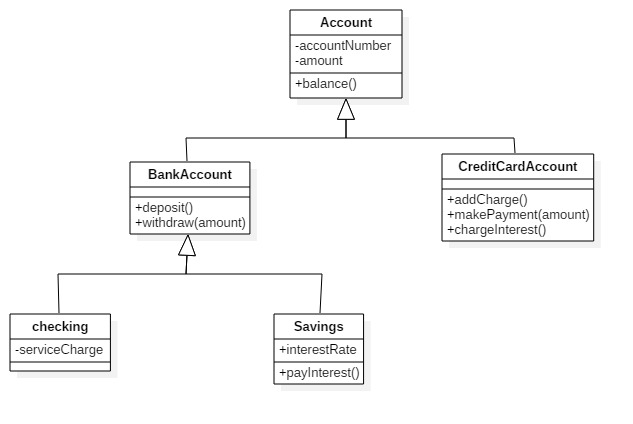
系统可通过个人电话的方式帮助用户找回密码，但前提是必须回答自己设置的密码找回问题。

**案例建模过程**：

1. 梳理关于账户信息领域的术语，了解账户有哪些类型。假设在一段文字中识别出账户的类型有：银行账户/信用卡账户/支票账户/储蓄账户
2. 确定类图中类的方法和属性。从字面上找到“类”以后，我们需要对所有的类进行分析，找到可以剔除的类、可以抽象成更高层次的类等。如果某些类可以作为另一个类的方法或者属性出现，而项目本身又不关注，那么可以剔除这些类，将其放在另一个类的方法或者属性中。如果由多个类有相同的功能、属性、方法等，那么就要考虑是否要对这些类抽象出一个父类。
3. 在这个案例中找到的共有四个类：银行账户、信用卡账户、支票账户、储蓄账户。首先，因为我们关注的就是账户的关系，所以不能提出这四个类。其次，由于四个账户类型同属于“账户”这个概念，对于所有的账户，都有账户编码和存款，也有查询余额方法，所以很容易地就抽象出了一个父类——账户，其属性和方法就被确定下来。最后，确定每个子类的属性和方法，针对不同的账户，可以有不同的方法和属性，要注意的是，父类拥有的属性和方法在子类中不应该再出现。
4. 梳理这四个账户类型之间的关系。由领域知识分析，我们知道支票和储蓄账户都属于银行账户，这样，支票和储蓄账户都继承自银行账户，信用卡账户与银行账户在同一个层次上，都继承自“账户”这一父类。
5. 由UML类图中泛化关系的画法为空心三角，把类图中的泛化关系画出来。

**案例结果**：

正解：



**案例总结**：

一般而言，识别具体类之间的泛化关系比较简单和明显，但在建模过程中，对多个类的父类进行抽象却比较困难，在本案例中，将四个账户类型统一抽象成父类“账户”，不仅将账户的层次关系体现的更为明显，也将不同账户类型联系在一起，建模的抽象思维体现的更加充分。从本案例，我们需要掌握以下知识点：

1. 泛化关系(Generalization)也就是继承关系，也称为“is-a-kind-of”关系，泛化关系用于描述父类与子类之间的关系，父类又称作基类或超类，子类又称作派生类。在UML中，泛化关系用带空心三角形的直线来表示。
2. 在代码实现时，使用面向对象的继承机制来实现泛化关系，如在Java语言中使用extends关键字、在C++/C#中使用冒号“：”来实现。
3. 在UML当中，对泛化关系有三个要求：
4. 子类与父类应该完全一致，父类所具有的属性、操作，子类应该都有；
5. 子类中除了与父类一致的信息以外，还包括额外的信息；
6. 可以使用父类的实例的地方，也可以使用子类的实例；

中英文对照词典：

账户：Account

银行账户：BankAccount

信用卡账户：CreditCardAccount

支票账户：Checking

储蓄账户：Saving