**变量多态性与方法多态性**

**一、变量多态性**

首先看下面的代码：

 观察静态main方法，其中members数组是一个FacultyMember[]类型的变量，即是存储FacultyMember对象引用的数组。再观察8行的randMember方法，其返回一个以FacultyMember为基类的对象引用，并存入members数组。

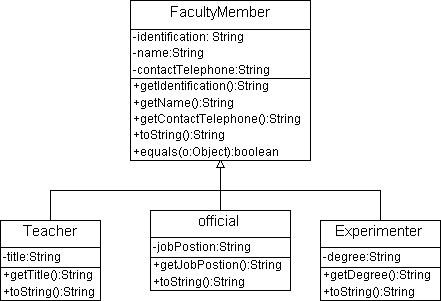
对比我们熟悉的C++。虽然我们不能直接这样进行存储子类对象，但可以存储一个以FacultyMember为基类的对象的指针于FacultyMember\*数组中。这样的操作体现了C++的多态性。

同样的原理，Java直接支持这样的操作，我们可以直接利用FacultyMember[]数组实现多态性，这方便了我们的编码过程。

在Java中，对象变量是多态的。这意味着基类的对象可以引用一个本类的对象，也可以引用任何子类的对象。

**二、方法多态性**

首先观察以下UML类图：



我们注意到子类继承了FacultyMember类，并且重写了toString()方法。那么一个向上转型的对象调用toString()方法会如何？通过运行代码，我们发现这种对象仍然调用了子类的toString()方法，并非基类的。

方法多态性是指Java虚拟机根据调用该方法的实例的类型来决定选择调用哪个方法。方法多态性的原理在于「动态绑定」。

首先了解一下Java中方法的「静态绑定」。如果对象调用的方法为private、static或final修饰的，或者这个方法是该类的构造器，编译器将明确获知对象调用的是哪个方法。于是编译器将这个唯一的方法与代码中调用的方法匹配，在运行时虚拟机直接调用该方法。这个过程被叫做「静态绑定」。

其次「动态绑定」是指子类重写了基类的某个方法（注意重写不等同与重载），虚拟机在运行时动态地调用这个方法。如B类继承A类，并重写了toString()方法，而B类对象向上转型，被A类对象变量引用，最终这个对象调用toString()方法，那么虚拟机在运行时会直接调用B类方法而非A类。

其原理是Java虚拟机出于效率问题，预先为每个类创建一个「方法表」。其中列出每个方法的签名（ 方法名字和参数列表构成方法签名，注意返回值类型并不构成签名）和实际调用的方法。当对象调用方法时，虚拟机查找方法表，调用距子类最近的类的方法。同样的，若调用隐式参数的方法super.f(param)，那么将查找基类的方法表。