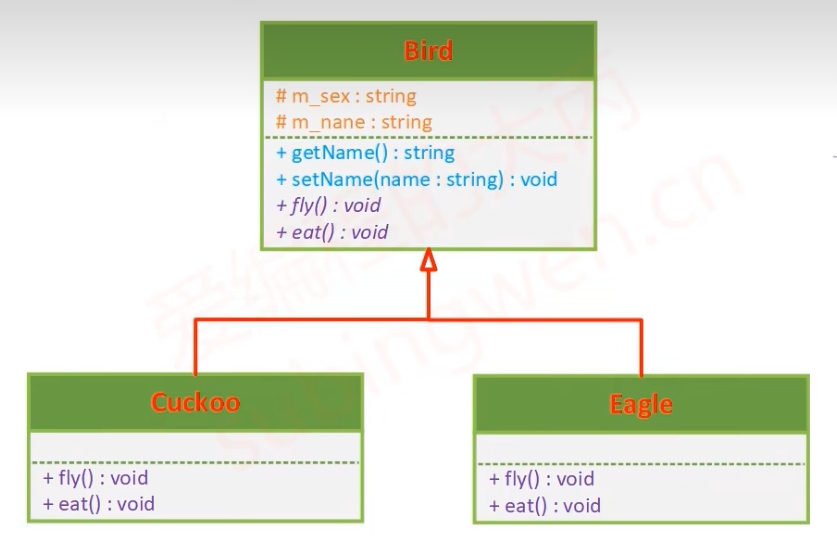
接下来简单看一下类间的5种关系。





每个类框的创建遵守之前的规则。而对于子类，我们可以看到这里只写出了子类重写的哪些函数(假设这些个子类没有额外的对象)。也就是说，对于子类来说，不需要对父类中的对象再进行描述，使用空心三角箭头就能表示继承关系并使得对应父类的成员不需要再被描述。

再来详细分析下，对于空心三角箭头，俩端分别连接父类和子类。其中，箭头指向的是父类，另一端连接的是子类。

接着来看到第二种类间关系，**关联**关系。

这种关系其实很常见，就是一个类中引用了一个类。这个类可能是别的类，也可能还是自身的类，依照引用关系和对象的不同存在多种关系。

1~单向关联关系

这种关系就是一个类中包括了另一个类(需要是一个与本类不同的类)，并且被包含的那个类不知道包含它的类的信息。父子或母子之类的关系的类建模

使用带箭头的实线进行连接

2~双向关联关系

这种关系就是一个类中包括了另一个类(需要是一个与本类不同的类)，被包含的类中也包含了包含它的类。学生与老师的类建模抽象。

使用不带箭头的实线进行连接

3~自关联关系

这种关系就是一个类中包含了指向本身同一种类的引用。就比如树的节点类。

使用指向自身的带箭头线进行连接。

当然，总总这些绘图线段都不是一定的，取决于使用的绘图软件。

接下来看下一种关系 **聚合**关系

这个关系乍看起来有点抽象。其实可以通过教室和学生，图书馆和藏书间的关系辅助理解。一个教室中可能会有多个学生，图书馆中也包含着系列藏书。这俩者之间构成了一个聚合关系。

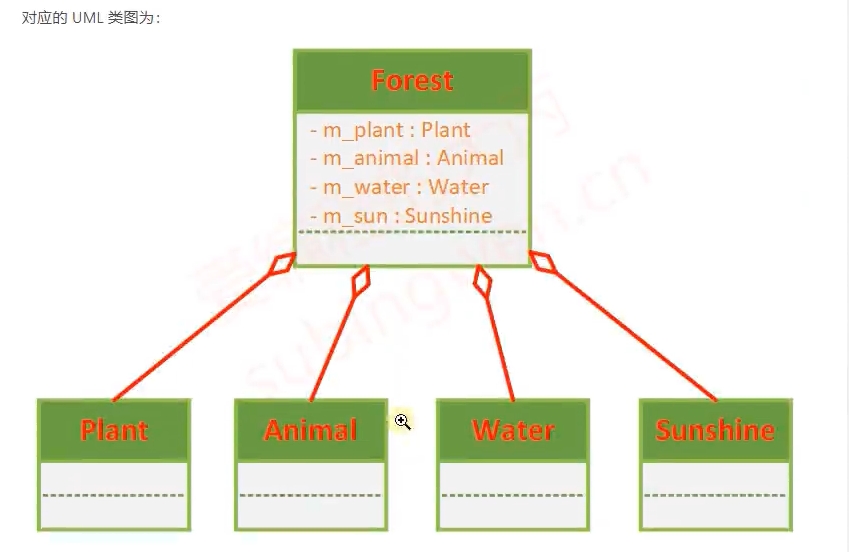
对于聚合关系，就比如教室和学生这种。我们需要理清下这俩者之间的关系。无论教室还是学生，都是可以独立存在的，但是吧，教室中没有学生这些对象的话，其实是没有存在的意义的。对于独立存在的学生，不同于教室，他们独立存在是有着自己的意义的。从某种意义上这个也可以成为识别聚合关系的一种方法。 同时，这些聚合关系中，拥有的对象通常不只有一个。就比如教室中的学生，数量通常都是多个。在这种架构下，我们去取出其中的一个学生对于教室这个整体其实是没有很大影响的，当然这得是在一种至少不是很少的情况下的。你如果要说只有几个学生的这种情况我也无话可说。

以一种C++的类来看吧，如果一个类中包含了多个别的类的数组，当我们对这个主类进行析构的时候，这些个包含的子类并没有被析构的话，可以看做这些类之间是一种聚合的关系。

在这里我其实对聚合关系和关联关系有点疑惑。不过可以这么理解，类间关系是不一定存在一种的。或者说，类间关系是由类的属性而产生的，而不是由类间关系而产生的类。聚合关系某种程度上可以看做是关联关系的子类。聚合关系一定是关联关系，但是关联关系不一定是聚合关系。

如果类间是一个聚合关系，那么这些类之间应该使用空心菱形的直线进行连接。菱形与聚合的整体相连，另一端与对应的部分相连。

举出一个聚合类型的例子



接下来看另一种关系 **组合**关系

使用实心菱形线段进行标识组合关系。与上相似故略过

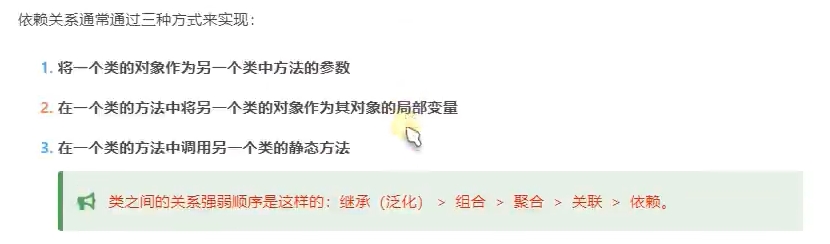
组合关系也是一种整体和部分的关系。很多方面它跟聚合关系都很相似。但是这里相对于聚合关系不同的是，这里的部分其实是依赖于整体而存活的，当主对象被析构时，这个组合关系的部分类不应该还存活着。举个例子，就比如头和嘴巴，眼睛，眉毛等。后面的部分在整体消失时不应该还存在(你杠就是你对)。

这样看来，通过组合关系联系的各个类相对于聚合关系有着更为紧密的联系。以C++的视角来看的话，就是当组合关系中的整体类被析构的时候，它对应的部分类也应该被析构掉。

接下来来看类与类之间最弱的一种关系 **依赖**关系

依赖关系的判断有点抽象。可以用排除法来进行。如果一个类既不是继承.关联.聚合.组合关系，那么它就是一个依赖关系。你可以看出这个关系有多弱了吧。

当然，还是有不通过排除法来进行判断的方法的



当我们一个类使用了另外一个类作为本类的函数参数。在函数参数内使用了别的类作为本类函数的一个局部变量，一个类中调用了别的类的静态方法。这三种情况下，这些个类之间构成了依赖关系。当然，其实这个相较于排除法其实更难判断了。

依赖关系使用带线头的虚线进行标识，由依赖的一方指向被依赖的一方。



对于类图关系，类之间是可能存在多种关系的，但是我们不需要将全部关系都列出来，我们只需要将这些类关系中最强的关系描述出来就行了。