## 面向对象的学习感悟

----AID1902 太原赵杨杨

封装顾名思义：就是封闭包装，就是将同种事物的多种信息及其行为聚集合成一个整体的数据块，外界能看到和使用的只是有限的外表。

对于行为来讲：

可以为外界提供多种必要的功能，而不显露实现功能的细节。类似于相机，有精美的外表，可以实现拍照的功能，但是相机里面究竟是如何实现拍照的，外界不得而知也不需要知道，同样，内部的实现细节的变化不会改变外部访问原则，体现了高内聚，低耦合的特征。而且在外部多个对象访问下，不会针对每个对象提供服务，而是同时进行服务。

对于设计来讲：

1、为了实现一个大的功能及其需要进行的繁杂的工作，必须进行分工以提高开发效率，封装实现了不同功能可以分开进行，极大提高了工作的效率。

2、封装可以实现将不同种类的事物分别包装，各自实现自己的功能，降低彼此之间因变化引起的影响。当然，对于变化性大的事物，为了避免其因变化而影响其他类，可对其进行单独的封装。

学生管理系统就很好的体现了这一点：

1、要实现学生管理系统的运行，需要学生的信息，包括姓名，年龄等等

2、而显示系统则需要这些数据的逻辑结果，实现增、删、改、显示、排序等等众多功能

3、而为了实现这些功能，还必须对学生信息进行逻辑处理，然后才能得到功能想要的结果。

基于此，将所有学生分为一大类，实现学生信息逻辑处理的方法函数等等归为一大类，而负责显示逻辑处理结果的功能归为一大类，这样一来视线就变得清晰很多，不必为了每个学生的信息单独进行处理，避免了很多的工作量。

class StudentModel:

class StudentManagerController:

class StudentManagerView:

当然封装也有禁忌：

1、避免过度封装，学生在管理系统中信息的种类全部是一样的，区别只在于每个学生信息的内容不同，符合类的要求和封装的思想，没必要分为男生、女生等等类别。像逻辑处理类中的各种功能，比如增加学生、删除学生等等，都是对学生信息进行批量处理，进行相同的操作，如果全都封装起来单独调用，只是徒增工作量，且不便于统一管理，违背了封装高内聚的原则。

2、避免封装过度，将功能集中在一起。例如游戏中，敌人和玩家，都有血量、攻击力等属性和攻击、掉血、死亡等功能，但是不能放在一起统一管理，因为玩家的功能显示和敌人不完全一样，敌人死了游戏可以继续，但是玩家死了游戏就结束，功能有所区别，实现起来就不能统一调用。

## 继承、多态

继承定义：

重用现有类的功能和概念，并在此基础上进行扩展。

继承特点和注意事项：

继承往往用在需要对具有相同的属性和行为的类进行统一管理的时候，父类大多是从子类抽象而来的、一个大的概念，比较稳定，而子类较父类而言要具体，容易变化。比如交通工具作为父类时，汽车、飞机等作为子类，人在使用交通工具时，交通工具可能是汽车，可能是飞机等等。子类拥有的所有属性功能，可以任意使用。但是并不是任何情况下，需要使用一个类的功能时就要使用继承，过度的继承会导致上层的类变化引起下层类的剧烈波动，这时候适当使用关联或依赖降低耦合度，可以起到很好地效果。而且，因为父类的稳定性、不易变化的特征，继承往往在隔离变化时被常用。在要求的功能中如果有太多的变化点，而这些变化点又有相同的属性时，使用继承将这些变化点统一管理在一个父类中，统一管理，避免对其他数据造成影响，对实现开闭原则有很好的推动作用。

多态：

多态与继承息息相关。子类在继承了父类的属性、方法后，如果一成不变的使用下去，这样的继承仅仅是代码上的复用，就出现了所谓的“坏的味道”。只有当子类实现多态，当调用父类方法时，子类在同一方法上有不同的表现，才能实现功能的可扩展性，这样的功能也就比较灵活。

## 六大原则

一、单一原则

对于单一职责原则，其核心思想为：一个类，最好只做一件事，只有一个引起它的变化因素。单一职责原则可以看做是低耦合、高内聚在面向对象原则上的引申，高内聚，就是指将类高度聚合，缩小，一个类中只完成一个任务。职责过多，可能引起它变化的原因就越多，这将导致职责依赖，相互之间就产生影响，从而大大损伤其内聚性和耦合度。通常意义下的单一职责，就是指只有一种单一功能，不要为类实现过多的功能点，以保证实体只有一个引起它变化的原因。 专注，是一个人优良的品质；同样的，单一也是一个类的优良设计。交杂不清的职责将使得代码看起来特别别扭牵一发而动全身，有失美感和必然导致丑陋的系统错误风险。

## 二、开放封闭原则

对于开放封闭原则，它是面向对象所有原则的核心，编程追求的目标就是最大化的实现开放封闭原则。 开放封闭原则，其核心思想是：代码应该是可扩展的，而不可修改的。也就是说，对增加新的功能开放，对修改封闭的。 因此，开放封闭原则主要体现在两个方面：1、对扩展开放，意味着有新的需求或变化时，可以对现有代码进行扩展，以适应新的情况。2、对修改封闭，并不是指这段代码写完了就不能动了，而是指在原有代码功能的基础上，增删功能时，不用修改其他的代码。 实现开开放封闭原则的核心思想就是对抽象编程，而不对具体编程，因为抽象相对稳定。让类依赖于固定的抽象，所以修改就是封闭的；通过多态的实现方法来改变继承的固有属性方法，实现新的拓展方法，所以就是开放的。 “需求总是变化”没有不变的软件，所以就需要用封闭开放原则来封闭变化满足需求，同时还能保持软件内部的封装体系稳定，不被需求的变化影响。

## 三、依赖倒置原则

对于依赖倒置原则，其核心思想是：依赖于抽象。具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象。 我们知道，依赖一定会存在于类与类、模块与模块之间。当两个模块之间存在紧密的耦合关系时，最好的方法就是使用继承定义一个抽象的概念，使得高层的类调用抽象类，而底层类实现抽象的具象化，以此来有效控制耦合关系，达到依赖于抽象的设计目标。 抽象的稳定性决定了系统的稳定性，因为抽象是不变的，依赖于抽象是面向对象设计的精髓，也是依赖倒置原则的核心。 依赖于抽象是一个通用的原则，而某些时候依赖于细节则是在所难免的，必须权衡在抽象和具体之间的取舍，方法不是一层不变的。

## 四、组合复用原则

组合复用，其核心思想是：使用关联关系，提取想要的功能或属性。 具体而言，组合复用原则体现在：一个类对另外一个类的关联应该建立在最需要的方法上，不要强迫依赖不用的方法，抽象类类并非需要所有的定义，在设计上这是“浪费”，而且在实施上这会带来潜在的问题，对胖接口的修改将导致一连串的客户端程序需要修改，有时候这是一种灾难因为这样只会使得耦合剧增。关联有效的将细节和抽象隔离，体现了对抽象编程的一切好处。在这种情况下，将被关联类分解为多个特点的定制化方法，使得关联类仅仅依赖于它们的实际调用的方法，从而解除了关联类不会依赖于它们不用的方法。 分离的手段主要有以下两种：1、委托分离，例如银行类和人物类，人物类拥有姓名、年龄等等属性，当人在银行取钱时，银行只需要客户提供姓名和取钱数量，并不需要其他的信息，这时候就体现出组合复用的优势。2、多重继承分离，通过方法的多继承来实现客户的需求，这种方式是较好的。

## 五、里氏替换原则

对于里氏替换原则，其核心思想是：子类必须能够替换其基类。这一思想体现为对继承机制的约束规范，只有子类能够替换基类时，才能保证系统在运行期内识别子类，这是保证继承复用的基础。在父类和子类的具体行为中，必须严格把握继承层次中的关系和特征，将基类替换为子类，程序的行为不会发生任何变化。同时，这一约束反过来则是不成立的，子类可以替换基类，但是基类不一定能替换子类。 里氏替换原则，主要着眼于对抽象和多态建立在继承的基础上，因此只有遵循了里氏替换原则，才能保证继承复用是可靠地。实现的方法是面向接口编程：将公共部分抽象为基类接口或抽象类，通过Extract Abstract Class，在子类中通过覆写父类的方法实现新的方式支持同样的职责。 里氏替换原则是关于继承机制的设计原则，违反了里氏替换原则就必然导致违反开放封闭原则。 里氏替换原则能够保证系统具有良好的拓展性，同时实现基于多态的抽象机制，能够减少代码冗余，避免运行期的类型判别。

## 六、迪米特法则

迪米特法则又叫最少知道原则，最早是在1987年由美国Northeastern University的Ian Holland提出。

通俗的来讲，就是一个类对自己依赖的类知道的越少越好。对于被依赖的类来说，无论逻辑多么复杂，都尽量地的将逻辑封装在类的内部，对外除了提供的public方法，不对外泄漏任何信息。

迪米特法则的初衷是降低类之间的耦合，由于每个类都减少了不必要的依赖，因此的确可以降低耦合关系。但是凡事都有度，虽然可以避免与非直接的类通信，但是要通信，必然会通过一个“中介”来发生联系，过分的使用迪米特原则，会产生大量这样的中介和传递类，导致系统复杂度变大。所以在采用迪米特法则时要反复权衡，既做到结构清晰，又要高内聚低耦合。

---------------------

作者：ttwzyy

来源：CSDN

原文：https://blog.csdn.net/ttwzyy/article/details/88648886

版权声明：本文为博主原创文章，转载请附上博文链接！