面向对象课程答辩

面向对象是一种编程思想，它以人类对现实世界的观察为基础，发展出了几个关键的概念：多态、封装、继承。大部分时间，我们可以以面向过程作为指导思想编写程序，但是当我们所编写的软件系统变得庞大和复杂的时候，面向对象就更加合适。使用面向对象的方式来编写程序时，我们经常会定义一个一个的类，并且放在不同的文件中，然后管理这些类之间的关系。

面向对象主要思想是:识别对象，分配职责，建立交互, 封装变化, 隔离变化, 执行变化

面向对象的主要特征为：封装，继承，多态

**一、封装:**

1.定义

从数据角度: 将一些基本数据类型复合成一个自定义类型。

从行为角度: 向类外提供功能，隐藏实现的细节。

从设计角度: 分而治之，封装变化，高内聚，低耦合

(1) 分而治之

-- 将一个大的需求分解为许多类，每个类处理一个独立的功能。

-- 拆分好处：便于分工，便于复用，可扩展性强。

例如：

雕版印刷术与活字印刷术对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 雕版印刷术 | 活字印刷术 | | | | |
| “字” | “字” | “字” | “字” | |
| “字” | “字” | “字” | “字” | |
| “字”“字”“字”“字 ” “字”“字”“字”“字 ” “字”“字”“字”“字 ” “ 字 ” “ 字 ”“ 字”“字 ” | “字” | “字” | “字” | “字” | |
| “字” | “字” | “字” | “字” | |
| “字” | “字” | “字” | | “字” |
| 一个字错，整个雕版作废 | 一个字错，替换错误字 | | | | |

(2) 封装变化

-- 变化的地方独立封装，避免影响其他类。

(3) 高 内 聚

-- 类中各个方法都在完成一项任务(单一职责的类)。

(4) 低 耦 合

-- 类与类的关联性与依赖度要低(每个类独立)，让一个类的改变，尽少影响其他类。

例如：

硬件高度集成化，又要可插拔

最高的内聚莫过于类中仅包含1个方法，将会导致高内聚高耦合。

最低的耦合莫过于类中包含所有方法，将会导致低耦合低内聚。

**二、继承:**

1.定义 :

重用现有类的功能与概念，并在此基础上进行扩展。

2.作用:

隔离客户端代码与实现方式(隔离用与做)

例如：

交通工具隔离了人与飞机/火车...的变化

class Vehicle:

"""

交通工具

"""

def transport(self, str\_pos):

# 不确定具体操作，人为制造错误

raise NotImplementedError()

# 提醒子类必须使用

class Car(Vehicle):

def transport(self, str\_pos):

print("行驶到", str\_pos)

class Airplane(Vehicle):

def transport(self, str\_pos):

print("飞到", str\_pos)

图形隔离了图形管理器与圆形/矩形...的变化

class Graphic:

"""

图形管理器

"""

def get\_area(self):

pass

class Circle(Graphic):

"""

圆形

"""

def \_\_init\_\_(self,radius):

self.radius = radius

def get\_area(self):

return self.radius \*\* 2 \* 3.14

class Rectangle(Graphic):

"""

矩形

"""

def \_\_init\_\_(self, length,width):

self.length = length

self.width = width

def get\_area(self):

return self.length \* self.width

公司的组织架构：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 董事长 | | | | | | | | | |
| 行政中心 | | | 营销中心 | | | 技术中心 | | | |
| 人力资源 | 行政部 | 后勤部 | 销售部 | 策划部 | 渠道部 | 研发部 | 检测部 | 产品部 | 售后部 |

3.优点：

一种代码复用的方式，以层次化的方式管理类。

4.缺点：

耦合度高

5.作用:

隔离客户端代码与功能的实现方式。

6.适用性:

多个类在概念上是一致的，且需要进行统一的处理。

**三、多态:**

1.定义:调用父一个方法,执行子类方法,不同实现方式不一样,所以表现形态就不一样.

2.作用: 继承将相关概念的共性进行抽象，多态在共性的基础上，体现类型的个性化（一 个行为有不同的实现）。

例如:

图形管理器调用图形的计算面积方法,执行的是圆形/矩形的计算面积方法.

def get\_total\_area(self):

"""

计算总面积

:return:

"""

total\_area = 0

for item in self.\_\_graphics:

total\_area += item.get\_area() 🡪 图形的面积

return total\_area

3.优点：增强程序扩展性，体现开闭原则。

**四、面向对象设计原则:**

开闭原则（Open Closed Principle）:增加新功能,不修改客户端代码.

比如:某个技能,增加新影响效果,只要创建新的效果类,不需要修改其他代码.

单一职责（Single Responsibility Principle）:每个类有且只有一个改变的原因.

比如:技能系统中,每个类的职责明确.

依赖倒置（Dependency Inversion Principle）:使用抽象(父),而不是用具体(子)

比如:技能释放器,调用的是影响,而不是伤害生命/降低防御力..

组合复用（Composite Reuse Principle）:使用关联关系,代替继承关系.

比如:技能释放器与影响效果,使用了关联关系.

里氏替换（Liskov Substitution Principle）:父类出现的地方,可以被子类替换

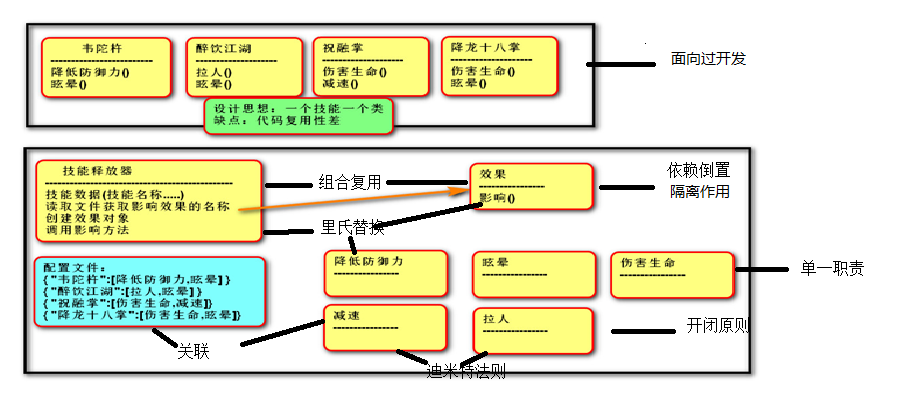
替换后,保持原有功能.

比如:技能释放器使用影响效果,但可以被所有具体效果替换.

迪米特法则（Law of Demeter）:低耦合

比如:影响效果的变化,不影响释放器.

例如：



面向过程与面向对象是不同的编程思想，两者经常会混合使用，面向对象更加适合庞大的项目，但是面向过程适合简单的项目。使用面向过程的时候，你将管理大量的函数和结构体，因为它是以函数和结构体作为模块的基础单元。使用面向对象的时候，你将管理大量的类，因为它是以类作为模块的基础单元。