面向对象课程总结

设计质量

单个类的设计与实现

在大一的时候，我曾经花了一些时间自学 C++，在这个过程中自然也接触了面向对象的思想，通过类来组织起数据和函数的方式让我十分激动，当时的想法就是这种方式太适合实现游戏中的一个个物体和对象了，也能够适应现实生活中的很多应用场景；而且继承、虚函数、友元等等方式对于明确各个对象之间的关系，实现相互间的合作也提供了很大的便利。但是当时更多地纠结在语法和算法上，并没有非常仔细对面向对象思想探究下去；之后在第二课堂的 Java 和软件工程课程上也算是又被教育了一遍面向对象的重要性，但是觉得也就这样了。

而在这个学期学习了面向对象设计课程之后，首先是对类和对象之间的关系更清晰了：类是一个用于构造对象的模板，规定了对象拥有的数据及其类型，对象能够执行的动作和对象状态的变化空间；而对象是一个具有计算能力的实体，封装其状态，对外部屏蔽细节，能够在相应状态上执行动作即方法，通过消息传递机制与其他对象交互。

同时通过课程学习，对于使用类来组织数据和方法的理解又更深入了一些：

1. 类不仅仅是包含数据那么简单，一个类管理的数据和这个类的职责密切相关，而类所承担的职责又与类之间形成的层次和协作结构有关；
2. 对象的属性定义了对象状态，而其方法则定义对象作用于其状态上的行为，而对象的状态也需要注意其内部状态到外部状态的映射；
3. 对于类的属性和方法，其可见性是十分重要的，要注意访问控制，隐藏尽可能多的细节；

因此在设计类的时候，需要注意许多细节：

1. 确定类的属性时需要确定类的逻辑/状态边界，不能一股脑地将数据都交给一个类来管理，要避免类保存的数据太多变成单纯的数据类，同时一些可能是多个类共享的数据的管理也需要在设计的时候考虑到；
2. 而在设计类的方法的时候，要考虑到类的职责和需要方法完成的功能，要避免一个方法完成过多的功能，因为这可能导致方法的代码膨胀冗长而变成过长方法，在实现的时候要善于把长方法分解，将功能提炼成多个精简的方法；
3. 设计方法的时候要明确方法运行后达到的效果是什么，方法运行时要求满足的条件是什么，方法运行时需要使用哪些数据，而这些也和之后学习的规格有关

当然，对类的设计需要考虑整个程序，在面向对象程序中，类是作为基本的编程单位，类之间协作完成程序的功能，根据程序需要完成的功能来分配类需要实现的功能。

类的层次

对于类的层次，我想到的就是继承以及多态之类，当然还有对接口的实现。

在最初接触面向对象时，我认为继承是一种很好的反映类之间关系的方法，而多态也帮助减少了很多重复的代码以及方便了对象的使用（指向派生类对象的基类指针），而在诸如工厂模式等设计模式中，继承和多态发挥的作用也是非常重要的。

继承是面向对象程序设计的三大特征(封装、继承和多态)之一，继承划分了类的层次性，父类代表的是更一般、更泛化的类，而子类则是更为具体、更为细化，继承是实现代码重用、扩展软件功能的重要手段，子类中与父类完全相同的属性和方法不必重写，只需写出新增或改写的内容，这就是说子类可以复用父类的内容，不必一切从零开始。而在 Java 中，可以通过实现接口的方式来完成所谓的多继承。

而在这个学期面向对象课程的学习中，我自然了解了更多和类的层次有关的内容，但是同时我也觉得我们的作业中对于这方面的训练并不够：首先是在电梯系列的作业中，虽然强制要求了继承一系列的调度器并重写相关的方法，但是重写其实就相当于子类提供了一个全新的自己的方法，我并不觉得对于继承的理解有什么太大的意义；而在之后的出租车系列作业中，要求对于继承出租车类的新出租车满足里氏替换原则等等，让我觉得还是有些意义的，但我觉得对于继承和与此带来的泛化与细化的应用还是很简单的；给我的感觉是这门课的代码训练重点并不是对于这些面向对象特性的运用，而只是促使大家在完成程序的过程中，对于类的实现采用面向对象的设计原则，而这些特征则融入到了代码实现中，并没有刻意强调和深入，只要大家能够更好更高效地实现功能。

类的协同

面向对象程序依靠类之间的协作完成程序的功能，除此之外，类之间的协同也是在进行类的设计时需要时刻注意，相互影响的。在进行类的设计的时候要考虑对象之间的协同来安排类需要承担的职责和需要管理的数据；而在对程序进行整体思考的时候要考虑如何将功能分配给各个类以及如何使各个类协作完成功能。

总的来说，类的协同是时刻存在和必须考虑的，每次作业都不例外。而电梯系列作业，则涉及到调度问题，需要一个控制/调度中心管理电梯的运动，同时也需要从请求队列获取请求；出租车系列作业则同样需要从请求队列获取请求，还需要一个控制中心根据请求开启抢单窗口，并将请求分配给某一辆出租车。这些都体现了类之间通过协同完成程序功能，而这种方式也使得程序实现的逻辑更容易理解。

设计原则

在实现之前要先进行设计，要注意简化类方法的职责，设计类之间的协同，注意空间与时间的平衡；在动手写代码之前进行设计还是很有用的，对于理清思路，减少重复推翻已写的代码很有帮助。我自己从第一次作业开始就很注意在实现代码之前明确需要的各个类和它们之间的关系，并提前打好架构草稿，因此在完成作业的时候只需要纠结具体的某个算法实现，在大的框架上没有出过任何问题。

而对于设计原则的检查，则是老师提到的非常经典的 SOLID 了，SRP（每个类或方法都只有一个明确的职责）、OCP（无需修改已有实现，而是通过扩展来增加新功能）、LSP（任何父类出现的地方都可以使用子类来代替，并不会导致使用相应类的程序出现错误）、ISP（当一个类实现一个接口类的时候，必须要实现接口类中的所有接口，否则就是抽象类）、DIP（高层模块不应该依赖于低层模块，二者都应该依赖于抽象；抽象不应该依赖于细节，细节应该依赖于抽象）。

对于 SRP，LSP，ISP 等，在设计之初就会考虑到，SRP 对于类的设计是有指导意义的；而 LSP 原则在涉及到类的继承和多态时则需要注意；OCP 开闭原则对扩展开放，对修改关闭，保持原有的测试代码仍然能够正常运行，我们只需要对扩展的代码进行测试，可以提高复用性和可维护性。

线程

在以前并没有十分明确的线程和进程的概念，只是大概听说，虽然通过 node.js 实现过多进程（js 是单线程语言），但是当时并不理解而且还绕了几个圈，由于是多进程，进程通信也是依靠管道和套接字。而在学了面向对象课程之后，对多线程的理解更深入了，能够较为熟练地编写多线程的程序，完成多个线程之间的通信和协作，再加上 OS 课程的配合，理解了死锁，也学会了通过上锁来避免出现一些因为多个线程同时读写相同内容导致的意外的

错误。

在多线程程序中，线程安全是非常重要的，对于数据共享要注意进行同步控制；而多线程在我们今后的学习和工作中也是十分普遍的，能够提高整体的处理性能。

规格化设计

规格提供了抽象的描述；编写规格使定义的抽象清除显示出来，使注意力迅速集中在不一致不完全和不明确的问题上；规格也定义了用户和实现者之间的契约；规格可以适用于许多不同的目的，并可以成为一个有用的维护工具。

在基于规格来实现代码时，也能感受到规格化设计带来的好处。使用 JSF 来撰写规格也是一种统一规范的手段吧。

测试验证

功能测试

检查程序功能是否完备，针对不同输入运行是否正确，可以从两点设计测试用例：

1. 根据程序需要处理的数据的范围来设计测试用例，检查其在正常情况和边界条件下的功能是否完备；
2. 分析程序运行过程中各个对象可能处于的状态，设计测试用例检查各个对象在不同状态下的功能是否正常；

例如对于电梯就可以检查其在 1 层和顶层的运作，检查其在上行下行或者停留状态时对捎带的反应；而在文件系统的测试中可以进行的操作处理更多，针对不同状态的测试显然也更多。

输入异常测试

输入异常测试应该说是最基础最普遍也是非常重要的一个测试，是针对程序鲁棒性的测试，检查程序在非正常情况下的反应，看其是否可以正确反馈错误信息，不会出错或崩溃。

在经过若干次作业之后，我们还是发现针对输入检查最好的方式还是正则表达式，不管是对于输入格式检查还是对于输入的部分内容的提取，正则表达式都是比较方便和高效的。

线程安全测试

对于线程安全的测试我想到的基本都是造成线程之间互相争夺资源，如果同步控制没有做好，有可能造成死锁现象的发生。

规格

通过类规格理解类管理的数据和类承担的功能；通过过程规格，了解方法正确执行的前置条件，方法带来的副作用和方法提供的结果。

自动化测试

自动化测试确实对于发现错误有帮助，对于程序进行测试还是需要自己设计测试用例，但是利用方法的分支覆盖率检查，对于没有覆盖到的分支设计新的测试用例，能够有效地发现一些隐藏的 bug。