宜宾学院

软件设计模式与体系结构设计报告书

学 院: 人工智能与大数据学部 班 级: 18 级 9 班

学生姓名: 邓文涛 学 号: 200109304

设计地点（单位） 6217

设计题目: **实现一个医院系统的部分功能**

完成日期： 2021 年 7 月 1 日

**1、使用模板方法模式+装饰器模式+组合模式设计出一个病人去医院就诊步骤**

1.1 问题分析

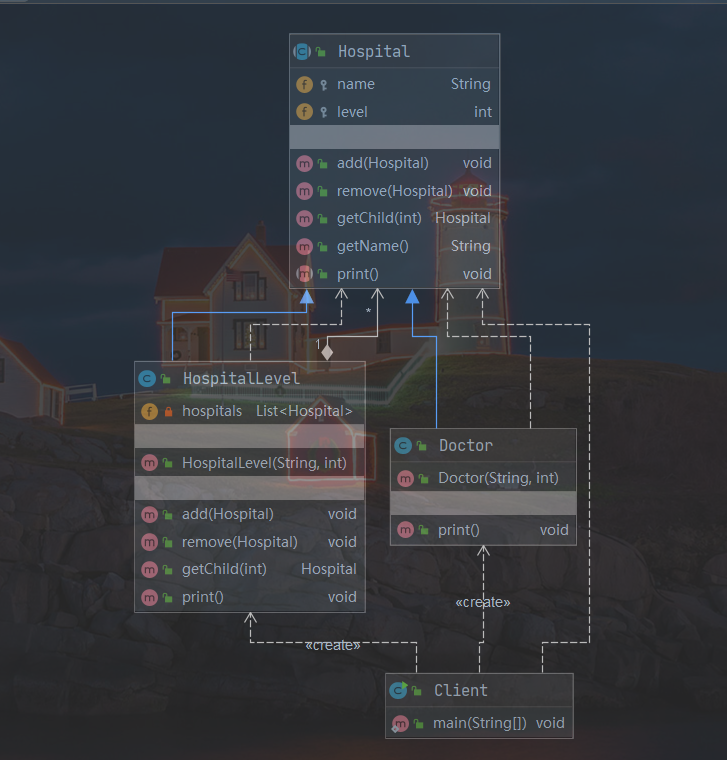
模板方法（Template Method）模式：定义一个操作中的算法骨架，而将算法的一些步骤延迟到子类中，使得子类可以不改变该算法结构的情况下重定义该算法的某些特定步骤。它是一种类行为型模式。对于病人就诊问题，病人看病需要经过**挂号**(选择诊疗科目、选择医生)**，医生确诊，付费，医生治疗，结束治疗**5个步骤，对于这五个步骤，简单分析，我们发现，除了挂号这个步骤不同的医院科目和医生类别可以进行选择，其他的步骤每个医院都是这样的流程。就是说在实现的时候挂号这个方法是一个抽象类，其他方法都是具体实现。

在病人确诊之后，医生会根据病人的情况开病例单，此时就需要先付费才能治疗，病历单包括“基础诊疗费”和其它药物或治疗方法的费用，当然它还需要满足方便扩展不同的药物或治疗方法的费用。此时就使用装饰器模式（Decorator Pattern），它允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。根据题意，抽象一个病例单类(**抽象构建角色**)，基础诊疗费(**具体构建角色**)，还需要一个**抽象装饰器类**，用与继承抽象构件，并包含具体构件的实例，可以通过其子类扩展具体构件的功能，其它药物或治疗方法的费用都为**具体装饰器角色**，而当还有其他费用的时候，就可以直接再添加具体装饰器角色类。

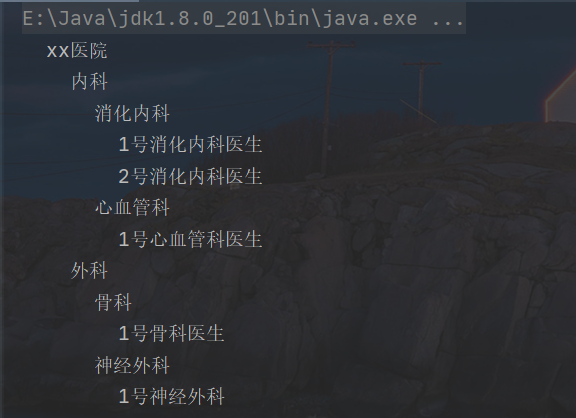
关于组合模式，是一种组合多个对象形成树形结构以表示具有 "整体—部分" 关系的层次结构，这种模式使用在病人选择挂号之前，病人可以整体浏览医院的体系结构和医生，再选择挂号合适的科室和医生。在设计的时候，医院的科系主要由内科和外科组成，内科又由消化内科和心血管科组成，消化内科里又包含那些医生，依次类推，可以发现，这就是**子菜单**和**子菜单项**的关系，子菜单表示每个科室或者类别，子菜单项里面只有具体实现，就是表示层级关系下的最后一层医生。最后通过遍历的方式，打印所有医生所在具体那个科室，供病人选择挂号。

1.2 设计思路

首先是关于挂号流程的 UML图，使用组合模式

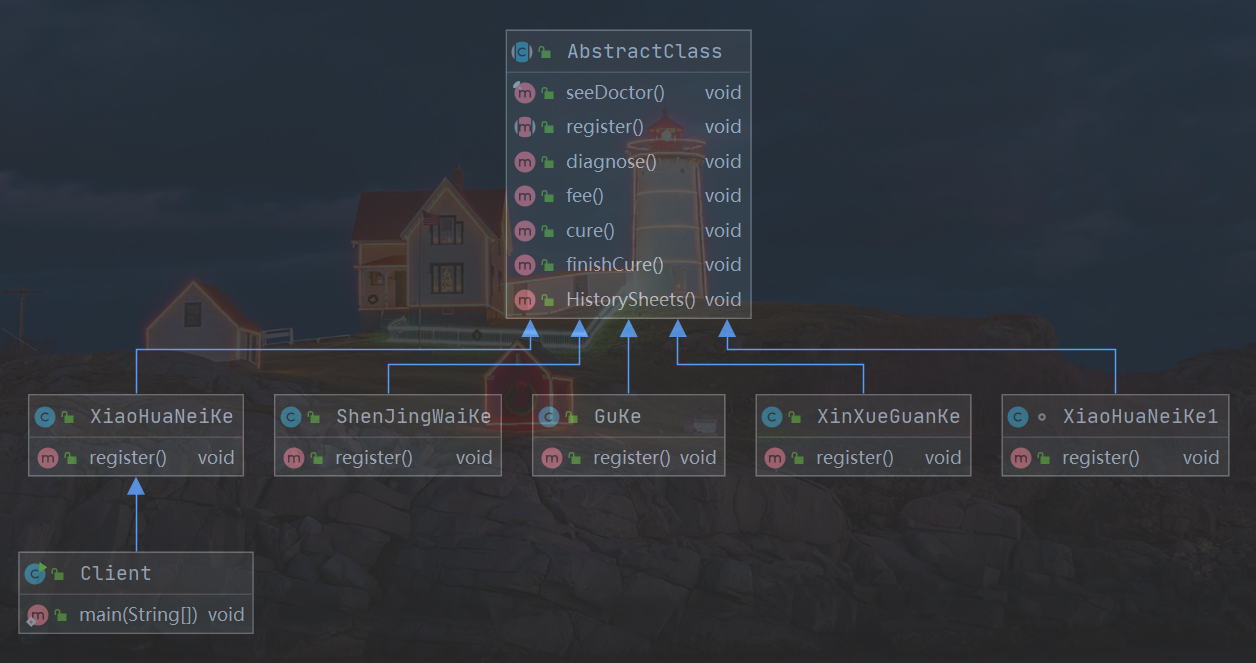


Hospital()类中定义名称name和层级level，用于描述每一层次的信息，添加子菜单方法add()，移除子菜单方法remove()，以继定义一个抽象类print()方法打印菜单名称的方法(包含子菜单和菜单项)，HospitalLevel()类用是于实现医院结构层次的菜单类，在层次当中要顶一个ArrayLIst的基于顺序表的结构用于存储每一次结构的包含的信息。Doctor()类是子菜单项，最后现在Client()的main()方法中实现把所有科室和级别分别通过add()方法加入到顺序表中，如下



病人就可以根据这个树状结构选择合适的科室和医生挂号。

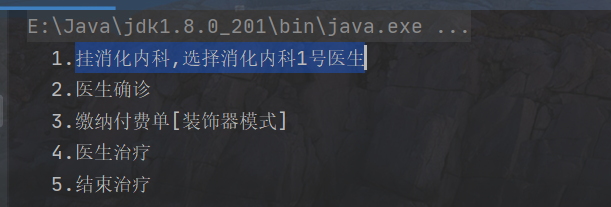
然后是关于看病流程的 UML图，使用模板方法模式



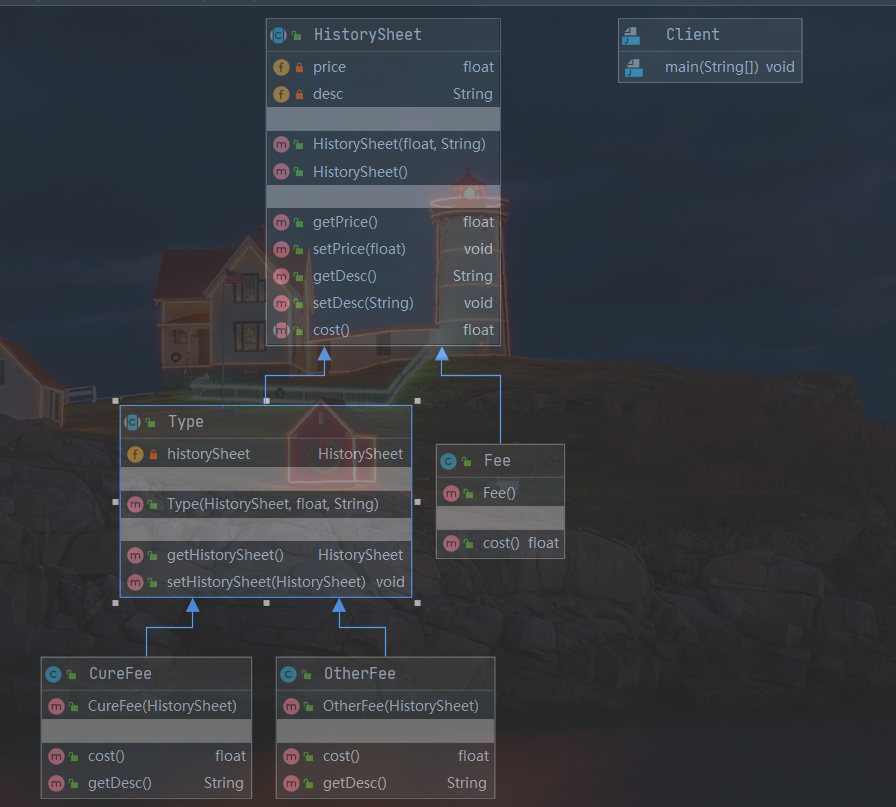
根据UML 图，首先是先定义一个抽象类，这个抽象类(AbstractClass)用来定义模板方法和普通方法方法，这个模板方法是不允许被继承的，用 final 修饰，然后在这个抽象类中实现所有方法，抽象方法用于继承具体实现，普通方法具体实现

*//模板方法定义  
public final void seeDoctor() {  
 register(); //1.挂号(选择诊疗科目,选择医生)  
 diagnose(); //2.医生确诊  
 fee(); //3.付费  
 cure(); //4.医生治疗  
 finishCure(); //5.结束治疗  
}*

在挂号这个过程中，有多少个医生就得创建多少个类，因为每个医生代表的科室或描述都不一样，这里我们使用挂消化内科,选择消化内科1号医生为例显示结果：



最后就是关于病历单的UML图，使用装饰器模式



HistorySheet()类为病历单类，属于抽象构建角色，定义了price 和 desc 用于描述病历单上费用和类型描述，cost()为抽象方法用于计算价格。

*//创建Fee()类为具体构建角色，就是基本的医药费  
public class Fee extends HistorySheet{  
 public Fee() {  
 super(45, "基础诊疗费");  
 }  
 @Override  
 public float cost() {  
 return getPrice();  
 }  
}*

Type()类为抽象装饰器类，是为了声明这个病历单，用于继承实现CurFee()类和OtherFee()类，这两个类分别代表治疗费用和其他费用，在重写的cost()方法中要加上基础诊疗费。都属于具体装饰器角色，当还有其他的费用时，就可以添加具体的类，达到了方便扩展的目的。结果如下：



1.3 编码与测试

最后一个部分就是整合这三种模式应用到一个系统上，我们就可以根据实际情况在相应的位置创建对象，最后在在Finish()这个类中实现调用。

整合代码如下：

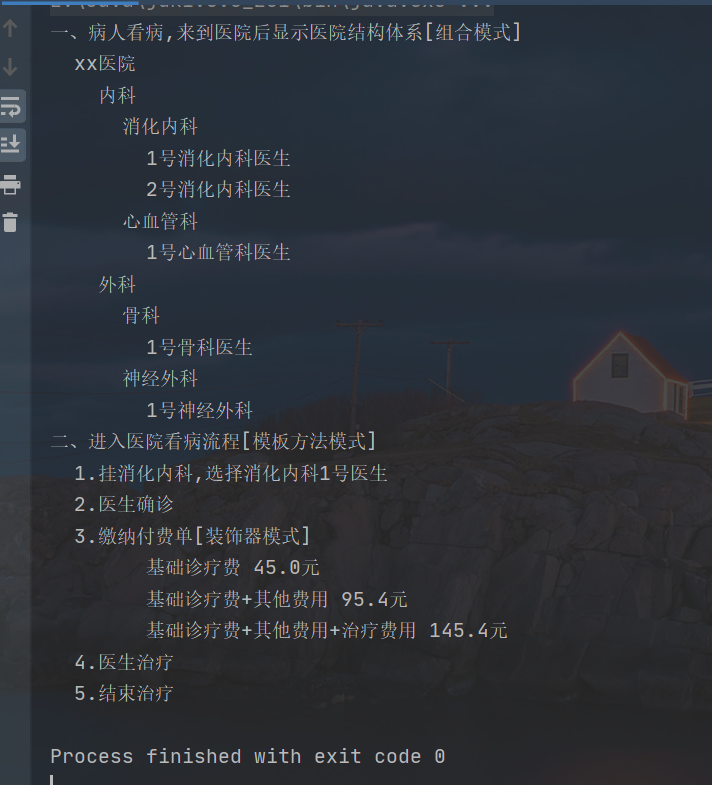
**第一部分**(在付费处实现装饰器模式计算费用)

*//1.挂号(选择诊疗科目,选择医生)  
public abstract void register();  
//2.医生确诊  
public void diagnose() {  
 System.out.println(" 2.医生确诊");  
}  
//3.付费  
public void fee() {  
 System.out.println(" 3.缴纳付费单[装饰器模式]");  
 HistorySheets();  
}  
//4.医生治疗  
public void cure() {  
 System.out.println(" 4.医生治疗");  
}  
//5.结束治疗  
public void finishCure() {  
 System.out.println(" 5.结束治疗");  
}*  
*//这个方法的主要目的是实现病历单,病例单是由装饰器模式设计,所以想要加其他上面费用,只需要在上面加就好  
public void HistorySheets() {  
 //基础费用(必定存在)  
 HistorySheet fee = new Fee();  
 System.out.println(" " + fee.getDesc() +" "+ fee.cost()+"元");  
 //基础费用+其他费用  
 fee = new OtherFee(fee);  
 System.out.println(" " + fee.getDesc() +" "+ fee.cost()+"元");  
 //基础费用+其他费用+治疗费用  
 fee = new CureFee(fee);  
 System.out.println(" " + fee.getDesc() +" "+ fee.cost()+"元");  
}*

**第二部分**(在Finish类中进行调用)

*public class Finish {  
 public static void main(String[] args) {  
 // 1.病人看病显示医院结构  
 System.out.println("一、病人看病,来到医院后显示医院结构体系[组合模式]");  
 Structure();  
 // 2.挂号，选择选择诊疗科目、选择医生  
 System.out.println("二、进入医院看病流程[模板方法模式]");  
 SeeDoctor();  
 }  
  
 private static void SeeDoctor() {  
 //消化内科病人看病流程  
 //创建对象  
 XiaoHuaNeiKe person = new XiaoHuaNeiKe();  
 //调用看病这个方法  
 person.seeDoctor();  
 }  
  
 private static void Structure() {  
 //创建菜单项  
 Hospital menu1 = new HospitalLevel("消化内科",3);  
 menu1.add(new Doctor("1号消化内科医生",4));  
 menu1.add(new Doctor("2号消化内科医生",4));  
  
 Hospital menu2 = new HospitalLevel("心血管科",3);  
 menu2.add(new Doctor("1号心血管科医生",4));  
 Hospital menu3 = new HospitalLevel("骨科",3);  
 menu3.add(new Doctor("1号骨科医生",4));  
 Hospital menu4 = new HospitalLevel("神经外科",3);  
 menu4.add(new Doctor("1号神经外科",4));  
 //创建二级菜单,把三级菜单加到二级菜单中  
 Hospital menu5 = new HospitalLevel("内科",2);  
 menu5.add(menu1);  
 menu5.add(menu2);  
 //menu5.print();  
 Hospital menu6 = new HospitalLevel("外科",2);  
 menu6.add(menu3);  
 menu6.add(menu4);  
 //创建一级菜单,把二级菜单加到一级菜单中  
 Hospital menu = new HospitalLevel("xx医院",1);  
 menu.add(menu5);  
 menu.add(menu6);  
 menu.print();  
 }  
}*

最后结果：



1.4 总结

通过本次作业，更加深刻的理解了这几种设计模式的概念以及使用方法，能够合理的应用在生活中的一些案例中。但是对于本次作业的结果来讲，还存在诸多不合理的设计，但是基本满足了大部分需求，对于三种设计模式的整合部分过于牵强了，效率相对较低，但后期还会进一步完善。