工作日志

|  |  |  |
| --- | --- | --- |
| 时间 | 工作内容 | 关键词 |
| 2022年5月26日 | 今天课上我们根据老师上课布置的实验十一的各项要求，对小组成员进行任务分工。本次实验的主要任务为：  1. 第五章课后习题14，故障树转割集树练习  2. 参考教材6.2，结合项目的进程和开发历程，从设计原则的几个方面，组员对负责设计的模块进行评估，思考存在的问题和解决方案。  3. 阅读下面DI资料（或查阅其它相关资料），学习依赖注入技术  4. 阅读下面设计模式资料（或查阅其它相关资料），结合项目的进程和开发历程，分析项目采用了那些设计模式  5. 给出4种设计模式的例子（语言不限，以组为单位），并总结其特点 （保存到每个小组选定的协作开发平台上）  6. 记录项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。  其中任务二我们分别对自己负责的功能模块进行评估，其中我负责的是用户信息管理模块。 | 实验十一任务分工 |
| 2022年5月27日 | 今天我主要做了实验十一的任务四，结合我们的学生选课信息管理系统分析可知，我们主要用了面向对象的设计方法。  首先我们这个项目的要求就是面向用户的包括学生与管理员，向他们提供便于操作的相应输入输出功能。我们在设计的过程中使用了很多UML用例图、对象图和类图用来表示相关数据和功能，并用这些作为设计目标，这些软件开发方法符合建立模型体现抽象思维过程和面向对象的方法。  此外，我们的这一软件系统将相应计算机系统中的数据以及与这个数据相关的一切操作语言都封装了起来，提供的是便于理解和使用功能接口，符合面向对象方法中“封装”的特点。在我们的软件生存期的各个阶段所开发出来的“部件”都是相应的类，在面向对象生存期的各个阶段对各个类进行细化，使类成为分析设计和实现的基本单元，而且它们也有继承和对象组合，这些也是面向对象设计方法的特点。 | 分析自己的项目设计方法  面向对象的设计方法 |
| 2022年5月28日 | 今天我主要做了实验十一的任务二，我对我负责的用户信息管理这一模块根据设计原则进行了评估。主要内容如下：  模块化角度：  接口：区分管理员登录与学生登录的方法。需不需要将身份作为参数提供。  解决方案：作为参数提供。或不作为参数提供都可以。  作为参数提供是否会提示身份选择错误。  解决方案：若是作为参数提供，则会提示。  信息隐藏：  增量式开发：  抽象：  通用性：管理员登录与学生登录，学生注册和管理员注册，管理员信息修改和学生信息修改设计方面具有通用性。 | 用户信息管理模块  根据设计原则进行评估 |
| 2022年5月29日 | 今天我和郑晓旭同学合作做了实验十一的任务五，我负责的主要内容如下：  1> 创建型模式—工厂方法模式（定义一个创建对象的接口，但由子类决定要实例化的类是哪一个，单产品系）  （1） 实例：  // 产品  public interface ICar {}  public class MidCar implements ICar {}  public class UpCar implements ICar {}  // 抽象工厂  public abstract class AbstractFactory {    public abstract ICar create();  }  // 实例工厂  public class MidCarFactory extends AbstractFactory {    @Override  public ICar create() {  return new MidCar();  }  }  public class UpCarFactory extends AbstractFactory {    @Override  public ICar create() {  return new UpCar();  }  }  （2） 特点：  把类的实例化延迟到子类；  一个工厂对应生产一个产品。  2> 创建型模式--抽象工厂模式（提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类，多产品系）  （1） 实例：  // 车产品  public interface ICar {}  public class MidCar implements ICar {}  public class UpCar implements ICar {}  // 颜色产品  public interface IColor {}  public class RedColor implements IColor {}  public class BlueColor implements IColor {}  // 抽象工厂  public abstract class AbstractFactory {    public abstract ICar createMidCar() throws Exception;  public abstract ICar createUpCar() throws Exception;  public abstract IColor fillRed() throws Exception;  public abstract IColor fillBlue() throws Exception;  }  // 车工厂，生产除车外的产品，会抛出自定义异常  public class CarFactory extends AbstractFactory {    @Override  public ICar createMidCar() throws Exception {  return new MidCar();  }  @Override  public ICar createUpCar() throws Exception {  return new UpCar();  }  @Override  public IColor fillRed() throws Exception {  throw new CqException("Please use ColorFactory to create!");  }  @Override  public IColor fillBlue() throws Exception {  throw new CqException("Please use ColorFactory to create!");  }  }  // 颜色工厂，生产除颜色外的产品，会抛出自定义异常  public class ColorFactory extends AbstractFactory {    @Override  public IColor fillRed() throws Exception {  return new RedColor();  }  @Override  public IColor fillBlue() throws Exception {  return new BlueColor();  }  @Override  public ICar createMidCar() throws Exception {  throw new CqException("Please use CarFactory to create!");  }  @Override  public ICar createUpCar() throws Exception {  throw new CqException("Please use CarFactory to create!");  }  }  （2） 特点：  当一个产品族中的多个对象被设计成一起工作时，它能保证客户端始终只使用同一个产品族中的对象；  产品族扩展非常困难，要增加一个系列的某一产品，既要在抽象工厂加代码，也要在工厂生产者里加代码。 | 设计模式 |
| 2022年5月30日 | 今天晚上小组成员分别展示了各自所完成的实验十一的内容，然后由组长总结并撰写最终要上交的实验报告。 | 实验十一成果汇总和报告撰写 |