工作日志

|  |  |  |
| --- | --- | --- |
| 时间 | 工作内容 | 关键词 |
| 2022年5月26号 | 今天老师根据讲授的有关设计模块的内容，给我们安排了新的任务：  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。  当天晚上我们小组开了会议，确认任务分配。每个人对自己所负责的模块进行评估，并提交评估结果。 | 任务分配  模块评估  面向对象设计  相关思考 |
| 2022年5月27号 | 今天我主要做了实验十一的任务四  根据我们的设计，我觉得我们应该采用伪合理的设计过程。先设计最难的最不明确的部分，因为隐蔽问题所引发的异常可能会迫使整个设计做出改变。然后设计我们熟知的拿手的部分。最后按照理想过程编写我们的文档。 | 项目设计方法分析 |
| 2022年5月28号 | 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：  具体功能如下：管理员可以修改课程信息，进行课程评分  评估：  模块化：我在本部分使用了自顶而下和自下而上相结合的方法，各个模块之间耦合度较小。  接口：本部分使用的接口较少，主要是管理员和学生两部分，管理员有3个接口，分别对应  其三个功能，学生只需要一个接口。我定义的接口能够完整的实现相应功能。  信息隐藏：由于我设计的这一部分所涉及的方法较少，因此对于各个方法没有使用信息隐藏，只是对对象进行了隐藏。  增量式开发：我在设计的时候应用了增量式开发，对各个模块的依赖关系进行分析后作出了使用图，然后设计。  抽象：我通过对模块进行分解，实现了层次结构，从而进行抽象  通用性：我设计的这个模块通用性并不好，基本只能在本项目中使用，主要是因为我是根据本项目进行的设计，未考虑通用性。  对于我的模块，我认为存在的问题是通用性与抽象做的不好。我准备通过将原来的特定上下文环境进行参数化来让它满足各种环境的工作。 | 管理员管理课程信息模块  根据设计原则进行评估 |
| 2022年5月29号 | 今天我和陈文盼同学合作做了实验十一的任务五，我负责的主要内容如下：  3> 创建型模式—建造者模式（将复杂的构建与其表示相分离，使得相同的构建过程可以创建不同的表示）  （1） 实例：  /\*\*  \* 自带建造者（除了必须字段，其它字段是可选的，易于自己选择不同的表示）  \*  \* 发送邮件参数  \*/  public class SendEmailParams {  // 收件人邮箱，支持多个，用英文逗号隔开  private final String to;  // 发件人账号和密码  private final EmailAccount emailAccount;  // 抄送，支持多个，用英文逗号隔  private final String cc;  // 秘密抄送，支持多个，用英文逗号隔开  private final String bcc;  // 邮件标题  private final String subject;  // 邮件正文  private final String body;  // 邮件内容编码，默认utf-8  private final String encoding;  // 附件列表，每个元素包含fileName和content两个字段  private final List<Attachment> attachments;  private SendEmailParams(Builder builder) {  this.to = builder.to;  this.emailAccount = builder.emailAccount;  this.cc = builder.cc;  this.bcc = builder.bcc;  this.subject = builder.subject;  this.body = builder.body;  this.encoding = builder.encoding;  this.attachments = builder.attachments;  }  public static class Builder {  // 必须的参数  private final String to;  private final EmailAccount emailAccount;  // 可选参数  private String cc = "";  private String bcc = "";  private String subject = "";  private String body = "";  private String encoding = "";  private List<Attachment> attachments = new ArrayList<Attachment>();  public Builder(String to, EmailAccount emailAccount) {  this.to = to;  this.emailAccount = emailAccount;  }  public Builder cc(String cc) {  this.cc = cc;  return this;  }  public Builder bcc(String bcc) {  this.bcc = bcc;  return this;  }  public Builder subject(String subject) {  this.subject = subject;  return this;  }  public Builder body(String body) {  this.body = body;  return this;  }  public Builder encoding(String encoding) {  this.encoding = encoding;  return this;  }  public Builder attachments(List<Attachment> attachments) {  this.attachments = attachments;  return this;  }  public SendEmailParams build() {  return new SendEmailParams(this);  }  }  }  （2） 特点：  各个具体的建造者相互独立，有利于系统的扩展；  调用者不必知道产品内部构建的细节，有利于减少细节风险；  如果产品的内部变化复杂，会增加很多的建造者类。  4> 行为型模式--责任链模式（对请求的发送者和接收者进行解耦）  （1） 实例：  public class Request {}  public class Response {}  public interface Filter {    void doFilter(Request request, Response response, Chain chain);  }  /\*\*  \* 驱动责任链传递  \*/  public interface Chain {    void handle(Request request, Response response);  }  /\*\*  \* 责任链实例  \*  \* @author CQJames 2020-07-13  \*/  public class FilterChain implements Chain {    private Queue<Filter> filterQueue = new LinkedList<>();    public FilterChain addFilter(Filter filter) {    filterQueue.offer(filter);  return this;  }    @Override  public void handle(Request request, Response response) {    if (filterQueue.isEmpty()) {  return;  }  Filter filter = filterQueue.poll();  filter.doFilter(request, response, this);  }  }  public class OneFilter implements Filter {    @Override  public void doFilter(Request request, Response response, Chain chain) {    chain.handle(request, response);  }  }  （2） 特点：  降低模块之间的耦合度；  不能保证请求一定被接收。 | 设计模式 |
| 2022年5月30号 | 小组成员展示实验结果，总结报告。 | 总结报告 |