**会议记录表**

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 2023年5月26日 | 星期 | 第十四周 |
| 队伍 | 啊对对队 | 会议地点 | 线上会议 |
| 进行时长 | 2小时 |
| 参会人员 | 孟小凡、尹国泰、黄泰达、吴百川 | | |
| 会议主题 | 设计模块（一 ），编写程序 | | |
| 会议内容 | **议题1：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**  Design Patterns-Elements of Reusable Object-Oriented Software.pdf  The GoF Design Patterns Reference.pdf  [Design Patterns - Wikipedia](https://en.wikipedia.org/wiki/Design_Patterns)  1.单例模式（Singleton Pattern）：用于确保一个类只有一个实例，并提供一个全局访问点。在人才招聘系统中，可能有一些全局资源或服务只需要被实例化一次，例如数据库连接池或配置管理器。  2.工厂模式（Factory Pattern）：用于根据不同的条件创建对象，而不需要在代码中直接实例化具体类。在人才招聘系统中，可能有多种类型的用户，如候选人、招聘经理等，可以使用工厂模式来创建对应的用户对象。  3.迭代器模式（Iterator Pattern）：用于提供一种统一的方式遍历集合对象的元素，而不需要暴露集合的内部实现。在人才招聘系统中，可能会有各种数据集合，如职位列表、候选人列表等，迭代器模式可以提供一种统一的遍历方式。  **议题2：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）**  1.建造者模式（Bulider）  工厂类模式提供的是创建单个类的模式，而建造者模式则是将各种产品集中起来进行管理，用来创建复合对象，所谓复合对象就是指某个类具有不同的属性，其实建造者模式就是前面抽象工厂模式和最后的Test结合起来得到的  [java] view plaincopy  public class Builder {  private List<Sender> list = new ArrayList<Sender>();  public void produceMailSender(int count){  for(int i=0; i<count; i++){  list.add(new MailSender());  }  }  public void produceSmsSender(int count){  for(int i=0; i<count; i++){  list.add(new SmsSender());  }  }  }  测试类：  [java] view plaincopy  public class Test {  public static void main(String[] args) {  Builder builder = new Builder();  builder.produceMailSender(10);  }  }  从这点看出，建造者模式将很多功能集成到一个类里，这个类可以创造出比较复杂的东西。所以与工程模式的区别就是：工厂模式关注的是创建单个产品，而建造者模式则关注创建符合对象，多个部分。因此，是选择工厂模式还是建造者模式，依实际情况而定。  2.单例模式（Singleton Pattern）：  特点：确保一个类只有一个实例，并提供全局访问点。  示例：一个日志记录器类，它只能创建一个实例，所有其他类需要访问该日志记录器时都通过该实例进行访问。  public class Logger {  private static Logger instance;  private Logger() {  // 私有构造函数，防止外部实例化  }  public static Logger getInstance() {  if (instance == null) {  instance = new Logger();  }  return instance;  }  public void log(String message) {  // 日志记录逻辑  System.out.println("Log: " + message);  }  }  3. 观察者模式：如果有更新，那么订阅了该话题/主题的客户端将被通知。  理解观察者模式的最简单方法是想象一个邮件列表，你可以在其中订阅任何主题，无论是开源、技术、名人、烹饪还是您感兴趣的任何其他内容。每个主题维护者一个它的订阅者列表，在观察者模式中它们相当于观察者。当某一个主题更新时，它所有的订阅者（观察者）都将被通知这次改变。并且订阅者总是能取消某一个主题的订阅。  让我们来看看观察者模式的代码示例，从主题/话题类开始：  package org.opensource.demo.observer;  public interface Topic {  public void addObserver([Observer][22] observer);  public void deleteObserver([Observer][22] observer);  public void notifyObservers();  }  这段代码描述了一个为不同的主题去实现已定义方法的接口。注意一个观察者如何被添加、移除和通知的。  这是一个主题的实现示例：  package org.opensource.demo.observer;  import java.util.List;  import java.util.ArrayList;  public class Conference implements Topic {  private List&lt;Observer&gt; listObservers;  private int totalAttendees;  private int totalSpeakers;  private [String][18] nameEvent;  public Conference() {  listObservers = new ArrayList&lt;Observer&gt;();  }  public void addObserver([Observer][22] observer) {  listObservers.add(observer);  }  public void deleteObserver([Observer][22] observer) {  int i = listObservers.indexOf(observer);  if (i &gt;= 0) {  listObservers.remove(i);  }  }  public void notifyObservers() {  for (int i=0, nObservers = listObservers.size(); i &lt; nObservers; ++ i) {  [Observer][22] observer = listObservers.get(i);  observer.update(totalAttendees,totalSpeakers,nameEvent);  }  }  public void setConferenceDetails(int totalAttendees, int totalSpeakers, [String][18] nameEvent) {  this.totalAttendees = totalAttendees;  this.totalSpeakers = totalSpeakers;  this.nameEvent = nameEvent;  notifyObservers();  }  }  这段代码定义了一个特定主题的实现。当发生改变时，这个实现调用它自己的方法。注意这将获取观察者的数量，它以列表方式存储，并且可以通知和维护观察者。  4.工厂模式（Factory Pattern）：  特点：封装对象的创建过程，通过工厂类来创建对象，使得客户端与具体类的实例化解耦。  示例：一个汽车制造工厂，工厂根据客户的需求（参数）生产不同类型的汽车，客户只需要告诉工厂要生产什么样的汽车，而不需要关心具体的制造过程。  public interface Car {  void drive();  }  public class SedanCar implements Car {  @Override  public void drive() {  System.out.println("Driving a sedan car");  }  }  public class SUVCar implements Car {  @Override  public void drive() {  System.out.println("Driving an SUV car");  }  }  public class CarFactory {  public Car createCar(String type) {  if (type.equalsIgnoreCase("sedan")) {  return new SedanCar();  } else if (type.equalsIgnoreCase("suv")) {  return new SUVCar();  } else {  throw new IllegalArgumentException("Invalid car type: " + type);  }  }  }  5.装饰者模式（Decorator Pattern）：  特点：动态地给一个对象添加额外的功能，避免了使用继承扩展功能的问题。  示例：一个咖啡店的订单系统，有一个基础咖啡类，通过装饰者模式可以动态地为该咖啡添加额外的配料，如牛奶、糖浆等，而无需修改基础咖啡类的代码。  public interface Coffee {  String getDescription();  double getCost();  }  public class SimpleCoffee implements Coffee {  @Override  public String getDescription() {  return "Simple coffee";  }  @Override  public double getCost() {  return 1.0;  }  }  public abstract class CoffeeDecorator implements Coffee {  protected Coffee decoratedCoffee;  public CoffeeDecorator(Coffee decoratedCoffee) {  this.decoratedCoffee = decoratedCoffee;  }  @Override  public String getDescription() {  return decoratedCoffee.getDescription();  }  @Override  public double getCost() {  return decoratedCoffee.getCost();  }  }  public class MilkDecorator extends CoffeeDecorator {  public MilkDecorator(Coffee decoratedCoffee) {  super(decoratedCoffee);  }  @Override  public String getDescription() {  return super.getDescription() + ", with milk";  }  @Override  public double getCost() {  return super.getCost() + 0.5;  }  }  这些设计模式都有各自的特点和适用场景，通过使用它们可以提高代码的可维护性、灵活性和可复用性。单例模式适用于需要全局访问一个实例的情况；观察者模式适用于对象间存在一对多的依赖关系，当一个对象改变时需要通知其他依赖者；工厂模式适用于需要封装对象创建过程的情况，客户端只需要与工厂接口交互；装饰者模式适用于动态地为对象添加额外功能的情况，避免了使用继承带来的静态限制。  6.享元模式（Flyweight Pattern）  主要用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。  享元模式尝试重用现有的同类对象，如果未找到匹配的对象，则创建新对象。我们将通过创建 5 个对象来画出 20 个分布于不同位置的圆来演示这种模式。由于只有 5 种可用的颜色，所以 color 属性被用来检查现有的 Circle 对象。  介绍  意图：运用共享技术有效地支持大量细粒度的对象。  主要解决：在有大量对象时，有可能会造成内存溢出，我们把其中共同的部分抽象出来，如果有相同的业务请求，直接返回在内存中已有的对象，避免重新创建。  何时使用： 1、系统中有大量对象。 2、这些对象消耗大量内存。 3、这些对象的状态大部分可以外部化。 4、这些对象可以按照内蕴状态分为很多组，当把外蕴对象从对象中剔除出来时，每一组对象都可以用一个对象来代替。 5、系统不依赖于这些对象身份，这些对象是不可分辨的。  如何解决：用唯一标识码判断，如果在内存中有，则返回这个唯一标识码所标识的对象。  关键代码：用 HashMap 存储这些对象。  应用实例： 1、JAVA 中的 String，如果有则返回，如果没有则创建一个字符串保存在字符串缓存池里面。 2、数据库的连接池。  优点：大大减少对象的创建，降低系统的内存，使效率提高。  缺点：提高了系统的复杂度，需要分离出外部状态和内部状态，而且外部状态具有固有化的性质，不应该随着内部状态的变化而变化，否则会造成系统的混乱。  使用场景： 1、系统有大量相似对象。 2、需要缓冲池的场景。  注意事项： 1、注意划分外部状态和内部状态，否则可能会引起线程安全问题。 2、这些类必须有一个工厂对象加以控制。  **议题3：网查询“阿里编程规范 （如：阿里巴巴JAVA开发手册）；华为 编程军规”等，对照自己的代码看有哪些不符合规范的地方，修改。**  阿里巴巴JAVA开发手册  <https://blog.csdn.net/chuige2013/article/details/127193250>  通过对阿里巴巴JAVA开发手册的研读，我们考虑从以下方面对我们的人才招聘系统进行修改：  1.代码规范和风格：参考阿里巴巴JAVA开发手册中的代码规范和风格，对系统中的Java代码进行审查和调整。确保代码的可读性、可维护性和一致性。  2.性能优化：通过分析系统的性能瓶颈和瓶颈点，对关键代码进行优化，提升系统的响应速度和并发处理能力。  3.异常处理和日志记录：检查系统中的异常处理机制，确保异常被正确捕获和处理，同时添加必要的日志记录，方便故障排查和系统监控。  4.安全性增强：评估系统的安全性，确保用户输入的数据得到充分的验证和过滤，以防止常见的安全漏洞，例如跨站脚本攻击（XSS）和SQL注入。  5.并发控制：考虑在系统中引入适当的并发控制机制，以处理多个用户同时访问和修改数据时的冲突和竞争条件。  6.用户界面优化：审查用户界面的设计和布局，优化用户交互体验，使系统更加直观、易用和用户友好。  7.扩展性和可维护性：通过采用模块化的架构和设计模式，确保系统具有良好的扩展性和可维护性。这样可以方便日后的功能扩展和修改。  8．测试和质量保证：增加适当的单元测试、集成测试和系统测试，以确保系统的质量和稳定性。引入自动化测试工具和流程，提高测试效率和覆盖范围。  9.文档和注释：完善系统的文档和注释，包括代码注释、API文档和用户手册等，以方便团队成员的协作开发和后续维护工作。 | | |
| 备注 | 无 | | |