### JAVA第五阶段—DAY08-JAVA作业答案

**1：什么是设计模式？**

就是经过实践验证的用来解决特定环境下特定问题的解决方案

**2：设计模式用来干什么？**

寻找合适的对象

决定对象的粒度

指定对象的接口

描述对象的实现

运用复用机制

重复使用经过实践验证的正确的，用来解决某一类问题的解决方案来达到减少

工作量、提高正确率等目的。

**3：什么是对象粒度**

对象中方法的多少就是粒度

**4：基本的Java编程设计应遵循的规则？**

面向接口编程，优先使用对象组合

**5：设计模式的应用范围**

所能解决的特定的一类问题中

**6：简述什么是单例模式，以及他解决的问题，应用的环境，解决的方案，模式的本质？**

在任何时间内只有一个类实例存在的模式。

需要有一个从中进行全局访问和维护某种类型数据的区域的环境下使用单例模式。

解决方案就是保证一个类只有一个类实例存在。

本质就是实例共用同一块内存区域。

**7：代码示例：单例模式的两种实现方法，并说明优缺点**

public class Test{

     public Test(){

    }

private static Test test = new Test();

     public static Test getInstance(){

     return test;

}

}

public class Test{

     private static Test test = null;

    private Test(){

}

    public static Test getInstance(){

   if(test==null){

  test = new Test();

}

   return test;

}

}

第二种方式不需每次都创建一个类实例，而只是在第一次创建

**8：简述什么是工厂模式，以及他解决的问题，应用的环境，解决的方案，模式的本质**

利用工厂来解决接口选择的问题的模式。

应用环境：当一个类无法预料要创建哪种类的对象或是一个类需要由子类来指定创建的对象时，就需要用到工厂模式。

解决方案：定义一个创建对象的接口,让子类来决定具体实例化哪一个类。

本质就是根据不同的情况来选择不同的接口。

**9：代码示例：工厂模式的实现方法**

public class Factory{

     public static Sample creator(int which){

     if (which==1){

     return new SampleA();

}else if (which==2)

       return new SampleB();

     }

    }

    }

     Public class MyFactory{

        Public static myFactory f = null;

        Public MyFactory(){

        }

        Public static MyFactory getInstance(){

           If(f==null){

              F=new MyFactory();

           }

        }

        Public DBDAO getDAO(){

            Return new DBDAOImpl();

        }

    }

**10：述什么是值对象模式，以及他解决的问题，应用的环境，解决的方案，模式的本质**

用来把一组数据封装成一个对象的模式。

解决问题：在远程方法的调用次数增加的时候，相关的应用程序性能将会有很大的下降。

解决方案：使用值对象的时候，可以通过仅仅一次方法调用来取得整个对象，而不是使用多次方法调用以得到对象中每个域的数值。

本质：就是把需要传递的多个值封装成一个对象一次性传过去。

**11：代码示例：值对象模式的实现方法**

public class UserModel{

     private String userId;

    private String userName;

public void setUserId(String id){

     this.userId = id;

    }

    public String getUserId(){

    return userId;

    }

    public void setUserName(String name){

   this.userName = name;

}

public String getUserName(){

      return userName;

    }

    }

**12：简述什么是DAO模式，以及他解决的问题，应用的环境，解决的方案，模式的本质**

DAO：数据访问对象

解决问题：根据数据源不同，数据访问也不同。根据存储的类型（关系数据库、面向对象数据库、纯文件等）和供应商实现不同，持久性存储（如数据库）的访问差别也很大。如何对存储层以外的模块屏蔽这些复杂性，以提供统一的调用存储实现。程序的分布式问题。

解决方案：是将数据访问逻辑抽象为特殊的资源，也就是说将系统资源的接口从其底层访问机制中隔离出来；通过将数据访问的调用打包，数据访问对象可以促进对于不同数据库类型和模式的数据访问。

DAO的本质：就是一层屏蔽一种变化。

本质：分层，是系统组件和数据源中间的适配器。（一层屏蔽一种变化）

**13：代码示例：DAO模式的实现方法**

public interface CustomerDAO {

public int insertCustomer(...);

public Collection selectCustomersVO(...);

}

**14：什么是开放-封闭法则(OCP)**

可扩展但是不可以更改已有的模块

对扩展是开放的 对修改是封闭

**15：什么是替换法则(LSP)**

使用指向基类（超类）的引用的函数，必须能够在不知道具体派生类（子类）对象

类型的情况下使用

**16：如何综合使用我们学过的设计模式来构建合理的应用程序结构**

是采用接口进行隔离，然后同时暴露值对象和工厂类，如果是需要数据存储的功能，又会通过DAO 模式去与数据存储层交互。

**17：构建常用的合理的Java应用包结构**

Ui(表现层)

business--factory,ebi,ebo

dao--factory,dao,impl