2152118 史君宝 软件工程　软件体系结构与设计模式 作业

1. （　）系统体系结构的最佳表示形式是一个可执行的软件原型。

A. 真

B. 假

选择：B

2. （　）软件体系结构描述是不同项目相关人员之间进行沟通的使能器。

A. 真

B. 假

选择：A

3. （　）良好的分层体系结构有利于系统的扩展与维护。

A. 真

B. 假

选择：A

4. （　）消除两个包之间出现的循环依赖在技术上是不可行的。

A. 真

B. 假

选择：B

5. （　）设计模式是从大量成功实践中总结出来且被广泛公认的实践和知识。

A. 真

B. 假

选择：A

6. 程序编译器的体系结构适合使用（　）。

A. 仓库体系结构

B. 模型－视图－控制器结构

C. 客户机／服务器结构

D. 以上选项都不是

选择：B

7. 网站系统是一个典型的（　　）。

A. 仓库体系结构

B. 胖客户机／服务器结构

C. 瘦客户机／服务器结构

D. 以上选项都不是

选择：C

8. 在分层体系结构中，（　　）实现与实体对象相关的业务逻辑。

A. 表示层

B. 持久层

C. 实体层

D. 控制层

选择：C

1. （　　）面向对象设计是在分析模型的基础上，运用面向对象技术生成软件实现环境下的设计模型。

A. 真

B. 假

选择：A

2. （　　）系统设计的主要任务是细化分析模型，最终形成系统的设计模型。

A. 真

B. 假

选择：A

3. （　　）关系数据库可以完全支持面向对象的概念，面向对象设计中的类可以直接对应到关系数据库中的表。

A. 真

B. 假

选择：B

4. （　）用户界面设计对于一个系统的成功是至关重要的，一个设计得很差的用户界面可能导致用户拒绝使用该系统。

A. 真

B. 假

选择：A

5. 内聚表示一个模块（　）的程度，耦合表示一个模块（　）的程度。

A. 可以被更加细化

B. 仅关注在一件事情上

C. 能够适时地完成其功能

D. 联接其他模块和外部世界

选择：B，D

6. 良好设计的特征是（　）。

A. 模块之间呈现高耦合

B. 实现分析模型中的所有需求

C. 包括所有组件的测试用例

D. 提供软件的完整描述

E. 选项 B 和 D

F. 选项 B、C 和 D

选择：E

7. （　）是选择合适的解决方案策略，并将系统划分成若干子系统，从而建立整个系统的体系结构；（　）细化原有的分析对象，确定一些新的对象、对每一个子系统接口和类进行准确详细的说明。

A. 系统设计

B. 对象设计

C. 数据库设计

D. 用户界面设计

选择：A，B

8. 下面的（　　）界面设计原则不允许用户保持对计算机交互的控制。

A. 允许交互中断

B. 允许交互操作取消

C. 对临时用户隐藏技术内部信息

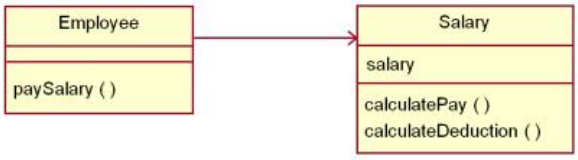
D. 只提供一种规定的方法完成任务

选择：D

1. 下图是某公司支付雇员薪水程序的一个简化 UML 设计类图，目前雇员薪水是按固定月薪支付的，系统需要准时支付正确的薪金，并从中扣除各种扣款。现在该公司准备增加“时薪”和“底薪+佣金”两种支付方式，考虑到良好的可扩展性，开发人员打算使用设计模式修改原有设计，以支持多种薪水支付方式。

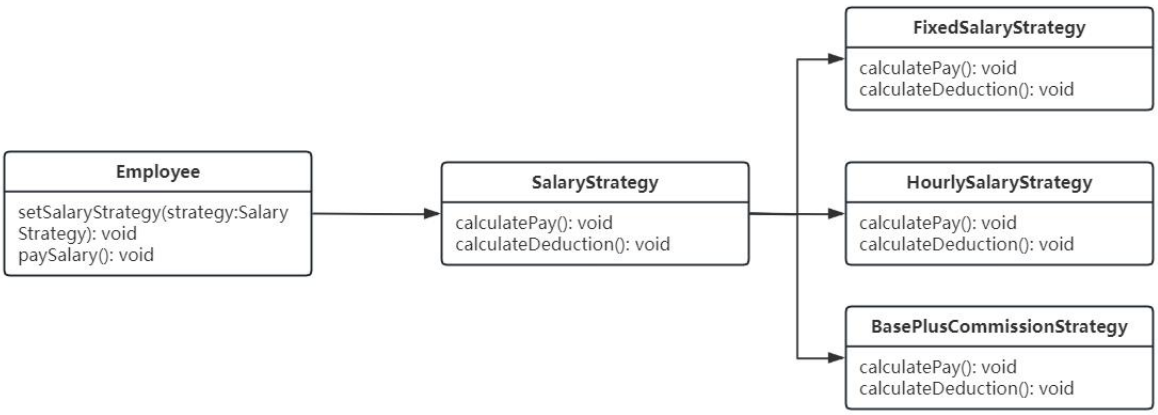
（1）你会选择什么设计模式？为什么？

（2）请画出修改后的 UML 设计类图，并用 C++语言编写实现该类图的程序。



（1）可以选择使用**策略模式**。策略模式定义了一系列算法，将每个算法封装起来，并使它们可以相互替换，使算法的变化独立于使用算法的客户。

（2）



```
#include <iostream>
using namespace std;

// 雇员类
class Employee {
private:
    SalaryStrategy *salary;

public:
    void setSalaryStrategy(SalaryStrategy *strategy) {
        salary = strategy;
    }

    void paySalary() {
        if (salary) {
            salary->calculatePay();
            salary->calculateDeduction();
        } else {
            cout << "No salary strategy set." << endl;
        }
    }
};

// 薪水策略基类
class SalaryStrategy {
public:
    virtual void calculatePay() = 0;
    virtual void calculateDeduction() = 0;
};

// 固定月薪策略
class FixedSalaryStrategy : public SalaryStrategy {
public:
    void calculatePay() override {
        cout << "Calculate fixed salary payment." << endl;
    }

    void calculateDeduction() override {
        cout <<"Calculate deduction for fixed salary payment." <<endl;
    }
};

// 按小时支付策略
```

```
class HourlySalaryStrategy : public SalaryStrategy {
public:
    void calculatePay() override {
        cout << "Calculate hourly salary payment." << endl;
    }

    void calculateDeduction() override {
        cout <<"Calculate deduction for hourly salary payment." <<endl;
    }
};

// 底薪加佣金支付策略
class BasePlusCommissionStrategy : public SalaryStrategy {
public:
    void calculatePay() override {
        cout <<"Calculate base salary plus commission payment." <<endl;
    }

    void calculateDeduction() override {
        cout << "Calculate deduction for base salary plus commission
payment." << endl;
    }
};
```