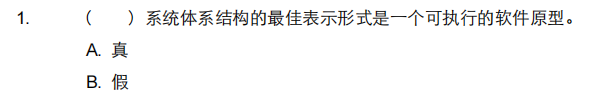
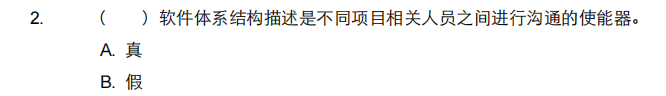
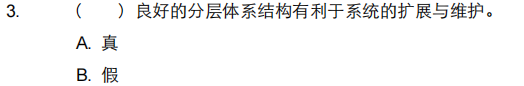
2152118 史君宝 软件工程 软件体系结构与设计模式 作业



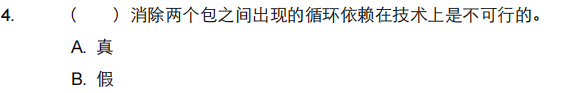
选择：B



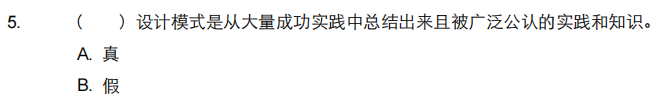
选择：A



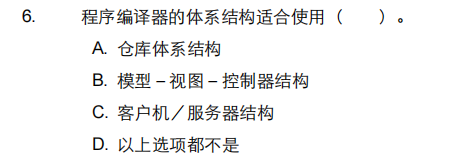
选择：A



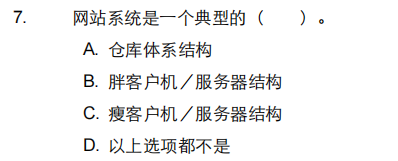
选择：B



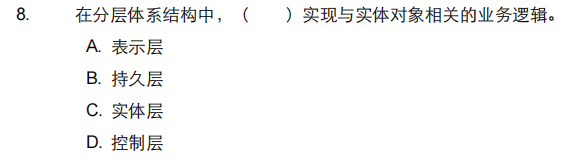
选择：A



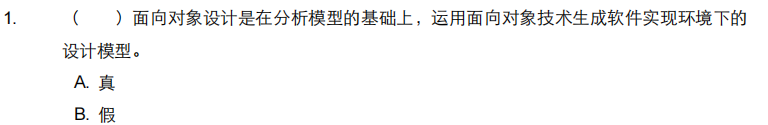
选择：B



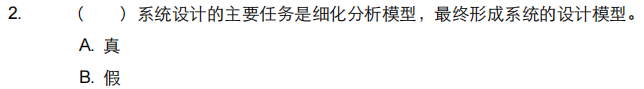
选择：C



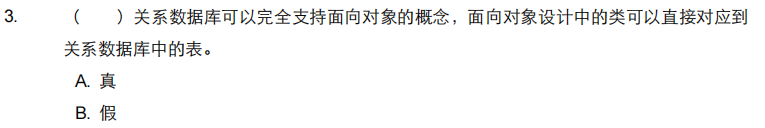
选择：C



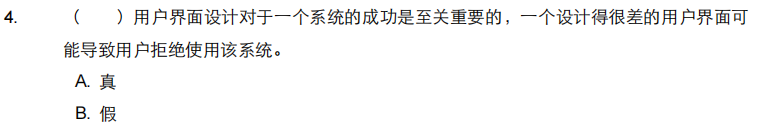
选择：A



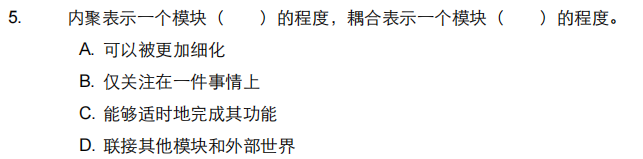
选择：A



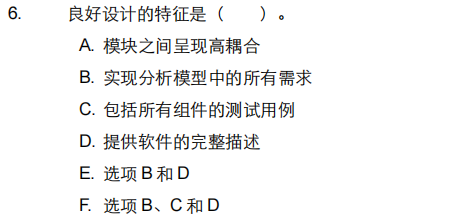
选择：B



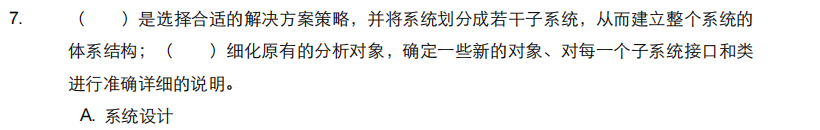
选择：A

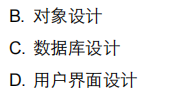


选择：B，D

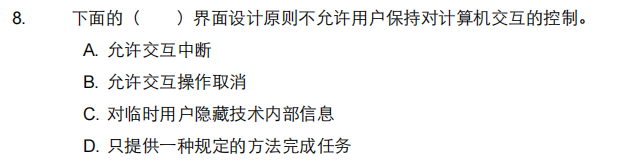


选择：E

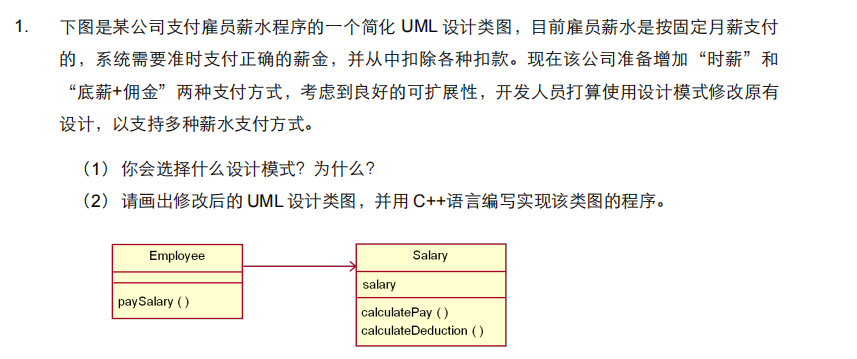




选择：A，B

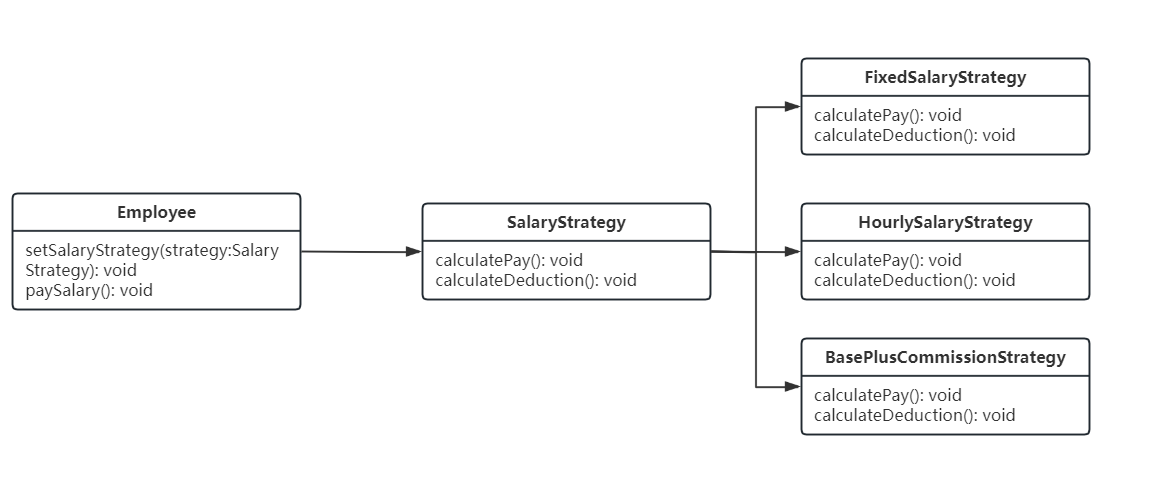


选择：D



（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;

}

};