全国计算机技术与软件专业技术资格（水平）考试

2022 年上半年 软件设计师 下午试卷

（考试时间 14:00～16:30 共 150 分钟）

请按下述要求正确填写答题卡

1．在答题纸的指定位置填写你所在的省、自治区、直辖市、计划单列市的名称。

2．在答题纸的指定位置填写准考证号、出生年月日和姓名。

3．答题纸上除填写上述内容外只能写解答。

4．本试卷共 6 道题，试题一至试题四是必答题，试题五至试题六选答 1 道。每题 15 分，满分 75 分。

5．解答时字迹务必清楚，字迹不清时，将不评分。

6．仿照下面的例题，将解答写在答题纸的对应栏内。

例题

2022 年上半年全国计算机技术与软件专业技术资格（水平）考试日期是（1）月（2）日。

因为正确的解答是“5 月 28 日”，故在答题纸的对应栏内写上“5”和“28”（参看下表）。

例题	解答栏
（1）	5
（2）	28

试题一（共 15 分）

阅读下列说明和数据流图，回答问题 1 至问题 4，将解答填入答题纸的对应栏内。

【说明】

某公司欲开发一款外卖订餐系统，集多家外卖平台和商户为一体，为用户提供在线浏览餐品、订餐和配送等服务。该系统的主要功能是：

（1）入驻管理。用户注册；商户申请入驻，设置按时间段接单数量阀值等。系统存储商户/用户信息。

（2）餐品管理。商户对餐品的基本信息和优惠信息进行发布、修改、删除。系统存储相关信息。

（3）订餐。用户浏览商户餐单，选择餐品及数量后提交订餐请求。系统存储订餐订单。

（4）订单处理。收到订餐请求后，向外卖平台请求配送。外卖平台接到请求后发布配送单，由平台骑手接单，外卖平台根据是否有骑手接单返回接单状态。若外卖平台接单成功，系统给支付系统发送支付请求，接收支付状态。支付成功，更新订单状态为已接单，向商户发送订餐请求并由商户打印订单，给用户发送订单状态；若支付失败，更新订单状态为下单失败，向外卖平台请求取消配送，向用户发送下单失败。若系统接到外卖平台返回接单失败或超时未返回接单状态，则更新订单状态为下单失败，向用户发送下单失败。

（5）配送。商户备餐后，由骑手取餐配送给用户。送达后由用户扫描骑手出示的订单上的配送码后确认送达，订单状态更改为已送达，并发送给商户。

（6）订单评价。用户可以对订单餐品、骑手配送服务进行评价，推送给对应的商户、所在外卖平台，商户和外卖平台对用户的评价进行回复。系统存储评价。

现采用结构化方法对外卖订餐系统进行分析与设计，获得如图 1-1 所示的上下文数据流图和图 1-2 所示的 0 层数据流图。

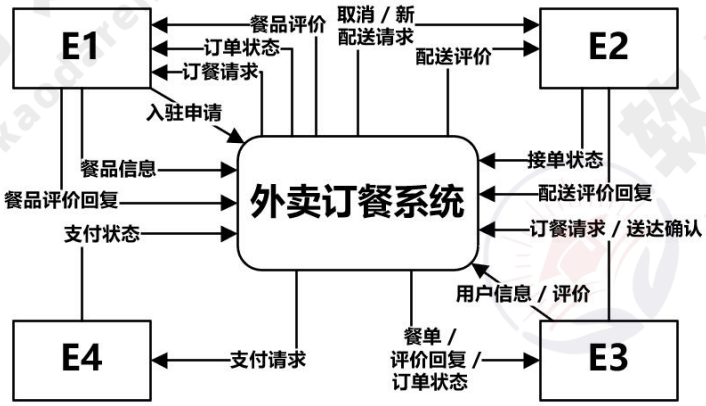


图 1-1 上下文数据流图

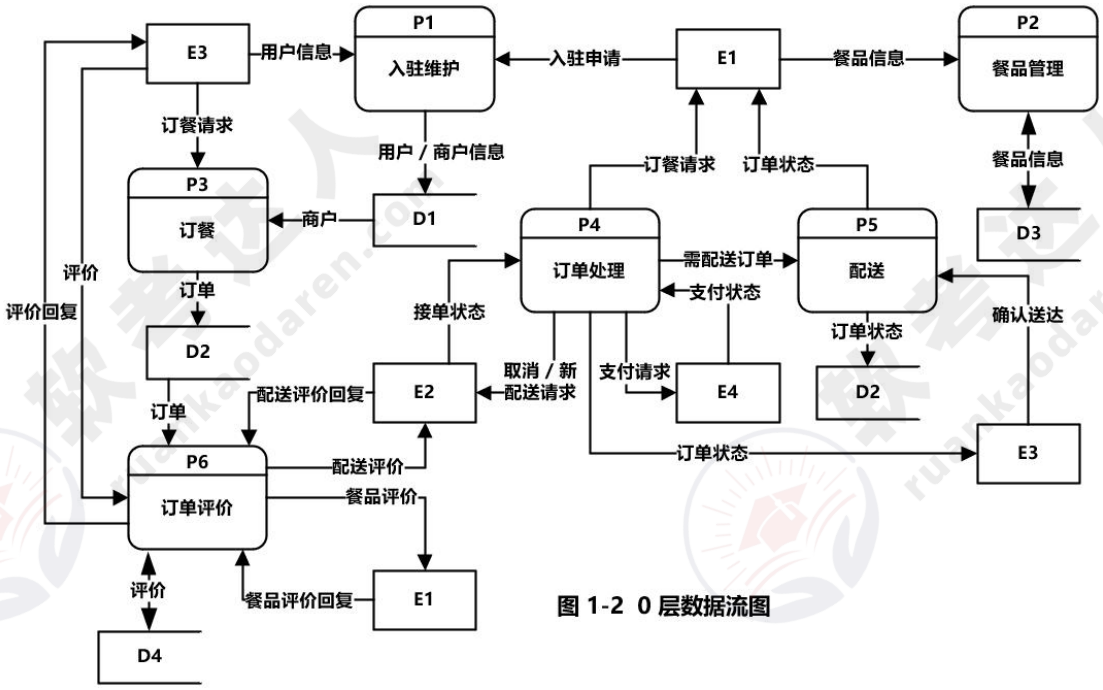


图 1-2 0 层数据流图

【问题 1】（4 分）

使用说明中的词语，给出图 1-1 中的实体 E1～E4 的名称。

E1：商户　E2：外卖平台　E3：用户　E4：支付系统

【问题 2】（4 分）

使用说明中的词语，给出图 1-2 中的数据存储 D1～D4 的名称。

D1：用户/商户信息表　D2：订餐订单信息表

D3：餐品信息表　D4：评价信息表

注：

D2：订餐订单或订单都可

D1～D4 数据存储的后缀为表、文件、信息表等符合题意的名称都可

【问题 3】（4 分）

根据说明和图中术语，补充图 1-2 中缺失的数据流及其起点和终点。

数据流名称	起点	终点
餐单	D3	P3
餐单	P3	E3
订餐请求	P3	P4
订单状态	P4	D2
配送码	P5	E3

上方这条数据流，不确定。因为父图中并没有由系统指向 E3 的数据流配送码，但是根据功能 5（也就是加工 P5）的描述来说，用户是需要扫描订单上的配送码之后才能进行确认送达的，所以好像也还算合理，这个和上一年（也就是 2021 年下半年试题一的农业专家的请求类似，也是父图中的实体没有数据流，但是根据说明中功能的描述应该有这么一条数据流）。总之，大家自行判断，在没有官方解析出来之前，我都不做争论～

【问题 4】（3 分）

根据说明，采用结构化语言对“订单处理”的加工逻辑进行描述。

```
收到订餐请求

向外卖平台请求配送。
外卖平台接到请求后发布配送单，由平台骑手接单，
外卖平台根据是否有骑手接单返回接单状态

IF 外卖平台接单成功 THEN
    系统给支付系统发送支付请求，接收支付状态
    IF 支付成功 THEN
        更新订单状态为已接单，
        向商户发送订餐请求并由商户打印订单，
        给用户发送订单状态
    ELSE
        更新订单状态为下单失败，
        向外卖平台请求取消配送，
        向用户发送下单失败
    ENDIF
ELSE IF 外卖平台返回接单失败或超时未返回接单状态
    更新订单状态为下单失败，
    向用户发送下单失败
ENDIF
```

收到订餐请求

向外卖平台请求配送。

外卖平台接到请求后发布配送单，由平台骑手接单，

外卖平台根据是否有骑手接单返回接单状态

IF 外卖平台接单成功 THEN

系统给支付系统发送支付请求，接收支付状态

IF 支付成功 THEN

更新订单状态为已接单，

向商户发送订餐请求并由商户打印订单，

给用户发送订单状态

ELSE

更新订单状态为下单失败，

向外卖平台请求取消配送，

向用户发送下单失败

ENDIF

ELSE IF 外卖平台返回接单失败或超时未返回接单状态

更新订单状态为下单失败，

向用户发送下单失败

ENDIF

试题二（共 15 分）

阅读下列说明，回答问题 1 至问题 4，将解答填入答题纸的对应栏内。

【说明】

为了提高接种工作，提高效率，并为了抗击疫情提供疫苗接种数据支撑，需要开发一个信息系统，下述需求完成该系统的数据库设计。

【需求分析结果】

（1）记录疫苗供应商的信息，包括供应商名称，地址和一个电话。

（2）记录接种医院的信息，包括医院名称、地址和一个电话。

（3）记录接种者个人信息，包括姓名、身份证号和一个电话。

（4）记录接种者疫苗接种信息，包括接种医院信息，被接种者信息，疫苗供应商名称和接种日期，为了提高免疫力，接种者可能需要进行多次疫苗接种，（每天最多接种一次，每次都可以在全市任意一家医院进行疫苗接种）。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图（不完整）如图 2-1 所示。

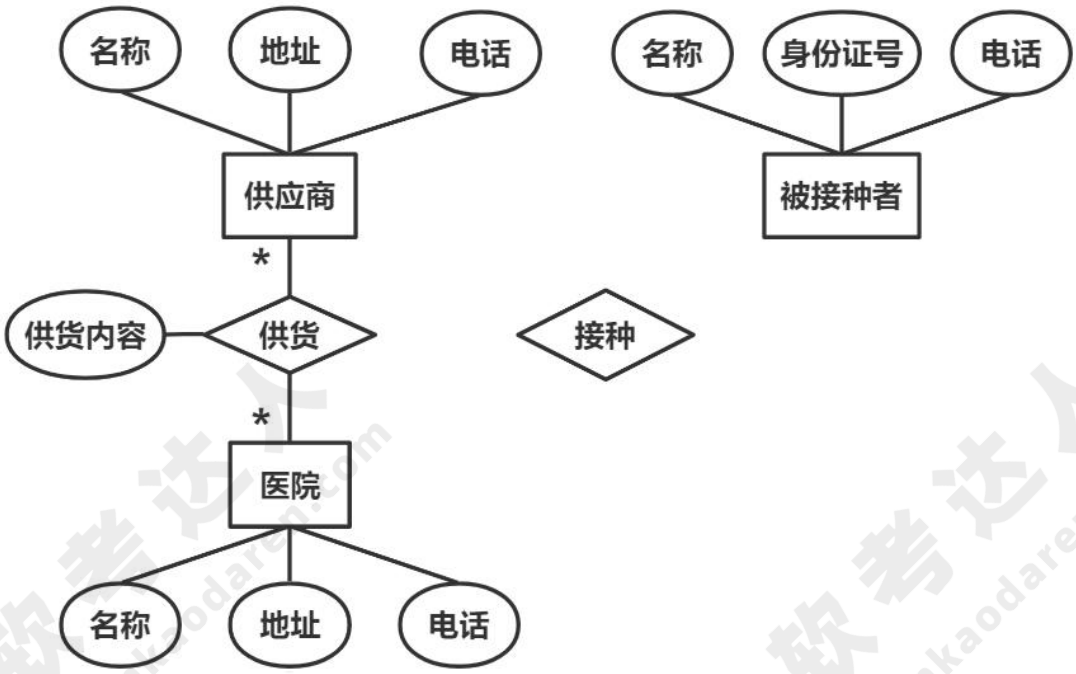


图 2-1 实体联系图

【逻辑结构设计】

根据概念模型设计阶段完成的实体联系图，得出如下关系模式（不完整）：

供应商（供应商名称，地址，电话）

医院（医院名称，地址，电话）

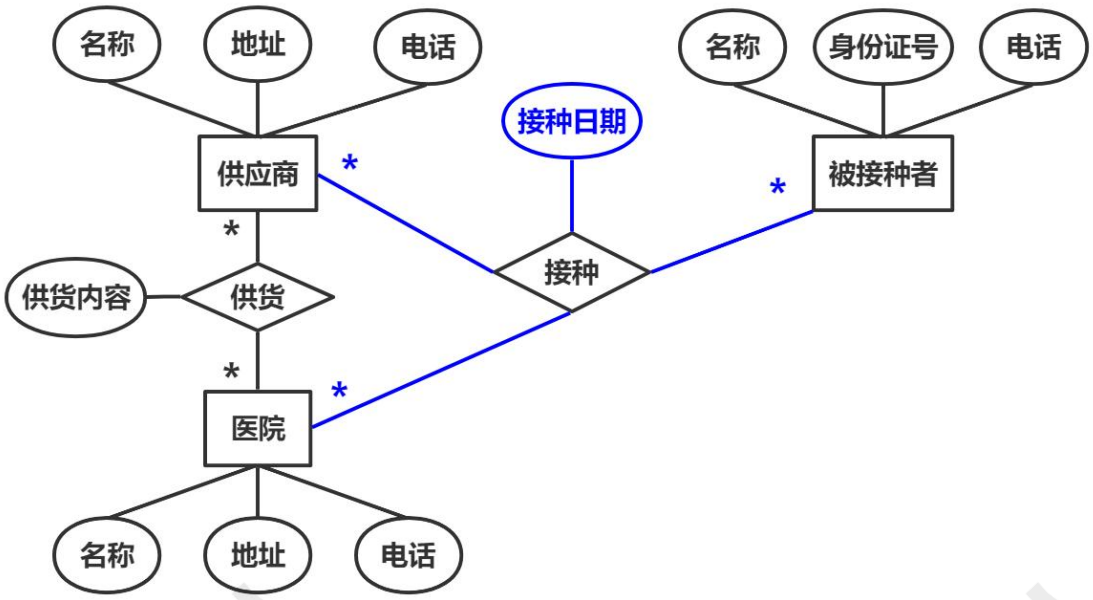
供货（供货商名称，（a），供货内容）

被接种者（姓名，身份证号，电话）

接种（接种者身份证号，（b），医院名称、供应商名称）

【问题 1】（4 分）

根据问题描述，补充图 2-1 的实体联系图（不增加新的实体）。



问题一 补充后的 图 2-1 实体联系图

【问题 2】（4 分）

根据题意，补充逻辑结构设计结果中的（a），（b）两处空缺，并标注主键和外键完整性约束。

（a）：医院名称

（b）：接种时间

第二问直接在（a）（b）两处写出属性，并标注上主键（实线）和外键（虚线）即可。

可以考虑补充供货关系的主键、外键和接种关系的主键、外键。当然可以加也可以不加

（注：这里不是必须的）

供货关系的主键：（供货商名称，医院名称）

外键：供货商名称，医院名称

接种关系的主键：（接种者身份证号，接种时间）

外键：接种者身份证号，医院名称，供应商名称

【问题 3】（7 分）

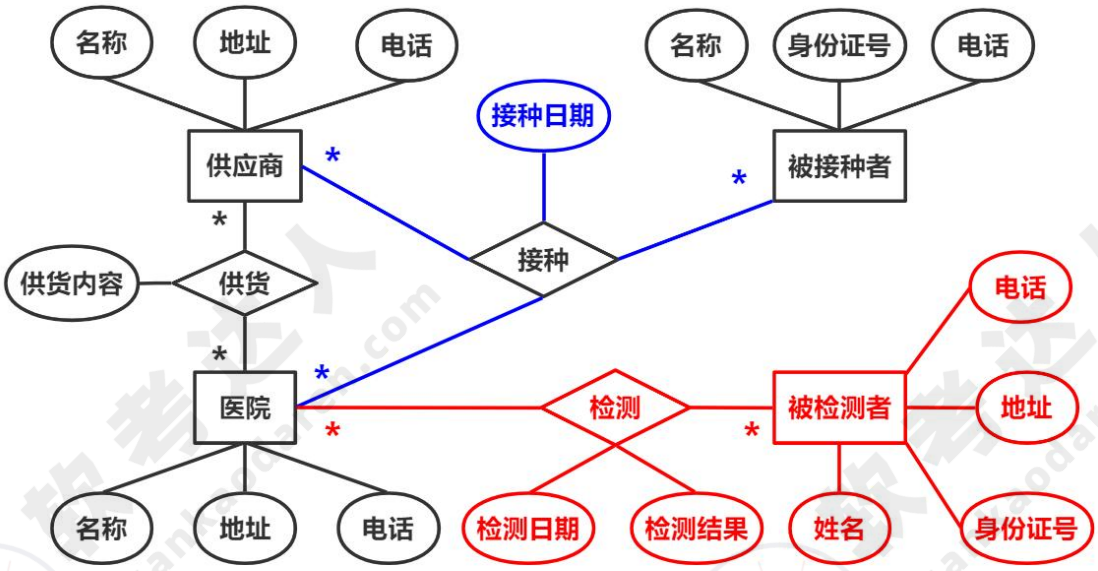
若医院还兼有核酸检测的业务，检测时可能需要进行多次核酸检测（每天最多检测一次），但每次都可以在全市任意一家医院进行检测。

请在图 2-1 中增加“被检测者”实体及相应的属性。医院与被检测者之间的“检测”联系及必要的属性，并给出新增加的关系模式。

“被检测者”实体包括姓名、身份证号、地址和一个电话。

“检测”联系包括检测日期和检测结果等。

不一定要把第一问补充的联系也画上去，当然画上去最好～



问题三 补充后的 图 2-1 实体联系图

被检测者（姓名，身份证号，地址，电话）

检测（医院名称，被检测者身份证号，检测日期，检测结果）

试题三（共 15 分）

阅读下列说明和 UML 图，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

某公司的人事部门拥有一个地址簿（AddressBook）管理系统（AddressBookSystem），用于管理公司所有员工的地址记录（PersonAddress）。员工的地址记录包括：姓名、住址、城市、省份、邮政编码以及联系电话等信息。

管理员可以完成对地址簿中地址记录的管理操作，包括：

（1）管理地址记录。根据公司的人员变动情况，对地址记录进行添加、修改、删除等操作。

（2）排序。按照员工姓氏的字典顺序或邮政编码对地址簿中的所有记录进行排序。

（3）打印地址记录。以邮件标签的格式打印一个地址单独的地址簿。

系统会对地址记录进行管理，为便于管理，管理员在系统中为公司的不同部门建立员工的地址簿的操作，包括：

（1）创建地址簿。新建一个地址簿并保存。

（2）打开地址簿。打开一个已有的地址簿。

（3）修改地址簿。对打开的地址簿进行修改并保存。

系统将提供一个 GUI（图形用户界面）实现对地址簿的各种操作。

现采用面向对象方法分析并设计该地址簿管理系统，得到如图 3-1 所示的用例图以及图 3-2 所示的类图。

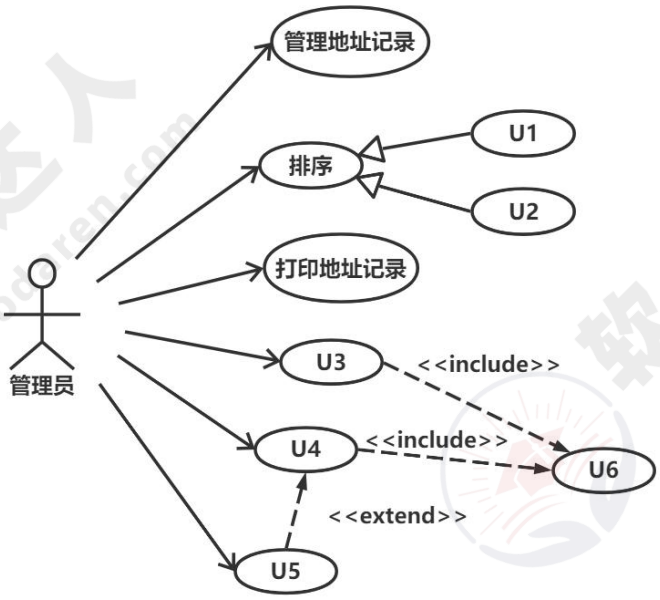


图 3-1 用例图

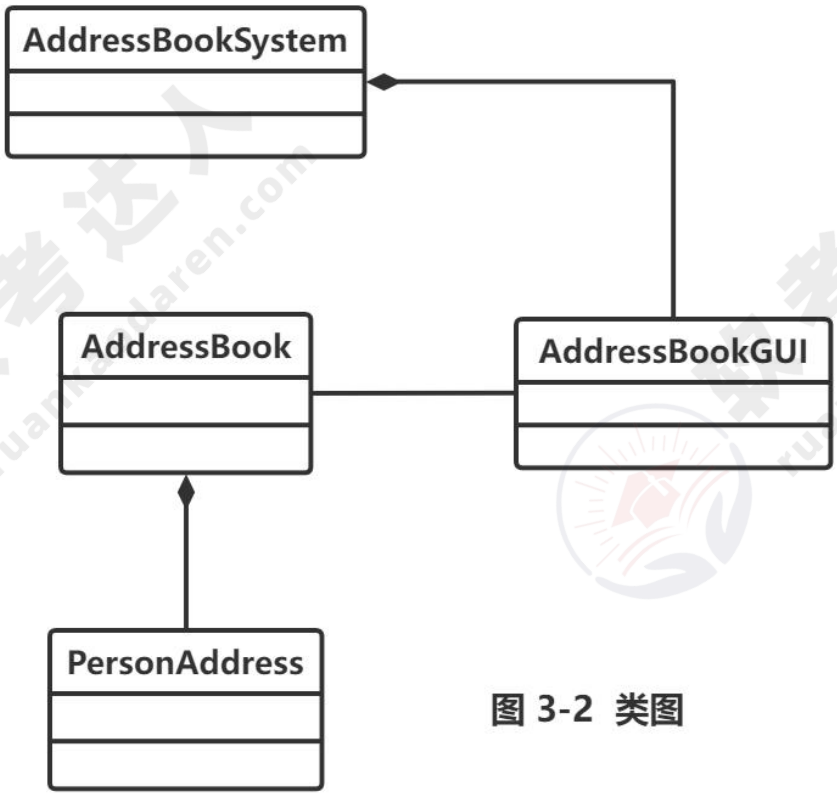


图 3-2 类图

【问题 1】（6 分）

根据说明中的描述，给出图 3-1 中 U1～U6 所对应的用例名。

答案不唯一，大家参考后自行判断～等之后出视频解析再解释

U1 和 U2 可以互换，加排序和不加排序这两个字应该都没问题。

答案一：

U1：按照员工姓氏的字典顺序排序

U2：按邮政编码排序

U3：创建地址簿

U4：打开地址簿

U5：修改地址簿

U6：保存地址簿

答案二：

U1：按照员工姓氏的字典顺序排序

U2：按邮政编码排序

U3：修改地址簿

U4：创建地址簿

U5：打开地址簿

U6：保存地址簿

答案三：

U1：按照员工姓氏的字典顺序排序

U2：按邮政编码排序

U3：打开地址簿

U4：修改地址簿

U5：保存地址簿

U6：创建地址簿

答案四：

U1：按照员工姓氏的字典顺序排序

U2：按邮政编码排序

U3：保存地址簿

U4：打开地址簿

U5：修改地址簿

U6：创建地址簿

答案五：

U1：按照员工姓氏的字典顺序排序

U2：按邮政编码排序

U3：创建地址簿

U4：修改地址簿

U5：打开地址簿

U6：保存地址簿

答案六：

U1：按照员工姓氏的字典顺序排序

U2：按邮政编码排序

U3：创建地址簿

U4：打开地址簿

U5：修改地址簿

U6：GUI（图形用户界面）

【问题 2】（5 分）

根据说明中的描述，给出图 3-2 中类 AddressBook 的主要属性和方法以及类 PersonAddress 的主要属性（可以使用说明中的文字）

AddressBook 的主要属性包括：部门号（名称、编码等）、姓名、住址、城市、省份、邮政编码以及联系电话等

AddressBook 的方法包括：添加地址记录、修改地址记录、删除地址记录、排序地址记录、打印地址记录、创建地址簿、打开地址簿、修改地址簿

PersonAddress 的主要属性包括：姓名、住址、城市、省份、邮政编码以及联系电话等信息

【问题 3】（4 分）

根据说明中的描述以及图 3-1 所示的用例图，请简要说明 include 和 extend 关系的含义是什么？

include 表示包含关系，含义为：如果系统用例较多，不同的用例之间存在共同行为，可以将这些共同行为提取出来，单独组成一个用例。当其他用例使用这个用例时，它们就构成了包含关系。

extend 表示扩展关系，含义为：在用例的执行过程中，可能出现一些异常行为，也可能会在不同的分支行为中选择执行，这时可将异常行为与可选分支抽象为一个单独的扩展用例，这样扩展用例与主用例之间就构成了扩展关系。一个用例常常有多个扩展用例。

这里包含和扩展的含义不唯一的，大家可以挑自己觉得最合理的进行记忆。然后就是根据说明中的描述和图 3-1 所示的用例图来说明 include 和 extend，应为问题一的答案不是很确定，这里也只能给大家将包含和扩展的含义写出来。也是等之后出视频解析，再进行解释。

试题四（共 15 分）

阅读下列说明和 C 代码，回答问题 1 至问题 3，将解答填入答题纸的对应栏内。

【说明】

某工程计算中要完成多个矩阵相乘（链乘）的计算任务，对矩阵相乘进行以下说明。

（1）两个矩阵相乘要求第一个矩阵的列数等于第二个矩阵的行数，计算量主要由进行乘法运算的次数决定。假设采用标准的矩阵相乘算法，计算$A_{n*m} * B_{n*p}$，需要$m * n * p$次乘法运算，即时间复杂度为$O(m * n * p)$

（2）矩阵相乘满足结合律，多个矩阵相乘时不同的计算顺序会产生不同的计算量。以矩阵$A1_{5*100}$，$A2_{100*8}$，$A3_{8*50}$三个矩阵相乘为例，若按$(A1 * A2) * A3$ 计算，则需要进行 $5 * 100 * 8 + 5 * 8 * 50 = 6000$ 次乘法运算，若按$A1 * (A2 * A3)$计算，则需要进行 $100 * 8 * 50 + 5 * 100 * 50 = 65000$ 次乘法运算。

矩阵连乘问题可描述为：给定n个矩阵，对较大的n，可能计算的顺序数量非常庞大，用蛮力法确定计算顺序是不实际的。经过对问题进行分析，发现矩阵连乘问题具有最优子结构，即若$A1 * A2 * \cdots * An$的一个最优计算顺序从第k个矩阵处断开，即分为$A1 * A2 * \cdots * Ak$和$Ak + 1 * Ak + 2 * \cdots * An$两个子问题，则该最优解应该包含$A1 * A2 * \cdots * Ak$的一个最优计算顺序和$Ak + 1 * Ak + 2 * \cdots * An$的一个最优计算顺序。据此构造递归式：

$$\text{cost}[i][j] = \begin{cases} 0 & \text{if i=j} \\ \min_{i \le k < j} \text{cost[i][k]} + \text{cost[k+1][j]} + p_i * p_{k+1} * p_{j+1} & \text{if i<j} \end{cases}$$

其中，cost[i][j]表示$Ai + 1 * Ai + 2 * \cdots * Aj + 1$ 的最优计算的代价。最终需要求解 cost[0][n - 1]

【C 代码】

算法实现采用自底向上的计算过程。首先计算两个矩阵相乘的计算量，然后依次计算 3 个矩阵、4 个矩阵、…、n个矩阵相乘的最小计算量及最优计算顺序。下面是算法的 C 语言实现。

（1）主要变量说明

n：矩阵数、

seq[]：矩阵维数序列

cost[][]：二维数组，长度为$n * n$，其中元素 cost[i][j]表示$Ai + 1 * Ai + 2 * \cdots *$

$Aj+1$ 的最优计算的计算代价

trace[][]：二维数组，长度为$n*n$，其中元素 trace[i][j]表示$Ai+1*Ai+2*\cdots*Aj+1$ 的最优计算对应的划分位置，即k

（2）函数 cmm

```
#define N 100
int cost[N][N];
int trace[N][N];
int cmm(int n, int seq[]) {
    int tempCost;
    int tempTrace;
    int i, j, k, p;
    int temp;
    for (i = 0; i < n; i ++ ) { cost[i][i] = 0; }
        for (p = 1; p < n; p ++ ) {
            for (i = 0; i < n - p; i ++ ) {
                __(1)__;
                tempCost = -1;
                for (k = i; __(2)__ ; k ++ ) {
                    temp = __(3)__;
                    if(tempCost == -1 || tempCost > temp) {
                        tempCost = temp;
                        tempTrace = k;
                    }
                }
                cost[i][j] = tempCost;
                __(4)__;
            }
        }
    return cost[0][n - 1];
```

}

【问题 1】（8 分）

根据以上说明和 C 代码，填充 C 代码中的空（1）～（4）。

（1）：j = i + p

（2）：k < j

（3）：cost[i][k] + cost[k + 1][j] + seq[i] * seq[k + 1] * seq[j + 1]

（4）：trace[i][j] = tempTrace

【问题 2】（4 分）

根据以上说明和 C 代码，该问题采用了__（5）__算法设计策略，时间复杂度为__（6）__。（用 O 符号表示）

（5）：动态规划

（6）：$O(n^3)$

【问题 3】（3 分）

考虑实例 n = 4，各个矩阵的维数：A1 为 15 * 5，A2 为 5 * 10，A3 为 10 * 20，A4 为 20 * 25，即维数序列为 15，5，10，20，25。则根据上述 C 代码得到的一个最优计算顺序为__（7）__（用加括号方式表示计算顺序），所需要的乘法运算次数为__（8）__。

（7）：A1 * （（A2 * A3） * A4） 或 A1 * （A2 * A3 * A4）

或：A1 （（A2 A3） A4） 或 A1 （A2 A3 A4）

（8）：5375

试题五（共 15 分）

阅读下列说明和 C++代码，将应填入___（n）___处的字句写在答题纸的对应栏内。

【说明】

在软件系统中，通常都会给用户提供取消、不确定或者错误的操作，允许将系统回复到原先的状态。现使用备忘录（Memento）模式实现该要求，得到如图 5-1 所示的类图。

Memento 包含了要被恢复的状态。Originator 创建并在 Memento 中存储状态。CareTaker 负责从 Memento 中恢复状态。

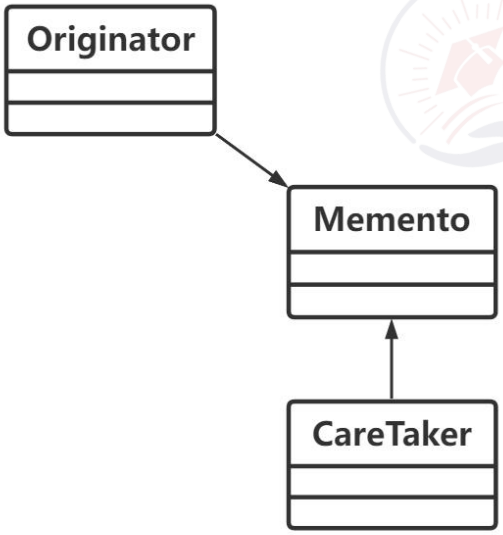


图 5-1 类图

【C++代码】

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;
```

```
class Memento {
    private:
        string state;

    public:
        Memento(string state) { this->state = state; }
        string getState() { return state; }
};

class Originator {
    private:
        string state;

    public:
        void setState(string state) { this->state = state; }
        string getState() { return state; }
        Memento saveStateToMemento() { return __(1)__; }
        void getStateFromMemento(Memento Memento) { state = __(2)__; }
};

class CareTaker {
    private:
        vector<Memento> mementoList;
    public:
        void __(3)__ { mementoList.push_back(state); }
        __(4)__ { return mementoList[index]; }
};
```

```
int main() {
    Originator *originator = new Originator();
    CareTaker *careTaker = new CareTaker();
    originator->setState("State #1");
    originator->setState("State #2");
    careTaker->add(__(5)__);
    originator->setState("State #3");
    careTaker->add(__(6)__);
    originator->setState("State #4");

    cout << "Current State: " + originator->getState() << endl;
    originator->getStateFromMemento(careTaker->get(0));
    cout << "First saved State: " + originator->getState() << endl;
    originator->getStateFromMemento(careTaker->get(1));
    cout << "Second saved State: " + originator->getState() << endl;
}
```

(1): Memento(state) 或 Memento(this->state) 或 Memento(getState())

(2): Memento.getState()

(3): void add(Memento state)

(4): Memento get(int index)

(5): originator->saveStateToMemento()

(6): originator->saveStateToMemento()

试题六（共 15 分）

阅读下列说明和 Java 代码，将应填入___（n）___处的字句写在答题纸的对应栏内。

【说明】

在软件系统中，通常都会给用户提供取消、不确定或者错误的操作，允许将系统回复到原先的状态。现使用备忘录（Memento）模式实现该要求，得到如图 6-1 所示的类图。

Memento 包含了要被恢复的状态。Originator 创建并在 Memento 中存储状态。CareTaker 负责从 Memento 中恢复状态。

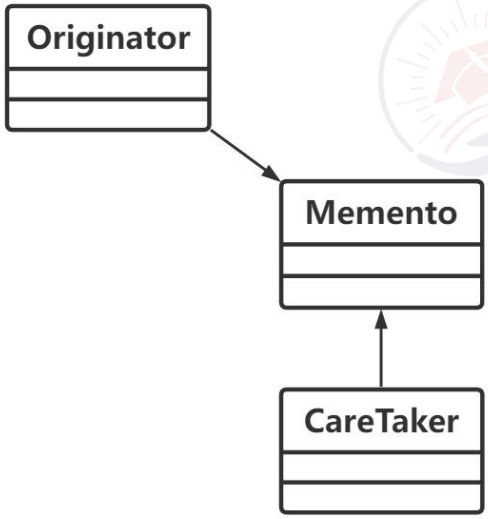


图 6-1 类图

【Java 代码】

```
import java.util.*;

class Memento {
    private String state;
    public Memento(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }
}
```

```
class Originator {
    private String state;
    public void setState(String state) {
        this.state = state;
    }

    public String getState() {
        return state;
    }

    public Memento saveStateToMemento() {
        return __(1)__;
    }

    public void getStateFromMemento(Memento Memento) {
        state = __(2)__;
    }
}

class CareTaker {
    private List<Memento> mementoList = new ArrayList<Memento>();

    public __(3)__ {
        mementoList.add(state);
    }

    public __(4)__ {
        return mementoList.get(index);
    }
}
```

```
class MementoPaneDemos {
    public static void main(String[] args) {
        Originator originator = new Originator();
        CareTaker careTaker = new CareTaker();
        originator.setState("State #1");
        originator.setState("State #2");
        careTaker.add(__(5)__);
        originator.setState("State #3");
        careTaker.add(__(6)__);
        originator.setState("State #4");

        System.out.println("Current State: " + originator.getState());
        originator.getStateFromMemento(careTaker.get(0));
        System.out.println("First saved State: " + originator.getState());
        originator.getStateFromMemento(careTaker.get(1));
        System.out.println("Second saved State: " + originator.getState());
    }
}
```

(1): new Memento(state) 或 new Memento(this.state)
或 new Memento(getState())

(2): Memento.getState()

(3): void add(Memento state)

(4): Memento get(int index)

(5): originator.saveStateToMemento()

(6): originator.saveStateToMemento()