

全国计算机技术与软件专业技术资格（水平）考试

2010 年下半年 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纸

- 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.仿照下面例题，将解答写在答题纸的对应栏内。

例题

2010 年下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“11 月 4 日”，故在答题纸的对应栏内写上“11”和“4”（参看下表）。

例题	解答栏
（1）	11
（2）	4

试题一

某时装邮购提供商拟开发订单处理系统，用于处理客户通过电话、传真、邮件或 Web 站点所下订单。其主要功能如下：

- (1) 增加客户记录。将新客户信息添加到客户文件，并分配一个客户号以备后续使用。
- (2) 查询商品信息。接收客户提交的商品信息请求，从商品文件中查询商品的价格和可订购数量等商品信息，返回给客户。
- (3) 增加订单记录。根据客户的订购请求及该客户记录的相关信息，产生订单并添加到订单文件中。
- (4) 产生配货单。根据订单记录产生配货单，并将配货单发送给仓库进行备货；备好货后，发送备货就绪通知。如果现货不足，则需向供应商订货。
- (5) 准备发货单。从订单文件中获取订单记录，从客户文件中获取客户记录，并产生发货单。
- (6) 发货。当收到仓库发送的备货就绪通知后，根据发货单给客户发货；产生装运单并发送给客户。
- (7) 创建客户账单。根据订单文件中的订单记录和客户文件中的客户记录，产生并发送客户账单，同时更新商品文件中的商品数量和订单文件中的订单状态。
- (8) 产生应收账户。根据客户记录和订单文件中的订单信息，产生并发送给财务部门应收账户报表。

现采用结构化方法对订单处理系统进行分析与设计，获得如图 1-1 所示的顶层数据流图和图 1-2 所示的 0 层数据流图。

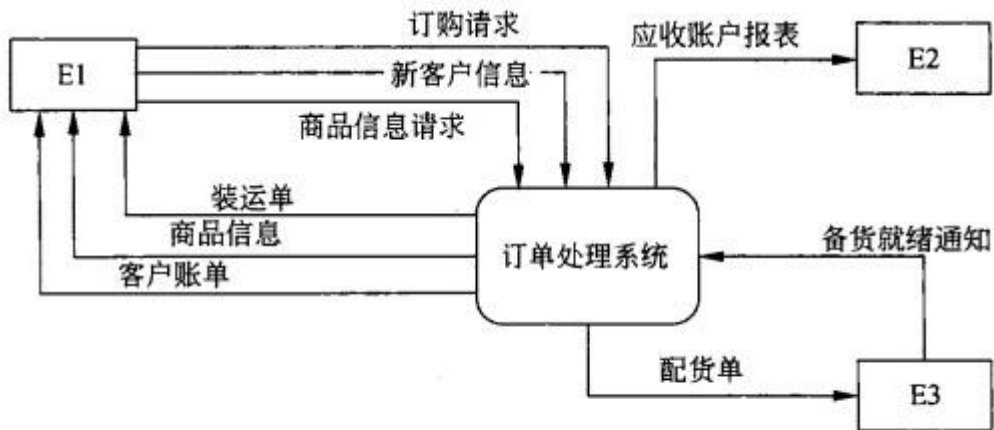


图 1-1 顶层数据流图

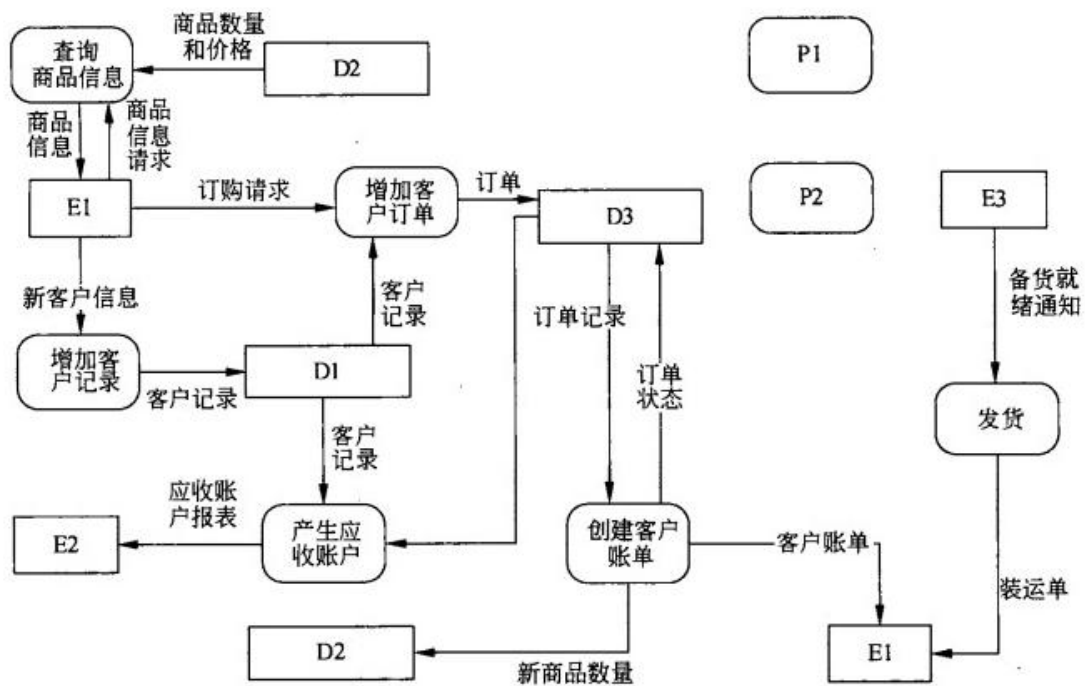


图 1-2 0层数据流图

【问题 1】

使用说明中的词语，给出图 1-1 中的实体 E1~E3 的名称。

【问题 2】

使用说明中的词语，给出图 1-2 中的数据存储 D1~D3 的名称。

【问题 3】

- (1) 给出图 1-2 中处理（加工）P1 和 P2 的名称及其相应的输入输出流。
- (2) 除加工 P1 和 P2 的输入输出流外，图 1-2 还缺失了 1 条数据流，请给出其起点和终点。

起 点	终 点

注：名称使用说明中的词汇，起点和终点均使用图 1-2 中的符号或词汇。

试题二

某公司拟开发一套小区物业收费管理系统。初步的需求分析结果如下：

（1）业主信息主要包括：业主编号、姓名、房号、房屋面积、工作单位、联系电话等。房号可唯一标识一条业主信息，且一个房号仅对应一套房屋；一个业主可以有一套或多套的房屋。

（2）部门信息主要包括：部门号、部门名称、部门负责人、部门电话等。一个员工只能属于一个部门，一个部门只有一位负责人。

（3）员工信息主要包括：员工号、姓名、出生年月、性别、住址、联系电话、所在部门号、职务和密码等。根据职务不同，员工可以有不同的权限：职务为“经理”的员工具有更改（添加、删除和修改）员工表中本部门员工信息的操作权限；职务为“收费”的员工只具有收费的操作权限。

（4）收费信息包括：房号、业主编号、收费日期、收费类型、数量、收费金额、员工号等。收费类型包括物业费、卫生费、水费和电费，并按月收取，收费标准如表 2-1 所示。其中：物业费=房屋面积（平方米）X 每平方米单价，卫生费=套房数量（套）X 每套房单价，水费=用水数量（吨）X 每吨水单价，电费=用电数量（度）X 每度电单价。

（5）收费完毕应为业主生成收费单，收费单示例如表 2-2 所示。

表 2-1 收费标准		
收费类型	单位	单价
物业费	平方米	1.00
卫生费	套	10.00
水 费	吨	0.70
电 费	度	0.80

表 2-2 收费单示例			
房号：A1608		业主姓名：李斌	
序号	收费类型	数量	金额
1	物业费	98.6	98.60
2	卫生费	1	10.00
3	水 费	6	4.20
4	电 费	102	81.60
合计	壹佰玖拾肆元肆角整		194.40
收费日期：2010-9-2			员工号：001

【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 中收费员和经理是员工的子实体。

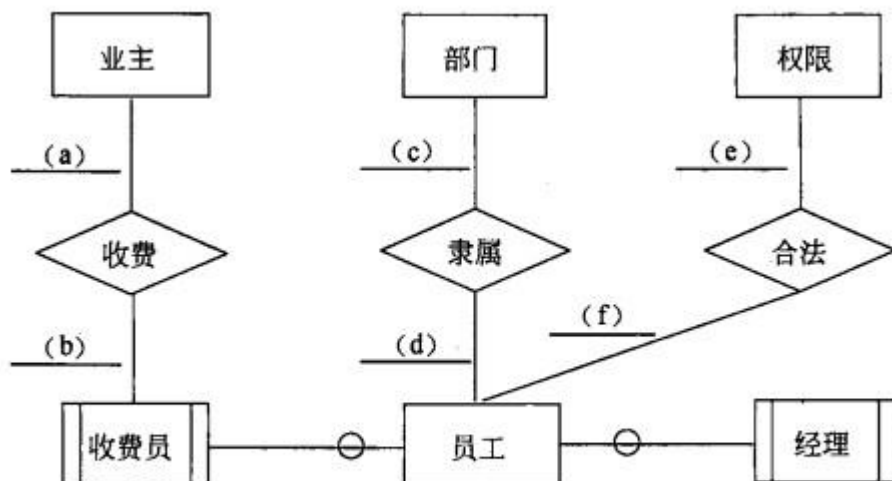


图 2-1 实体联系图

【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）：

业主（ _____（1）_____, 姓名, 房屋面积, 工作单位, 联系电话）

员工（ _____（2）_____, 姓名, 出生年月, 性别, 住址, 联系电话, 职务, 密码）

部门（ _____（3）_____, 部门名称, 部门电话）

权限（职务, 操作权限）

收费标准（ _____（4）_____ ）

收费信息（ _____（5）_____, 收费类型, 收费金额, 员工号）

【问题 1】

根据图 2-1, 将逻辑结构设计阶段生成的关系模式中的空（1）～（5）补充完整，然后给出各关系模式的主键和外键。

【问题 2】

填写图 2-1 中（a）～（f）处联系的类型（注：一方用 1 表示，多方用 m 或 n 或* 表示），并补充完整图 2-1 中的实体、联系和联系的类型。

【问题 2】

业主关系属于第几范式？请说明存在的问题。

试题三

某网上药店允许顾客凭借医生开具的处方，通过网络在该药店购买处方上的药品。该网上药店的基本功能描述如下：

(1) 注册。顾客在买药之前，必须先在网上药店注册。注册过程中需填写顾客资料以及付款方式（信用卡或者支付宝账户）。此外顾客必须与药店签订一份授权协议书，授权药店可以向其医生确认处方的真伪。

(2) 登录。已经注册的顾客可以登录到网上药房购买药品。如果是没有注册的顾客，系统将拒绝其登录。

(3) 录入及提交处方。登录成功后，顾客按照“处方录入界面”显示的信息，填写开具处方的医生的信息以及处方上的药品信息。填写完成后，提交该处方。

(4) 验证处方。对于已经提交的处方（系统将其状态设置为“处方已提交”），其验证过程为：

①核实医生信息。如果医生信息不正确，该处方的状态被设置为“医生信息无效”，并取消这个处方的购买请求；如果医生信息是正确的，系统给该医生发送处方确认请求，并将处方状态修改为“审核中”。

②如果医生回复处方无效，系统取消处方，并将处方状态设置为“无效处方”。如果医生没有在 7 天内给出确认答复，系统也会取消处方，并将处方状态设置为“无法审核”。

③如果医生在 7 天内给出了确认答复，该处方的状态被修改为“准许付款”。系统取消所有未通过验证的处方，并自动发送一封电子邮件给顾客，通知顾客处方被取消以及取消的原因。

(5) 对于通过验证的处方，系统自动计算药品的价格并邮寄药品给已经付款的顾客。该网上药店采用面向对象方法开发，使用 UML 进行建模。系统的类图如图 3-1 所示。

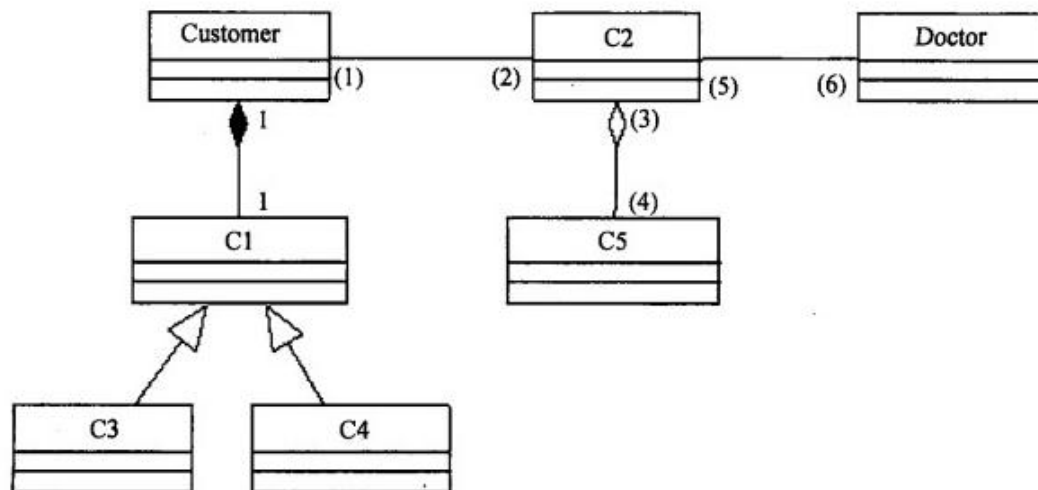


图 3-1 类图



【问题 1】

根据说明中的描述，给出图 3-1 中缺少的 C1~C5 所对应的类名以及 (1)~(6) 处所对应的多重度。

【问题 2】

图 3-2 给出了“处方”的部分状态图。根据说明中的描述，给出图 3-2 中缺少的 S1~S4 所对应的状态名以及 (7)~(10) 处所对应的迁移 (transition) 名。

【问题 3】

图 3-1 中的符号 “” 和 “” 在 UML 中分别表示类和对象之间的哪两种关系？两者之间的区别是什么？

试题四

堆数据结构定义如下：

对于 n 个元素的关键字序列 a_1, a_2, \dots, a_n ，当且仅当满足下列关系时称其为堆。

$$\begin{cases} a_i \leq a_{2i} \\ a_i \leq a_{2i+1} \end{cases} \quad \text{或} \quad \begin{cases} a_i \geq a_{2i} \\ a_i \geq a_{2i+1} \end{cases} \quad \text{其中, } i=1,2,\dots,\lfloor \frac{n}{2} \rfloor$$

在一个堆中，若堆顶元素为最大元素，则称为大顶堆；若堆顶元素为最小元素，则称为小顶堆。堆常用完全二叉树表示，图 4-1 是一个大顶堆的例子。

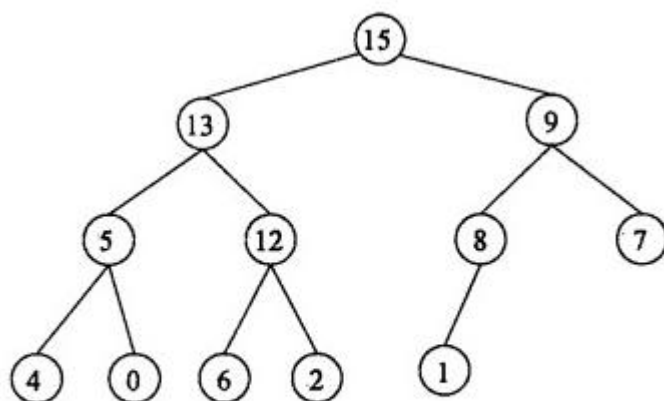


图 4-1 大顶堆示例

堆数据结构常用于优先队列中，以维护由一组元素构成的集合。对应于两类堆结构，优先队列也有最大优先队列和最小优先队列，其中最大优先队列采用大顶堆，最小优先队列采用小顶堆。以下考虑最大优先队列。

假设现已建好大顶堆 A ，且已经实现了调整堆的函数 $\text{heapify}(A, n, \text{index})$ 。

下面将 C 代码中需要完善的三个函数说明如下：

(1) $\text{heapMaximum}(A)$ ：返回大顶堆 A 中的最大元素。

(2) $\text{heapExtractMax}(A)$ ：去掉并返回大顶堆 A 的最大元素，将最后一个元素“提前”到堆顶位置，并将剩余元素调整成大顶堆。

(3) $\text{maxHeapInsert}(A, \text{key})$ ：把元素 key 插入到大顶堆 A 的最后位置，再将 A 调整成大顶堆。

优先队列采用顺序存储方式，其存储结构定义如下：

```

#define PARENT(i) i/2
typedef struct array{
    int *int_array; //优先队列的存储空间首地址
    int array_size; //优先队列的长度
    int capacity; //优先队列存储空间的容量
} ARRAY;

```

【C 代码】

(1) 函数 heapMaximum

```
int heapMaximum(ARRAY *A){ return ____ (1) ____; }
```

(2) 函数 heapExtractMax

```

int heapExtractMax(ARRAY *A){
    int max;
    max = A->int_array[0];
    ____ (2) ____;
    A->array_size --;
    heapify(A,A->array_size,0); //将剩余元素调整成大顶堆
    return max;
}

```

(3) 函数 maxHeapInsert

```

int maxHeapInsert(ARRAY *A,int key){
    int i,*p;
    if (A->array_size == A->capacity) { //存储空间的容量不够时扩充空间
        p = (int*)realloc(A->int_array, A->capacity *2 * sizeof(int));
        if (!p) return -1;
        A->int_array = p;
        A->capacity = 2 * A->capacity;
    }
    A->array_size ++;
}

```

```

        heapify(A,A->array_size,0); //将剩余元素调整成大顶堆
    return max;
}

(3) 函数 maxHeapInsert

int maxHeapInsert (ARRAY *A,int key){
    int i,*p;
    if (A->array_size == A->capacity) { //存储空间的容量不够时扩充空间
        p = (int*)realloc(A->int_array, A->capacity *2 * sizeof(int));
        if (!p) return -1;
        A->int_array = p;
        A->capacity = 2 * A->capacity;
    }
    A->array_size ++;
    i = (3);
    while (i > 0 && (4)){
        A->int_array[i] = A->int_array[PARENT(i)];
        i = PARENT(i);
    }
    (5);
    return 0;
}

```

【问题 1】

根据以上说明和 C 代码，填充 C 代码中的空 (1) ~ (5)。

【问题 2】

根据以上 C 代码，函数 heapMaximum、heapExtractMax 和 maxHeapInsert 的时间复杂度的紧致上界分别为 (6)、(7) 和 (8) (用 O 符号表示)。

【问题 3】

若将元素 10 插入到堆 $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$ 中，调用 maxHeapInsert 函数进行操作，则新插入的元素在堆 A 中第 (9) 个位置 (从 1 开始)。

从下列的 2 道试题（试题五至试题六）中任选 1 道解答。
如果解答的试题数超过 1 道，则题号小的 1 道解答有效。

试题五

某公司的组织结构图如图 5-1 所示，现采用组合（Composition）设计模式来构造该公司的组织结构，得到如图 5-2 所示的类图。

其中 Company 为抽象类，定义了在该组织结构图上添加（Add）和删除（Delete）分公司/办事处或者部门的方法接口。类 ConcreteCompany 表示具体的分公司或者办事处，分公司或办事处下可以设置不同的部门。类 HRDepartment 和 FinanceDepartment 分别表示人力资源部和财务部。

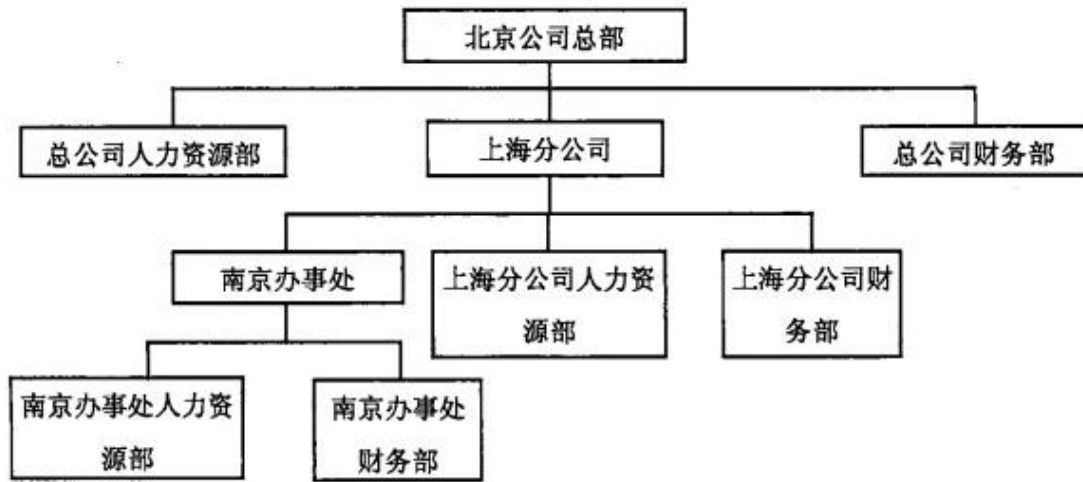


图 5-1 组织结构图

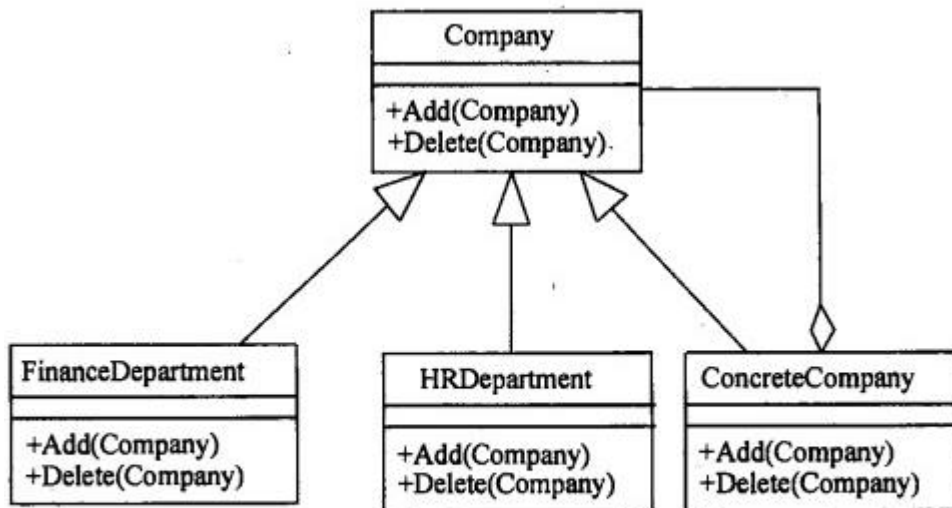


图 5-2 类图

【问题 1】

【C++代码】

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

class Company {                                // 抽象类
protected:
    string name;
public:
    Company(string name) {     (1)     = name; }
        (2)    ;                // 增加子公司、办事处或部门
        (3)    ;                // 删除子公司、办事处或部门
};

class ConcreteCompany : public Company {
private:
    list<     (4)     > children;                // 存储子公司、办事处或部门
public:
    ConcreteCompany(string name) : Company(name) { }
    void Add(Company* c) {     (5)    .push_back(c); }
    void Delete(Company* c) {     (6)    .remove(c); }
};

class HRDepartment : public Company {
public:
    HRDepartment(string name) : Company(name) {}    // 其他代码省略
};

class FinanceDepartment : public Company {
public:
    FinanceDepartment(string name) : Company(name) {}    // 其他代码省略
};

void main() {
    ConcreteCompany *root = new ConcreteCompany("北京总公司");
```

```

root->Add(new HRDepartment("总公司人力资源部"));
root->Add(new FinanceDepartment("总公司财务部"));

ConcreteCompany *comp = new ConcreteCompany("上海分公司");
comp->Add(new HRDepartment("上海分公司人力资源部"));
comp->Add(new FinanceDepartment("上海分公司财务部"));
    (7) ;

ConcreteCompany *comp1 = new ConcreteCompany("南京办事处");
comp1->Add(new HRDepartment("南京办事处人力资源部"));
comp1->Add(new FinanceDepartment("南京办事处财务部"));
    (8) ;    //其他代码省略
}

```

试题六

某公司的组织结构图如图 6-1 所示，现采用组合（Composition）设计模式来设计，得到如图 6-2 所示的类图。

其中 Company 为抽象类，定义了了在组织结构图上添加（Add）和删除（Delete）分公司/办事处或者部门的方法接口。类 ConcreteCompany 表示具体的分公司或者办事处，分公司或办事处下可以设置不同的部门。类 HRDepartment 和 FinanceDepartment 分别表示人力资源部和财务部。

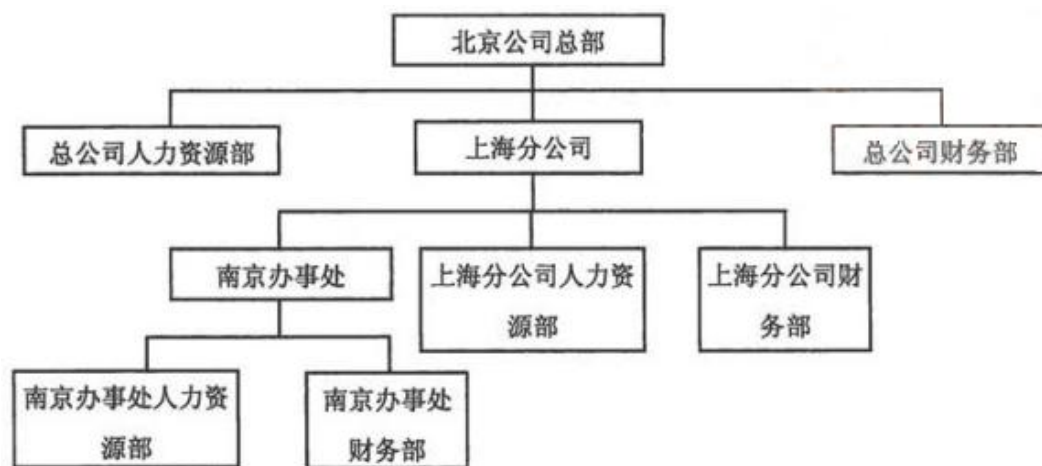


图 6-1 组织结构图

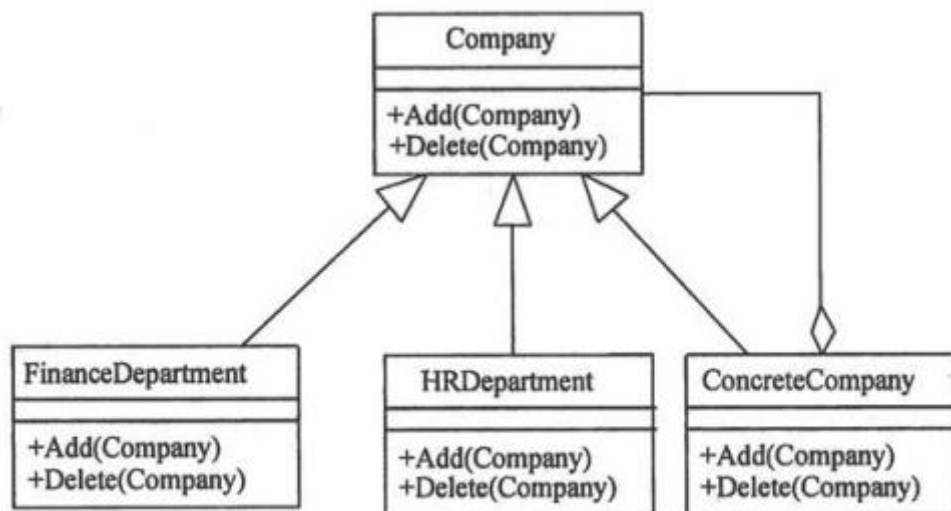


图 6-2 类图

【问题 1】

【Java 代码】

```
import java.util.*;

(1) Company {
    protected String name;
    public Company(String name) { (2) = name; }
    public abstract void Add(Company c);        // 增加子公司、办事处或部门
    public abstract void Delete(Company c);     // 删除子公司、办事处或部门
}

class ConcreteCompany extends Company {
    private List< (3) > children = new ArrayList< (4) >();
    // 存储子公司、办事处或部门
    public ConcreteCompany(String name) { super(name); }
    public void Add(Company c) { (5).add(c); }

    public void Delete(Company c) { (6).remove(c); }
}

class HRDepartment extends Company {
    public HRDepartment(String name) { super(name); }
    // 其他代码省略
}

class FinanceDepartment extends Company {
    public FinanceDepartment(String name) { super(name); }
    // 其他代码省略
}

public class Test {
    public static void main(String[] args) {
        ConcreteCompany root = new ConcreteCompany("北京总公司");
        root.Add(new HRDepartment("总公司人力资源部"));
        root.Add(new FinanceDepartment("总公司财务部"));

        ConcreteCompany comp = new ConcreteCompany("上海分公司");
        comp.Add(new HRDepartment("上海分公司人力资源部"));
        comp.Add(new FinanceDepartment("上海分公司财务部"));
        (7);
    }
}
```



```
ConcreteCompany compl = new ConcreteCompany("南京办事处");  
compl.Add(new HRDepartment("南京办事处人力资源部"));  
compl.Add(new FinanceDepartment("南京办事处财务部"));  
_____(8)____;      // 其他代码省略  
}  
}
```