全国计算机技术与软件专业技术资格（水平）考试

中级 软件设计师 2010 年 下半年 下午试卷 案例

（考试时间 150 分钟）

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。

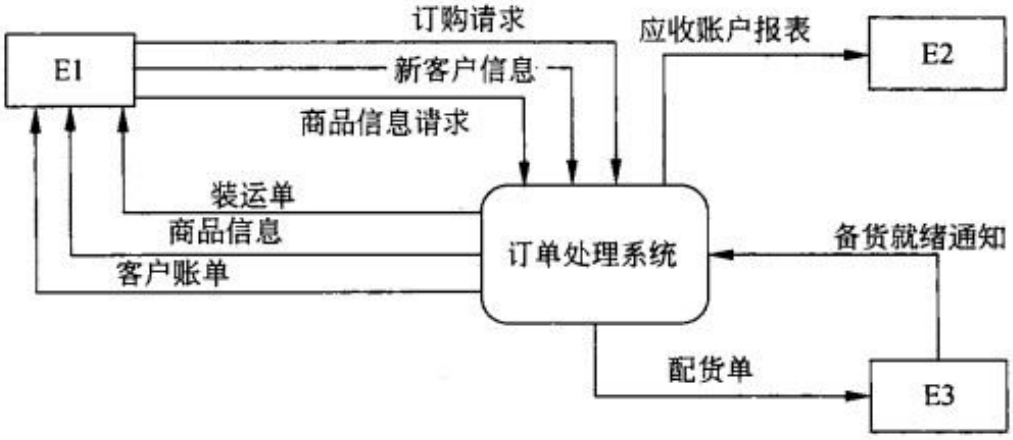


图 1-1　顶层数据流图

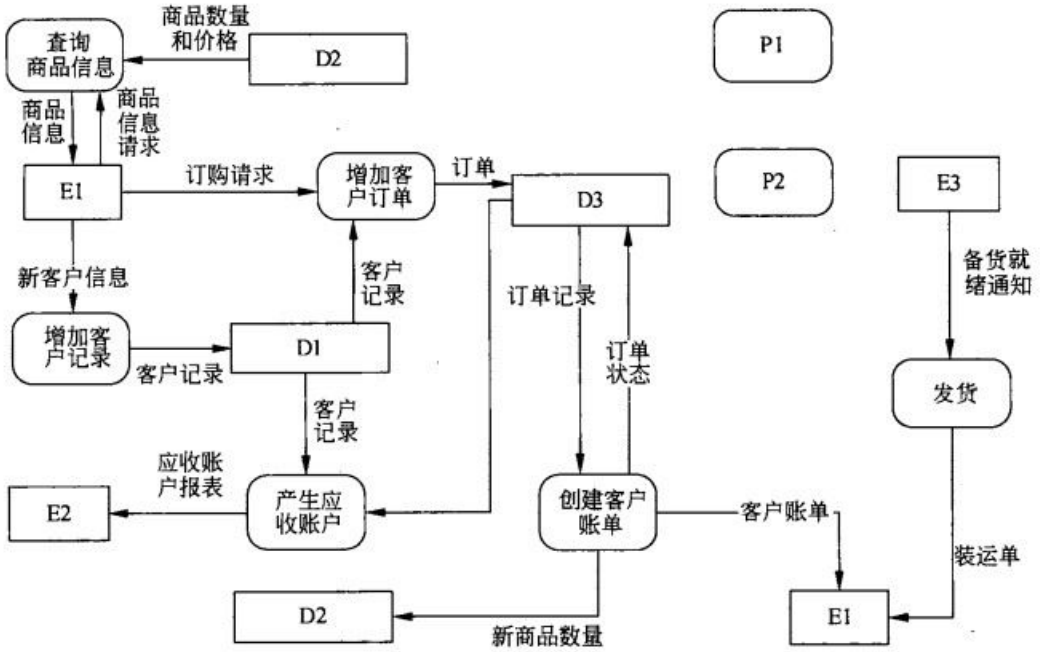


图 1-2　0 层数据流图

问题：1.1

使用说明中的词语，给出图 1-1 中的实体 E1　E3 的名称。

问题：1.2

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：1.3

(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。

【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。

【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)：

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

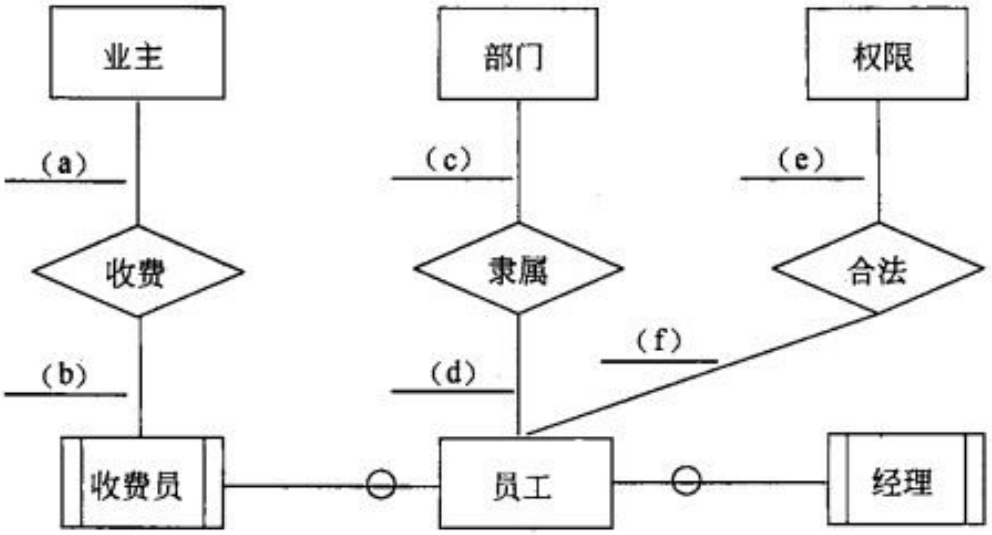


图 2-1　实体联系图

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：2.1

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：2.2
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：2.3
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。

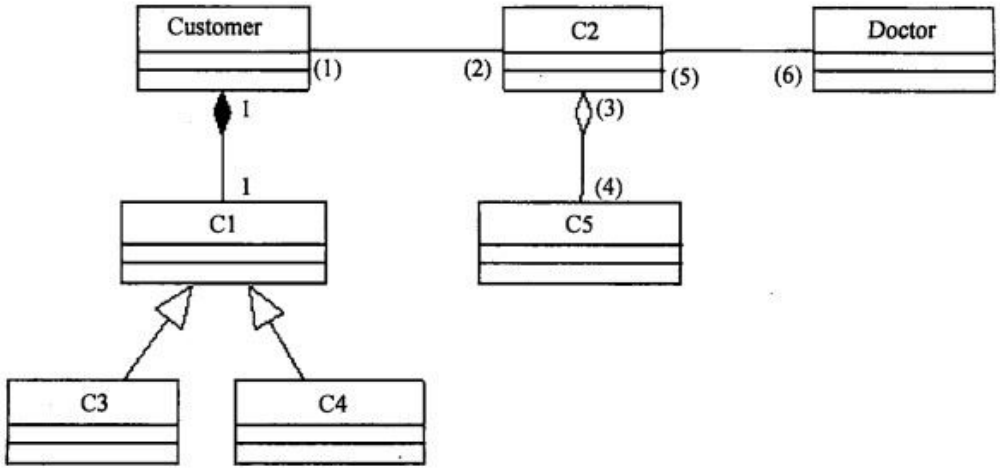


图 3-1 类图

问题： 3.1

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： 3.2

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： 3.3

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，…， a_n}，当且仅当满足下列关系时称其为堆。

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。

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 heapify(A， n， index)。

下面将 C 代码中需要完善的三个函数说明如下：

(1) heapMaximum(A)：返回大顶堆 A 中的最大元素。

(2) 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) maxHeapInsert(A, 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:

$$\begin{cases} a_i \leqslant a_{2i} \\ a_i \leqslant a_{2i+1} \end{cases} \quad 或 \quad \begin{cases} a_i \geqslant a_{2i} \\ a_i \geqslant a_{2i+1} \end{cases} \quad 其中，\ i=1,2,\cdots,\left\lfloor n/2 \right\rfloor$$

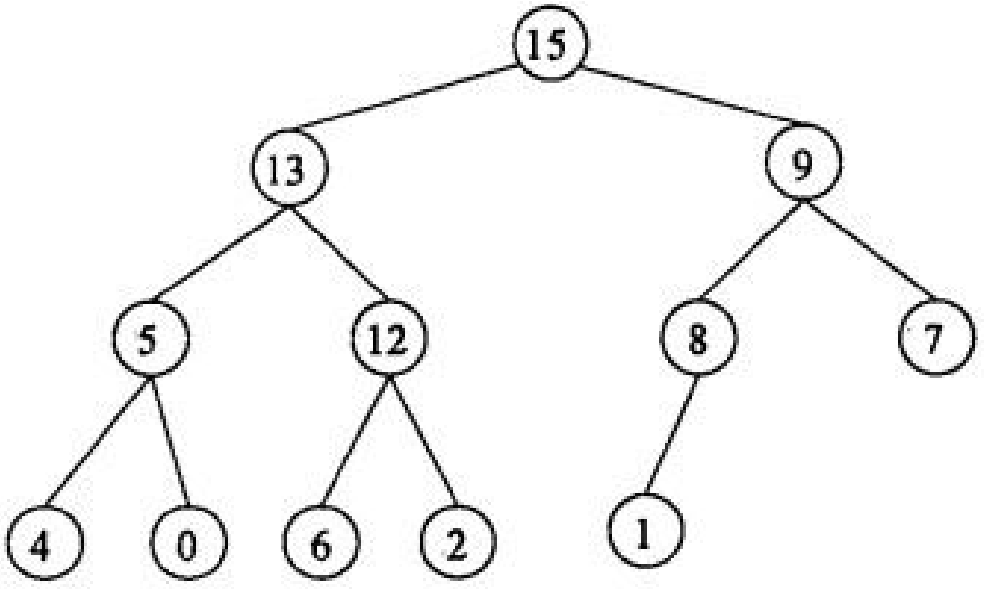


图 4-1　大顶堆示例

```
#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：4.1

根据以上说明和 C 代码，填充 C 代码中的空(1)（5)。

问题：4.2

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：4.3

若将元素 10 插入到堆 A =<15，13，9，5，12，8，7,4，0，6,2，1>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)。

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。

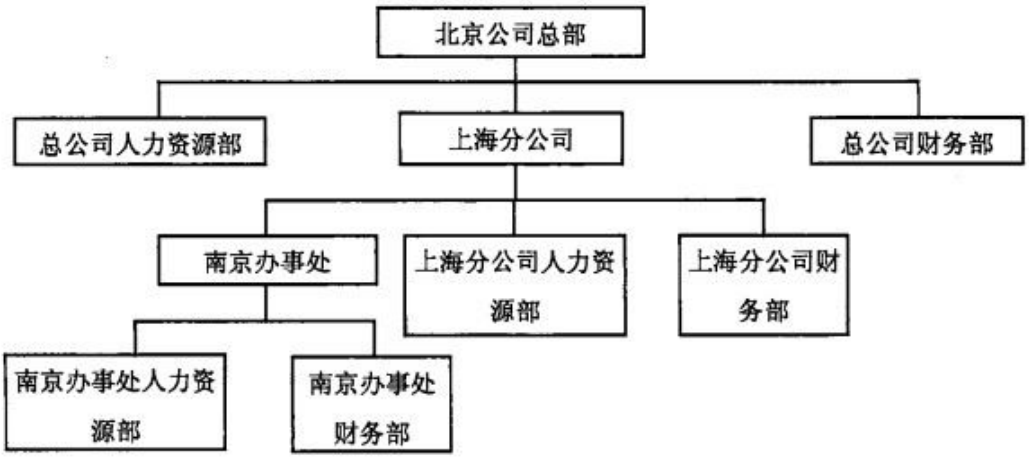


图 5-1　组织结构图

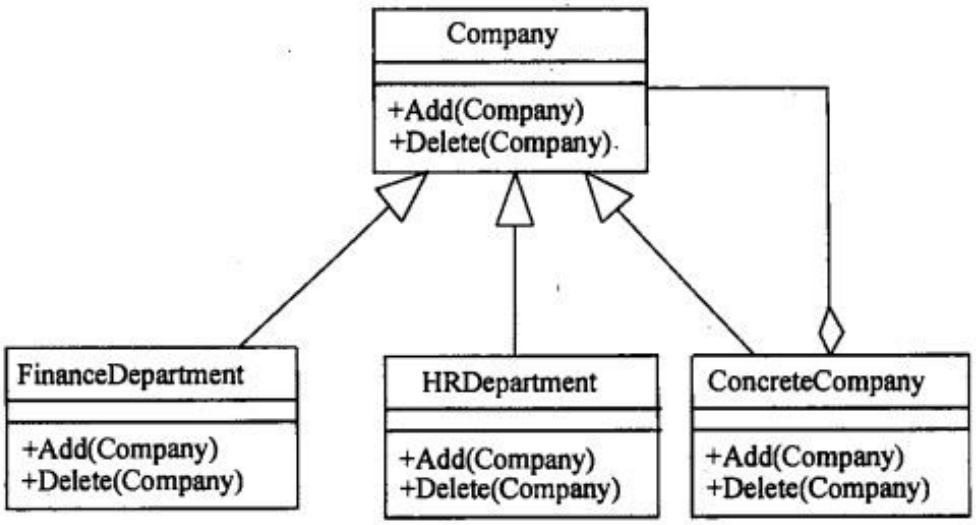


图 5-2　类图

问题：5.1

【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。

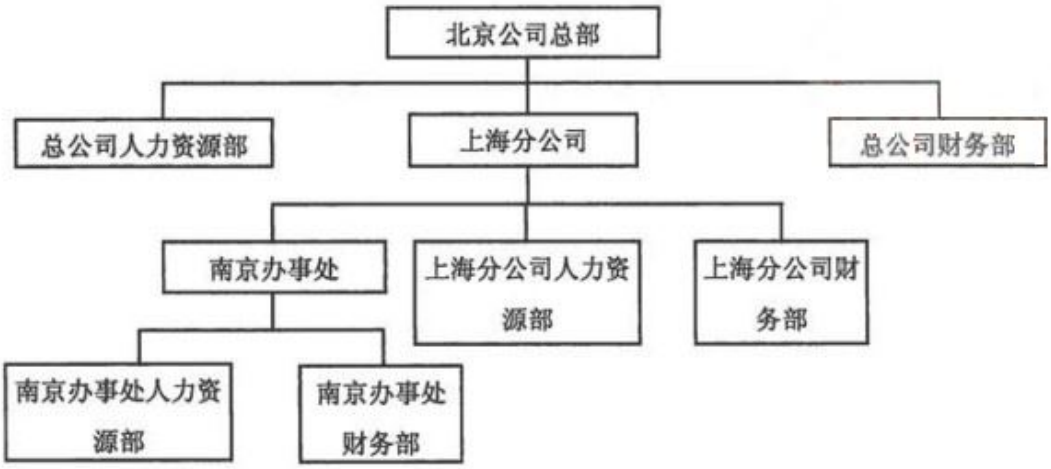


图 6-1　组织结构图

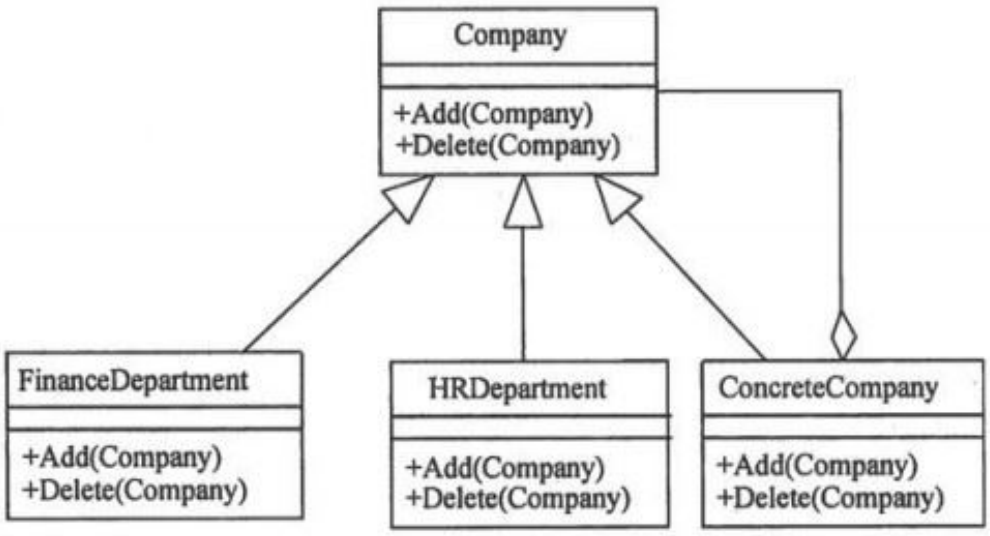


图 6-2　类图

问题：6.1

【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 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
    }
  }
```

试题一 答案： 解析： E1 ： 客户 E2:财务部门 E3:仓库

本问题考查顶层 DFD 。顶层 DFD 一般用来确定系统边界，将待开发系统看作一个加工，因此图中只有唯一的一个处理和一些外部实体，以及这两者之间的输入输出数据流。题目要求根据描述确定图中的外部实体。根据题目中的描述，并结合已经在顶层数据流图中给出的数据流进行分析。从题目的说明中可以看出：客户提交商品信息请求、订购请求等；将配货单发送给仓库、仓库向系统发送备货就绪通知；发送给财务部门应收账户报表。由此可知该订单系统有客户、仓库和财务部门三个外部实体。对应图 1-1 中数据流和实体的对应关系，可知 E1 为客户， E2 为财务部门， E3 为仓库。本题中需注意说明(4)中向供应商订货是系统外部的行为，因此，供应商并非本系统的外部实体。
D1 ： 客户文件 D2:商品文件 D3 ： 订单文件

本问题考查 0 层 DFD 中数据存储的确定。根据说明中的以下描述：将新客户信息添加到客户文件；从商品文件中查询商品的价格和可订购数量等商品信息；产生订单并添加到订单文件中，得出数据存储为客户文件、商品文件以及订单文件，再根据图 1-2 中 D1 的输入和输出数据流均为客户记录， D2 的输入数据流为从处理“创建客户账单”来的新商品数量，输出数据流为到处理“查询商品信息”的商品数量和价格， D3 的输入数据流为从处理“增加客户订单”来的订单，可知， D1 为客户文件， D2 为商品文件， D3 为订单文件。
(1) 处理(加工)名称，数据流。

上表中各行次序无关，但每条数据流的名称、起点、终点必须相对应。
P1 和 P2 可互换，即 P1 为“准备发货单”、 P2 为“产生配货单”。

本问题考查 0 层 DFD 中缺失的处理和数据流。从说明中的描述功能和图 1-2, 可知 产生配货单和准备发货单没有在图 1-2 中，即缺少两个处理：产生配货单和准备发货单。 根据说明(4)中的描述：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。可知，产生配货单的输入流为订单记录，该输入流的起 点为订单文件(D3)，输出流为配货单，其终点为仓库(E3)。根据说明(5) 中的描述: 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。可知，准备发货单的输入流为订单记录和客户记录，订单记录的起点为订单文件，客户记录的起点为客户文件；输出流为发货单。再根据说明(6) 中处理发货的描述：根据发货单给客户发货，发货单的终点为处理发货。产生配货单和准备发货单分别对应 P1 和 P2 （或 P2 和 P1)。

P1 和 P2 及其输入输出流均识别出来之后，再对照说明和图 1-2, 以找出缺少的另外一条数据流。对照说明(7) 中的描述：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。因此，创建客户账单缺少一条输入流：客户记录，其起点为客户文件(D1)。

P1: 产生配货单　　P2: 准备发货单

数据流名称	起　点	终　点
订单记录	D3 或 订单文件	P1 或 产生配货单
配货单	P1 或 产生配货单	E3 或 仓库
订单记录	D3 或 订单文件	P2 或 准备发货单
客户记录	D1 或 客户文件	P2 或 准备发货单
发货单	P2 或 准备发货单	发货

起　点	终　点
D1 或 客户文件	创建客户账单

试题二 答案：　解析：

根据题意，业主关系中信息主要包括：业主编号、姓名、房号、房屋面积、工作单位、联系电话等，因此，空(1) 应填写“业主编号，房号”。又因为房号可唯一标识一 条业主信息，所以以“房号”为主键。完整的关系模式如下：

业主(业主编号，房号，姓名，房屋面积，工作单位，联系电话)

根据题意，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，因此，空(2) 应填写“员工号，所在部门号”。又因为员工号可唯一标识一条员工信息，所以“员工号”为主键。根据题意，一个员工只能属于一个部门，“所在部门号”应参照部门关系的“部门号”，因此，“所在部门号”为外键。 完整的关系模式如下：

员工(员工号，所在部门号，姓名，出生年月，性别，住址，联系电话，职务，密码)

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，因此，部门关 系的空(3) 应填写“部门号，部门负责人”，显然该关系的主键为“部门号”。又因为部门关系的“部门负责人”应参照员工关系的“员工号”，因此，“部门负责人”为外键。

根据题意分析收费标准关系的空(4) 应填写“收费类型，单位，单价”，这样收费 信息关系可以根据收费类型(如水费、电费或物业费)去收费标准关系中查出单价来计算收费金额。显然收费标准关系的主键为“收费类型”。

收费信息的空(5) 应填写“房号，业主编号，收费日期”，由于“房号，业主编号， 收费

日期”能唯一确定该关系的每一个元组，故“房号，业主编号，收费日期”为关系的主键。又由于房号、员工号分别为业主和员工关系的主键，故“房号，员工号”为收费信息关系的外键。完整的关系模式如下：
收费信息(房号，业主编号，收费日期，收费类型，收费金额，员工号)

根据题意，一个员工可以为多个业主收费，同样一个业主也可以有多个员工为其收费，因此业主和收费员之间的收费联系为多对多。故空(a)应填写*，空(b)应填写*。
因为一个员工只能属于一个部门，所以部门与员工之间的隶属联系是一对多的。故空(c)应填写 1，空(d)应填写*。
根据题意，职务不同员工可以有不同的权限，所以权限和员工之间的合法联系是一对多。又由于收费员收费时必需根据收费类型(如水费、电费或物业费)到收费标准关系中查出单价来计算收费金额，所以需要增加一个收费标准关系，以及收费标准到收费联系的连线。

业主关系属于第 2 范式。
问题是当某业主有多套住房时，属性“业主编号，姓名，房屋面积，工作单位，联系电话”等信息在业主关系表中重复存储，存在数据冗余。

由业主关系可知：房号一业主编号，业主编号一姓名，房号一姓名，所以存在传递依赖房号一姓名。故业主关系属于第 2 范式。业主关系存在的问题是当某业主有多套住房时，属性“业主编号，姓名，房屋面积，工作单位，联系电话”等信息在业主关系表中重复存储，存在数据冗余。

(1) 业主编号，房号
主键：房号　　外键：无
(2) 员工号，所在部门号
主键：员工号　　外键：所在部门号
(3) 部门号，部门负责人
主键：部门号　　外键：部门负责人
(4) 收费类型，单位，单价
主键：收费类型　　外键：无
(5) 房号，业主编号，收费日期
主键：房号，业主编号，收费日期　外键：房号，员工号

(a) n，或 m，或*

(b) n，或 m，或*

(c) 1

(d) n，或 m，或*

(e) 1

(f) n，或 m，或*

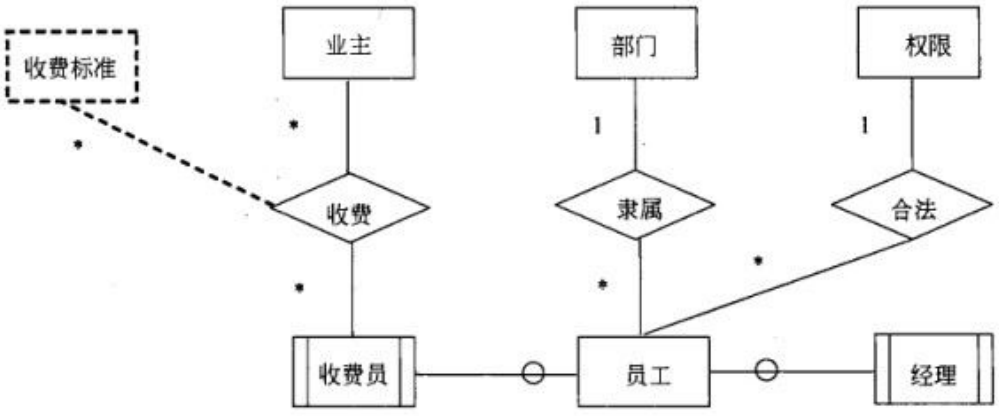


图 2-1　补充完整的实体联系图

试题三 答案：　解析：

本问题考查 UML 的类图。类图展现了一组对象、接口、协作和它们之间的关系。在面向对

象系统的建模中，最常用的模型之一就是类图。

类图用于对系统的静态设计视图建模。这种视图主要支持系统的功能需求，即系统要提供给用户的服务。但对系统的静态设计视图建模时，通常有三种使用方式:

(1）对系统的词汇建模

对系统的词汇建模涉及做出这样的决定：哪些抽象是考虑中的系统的一部分，哪些抽象处于系统边界之外。用类图详细描述这些抽象和它们的职责。

(2）对简单的协作建模

协作是一些共同工作的类、接口和其他元素的群体，该群体提供的一些合作行为强于所有这些元素的行为之和。例如当对分布式系统的事务语义建模时，不能仅仅盯着一 个单独的类来推断要发生什么，而要有相互协作的一组类来实现这些语义。用类图对这组类以及它们之间的关系进行可视化和详述。

(3）对逻辑数据库模式建模

将模式看作数据库的概念设计的蓝图。在很多领域中，要在关系数据库或面向对象数据库中存储永久信息。可以用类图对这些数据库的模式建模。

本题主要使用类图对系统词汇进行建模。题目中已经给出了类图的基本框架及部分的类，要求考生将类图中其余的类补充完整。在解答这类题目时，需要细心阅读说明中 的文字，并记录和整理其中出现的名词。这些名词将来有可能成为类。其次应特别关注类图中出现的特殊关联关系，如继承关系、聚集/组装关系等。

在本题中，首先考查类图中的 Customer 、 C2 和 Doctor 这三个类。由说明可知，在网上购药时，顾客与医生之间不会直接发生交互，而是通过顾客持有的“处方”而发生关联。由此可以确定 C2 对应的类应该是“处方”。

C2 与 C5 之间是聚集关系，其中 C2 表示整体类， C5 表示部分类。由于已经确定了 C2 表示的是“处方”类，那么 C5 表示就应该是处方所包含的内容。处方中包含的是药品，所以 C5 对应的类应该是“处方上的药品，’ 。

下面来分析类图中的继承关系。继承关系表示类之间的“一般/特殊”关系。 C1 表 示一般类， C3 和 C4 是 C1 的两个具体类；并且这三个类与 Customer 之间具有组装关系。 那么在说明中出现的所有名词词汇中，具有明显的一般/特殊关系的就是“付款方式”、“信用卡”和“支付宝账户”。“信用卡”和“支付宝账户”是具体的付款形式，当顾客付款的时候选择二者中的一个。而且每一次付款都与一个特定的顾客(即类 Customer 的一个实例)相关，没有顾客就不会发生付款行为。所以 C1 对应的类应该是“付款方式”、 C3 和 C4 分别对应的是类“信用卡”、“支付宝账户”。

多重度表示一个类的实例与多少个另一个类的实例发生关联。因此，在确定多重度时需要关注说明中关于类之间关系的描述。

首先来看 C2 和 C5, 这两个类之间是聚集关系。前面已经确定了 C2 和 C5 分别对应类“处方”和“处方上的药品”。一张处方上应包含 1 种或多种药品。这样很容易确定出(3）和(4)的多重度应分别为 1 以及 1..*。

“处方”和“医生”之间的关系如下：一名医生可以开多张处方，也可以不开处方， 所以(5）处的多重度应该为 0..*;而一张处方必定是由一名医生开具的，所以(6）处的多重度应该为 1 。

“顾客”与“处方”之间的关系如下：一个顾客可以持有多张处方来买药，也可以没有处方，这样就不会发生购买行为。所以(2）处的多重度应该为 0..*。而每张处方一定属于一名顾客，所以(1）处的多重度应该为 1 。

状态图关注系统的动态视图，它注重描述可能的状态序列，以及在特定状态下对象对外部离散事件的响应动作。

本题考查的是类“处方”的对象的状态变化。关于网上药店对“处方”的处理流程，在说明的(4)验证处方中，给出了详细的描述。对该描述进行分析之后，可以用下面的表来说明“处方”在整个验证流程中所经历的状态。

下一步工作就是把上表中的信息与题中的状态图对应起来。

由说明可知，处方提交后的第一步操作就是核实医生信息，而这个操作会产生两种结果：医生信息正确，或者不正确。医生信息不正确会使处方的状态变更为“医生信息无效”，并导致购买行为被取消，即表中的第一行。对于这种情况，“处方”的状态变更 轨迹为：处方已提交→医生信息无效→结束。而在状态图中与这条轨迹匹配的状态序列 就是：处方已提交→S3→结束。由此可以确定， S3 对应的就是状态“医生信息无效”， 而(7）对应的迁移就是“医生信息不正确”。

相应地，就可以判断出(8）应该代表的是核实医生信息的另一种结果，因此(8）对应的迁移应该是“医生信息正确”。由上表可知，医生信息正确时，处方状态会变更为“审核中”，这样 S1 对应的状态就是“审核中”。

但处方在状态“审核中”时，实际上会有三个后续状态：一个是图中已经给出的“准许付款”，另外两个是“无效处方”和“无法审核”。而产生这两个状态的原因分别是“医 生回复处方无效”和“医生没有在 7 天内给出答复”。由此得出，(9）对应“医生回复处方无效”， S4 对应状态“无效处方”；(10）对应“医生没有在 7 天内给出答复”， S2 对应“无法审核”。

如果 S2 为状态“无效处方”，那么(10）就对应着“医生回复处方无效”； S4 对应状态“无法审核”，那么(9）就对应着“医生没有在 7 天内给出答复”。.

C1：付款方式　　C2：处方　　C3：信用卡　　C4：支付宝账户

C5：处方上的药品（或药品）　　（C3，C4 可以互换）

（1）1　　（2）0..*　　（3）1

（4）1..*　　（5）0..*　　（6）1

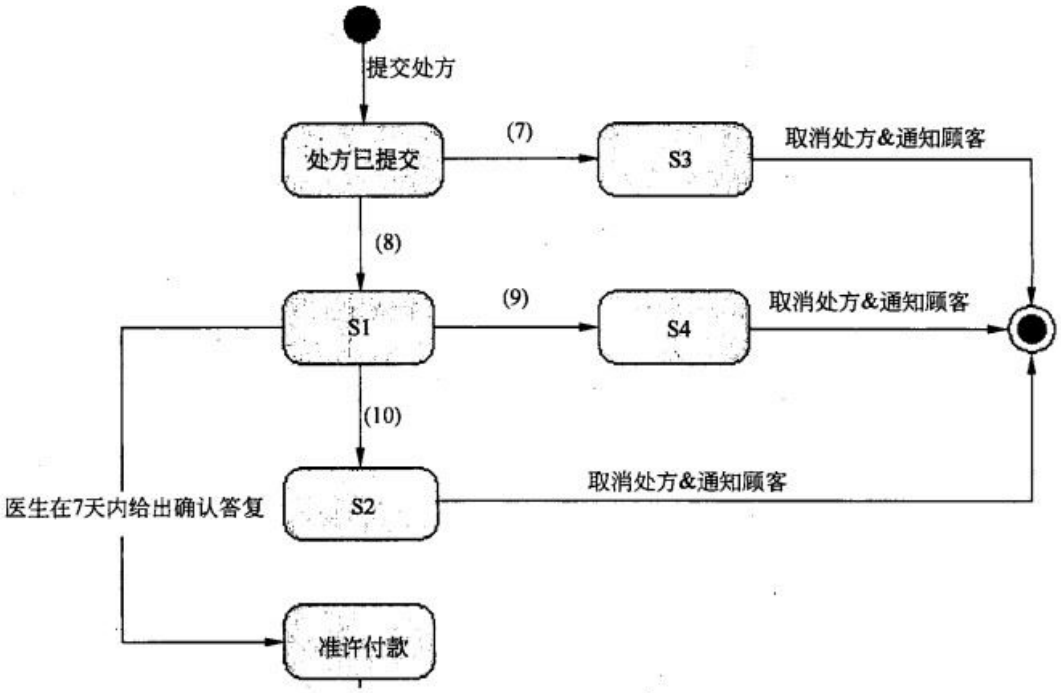


图 3-2　状态图

S1：审核中　S2：无法审核　　S3：医生信息无效　　S4：无效处方

（7）医生信息不正确　　（8）医生信息正确

（9）医生回复处方无效　　（10）医生没有在 7 天内给出确认答复

或者：

S2：无效处方　　S4：无法审核

（9）医生没有在 7 天内给出确认答复　　（10）医生回复处方无效

S1、S3、（7）、（8）同上

处 方 状 态	产生该状态的原因	验 证 结 果
医生信息无效	医生信息不正确	不通过
审核中	医生信息正确	----------
无效处方	医生回复处方无效	不通过
无法审核	医生没有在 7 天内给出答复	不通过
准许付款	医生在 7 天内给出确认答复	通过

◆表示组合（composition），◇表示聚合（aggregation）。

在组合关系中，整体对象与部分对象具有同一的生存周期。当整体对象不存在时，部分对象也不存在。（1 分）

而在聚合关系中，对整体对象与部分对象没有这样的要求。

试题四 答案： 解析： (1) A->int_array[0]

(2) A->int_array[0] = A->int_array[A->array_size-1]

(3) A->array_size-1

(4) A->int_array[PARENT(i)]

(5) A->int_array[i] = key

据题干说明，函数 heapMaximum 返回大顶堆 A 的最大元素，即堆顶元素，因此空(1) 处应填 A->int_array[0]。

函数 heapExtractMax(A)取出大顶堆 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。因此在将堆顶元素赋给 max 后，应该将堆的最后 一个元素移到堆顶位置，即空(2) 处应填 A->int_array[0] = A->int_array[A->array_size-1]。

函数 maxHeapInsert(A， 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。该函数前面的代码行考虑的是当存储空间不够时扩展存储空间。而后面是根据 该函数的定义实现的问题求解的算法表示， A->array_size ++;表示为堆的规模增加 1， i 表示堆的最后一个元素的下标，即新插入的元素的下标，应该为 A->array_size-1 。 while 循环是自下而上调整堆，当还没有到堆顶位置，且新插入的元素大于其父亲元素，即 A->int_array[PARENT(i)] int_array[i] = key 。

(6) *O* (1)

(7) *O* (lgn)

(8) *O* (lgn)

本问题考查算法的时间复杂度。

根据上述 C 代码，函数 heapMaximum 返回数组 A 的第 1 个元素，因此为常数时间即 *O* (1) 。

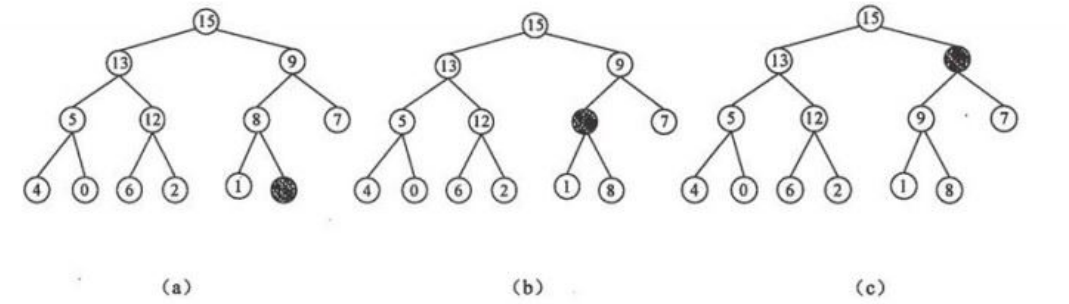
函数 heapExtractMax 首先将数组 A 的第 1 个元素的值放到变量 max 中，然后将最后一个元素提到堆顶，最后再进行堆的调整，因此该时间复杂度实际上是调整堆的时间复杂度，即 *O*(lgn)。

函数 maxHeapInsert 将一个元素 key 插入到堆 A 中，具体的过程为先将堆的规模增加 1，然后将元素插入到堆的最后一个位置，最后自下而上调整该元素，其时间复杂度 为堆(二叉树)的高度，即 *O*(lgn)。

(9) 3

将元素 10 插入到堆 A = <15, 13,9,5, 12,8,7,4,0,6,2, 1〉中，根据 maxHeapInsert 函数进行操作，则过程如下图(a) (c)所示。

新插入的元素 10 在堆 A 中处于第 3 个位置， 15 和 13 分别处于第 1 和第 2 个位置。



试题五 答案： 解析：

Composite 模式将对象组合成树形结构以表示“整体-部分”的层次结构，其中的组合对象使得用户可以组合基元对象以及其他的组合对象，从而形成任意复杂的结构。

Composite 模式使得用户对单个对象和组合对象的使用具有一致性。

Composite 模式的结构如下图所示。

其中：

•类 Component 为组合中的对象声明接口，在适当的情况下，实现所有类共有接口的缺省行为，声明一个接口用于访问和管理 Component 的子部件；

•类 Leaf 在组合中表示叶节点对象，叶节点没有子节点；并在组合中定义图元对象的行为；

•类 Composite 定义有子部件的那些部件的行为，存储子部件，并在 Component 接口中实现与子部件有关的操作；

•类 Client 通过 Component 接口操纵组合部件的对象。

下列情况可以使用 Composite 模式：

(1）表示对象的整体-部分层次结构；

(2）希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

图 5-2 中的 Company 对应的就是上图中的类 Component，ConcreteCompany 对应的是类 Composite;而上图中的 FinanceDepartment 和 HRDepartment 扮演的就是类 Leaf 的角色。由于类 Company 的作用是为其子类提供统一的操作接口，所以将其定义为抽象类。 在 C# 中，抽象类的定义是：至少包含一个纯虚拟函数的类。而纯虚拟函数是没有函数体的虚拟函数，其作用是为子类提供统一接口。若要使用纯虚拟函数，必须在子类中对其进行重置。定义纯虚拟函数的语法为：

virtual ()=0;

空(1) （ 3）考查的是如何定义抽象类 Company 。 Company 提供了两个方法接口 Add 和 Delete, 即该类中应包含两个纯虚拟函数。如何确定 Add. 和 Delete 的函数原型呢？ 这要借助于 Company 的子类。因为子类重置父类定义的虚拟函数时，不能改变其接口定义。所以从 ConcreteCompany 中的 Add 和 Delete 方法就能够确定出空(2) 和(3) 处应 分别填入 “virtualvoidAdd(Company* c) = 0” 和 “virtualvoidDelete(Company* c) = 0”。 空(1) 考察的是在构造函数中如何给数据成员赋初值。当构造函数的参数与类的数据成员同名时，可以借助 this 指针来进行区别，因此空(1) 处应填入 this->name 。

空(4) （ 6）考查对模式中 Composite 节点的定义。由图 5-2 可知， ConcreteCompany 与 Company 之间是聚集关系，即 ConcreteCompany 的实例中包含多个 Company 的子类 的实例。为了表示这种聚集关系，使用了 C++标准类库中的类模板 list 。 C++的类模板必须在实例化之后才能使用。实例化类模板时，要给出类型实参。由于 children 表示的是 类 Company 的子类的实例集合，所以空(4) 处应填入 Company*。空(5) 和(6) 处 分别使用了 list 中提供的方法来实现添加和删除子公司、办事处或部门。 children 是 list 的实例，所以空(5) 和(6) 处都应填入 children 。

空(7) 和(8) 考查的是组合模式的使用。由图 5-1 可知，组织结构图的根目录是 “北京总公司”，“上海分公司”应该插入在根目录之下。所以空(7) 处应填入 root->Add(Comp)。而“南京办事处”是以“上海分公司”为根的子树中的节点，应插入 在“上海分公司”这个节点的下面。对象 comp 表示的是以“上海分公司”为根的子树的根节点，所以空(8) 处应该填入 comp->Add(comp1)。

（1）this->name

（2）virtual void Add(Company* c) = 0

（3）virtual void Delete(Company* c) = 0 .

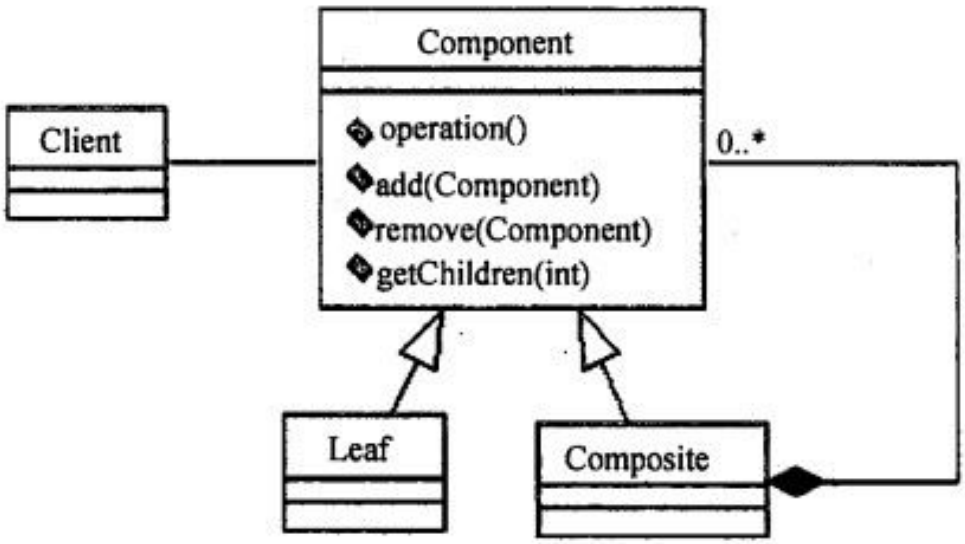
（4）Company*

（5）children

（6）children

（7）root->Add(comp)

（8）comp->Add(comp1)



试题六 答案： 解析：

Composite 模式将对象组合成树形结构以表示“整体-部分”的层次结构，其中的组合对象使得你可以组合基元对象以及其他的组合对象，从而形成任意复杂的结构。 Composite 模式使得用户对单个对象和组合对象的使用具有一致性。

Composite 模式的结构如下图所示。

其中：

•类 Component 为组合中的对象声明接口，在适当的情况下，实现所有类共有接口的缺省行为，声明一个接口用于访问和管理 Component 的子部件；

•类 Leaf 在组合中表示叶节点对象，叶节点没有子节点；并在组合中定义图元对象的行为;

•类 Composite 定义有子部件的那些部件的行为，存储子部件，并在 Component 接口中实现与子部件有关的操作；

•类 Client 通过 Component 接口操纵组合部件的对象。

下列情况可以使用 Composite 模式：

(1) 表示对象的整体-部分层次结构；

(2) 希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

图 6-2 中的 Company 对应的就是上图中的类 Component，ConcreteCompany 对应的是类 Composite;而上图中的 FinanceDepartment 和 HRDepartment 扮演的就是类 Leaf 的角色。

由于类 Company 的作用是为其子类提供统一的操作接口，所以将其定义为抽象类。空(1)（2）考查的是如何定义抽象类 Company 。在 Java 中，可以通过在类名之前加 abstract 关键字来定义抽象类，因此空(1) 处应填入 abstractclass 。空(2) 考查的是在构造函数中如何给数据成员赋初值。当构造函数的参数与类的数据成员同名时，可以借助 this 指针来进行区别，因此空(2) 处应填入 this.name 。

空(3) （6）考查对模式中 Composite 节点的定义。由图 5-2 可知，ConcreteCompany 与 Company 之间是聚集关系，即 ConcreteCompany 的实例中包含多个 Company 的子类 的实例。为了表示这种聚集关系，使用了 Java 包中的类模板 List 。类模板必须在实例化之后才能使用。实例化类模板时，要给出类型实参。由于 children 表示的是类 Company 的子类的实例集合，所以空(3) 和(4) 处都应填入 Company 。空(5) 和(6) 处分别 使用了 List 中提供的方法来实现添加和删除子公司、办事处或部门。 children 是 list 的实例，所以空(5) 和(6) 处都应填入 children 。

空(7) 和(8) 考查的是组合模式的实用。由图 6-1 可知，组织结构图的根目录是 “北京总公司”，“上海分公司”应该插入在根目录之下。所以空(7) 处应填入 root.Add(comp)0 而“南京办事处”是以“上海分公司”为根的子树中的节点，应插入在“上海分公司”这个节点的下面。对象 comp 表示的是以“上海分公司”为根的子树的根节点，所以空(8) 处应该填入 comp.Add(comp1)。

（1）abstract class

（2）this.name

（3）Company

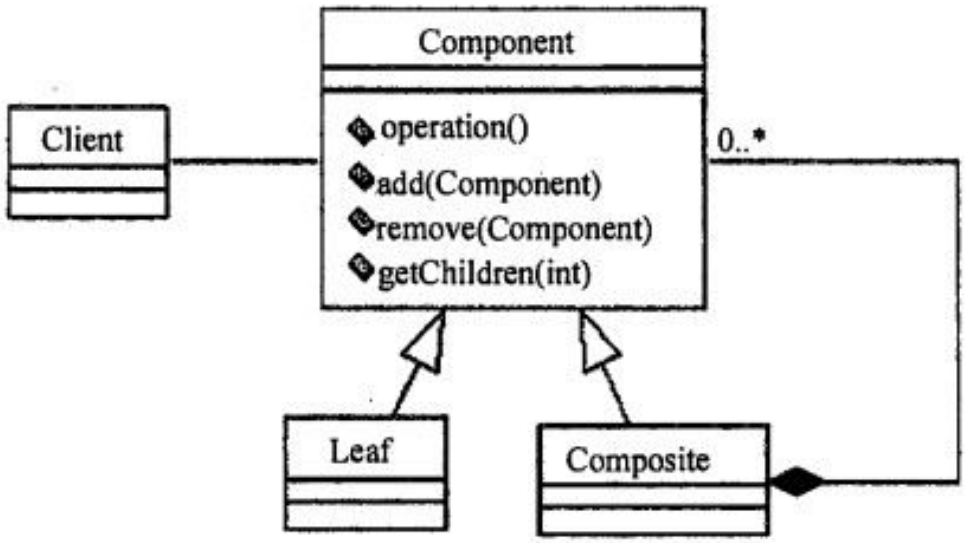
（4）Company

（5）children

（6）children

（7）root.Add(comp)

（8）comp.Add(comp1)



苹果 扫码或应用市场搜索“软考真题”下载获取更多试卷

安卓 扫码或应用市场搜索“软考真题”下载获取更多试卷