# 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 2011 年 下半年 下午试卷 案例

（考试时间 150 分钟）

**试题一** 某公司欲开发招聘系统以提高招聘效率，其主要功能如下：

(1）接受申请

验证应聘者所提供的自身信息是否完整，是否说明了应聘职位，受理验证合格的申请，给应聘者发送致谢信息。

(2）评估应聘者

根据部门经理设置的职位要求，审查已经受理的申请；对未被录用的应聘者进行谢绝处理，将未被录用的应聘者信息存入未录用的应聘者表，并给其发送谢绝决策；对录用的应聘者进行职位安排评价，将评价结果存入评价结果表，并给其发送录用决策，发送录用职位和录用者信息给工资系统。

现采用结构化方法对招聘系统进行分析与设计，获得如图 1-1 所示的顶层数据流图、图 1-2 所示 0 层数据流图和图 1-3 所示 1 层数据流图。

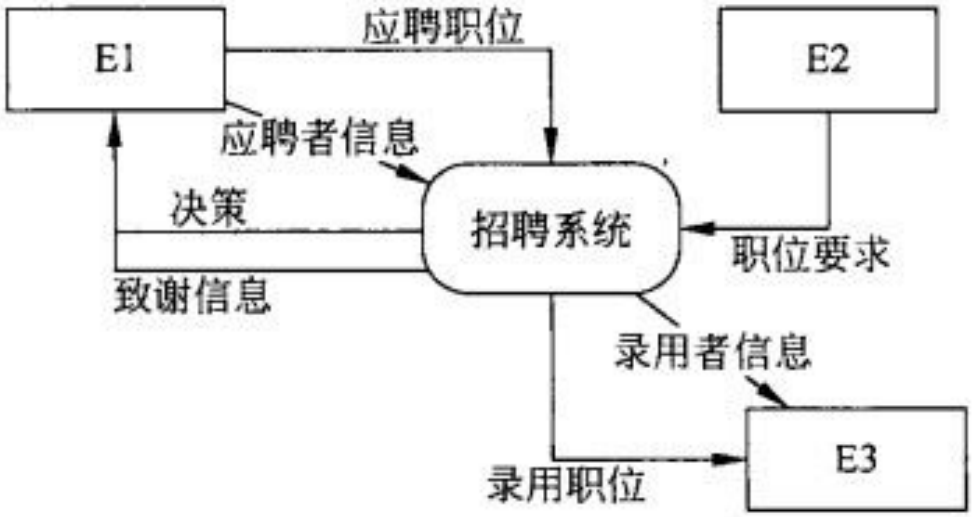



图 1-1　顶层数据流图

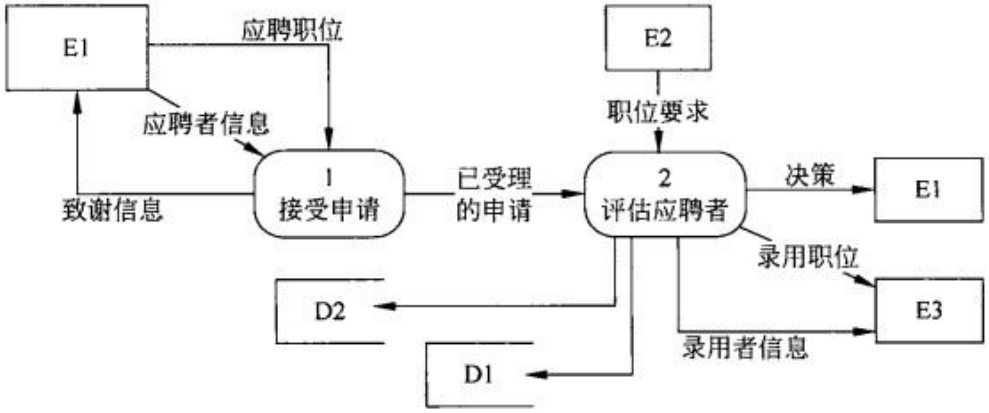



图 1-2　0 层数据流图

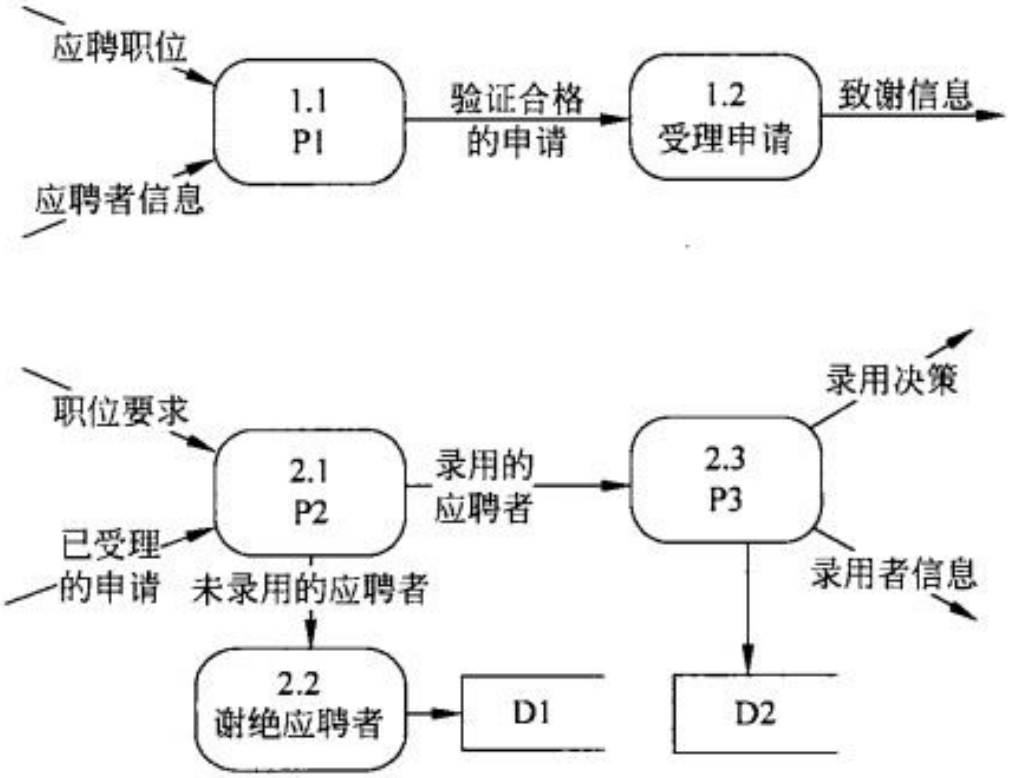



图 1-3　1 层数据流图

**问题：1.1**

使用说明中的术语，给出图中 E1　E3 所对应的实体名称。

**问题：1.2**

使用说明中的术语，给出图中 D1　D2 所对应的数据存储名称。

**问题：1.3**

使用说明和图中的术语，给出图 1-3 中加工 P1　P3 的名称。

**问题：1.4**

解释说明图 1-2 和图 1-3 是否保持平衡，若不平衡请按如下格式补充图 1-3 中数据流的名称以及数据流的起点或终点，使其平衡(使用说明中的术语或图中符号)。

| 数据流名称 | 起点 | 终点 |
|---|---|---|
| | | |

**试题二**　某物流公司为了整合上游供应商与下游客户，缩短物流过程，降低产品库存，需要构建一个信息系统以方便管理其业务运作活动。

【需求分析结果】

(1）物流公司包含若干部门，部门信息包括部门号、部门名称、经理、电话和邮箱。一个部门可以有多名员工处理部门的日常事务，每名员工只能在一个部门工作。每个部门有一名经理，只需负责管理本部门的事务和人员。

(2）员工信息包括员工号、姓名、职位、电话号码和工资；其中，职位包括：经理、业务员等。业务员根据托运申请负责安排承运货物事宜，例如：装货时间、到达时间等。一个业务员可以安排多个托运申请，但一个托运申请只由一个业务员处理。

(3）客户信息包括客户号、单位名称、通信地址、所属省份、联系人、联系电话、银行账号，其中，客户号唯一标识客户信息的每一个元组。每当客户要进行货物托运时，先要提出货物托运申请。托运申请信息包括申请号、客户号、货物名称、数量、运费、出发地、目的地。其中，一个申请号对应唯一的一个托运申请；一个客户可以有多个货物托运申请，但一个托运申请对应唯一的一个客户号。

【概念模型设计】

根据需求阶段收集的信息，设计的实体联系图和关系模式(不完整)如图 2-1 所示。

【关系模式设计】

部门(部门号，部门名称，经理，电话，邮箱)

员工(员工号，姓名，职位，电话号码，工资，(a))

客户((b)单位名称，通信地址，所属省份，联系人，联系电话，银行账号)

托运申请((c)，货物名称，数量，运费，出发地，目的地)

安排承运((d)，装货时间，到达时间，业务员)

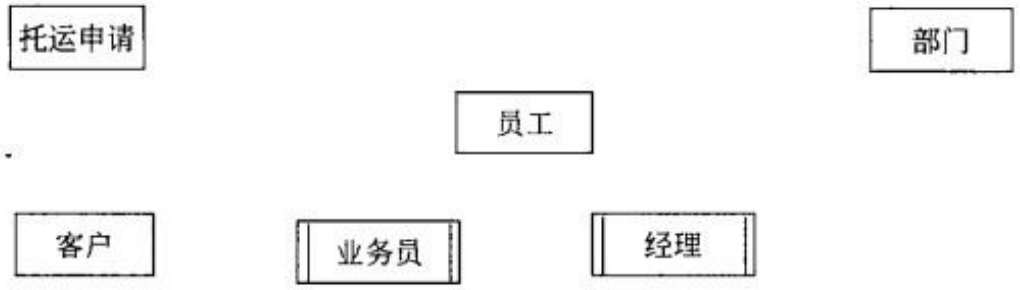



图 2-1　实体联系图

**问题：2.1**
根据问题描述，补充四个联系、联系的类型，以及实体与子实体的联系，完善图 2-1 所示的实体联系图。

**问题：2.2**
根据实体联系图，将关系模式中的空(a)～(d)补充完整。分别指出部门、员工和安排承运关系模式的主键和外键。

**问题：2.3**
若系统新增需求描述如下：
为了数据库信息的安全性，公司要求对数据库操作设置权限管理功能，当员工登录系统时，系统需要检查员工的权限。权限的设置人是部门经理。为满足上述需要，应如何修改(或补充)图 2-1 所示的实体联系图，请给出修改后的实体联系图和关系模式。

**试题三**　Pay&Drive 系统(开多少付多少)能够根据驾驶里程自动计算应付的费用。

系统中存储了特定区域的道路交通网的信息。道路交通网由若干个路段(RoadSegment)构成，每个路段由两个地理坐标点(Node)标定，其里程数(Distance)是已知的。在某些地理坐标点上安装了访问控制(AccessControl)设备，可以自动扫描行驶卡(Card)。行程(Trajectory)由一组连续的路段构成。行程的起点(Entry)和终点(Exit)都装有访问控制设备。

系统提供了 3 种行驶卡。常规卡(RegularCard)有效期(ValidPeriod)为一年，可以在整个道路交通网内使用。季卡(SeasonCard)有效期为三个月，可以在整个道路交通网内使用。单次卡(MinitripCard)在指定的行程内使用，且只能使用一次。其中，季卡和单次卡都是预付卡(PrepaidCard)，需要客户(Customer)预存一定的费用。

系统的主要功能有：客户注册、申请行驶卡、使用行驶卡行驶等。

使用常规卡行驶，在进入行程起点时，系统记录行程起点、进入时间(DateOfEntry)等信息。在到达行程终点时，系统根据行驶的里程数和所持卡的里程单价(UnitPrice)计算应付费用，并打印费用单(Invoice)。

季卡的使用流程与常规卡类似，但是不需要打印费用单，系统自动从卡中扣除应付费用。

单次卡的使用流程与季卡类似，但还需要在行程的起点和终点上检查行驶路线是否符合该卡所规定的行驶路线。

现采用面向对象方法开发该系统，使用 UML 进行建模。构建出的用例图和类图分别如图 3-1 和图 3-2 所示。

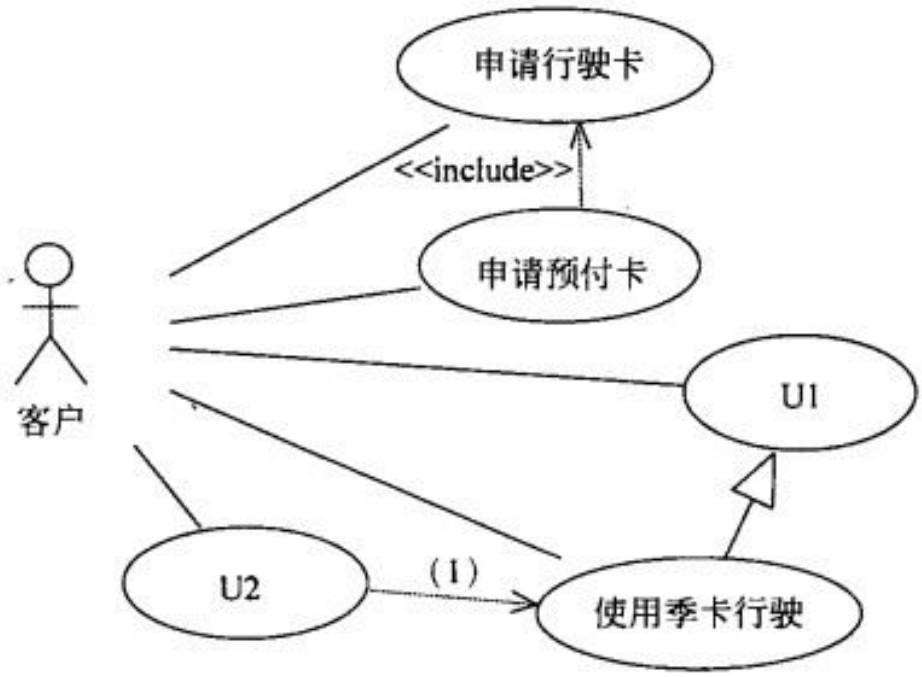



图 3-1　用例图

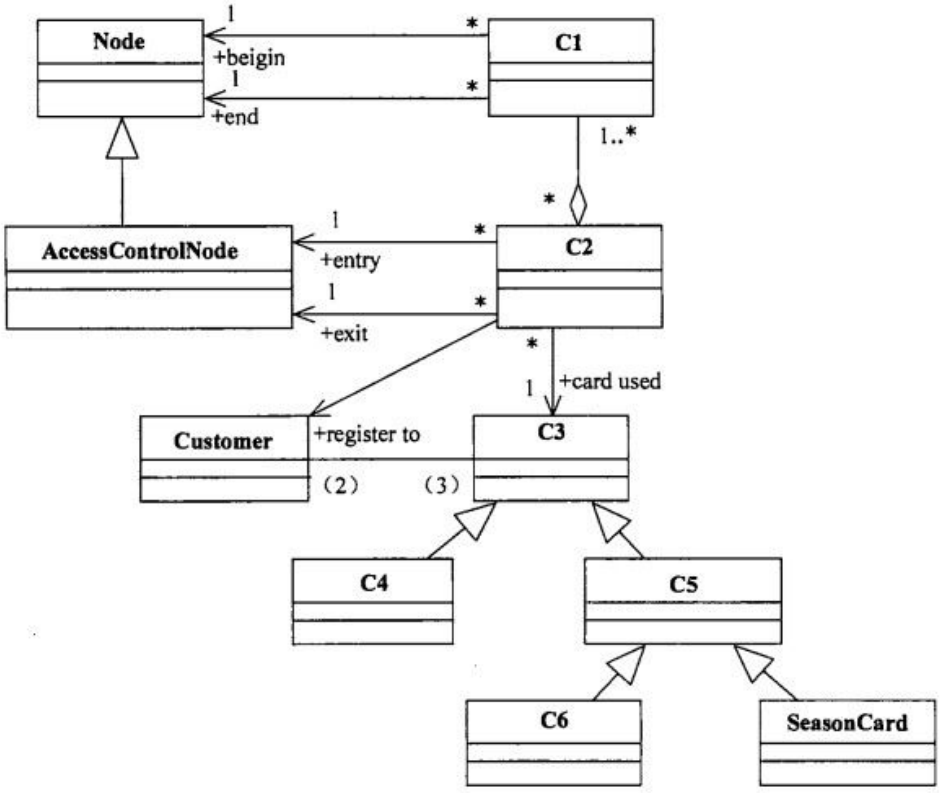



图 3-2　类图

**问题：3.1**

根据说明中的描述，给出图 3-1 中 U1 和 U2 所对应的用例，以及(1)所对应的关系。

**问题：3.2**
根据说明中的描述，给出图 3-2 中缺少的 C1　C6 所对应的类名以及(2)（ 3）处所对应的多重度(类名使用说明中给出的英文词汇)。
**问题：3.3**
根据说明中的描述，给出 RoadSegment 、 Trajectory 和 Card 所对应的类的关键属性（属性名使用说明中给出的英文词汇)。

**试题四**　设某一机器由 n 个部件组成，每一个部件都可以从 m 个不同的供应商处购得。供应商 j 供应的部件 i 具有重量 $W_{ij}$ 和价格 $C_{ij}$。设计一个算法，求解总价格不超过上限 cc 的最小重量的机器组成。

**采用回溯法来求解该问题：**

首先定义解空间。解空间由长度为 n 的向量组成，其中每个分量取值来自集合{1，2，・・・，m} 将解空间用树形结构表示。

接着从根结点开始，以深度优先的方式搜索整个解空间。从根结点开始，根结点成为活结点，同时也成为当前的扩展结点。向纵深方向考虑第一个部件从第一个供应商处购买，得到一个新结点。判断当前的机器价格($c_{11}$)是否超过上限(cc)，重量($w_{11}$) 是否比当前已知的解(最小重量)大，若是，应回溯至最近的一个活结点；若否，则该新结点成为活结点，同时也成为当前的扩展结点，根结点不再是扩展结点。继续向纵深方向考虑第二个部件从第一个供应商处购买，得到一个新结点。同样判断当前的机器价格($c_{11}+c_{21}$)是否超过上限(cc)，重量($w_{11}+w_{21}$)是否比当前已知的解(最小重量)大。若是，应回溯至最近的一个活结点；若否，则该新结点成为活结点，同时也成为当前的扩展结点，原来的结点不再是扩展结点。以这种方式递归地在解空间中搜索，直到找到所要求的解或者解空间中已无活结点为止。

**问题：4.1**
【C 代码】

下面是该算法的 C 语言实现。

（1）变量说明

n：机器的部件数

m：供应商数

cc：价格上限

w[][]：二维数组，w[i][j]表示第 j 个供应商供应的第 i 个部件的重量

c[][]：二维数组，c[i][j]表示第 j 个供应商供应的第 i 个部件的价格

bestW：满足价格上限约束条件的最小机器重量

bestC：最小重量机器的价格

bestX[]：最优解，一维数组，bestX[i]表示第 i 个部件来自哪个供应商

cw：搜索过程中机器的重量

cp：搜索过程中机器的价格

x[]：搜索过程中产生的解，x[i]表示第 i 个部件来自哪个供应商

i：当前考虑的部件，从 0 到 n–1

j：循环变量

（2）函数 backtrack

```
int n = 3;
int m = 3;
int cc = 4;
int w[3][3] = {{1,2,3},{3,2,1},{2,2,2}};
int c[3][3] = {{1,2,3},{3,2,1},{2,2,2}};
int bestW = 8;
int bestC = 0;
int bestX[3] = {0,0,0};
int cw = 0;
int cp = 0;
int x[3] = {0,0,0};
int backtrack(int i){
```

```
    int j = 0;
    int found = 0;
    if(i > n - 1){  /*得到问题解*/
        bestW = cw;
        bestC = cp;
        for(j = 0; j < n; j++){
            ____(1)____;
        }
return 1;
    }
    if(cp <= cc){ /*有解*/
        found = 1;
    }
    for(j = 0; ___(2)___; j++){
     /*第 i 个部件从第 j 个供应商购买*/
____(3)____;
        cw = cw + w[i][j];
        cp = cp + c[i][j];
if(cp <= cc &&___(4)___){ /*深度搜索，扩展当前结点*/
            if(backtrack(i + 1)){   found = 1;  }
        }
     /*回溯*/
        cw = cw - w[i][j];
        ___(5)___ ;
    }
    return found;
}
```

**试题五** 某大型商场内安装了多个简易的纸巾售卖机，自动出售 2 元钱一包的纸巾，且每次仅售出一包纸巾。纸巾售卖机的状态图如图 5-1 所示。

采用状态(State)模式来实现该纸巾售卖机，得到如图 5-2 所示的类图。其中类 State 为抽象类，定义了投币、退币、出纸巾等方法接口。类 SoldState 、 SoldOutState> NoQuarterState 和 HasQuarterState 分别对应图 5-1 中纸巾售卖机的 4 种状态：售出纸巾、纸巾售完、没有投币、有 2 元钱。

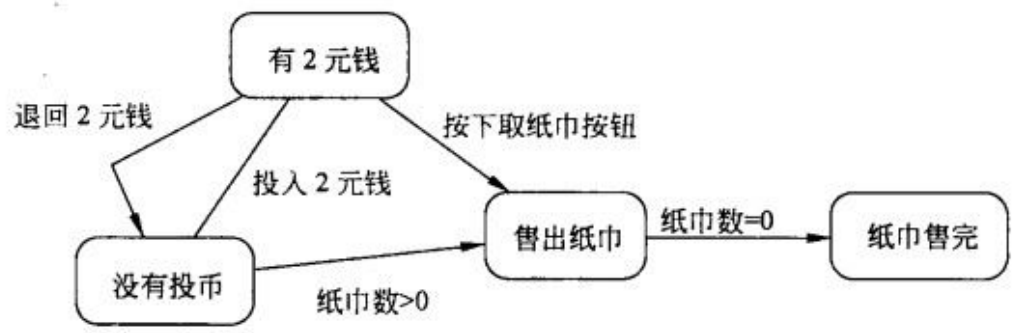



图 5-1 纸巾售卖机状态图

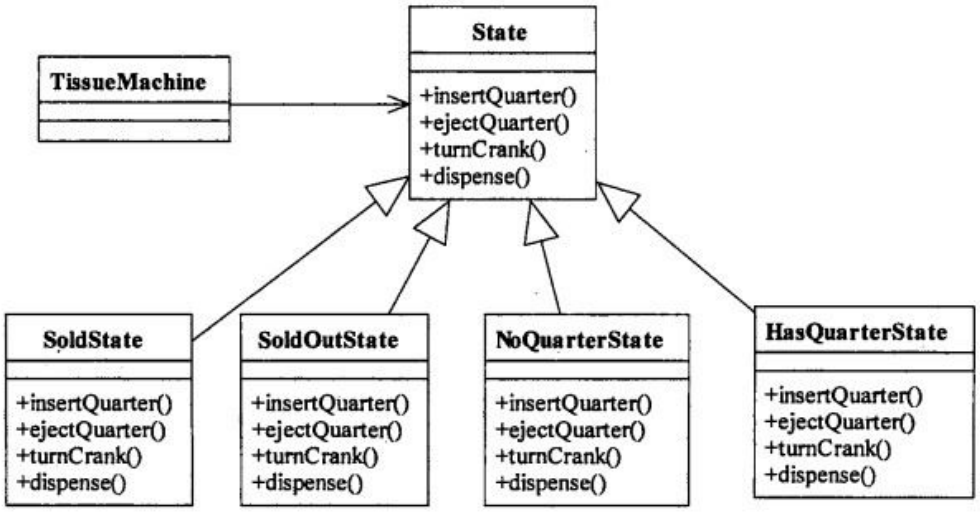



图 5-2 类图

**问题： 5.1**

## 【C++代码】

```
#include <iostream>
using namespace std;
// 以下为类的定义部分
class TissueMachine;                    // 类的提前引用
class State {
public:
   virtual void insertQuarter() = 0;//投币
   virtual void ejectQuarter() = 0; //退币
   virtual void turnCrank()= 0;      //按下“出纸巾”按钮
   virtual void dispense() = 0;      //出纸巾
};
/* 类 SoldOutState、NoQuarterState、HasQuarterState、SoldState 的定义省略，
每个类中均定义了私有数据成员 TissueMachine* tissueMachine; */
class TissueMachine {
private:
   __(1)__ *soldOutState, *noQuarterState, *hasQuarterState,*soldState,
*state ;
   int count;                           //纸巾数
public:
  TissueMachine(int numbers);
  void setState(State* state);
  State* getHasQuarterState();
  State* getNoQuarterState();
  State* getSoldState();
  State* getSoldOutState();
  int getCount();
  //其余代码省略
};
```

```
//以下为类的实现部分
void NoQuarterState ::insertQuarter() {
tissueMachine->setState(   (2)   );
}
void HasQuarterState ::ejectQuarter() {
tissueMachine->setState(   (3)   );
}
void SoldState ::dispense() {
    if(tissueMachine->getCount() > 0) {
      tissueMachine->setState(   (4)   );
}
else {
      tissueMachine->setState(   (5)   );
}
}  //其余代码省略
```

**试题六**　某大型商场内安装了多个简易的纸巾售卖机，自动出售 2 元钱一包的纸巾，且每次仅售出一包纸巾。纸巾售卖机的状态图如图 6-1 所示。

采用状态(State)模式来实现该纸巾售卖机，得到如图 6-2 所示的类图。其中类 State 为抽象类，定义了投币、退币、出纸巾等方法接口。类 SoldState 、 SoldOutState 、 NoQuarterState 和 HasQuarterState 分别对应图 6-1 中纸巾售卖机的 4 种状态：售出纸巾、纸巾售完、没有投币、有 2 元钱。

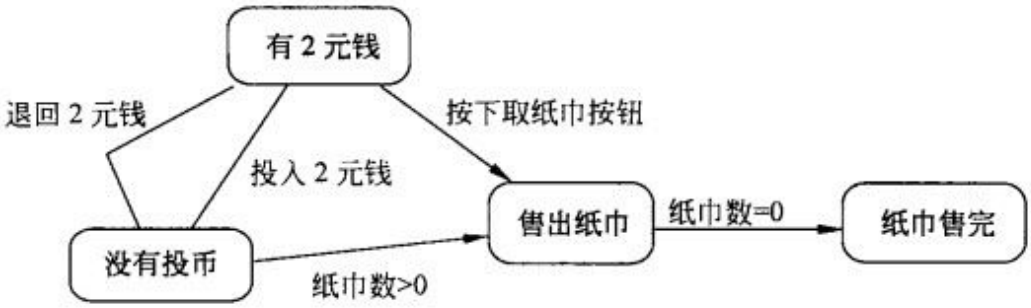



图 6-1　纸巾售卖机状态图

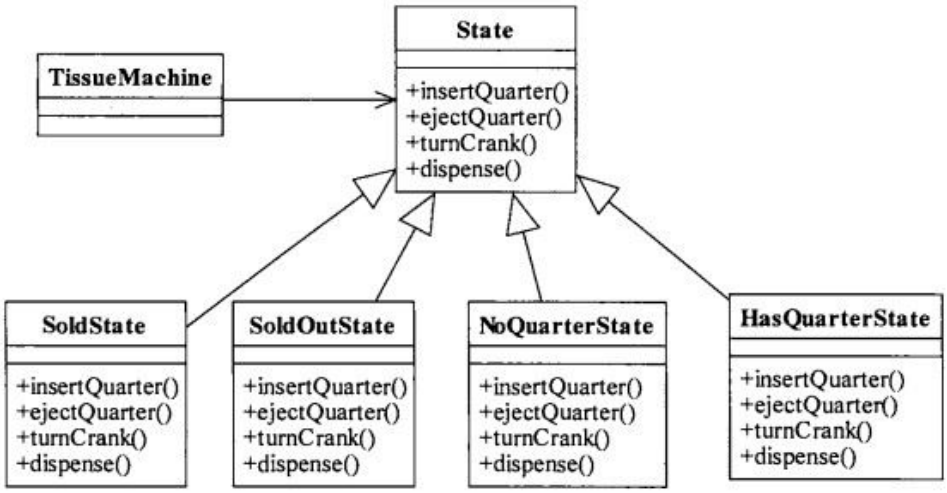



图 6-2　类图

**问题：6.1**

【Java 代码】

```
import java.util.*;
interface State {
  public void insertQuarter();        //投币
  public void ejectQuarter();         //退币
  public void turnCrank();            //按下“出纸巾”按钮
  public void dispense();             //出纸巾
}
class TissueMachine {
   (1)  soldOutState, noQuarterState, hasQuarterState, soldState,
state;
  state = soldOutState;
  int count = 0;                      //纸巾数
  public TissueMachine(int numbers) { /* 实现代码省略 */ }
public State getHasQuarterState() { return hasQuarterState; }
  public State getNoQuarterState()  { return noQuarterState; }
  public State getSoldState()       { return soldState;      }
  public State getSoldOutState()    { return soldOutState;   }
  public int getCount()           { return count;         }
//其余代码省略
}

class NoQuarterState implements State {
  TissueMachine tissueMachine;
  public void insertQuarter() {
    tissueMachine.setState(  (2)  );
  }
//构造方法以及其余代码省略
}
class HasQuarterState implements State {
  TissueMachine tissueMachine;
  public void ejectQuarter() {
    tissueMachine.setState(  (3)  );
  }
//构造方法以及其余代码省略
}
class SoldState implements State {
  TissueMachine tissueMachine;
  public void dispense() {
    if(tissueMachine.getCount() > 0)  {
```

```
        tissueMachine.setState(__(4)__);
    } else {
        tissueMachine.setState(__(5)__); }
      }
    }
```

**试题一 答案：　解析：**　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。外部实体可以是使用系统的用户，也可以是为系统提供输入或接收系统输出的外部系统。本问题要求根据描述确定图中的外部实体。应仔细分析题目中描述;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，与系统的交互者包括应聘者、部门经理和工资系统。分析说明中的描述可知，应聘者提供自身信息，并接收系统验证合格后的致谢信息等。部门经理设置职位要求。对录用者而言，将其录用职位和信息发送给工资系统。对应图 1-1 中数据流和实体的对应关系，可知 EI 为应聘者，E2 为部门经理， E3 为工资系统。

本问题考查 DFD 中数据存储的确定。本题中涉及的数据存储只有 2 个，一个是存储未被录用的应聘者信息，即未录用的应聘者表：另一个是存储对录用的应聘者进行职位安排评价的评价结果，即评价结果表。可以确定图 1-2 中 D1 和 D2 为未录用的应聘者表和评价结果表，因为有一个处理与这两个数据存储相关，需要再对应图 1-3, 可确认 D1 为未录用的应聘者表， D2 为评价结果表。

本问题考查 1 层 DFD 中缺失的处理。从说明(1) 中接受申请的描述功能，需先对应聘者信息进行验证，受理验证合格的申请，可知缺失的处理 P1 为验证信息。说明(2) 中，根据职位要求，审查已经受理的申请，对录用者进行职位安排评价，可知缺失的处理 P2 为审查申请， P3 为职位安排评价。

不平衡。图 1-2 中加工的输入输出流与其子图 1-3 中的输入输出流的数量不同。

本问题考查绘制分层 DFD 时的注意事项。在分层 DFD 中，需要保持父图与子图的平衡。即父图中某加工的输入输出数据流必须与其子图的输入输出数据流在数量和名字上相同，或者父图的一个输入(或输出)数据流对应于子图中几个输入(或输出)数据流，而子图中组成这些数据流的数据项全体正好是父图中的这一个数据流。

本题中，图 1-2 中加工的输入输出流与其子图 1-3 中的输入输出流的数量不同。也无需将

父图中一条数据流分解成子图中多条数据流，因此，补充子图中缺失的输入或输出数据流：录用职位、已受理的申请、谢绝决策。

| 数据流名称 | 起　点 |
| --- | --- |
| 录用职位 | P3 或 2.3 职位安排评价 |
| 已受理的申请 | 1.2 受理申请 |
| 谢绝决策 | 2.2 谢绝应聘者 |

**试题二 答案：　解析：**

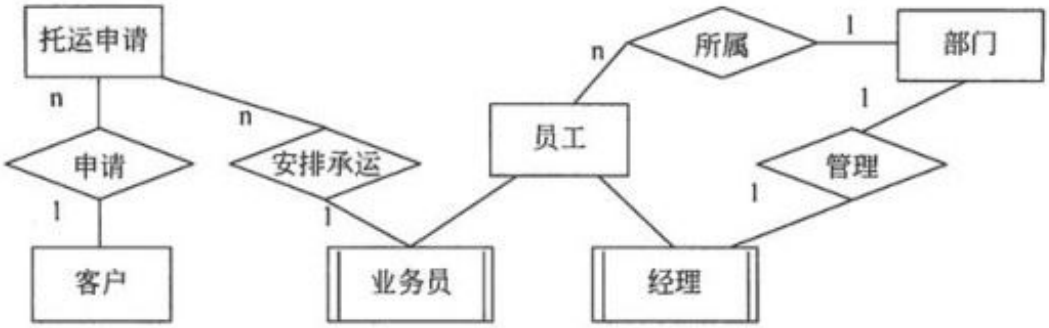
两个实体集之间的联系类型分为三类：一对一 (1 ： 1)联系、一对多(1 ： n)联系 和多对多(m ： n)联系。

根据题意，每名员工只能在一个部门工作，所以部门和员工之间有一个 1 ： n 的“所属”联系；由于每个部门有一名经理，只需负责管理本部门的事务和人员，因此部门和经理之间有一个 1 ： 1 的“管理”联系；由于一个业务员可以安排多个托运申请，但一个托运申请只由一个业务员处理，故业务员和托运申请之间有一个 1 ： n 的“托运”联系； 又由于一个客户可以有多个货物托运申请，但一个托运申请对应唯一的一个客户号，故客户和托运申请之间有一个 1 ： n 的“申请”联系。

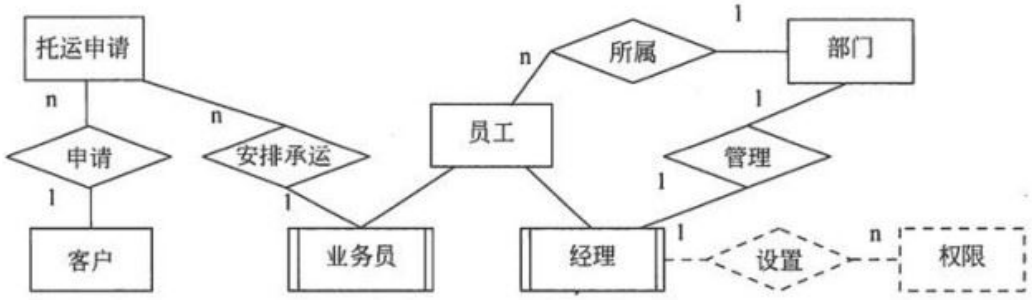
根据题意，部门和员工之间有一个 1 ： n 的“所属”联系需要将一端的码并入多端，故员工关系模式中的空(a)应填写部门号；在客户关系模式中，客户号为主键，故空(b) 应填写客户号；在托运申请关系模式中；申请号、客户号为主键，故空(c)应填写申请号、客户号；又由于一个业务员可以安排多个托运申请，但一个托运申请只由一个业务员处理，因此在安排承运关系模式中，申请号为主键，故空(d)应填写申请号。

部门关系模式中的部门号为主键，经理为外键；因为经理来自员工关系。员工关系模式中的员工号为主键，部门号为外键，因为部门号来自部门关系。安排承运关系模式中的申请号为主键，业务员为外键，因为业务员来自员工关系。

根据题意，权限的设置人是部门经理，因此，需要建立一个权限关系模式，以及经理到权限之间的 1 ： n 的“设置”联系。




| | | |
|---|---|---|
| (a) | 部门号 | |
| (b) | 客户号 | |
| (c) | 申请号，客户号 | |
| (d) | 申请号 | |
| 部门 | 主键：部门号 | 外键：经理 |
| 员工 | 主键：员工号 | 外键：部门号 |
| 安排承运 | 主键：申请号 | 外键：业务员 |




**试题三 答案： 解析：** U1：使用常规卡行驶

U2：使用单次卡行驶

(1)： extend

本问题要求将图 3-1 所给出的用例图补充完整。用例图的构成要素有：参与者、用例以及用例之间的关系。图中缺少了两个用例，以及一个用例关系。解答此题时，首先应从说明中找到所有的用例。

用例表示系统的一个单一业务功能。从题目的描述中可以看出，系统的主要功能就是申请行驶卡，以及使用行驶卡行驶。由于行驶卡分为三种，所以在说明中详细描述了三种行驶卡的使用方法。再结合用例图来看，缺少的两个用例与用例“使用季卡行驶”有关联关系，由此可以推断出，需要补充的这两个用例必定与另两种行驶卡相关，分别为“使用常规卡行驶”和“使用单次卡行驶”。

下面需要解决的问题是这两个用例与 U1 和 U2 的对应关系。这就需要仔细考查一下用例图

所给出的用例关系。由图 3-1 可知， U1 和“使用季卡行驶”之间是泛化（generalization)关系。当多个用例共同拥有一种类似的结构和行为时，可以将它们的共性抽象为父用例，其他的用例作为泛化关系中的子用例。在用例的泛化关系中，子用例是父用例的一种特殊形式，子用例继承了父用例所有的结构、行为和关系。根据说明中的“季卡的使用流程与常规卡类似，但是不需要打印费用单，系统自动从卡中扣除应付费用”可知， U1 应该对应着用例“使用常规卡行驶”。由此不难得出 U2 对应着用例“用单次卡行驶”。

现在图中只剩下(1) 处的用例关系没有确定。用例之间的关系在用例图上只有三种：包含(include)、扩展(extend)和泛化(generalization)。

包含关系是指当多个用例中存在相同事件流时，可以把这些公共事件流抽象成为公共用例，这个公共用例称为抽象用例，而原始用例称为基础用例。基础用例和抽象用例之间是包含关系。

如果一个用例明显地混合了两种或两种以上的不同场景，则可以将这个用例分为一个基本用例和多个扩展用例。扩展关系用〞《 extend 》〞表示，箭头指向基本用例。

包含关系和扩展关系的区别在于，抽象用例中的事件流一定要插入到基本用例中去，并且插入点只有一个，通常抽象用例不能脱离基本用例而独立存在。扩展用例的事件流往往可以抽象为基本用例的备选事件流，在扩展关系中，可以根据一定的条件来决定是否将扩展用例的事件流插入到基本用例的事件流中，并且插入点可以有多个。

根据以上分析可知，(1) 处的用例关系选择“《 extend 》”最为合适。

C1： RoadSegment

C2： Trajectory

C3： Card

C4： RegularCard

C5： PrepaidCard

C6： MinitripCard

(2) 1

(3) 1..3

本问题考查的是类图建模。解题的重点在于根据类图中提供的类及类之间的关联关系，推断出剩余的类。

可以先观察一下类图。可以看到，需要补充的类基本上集中在两个结构上：聚集结构(类 C1 和 C2)以及继承结构(类 C3 C6)。继承结构是比较容易辨识的类之间的关联关系，图上给出了其中的一个子类 SeasonCard 。以这个类为线索，回到说明中寻找与类 SeasonCard 相关的其他类。从说明中可知，“系统提供了 3 种卡”，常规卡、季卡、单次卡，而“季卡和单次卡都是预付卡”。这些描述暗示，“季卡”、“单次卡”与“预付

卡” 之间存在着特殊/一般关系，即“is-a”关系，这是继承结构的典型标志。由此可以得出类 C5 和 C6 应该分别对应 PrepaidCard（预付卡）和 MinitripCard（单次卡）。根据 C5 和 C6 所对应的类，可以推断出， C4 和 C3 必定也是与行驶卡相关的类。三种卡中，已经有两种卡有了对应的类，还剩下一种卡即“常规卡”。而“常规卡”只能是与“预付卡” 同层次的概念，所以只能对应于 C4，C3 表示的是能代表所有这几种卡的公共概念。所 以 C3 和 C4 应分别对应于 Card 和 RegularCard 。确定了 C3 之后，就可以识别出(2) 和(3) 处的多重度。 Customer 和 Card 之间是持有和被持有的关系，由于系统中只有 3 种卡，所以一个客户最多只能有 3 种卡，所以(3) 处应填 1..3 。而对于任何一张卡来说，只能有唯一地一个所属人，因此(2) 处应填 1 。

现在还剩下类 C1 和 C2 没有确定。由于这两个类之间是聚集关系，所以需要在说明中寻找具有“部分-整体”关系的概念。由说明中的“行程(Trajectory)由一组连续的路段构成”可知， C1 和 C2 应分别对应于 RoadSegment 和 Trajectory 。

RoadSegment 的属性： Distance

rajectory 的属性： Entry 、 Exit 、 DateOfEntry

Card 的属性： UnitPrice 、 ValidPeriod

本问题考查类的关键属性的识别。由说明中给出的描述可知，类 RoadSegment 的属性至少应包括 Distance;类 Trajectory 的属性至少应包括 Entry 、 Exit 和 DateOfEntry;类 Card 的属性至少应包括 UnitPrice 、 ValidPeriod 。

**试题四 答案：　解析：** (1) bestX[j] = x[j]

(2) j

(3) x[i] = j

(4) cw

(5) cp = cp-c[i][j]

本题考查算法的设计和分析技术中的回溯法。

回溯法是一种系统搜索问题解的方法，在包含问题所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树。算法在到达解空间树的任一结点时，总是先判断该结点是否肯定不包含问题的解。若肯定不包含，则跳过对以该结点为根的子树的系统搜索，逐层向其祖先结点回溯；否则进入该子树，继续按深度优先的策略进行搜索。回溯法在求问题的最优解时，要回溯到根，且根结点的所有子树都已经被搜索遍才结束。

根据上述思想和题干说明，对实例：部件数 n = 3, 厂商数 m=3, 具体的重量和价格 如表 4-1 所示

。

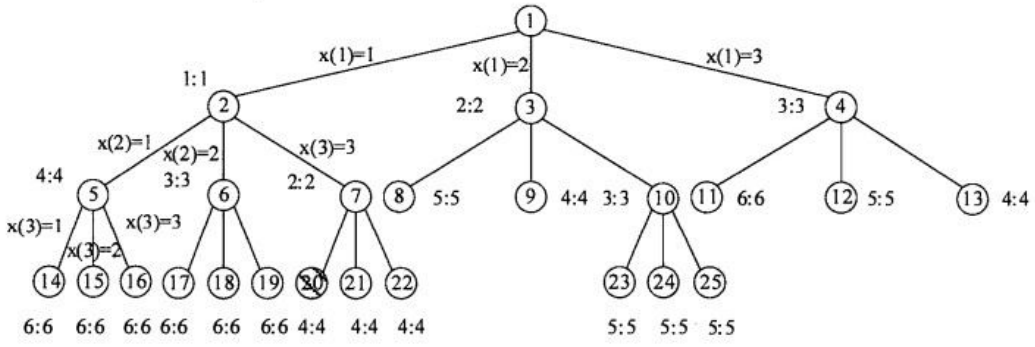
构造该实例的解空间树如图 4-1 所示。

图 4-1 中结点编号表示生成该结点的顺序。边上的编号表示哪个部件选择哪个厂商， 如 x(2) =1，表示第 2 个部件来自厂商 1 。结点旁边的两个数字表示当前解或部分解对应的重量和价格，如 2: 2 表示重量为 2，价格为 2 。从图 4-1 可以看出，最优解是结点 20 表示的解，即 x(1) =1, x(2) =3, x(3) =1, 即第 1 个部件来自厂商 1，第 2 个部件来自厂商 3, 第 3 个部件来自厂商 1, 总的价格和重量分别为 4 和 4 。当然，本实例的最优解还可 以是 x(1) =1， x(2) =3, x(3) =2 和 x(1) =1, x(2) =3, x(3) =3, 分别对应解空间树上的 21 号和 22 号结点。

代码中的空(1) 处是得到问题解之后，将搜索过程中产生的重量 cw 、价格 cp 和解 x 放到最终重量 bestw 、价格 bestc 和解 bestX 中，因此空格(1) 处填写 bestX[j] = x[j]。空(2) 处的 for 循环是考虑第 i 个部件选择哪个厂商，因此 j 从 0 到 m-1 依次检查，此处应填 j

**表 4-1 每个部件的重量和价格**

| | | n | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | | 2 | | 3 | |
| | | $w_1$ | $c_1$ | $w_2$ | $c_2$ | $w_3$ | $c_3$ |
| m | 1 | 1 | 1 | 2 | 2 | 3 | 3 |
| | 2 | 3 | 3 | 2 | 2 | 1 | 1 |
| | 3 | 2 | 2 | 2 | 2 | 2 | 2 |



**试题五 答案：　解析：**　(1) State

(2) tissueMachine->getHasQuarterState()

(3) tissueMachine->getNoQuarterState()

(4) tissueMachine->getNoQuarterState()

(5) tissueMachine->getSoldOutState()

本题考查状态(State)模式的概念及应用。

状态模式是一种对象的行为型模式，允许一个对象在其内部状态改变时改变它的行为，对象看起来似乎修改了它的类。状态模式的类图如下所示：

状态模式主要解决的是控制一个对象转换的条件表达式过于复杂的情况。把状态的判断逻辑转移到表示不同状态的一系列类当中，可以把复杂的判断逻辑简化。状态模式的好处是将与特定状态相关的行为局部化，并且将不同状态的行为分割开来。

题目利用状态模式来实现一个简易的纸巾售卖机。售卖机的状态转换图已经在题目中给出，类 SoldState 、 SoldOutState 、 NoQuarterState 和 HasQuarterState 分别用来表示售卖机的 4 种不同状态，对应于状态模式中的 ConcreteState1，...ConcreteStateN 。题目所设置的填空，主要集中在状态转换上。因此解答该题时，要求在理解状态模式内涵的基础上，依据纸巾售卖机的状态转换原则，给出正确的状态设置。

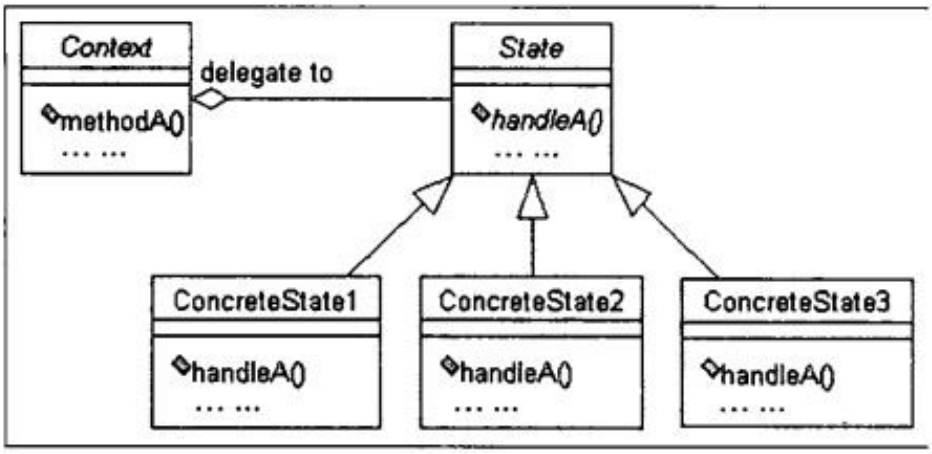
空(1) 出现在类 TissueMachine 的数据成员定义部分。状态模式封装了状态的转换过程，但是它需要枚举可能的状态，因此需要确定状态种类。因此在类 TissueMachine 中需定义出所有可能的状态对象。根据所给出的对象名称及说明中的描述，可知(1) 处应填入的类名为 State 。

空(2) （ 5）都是与状态转换相关的，要求填写类 TissueMachine 中的方法 setState 在不同调用处的实际参数。根据方法的名称及调用方式，可以推断出这个方法的功能就是设置自动售卖机的当前状态。要填出这些空，只要对照图 5-1 的状态转换图，根据状态转换的条件确定出当前状态及下一状态即可。

空(2) 出现在方法 insertQuarter 内，即给纸巾售卖机投入 2 元钱。根据状态图，“投入 2 元钱”之后，售卖机应转换到“有 2 元钱”的状态。“有 2 元钱”对应的状态的类为″HasQuarterState”，所以空(2) 处应填类 HasQuarterState 的对象。由于 hasQuarterState 是类 TissueMachine 的私有数据成员，不能直接访问，所以只能通过调用相关的 get 方法来获取该对象。由此得出(2) 应填 tissueMachine-> getHasQuarterState() 。

同理，空(3) 表示的状态是从“有 2 元钱”状态，经历“退回 2 元钱”事件之后的状态，及“没有投币”状态。所以空(3) 处应填 tissueMachine->getNoQuarterStat() 。

空(4) 和(5) 处分别表示卖出一包纸巾之后，售卖机应该转换到的下一个状态。 这个跟售卖机中的纸巾数有关，如果还有纸巾，则转换到“没有投币”状态，如果没有纸巾了，则转换到“纸巾售完”状态，因此，空(4) 处应填 tisSUeMachine->getNoQuarterState() ，空(5) 处应填 tissueMachine->getSoldOutState() 。




**试题六 答案： 解析：** (1) State

(2) tissueMachine.getHasQuarterStat()

(3) tissueMachine.getNoQuarterState()

(4)tissueMachine.getNoQuarterState()

(5) tissueMachine.getSoldOutState()

本题考查状态(State)模式的概念及应用。

状态模式是一种对象的行为型模式，允许一个对象在其内部状态改变时改变它的行为，对象看起来似乎修改了它的类。状态模式的类图如下所示：

状态模式主要解决的是控制一个对象转换的条件表达式过于复杂的情况。把状态的判断逻辑转移到表示不同状态的一系列类当中，可以把复杂的判断逻辑简化。状态模式的好处是将与特定状态相关的行为局部化，并且将不同状态的行为分割开来。

题目利用状态模式来实现一个简易的纸巾售卖机。售卖机的状态转换图已经在题目中给出，类 SoldState 、 SoldOutState 、 NoQuarterState 和 HasQuarterState 分别用来表示售卖机的 4 种不同状态，对应于状态模式中的 ConcreteState1 ，...ConcreteStateN 。题目所

设置的填空，主要集中在状态转换上。因此解答该题时，要求在理解状态模式内涵的基础上，依据纸巾售卖机的状态转换原则，给出正确的状态设置。

空(1) 出现在类 TissueMachine 的数据成员定义部分。状态模式封装了状态的转换过程，但是它需要枚举可能的状态，因此需要实现确定状态种类。因此在类 TissueMachine 中需定义出所有可能的状态对象。根据所给出的对象名称及说明中的描述，可知(1) 处 应填入的类名为 State 。

空(2) （ 5）要求填写类 TissueMachine 中的方法 setState 在不同调用处的实际参数。这里的一个难点在于题目中没有显示地给出方法 setState 的原型及语义，这要求考生根据面向对象程序设计风格及说明中给出的应用场合来推断 setState 的内涵及原型，主要是确定其参数列表。

在面向对象程序设计中，为了做到封装，通常都会把数据成员定义为私有的。私有的数据成员对象不能直接访问，因此在类中都会提供 2 组访问私有数据成员的方法，分别为 get... 方法和 set... 方法(…代表对应的数据成员名称)。 get... 方法表示获取私有数据成员的值，其返回值类型为对应的数据成员的类型； set... 方法表示对数据成员进行赋值，所要赋的值通常通过参数传递进去，方法的返回值类型通常为 void 。根据面向对象 程序设计的这些特点，以及状态模式的内涵及应用场合，可以推断出 setState 方法的功能就是设置纸巾售卖机的当前状态。纸巾售卖机在任一时刻只能处于一个唯一的状态， 由状态模式可知，纸巾售卖机的状态都是用状态对象表示的，由此就可以确定出， setState 方法的参数只要一个就可以了，就是表示纸巾售卖机下一状态的状态对象。

经过以上分析之后，可以明确空(2) （ 5）空所填的内容都应与状态转换相关。因此要填充这些空，只要对照图 5-1 的状态转换图，根据状态转换的条件确定出当前状态及下一状态即可。

空(2) 出现在方法 insertQuarter 内，即给纸巾售卖机投入 2 元钱。根据状态图，“投入 2 元钱”之后，售卖机应转换到“有 2 元钱”的状态。“有 2 元钱”对应的状态的类为“ HasQuarterState”，所以空(2) 处应填写类 HasQuarterState 的对象。由此得出(2) 应填 tissueMachine.getHasQuarterState() 。

同理，空(3) 表示的状态是从“有 2 元钱”状态，经历“退回 2 元钱”事件之后 的状态，及“没有投币”状态。所以空(3) 处应填 tissueMachine.getNoQuarterState() 。

空(4)和(5) 处分别表示卖出一包纸巾之后，售卖机应该转换到的下一个状态。 这个跟售卖机中的纸巾数有关，如果还有纸巾，则转换到“没有投币”状态，如果没有纸巾了，则转换到“纸巾售完”状态，因此，空(4)处应填 tissueMachine.getNoQuarterState() ，空(5) 处应填 tissueMachine.getSoldOutState() 。