面向对象建模分析案例测试

班级：信1405-2班学号：20142862姓名：孟祥通

1、阅读下列说明和 UML 图，回答问题 1 至问题 4。

【说明】

某企业为了方便员工用餐，为餐厅开发了一个订餐系统（COS：Cafeteria

Ordering

System），企业员工可通过企业内联网使用该系统。企业的任何员工都可以

查看菜单和今日特价。系统的顾客是注册到系统的员工，可以订餐（如果未登

录，需先登录）、注册工资支

付、预约规律的订餐，在特殊情况下可以覆盖预订。

餐厅员工是特殊顾客，可以进行备餐、生成付费请求和请求送餐，其中对于

注册工资支付的顾客生成付费请求并发送给工资系统。

菜单管理员是餐厅特定员工，可以管理菜单。送餐员可以打印送餐说明，

记录送餐信息（如送餐时间）以及记录收费（对于没有注

册工资支付的顾客，由送餐员收取现金后记录）。顾客订餐过程如下：

1. 顾客请求查看菜单；

2. 系统显示菜单和今日特价；

3. 顾客选菜；

4. 系统显示订单和价格；

5. 顾客确认订单；

6. 系统显示可送餐时间；

7. 顾客指定送餐时间、地点和支付方式；

8. 系统确认接受订单，然后发送 Email 给顾客以确认订餐，同时发送相

关订餐信息通知给餐厅员工。

系统采用面向对象方法开发，使用 UML 进行建模。系统的顶层用例图和

一次订餐的活动图初稿分别如图 1-1 和图 1-2 所示。

【问题 1】根据【说明】中的描述，给出图 1-1 中 A1 和 A2 所对应的参

与者。

【问题 2】根据【说明】中的描述，给出图 1-1 中缺少的四个用例及其所

对应的参与者。

【问题 3】根据【说明】中的描述，给出图 1-2 中（1）～（4）处对应的

活动名称或图形符号。

【问题 4】指出图 1-1 中员工和顾客之间是什么关系，并解释该关系的内

涵。

答案：

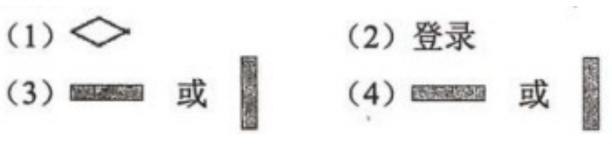
【问题 1】A1 工资系统 A2菜单管理员

【问题2】查看当日特价员工

注册工资支付顾客和工资系统（或顾客和A1）

生成付费请求餐厅员工和工资系统（或餐厅员工和A1）

管理菜单 菜单管理员（或A2）

【问题3】

【问题 4】泛化关系（一般/特殊关系，继承关系），泛化关系描述了一个参与者可以完成另一个参与者同样的任务，并可补充额外的角色功能。

2、试题三（共 15 分）

阅读下列说明和图，回答问题 1 至问题 3。

【说明】

某银行计划开发一个自动存提款机模拟系统（ATM System）。系统通过读

卡器 CardReader）读取 ATM 卡；系统与客户（Customer ）的交互由客户控制

台（CustomerConsole）实现；银行操作员（Operator）可控制系统的启动（System

Startup）和停止（System Shutdown）；系统通过网络和银行系统（Bank）实现通

信。当读卡器判断用户已将 ATM 卡插入后，创建会话（Session）。会话开始后，

读卡器进行读卡，并要求客户输入个人验证码（PIN）。系统将卡号和个人验证

码信息送到银行系统进行验证。验证通过后，客户可从菜单选择如下事务

（Transaction）：

1. 从 ATM 卡账户取款（Withdraw）；

2. 向 ATM 卡账户存款（Deposit）；

3. 进行转账（Transfer）；

4. 查询（Inquire）ATM 卡账户信息。

一次会话可以包含多个事务，每个事务处理也会将卡号和个人验证码信息送

到银行系统进行验证。若个人验证码错误，则转个人验证码错误处理（Invalid PIN

Process）。每个事务完成后，客户可选择继续上述事务或退卡。选择退卡时，系

统弹出 ATM 卡，会话结束。

系统采用面向对象方法开发，使用 UML 进行建模。系统的顶层用例图如

图 2-1 所示，一次会话的序列图（不考虑验证）如图 2-2 所示。消息名称参见

表 2-1。

表 2-1 可能的消息名称列表

名称说明名称说明

cardInserted( ) ATM 卡已插入

performTransaction( ) 执行事务

performSession( ) 执行会话 readCard( ) 读卡

readPIN( ) 读取个人验证码 PIN 个人验证码信息

creat(atm, this, card,

pin)

为当前会话创建事

务

create(this) 为当前 ATM 创

建会话

card ATM 卡信息 doAgain 执行下一个事务

ejectCard( ) 弹出 ATM 卡

【问题 1】根据【说明】中的描述，给出图 2-1 中 A1 和 A2 所对应的参

与者，U1 至 U3 所对应的用例，以及该图中空（1）所对应的关系。（U1 至

U3 的可选用例包括：Session、Transaction、Insert Card、Invalid PIN Process 和

Transfer）

【问题 2】根据【说明】中的描述，使用表 2-1 中的英文名称，给出图 2-2

中 6～9 对应的消息。

【问题 3】解释图 2-1 中用例 U3 和用例 Withdraw、Deposit 等四个用例

之间的关系及其内涵。

答案：

【问题 1】A1：Customer A2：Bank

U1：Session U2：Invalid PIN Process U3：Transaction

 (1)：＜＜extend＞＞

【问题 2】6：readPIN() 7:PIN 8:creat(atm,this,card,pin)9:preformTransaction()

【问题 3】Transaction是一个抽象泛化用例，具有其他事务类型共有的属性和行为，每个具体的事务类型继承它，并实现适合自己的特定的操作。

3、阅读下列说明和图，回答问题 1 至问题 4。

【说明】

在线会议审稿系统（ORS：Online Reviewing System）主要处理会议前期

的投稿和审稿事务，其功能描述如下：

1.用户在初始使用系统时，必须在系统中注册（register）成为作者或审稿人。

2.作者登录（login）后提交稿件和浏览稿件审阅结果。提交稿件必须在规定

提交时间范围内，其过程为先输入标题和摘要、选择稿件所属主题类型、选择稿

件所在位置（存储位置）。上述几步若未完成，则重复；若完成，则上传稿件至

数据库中，系统发送通知。

3.审稿人登录后可设置兴趣领域、审阅稿件给出意见以及罗列录用和（或）

拒绝的稿件。

4.会议委员会主席是一个特殊审稿人，可以浏览提交的稿件、给审稿人分配

稿件、罗列录用和（或）拒绝的稿件以及关闭审稿过程。其中关闭审稿过程须包

括罗列录用和（或）拒绝的稿件。

系统采用面向对象方法开发，使用 UML 进行建模。在建模用例图时，常

用的方式是先识别参与者，然后确定参与者如何使用系统来确定用例，每个用例

可以构造一个活动图。参与者名称、用例和活动名称分别参见表 3-1、表 3-2 和

表 3-3。系统的部分用例图和提交稿件的活动图分别如图 3-1 和图 3-2 所示。

【问题 1】

根据【说明】中的描述，使用表 3-1 中的英文名称，给出图 3-1 中 A1～A4 所

对应的参与者。

【问题 2】

根据【说明】中的描述，使用表 3-2 中的英文名称，给出图 3-1 中 U1～U3 所对应的用例。

【问题 3】（4 分）

根据【说明】中的描述，给出图 3-1 中（1）和（2）所对应的关系。

【问题 4】（4 分）

根据【说明】中的描述，使用表 3-2 和表 3-3 中的英文名称，给出图 3-2 中

Action1～Action4 对应的活动。

答案：

【问题1】A1：User A2：Author A3：Reviewer A4：PCChair

【问题2】U1：list accepted/rejected papers U2：browse submitted papers U3：assign paper to reviewer 注：U2和U3的答案可互换

【问题3】（1）：<<extend>>（2）：<<include>>

【问题4】Action1：enter title and abstract Action2：select subject group

Action3：select paper location Action4：upload paper

4、阅读下列说明和图，回答问题 1 至问题 4。

【说明】

已知某唱片播放器不仅可以播放唱片，而且可以连接电脑并把电脑中的歌曲

刻录到唱片上（同步歌曲）。连接电脑的过程中还可自动完成充电。

关于唱片，还有以下描述信息：

1．每首歌曲的描述信息包括：歌曲的名字、谱写这首歌曲的艺术家以及演

奏这首歌曲的艺术家。只有两首歌曲的这三部分信息完全相同时，才认为它们是

同一首歌曲。艺术家可能是一名歌手或一支由 2 名或 2 名以上的歌手所组成的乐

队。一名歌手可以不属于任何乐队，也可以属于一个或多个乐队。

2．每张唱片由多条音轨构成；一条音轨中只包含一首歌曲或为空，一首歌

曲可分布在多条音轨上；同一首歌曲在一张唱片中最多只能出现一次。

3. 每条音轨都有一个开始位置和持续时间。一张唱片上音轨的次序是非常

重要的，因此对于任意一条音轨，播放器需要准确地知道，它的下一条音轨和上

一条音轨是什么（如果存在的话）。

根据上述描述，采用面向对象方法对其进行分析与设计，得到了如表 4-1 所

示的类列表、如图 4-1 所示的初始类图以及如图 4-2 所示的描述播放器行为的

UML 状态图。

【问题 1】（3 分）

根据说明中的描述，使用表 4-1 给出的类的名称，给出图 4-1 中的 A~F 所对

应的类。

【问题 2】（6 分）

根据说明中的描述，给出图 4-1 中（1）~（6）处的多重度。

【问题 3】（4 分）

图 4-1 中缺少了一条关联，请指出这条关联两端所对应的类以及每一端的多

重度。

答案：

【问题 1】A:artistB：song C：band D：musician E：track F：album

【问题 2】（1）0..\*（2）2..\*（3）0..1（4）1..\*（5）1..\*（6）1

【问题 3】track或E 0..1

track或E 0..1

5、阅读下列说明和图，回答问题 1 至问题 3。

[说明]

某图书管理系统的主要功能如下：

1．图书管理系统的资源目录中记录着所有可供读者借阅的资源，每项资源

都有一个唯一的索引号。系统需登记每项资源的名称、出版时间和资源状态（可

借阅或已借出）。

2．资源可以分为两类：图书和唱片。对于图书，系统还需登记作者和页数；

对于唱片，还需登记演唱者和介质类型（CD 或者磁带）。

3．读者信息保存在图书管理系统的读者信息数据库中，记录的信息包括：

读者的识别码和读者姓名。系统为每个读者创建了一个借书记录文件，用来保存

读者所借资源的相关信息。

现采用面向对象方法开发该图书管理系统。识别类是面向对象分析的第一

步。比较常的识别类的方法是寻找问题描述中的名词，再根据相关规则从这些名

词中删除不可能成为类的名词，最终得到构成该系统的类。表 5-1 给出了[说明]

中出现的所有名词。

[问题 1]

表 5-2 所给出的类并不完整，根据[说明]和表 3-1，将图 3-1 中的（a）~

（c）处补充完整。

[问题 2]

根据[说明]中的描述，给出图 5-1 中的类 CatalogItem 以及（b）、（c）处所

对应的类的关键属性（使用表 3-1 中给出的词汇），其中，CatalogItem 有 4 个

关键属性；（b）、（c）处对应的类各有 2 个关键属性。

[问题 3]

识别关联的多重度是面向对象建模过程中的一个重要步骤。根据[说明]中给

出的描述，成图 5-1 中的(1)~(6)。

答案：

[问题 1]a 资源目录 b 图书 c 唱片

[问题 2] 资源条目属性：索引号、名称、出版时间、资源状态

图书属性：作者、页数

唱片属性：演唱者、介质类型

[问题 3]（1）1 （2）0··n或0··\* (3) 0··n或0··\* (4)1

6、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]

某游戏公司现欲开发一款面向儿童的模拟游戏，该游戏主要模拟现实世界中

各种鸭子的发声特征、飞行特征和外观特征。游戏需要模拟的鸭子种类及其特征

如表 6-1 所示：

为支持将来能够模拟更多种类鸭子的特征，采用策略设计模式(Strategy)设计的类图如图

6-1 所示：

其中， Duck 为抽象类，描述了抽象的鸭子，而类 RubberDuck、 MallardDuck、

CottonDuck 和 RedHeadDuck 分别描述具体的鸭子种类，方法 fly()、quack()和

display()分别表示不同种类的鸭子都具有飞行特征、发声特征和外观特征；类

FlyBehavior 与 QuackBehavior 为抽象类，分别用于表示抽象的飞行行为与发声

行为；类 FlyNoWay 与 FlyWithWings 分别描述不能飞行的行为和用翅膀飞行

的行为；类 Quack、Squeak 与 QuackNoWay 分别描述发出“嘎嘎”声的行为、

发出橡皮与空气摩擦声的行为与不发声的行为。请填补以下代码中的空缺。

[C++代码]

#include<iostream>

using namespace （1）;

class FlyBehavior {

public : （2） fly() = 0;

};

class QuackBehavior {

public:（3） quack() = 0;

};

class FlyWithWings:public FlyBehavior{

public: void fly(){ cout << "使用翅膀飞行！" << endl; }

};

class FlyNoWay:public FlyBehavior{

public: void fly(){ cout << "不能飞行！" << endl; }

};

class Quack:public QuackBehavior{

public: void quack(){ cout << "发出\'嘎嘎\'声！" << endl; }

};

class Squeak:public QuackBehavior{

public: void quack(){ cout << "发出空气与橡皮摩擦声！" << endl; }

};

class QuackNoWay:public QuackBehavior{

public: void quack (){ cout << "不能发声！" << endl; }

};

class Duck {

protected:

FlyBehavior \* （4） ;

QuackBehavior \* （5） ;

public:

void fly(){ （6） ; }

void quack() { （7） ; };

virtual void display()=0;

};

class RubberDuck: public Duck {

public:

RubberDuck(){

flyBehavior = new （8） ;

quackBehavior = new （9） ;

}

~RubberDuck(){

if(!flyBehavior) delete flyBehavior;

if(!quackBehavior) delete quackBehavior;

}

void display(){ /\*此处省略显示橡皮鸭的代码 \*/ }

};

//其它代码省略

答案：

1 std  
2 virtualvoid  
3 virtualvoid  
4 flyBehavior  
5 quackBehavior  
6 flyBehavior->fly();  
7 quackBehavior->quack();  
8 FlyNoWay();  
9 Squeak()；

7、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]

某游戏公司现欲开发一款面向儿童的模拟游戏，该游戏主要模拟现实世界中

各种鸭子的发声特征、飞行特征和外观特征。游戏需要模拟的鸭子种类及其特征

如表 7-1 所示：

为支持将来能够模拟更多种类鸭子的特征，采用策略设计模式(Strategy)

设计的类图如图 7-1 所示：

其中， Duck 为抽象类，描述了抽象的鸭子，而类 RubberDuck 、

MallardDuck、CottonDuck 和 RedHeadDuck 分别描述具体的鸭子种类，方法

fly()、 quack()和 display()分别表示不同种类的鸭子都具有飞行特征、发声特征和

外观特征；接口 FlyBehavior 与 QuackBehavior 分别用于表示抽象的飞行行为

与发声行为；类 FlyNoWay 与 FlyWithWings 分别描述不能飞行的行为和用翅

膀飞行的行为；类 Quack、Squeak 与 QuackNoW ay 分别描述发出“嘎嘎”声

的行为、发出橡皮与空气摩擦声的行为与不发声的行为。请填补以下代码中的空

缺。

[Java 代码]

（1） FlyBehavior {

public void fly();

};

（2） QuackBehavior {

public void quack();

};

class FlyWithWings implements FlyBehavior{

public void fly(){ System.out.println("使用翅膀飞行！"); }

};

class FlyNoWay implements FlyBehavior{

public void fly(){ System.out.println("不能飞行！" ); }

};

class Quack implements QuackBehavior{

public void quack(){ System.out.println("发出\'嘎嘎\'声！" ); }

};

class Squeak implements QuackBehavior{

public void quack(){ System.out.println("发出空气与橡皮摩擦声！"); }

};

class QuackNoWay implements QuackBehavior{

public void quack (){ System.out.println("不能发声！" ); }

};

abstract class Duck {

protected FlyBehavior （3）;

protected QuackBehavior （4） ;

public void fly(){ （5） ; }

public void quack() { （6） ; };

public （7） void display();

};

class RubberDuck extends Duck {

public RubberDuck(){

flyBehavior = new （8） ;

quackBehavior = new （9） ;

}

public void display(){ /\* 此处省略显示橡皮鸭的代码 \*/ }

};

//其它代码省略

答案：

1 interface  
2 interface  
3 flybehavior  
4 quackbehavior  
5 flybehavior.fly()  
6 quackbehavior.quack()  
7 abstuact  
8 FlyNoWay  
9 squeak