# 《统一建模语言UML》模拟题2

**一、选择题，每题1分，共计20分。每个括号里的数字为该空的题号，从该题号对应的备选答案中选择最佳的一项，每空限填一个选项。请按如下表格在答题册上答此题。**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1~5题** |  |  |  |  |  |
| **6~10题** |  |  |  |  |  |
| **11~15题** |  |  |  |  |  |
| **16~20题** |  |  |  |  |  |

在几种动态图中，交互图（顺序图和通信图）适合描述 （1） 的协作行为；状态图适合描述 （2） 的行为；活动图适合描述 （3） 的总面貌。

（1） A. 单个用例中单个对象自身 B. 单个用例中多个对象之间

C. 跨越多个用例的多个对象 D. 跨越多个用例的单个对象

（2） A. 单个用例中单个对象自身 B. 单个用例中多个对象之间

C. 跨越多个用例的多个对象 D. 跨越多个用例的单个对象

（3） A. 单个用例中单个对象自身 B. 单个用例中多个对象之间

C. 跨越多个用例的多个对象 D. 跨越多个用例的单个对象

如下所示的顺序图中，Account类必须实现的方法有 （4） 。

（4） A. withdraw(anAmount )

B. checkBalance (anAmount )和log(accountAction)

C. withdraw(anAmount )、checkBalance (anAmount )和acknowledge( )

D. withdraw(anAmount )和acknowledge ( )



为了描述和理解系统中的控制机制，如为了描述一个设备控制器在不同情况下所要完成的动作， （5） 最有用。

（5） A. 状态图 B. 交互图 C. 类图 D. 活动图

对一个复杂用例中的业务处理流程进行进一步建模的最佳工具是 （6） 。

（6） A. 状态图 B. 顺序图 C. 类图 D. 活动图

顺序图应该创建在Rational Rose的 （7） 视图中。

（7） A. Use Case View B. Logical View C. Component View D. Deployment View

假设有如右侧所示类图， （8） 所示的对象图**不是**它的合法实例。

（8） A. B. C. D.

一个银行业务系统的部署图如下图所示，下面叙述不正确的是 （9） 。



（9）A. 与GUI有关的类应该部署在Branch Client上

B. 这个图表示一个三层的体系结构，不管Branch Client、Financial App Server、Database Server是运行在同一台机器上还是在不同的机器上

C. 为了系统的可伸缩性，与业务逻辑有关的对象应该部署在Financial App Server上

D. 为了系统的可伸缩性，与业务逻辑有关的对象应该部署在Branch Client上

在进行面向对象设计时，采用设计模式能够 （10） 。 （11） 模式将对象组合成树形结构以表示“部分-整体”的层次结构，并使得用户对单个对象和组合对象的使用具有一致性。

（10）A. 增强软件的易安装性 B. 改善代码的平台可移植性

C. 改善代码的可理解性 D. 复用相似问题的相同解决方案

(11) A. 观察者（Observer） 　 B. 桥接（Bridge）

C. 外观（Facade） D. 组合（Composite）

下面不是用例之间主要关系的是 （12） 。

（12） A. 扩展 B. 包含 C. 依赖 D. 泛化

（13）是一种很强的“拥有”关系，“部分”和“整体”的生命周期通常一样。整体对象完全支配其组成部分，包括它们的创建和销毁等；（14）同样表示“拥有”关系，但有时候“部分”对象可以在不同的“整体”对象之间共享，并且“部分”对象的生命周期也可以与“整体”对象不同，甚至“部分”对象可以脱离“整体”对象而单独存在。上述两种关系都是（15）关系的特殊种类。

（13） A. 聚合 B. 组合 C. 继承 D. 关联

（14） A. 聚合 B. 组合 C. 继承 D. 关联

（15） A. 聚合 B. 组合 C. 继承 D. 关联

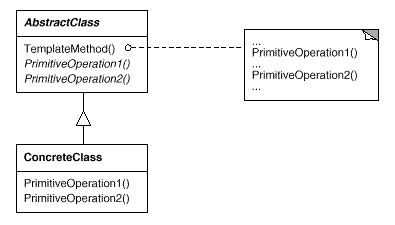
阅读如下UML类图，关于该设计的叙述中，正确的是（16）。

（16）A. 定义了一个用于创建对象的接口，由其子类来决定具体实例化哪个类的对象

B. 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

C. 定义一个操作中的算法的骨架，而将一些步骤延迟到子类中

D. 应用继承机制来实现灵活的对象创建方式，属于类创建型模式



UML中关联的多重度是指 （17） 。

（17）A. 一个类有多个方法被另一个类调用

B. 一个类的实例能够与另一个类的多个实例相关联

C. 一个类的某个方法被另一个类调用的次数

D. 两个类所具有的相同的方法和属性

下图属于UML中的 （18） ，它是UML2新增的一种表示类、组件或接口内部结构的一种图。图中所标出的①至⑥分别表示代表 （19） 。

（18）A. 对象图 B. 组件图 C. 交互概观图 D. 组合结构图

（19）A. 部件(part) 、端口(port) 、委托连接器(delegate connector) 、组装连接器 (assembly connector) 、提供接口(provide interface) 、请求接口(required interface)

B. 部件(part) 、角色(role) 、组装连接器 (assembly connector) 、 委托连接器(delegate connector)、提供接口(provide interface) 、请求接口(required interface)

C. 部件(part) 、端口(port) 、委托连接器(delegate connector) 、组装连接器 (assembly connector) 、请求接口(required interface) 、提供接口(provide interface)

D. 部件(part) 、角色(role) 、委托连接器(delegate connector) 、组装连接器 (assembly connector) 、请求接口(required interface) 、提供接口(provide interface)



⑤

①

②

③

④

⑥

UML系统分析阶段定义的包及包之间的关系描述了系统的 （20） 。

（20）A. 状态 B. 系统体系结构 C. 静态模型 D. 功能要求

**二、判断正误，每题1分，共计10分。请按如下表格在答题纸上答此题。**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1~5题** |  |  |  |  |  |
| **6~10题** |  |  |  |  |  |

1. 采用UML分析用户需求时，如果用例UC1可以出现在用例UC2**出现的**任何位置，那么UC1和UC2之间的关系是扩展关系。

2. 一个类和自身之间可以存在关联关系，表明了该类的两个对象之间的链接。在这种情况下关联终端名（角色名）是不可以省略的。

3. 用例模型是用例图的别名，通过创建参与者、用例以及它们之间的关系来表达系统的功能需求。

4. 软件架构师（Software architect）就是软件设计师（designer）,不仅负责软件架构设计，还要负责子系统设计和类设计等设计任务。

5. 浏览图和主题图是RSA（Rational Software Architect）工具中支持的两种查询图，其中浏览图捕获模型或代码状态的快照（snapshot），并存储为项目资源的一个组成部分，在项目代码变更时，可通过刷新命令对图形自动更新。

6. Analysis mechanisms are used during analysis to reduce the complexity of analysis and to improve its consistency by providing designers with a shorthand representation for complex behavior.

7. 体系结构分析的输出工件包括：分析类、分析模型和用例实现。

8. 在RSA（Rational Software Architect）中，透视图（Perspective）定义了编辑器（Editor）和视图（View）集合，并针对特定的任务和职能对其提供初始的布局。Resource、Modeling、Java和Debug都是RSA提供的透视图。

9. 在RUP中，体系结构分析（Architecture Analysis）在项目中只进行一次，主要发生在细化阶段（elaboration phase）；而用例分析（Use Case Analysis）则是在项目的每次迭代中都进行一次。

10. 在面向对象分析与设计中，控制类是应用领域中的核心类，一般用于保存系统中的信息以及提供针对这些信息的相关处理行为。

**三、完成下列各题，在答题纸上写出答案，共计25分。**

1. （5分）用一个示意图解释RUP过程模型，要求说明生命周期的四个阶段（phase）、9个工作流（discipline）、迭代的含义以及在RUP中的应用。

2. （5分）说明UML结构中构造快（Building Block），要求说明4种事物（Things）、4种重要关系（Relationship）以及至少10种图（diagram），并指明那些图是表达系统行为的，哪些图是表达系统结构的。

3.（3分）举例说明什么是Stereotype（构造型）以及构造型的三种表达方式。

4. （3分）以下图所示的类图片段为例，说明对象模型与数据模型，以及类图与源程序之间的对应关系。（注：①请根据该类图片段给出数据库表table，并在其上标出主键和外键；②用C++或Java语言给出与该类图片段对应的程序片段）



5. （3分）简要说明体系结构分析(Architecture Analysis)的目的和主要步骤。

6. （3分）说明是用例实现（Use-case Realization），用例实现结构中应该包含哪些图。

7. （3分）类之间的依赖关系在程序中有多种不同的实现，任意写一段程序代码（Java或C++），用以指明至少一种实现方式。

**四、综合应用题（45分）**

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

　　【说明】

　　某大学拟开发一个用于管理学术出版物（Publication） 的数字图书馆系统，用户可以从该系统查询或下载己发表的学术出版物。系统的主要功能如下：

　　1.登录系统。系统的用户 （User）仅限于该大学的学生 （Student） 、教师（Faculty）和其他工作人员（Staff） 。在访问系统之前，用户必须使用其校园账户和密码登录系统。

　　2. 查询某位作者 （Author）的所有出版物。系统中保存了会议文章 （ConfPaper）、期刊文章 （JournalArticle）和校内技术报告（TechReport ）等学术出版物的信息，如题目、作者以及出版年份等。除此之外，系统还存储了不同类型出版物的一些特有信息；

　　（1）对于会议文章，系统还记录了会议名称、召开时间以及召开地点；

　　（2）对于期刊文章，系统还记录了期刊名称、出版月份、期号以及主办单位；

　　（3）对于校内技术报告，系统记录了由学校分配的唯一ID。

　　3. 查询指定会议集（Proceedings）或某个期刊特定期（Edition） 的所有文章。会议集包含了发表在该会议（在某个特定时间段、特定地点召开）上的所有文章。期刊的每一期在特定时间发行，其中包含若干篇文章。

　　4.下载出版物。系统记录每个出版物被下载的次数。

　　5.查询引用了某篇出版物的所有出版物。在学术出版物中引用他人或早期的文献作为相关工作或背景资料是很常见的现象。用户也可以在系统中为某篇出版物注册引用通知，若有新的出版物引用了该出版物，系统将发送电子邮件通知该用户。

　　现在采用面向对象方法对该系统进行开发，得到系统的初始设计类图如图1所示。



图1 初始设计类图

【问题1】（5分）根据说明中的描述，给出图1中C1~C9所对应的类名。

【问题2】（2分）根据说明中的描述，给出图1中类C6~C9的属性。

【问题3】（3分）第五项需求中提到“用户也可以在系统中为某篇出版物注册引用通知，若有新的出版物引用了该出版物，系统将发送电子邮件通知该用户”，在图1添加适当的类和关系，并适当添加类的操作和对操作的注释，使之成为观察者模式（Observer）的一个应用实例，并解释哪个类（或哪几个类）扮演了Observer角色，哪个类（或哪几个类）扮演了Subject角色，改进后的设计有什么优点。

2. 如图2所示是“海洋生物知识介绍软件”的初始设计。利用该软件，学生可以选取一种海洋生物，然后观看该海洋生物的生长过程与生长环境方面的视频、音频或文字介绍。当前该程序只能进行金枪鱼（Tuna）、鲑鱼（Salmon）、鲍鱼（Abalone）与扇贝（Scallop）的介绍。这4类有一个共同的方法show()。该方法用于介绍该类海洋生物的生活习性等，可能是视频、音频或文字介绍。MarineLife类负责选择一种海洋生物，其chooseFish()和chooseShellfish()方法负责选择一种海洋鱼类（Tuna，Salmon）或者贝类（Abalone，Scallop）。该类的另外一个方法display()负责调用相应的类的show()方法，用于介绍所选的海洋生物。

[问题 1]（2 分）举例说明该设计有何不足之处；

[问题 2]（3 分）按照自己的想法，重新设计，并且画出新的设计类图。（注：新设计至少应该包括两个接口类，一个是鱼类，一个是贝类）



图2 “海洋生物知识介绍软件”类图

3. （5分）某大学的学生注册系统的用例模型中包含一个名为“Change Student Information”（修改学生信息）的用例，允许学生修改自己的个人信息，具体的活动步骤为：学生登录系统，填写Web表单并单击Submit（提交）按钮，该表单发送给Web服务器，然后由它把数据传递给大型机，大型机访问STUDENT数据库，然后把“Not Found”消息或所选的学生数据传递给Web服务器，也就是说，如果没有找到学生记录，则Web服务器在Web页上显示一则错误信息，如果找到了学生记录，则Web服务器格式化一个新的Web页，在一个Web表单中包含当前学生数据。

如果学生在Web页表单上输入修改数据，并单击Submit按钮，则修改数据传输到服务器，并且一个确认修改数据的程序开始运行。如果修改数据有错误，则发送一则错误消息给该Web页。如果修改数据有效，则更新学生记录，并写入Change Student Journal Record。一次有效更新以后，把一个确认Web页发送给浏览器，继而活动终止。

按照上述内容，完成如图3所示的活动图。（请根据下表中的活动名称，填写活动图中的序号所对应的活动。注：(1) 9和10所代表的活动可替换；(2)下表中的“显示错误消息”出现两次）

|  |  |  |
| --- | --- | --- |
| Create Change Student Journal Record | Display Confirmation | Receive Web Form |
| Enter Changes | Display Error Message | Update Student Record |
| Display Current Student Data | Logon | Get Student Record |
| Display Error Message | Validate changes |  |



图3 修改学生信息相关活动图

4. (18分) Musixscore.com为用户提供音乐的在线服务。在“browse music”网页上，用户从下拉列表中选择音乐类型。网页使用Ajax获得一系列与该类型匹配的演奏家、音乐家或团体，并用下拉列表进行格式化显示。从演奏家下拉列表中选择一个选项后，Web页使用Ajax增加第3个下拉列表，显示该演奏家的所有CD和音乐作品。选择某个CD后，Web页使用Ajax产生第4个下拉列表，包含该CD内的所有歌曲，用户可以多重选择。单击“Add To Shop Cart（添加到购物车）”图片后，所选的歌曲会添加到购物车。用户可以任意改变下拉列表以选择其他音乐，方法类似。

（1）为Browse Music编写一个用例规约，其事件流根据上述描述进行定义。

（2）使用边界对象、控制对象和实体对象绘制一个顺序图。

（3）编写一个列表，包含所有被传递到这些类的实例的消息，包括名称、参数和数据类型以及返回值的数据类型，可以作出有关数据的任何假设。

（4）根据你画的顺序图创建相应的协作图。

（5）创建一个初始的VOPC。

3. 阅读下列说明和程序代码，将应填入（n）处的字句填写完整。**（问题1和问题2任选一题完成）**

策略模式是一种对象行为型模式，它定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。该模式使得算法可独立于使用它的客户而变化。其结构如图4所示：

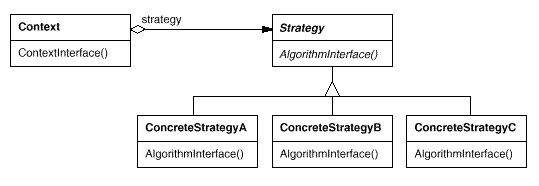


图4 策略模式类图

某软件公司现欲开发一款飞机飞行模拟系统，该系统主要模拟不同种类飞机的飞行特征与起飞特征。需要模拟的飞机种类及其特征见表3。

表3 需模拟的飞机种类及其特征

|  |  |  |
| --- | --- | --- |
| 飞机种类 | 起飞特征 | 飞行特征 |
| 直升机(Helicopter) | 垂直起飞(VerticalTakeOff) | 亚音速飞行(SubSonicFly) |
| 客机(AirPlane) | 长距离起飞(LongDistanceTakeOff) | 亚音速飞行(SubSonicFly) |
| 歼击机(Fighter) | 长距离起飞(LongDistanceTakeOff) | 超音速飞行(SuperSonicFly) |
| 鹞式战斗机(Harrier) | 垂直起飞(VerticalTakeOff) | 超音速飞行(SuperSonicFly) |

为支持将来模拟更多种类的飞机，采用策略设计模式设计的类图如图5所示。

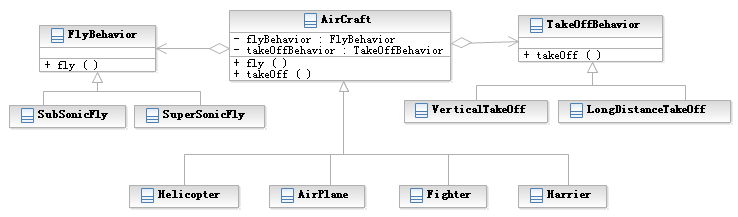


图5 飞机飞行模拟系统类图

在图5中，AirCraft为抽象类，描述了抽象的飞机，而类Helicopter、AirPlane、Fighter和Harrier分别描述具体的飞机种类，fly（）和takeoff（）分别表示不同飞机都具有飞行特征和起飞特征；类FlyBehavior与TakeOffBehavior为抽象类，分别用于表示抽象的飞行行为与起飞行为；类SubSonicfly与SuperSonicFly分别描述亚音速飞行和超音速飞行的行为；类VerticalTakeOff和LongDistanceTakeOff分别描述垂直起飞与长距离起飞的行为。

**【问题1】（7分）Java代码**

**interface** FlyBehavior {

**public** **void** fly();

};

**class** SubSonicFly **implements** FlyBehavior{

**public** **void** fly() { System.*out*.println("亚音速飞行！"); }

};

**class** SuperSonicFly **implements** FlyBehavior{

**public** **void** fly() { System.*out*.println("超音速飞行"); }

};

**interface** TakeOffBehavior {

**public** **void** takeOff();

};

**class** VerticalTakeOff **implements** TakeOffBehavior{

**public** **void** takeOff() { System.*out*.println("垂直起飞！"); }

};

**class** LongDistanceTakeOff **implements** TakeOffBehavior{

**public** **void** takeOff() { System.*out*.println("长距离起飞！"); }

};

**abstract** **class** AirCraft{

**protected** （1） ;

**protected** （2） ; //(1)、(2)可互换

**public** **void** fly() { （3） ;}

**public** **void** takeOff() { （4） ;}

};

**class** Helicopter （5） Aircraft{

**public** Helicopter() {

flyBehavior= **new** （6） ;

takeOffBehavior=**new** （7） ;

}

}

//其他代码省略

**【问题2】（7分）C++代码**

#**include** <iostream>

**using** **namespace** std;

**class** FlyBehavior {

**public**: **virtual void** fly()=0;

};

**class** SubsonicFly: **public** FlyBehavior {

**public**: **void** fly() { cout << "亚音速飞行！" << endl; }

}；

**class** SuperFly: **public** FlyBehavior {

**public**: **void** fly() { cout << "超音速飞行！" << endl; }

}；

**class** TakeOffBehavior {

**public**: **virtual void** TakeOff()=0;

};

**class** VerticalTakeOff: TakeOffBehavior {

**public**: **void** TakeOff() { cout << "垂直起飞！" << endl; }

};

**class** LongDistanceTakeOff {

**public**: **void** TakeOff() { cout << "长距离起飞！" << endl; }

};

**class** AirCraft {

**protected**:

(1) ;

(2) ;//(1)、(2)可互换

**public**:

**void** fly() { (3) ;}

**void** TakeOff() { (4) ;}

};

**class** Helicopter: **publi**c Aircraft {

**public**:

Helicopter() {

flyBehavior = **new** (5) ;

takeOffBehavior = **new** (6) ;

}

(7) {

if (!flyBehavior) **delete** flyBehavior;

if (!takeOffBehavior) **delete** takeOffBehavior;

}

};

//其他代码省略