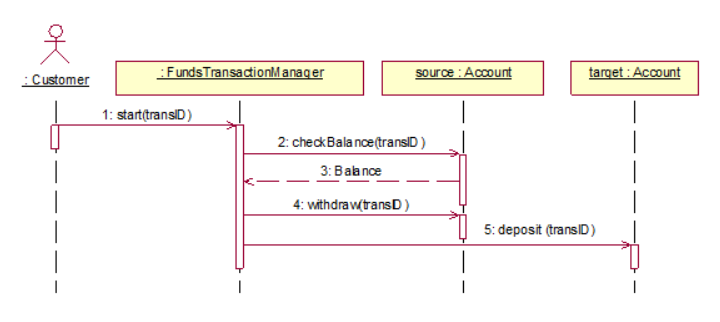
2024-软件工程-作业9

题量: 12 满分: 100 作答时间:11-06 08:51至11-16 08:51

一. 单选题（共4题，16分）

1. (单选题, 4分) 如下所示的顺序图中，Account类必须实现的方法有（ ） 。



A、start ( )

B、checkBalance ( ) 和 withdraw ( )

C、deposit ( )

**D、checkBalance ( )、 withdraw ( ) 和deposit ( )**

2. (单选题, 4分) 在面向对象分析中按照构造型可以将分析类分为三种：边界类、实体类和控制类，下面关于分析类的描述中不正确的是（ ）。

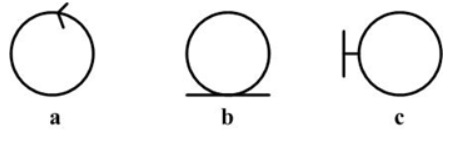
A实体类主要用来描述必须存储的信息，以及与这些信息直接相关的操作；边界类用来封装系统和外部要素间交互的边界；控制类封装Use Case中行为的协调。

B边界类把系统其他部分（实体类、控制类）与外部环境隔离，当系统与外界环境之间的交互发生变化时，只需要修改边界类。

C控制类把Use Case特有的行为与系统其他部分（实体类、边界类）隔离开来；从而，实体类和边界类有可能跨越多个Use Case。

**D实体类与系统外部环境以及特定Use Case的控制逻辑要弱耦合，实体类对象通过调用控制类对象和边界类对象的操作来完成Use Case的功能。**

1. (单选题, 4分) 在UML图形语言中，下图中a、b、c三种图形符号按照顺序分别表示（ ）。



A边界对象、实体对象、控制对象

B实体对象、边界对象、控制对象

**C控制对象、实体对象、边界对象**

D边界对象、控制对象、实体对象

4. (单选题, 4分) 以下关于分析类的叙述中，错误的是（ ）。

A“分析类”是概念层的内容，从它们可捕获系统对象模型的雏形。

B“分析类”相当粗略，并且不要往其中添加技术细节。

C按照职责可以将分析类划分为：实体类、边界类和控制类。

**D边界类描述系统外部环境与内部运作之间的交互，为此要确定系统对用户的响应或消息，并要对界面的可视方面建模。**

二. 判断题（共1题，4分）

5. (判断题, 4分) 在面向对象分析与设计中，控制类是应用领域中的核心类，一般是系统内对象和系统外参与者的联系媒介；边界类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；实体类主要是协调上述两种类或对象之间的交互。

A对

**B错**

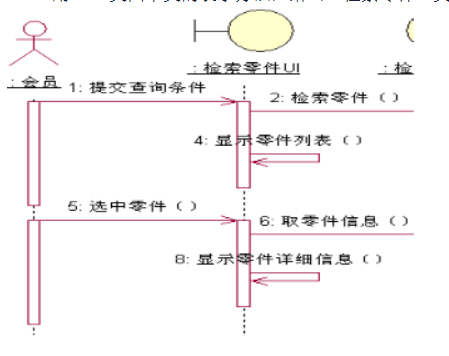
三. 简答题（共5题，55分）

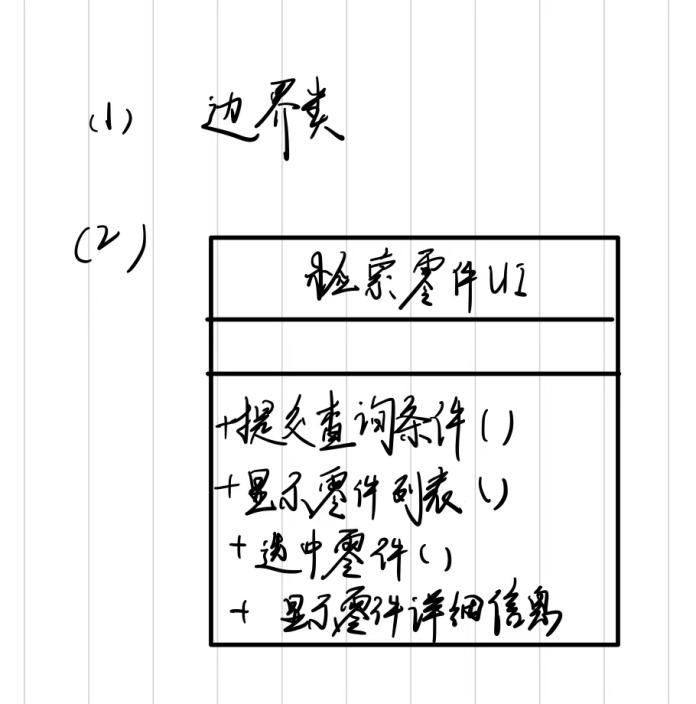
6. (简答题, 10分)

根据下面的部分顺序图，回答如下问题：

（1）“检索零件UI”类属于哪种分析类（边界类、实体类、控制类）？

1. 用UML类图中类的表示方法，给出“检索零件UI类”的图形表示，要求表示它的名字和职责。





7. (简答题, 10分) 如图所示类图为某软件工程师所设计的航空公司的“乘客账户管理程序”片段。乘客暂时分为金卡乘客和银卡乘客。每次乘客乘坐该航空公司的飞机都可以获得一定的分数。积分较低的乘客为银卡客户，拥有SilverAcct；当某乘客的积分达到一定分数的时候，自动升级为金卡客户，拥有GoldenAcct。在该设计中的账户管理类AccountManagement包含银卡账户与金卡账户。在该类中，创建SilverAcct或者GoldenAcct对象，然后决定是否给予该客户机票折扣，或者其他的奖励。

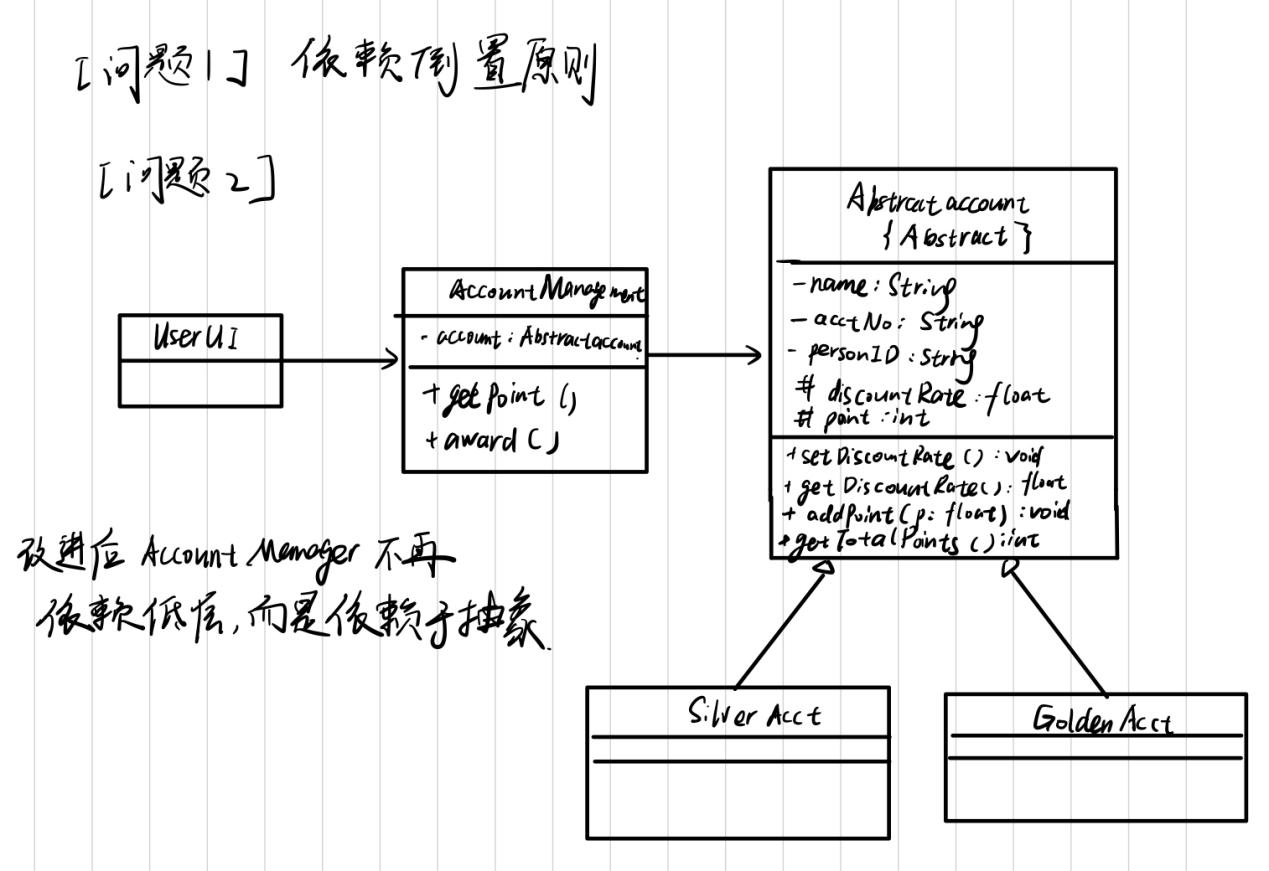
观察此设计可以发现，虽然使用的是面向对象的设计形式，但是采用了结构化的设计思想，从而设计结果缺乏面向对象系统应有的优点。

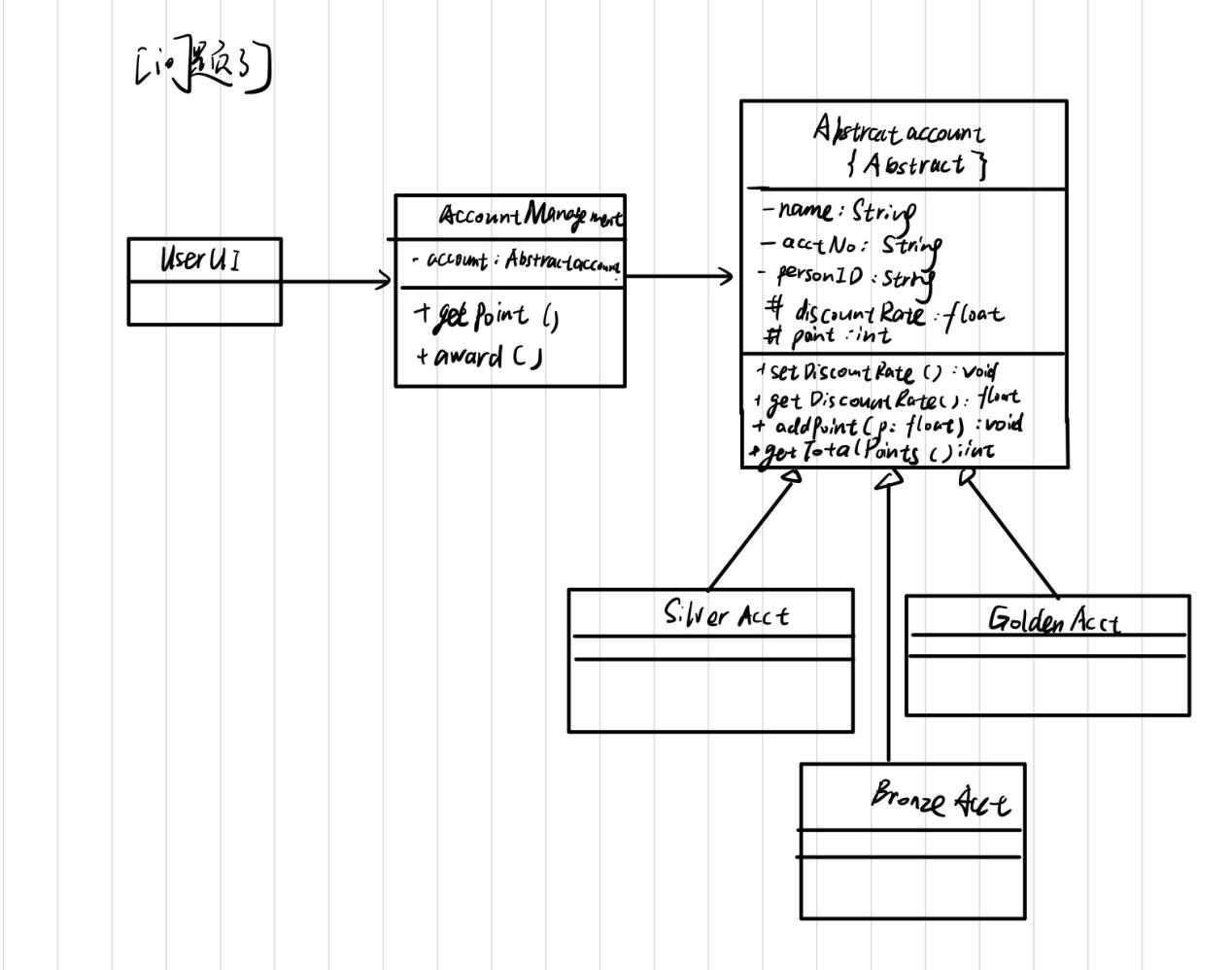
【问题1】该设计违反了哪一个面向对象设计原则？（可选项：单一职责原则、接口隔离原则、依赖倒转原则、李氏替换原则、开闭原则）

【问题2】重新设计该系统，即给出改进后的类图，并说明重新设计的系统拥有哪些优点。

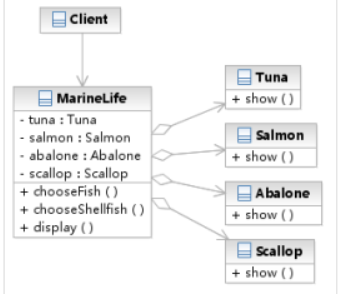
【问题3】如需增加一种铜卡客户，拥有BronzeAcct，那么在设计改进后的类图上如何体现？（给出改进后的类图）





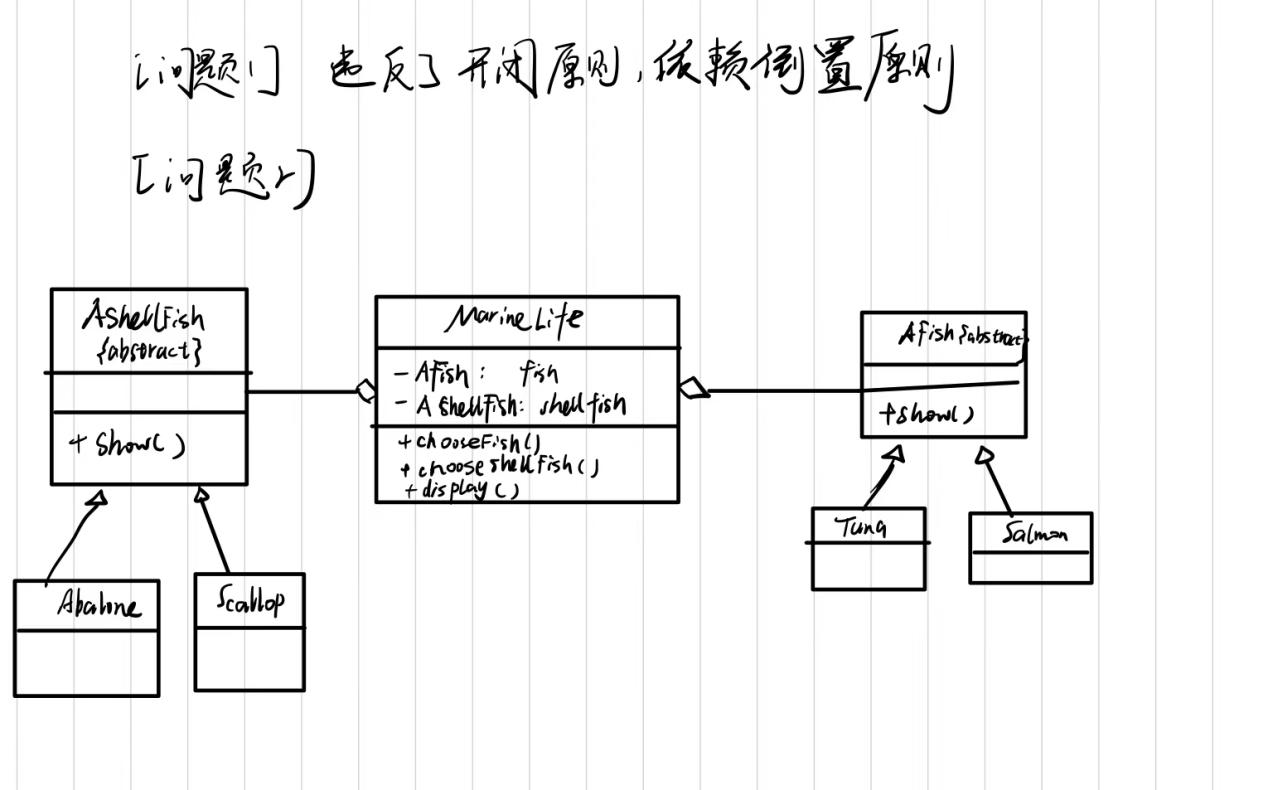


1. (简答题, 10分) 如图所示是“海洋生物知识介绍软件”的初始设计。利用该软件，学生可以选取一种海洋生物，然后观看该海洋生物的生长过程与生长环境方面的视频、音频或文字介绍。当前该程序只能进行金枪鱼（Tuna）、鲑鱼（Salmon）、鲍鱼（Abalone）与扇贝（Scallop）的介绍。这4类有一个共同的方法show()。该方法用于介绍该类海洋生物的生活习性等，可能是视频、音频或文字介绍。MarineLife类负责选择一种海洋生物，其chooseFish()和chooseShellfish()方法负责选择一种海洋鱼类（Tuna，Salmon）或者贝类（Abalone，Scallop）。该类的另外一个方法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]举例说明该设计有何不足之处；

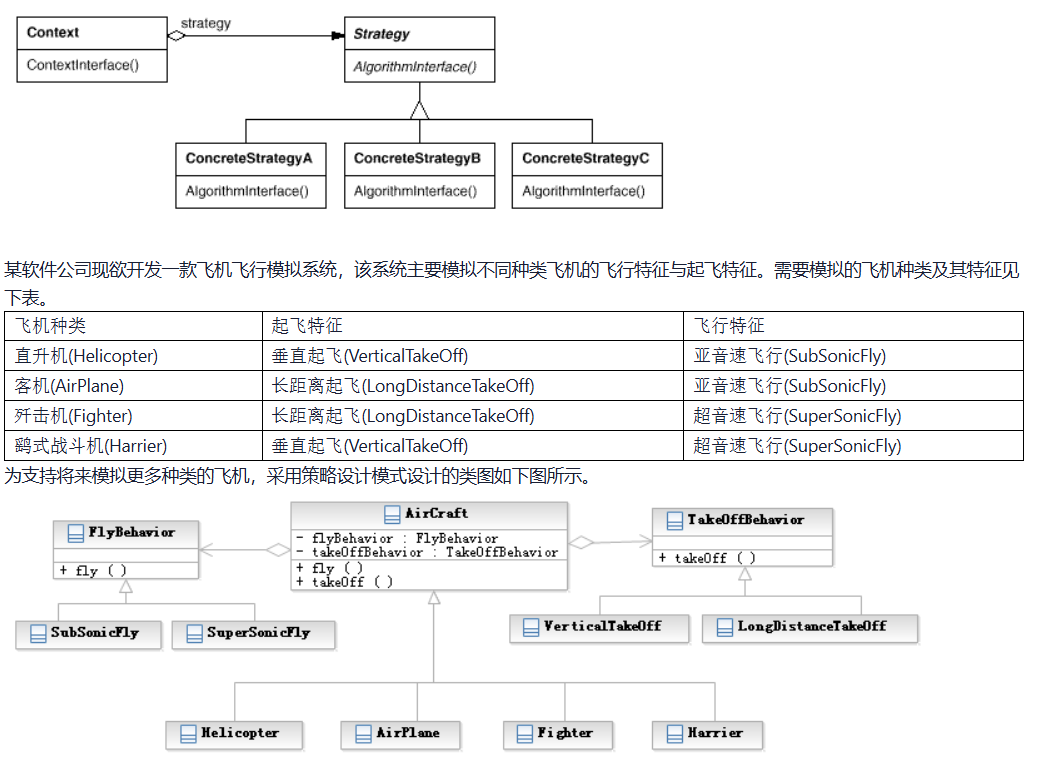
[问题 2]按照自己的想法，重新设计，并且画出新的设计类图。（注：新设计至少应该包括两个接口类，一个是鱼类，一个是贝类）



9. (简答题, 15分)

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。其结构如下图所示：



在上图中，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】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】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: public Aircraft {

public:

Helicopter() {

flyBehavior = new (5) ;

takeOffBehavior = new (6) ;

}

(7) {

if (!flyBehavior) delete flyBehavior;

if (!takeOffBehavior) delete takeOffBehavior;

}

};

//其他代码省略

【问题1】

（1）FlyBehavior flybehavior

（2）TakeOffBehavior takeOffBehavior

（3）flyBehavior.fly()

（4）takeOffBehavior.takeOff()

（5）extends

（6）SubsonicFly0

（7）VerticalTakeOff()

【问题2】

（1）FlyBehavior\* flyBehavior

（2）TakeOffBehavior\* takeOffBehavior

（3）flyBehavior->fly()

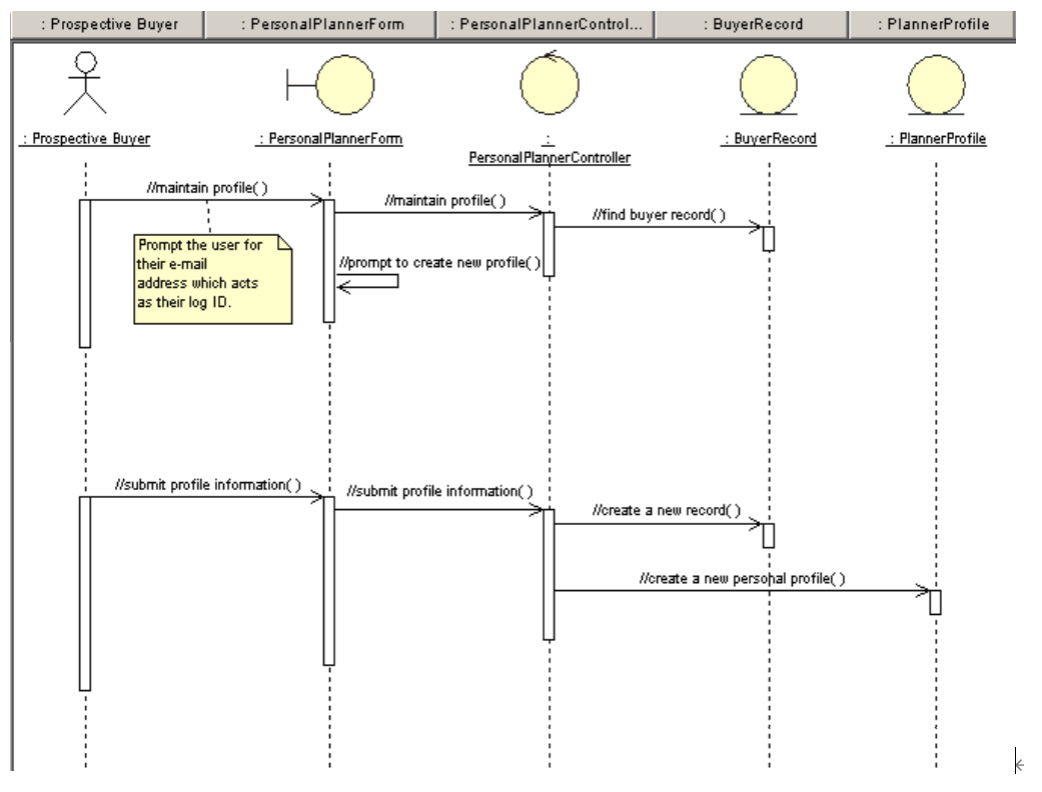
（4）takeOffBehavior->takeOff()

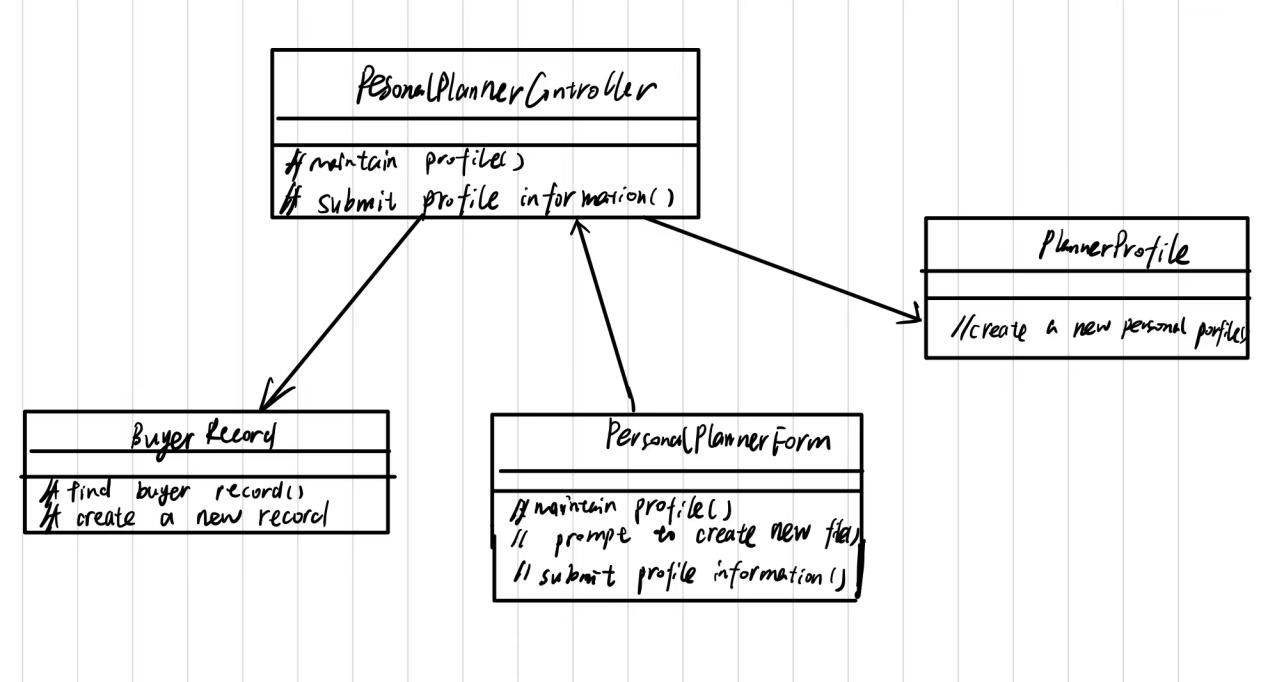
（5）SubSonicFly()

（6）VerticalTakeOff()

（T）~Helicopter()

10.(简答题, 10分) 根据如下的顺序图（sequence diagram），识别类、类的职责，以及类之间的关系，进而画出参与类图（VOPC）。





11. (完型填空, 5分) 在面向对象分析与设计中，（ 1 ）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；（ 2 ）是系统内对象和系统外参与者的联系媒介；（ 3 ）主要是协调上述两种类或对象之间的交互。

(1)

A分析类

B控制类

C边界类

**D实体类**

(2)

A分析类

B控制类

**C边界类**

D实体类

(3)

A分析类

**B控制类**

C边界类

D实体类

12. (完型填空, 20分) 开-闭原则（Open-Close Principle，OCP）是面向对象的可复用设计的基石，开-闭原则是指一个软件实体应当对 （ 1 ） 开放，对 （ 2 ） 关闭；里氏替换原则（Liskov Substitution Principle，LSP）是指任何 （ 3 ） 可以出现的地方， （ 4 ） 一定可以出现。

(1)

A修改

**B扩展**

C分析

D设计

(2)

**A修改**

B扩展

C分析

D设计

(3)

A常量

B变量

**C基类对象**

D子类对象

(4)

A常量

B变量

C基类对象

**D子类对象**