Student ID_____Student Name_____

Advanced Software Development DE Final Exam

June 25, 2016

PRIVATE AND CONFIDENTIAL

1. **Allotted exam duration is 2 hours.**
2. **Closed book/notes.**
3. **No personal items including electronic devices (cell phones, computers, calculators, PDAs).**
4. **Cell phones must be turned in to your proctor before beginning exam.**
5. **No additional papers are allowed. Sufficient blank paper is included in the exam packet.**
6. **Exams are copyrighted and may not be copied or transferred.**
7. **Restroom and other personal breaks are not permitted.**
8. **Total exam including questions and scratch paper must be returned to the proctor.**

7 blank pages are provided for writing the solutions and/or scratch paper. All 7 pages must be handed in with the exam

BE VERY CAREFUL WITH THE GIVEN 2 HOURS AND USE YOUR TIME WISELY. THE ALLOTED TIME IS GIVEN FOR EVERY QUESTION.

Write your name and student id at the top of this page.

Question 1 [30 points] {40 minutes}

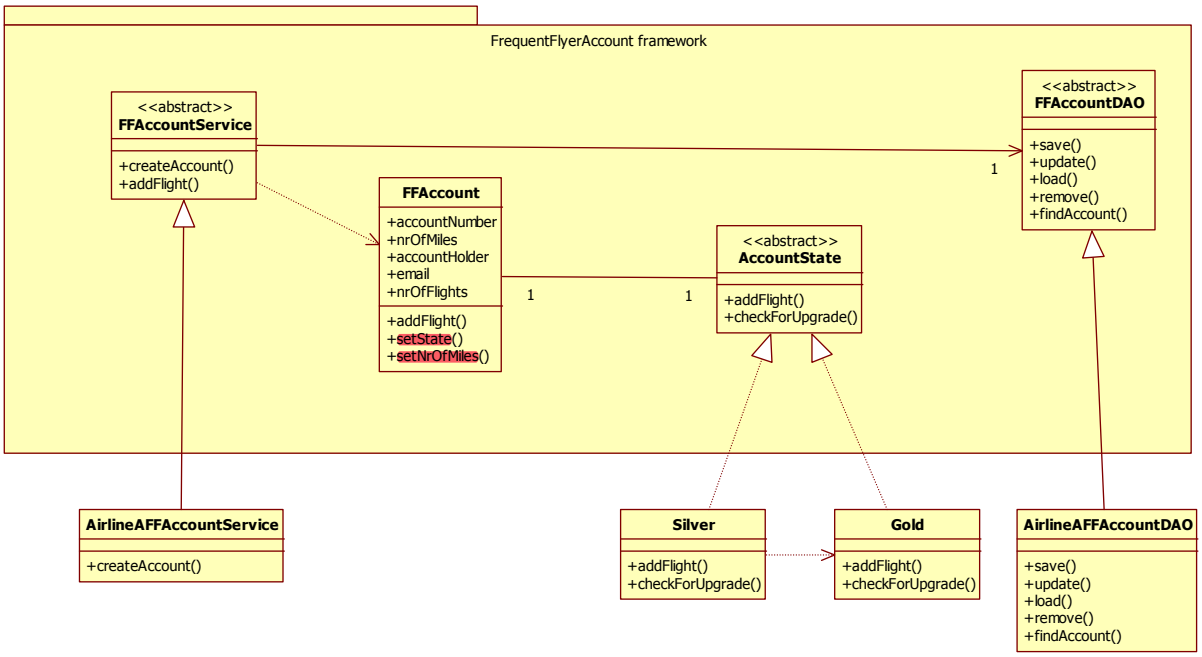
Your company writes software for different airlines, and airline A requests you to develop an Frequent Flyer Program application with the following requirements:
There are 2 types of accounts, "silver" and "gold".
Everyone starts with a "silver" account.
When you have more than 10.000 miles or more than 15 flights, you are upgraded to a "gold" account.
Silver accounts receive the same number of miles as the actual miles of their flights.
Gold accounts receive 2 times the number of miles as the actual miles of their flights.

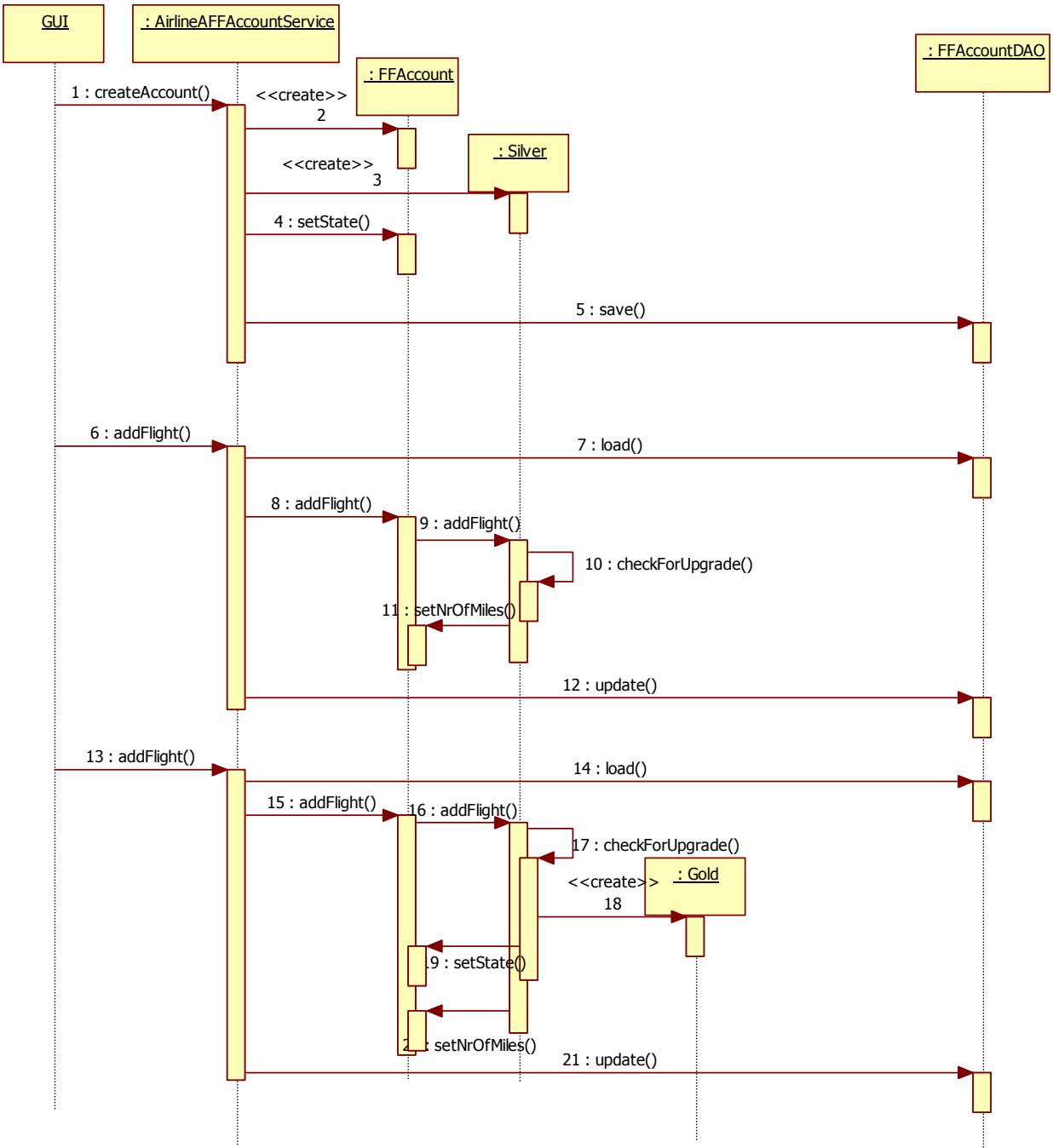
Another airline, airline B, also wants a Frequent Flyer Program application, but it wants more account types (bronze, silver, gold, platinum) and it has different business rules regarding upgrades and bonus miles.
It is very likely that other airlines also want a Frequent Flyer Program application.

You decide to write a generic Frequent Flyer Program framework with the following requirements:
The framework should support functionality for adding and removing accounts

- An account has the following properties: accountNumber, nrOfMiles, accountHolder, email.
- All accounts are stored in the database
- It should be easy to add new account types with its own business rules regarding upgrades and bonus miles.

a. Draw the class diagram of your Frequent Flyer Program framework. **Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc) to communicate the important parts of your design.** Make use of all the best practices we learned in this course.
b. Within the class diagram of part a, draw the necessary classes that you need to implement the Frequent Flyer Program application for airline A. Show clearly which classes are within the framework and which classes are outside the framework.
c. Draw the sequence diagram with the following scenario for airline A:
    1. A new account is created for customer 1.
    2. Customer 1 travels 9.000 miles, and these miles are added to her account.
    3. Customer 1 travels 3.000 miles, and these miles are added to her account.


FrequentFlyerAccount framework
<<abstract>>
FFAccountService
+createAccount()
+addFlight()
FFAccount
+accountNumber
+nrOfMiles
+accountHolder
+email
+nrOfFlights
+addFlight()
+setState()
+setNrOfMiles()
1
1
<<abstract>>
AccountState
+addFlight()
+checkForUpgrade()
<<abstract>>
FFAccountDAO
+save()
+update()
+load()
+remove()
+findAccount()
1
AirlineAFFAccountService
+createAccount()
Silver
+addFlight()
+checkForUpgrade()
Gold
+addFlight()
+checkForUpgrade()
AirlineAFFAccountDAO
+save()
+update()
+load()
+remove()
+findAccount()

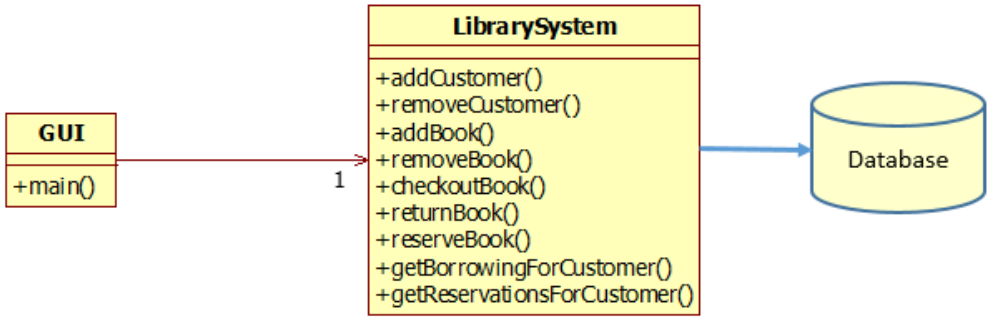

GUI
: AirlineAFFAccountService
: FFAccount
: FFAccountDAO
: Silver
: Gold
1 : createAccount()
<<create>>
2
<<create>>
3
4 : setState()
5 : save()
6 : addFlight()
7 : load()
8 : addFlight()
9 : addFlight()
10 : checkForUpgrade()
11 : setNrOfMiles()
12 : update()
13 : addFlight()
14 : load()
15 : addFlight()
16 : addFlight()
17 : checkForUpgrade()
<<create>>
18
19 : setState()
: setNrOfMiles()
21 : update()

Question 2 [35 points] {40 minutes}

Three years ago we wrote a library application with the following business requirements:

- The application should record customer information: name, phone, email, street, city, zip.
- The application should record book information: title, isbn, author.
- The application records the borrowing of every book. It records the checkout-date when the customer takes a book home and it records the return-date and computes the possible fee when the customer returns a book.
- When a customer checks out a book we should be able to record (checkout-date, return-date, fee) and handle the different books that a customer borrows.
- We should be able to record and handle the reservations of a customer.
- We can have multiple copies of the same book title. Every copy has a unique scancode.

This library system has the following design:



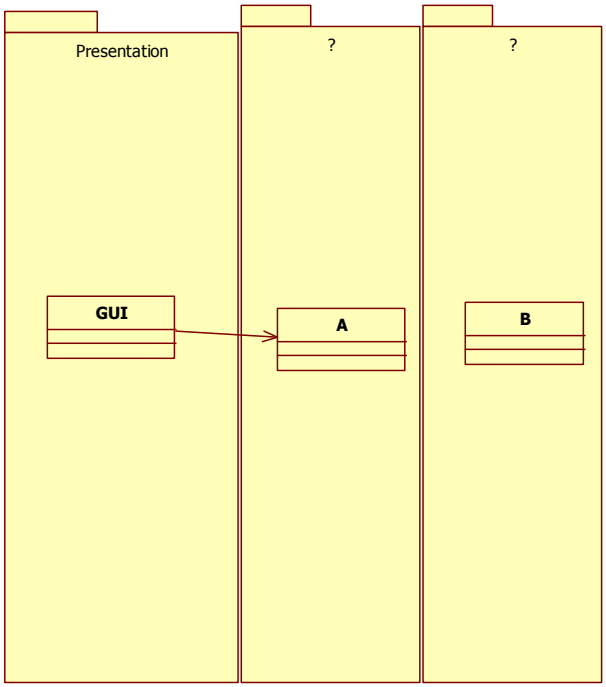
All implementation code is written in 1 class.
Now we have to redesign this library application with the following additional requirements:

- In the existing system we can have only one copy of every book. In the new system it should be possible to have multiple copies of a certain book.
- We will use the Spring framework to implement this application.
- Whenever a customer returns a book, the system will send an email to the first customer that reserved this book so that this customer know the book is available.
- We want to log every action of our application.

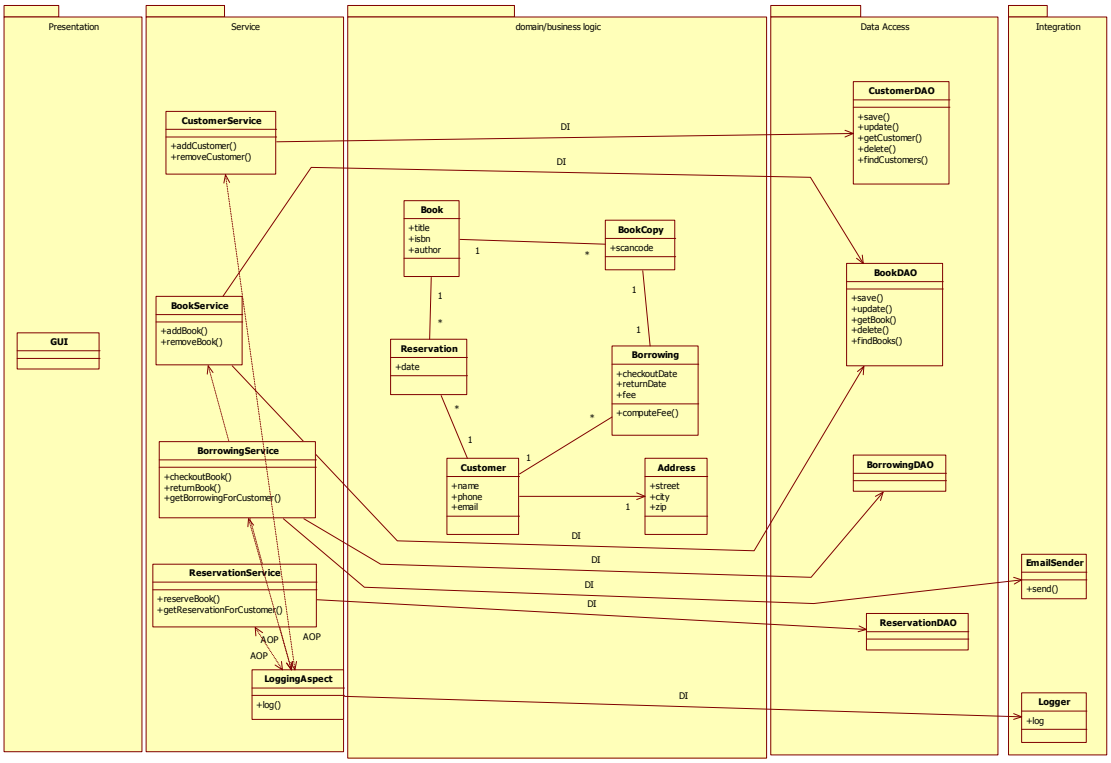
Draw the **UML class diagram** of the new library application. **Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc) to communicate the important parts of your design.** Make use of all the best practices we learned in this course.
Show very clearly where you use Dependency Injection (DI), AOP or certain design patterns.
To show where to use DI, use the following notation:

To show where to use AOP, use the following notation:

In your UML class diagram show the different layers of your design, and place the classes in the particular layers, similar as the solution of the last shopping application lab. For the GUI you can use just one GUI class that represents the complete GUI implementation.
So your class diagram should look like the example below (see next page).

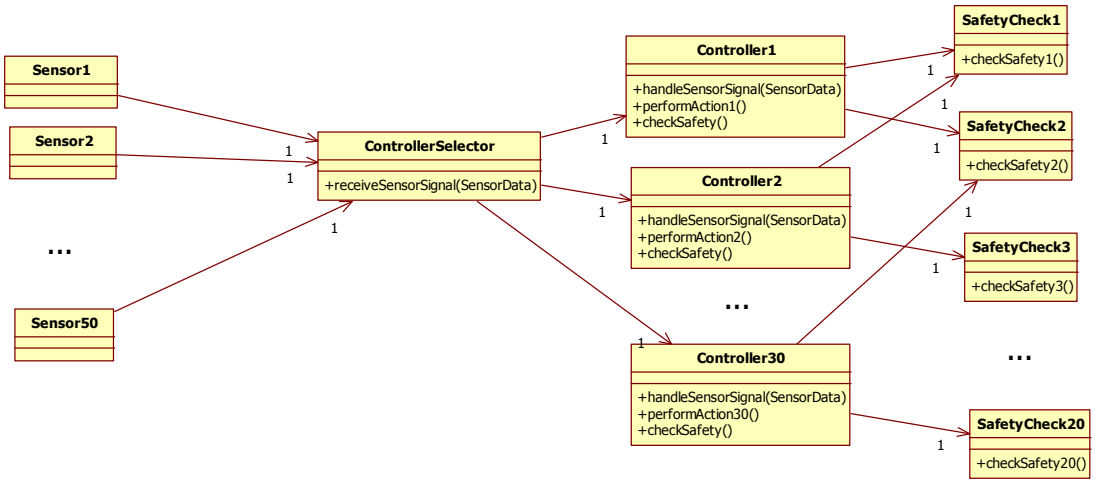


Solution


Presentation
Service
domain/business logic
Data Access
Integration
GUI
CustomerService
+addCustomer()
+removeCustomer()
BookService
+addBook()
+removeBook()
BorrowingService
+checkoutBook()
+returnBook()
+getBorrowingForCustomer()
ReservationService
+reserveBook()
+getReservationForCustomer()
LoggingAspect
+log()
Book
+title
+isbn
+author
BookCopy
+scancode
Reservation
+date
Borrowing
+checkoutDate
+returnDate
+fee
+computeFee()
Customer
+name
+phone
+email
Address
+street
+city
+zip
CustomerDAO
+save()
+update()
+getCustomer()
+delete()
+findCustomers()
BookDAO
+save()
+update()
+getBook()
+delete()
+findBooks()
BorrowingDAO
ReservationDAO
EmailSender
+send()
Logger
+log
DI
AOP
1
*

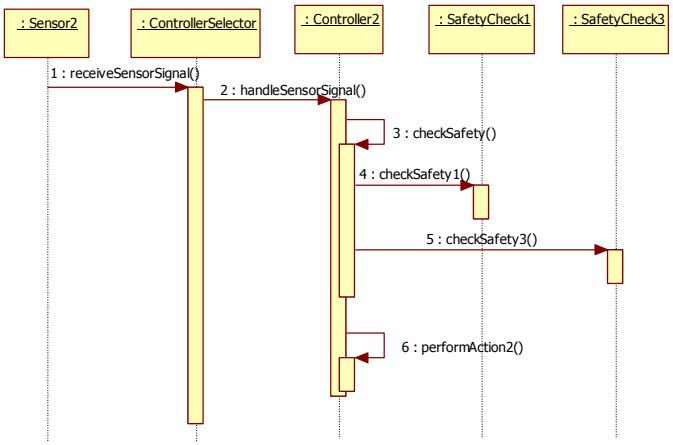
Question 3 [30 points] {40 minutes}

Suppose your company needs to design the software for the newest electrical car from Toyota. One of your colleagues came up with the following design:



The car has many sensors (around 50 sensors). The sensor signals are send to the ControllerSelector. The ControllerSelector then calls the necessary Controllers to handle this sensorData. The car has around 30 of these controllers. Sometimes one sensor signal needs to be handled by one controller, but most of the times one sensor signal needs to be handled by multiple controllers.

Before the controller can perform the necessary action it first has to do a number of safety checks. When all the safety checks are OK, then the controller is able to execute the necessary action.
Here you see an example scenario:



Sensor2 sends a sensor signal to the ControllerSelector. The ControllerSelector decides that Controller2 needs to handle this signal. Controller2 first does safetycheck1 and safetycheck2, and then performs the necessary action2.
It is important for Toyota that they can easily add new sensors, controllers and safety checks so that this software can be reused in many different cars.

The problem is that your colleague did not study design patterns and framework design. Luckily you took the ASD course, and you can help you colleague to make a better design.

a. Draw the **UML class diagram** of your design. **Make sure you add all necessary UML elements (interfaces, abstract classes, attributes, methods, multiplicity, etc) to communicate the important parts of your design.** Make use of all the best practices we learned in this course.
b. Draw the sequence diagram of the same scenario that is given in this exercise, but then with your design.

Solution