Year 2011 Semester 1

Programming 1 (COSC1073/2362)

Assignments 3

Ass #	*submission mode*	*Due Date*	*Weight*
Ass 3	**submission/demo**	**week 12 - 11:59pm Sunday May 29**	**12 %**
	(demo will be in week 13 – dates/times TBA)		

Assignments 3 (Demo based on Final-Submission - 120 Marks)
(Covers Chapters 1 to 12)

Flight Management System

Overview

You are required to take the trial flight management program described in assignments 2A and 2B and extend its functionality to manage all of the common operations, taking into account passenger related functionality in the QuickJet system.

Passengers are classified into two main categories, economy class (a basic passenger ticket with a fixed maximum luggage allowance) and business class (a premium ticket which includes a flexible luggage allowance and a complimentary in-flight meal of the passenger's choosing).

Hence you are required to write an abstract class named **Passenger** with subclasses **EconomyPassenger** and **BusinessPassenger**.

You are also required to enhance the Flight and RegionalFlight classes developed in the assignments 2A and 2B. Exceptions must be thrown in the Flight/RegionalFlight classes when an attempt is made to open, close or reserve seats for a flight incorrectly. The functionality for delaying flights should also be updated to incorporate exception handling, both for not being in the correct state to delay and when the flight is cancelled. You may either use the standard Exception class or create your own exception class when incorporating exception handling into the functionality for opening, closing, delaying and reserving seats for a flight.

In all cases any exception thrown by any of the operations described above must be handled in an appropriate fashion in the application (ManageHiring) class described below

The final part requires you to write a class named **ManageFlights** that uses **Passenger** (and its subclasses), and **Flight** (and it subclass) objects to facilitate the common operations. It also requires you to read and write Passenger and Flight objects to text files in a manner that allows all flight and passenger details to be loaded back into the program when it is started up again.

You are free to use arrays or any generic classes (JCF collections) for storing the elements as you see fit (using JCF collections will require some reading ahead and thus is not expected, although if you want to use them then you are welcome to).

Section I - Incorporating Exception Handling for Flights (18 + 12 marks)

In this section you are required to modify the methods (open(), close() …) of Flight and RegionalFlight to throw exceptions when invalid calls are made (see below). The exception thrown must indicate the cause of error, allowing the caller to respond appropriately and recover if possible.

```
public double open(… ) throws Exception
{
   ...
   if (...)
       throw new Exception(...);
    else {
       // process as normal
             ...
    }
}

public void close (...) throws Exception, ...
{
}

public void reserveSeats(...) throws Exception, ...
{
}

public void delay(...)throws Exception, ...
{
}
```

Note that you are welcome to define your own exception type(s) for this requirement and even throw different exception types for different problems (eg. FlightCancelledException, BookingLimitException, etc) if you so wish.

18 marks are allocated to throwing the exceptions appropriately and 12 marks are allocated to catching them in your ManageFlights (application) class.

Section II - Passenger and subclasses (30 marks)

Passenger class

A unique **passenger ID** and **name** is recorded for each passenger, as well as the flight number of the flight they are booked onto. All passengers have a set luggage allowance, although this may be flexible for some types of passenger. In this part you are required to write the **abstract** class named **Passenger** with:

(i) Instance variables **passenger-ID, flight-number,** and **name.**

(ii) Constructor taking as arguments the value for **passenger-ID, flight-number** and **name.**

(iii) An abstract method **double checkInBaggage(double weight)** to take as argument a double value for the weight of the passenger's baggage and return a double value indicating the additional cost to the passenger (if any).

(iv) An appropriate accessor for the **passenger-ID** (other accessors are optional).

(v) An appropriate **print()** method which displays the instance variables to the screen.

Subclass EconomyPassenger

Economy class passengers have a standard luggage allowance of 15kg and are permitted to check in up to an additional 5kg of baggage over and above the standard 15kg allowance at a cost of $20 per kg.

In this part you are required to write the class **EconomyPassenger** extending **Passenger** with:

(i) No new instance variables are required.

(ii) Constructor taking as arguments values for **passenger-ID, flight-number** and **name**.

(iii) Implementation for the abstract method **double checkInBaggage(double weight)** declared in the superclass Passenger, which takes a double value (the total weight of the baggage being checked in) and returns a double (the excess baggage charge, if any).

If the baggage weight exceeds the maximum for an EconomyPassenger then the method should return a result of -1 indicating that the baggage check-in has failed

(iv) Override the print() method so that the passenger type (Economy) is also printed.

Subclass BusinessPassenger

All business passengers have a standard luggage allowance of 20kg and are permitted to check in as much excess luggage as they wish at the same cost of $20 per kg.

Business passengers also get a complimentary in-flight meal and they can choose from chicken, fish or vegetarian options.

In this part you are required to write the class **BusinessPassenger** extending **Passenger** with:

(i) Instance variable for the preferred meal type.

(ii) Constructor taking as arguments values for **passenger-ID, flight-number, name** and **meal type**.

(iii) Implementation for the abstract method **double checkInBaggage(double weight)** declared in the superclass Passenger, which takes a double value (the total weight of the baggage being checked in) and returns a double (the excess baggage charge, if any).

(iv) Override the print() method so that the passenger type (Business) is also printed along with the requested meal type.

Section III - Managing the Flights/Passengers (55 Marks)

In this part you are required to write a class named **ManageFlights** following the steps outlined below allowing Passenger and Flight objects to be created, stored in arrays (or JCF collections if you so wish), manipulated, written and read back from files. All the Passenger (and subclass objects) should be stored in a file named passengers.txt and all the Flight (and subclass) objects should be stored in a file named flights.txt. Whenever a flight is closed the passenger manifest for that flight must be written to a separate flight

Overall Requirements

The users should be allowed to perform the following operations. You may write separate method for each of this activity. You are expected to write a menu-driven program. It should incorporate necessary input validation and exception handling allowing recovery whenever possible.

Notes:

"**Unchanged**" features do not need any additional work, "**updated**" features need to be changed to incorporate exception handling as described previously and "**new"** features need to be added to the ManageFlights class.

You may also use the sample solution for assignment 2B as the basis for your program in assignment 3 if you wish.

1. Schedule a new flight (Flight or RegionalFlight) – **new (10 marks)**

    - should allow the user to add a Flight or RegionalFlight to the schedule. Input validation must ensure that flight number is unique and that departure/flight times are valid (ie. arrival time does not go beyond midnight)

2. Display All Flights – **unchanged**

    - should display the details for all flights (including status).

3. Reserve Seats for a flight – **updated**

    - any exceptions that are thrown from the Flight classes must be caught and reported

4. Open a Flight – **updated**

    - should display a message stating that boarding of the flight may commence if successful

    - any exceptions that are thrown from the Flight classes must be caught and reported

5. Close a Flight – **updated**

    - should display a message stating that the flight has now departed if successful

    - any exceptions that are thrown from the Flight classes must be caught and reported.

6. Delay a Flight – **updated**

    - should report the new departure time for the flight if successful

    - any exceptions that are thrown from the Flight classes must be caught and reported

7. Check In Passenger – **new (10 marks)**

    - should create a new passenger record for the passenger checking in for their flight.

    - should determine the excess baggage charge (if any) for the passenger.

    - if the specified flight number is not found, their flight is not currently boarding or they have exceeded the baggage limit (which will only happen for EconomyPassengers) then a suitable error message should be displayed.

    - you may assume that if the specified flight number is valid and the flight is currently boarding then the passenger is meant to be on that flight (there is no need to check to see if the number of checked in passengers does not exceed the number of reserved seats on the flight).

8. Display details for all passengers that have checked in on a specified flight – **new (5 marks)**

    - display the passenger ID and name of all passengers that have checked in on a specified flight (a suitable error message should be displayed if a flight with the specified id is not found).

9. Transfer Regional Flight Passengers – **new (10 marks)**

    - should open the second leg of the regional flight, transfer all checked-in passengers from the first leg of the regional flight to the second leg and close the second leg of the flight

    - this may involve some minor changes to the RegionalFlight class so that the second leg of the RegionalFlight can be accessed (this has been done in the sample solution).

    - this can only be done for RegionalFlight objects only – if the specified flight number is for a “normal” Flight (not a RegionalFlight) then a suitable error message should be displayed to the screen.

10. File Handling – **new (20 marks)**

- Incorporate file handling into the FlightManagement class so that details for all Flight and Passenger objects in the system are written out to text files when the program ends (user selects the quit option from the menu) and read back into the program from the text files so that the same set of Flight and Passenger objects can be recreated when the program is restarted.

- The details of all flight objects (both Flight and RegionalFlight) should be written out together to the one file "flights.txt", likewise the details of all passenger objects (both EconomyPassenger and BusinessPassenger) should also be written out together to the one file "passengers.txt".

- If no flights.txt file can be opened then the program should still run without any flight or passenger data having been loaded (meaning flights will need to be created and bookings/passenger check-ins recorded by the user during the normal course of running the program).

- The format of the text files is up to individual students – however your program must be able to read in the data from the files it has written to and will need to differentiate between the two types of objects in both the flight and passenger files.

- Serialization of objects for the purpose of writing to and reading from files is not permitted.

Section IV - Bonus marks for printing of flight manifests (20 marks)

Up to 20 bonus marks will be awarded for submissions in which the flight manifest (list of passenger ID's and names) to a file with the name "Manifest-<flight number>.txt" (eg. "Manifest-QJ101.txt") when a flight is closed for departure.

For RegionalFlights when passengers who have boarded the first leg will are transferred to the second leg you also need to print the manifest for the second leg..

The manifest should list the flight number, departure point, destination, departure time and arrival time, as well as the list of passenger ID's and names.

The manifest should be printed in a neat, formatted manner.

Coding style (5 marks)

Assessable points (things we are looking for):

A. Levels of 3 or 4 spaces used to indent (not tabs)

B. Indentation levels consistent throughout program

C. A new level of indentation added for each new class/method/control structure used

D. Going back to the previous level of indentation at the end of a class/method/control structure (before the closing brace for blocks)

E. Lines of code not exceeding 80 characters in length - lines which will exceed this limit are split into two or more segments where required
(just try to stick somewhere near this limit wherever possible)

F. Expressions are well spaced out

G. Source spaced out into logically related segments

H. Commenting – internal code comments describing non-trivial code segments and a comment with your name and student number at the top of each file. Javadoc comments are not expected or required.

Marks will be allocated to the following:

1. Correctness
Carry out proper testing (both White-box and Black-box)

2. Proper handling of text files
Check that all the objects are correctly saved to a text file and restored

3. Incorporating exception handling
Catch and handle exceptions where possible

4. Promoting code reuse through inheritance and polymorphism
Write polymorphic code where appropriate

5. Avoid knock-on effect by encapsulating all the classes
Make all the instance variables private and provide the necessary accessors and mutators

7. User Friendly Features
Print appropriate prompts and messages

8. Documentation &Adherence to Coding Guidelines
Write appropriate comments (do not state the obvious) to explain your code. Adhere to the Coding Guidelines.

Submission instructions

You must **zip** up the **source code files for the Flight, RegionalFlight, Passenger, EconomyPassenger and BusinessPassenger and ManageFlights classes, as well as the source code for your menu-driven application class (including any "helper" classes you may be using)** and submit the resulting zip file into weblearn by **11:59pm Sunday May 29 (end of week 12)**.

If you fail to submit a plain zip file (ie. a rar file or some other compression format) or if you fail to include the source code for one or more classes then the marking of your submission may be delayed and/or any parts of the assignment for which you have not submitted source code may not be assessed.

We will be running demonstrations for assignment 3 during week 13 as a means of providing quick feedback to students – assignments which have not been submitted by the deadline stated above can still be demonstrated and then submitted after the demonstration is complete – such submissions **will be penalised 10% of the mark awarded for each day the submission is late**.

The arrangements for these demonstration sessions and the late submission period will be discussed in further detail closer to the end of the semester when the details for the sessions have been finalized.

This submission will be marked out of a score of 120 (with 20 additional bonus marks available) and contributes 12% (with an additional bonus of 2% available) towards your final result.