CSCI 1101 – 2017
Laboratory Report 4

Your task is to complete the assigned work using JGrasp or an IDE of your choosing. You may use your own computer, or one of the lab computers provided.

Your submission should be a **ZIP** file containing your source code files. You should submit your **ZIP file** on Brightspace:
http://dal.brightspace.com

Submission Deadlines (firm):
Monday Labs: due Wednesday by 12:00pm (noon)
Friday Labs: due Sunday by 12:00pm (noon)

NB:

- Try to submit this report *during* the lab so that your TA can check it for you before you submit!
- Attendance is mandatory in all labs, and will form part (10%) of your overall lab grade
- Acknowledge any help that you obtained from friends, TAs, the Learning Centre, etc., by adding comments to your code where appropriate. Obtaining help is fine, *so long as you acknowledge it!*
- Any students who cannot log on to the lab computers should speak to the Help Desk to set up their account.
- Textbooks, class handouts, and any other materials are welcomed and encouraged in all labs!
- Food and drink are not permitted in the computer labs
- Late labs are ***not*** accepted! It is known that computer errors, power outages, and network lag are 105% more likely to occur between 11:55-11:59am, in the moment they can do the most damage. Account for this, and give yourself the chance to make a timely submission!

Header Comments

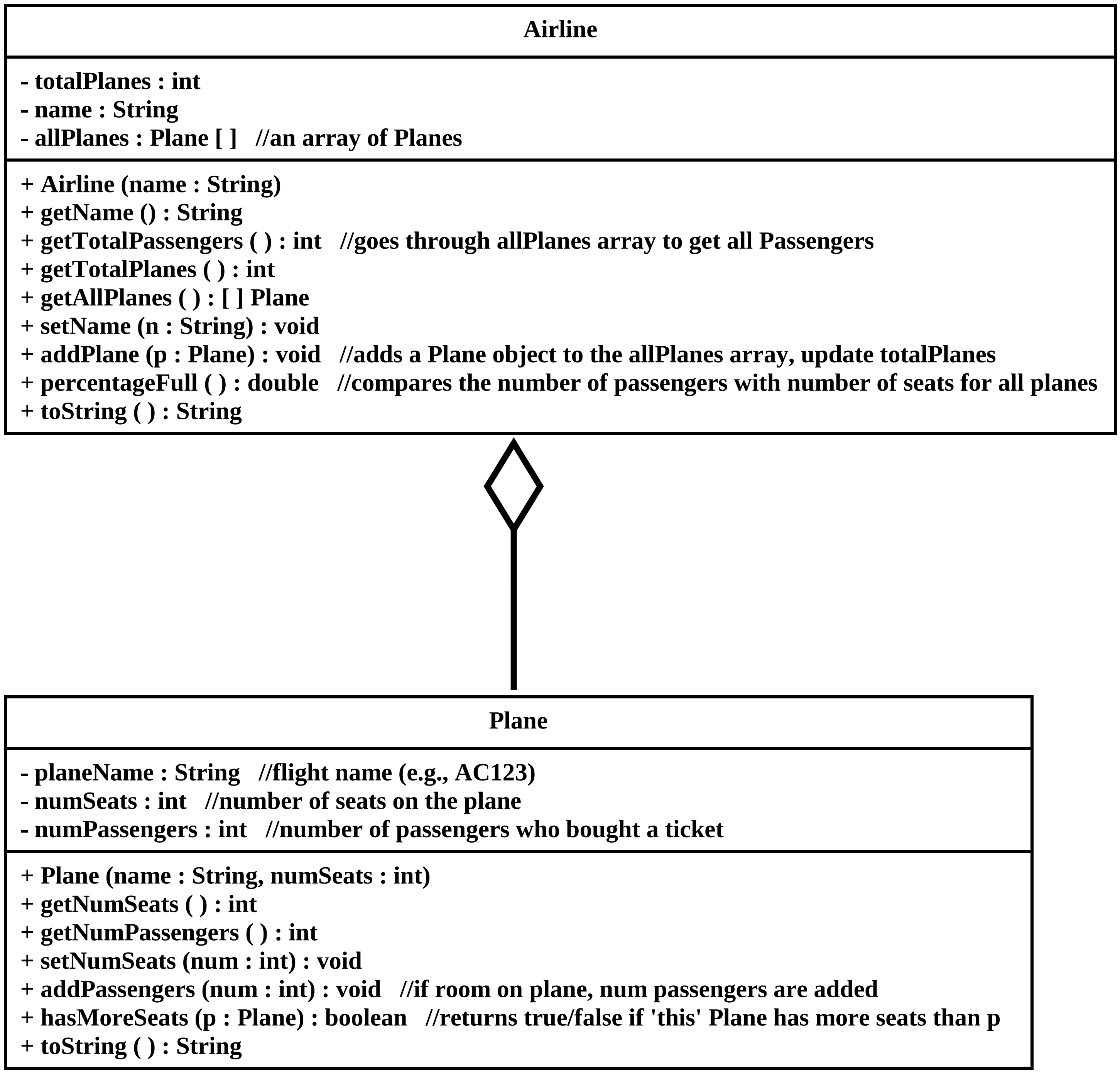
Your code should now include header comments for **all** of your class (.java) files. The comment should include the lab/assignment number, the course (CSCI 1101), the name of your program and a short description of the entire class, the date, your name and Banner ID, and a declaration that matches the first page of this document (e.g., whether you received help). See the example below for what a header comment should include:

```
/*Lab1, Question 1 CSCI 1101
   Student.java holds information about a student at Dalhousie in CSCI1101 and
   their grades
   June 29, 2015
   John Smith B00112345
   This is entirely my own work. */

public class Student {
//rest of Code
```

If applicable, your demo class should then also have a similar header:

```
/*Lab1, Question 1  - demo class CSCI 1101
   StudentDemo.java is a demo program for the Student class. It creates student
   objects, and compares different students.
   June 29, 2015
   John Smith B00112345
   I received help with creating Student objects from my TA but the rest is my
   own work. */

public class StudentDemo {
//rest of Code
```

Exercise 1
Implement the following UML diagram.



```
Airline
- totalPlanes : int
- name : String
- allPlanes : Plane [ ]   //an array of Planes
+ Airline (name : String)
+ getName () : String
+ getTotalPassengers ( ) : int   //goes through allPlanes array to get all Passengers
+ getTotalPlanes ( ) : int
+ getAllPlanes ( ) : [ ] Plane
+ setName (n : String) : void
+ addPlane (p : Plane) : void   //adds a Plane object to the allPlanes array, update totalPlanes
+ percentageFull ( ) : double   //compares the number of passengers with number of seats for all planes
+ toString ( ) : String
```

```
Plane
- planeName : String   //flight name (e.g., AC123)
- numSeats : int   //number of seats on the plane
- numPassengers : int   //number of passengers who bought a ticket
+ Plane (name : String, numSeats : int)
+ getNumSeats ( ) : int
+ getNumPassengers ( ) : int
+ setNumSeats (num : int) : void
+ addPassengers (num : int) : void   //if room on plane, num passengers are added
+ hasMoreSeats (p : Plane) : boolean   //returns true/false if 'this' Plane has more seats than p
+ toString ( ) : String
```

Notes on the classes:

Airline class

- The constructor in the Airline class sets name, initializes the totalPlanes to 0, and initializes the array to hold 10 planes.
- The Airline toString method returns the name of the Airline, total number of planes, and the total number of passengers that have flown on all their planes.
- The addPlane method adds a Plane object to the array of Planes that the Airline has
- percentageFull returns numberOfAllPassengers / numberOfAllSeats in the Planes in the airline to see how full all the planes are.

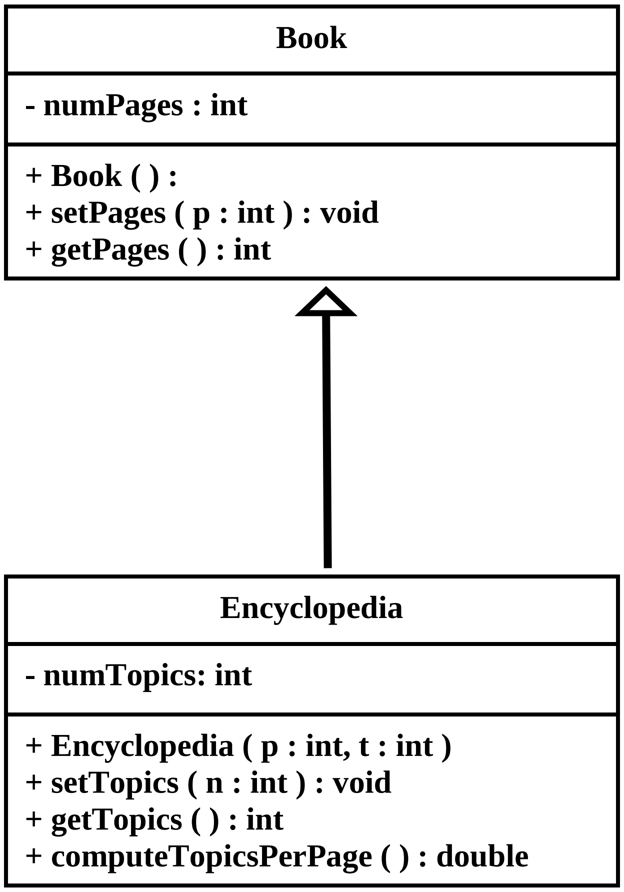
Plane class

- The constructor sets the name of the plane (flight name), the total number of seats on the plane and sets numPassengers to 0
- addPassenger method increases the numPassengers by the number **`num`** (do appropriate checks in this method)
- hasMoreSeats (Plane p) compares two Plane objects (this Plane and Plane p) to see which one has more seats on the plane (ie., see which one is bigger).

Properly test your classes by performing the following operations in a demo class, at minimum.

Create an Airline class with at least 3 Plane objects. Add passengers to all the Planes. Add the planes to your airline. Print out the totalNumber of planes that the airline has. Print out the total number of passengers on all the planes (use the method getTotalPassengers in the Airline class). Add some more passengers to your second plane. Print out the percentageFull (totalPassengers from all planes / totalSeats from all planes) on all the Planes. Print out the plane object that has the most seats. Call the toString method in the Airline Class.

Exercise 2
Implement the following UML diagram.


Book
- numPages : int
+ Book ( ) :
+ setPages (p : int) : void
+ getPages ( ) : int
Encyclopedia
- numTopics: int
+ Encyclopedia (p : int, t : int)
+ setTopics (n : int) : void
+ getTopics ( ) : int
+ computeTopicsPerPage ( ) : double

The constructor in the Encyclopedia (this is a paper version of Wikipedia; you can read about them on Wikipedia) class sets the number of pages and number of topics. The method computeTopicsPerPage returns the average number of topics per page.

Test your classes appropriately with at least two different instances.