CPS209 Assignment 1:

Flight Reservation Simulator

Due Date: Wed., Mar. 24 11:59pm

You are to write a Flight Reservation Simulator program. This programming assignment will increase your knowledge of array lists, objects and classes, inheritance, and interfaces. **You must do this assignment alone – no groups. Do not attempt to find source code on the web for this assignment. It will not help you and you risk extremely serious consequences. Your program will be checked for plagiarism.** Begin designing and programming early! This assignment is worth 10 percent of your mark. **If there is some part of the assignment you do not understand, please contact Prof. McInerney or Prof. Doliskani (via email or ask during lectures) and we will clarify the issue.**

NOTE: some parts of the assignment are using java concepts we have not yet studied (e.g. interface Comparable and interface Comparator, the use of the *super* keyword). These topics will be covered in detail in the two weeks after the break. You may want to wait until these topics are covered before writing the code that uses them. I suggest you start by implementing the commands in FlightReservationSystem.java in this order: MYRES, SEATS, RES, CANCEL, RESFCL and then the SORT commands. See grading scheme below as a guide. Look at the command LIST and follow through the method calls to see how it works.

Program Functionality Requirements:

Please look at the video included with this assignment. Below is a description of the classes for your assignment and list of methods and variables. **NOTE: I have posted skeleton code for you of all the necessary classes. Some of these classes are complete, most are not. You are to fill in code for the methods according to the instructions below and according to the comments in the skeleton code for each class. This makes the assignment much easier to mark for the TAs and ensures you are adhering to proper OO design. To get full marks on this assignment, there is also a part that asks you to design and implement an additional class. We** strongly suggest you build your program in pieces. Get one piece of functionality compiled and working before working on the next piece. Read the marking scheme below and use it as a guide. In the lectures we will answer questions about the assignment – please speak up!

This assignment is a simple version of a Flight Reservation System that models flights departing from Pearson airport during a single day. Look inside FlightManager.java and you will see that we have preprogrammed a list of 10 or so flights. Please do not change these as they will be used for marking. Please look closely at the skeleton code and comments within. For this first assignment, you should always strive to use the methods outlined in the skeleton code rather than creating your own. In other words, we are controlling the overall design in this assignment. Here is the list of classes you are to modify:

1. **Flight:** class ***Flight*** contains several instance variables (see the skeleton code) to model an airline flight departing Toronto and flying to several cities. Each flight has a string identifier ***flightNum*** (for example "AC210" for Air Canada flight 210). See the skeleton code and fill in the methods based on the comments.

2. **Aircraft:** class **Aircraft** models an aircraft type (e.g. Boeing 747) and contains information about the model name, the maximum number of economy seats and the maximum number of first class seats (most often this is 0). **Most of the code for this class is already written for you. You are to make class Aircraft implement the Comparable interface – see the skeleton code for comments.**

3. **Reservation: c**lass **Reservation** models a simple electronic reservation containing flight information, and whether the reservation is for a first-class seat. This class has been written for you.

4. **FlightManager:** this class contains the bulk of the logic for the system. It maintains an array list of Flight objects. **S**ome of the methods have been written for you. Fill in the code for the other methods based on the comments. The constructor method has been written for you and 7 or 8 flights have been crated for you and added to the flights list. Also, several aircraft type have been created and added to the airplanes array list. There are other instance fields that have been created for you as well.

5. **FlightReservationSystem:** This class has the **main() method and is the user interaction part. Some skeleton code has been provided for you with some comments.** In a ***while loop***, a scanner reads a line of input from the user. The input lines contain words (Strings). Some input lines contain a single word that represents a command (an action). Some lines contain a command word and a parameter string. See the class **FlightReservationSystem** for a list of all the commands. Fill in the code based on the comments. You are strongly urged to implement a command (i.e. write the minimal necessary code in the various classes to get a command working) and test it before moving on to the next command. Use the "list" command as a guide as well as the "seats" command.

6. **LongHaulFlight:** a subclass of class **Flight.** It models a long flight to a far-away destination. Long haul flights almost always have different classes of seats (first class, business, economy etc.). In this implementation, it adds variables to keep track of the number of first-class passengers as well as the seat type (first class or economy). It **inherits** all variables and methods from class Flight. It **overrides** inherited methods reserveSeat() and cancelSeat() and has them call new methods reserveSeat(String seatType) and cancelSeat(String seatType) respectively. See the comments in the code to see what needs to be filled in.

7. Design your own class **Passenger** that contains a name, passport number, and a seat number field. Add methods that you deem appropriate including an equals method to compare two Passenger objects (Override the **boolean equals(Object other)** method inherited from superclass **Object**) based on name and passport number. Add functionality to class **Flight** so that a flight keeps a list of its passengers and such that a passenger is assigned a random seat number (use the class **Aircraft** capacity to determine a valid seat number range for a Flight). Class Flight should ensure that there are no duplicate passengers when reserving a seat. Add appropriate methods to class **FlightManager. Add new commands** to class **FlightReservationSystem** (i.e. <u>do not alter the existing commands!!!</u>). Add the

commands **RESPSNGR** (reserve a flight for a passenger with name and passport number), **CNCLPSNGR** (cancel flight reservation), and **PSNGRS** (print list of passengers for a given flight). You may need to add a couple of fields to class **Reservation** as well. NOTE: only implement this functionality for class Flight – no need to consider class **LongHaulFlight** and first-class passengers.

8. Make sure you comment your code. I strongly urge you to use JavaDoc.

9. ***BONUS***: Replace the ***errorMsg*** field in class **FlightManager** with customized exceptions. Instead of methods setting errorMsg and returning true/false, **throw** these customized exceptions in class **FlightManager** (example ***FlightFullException***) In class **FlightReservationSystem**, ***catch*** these exceptions and print an appropriate message to the user. One should always handle (i.e. catch) exceptions as close to the user as possible in an interactive system. Gather all your exception handling in one place a system.

Grading (Out of 10 marks)

Highest Grade Possible: 11

Basic: commands CRAFT, MYRES, RES (no long-haul flight), CANCEL (no long haul), SEATS and code of classes written according to proper OO design, correct use of inheritance	6 marks
Intermediate: commands RESFCL, CANCEL (incorporating long haul flight cancel)	2 marks
3 SORT commands (see code)	1 mark
Class Passenger functionality	1 mark
Customized exceptions as specified above in BONUS	1 mark
Penalty for insufficient comments or bad program structure or bad use of inheritance and interfaces	Up to -1 marks

Submitting Your Assignment: READ CAREFULLY!!

- Use D2L to submit your assignment.
- Inside each of your ".java" files have your name and student id as comments at the top of the file. Make sure your files can be compiled as is without any further modification!!
- Include a README.txt file that describes what aspects of your program works and what does not work (or partially works). If your program does not compile, state this!! You may still get 1 or 2 (out of 10) part marks. The TA will use this file as a guide and make fewer marking mistakes.

- Place all your “.java” file(s) and README.txt file into an archive (zip or rar archiving format only).
- Open your archive and make sure your all your files are there.
- **Do not include ".class" files!!!!**
- **Do not use any Package keyword in your java files!!!**
- Once everything is ready, submit it.