**Flight Reservation Simulator Part 2**

You are to extend the Flight Reservation Simulator program from previous project. This programming project will increase your knowledge of Java Collections, file IO and exceptions, objects and classes, inheritance, and interfaces.

### Program Functionality Requirements:

### Please look at the video included with this project. Below is a description of the new functionality for your project and some new methods and modifications to the existing functionality as well as modifications to the user interface.

### FlightResevervationSystem: There are now only 5 commands: LIST, RES, CANCEL, SEATS, PASMAN, MYRES. The commands take the following arguments:

### LIST: as before - lists flight information for all flights

### RES flight name passport seat: reserve a flight for passenger with name, passport and specified seat. For example: res UA267 McInerney DD1234 7B. A first class seat is designated by a row, one of ABCD and a ‘+’ character (e.g. 4A+)

### CANCEL flight name passport: cancel a reservation on flight for passenger with name and passport

### SEATS flight: prints out the seats on this flight (see video). If seat is occupied, print “XX “ instead of the seat string.

### PASMAN flight: prints the passenger manifest for this flight (i.e information about all passengers – see video)

### MYRES: as before in A1 – prints all reservations (see the video)

### Reservation: There are several changes and additions to the Registry class:

### A reservation object contains fields (add any others you think you need):

### String flightNum;

### String flightInfo;

### boolean firstClass;

### String passengerName;

### String passengerPassport;

### String seat;

### Class Reservation overrides the public boolean equals(Object other)method and compares Reservation objects for equality based on flightNum, passengerName, and passengerPassport.

### 

### Aircraft: class Aircraft has a new field called seatLayout: String[][] seatLayout. This is a 2D array of strings. Add methods to the class to specify the rows and columns for this aircraft. If the capacity is, for example 44, then make the number of rows 4 and the number of columns 11. Note that when this 2D array is printed (see video) the array columns are actually labelled as rows – this models the idea of a row on a real airplane. Always use a seat capacity divisible by 4 for simplicity.

### Passenger:

### class Passenger should have fields:

### private String name;

### private String passport;

### private String seat;

### private String seatType;

### Override the equals method and a passenger is equal to another passenger if they have the same name and passport.

### Flight:

### In addition to fields from A1, class Flight should also have:

### protected ArrayList<Passenger> manifest;

### protected TreeMap<String, Passenger> seatMap;

### manifest is an array list of Passenger objects. The map seatMap maps a seat string (e.g. “7A” or “2B+”) to a Passenger object. When a passenger is added to the flight, this seat map should be updated.

### Add a method FlightType getFlightType() to class Flight. It returns the type of flight (the default is MEDIUMHAUL). I suggest you use an enum of flight types like so:

### *public static enum FlightType {SHORTHAUL, MEDIUMHAUL, LONGHAUL};*

### Override this method in class LongHaulFlight such that it returns *FlightType.LONGHAUL.* This will allow you to distinguish between a regular flight object and a long haul flight.

### Class Passenger only needs one reserve method and one cancel method. Information about whether the user wants a first class seat is indicated in the seat string (e.g. “3B+”) by the “+”

### public void cancelSeat(Passenger p)

### public void reserveSeat(Passenger p, String seat)

### Add a method public void printPassengerManifest() that prints all the passengers on the flight

### Add a method public void printSeats() that makes use of the seatLayout[][] in the aircraft object for this flight. See the video for an example of how to print the seats. If a seat is occupied, print “XX “ instead of the seat string (Hint: make use of the seatMap!).

### Create exception classes like DuplicatePassengerException, PassengerNotInManifestException, SeatOccupiedException. Throw these exceptions in the class Flight methods appropriately. Change the methods in class Flight to return void instead of Boolean and delete the errorMsg variable. Hint: add the exception classes in the Flight.java file outside of class Flight. Do not make them public. Catch all exceptions in class FlightReservationSystem

### LongHaulFlight: This class keeps track of the number of first class passengers. It overrides methods (and makes use of methods via super.) in superclass Flight.

### FlightManager:

### Alter existing methods (and add new methods) to support the new/modified commands in FlightReservationSystem.

### In the constructor method, remove the hardcoded Flight objects and instead load the information for all flights from a file called flights.txt (flights.txt has been posted with this project). Here is an example line from this file:

### United\_Airlines Dallas 1400 42

### The first string on a line is the name of the airline (first word separated from second with an underscore). Read this airline string in and convert it to an airline string with two separate words (e.g. “United Airlines”). The next string is the destination city. After that is the departure time string. The final number is an integer indicating the required passenger capacity of the aircraft. Find an aircraft that is just big enough. Create flight objects from this file and add them to an arraylist. Catch any I/O exceptions in class FlightReservationSystem

### Change the array list of flights to a map (e.g. TreeMap) that maps the flightNum string to a Flight object. Update all methods to use this map instead of the array list.

### BONUS:

### Add command preboard flight which creates a priority queue of the passengers based on their seat row. Add command queue flight which prints the queue. Then add command board flight startrow endrow which removes passengers from the queue whose seat row is in the range. Add a field to the Passenger class indicated a passenger has boarded. When the boarding is done, change the flight status to INFLIGHT.

### Create a set (use Java TreeSet or HashSet) of crew members in FlightManager (i.e. create several CrewMember objects of various titles). Create a Person class and a CrewMember class. Make class Passenger and class CrewMember subclasses of Person. A CrewMember has a title (e.g. pilot, navigator, attendant). Each plane needs a pilot, some number of flight attendants, and a navigator. Remove crewmembers from the set and add them to a flight (alter class Flight so that it has a set of CrewMembers). Before adding crewmembers to a flight, FlightManager should ensure the same crewmember is not added twice (Hint: make use of the properties of a Java set). Check to ensure you are not adding the same crewmember to different flights (i.e. there should be no intersection between flight crew sets). Create some commands in class FlightReservationSystem to support this functionality. Include a README file with your submission detailing the new commands.