Plane Two Dimensional Array Implementation

In this project you will develop a plane management menu by implementing classes which handle the creation, display, and control of the Passenger information behind the scenes. Be sure to read and follow the specifications carefully.

RELEVANT SKELETON CLASSES

- Plane2DArray
- PlaneOutput
- GeneratePlane

NOTES

- To start the assignment, create a package called plane.[first three letters last three letters] i.e. plane.dombar. All of your project files will be located within this package. Your package will ONLY contain the skeleton classes provided (NO MAIN)

- Create classes within the package with the same name as the skeleton code then copy and paste the code from Github into them. Alter the package for each class accordingly. i.e. each class will have `package plane.xxxxxx` at the top of it.

- If you would like for the code to be graded, ONLY send the package as a zip. It will be returned to you with comments giving suggestions and critique of the project. The project will not be graded if any of the methods yield run-time/ compile-time errors or if any portions of the assignment disregard stated requirements. I am happy to address questions about the project during tutoring hours.

- All methods within the skeleton code must be completed. Feel free to add any helper variables and methods needed to assist public methods / the classes functionality. All extra methods / variables added that aren't included in the skeleton code must be private. Methods should be sub-divided in an efficient and readable manner. Do not have methods that do everything within a class. Any method included in the skeleton code but not mentioned in the specifications is assumed to be self explanatory.

- Included with each Plane implementation will be a data structure called `seatList`. This has to be the main data structure used to store passengers and cannot be replaced by another.

- Referencing the code in the library will be vital to completing the assignment. This should be done in github. If you are looking at the code in your IDE you will

be looking at decompiled class files which will NOT contain the proper java doc comments and have odd method structure. Code posted in the Github will retain all the proper java doc comments and be in a form that is intended for viewing.

- Read the entire specifications before proceeding with coding, sections can be done out of order if you would like. User prompts mentioned within the specifications are just suggestions and do not need to exactly match. Inputs will be coming from the outside world so ALL possibilities of what the user can input needs to be accounted for i.e. numbers out of range, strings, string with whitespace, etc.

PART 1 : Implementing the GeneratePlane Class

This class handles satisfying the constructor of your Plane2DArray class and ensures the rows and seats in each row are within range.

METHOD NOTES

- `createPlane()` Upon calling the method users will be prompted to enter the name of the plane followed by the total number of rows and finally the total number of seats in each row. The plane name has no edge cases, rows should be an int greater than 0, and seats should be an int between 1 and 26 (inclusive). Incorrect inputs for row and seats will be prompted accordingly and continually shown until a correct number is input.

  This method takes in an int representing which plane implementation is to be returned. As of right now you will only have completed one, but that will change in future projects. A simple if-else will suffice such that if the parameter is 1 a new instance of Plane2DArray is returned with its constructor satisfied with the information you have collected, otherwise null.

PART 2 : Implementing the PlaneOutput Class

This class handles the plane menu and getting inputs from the user in order to make changes to the plane. Seat numbers and names of passengers provided by the user need to be case-sensitive such that seats are in the form “1A”, “2B”, etc. and names match current passenger names exactly.

METHOD NOTES

- `planeMenu()` This method will print the plane menu and get inputs from user corresponding to options in the menu. An example of the plane menu is shown below. Invalid inputs for menu options should be prompted upon them being entered. Once a menu option is completed / invalid menu option is entered, the plane menu will be printed again until q is entered. The name of the plane plus the word menu will be shown at the top of the menu.

```
My Plane Menu
     a : Add a passenger to a specific seat
     n : Add a passenger to the next available seat
     r : remove a passenger from the plane
     s : swap two passengers seat
     p : print plane map
     q : quit menu
>
```

- `menuAddPassengerAt()` The method initially asks the user to enter a seat they want to add a passenger to. If the seat is incorrectly formatted, a prompt stating so will immediately follow and the method will exit. If the seat is properly formatted, The user will then be asked to enter the passengers name followed by whether or not the were successfully added.

- `menuAddPassengerNextAvailable()` The method asks to enter the passengers names and then prompts whether the passenger was successfully added or the plane is full.

- `menuRemovePassenger()` The method asks to enter the passengers names and then prompts whether the passenger was successfully removed or that they do not exist

- `menuSwapPassengers()` The method asks to enter the first passengers name and then the second (two separate prompts). It then prompts that the two passengers were successfully swapped or that the passengers were unable to be located.

PART 3 : Implementing the Plane2DArray Class

This class handles the control of the plane and its corresponding data structure. This class may not contain ANY print statements, as this will be handled by the other two classes. This need to be designed as a stand alone class with method parameters not being assumed to be already validated, i.e. methods don't only work properly based on the order they are called within the PlaneOutput class and can be called in any order.

METHOD NOTES

- `addPassengerAt()` If the seatNumber parameter is provided a valid seat a new Passenger instance will be added to the array with the constructor satisfied with an int array containing the index of the row and seat and the name provided. For example "1A" or "4C" passed in will be used used to add a new passenger to `seatList[0][0]` or `seatList[3][2]` with their `passengerSeat` initialized to {0,0} or {3,2}. If the seat is already occupied a new passenger cannot be added there. Improperly formatted seats or occupied seats will return false and true otherwise

- `AddPassengerNextAvailable()` This will add a passenger to the first available seat. Assuming seats 1A, 2C and 4D are occupied, the passenger will be added to seat 1B. Successful adds return true while unsuccessful adds return false. Construction of the new passenger is the same as addPassengerAt()

- `removePassenger()` If the specified passenger exists they are removed from the plane and the method returns true, otherwise false.

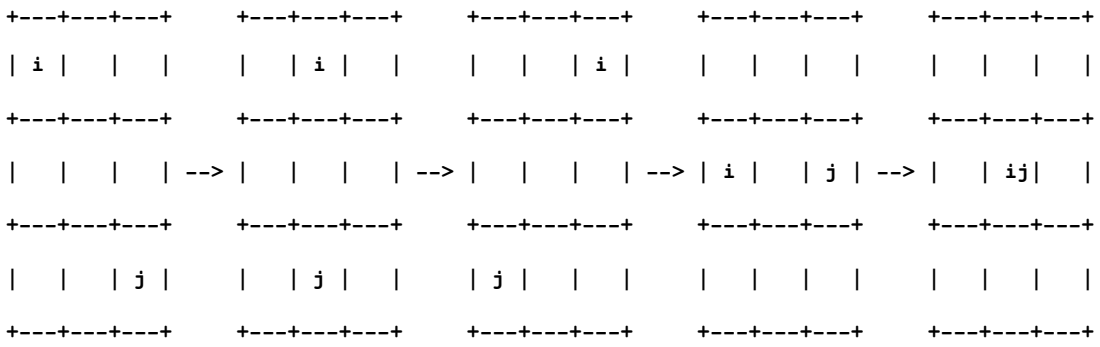
- `swapPassengers()` If one or both of the passengers don't exist the method returns false, otherwise false.

- `generatePlaneMap()` Below is an example of what the plane map should look like with empty seats denoted by an * and occupied ones by the passengers name.

```
Row 1 | 1A Joe, 1B Bobo, 1C *,
Row 2 | 2A *, 2B *, 2C John Smith,
Row 3 | 3A *, 3B John Jones, 3C *,
```

- `validateRowAndSeat()` This method will be provided a string and if it is in the form of a properly formatted seat number and valid, an int array containing its corresponding row and seat indexes will be returned, otherwise null.

- `locatePassenger()` This method traverses the array in order to locate a Passenger whose name matches the one provided. The method will simultaneously search both ends of the until the seats to be checked meet in the middle. A visualization of this is provided below.



The traversal technique doesn't need to be identical to this just such that both the first and last rows are traversed simultaneously. Arrays with all varying row and seats in each row need to be accounted for when developing the traversal algorithm. If the passenger is found at either the i or j positions during the traversal, the Passenger instance is returned. Otherwise null is returned.