Program #3 - Air 161

"Fly the friendly skies of Air 161"

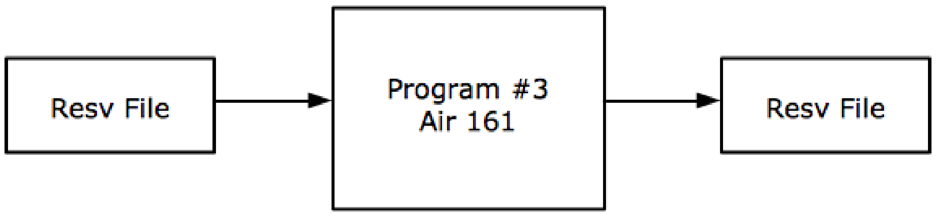
Logistics:

- Due: **Mon May 18, 2015**
- Worth: **6 points** (6% of your class grade)

In Program #3, we'll create a simple console-based reservation system for your favorite airline and mine, Air 161.

1. Introduction

The flow of Program #3 is: 1) read a reservations file, 2) modify the reservations using console text commands, and 3) save the new resv file. Like this:



The Resv file format will be our usual CSV with lines beginning with '#' as comments.

2. Commands

Commands are typed in your console. They include:

- **reserve <seat> <passenger>** - make reservation for a seat (seat must be open!)
- **unreserve <seat>** - remove seat reservation
- **report** - print all seats and their passengers (or open)
- **open** - list all open seats on the plane
- **save** - save the reservations to their file
- **exit** - exit the console program (ask about saving changes!)

I'll paste a simple sample session (SSS) using these commands in the design notes.

3. Design

Your design should include classes like these:

- `Resv` - a reservation: seat/passenger data
- `FlightResvs` - holds all a list of `Resv` objects for a flight
- `ResvConsole` - command console for changing resvs
- `Program3` - `main()` (Hello, Program #3)

For sure, your classes don't have to match these exactly. **You** are the designer. But I don't want to see HUGE `main()` methods or other disorganized solutions.

About your solution… I have these *other* specific design requirements:

- **param to main()** - If a resv file name is passed as a parameter to `main()`, then use it. Otw, ask the user for a resv file at the start of the session.
- **toString()** - Write a `toString()` method for some classes (`Resv`, `FlightResv`) and call it
- **StringBuilder class** - Use a `StringBuilder` to build a complex string somewhere (`toString()` anyone?)
- **Exception** - Create an exception class for at least one error in the console.
- **GUI** - your class design should be done with a GUI application in mind as well. That is, can your console classes be usable to create a GUI (as our WordTracker was in Program #2, eh!).

Go!

4. Grading

Create a `program3` folder in your k: drive.
Place these files in that folder:

- A `README` file describing the state of your program
- All the Java files that comprise your Program #2 solution

All your code must follow our 161 Coding Guidelines. Ugly code will be penalized with a 0-100% reduction in points. A program that doesn't even compile is worth 0 points.

And don't forget... all the special Program #3 design requirements listed above!

Good luck.
yow, bill

PS - Design notes are here: Program #3 Design Notes