Spring 2015 CS151
Instructor: Dr. Kim

Programming Assignment 1
Airline Reservation System

Objectives

- To exercise the analysis, design, and implementation activities of software development.
- To learn requirement analysis using use cases
- To learn noun-verb analysis and CRC technique
- To produce UML class diagrams and UML sequence diagrams; Use a software tool to draw UML diagrams

---

In this assignment, you will implement an airplane seat reservation system. An airplane has two classes of service (first and economy). Each class has a sequence of seat rows. The following figure shows the seat chart of the airplane.

In the above seat chart, the seat rows are numbered as 1,2, etc.

A reservation request has one of two forms. One form is that an individual passenger request consists of the passenger name, a class of service, and a seat preference such as W(indow), C(enter), or A(isle) seats. The reservation program either finds a matching seat (the search may start from the first row of the class.) and reserves it for the passenger, or it fails if no matching seat is available. (For example, if the user's preference is a W(indow) seat and none is available, tell the user no Window seat is available and ask for another seat preference.) The other form is that a group request consists of a list of passenger names and a class of service. (Groups cannot specify a seat preference). The reservation program finds the first row of adjacent seats in a seat row that is sufficient to accommodate the group, or if no such seat row exists, finds the row with the largest number of adjacent seats in any seat row, fills it up with members of the group, and repeats that process (finding the row with the largest number of adjacent seats) until all members of the group have been seated. If there is insufficient space to seat all members of the group, none should be seated.

A cancellation request consists of the name of one or more passengers. The seats that are occupied by passengers in the request are emptied; passengers in the request that are not seated are ignored.

There are two kinds of report: seat availability chart and manifest list. A seat availability chart shows the available seats of each row of each class as shown below. You can guess the rest of the seat availability chart based on the example for the First Class.

Availability List:

```
First
1: B,D
2: C,D
Economy
```

A manifest lists the occupied seats and the passengers seated in them:

```
First
1A: John Williams
1C: Harry Hacker
2A: Sarah Michaels
2B: Jonah Arks
Economy
```

Your program must be called **ReservationSystem**. Use a command line argument to read the name of a file that holds information from a prior program run. If the file does not exist, the program creates one. You may use the exists() method of the File class to check.

The following shows how you would run the program on command prompt.

```
java ReservationSystem flightname
```

where flightname is the name of a file that holds information from a prior program run.

In Eclipse, follow Run-> Run Configurations -> Arguments, enter the command line argument, in our case flightname, in the box named Program Arguments, and run the program.

The user interacts with the program to make a reservation, and the user input is read from standard input (System.in). The user interface for this program is as follows. Prompt the user:

```
Add [P]assenger, Add [G]roup, [C]ancel Reservations, Print Seating [A]vailability Chart, Print [M]anifest, [Q]uit
```

The following describes the function(s) of each option.

- Add [P]assenger
  When the user wants to add a passenger, prompt the user:

  ```
  Name: John Smith
  Service Class: First
  Seat Preference: W
  ```

  Print the seat number such as 1A on which the passenger has been seated, or Request failed.
- Add [G]roup
  When the user wants to add a group, prompt as follows: (bold are user inputs; the group member names should be separated by a comma)

  ```
  Group Name: cssjsu
  Names: Venkathan Subramanian,Karl Schulz
  Service Class: Economy
  ```

  Print the seat numbers on which the passengers have been seated, or Request failed.

- [C]ancel Reservations
  First prompt the user asking if the cancellation is for an [I]ndividual or a [G]roup. When the user wants to cancel an [I]ndividual reservation (as opposed to a [G]roup reservation), prompt as follows:

  ```
  Cancel [I]ndividual or [G]roup? I
  Names: John Smith
  ```

  When the user wants to cancel a group reservation, prompt as follows: (bold are user inputs)

  ```
  Group Name: cssjsu
  ```

  A cancellation of a group reservation will cancel the reservations of all group members. For simplicity, let's assume an individual group member cannot cancel his/her reservation only.
- Print Seating [A]vailability Chart.
  Prompt the user asking the desired service class. For example:

  ```
  Service Class: First
  ```

  or

  ```
  service Class: Economy
  ```

The program displays the seat availability chart of the service class only.

- Print [M]anifest
  Prompt the user asking the desired service class. For example:

  ```
  Service Class: First
  ```

  or

  ```
  service Class: Economy
  ```

  The program displays the manifest of the requested service class only.
- [Q]uit
  When the user quits, save the reservations in the text file whose name was given on the command line when the program starts. As a designer, you should determine what to save in which order to store the reservations already made. This file contains the final reservations only.

Note 1

Here is a simple scenario to understand this requirement. When you run the program for the first time as follows

```
java ReservationSystem CL34
```

the file doesn't exist, and thus the program creates one. Then, suppose reservations X, Y, Z are made, and the program quits. Then, X, Y, Z are stored in the file CL34. In the second time of program execution, the ReservationSystem restores reservations (X, Y, Z) from CL34. Suppose Y is cancelled, and A and B are added. The file contains X, Z, A, and B only.

Note 2

When the program starts, data from the file CL34 will initialize data structure(s) of your choice (e.g. array list) that will hold data throughout the program execution. Adding and cancelling will change the content of the data structure(s), not the file CL34. Only when the Quit option is chosen, the latest snap shot of the data structure(s) will be saved in the file CL34 in a proper format.

Submission

- Hardcopy: **Due: Wednesday, February 18 (MW section) and Thursday, February 19 (TR sections) in class**
  - Use cases
  - A set of final CRC cards
  - UML class diagram (submit a simple version of class diagram, draw it using a software tool)
  - UML sequence diagram (draw it using a software tool)

- Softcopy of your implementation through the course web page: **Due: Friday, February 20 11:59pm**.
  - All source programs you wrote (.java) required to run the application.
    - Name the class with main method as ReservationSystem.
    - Put javadoc comments in the source codes.
    - Submit .java files only.
    - Do NOT use packages.
    - Put all .java files in a directory called **hw1**
    - Put input.txt (described below) in the **hw1** directory as well.
    - In the **hw1** directory, run

      ```
      javadoc -d ./doc *.java
      ```

      That will automatically generate all the javadoc pages and put them in the **doc** directory.
    - Zip **hw1** to **hw1.zip** and submit it.

- A file called input.txt that contains a sample set of inputs with which you have tested your program. I am going to use input.txt to run your program using input redirection as shown below.

```
java ReservationSystem CL34 < input.txt
```

  assuming that the file CL34 doesn't exist when the program executes for the first time. Eclipse does not support a way to redirect input in a stragitforward manner. (I figured out how to redirect output in Eclipse, but not input. Anyone ?) If you want to try input redirection, run the program on command line prompt.

- For testing purposes, you can refer to the provided sample_input.txt and sample_CL34 files. `sample_CL34` uses the first line to denote the available service classes and their corresponding row numbers. For example, flight CL34 provides First and Economy classes. The First class runs from row #1 to row #2. The Economy class runs from row #10 to row #29.

  The next lines in `sample_CL34` denote which seats are taken on the CL34 flight using the following conventions (comma separated):

```
Seat Number, [I]ndividual or [G]roup, Passenger Name (if individual) or Group Name (if group), Passenger Name (if group)
```

  Running **`java ReservationSystem CL34 < sample_input.txt`** the first time, when you don't have an existing CL34 file yet, will generate a CL34 that has the content similar to that of the provided `sample_CL34` file.