Chris Piech
CS 106A

Section #8
March 5, 2018

Section Handout #8: Data Structures and The Internet

Based on a problem by Brandon Burr and Patrick Young

Your task for this week is to write server and client programs that allow the user to plan a round-trip flight route. Note that these are long programs, so we only expect you to get through the server program in section. Feel free to work on the client program on your own. First, here's what a sample run of the client program might look like:

A critical issue in building these programs is designing appropriate data structures to keep track of the information you'll need in order to produce flight plans. You'll need to both have a way of keeping track of information on available flights that you read in from the **`flights.txt`** file, as described below, as well as a means for keeping track of the flight routes that the user is choosing in constructing their flight plan. Consider how both **`ArrayList`**s and **`HashMap`**s might be useful to store the information you care about.

1. FlightPlannerServer

There are two types of requests that the server needs to handle from the client in order for the client to do its job:

Command	Parameters	Response
`getAllCities`	N/A	Returns the list of all cities, as a string, that the user can visit.
`getDestinations`	**`city (String)`**	Returns a list of all cities that the user can travel to *from the given city.*

The flight data come from a file named **`flights.txt`**, which has the following format:

- Each line consists of a pair of cities separated by an arrow indicated by the two-character combination **`->`**, as in

```
New York -> Anchorage
```

- The file may contain blank lines for readability (you should just ignore these).

The entire data file used to produce this sample run appears below.

```
San Jose -> San Francisco
San Jose -> Anchorage

New York -> Anchorage
New York -> San Jose
New York -> San Francisco
New York -> Honolulu

Anchorage -> New York
Anchorage -> San Jose

Honolulu -> New York
Honolulu -> San Francisco

Denver -> San Jose

San Francisco -> New York
San Francisco -> Honolulu
San Francisco -> Denver
```

Your server should:

- Read in the flight information from the file **`flights.txt`** and store it in an appropriate data structure when it begins running.
- Respond to **`getAllCities`** and **`getDestinations`** requests as described above.

Hint: when a server receives a request, it must respond with a String. If you need to respond with a *list* (e.g. an **`ArrayList`**), one method is to return the value of that list's **`toString`** method. For instance:

```
ArrayList<String> myList = ...
String stringToSend = myList.toString();
```

2. FlightPlannerClient

The client should behave as in the screenshot above. Specifically, it should:

- Display the complete list of cities.
- Allow the user to select a city from which to start.
- In a loop, print out all the destinations that the user may reach directly from the current city, and prompt the user to select the next city.
- Once the user has selected a round-trip route (i.e., once the user has selected a flight that returns them to the starting city), exit from the loop and print out the route that was chosen.

As mentioned in the server description, the client will need to contact the server to get a list of all cities, and to get the list of cities that can be traveled to from a particular city. As an example, if you want to make the **`getDestinations`** request to the server, you would write something like the following:

```
String city = ...
try {
   Request request = new Request("getDestinations");

   // specify which city to get possible destinations for
   request.addParam("city", city);

   String result = SimpleClient.makeRequest(HOST, request);
   ArrayList<String> cities = makeListFromString(result);
} catch (IOException e) {
   // an error occurred, do something...
}
```

You'll notice the use of the **`makeListFromString`** method in the above example. When a server sends back a response that is the **`toString`** of an **`ArrayList`**, you need to first convert that string to an **`ArrayList`** in order to use it. We have provided a method **`makeListFromString`** that does just that: it takes a string in the format of an **`ArrayList`** **`toString`** and returns the **`ArrayList`** it represents.