Alisha Adam & Rohit Talreja
CS 106A – Summer 2016

Section Handout #6 – ArrayLists, HashMaps

Based on handouts by Keith Schwarz and Marty Stepp

1. Unique Names. Write a program that asks the user for a list of names (one per line) until the user enters a blank line (i.e., just hits return when asked for a name). At that point the program should print out the list of names entered, where each name is listed only once (i.e., uniquely) no matter how many times the user entered the name in the program. For example, your program should behave as follows:

```
Enter name: Alice
Enter name: Bob
Enter name: Alice
Enter name:
Unique name list contains: Alice Bob
```

2. Remove Even Length. Write a method named **`removeEvenLength`** that takes an `ArrayList` of strings as a parameter and removes all of the strings of even length from the list. For example, if an `ArrayList` variable named `list` contains the values `["hi", "there", "how", "is", "it", "going", "good", "sirs"]`, the call of `removeEvenLength(list);` would change it to store `["there", "how", "going"]`.

3. Mirror. Write a method named **`mirror`** that accepts an `ArrayList` of strings as a parameter and produces a mirrored copy of the list as output. For example, if an `ArrayList` variable named `list` contains the values on the left before your method is called, after a call of `mirror(list);` it should contain the values on the right:

```
["how", "are", "you?"] => ["how", "are", "you?", "you?", "are", "how"]
```

4. Switch Pairs. Write a method named **`switchPairs`** that switches the order of values in an `ArrayList` of strings in a pairwise fashion. Your method should switch the order of the first two values, then switch the order of the next two, switch the order of the next two, and so on. For example, if an `ArrayList` variable named `list` initially stores these values:

```
["four", "score", "and", "seven", "years", "ago"]
```

Your method should switch the first pair, "four" and "score", the second pair, "and" and "seven", and the third pair, "years", "ago". So the call of `switchPairs(list);` would yield this list:

```
["score", "four", "seven", "and", "ago", "years"]
```

If there are an odd number of values, the final element should not be moved (such as "hamlet" below):

```
["to", "be", "or", "not", "to", "be", "hamlet"]
```

5. Flight Planner

Your task for this section is to write a program that reads in a file containing flight destinations from various cities, and then allow the user to plan a round-trip flight route. Here's what a sample run of the program might look like:

The flight data come from a file named `flights.txt`, which consists of several lines of text, each of which lists a single flight. Each flight is represented as the source city, followed by a space, then an arrow written as `->`, another space, and then the name of the destination city. For readability, the file may contain blank lines, which your program should just skip over. For example, the 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 program should:

1. Read in the flight information from the file `flights.txt` and store it in an appropriate data structure.
2. Display the complete list of cities.
3. Allow the user to select a city from which to start.
4. 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.
5. 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.

A critical issue in building this program 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 well as a means for keeping track of the flight routes that the user is choosing in constructing their flight plan. Consider how both `ArrayLists` and `HashMaps` might be useful to keep track of the information you care about.