CS 106A Section 7 Handout (Week 8)

ArrayList

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"]
```

This document and its contents are copyright © Stanford University. All rights reserved.

CS 106A Section 7 Handout (Week 8)

HashMaps

5. **Name Counts.** 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 how many times each name in the list was entered. A sample run of this program is shown below.

```
Enter name: Alice
Enter name: Bob
Enter name: Alice
Enter name: Chelsea
Enter name:
Entry [Chelsea] has count 1
Entry [Alice] has count 2
Entry [Bob] has count 1
```

6. **Contains3.** Write a method called `contains3` that accepts an `ArrayList` of strings as a parameter and returns true if any single string occurs at least 3 times in the list, and false otherwise.

7. **Intersect.** Write a method called `intersect` that accepts two `HashMap`s from strings to integers as parameters and returns a new map containing only the key/value pairs that exist in both of the parameter maps. For a key/value pair to be included in your result, not only do both parameter maps need to contain a mapping for that key, but they need to map that key to the same value. For example, consider the following two maps:

```
{Janet=87, Logan=62, Whitaker=46, Alyssa=100, Stefanie=80, Jeff=88, Kim=52, Sylvia=95}
{Logan=62, Kim=52, Whitaker=52, Jeff=88, Stefanie=80, Brian=60, Lisa=83, Sylvia=87}
```

Calling your method on the preceding maps would return the following new map (the order of the key/value pairs does not matter):

```
{Logan=62, Stefanie=80, Jeff=88, Kim=52}
```

8. **maxOccurrences.** Write a method called `maxOccurrences` that accepts an `ArrayList` of integers as a parameter and returns the number of times the most frequently occurring integer (the "mode") occurs in the list. If the list is empty, return 0.

9. **Reverse.** Write a method called `reverse` that accepts a `HashMap` from integers to strings as a parameter and returns a new `HashMap` of strings to integers that is the original's "reverse". The reverse of a map is defined here to be a new map that uses the values from the original as its keys and the keys from the original as its values. Since a map's values need not be unique but its keys must be, it is acceptable to have any of the original keys as the value in the result. In other words, if the original map has pairs (k1, v) and (k2, v), the new map must contain either the pair (v, k1) or (v, k2).

For example, for the following map:

```
{42=Marty, 81=Sue, 17=Ed, 31=Dave, 56=Ed, 3=Marty, 29=Ed}
```

Your method could return the following new map (the order of the key/value pairs does not matter):

```
{Marty=3, Sue=81, Ed=29, Dave=31}
```

10. **FlyTunes.** *(Thanks to Mehran Sahami for this problem.)* Digital music applications such as iTunes allow you to store a number of music albums, each containing a number of tracks. Your job in this problem is to brainstorm how to design the classes for a new music application called **FlyTunes** that does much the same thing. In FlyTunes, a song in your library must include the song's title, the artist who recorded it, and the song running time in seconds. Each album then consists of an ordering of songs, along with the title of the album. The FlyTunes application itself records information for as many albums as you choose to store in it, and allows users to search by album. If you were writing the code for this program, what **classes** would you use? What **private fields** would each class provide, and what **public methods**? You do not have to write the Java code, but brainstorm an **outline** of what would be in each class. ArrayLists and HashMaps may come in handy here.

This document and its contents are copyright © Stanford University. All rights reserved.