Section Handout #6: HashMaps, ArrayLists, and Classes

Sections 6A and 6B

1. Name Counts

```java
/* File: CountNames.java
* ---------------------
* This program reads a list of names from the user and lists out
* how many times each name appeared in the list.
*/
import acm.program.*;
import java.util.*;
public class CountNames extends ConsoleProgram {
    public void run() {
        HashMap<String, Integer> nameMap = new HashMap<String, Integer>();
        readNames(nameMap);
        printMap(nameMap);
    }
    /*
    * Reads a list of names from the user, storing names and how many
    * times each appeared in the map that is passed in as a parameter.
    */
    private void readNames(Map<String, Integer> map) {
        while (true) {
            String name = readLine("Enter name: ");
            if (name.equals("")) break;
            // See if that name previously appeared in the map. Update
            // count if it did, or create a new count if it didn't.
            int count = 0;
            if (map.containsKey(name)) {
                // get old value of count, and create new value for it
                // that is 1 greater than old value
                count = map.get(name) + 1;
            } else {
                // create a new int with value 1
                count = 1;
            }
            map.put(name, count);
        }
    }
    /*
    * Prints out list of entries (and associated counts) from the map
    * that is passed in as a parameter.
    */
    private void printMap(Map<String, Integer> map) {
        for (String key : map.keySet()) {
            int count = map.get(key);
            println("Entry [" + key + "] has count " + count);
        }
    }
}
```
2. Intersect

```java
private HashMap<String, Integer> intersect(
    HashMap<String, Integer> map1, HashMap<String, Integer> map2) {
    HashMap<String, Integer> result = new HashMap<String, Integer>();
    for (String key : map1.keySet()) {
        int value = map1.get(key);
        if (map2.containsKey(key) && value == map2.get(key)) {
            result.put(key, value);
        }
    }
    return result;
}
```

3. Reverse

```java
private HashMap<String, Integer> reverse(HashMap<Integer, String> map) {
    HashMap<String, Integer> result = new HashMap<String, Integer>();
    for (int key : map.keySet()) {
        String value = map.get(key);
        result.put(value, key);
    }
    return result;
}
```

4. How Unique!

```java
/*
 * File: UniqueNames.java
 * ----------------------
 * This program asks the user for a list of names (one per line)
 * until the user enters a blank line. Then the program prints
 * out the list of names entered, where each name is listed only
 * once (i.e., uniquely)
 */

public class UniqueNames extends ConsoleProgram {
    public void run() {
        ArrayList<String> list = new ArrayList<String>();
        while (true) {
            String name = readLine("Enter name: ");
            if (name.equals("")) break;
            if (!list.contains(name)) {
                list.add(name);
            }
        }
        println("Unique name list contains: ");
        printList(list);
    }
    /* Prints out contents of ArrayList, one element per line */
    private void printList(ArrayList list) {
```
for(int i = 0; i < list.size(); i++) {
    println(list.get(i));
}

5. Remove Even Length.
private void removeEvenLength(ArrayList<String> list) {
    for (int i = list.size() - 1; i >= 0; i--)
        if (list.get(i).length() % 2 == 0)
            list.remove(i);
}

6. Mirror
private void mirror(ArrayList<String> list) {
    for (int i = list.size() - 1; i >= 0; i--)
        list.add(list.get(i));
}

7. Switch Pairs
private void switchPairs(ArrayList<String> list) {
    for (int i = 0; i < list.size() - 1; i += 2)
        String first = list.remove(i);
        list.add(i + 1, first);
}

8. Student
// Student object repr. a Stanford student w/ name, ID, and unit count.
public class Student {

    private String name;   /* The student's name */
    private int ID;        /* The student's ID number */
    private double unitsEarned; /* number of units student has */

    /** Constant: the number of units required for graduation */
    public static final double UNITS_TO_GRADUATE = 180.0;

    // Creates a new student object with given name, ID, and 0 units.
    public Student(String studentName, int studentID) {
        name = studentName;
        ID = studentID;
        unitsEarned = 0;
    }

    // Returns the name of this student.
    public String getName() {
        return name;
    }
}
// Returns the ID number of this student.
public intgetID() {
    return ID;
}

// Returns the number of units earned.
public double getUnits() {
    return unitsEarned;
}

// Increments the earned units by the given number of units.
public void incrementUnits(double additionalUnits) {
    unitsEarned += additionalUnits;
}

// Returns whether or not the student has enough units to graduate.
public boolean hasEnoughUnits() {
    return unitsEarned >= UNITS_TO_GRADUATE;
}

// Creates a string IDing this student, such as "Marty (#223)".
public StringtoString() {
    return name + " (#" + ID + ")";
}

9. TimeSpan

// A TimeSpan object represents duration of time in hours and minutes.
public class TimeSpan {
    private int hours;
    private int minutes;

    // Constructs a time span with the given # of hours and minutes.
    public TimeSpan(int initialHours, int initialMinutes) {
        hours = 0;
        minutes = 0;
        add(initialHours, initialMinutes);
    }

    // Adds given hours/minutes to time span, wrapping hours if req’d.
    public void add(int hr, int mn) {
        hours += hr;
        minutes += mn;
        if (minutes >= 60) {
            minutes -= 60;  // convert 60 min --> 1 hour
            hours++;
        }
    }

    // Returns the hours represented by this time span.
    public double getHours() {
        return hours;
    }

    // Returns the minutes represented by this time span.
    public double getMinutes() {
        return minutes;
10. Subclassing GCanvas

/*
 * File: RandomCirclesCanvas.java
 * ------------------------
 * This GCanvas subclass adds the ability to also draw random circles.
 * Each circle has a randomly chosen color, a randomly chosen
 * radius between 5 and 50 pixels, and a randomly chosen
 * position on the canvas, subject to the condition that
 * the entire circle must fit inside the canvas without
 * extending past the edge.
 */

import acm.graphics.*;
import acm.util.*;

public class RandomCirclesCanvas extends GCanvas {

    private static final double MIN_RADIUS = 5;
    private static final double MAX_RADIUS = 50;

    public void drawRandomCircle() {
        double r = rgen.nextDouble(MIN_RADIUS, MAX_RADIUS);
        double x = rgen.nextDouble(0, getWidth() - 2 * r);
        double y = rgen.nextDouble(0, getHeight() - 2 * r);
        GOval circle = new GOval(x, y, 2 * r, 2 * r);
        circle.setFilled(true);
        circle.setColor(rgen.nextColor());
        add(circle); // adds it to ourself!
    }

    /* Private instance variable */
    private RandomGenerator rgen = RandomGenerator.getInstance();
}