

Name: _____

CMPU 161 — Spring 2009

Exam #2

This exam should have four problems (not counting problem #0) and contain six pages.
Closed book and notes.

Problem 0: [1 point]

On a scale of 0–5, where 5 is highest, I think I deserve a _____ for overall class participation.
(Feel free to justify your answer.)

Problem 1: [20 points]

1a) In class, you learned that Binary Search is faster than Linear Search. Explain, briefly, why that's the case.

1b) If Binary Search is so great, why would anyone ever use Linear Search?

Problem 2: [24 points]

Refer to this monstrously mysterious method when answering the questions below.

```
public static boolean mystery(String[] a, String[] b)
{
    String x = a[0];
    for(String z : a)
    {
        if (z.compareTo(x) > 0)
            x = z;
    }
    String y = b[0];
    for(String z : b)
    {
        if (z.compareTo(y) < 0)
            y = z;
    }
    return (y.compareTo(x) >= 0);
}
```

2a) What does this method return if passed arrays containing ["alice", "bob"] and ["catherine", "doug"] as inputs?

2b) What does this method return if passed arrays containing ["alice", "catherine"] and ["bob", "doug"] as inputs?

2c) Describe, in *English*, what the method does in general. (To get in the right frame of mind, think about what you'd write as a *comment* for this method.)

Problem 3: [25 points] The next page shows the outline of the Combination class from the ComboGuesser assignment. In class we discussed ways to verify that the constructor was really filling new Combination instances with random values. One way to get a handle on this would be to look at the *average* of the values in the combination. Below, define a new Combination method called `getAverageValue` that takes no arguments, and returns an integer that is the average of the values in the combination. (We'll lose precision since it returns the average as an integer, but that's ok.)

```
> Combination c = new Combination(100, 1000);
> c.getAverageValue()
501    (int)
> c = new Combination(100, 200);
> c.getAverageValue()
98    (int)
```

```

import java.util.Random;

/**
 * The Combination class models the combination for an electronic
 * lock. A combination consists of a sequence of non-negative
 * integers (stored in an array), each ranging from [0..max]. The
 * constructor creates a Combination instance whose values are
 * randomly chosen, but the mutator method setValue can be used to
 * change any value in the combination sequence.
 *
 * @author Brad Richards
 * @version 1.0
 */

public class Combination {
    private int[] key;        // The sequence of integers
    private int max;         // Largest possible value
    private Random randGen;  // Our random number generator

    /**
     * The constructor takes a pair of arguments that describe
     * the length and maximum value of the numbers in the
     * combination sequence, and builds a Combination instance
     * populated with random values.
     *
     * @param length The number of values in the combination
     * @param max    The largest possible value
     */

    public Combination(int length, int max) {
        randGen = new Random(); // Create random # generator
        this.max = max;         // Record max for later use
        key = new int[length]; // Create an array

        // Work through the array, putting a random value in each
        // position. Note that the range used in nextInt is
        // max+1 since we allow max to appear in the sequence.

        for(int i=0; i<key.length; i++)
            key[i] = randGen.nextInt(max+1);
    }

    // Other methods omitted
}

```

Problem 4: [30 points] The MessageMunger code from class is shown on the next page (most of it, anyway). Write a new MessageMunger method, `getVocabulary`, that returns an ArrayList containing the Strings that appear as *values* in the munger's wordMap. (Hint: Don't forget about the `keySet` method in the HashMap class.) For example:

```
> MessageMunger m = new MessageMunger();
> m.getVocabulary()
[]
> m.addMapping("exam", "test");
> m.addMapping("fun", "entertaining");
> m.getVocabulary()
[entertaining, test]
```

I'll do the hard part to get you started:

```
public ArrayList<String> getVocabulary() {
```

```

import java.util.HashMap;

/**
 * The MessageMunger class stores mappings from one word to another,
 * and can apply them to rewrite strings provided by the user.
 *
 * @author Brad Richards & class
 * @version 1.0
 */
public class MessageMunger
{
    // wordMap maps input words to their replacements
    private HashMap<String, String> wordMap;

    /**
     * The constructor builds a new HashMap that maps
     * Strings to Strings.
     */
    public MessageMunger() {
        wordMap = new HashMap<String, String>();
    }

    /**
     * The user can add new mappings if they wish
     */
    public void addMapping(String original, String updated) {
        wordMap.put(original, updated);
    }

    /**
     * The munge method breaks apart the input string into "words"
     * and returns a similar string in which input words appearing
     * as keys in the HashMap are replaced with their
     * corresponding values.
     */
    public String munge(String msg) {
        String result = "";
        // Break the input string into words at the spaces
        String[] words = msg.split(" ");
        // Look up each word.  If we find a replacement, use
        // it, otherwise stick with the original.
        for(String word : words) {
            if (wordMap.containsKey(word))
                result += wordMap.get(word)+" ";
            else
                result += word+" ";
        }
        return result;
    }
    // Remaining methods omitted
}

```