

Solutions to Practice Final Exam

Portions of this handout by Eric Roberts

Problem 1: Short answer (15 points)

Answer for 1a:

When an object is passed into method, a *reference* to the object (i.e., its address in memory) is what is actually being passed to the method (this is called *pass-by-reference*). Any changes made to the object are made through that reference (address), so the place in memory where that original object resides is modified. As a result, when the method completes, any changes made to the object persist since the changes were made to the same place in memory as where the original object resided.

When an `int` is passed as a parameter to a method, the method actually receives a *copy* of the `int`'s value (this is called *pass-by-value*). Any changes made to that parameter in the method are just changing this copy of the value, so the original `int` variable that was passed in as a parameter is not modified.

Answer to 1b:

As written, the program leaves the array in the following state:

list				
50	10	10	10	10

If you had wanted `mystery` to “rotate” the array elements, you would need to run the loop in the opposite order to ensure that no elements are overwritten, like this:

```
private void mystery(int[] array) {
    int tmp = array[array.length - 1];
    for (int i = array.length - 1; i > 0; i--) {
        array[i] = array[i - 1];
    }
    array[0] = tmp;
}
```

Problem 2: Graphics and Interactivity (35 points)

```
/*
 * File: EtchASketch.java
 * -----
 * This program solves the Etch-a-Sketch problem from the practice final
 */

import acm.graphics.*;
import acm.program.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
```

```

public class EtchASketch extends GraphicsProgram {

    /** Cross size */
    private static final double CROSS_SIZE = 10;

    /** Step size */
    private static final double STEP_SIZE = 20;

    /** Initialize the application */
    public void init() {
        add(new JButton("North"), SOUTH);
        add(new JButton("South"), SOUTH);
        add(new JButton("East"), SOUTH);
        add(new JButton("West"), SOUTH);
        x = getWidth() / 2;
        y = getHeight() / 2;
        double delta = CROSS_SIZE / 2;
        cross = new GCompound();
        cross.add(new GLine(-delta, -delta, delta, delta));
        cross.add(new GLine(-delta, delta, delta, -delta));
        add(cross, x, y);
        addActionListeners();
    }

    /** Called when an action event is detected */
    public void actionPerformed(ActionEvent e) {
        String cmd = e.getActionCommand();
        if (cmd.equals("North")) {
            moveCross(0, -STEP_SIZE);
        } else if (cmd.equals("South")) {
            moveCross(0, STEP_SIZE);
        } else if (cmd.equals("East")) {
            moveCross(STEP_SIZE, 0);
        } else if (cmd.equals("West")) {
            moveCross(-STEP_SIZE, 0);
        }
    }

    /**
     * Moves the cross and adds a red line to the canvas connecting its
     * old and new positions.
     */
    private void moveCross(double dx, double dy) {
        GLine line = new GLine(x, y, x + dx, y + dy);
        line.setColor(Color.RED);
        add(line);
        x += dx;
        y += dy;
        cross.move(dx, dy);
    }

    /** Private instance variables */
    private GCompound cross;
    private double x, y;
}

```

Problem 3: Strings (35 points)

```

/*
 * File: CheckWordLadder.java
 * -----
 * Solution for checking a word ladder from the practice final exam.
 */

import acm.program.*;

/** Checks to see whether a word ladder is legal */
public class CheckWordLadder extends ConsoleProgram {

    public void run() {
        println("Program to check a word ladder.");
        println("Enter a sequence of words ending with a blank line.");
        String previous = null;
        String current = null;
        while (true) {
            while (true) {
                current = readLine();
                if (current.equals("")) break;
                if (isLegalLadderPair(previous, current)) break;
                println("That word is not legal. Try again.");
            }
            if (current.equals("")) break;
            previous = current;
        }

        /** Method: isLegalLadderPair(previous, current)
         * Checks to see if it is legal to link the two words in a
         * word ladder.
         */
        private boolean isLegalLadderPair(String previous, String current) {
            if (!lexicon.isEnglishWord(current)) return false;
            if (previous == null) return true;
            if (previous.length() != current.length()) return false;
            return countCharacterDifferences(previous, current) == 1;
        }

        /** Method: CountCharacterDifferences(s1, s2)
         * Counts the number of character positions in s1 and s2 that contain
         * different characters.
         */
        private int countCharacterDifferences(String s1, String s2) {
            int count = 0;
            for (int i = 0; i < s1.length(); i++) {
                if (s1.charAt(i) != s2.charAt(i)) {
                    count++;
                }
            }
            return count;
        }

        /** Private instance variables */
        private Lexicon lexicon = new Lexicon("english.dat");
    }
}

```

Problem 4: Arrays (20 points)

```
/** Method: checkUpperLeftCorner
 *
 * This method checks the upper left corner of a Sudoku array
 * to see if it correctly contains one copy of each digit
 * between 1 and 9. If so, the method returns true. If it
 * contains values that are duplicated or out of range, the
 * method returns false.
 */
private boolean checkUpperLeftCorner(int[][] matrix) {
    boolean[] alreadyUsed = new boolean[10];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            int digit = matrix[i][j];
            if (digit < 1 || digit > 9) return false;
            if (alreadyUsed[digit]) return false;
            alreadyUsed[digit] = true;
        }
    }
    return true;
}
```

Problem 5: Data structure design (25 points)

```
/*
 * File: StringQueue.java
 * -----
 * This program implements the MinimalStringQueue interface using
 * an ArrayList for internal storage.
 */

import java.util.*;

/** Implements an ArrayList queue */
public class StringQueue implements MinimalStringQueue {

    /** Creates a new empty queue. */
    public StringQueue() {
        waitingLine = new ArrayList<String>();
    }

    /** Adds a new String to the end of the queue */
    public void add(String str) {
        waitingLine.add(str);
    }

    /** Removes and returns the first String (or null if queue is empty) */
    public String poll() {
        if (waitingLine.isEmpty()) return null;
        String first = waitingLine.get(0);
        waitingLine.remove(0);
        return first;
    }

    /** Returns the number of entries in the queue. */
    public int size() {
        return waitingLine.size();
    }

    /** Private instance variables */
    private ArrayList<String> waitingLine;
}
```

Problem 6: Java programming (30 points)

```

/** Method: isGooglewhack(word1, word2)
 *
 * Returns true if word1 and word2 appear on exactly one web page,
 * as reported by googleSearch.
 */
private boolean isGooglewhack(String word1, String word2) {
    String[] pages1 = googleSearch(word1);
    String[] pages2 = googleSearch(word2);
    int matches = 0;
    for (int i = 0; i < pages1.length; i++) {
        if (findStringInArray(pages1[i], pages2) != -1) {
            matches++;
            if (matches > 1) return false;
        }
    }
    return (matches == 1);
}

/** Method: findStringInArray(key, array)
 *
 * Returns the index of the first occurrence of key in the array.
 * If key does not appear in the array, findStringInArray
 * returns -1.
 */
private int findStringInArray(String key, String[] array) {
    for (int i = 0; i < array.length; i++) {
        if (key.equals(array[i])) return i;
    }
    return -1;
}

```

Problem 7: Using data structures (20 points)

```
/** Method: commonKeyValuePairs(map1, map2)
 *
 * Returns a count of the number of common key/value pairs in the
 * two HashMaps that are passed in.
 */
public int commonKeyValuePairs(HashMap<String,String> map1,
                               HashMap<String,String> map2) {

    int count = 0;

    // Get iterator over map1
    Iterator<String> it = map1.keySet().iterator();

    while (it.hasNext()) {
        // Get key from map1
        String key = it.next();

        // See if that keys exists in map2
        if (map2.containsKey(key)) {
            // Look up values associated with key in both maps
            String map1Value = map1.get(key);
            String map2Value = map2.get(key);

            // See if values are equal
            if (map2Value.equals(map1Value)) {
                count++;
            }
        }
    }

    return count;
}
```