Eric Roberts
CS 106A

Handout #58A
May 25, 2012

Answers to Practice Final #1

Review session: **Wednesday, June 6, 7:30–9:30 P.M. (Hewlett 200)**

Scheduled finals: **Friday, June 8, 8:30–11:30 A.M. (Dinkelspiel Auditorium)**
Monday, June 11, 8:30–11:30 A.M. (Hewlett 200)

Problem 1—Short answer (10 points)

1a) 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;
}
```

1b)

```
private void printAscendingByCount(Map<String,Integer> map) {
   Map< int , ArrayList<String> > reverseMap
        = new TreeMap< int , ArrayList<String>>();
   for (String key : map.keySet()) {
      int count = map.get(key);
      ArrayList<String> keyList = reverseMap.get(count);
      keyList.add(key);
      reverseMap.put(count, keyList);
   }
   for (Integer count : reverseMap.keySet()) {
      ArrayList<String> valueList = reverseMap.get(count);
      for (int i = 0 ; i < valueList.length ; i++) {
         key = valueList.get(i);
         println(key + " = " + count);
      }
   }
}
```

1. The primitive type **int** in the initial declaration must instead be the wrapper type **Integer**.

2. Until you encounter the first value, the entry for a particular count will be **`null`**. To account for this case, it is necessary to check for the **`null`** value before adding an element, as in the following code:

```
ArrayList<String> keyList = reverseMap.get(count);
if (keyList == null) keyList = new ArrayList<String>();
```

3. The reference to **`valueList.length`** should be **`valueList.size()`**.
4. The variable **`str`** is undeclared in the second loop.

Problem 2—Using the `acm.graphics` library (15 points)

```
/**
 * Creates a GCompound object that represents a pie chart composed
 * of the data in the array.  The reference point of the GCompound
 * is the center of the circle.
 *
 * @param r The radius of the pie chart
 * @param data An array specifying the values to be plotted
 * @return A GCompound containing the pie chart
 */
   private GCompound createPieChart(double r, double[] data) {
      GCompound gc = new GCompound();
      double total = sumArray(data);
      double start = 0;
      for (int i = 0; i < data.length; i++) {
         double sweep = 360.0 * data[i] / total;
         GArc arc = new GArc(-r, -r, 2 * r, 2 * r, start, sweep);
         arc.setFilled(true);
         arc.setFillColor(WEDGE_COLORS[i % WEDGE_COLORS.length]);
         gc.add(arc);
         start += sweep;
      }
      return gc;
   }

/**
 * Returns the sum of the array.
 *
 * @param array An array of double values
 * @return The sum of those values
 */
   private double sumArray(double[] array) {
      double total = 0;
      for (int i = 0; i < array.length; i++) {
         total += array[i];
      }
      return total;
   }
```

Problem 3—Strings (15 points)

```
/*
 * 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 = "-";
      String current = "-";
      while (!current.equals("")) {
         while (true) {
            current = readLine();
            if (current.equals("") || isLegalPair(previous, current)) break;
            println("That word is not legal.  Try again.");
         }
         previous = current;
      }
   }

/*
 * Checks to see if it is legal to link the two words in a
 * word ladder.  To be legal, a word must:
 *
 *    1. Be in the dictionary as a legal word
 *    2. Match the previous word, except for the first word in the chain.
 *       This code sets previous to "-" if there is no previous word.
 */
   private boolean isLegalPair(String previous, String current) {
      if (!english.contains(current)) return false;
      if (previous.equals("-")) return true;
      if (previous.length() != current.length()) return false;
      return countCharacterDifferences(previous, current) == 1;
   }

/*
 * 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 english = new Lexicon("EnglishWords.txt");
}
```

Problem 4—Arrays (10 points)

There are many ways to solve this problem. One of the most straightforward is to keep a tabulating array that counts how many times each element occurs, as follows:

```
/*
 * Checks to see whether the array of dice contains two pairs.
 * This method operates by keeping a count of the number of dice
 * on which each of the six possible outcomes of a die exist.
 */
   private boolean checkTwoPairCategory(int[] dice) {
      int[] counts = new int[6 + 1];
      for (int i = 0; i < dice.length; i++) {
         counts[dice[i]]++;
      }
      int pairCount = 0;
      for (int i = 1; i <= 6; i++) {
         if (counts[i] == 2) pairCount++;
      }
      return pairCount == 2;
   }
```

Problem 5—Building graphical user interfaces (20 points)

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

import acm.graphics.*;
import acm.program.*;
import java.awt.*;
import java.awt.event.*;
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;

/** Constructor */
   public EtchASketch() {
      add(new JButton("North"), SOUTH);
      add(new JButton("South"), SOUTH);
      add(new JButton("East"), SOUTH);
      add(new JButton("West"), SOUTH);
      add(new JButton("South"), SOUTH);
      addActionListeners();
   }

/** Runs the program */
   public void run() {
      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, getWidth() / 2, getHeight() / 2);
   }

/** 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);
      }  else if (cmd.equals("Clear")) {
         removeAll();
         add(cross, getWidth() / 2, getHeight() / 2);
      }
   }
```

☞

```
/**
 * 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 6—Data structures (20 points)

```
/*
 * File: FacebookRefund.java
 * -------------------------
 * This program calculates the refund due to a customer because of
 * slow execution of trades by Morgan Stanley.
 */

import acm.program.*;
import acm.util.*;
import java.io.*;
import java.text.*;
import java.util.*;

public class FacebookRefund extends ConsoleProgram {

   public void run() {
      Map<Date,Double> sharePrice = readSharePriceData("FBSharePrice.txt");
      Date ordered = stringToDate(readLine("Sell ordered at: "));
      Date executed = stringToDate(readLine("Sell executed at: "));
      int nShares = readInt("Number of shares: ");
      if (sharePrice.containsKey(ordered) && sharePrice.containsKey(executed))
         double amountPaid = nShares * sharePrice.get(executed);
         double amountDue = nShares * sharePrice.get(ordered);
         double balance = amountDue - amountPaid;
         if (balance > 0) {
            println("Refund due of $" + balance);
         } else {
            println("No refund due");
         }
      } else {
         println("No shares sold at that time");
      }
   }
```

☞

```
/**
 * Reads and returns a Map containing the date-to-price conversion data.
 */
   private Map<Date,Double> readSharePriceData(String filename) {
      try {
         Map<Date,Double> map = new TreeMap<Date,Double>();
         BufferedReader rd = new BufferedReader(new FileReader(filename));
         while (true) {
            String line = rd.readLine();
            if (line == null) break;
            int equalSign = line.indexOf('=');
            if (equalSign == -1) throw new ErrorException("Missing =");
            Date date = stringToDate(line.substring(0, equalSign).trim());
            double price = Double.parseDouble(line.substring(equalSign + 1).tr
            map.put(date, price);
         }
         return map;
      } catch (IOException ex) {
         throw new ErrorException(ex);
      }
   }

/**
 * Parses a string into a java.util.Date structure.
 */
   private Date stringToDate(String str) {
      try {
         return new SimpleDateFormat("MM/dd/yyyy hh:mma").parse(str);
      } catch (ParseException ex) {
         throw new ErrorException(ex);
      }
   }
}
```

Problem 7—Essay: Extensions to the assignments (10 points)

This extension will require the following changes to the Adventure program:

- In **`AdvObject`**, add a new field to record the current location; implement methods **`setLocation`** and **`getLocation`** to set and retrieve the location of the object. Use 0 as a location to indicate that it is being carried.
- In **`Adventure`**, update the take, drop, and initial placement logic to call **`setLocation`** on the object.
- Also in **`Adventure`**, expand the code that interprets commands so that it recognizes **`ZAP`** along with the other built-in commands such as **`LOOK`**, **`TAKE`**, and **`DROP`**. The exact form of the code will vary depending on how you implemented the command logic, but presumably the code will look something like this:

```
if (verb.equalsIgnoreCase("ZAP")) {
   game.executeZapCommand(obj);
}
```

- Also in **`Adventure`**, implement the **`executeZapCommand`** command so that it has the desired functionality. The code will look something like this:

```
public void executeZapCommand(AdvObject obj) {
   int location = obj.getLocation();
   ArrayList<AdvObject> source;
   if (location == 0) {
      source = inventory;
   } else {
      source = rooms[location].getObjectList();
   }
   source.remove(obj);
   location = obj.getInitialLocation();
   rooms[location].getObjectList().add(obj);
   obj.setLocation(location);
}
```

- You should update the list of commands displayed by the **`Help`** command to include the new **`ZAP`** command.