Chris Piech
CS106A

Handout #extra
March 19, 2017

Additional Practice Problem Solutions

1. Read integers in the console from a user until the user enters a blank like. For each print "EVEN" if the integer is even and "ODD" if the integer is odd.

Solution:
```
while(true) {
   String line = readLine("?");
   if(line.equals("")) break;
   int value = Integer.parseInteger(line);
   if(value % 2 == 0) {
      println("EVEN");
   } else {
      println("ODD");
   }
}
```

2. Write a double for loop to print out a rectangle of "0"s.

Solution:
```
while(true) {
   for(int row = 0; row < LINES; row++) {
      String line = "";
      for(int col = 0; col < COLS; col++) {
         line += "0";
      }
      println(line);
   }
}
```

3. Populate an array of size 10 with random booleans.

Solution:
```
boolean[] arr = new boolean[10];
RandomGenerator rg = RandomGenerator.getInstance();
for(int i = 0; i < arr.length; i++) {
   arr[i] = rg.nextBoolean();
}
```

4. Populate a grid with 3 rows and 2 columns where each value of the grid has a value equal to 2.

Solution:
```
int[][] grid = new int[3][2];
for(int r = 0; r < grid.length; r++) {
   for(int c = 0; c < grid[0].length; c++) {
      grid[r][c] = 2;
   }
}
```

5. Populate a grid with 3 rows and 2 columns where each value of the grid has a value equal to its row index plus its col index.

Solution:
```
int[][] grid = new int[3][2];
for(int r = 0; r < grid.length; r++) {
   for(int c = 0; c < grid[0].length; c++) {
      grid[r][c] = r + c;
   }
}
```

6. Write a method

```
   private void printMatrix(double[][] grid)
```

that prints the contents of a grid to the screen (in any reasonable format).

Solution:

```
private void printMatrix(double[][] grid) {
   for(int r = 0; r < grid.length; r++) {
      String rowString = “”;
      for(int c = 0; c < grid[0].length; c++) {
         if(c != 0) rowString += “,  ”;
         rowString += grid[r][c];
      }
      println(rowString);
   }
}
```

7. Open a file named “doubles.txt” with 10 lines, one double per line and store each double in an array.

Solution:

```
double[] values = new double[10];
try {
   BufferedReader rd =
      new BufferedReader(new FileReader("doubles.txt"));
   for(int i = 0; i < 10; i++) {
      values[i] = Double.parseDouble(rd.readLine());
   }
} catch(IOException e) {
   e.printStackTrace();
}
```

8. Write a **`GraphicsProgram`** such that when a mouse is clicked, you add a single 10 x 10-pixel rectangle with upper left corner in the place the mouse was clicked.

Solution:
```
public class Clicker extends GraphicsProgram {
   public void run() {
      addMouseListeners();
   }

   public void mouseClicked(MouseEvent e) {
      int x = e.getX();
      int y = e.getY();
      GRect r = new GRect(10, 10);
      add(r, x, y);
   }
}
```

9. Make a hashmap that associates strings to doubles and put in the following entries which associates universities and their endowment (in billions of dollars):
```
   "Stanford" -> 22.4
   "Berkeley" -> 4.0
   "Harvard" -> 35.0
```
Then write a loop that can iterate over all of the key/value pairs and print them out. Any order / format is fine.

Solution:
```
HashMap<String, Double> map = new HashMap<String, Double>();
map.put("Stanford", 22.4);
map.put("Berkeley", 4.0);
map.put("Harvard", 35.0);
for(String key : map.keySet()) {
   double value = map.get(key);
   println(key + " -> " + value);
}
```

10. Put 10 random Integer pairs (both the key and the value are in the range 0, 100) in a hashmap. Then print out all the values whose keys are even.

Solution:

```
HashMap<Integer, Integer> map = new HashMap<String, Integer>();
for(int i = 0; i < 10; i++) {
     int key = rg.nextInt(0, 100);
     int value = rg.nextInt(0, 100);
     map.put(key, value);
}
for(Integer key : map.keySet()) {
   if(key % 2 == 0) {
     println(map.get(key));
   }
}
```

11. Write a method:

   `String addQuotation(String str)`

Which takes in an input string and add quotation marks (") to the start and end of the string. Return the result.

Solution:

```
String addQuotation(String str) {
   return "\"" + str + "\"";
}
```

11. Use a loop to create a string that contains all of the capital letters in the alphabet, eg: "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

Solution:

```
String letters = "";
for(char ch = 'A'; ch <= 'Z'; ch++) {
   letters += ch;
}
```

12. Write a method:

    String makeCamelCase(String input)

That, returns the input in camelcase. The input is assumed to be all lowercase. Whenever the input has a space, remove the space and make the next letter uppercase. For example:
If the input was "**this is a test**", you would return "**thisIsATest**".

Solution:

```
String makeCamelCase(String input) {
   String[] parts = input.split(" ");
   String str = "";
   str += parts[0];
   for(int i = 0; i < parts.length; i++) {
      String part = parts[i];
      char firstChar = part.charAt(0);
      str += Character.toUpperCase(firstChar);
      str += part.substring(1);
   }
   return str;
}
```

13. Write a ConsoleProgram that has two buttons one which says “yes” and one which says “no”. Each time a user clicks on a button you should printout whether or not the “yes” button has been pressed more, the “no” button has been pressed more, or whether they have been pressed an equal number of times.

Solution:
```
public class Buttons extends ConsoleProgram {
   private int numYes = 0;
   private int numNo = 0;

   public void init(){
      add(new JButton("Yes"), SOUTH);
      add(new JButton("No"), SOUTH);
      addActionListeners();
   }

   public void actionPerformed(ActionEvent e) {
      if(e.getActionCommand().equals("Yes")) {
         numYes++;
      } else {
         numNo++;
      }
      if(numYes > numNo) {
         println("More yes");
      }
      if(numNo > numYes) {
         println("More no");
      }
      if(numNo == numYes) {
         println("Equal")
      }
   }
}
```

14. Write a variable type **`IdCounter`** which has a single public method `getNextId`. `getNextId` should return an integer, and should never return the same integer twice. For example, a user of the **`IdCounter`** class might do something like this:

```
IdCounter idCounter = new IdCounter ();
int idForChris = idCounter.getNextId();  // can return any id for chris
int idForNick = idCounter.getNextId();   // should return a different id
```

Solution:

```
public class IdCounter {
   int nextId = 0;

   public IdCounter() {
      // do nothing
   }

   public int getNextId() {
      nextId++;
      return nextId;
   }
}
```

Long Problem 1: Strings (15 Points)

There are many solutions to this problem. Generally, the outline of the solution is to start off with a method like this:

```
private String longestIsogram(ArrayList<String> allWords) {
      String longest = "";
      for (String word: allWords)
            if (isIsogram(word) && word.length() > longest.length())
                  longest = word;

      return longest;
}
```

Then to implement the **isIsogram** method to check whether or not a string is an isogram. There are many solutions to this problem; here are a four of them:

```
private boolean isIsogram(String word) {
  boolean[] used = new boolean[26];
  for (int i = 0; i < word.length(); i++){
     char ch = word.charAt(i);
     if (used[ch - 'a']) return false;
     used[ch - 'a'] = true;
  }
  return true;
}
```

```
private boolean isIsogram(String word) {
 for (int i = 0; i < word.length(); i++) {
    char ch = word.charAt(i);
    if (word.indexOf(ch, i + 1) != -1)
      return false;
  }
  return true;
}
```

```
private boolean isIsogram(String word) {
 String used = "";
 for (int i = 0; i < word.length(); i++) {
    char ch = word.charAt(i);
    if (used.indexOf(ch) != -1)
      return false;
    used += ch;
  }
  return true;
}
```

```
private boolean isIsogram(String word) {
 for (int i = 0; i < word.length(); i++) {
    char ch = word.charAt(i);
    for (int j = i + 1; j < word.length();
         j++) {
      if (word.charAt(j) == ch) {
        return false;
      }
    }
  }
  return true;
}
```

Problem 2: The Never-ending Birthday Party (25 Points)

```
import acm.program.*;
import acm.util.*;
public class NeverendingBirthdayParty extends ConsoleProgram {
    public void run() {
        RandomGenerator rgen = RandomGenerator.getInstance();
        boolean[] used = new boolean[366];
        int numLeft = 366;
        int numPeople = 0;

        while (numLeft > 0) {
            int birthday = rgen.nextInt(0, 365);
            if (!used[birthday]) {
                 numLeft--;
                 used[birthday] = true;
            }
            ++numPeople;
        }
        println("We needed " + numPeople + " in our group.");
    }
}
```

Problem 3: Arrays (25 points)

```
/** Method: isMagicSquare
 * This method checks an n x n grid to see if it's a magic square.
 */
private boolean isMagicSquare(int[][] matrix, int n) {
  /* A 0 x 0 square is valid, in a weird way. */
   if (n == 0) return true;

  /* If we don't see all numbers in the range 1 to n², we can report
   * failure.                                                      */
   if (!allExpectedNumbersFound(matrix, n)) return false;

  /* Sum up the first row to get its value. */
   int expected = rowSum(matrix, 0, n);

  /* Check that all rows and columns have this value. */
   for (int i = 0; i < n; i++) {
      if (rowSum(matrix, i, n) != expected ||
          colSum(matrix, i, n) != expected)
         return false;
   }
   return true;
}

/** Method: allExpectedNumbersFound
 * This method returns whether all the numbers 1 … n² are present in the
 * given grid.
 */
private boolean allExpectedNumbersFound(int[][] square, int n) {
/* Make an array of n² + 1 booleans to track what numbers are found.  The
 * +1 is because the numbers range from 1 to n² and we have to ensure that
 * there's sufficient space.
 */
   boolean[] used = new boolean[n * n + 1];

   /* Iterate across the grid and ensure that we've seen everything. */
   for (int row = 0; row < n; row++) {
      for (int col = 0; col < n; col++) {
         /* Make sure the number is in range. */
         if (square[row][col] < 1 || square[row][col] > n * n)
            return false;

         /* Make sure it isn't used. */
         if (used[square[row][col]])
            return false;

         /* Mark the square used. */
         used[square[row][col]] = true;
      }
   }
   /* At this point, we know that all numbers are in range and there are
    * no duplicates, so everything is valid.
    */
   return true;
}
```

```java
/** Method: rowSum
 * Returns the sum of the given row of the grid.
 */
private int rowSum(int[][] grid, int row, int n) {
   int sum = 0;
   for (int i = 0; i < n; i++) {
      sum += grid[row][i];
   }
   return sum;
}

/** Method: colSum
 * Returns the sum of the given column of the grid.
 */
private int colSum(int[][] grid, int col, int n) {
   int sum = 0;
   for (int i = 0; i < n; i++) {
      sum += grid[i][col];
   }
   return sum;
}
```