Nick Troccoli
CS 106A

Practice Final
August 12, 2017

Solutions to Practice Final Exam

Based on handouts by Marty Stepp, Mehran Sahami, Eric Roberts and Patrick Young

Problem 1: Java expressions, statements, and methods (20 points)

Answer for 1a:

```
2 3 [0, 0, 17, 0]
3 1 [0, 0, 17, 0]
1 0 [17, 0, 17, 0]
0 1 [17, 0, 17, 0]
```

Answers for 1b:

a) **{four=quatre, one=un, cinq=five, deux=two, three=trois}**
b) **{computer=program, car=drive, board=skate}**
c) **{ebert=siskel, heads=tails, begin=end, boy=girl, first=last}**
d) **{seed=tree, light=tree, tree=violin, cotton=shirt}**

Problem 2: SignMaker (25 points)

```
public class SignMaker extends GraphicsProgram {
      private int labelY;
      private JTextField line;
      private JTextField font;

      public void init() {
            line = new JTextField(30);
            line.addActionListener(this);
            font = new JTextField(15);
            font.setText("Times-Bold-36");
            labelY = 0;
            add(new JLabel("Line: "), SOUTH);
            add(line, SOUTH);
            add(new JLabel(" Font: "), SOUTH);
            add(font, SOUTH);
      }

      public void actionPerformed(ActionEvent e) {
            if (e.getSource() == line) {
                  GLabel label = new GLabel(line.getText());
                  label.setFont(font.getText());
                  labelY += label.getHeight();
                  double x = (getWidth() - label.getWidth()) / 2;
                  add(label, x, labelY);
                  line.setText("");
            }
      }
}
```

Problem 3: The Neverending Birthday Party (25 points)

```
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 4: Magic Squares (35 points)

```
    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 1 to n^2, 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 … n^2 are present in
     * the given grid.
     */
    private boolean allExpectedNumbersFound(int[][] square, int n) {
    /* Make an array of n^2 + 1 booleans to track what numbers are found.
     * The +1 is because the numbers range from 1 to n^2 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;
   }

   /** 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;
   }
```

Problem 5: Favorite Letters (25 points)

```
public class FavoriteLetters extends Program {
      private JTextField letterField;
      private JLabel output;
      private ArrayList<String> letters;

      public void init() {
            letters = new ArrayList<>();

            output = new JLabel("[]");
            add(output, NORTH);

            letterField = new JTextField(10);
            letterField.setActionCommand("Add");
```

```
            letterField.addActionListener(this);
            add(letterField, SOUTH);

            add(new JButton("Add"), SOUTH);
            add(new JButton("Remove"), SOUTH);

            addActionListeners();
      }

      public void actionPerformed(ActionEvent event) {
            String cmd = event.getActionCommand();
            String letter = letterField.getText().toLowerCase();
            if (letter.length() == 1) {
                  if (cmd.equals("Add")) {
                        if (letters.contains(letter)) {
                              letters.add(0, letter);
                        } else {
                              letters.add(letter);
                        }
                  } else if (cmd.equals("Remove")) {
                        while (letters.contains(letter)) {
                              letters.remove(letter);
                        }
                  }
            }

            // update display
            letterField.setText("");
            output.setText(letters.toString());
      }
}
```

Problem 6: SubMaps (25 points)

```
private boolean isSubMap(HashMap<String, String> map1,
                         HashMap<String, String> map2) {
      for (String key : map1.keySet()) {
            if (!map2.containsKey(key) ||
                !map1.get(key).equals(map2.get(key))) {
                  return false;
            }
      }
      return true;
}
```

Problem 7: String Queue (25 points)

```
public class StringQueue {

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

/** 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);
   }

/** Adds a new String to a random index in the queue */
   public void addRandom(String str) {
      RandomGenerator rgen = RandomGenerator.getInstance();
      int randIndex = rgen.nextInt(waitingLine.size());
      waitingLine.add(randIndex, 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();
   }
}
```