SI@UCF Java GUI Test #2 Solution
June 26, 2019

1) (15 pts) We define a string x to be dominated by a string y if the string y contains x as a substring. (Note: this means that a string does dominate itself.) Thus, "sci" is dominated by "computer**sci**ence", but it is not dominated by "engineering". Write a method that takes in an ArrayList of Strings (each unique) and returns another ArrayList of Strings that only contains strings from the input list that are not dominated by any other (different) string in the input list. For example, if the input list was ["ate", "sandwich", "and", "plate", "late", "latte", "tea"] then the method should return the list ["sandwich", "plate", "latte", "tea"] since "ate" and "late" are dominated by "plate" and "and" is dominated by "sandwich".

```
public static ArrayList<String> notDominated(ArrayList<String> list) {

      ArrayList<String> res = new ArrayList<String>();

      for (int i=0; i<list.size(); i++) {

            boolean include = true;

            for (int j=0; j<list.size(); j++) {

                  if (i == j) continue;

                  if (list.get(j).contains(list.get(i))) {
                        include = false;
                        break;
                  }
            }

            if (include) res.add(list.get(i));
      }

      return res;
}
```

Grading: 2 pts create empty list, 2 pts look, 10 pts for mechanism to determine if a string isn't dominated by any other string (lots of ways to do this, so give partial accordingly), 1 pt return

2) (15 pts) Write a method that takes in a one dimensional array nums and a positive integer k, and returns the maximum average of k consecutive values in the array. It's guaranteed that the length of the array is greater than or equal to k. (For example, if nums = [3, 9, 2, 6, 7] and k = 2, then the desired result is 6.5 since the average of the last 2 items is 6.5 and this is more than the average of any other consecutive pair of items. These other averages are 6, 5.5 and 4, respectively.)

```
public static double maxKAve(int[] nums, int k) {

     long sum = 0;
     for (int i=0; i<k; i++)
          sum += nums[i];

     long max = sum;

     for (int i=k; i<nums.length; i++) {
          sum = sum + nums[i] - nums[i-k];
          if (sum > max) max = sum;
     }

     return (double)max/k;
}
```

Here is an alternate, more straight-forward solution:

```
public static double maxKAveAlt(int[] nums, int k) {

     long sum = 0;
     for (int i=0; i<k; i++)
          sum += nums[i];
     long max = sum;

     for (int i=1; i<=nums.length-k; i++) {

          sum = 0;
          for (int j=i; j<i+k; j++)
               sum += nums[j];

          if (sum > max) max = sum;
     }
     return (double)max/k;
}
```

Grading: 2 pts use initial sum as best instead of setting max to 0 or some other fixed value, 5 pts for trying each interval somehow, 5 pts for getting the correct sum/avg of an interval, 2 pts for updating max, 1 pt for returning double (not int).

3) (20 pts) A semi-magic square is a square of integers such that each row and each column add up to the same value. Complete the static method `isSemiMagicSquare` below that takes in a two dimensional integer array (guaranteed to be a square) and returns true if and only if the array represents a semi-magic square. (Hint: Add up the first row to determine what the "magic sum" must be. Then check all of the rest of the rows and columns against this sum.) Feel free to add extra static methods that you call from `isSemiMagicSquare`.

```
public static boolean isSemiMagicSquare(int[][] grid) {

     int n = grid.length;
     int magicsum = 0;
     for (int i=0; i<n; i++)
          magicsum += grid[0][i];

     for (int i=0; i<n; i++) {

          int sum = 0;
          for (int j=0; j<n; j++)
               sum += grid[i][j];

          if (sum != magicsum) return false;

          sum = 0;
          for (int j=0; j<n; j++)
               sum += grid[j][i];

          if (sum != magicsum) return false;
     }

     return true;
}
```

Grading: 4 pts getting a valid magic sum. 7 pts for correctly returning false if any row is wrong, 7 pts for correctly returning false if any col is wrong, 2 pts for returning true otherwise.

Consider building a ThumbDrive class which manages files stored on a thumb drive. To keep the design simple, we'll assume that there are no directories and that each file is in the root directory of the ThumbDrive. Here is the code for a class MyFile:

```
class MyFile {

    private String name;
    private int bytes;

    public MyFile(String n, int b) {
        name = n;
        bytes = b;
    }

    public int getSize() {
        return bytes;
    }
}
```

For this question you will write three methods in the ThumbDrive class: the constructor, canAdd, and add. Here is an outline of the class without the code you will write:

```
public class ThumbDrive {

    public int capacityBytes;
    public int usedBytes;
    public ArrayList<MyFile> files;

    public ThumbDrive(int spaceInBytes) {}

    public boolean canAdd(MyFile f) {}

    public void add(MyFile f) throws SpaceException {}
}
```

4) (10 pts) In this question you will design the constructor for the ThumbDrive class. The constructor takes in an integer, representing the number of bytes the ThumbDrive will be able to store. When we construct the object, no files are added to the ThumbDrive and all bytes are free to be used. Note that the files ArrayList must be initialized to be an empty list and both capacityBytes and UsedBytes should be initialized.

```
public ThumbDrive(int spaceInBytes) {
     capacityBytes = spaceInBytes;
     usedBytes = 0;
     files = new ArrayList<MyFile>();
}
```

Grading: 3 pts capacity, 3 pts used, 4 pts files

5) (5 pts) In this you will write a method that takes in a file and returns true if there is enough extra space left over on the ThumbDrive to add this file to it. Return false otherwise.

```
public boolean canAdd(MyFile f) {

     return f.getSize() <= capacityBytes - usedBytes;

}
```

Grading: 1 pt return, 1 pt getSize, 1 pt <= 2 pts cap-used

6) (10 pts) Below is the code for the SpaceException class. This Exception indicates that there is not enough space on a ThumbDrive to execute the desired operation. The add method, which takes in a MyFile, potentially throws this exception if the user attempts to add a file for which there isn't enough space. Complete the add method so that if there isn't enough room for the MyFile object, the SpaceException is thrown, but if there is enough space, the object is added to the ThumbDrive.

```
public class SpaceException extends Exception {
    public SpaceException(String s) {
        super(s);
    }
}

public void add(MyFile f) throws SpaceException {

     if (!canAdd(f))
          throw new SpaceException("Ahh, ran out of space!!!");

     usedBytes += f.getSize();
     files.add(f);

}
```

Grading: if to check exception 2 pts, throw 3 pts, usedSize update is 3 pts, files update is 2 pts.

7) (20 pts) Complete the class below to draw a box on the screen that will change position whenever you click inside it. You will initialize the box with random coordinates, and make sure that the position is always such that the ENTIRE box is in the frame. In the mouseClicked(MouseEvent e) method, you will check if the mouse position on click is within the bounds of the box. If so, you need to change the position of the box and repaint.

```
public class TestQuestion extends JComponent implements MouseListener {
      final public static int WIDTH = 500;
      final public static int HEIGHT = 500;
      int boxX, boxY, boxSize;
      Random r;

      public static void main(String[] args) {
            JFrame frame = new JFrame("Test!");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(WIDTH, HEIGHT);
            TestQuestion program = new TestQuestion(50);
            program.setFocusable(true);
            program.addMouseListener(program);
            frame.add(program);
            frame.setResizable(false);
            frame.setVisible(true);
      }
      /*** FILL IN CODE HERE to generate initial boxX, boxY so that box is
           Randomly placed on the screen and has dimensions size by size.***/
      public TestQuestion(int size) {

            boxSize = size;
            r = new Random();

            boxX = r.nextInt(WIDTH - boxSize);
            boxY = r.nextInt(HEIGHT - boxSize);


      }
      public void paintComponent(Graphics g) {
            g.drawRect(boxX, boxY, boxSize, boxSize);
      }
      /*** Write this so we move the box only if the click e has occurred
           within the box ***/
      public void mouseClicked(MouseEvent e) {

            if (e.getX() >= boxX && e.getX() <= boxX + boxSize
                        && e.getY() >= boxY && e.getY() <= boxY + boxSize) {
                  boxX = r.nextInt(WIDTH - boxSize);
                  boxY = r.nextInt(HEIGHT - boxSize);

                  repaint();
            }
      }
}
```

Grading: Constructor 8 pts – 1 pt boxSize, 1 pt random, 3 pts boxX, 3 pts boxY, mouse 12 pts – 8 pts if statement, 1 pt boxX, 1 pt boxY, 2 pts repaint

8) (5 pts) What is one item you can order at Red Lobster? **Lobster** **(Give to all)**