COS 161 – Algorithms in Programming

Project 03 – Concentration

Objectives

The objective of this assignment is to become familiar with creating Maps and Sets and utilizing them to store information.

General Instructions for all Assignments

For each assignment you will write a Java program and test it. Start your programs with a comment block such as:

```
/*
   NAME: <your name>
   COS 161, Summer 2020, Prof. <instructor name>
   Project ##
   File Name: CLASS_NAME.java
*/
```

Your programs should be neatly formatted, and follow the indenting, formatting, spacing, and commenting practices taught in class.

Read each assignment carefully to see what is required. If you do not understand something in the assignment, ask a tutor or the instructor. You will get partial credit if you finish only part of an assignment or it is not working correctly. Turn it in and explain what you completed and what issues it has. It is usually better to turn in an imperfect assignment on the due date, rather than falling behind in the course.

Part 1 (15 points) Setup the Cards and Collections

For this project, we are going to replicate the guessing / card game of Concentration. If you are unfamiliar with the game, you can find more info at the following link:
https://en.wikipedia.org/wiki/Concentration_(card_game)

For our version, we will be building a two-player variant using the drawing panel. First, create a Class called MemoryCard. This class must have the following:

- Private fields, String shapeType, Color color, boolean isUncovered with Getters and Setters.
- Public toString() method that returns a formatted display of shapeType, color, and isUncovered

Next, create a Java file called Concentration that will be the client program for this project. Create two static Sets (Hash or Tree) of type MemoryCard that will correspond to the cards that each player has uncovered (call them whatever you would like to). Also, create a static Map (Tree recommended) called gameBoard that will store the cards in play. This should map Integer values onto MemoryCard values. Next, make a static ArrayList of type MemoryCard called cardDeck.

After the Collections are created, make a method called fillBoard() that fills cardDeck with 36 pairs of new MemoryCards (meaning 18 unigue MemoryCards each added twice). For colors, you can choose any 6, for shapes you must have 6 each (one of each color you picked) of the following 3 shapes: Circle, Square, Triangle. That should give you the 18 unique pairs you need. Each should have an initial isUncovered value of false. Make sure one you are done filling the List, you Shuffle it.

Next, fill the gameBoard Map with the Integers 0 – 35 mapped to the MemoryCards stored in cardDeck. Since they have been shuffled this should give you a randomized board full of all the necessary cards.

Part 2 (20 Points) Displaying the Cards

For this part, you must use the DrawingPanel to display the Cards. Create a method called drawBoard() that will draw a 6 by 6 grid of rectangles. This method must use an Iterator to look at every element of gameBoard. Every time this method is called, it should first blank the entire board (see Tic Tac Toe project for an example of how to do this).

For each element it should do the following: Check if isUncovered is true, if so, draw the correct shape inside the rectangle corresponding to that card. If it is false, print the number inside the rectangle instead.

This should give you an initial output of 6 rows and 6 columns of rectangles with 0 – 35 printed inside them. Once you think you have this

right, edit the gameBoard Map temporarily to set all MemeryCard isUncovered values to true. The board should now display 36 shapes of various colors (18 unique pairs) inside the 36 card rectangles.

Part 3 (45 Points) Playing the Game

For this part, you must take in alternating player input allowing each player to guess two cards, then display those cards flipped. If a player correctly guesses two matching cards, those cards should be added to that players corresponding Set and the isUncovered property of the MemoryCards of the match should remain true. If there is no match, the isUncovered values should be set back to false, the game board should be redrawn, and the next player should be prompted for a guess. You can choose whether to reveal both guessed cards at once or have them show up one at a time.

If a player guesses a match successfully, they should be able to guess two more cards, repeated until they incorrectly guess a pair.

The game should end once all cards are revealed. The game should print out the winner based on which player's Set of "captured" cards is larger. You should also print out the winning players set of winning cards.

Extra Credit (10 points) Make it Bigger, add more shapes

Increase the board size to 8 by 8, reduce the number of colors to 4, and increase the number of shapes to 8 (giving you the 32 unique pairs you need). You can do whatever you like for the shapes, but they must be distinct. Some suggestions would be two dots, two squares, two triangles, diamond, pentagon, rectangle (make sure it does not look like a square!), or even something more exotic – be creative!

What to Turn In

Turn in a solution document as a single pdf file. This solution document should contain the following (make sure formatting is clear):

- The toString() method in MemoryCard
- Whatever method(s) you made that check for two matching MemoryCards

- At least two screenshots of console input of game being played.
- At least three screenshots of the game board DrawingPanel

Once you have the solution document ready, submit it as an attachment on Brightspace. Be sure to also attach all the .java files you created or edited in this project. Do not zip or compress the files, submissions with archives of any type will be ignored.