Homework Assignment #3

CS5004 – Object-Oriented Design
Northeastern University – Silicon Valley
Summer 2019

Due 06/02 at 11:00pm PDT

Instructions: You must submit a `.java` file per problem. Note that we use Java 8 or higher. Submit your files on Blackboard.

Grading: Each programming problem is graded as follows

- A submission which does not compile gets 0.
- A submission which compiles but does something completely irrelevant gets 0.
- A submission which works (partially) correctly, gets (up to) %80 of the total credit.
- %20 is reserved for the coding style. Follow the coding style described in the book.

Problem 1 [10pts]. The value of e^x can be approximated by the following sum

$$e^x = 1 + x + x^2/2! + x^3/3! + ... + x^n/n!$$

Write a program that takes as input x and n and computes the above sum. You should not use any library facilities. Use a loop. Input is given by user via console.

Problem 2 [30pts]. The game of Pig is a simple two-player dice game in which the player to reach 100 or more points wins. Players take turn. On each turn, a player rolls a six-sided die:

- If the player rolls a 1, then the player gets no new points and the turn is over.
- If the player rolls 2-6, the they can either
  - ROLL AGAIN or
  - HOLD. At this point, the sum of all rolls is added to the player's score and the turn is over.

Write a program that plays the game of Pig, where one player is the computer and the other player is a human (user of your program). When it's the user's turn, the program must show the current score of both players and the result of the roll. Allow the user to input "r" to roll again and "h" to hold.
The computer should play according to the following strategy: Keep rolling until it has accumulated 20 or more points, then hold.
Your program should monitor the game for a winner. The human player goes first.

Problem 3 [60pts]. A common single-player memory game recommended for young children is played in the following way. You start with a deck of cards that contain identical pairs. For example, two 1's, two 2's, and so on. The cards are shuffled and placed face down on a table. In each round, the player chooses two cards that are faced down, turns them up, and if its is a match, they are left face up. If the two cards do not match, they are returned to their original face down position. The game continues until all cards are face up.
Write a program that implements the game. Use 16 cards (pairs of 1 through 8) laid on a 4×4 grid or table. Initially, all cards are faced down and your program must show the following

$$\begin{array}{c|cccc}
 & 1 & 2 & 3 & 4 \\
\hline
1 & \star & \star & \star & \star \\
2 & \star & \star & \star & \star \\
3 & \star & \star & \star & \star \\
4 & \star & \star & \star & \star
\end{array}$$

where a star indicates a face-down card. In each round the user is asked to provide two coordinates; for example, $(1,2)$ and $(3,3)$ where $(1,2)$ means the card on the first row and second column. Note that this coordinate system starts at 1 not 0. Moreover, as indicated by the the numbers on the table above, the coordinate system starts from top left. At this point, your program must first check that the two requested positions are valid; that is, they are inside the table and do not point to revealed cards (if not valid print a warning and ask again). Next, the program reveals the two cards for 2 seconds and hides them again. How to do that? You can simply output a large number of newlines (to force the old board out of the screen) and output a new table. About timing, you need to use Java's TimeUnit package. First import `java.util.concurrent.TimeUnit` in your code. Second, you can use `TimeUnit.SECONDS.sleep(2);` to enforce a pause of 2 seconds.
To recap, each turns starts by (1) getting an input from user (except the first round in which the program should print the above table), (2) printing another table with revealed cards, (3) if the cards match, the next round starts; else the current table is showed for 2 seconds and the original table is shown again. You do not need to randomly generate the table. Instead, use a fixed table in your code.
HINT: You need to use 2-dimensional arrays.
BONUS: Randomly generate a *valid* table on each execution. You get +20pts for this.