Project #3   project deal line: December 11th by Midnight
What to turn in:

1. You must email a soft copy to: csus.csc15@gmail.com. The subject of the email must indicate the following. An email without the proper subject will not be graded.
    a. Last name, first name
    b. HW #
    c. Section number
    d. Lecture time

Memory Matching Game
A common memory matching game played by children is to start with a deck of cards that contain identical pairs. For example, given six cards in the deck, two might be labeled 1, two might be labeled 2 and two might be labeled 3. The cards are shuffled and placed face down on the table. A player select two cards that are face down, turns them up, and if the cards match they are left face up. If the cards do not match they are returned to their original face down position. The game continues until all the cards are face up.

The problem
Write a program that plays the memory matching game. Your program must have three different classes:

- **Card** class encapsulating the concept of the card
- **MemoeyGame** class encapsulating the concept of the memory game.
- **playMemoryGame** class that will control the play of the game using the methods from the MemoryGame class.
-

UML diagram for the Card class

Card
-value: int -faceUp: boolean
+Card(int initValue) +isFaceUp() : boolean +getValue(): int +flipCard() : void

Instance variables
An integer variable representing the value of the card
A boolean variable indicating if the card is face up or down

Methods descriptions

- **Card(int initValue) :** Explicit value constructor initializes **value** to initValue and **faceUp** to false.
- **isFaceUp():** Returns the value of the variable **faceUp**.
- **getValue():** Returns the value of the instance variable **value.**

- **flipCard() :** Reverses the value of the variable **faceUp.** For example, if faceUp is false, it becomes true.

UML diagram for the MemoryGame class

MemoryGame
+BOARD_SIZE : int = 4
+NUM_SWAPS : int = 1000
- cards: card[][]
+ MemoryGame()
-initalizeCards(): void
+shuffleCards() : void
+showBoard() : void
+cardsMatch(int row1, int col1, int row2, int col2): boolean
+isFaceUp(int row, int col) : boolean
+flipCards(int row, int col1,int row2, int col2) : void
-hideBoard():void
-cardsFaceDown() : void

MemoryGame class
Instance variables

- A 4x4 array of **Card** representing sixteen cards.

Constants

- **BOARD_SIZE** represents the dimensions of the array **cards.**
- **NUM_SWAPS** represents the number of times that the content of the array **cards** needs to be swapped.

Method descriptions

- **MemoryGame()**
  No-argument constructor instantiates and initializes the array **cards.** This constructor calls the method initalizeCards() to set the instance variable **value** for each card(each element of the array).

- **initalizeCards()**
  Initialize the array **cards** to pair of numbers 1 through 8 in the known locations. Table below shows the content of the array **cards** after the initialization.

```
       1 2 3 4
ÏÏÏ   =========
ÏÏÏ1 | 1 2 3 4 |
ÏÏÏ2 | 5 6 7 8 |
ÏÏÏ3 | 1 2 3 4 |
ÏÏÏ4 | 5 6 7 8 |
ÏÏÏ   =========
```

- **shuffleCards()**
  Call the method **cardsFaceDown**
  Repeatedly (NUM_SWAPS times) selects two cards at random locations and swap them**.**
  This method requires:
    - a loop to go through the array **cards**
      - generate 4 different random numbers row1, col1, row2, col2
      - swap the values at the location [row1][col1] with the value at the location [row2][col2]

  Table below shows the array cards after its elements have been shuffled.

```
        1 2 3 4
ÏÏÏ   =========
ÏÏÏ1 | 3 4 8 2 |
ÏÏÏ2 | 5 6 5 1 |
ÏÏÏ3 | 3 7 4 7 |
ÏÏÏ4 | 6 2 1 8 |
ÏÏÏ   =========
```

- **showBoard()**
  Call the method hideBoard() to hide the previously printed board**.**
  Generate the current state of the array ***cards*** in the format shown in the output.
  It goes through the arrays **cards:**
    - If the instance variable **faceUp** of **cards[row][col]** equals false then a * is printed on the screen.
    - If the instance variable **faceUp** of cards[row][col] equals true then the instance variable **value** of cards[row1][col1] is printed on the screen.

  Table below shows the output at the beginning of the game where no match has been found yet.

```
       1   2   3  4
      =========
ÏÏÏ1 | * * * * |
ÏÏÏ2 | * * * * |
ÏÏ 3 | * * * * |
ÏÏ 4 | * * * * |
ÏÏÏ   =========
```

  Table below show the output where a match has been found.

```
       1 2 3 4
ÏÏÏ   =========
ÏÏÏ1 | 3 * * * |
ÏÏÏ2 | * * * * |
ÏÏÏ3 | 3 * * * |
ÏÏÏ4 | * * * * |
ÏÏÏ   =========
```

- **cardsMatch**(int row1, int col1, int row2, int col2)

Returns true if the cards at location **cards[row1][col1]** and **cards[row2][col2]** matches otherwise it will return false.
For example: cardsMatch(1,1,1,3) will return true since cards[1][1] is equal to cards[3][1].

- **isFaceUp(int row, int col)**
  Returns the value of the instance variable **faceUp** for the array element cards[row][col] **.**This method should call the method isFaceUp() in the Card class.

- **flipCards(int row1, int col1, int row2, int col2)**
  Reverse the value of the instance variable **faceUp** for the elements cards[row1][col1] and cards[row2][col2] by calling the **flipCard** method in the Card class.

- **hideBoard()**
  Hides the last board printed on the screen by scrolling it down using System.out.println() .For example to scroll down 10 lines repeat System.out.println() 10 times.

- **cardsFaceDown()**
  Sets the value of the instance variable **faceUp** to false for all the elements of the array **cards** by calling the method **flipCard** in the **Card** class.

PlayMemoryGame class

Public static void main (String[] args)

- Declare and instantiate an object of the **MemoryGame** class called **game.**
- Declare the variable **turnCount** to keep track of the numbers that the player takes turn
- Shuffle the cards.
- Loop until the player does not want to start a new game
  - Loop until all the matches have been found
    - Print the board on the screen
    - Call the method `getValidInt (String prompt, int min, int max)` to get values for row1, col1
      - While the card at the chosen location is already matched, call the getValidInt method to get another row1, col1
    - Call the method `getValidInt (String prompt, int min, int max)` to get values for row2, col2
      - While the card at the chosen location is already matched, call the getValidInt method to get values for row2, col2
    - Call the method **flipCards** to flip the cards at positions chosen by the player
    - Call the method **showBoard()** to show the board
    - Call the method **cardsMatch (**row1, col11, row2, col2)
      - If the selected cards match print a message that you found a match.
      - If the selected cards do not match flip the cards at locations row1, col1 and row2, col2.
  - After all the matches is found
    - output **turnCount**
    - print a message that game is over

        - Prompt the user if he/she wants to play again.
            - If player wants to play again
                - Reset the variables
            - If the player does not want to play
                - Print a goodbye message

Note: This method has 2 extra credit points.

public static int getValidInt(String prompt, int min, int max)
Arguments
prompt: string to prompt the user
min: minimum number that the user can enter
max : maximum number that the user can enter
Return value: an integer (row or column number)

Method description

- loop until you read a valid number from the keyboard
    - if the number being read from the keyboard is a valid integer number (use the **hasNextInt** method to do the validation)
        - If it is a Valid integer number
            - Reads it from the keyboard using **nextInt** method and store it in the variable **num**
                - if **num** is between max and min inclusive, you are done
                - If num is not within the range print error message
        - If it is a not a valid integer number
            - Print an error message

Sample output

```
Welcome to the Memory Game.
The goal is to find all the matching pairs in as few turns as possible.
At each turn select two different positions on the board to see if they match.

 Press any key to start the game.
```

```
    1 2 3 4
   =========
1 | * * * * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
Where is the first card you wish to see?
           Row: 1
           Col: 1
Where is the second card you wish to see?
           Row: 1
           Col: 3
    1 2 3 4
   =========
1 | 8 * 8 * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 * 8 * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
Where is the first card you wish to see?
           Row: 1
           Col: 2
Where is the second card you wish to see?
           Row: 4
           Col: 4
    1 2 3 4
   =========
1 | 8 2 8 * |
2 | * * * * |
3 | * * * * |
4 | * * * 2 |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 2 8 * |
2 | * * * * |
3 | * * * * |
4 | * * * 2 |
   =========
Where is the first card you wish to see?
           Row: 1
           Col: 4
Where is the second card you wish to see?
           Row: 3
           Col: 3
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | * * * * |
3 | * * 7 * |
4 | * * * 2 |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | * * * * |
3 | * * 7 * |
4 | * * * 2 |
   =========
Where is the first card you wish to see?
           Row: 2
           Col: 1
Where is the second card you wish to see?
           Row: 4
           Col: 2
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 * * * |
3 | * * 7 * |
4 | * 4 * 2 |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 * * * |
3 | * * 7 * |
4 | * 4 * 2 |
   =========
Where is the first card you wish to see?
```

```
           Row: 2
           Col: 2
Where is the second card you wish to see?
           Row: 3
           Col: 1
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 * * |
3 | 6 * 7 * |
4 | * 4 * 2 |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 * * |
3 | 6 * 7 * |
4 | * 4 * 2 |
   =========
Where is the first card you wish to see?
           Row: 2
           Col: 3
Where is the second card you wish to see?
           Row: 4
           Col: 3
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 3 * |
3 | 6 * 7 * |
4 | * 4 3 2 |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 3 * |
3 | 6 * 7 * |
4 | * 4 3 2 |
   =========
Where is the first card you wish to see?
           Row: 2
           Col: 4
Where is the second card you wish to see?
           Row: 4
           Col: 1
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 3 1 |
3 | 6 * 7 * |
4 | 1 4 3 2 |
   =========
You found a match


Press any key to continue.
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 3 1 |
3 | 6 * 7 * |
4 | 1 4 3 2 |
   =========
Where is the first card you wish to see?
           Row: 3
           Col: 2
Where is the second card you wish to see?
           Row: 3
           Col: 4
```

```
    1 2 3 4
   =========
1 | 8 2 8 7 |
2 | 4 6 3 1 |
3 | 6 5 7 5 |
4 | 1 4 3 2 |
   =========
You found a match


Press any key to continue.

        CONGRADULATIONS! You found all the matching pairs!

You did it in 8 turns.
That is your best score so far!


Do you want to start another game?
yes
```

```
Press any key to start the game.
```

```
    1 2 3 4
   =========
1 | * * * * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
Where is the first card you wish to see?
           Row: 2e
ERROR: 2e is not a valid integer (1...4)
           Row: 3
           Col: 5
ERROR: 5 is not in the valid range (1...4)
           Col: 3

    1 2 3 4
   =========
1 | * * * * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
First card: (row 3, col 3)
Where is the second card you wish to see?
        Row: 2
        Col: 4
    1 2 3 4
   =========
1 | * * * * |
2 | * * * 6 |
3 | * * 3 * |
4 | * * * * |
   =========
Sorry, No match.
Press any key to continue.
```

```
    1 2 3 4
   =========
1 | * * * * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
Where is the first card you wish to see?
           Row: 5
ERROR: 5 is not in the valid range (1...4)
           Row: 1
           Col: 1
    1 2 3 4
   =========
1 | * * * * |
2 | * * * * |
3 | * * * * |
4 | * * * * |
   =========
First card: (row 1, col 1)
Where is the second card you wish to see?
        Row: 2
        Col: 3
```

```
    1 2 3 4
   =========
1 | 4 * * * |
2 | * * 8 * |
3 | * * * * |
4 | * * * * |
   =========
Sorry, No match.
Press any key to continue.
```