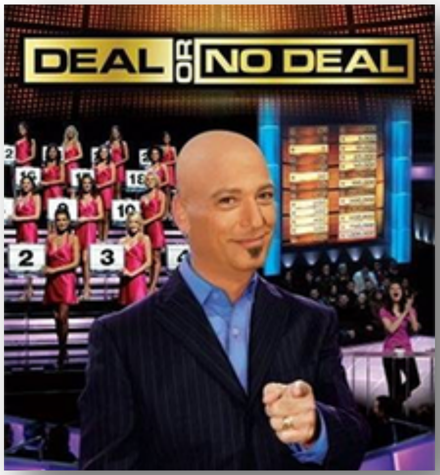
CSE1341 - Lab 5 Programming Assignment
Game with Multiple Methods
Due Saturday, March 30th, 2019 at 6:00 AM



This assignment will require a Pre-Lab and a Lab.

Our fifth Lab will be to build a variation of the TV game show *Deal or No Deal*. If you are unfamiliar with the game, it might help to view an episode online. You will build this in four incremental versions. Its best to build one increment successfully, then use that as the starting point for the next version.

Pre-Lab (5 POINTS): Create the DealGame1 class with method headers for *main* and *displayCases*. Read the DealGame1 instructions thoroughly and add comments describing the logic you will be adding within the body of both of the methods.

DealGame1 [30 points]

Create a class called DealGame1.java which contains two methods: *main* and *displayCases*.

Class structure and static variables
You will need two static variables that can be used in all methods in the class:

- An *int* array named c*ases* which contains the values of each of the 24 cases used in the game, with the following values:
- An *int* variable named *selectedCaseValue* that will contain the (hidden) dollar value of the case selected by the player at the beginning of the game.

Illustration of the cases array:

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
cases	1	5	10	25	50	75	100	200	300	400	500	1000	5000	10000	etc…

Note: Be careful in your code when prompting the user to select a case number. The index number is one less than the case number (e.g. Case #6, which contains $75, is at index # 5) *main* method

Contains the complete flow of the game, with a call to the *displayCases* method where needed.

Pseudocode:

- Welcome message
- Prompt user to select *their* case. Store the dollar amount in the selected case in the static int variable you declared in the class. *Tip: Watch your index numbers. If the player selects case #10, its index number is 9 (containing $400)*
- Tip: remember to import java.util.Scanner
- Save the contents of the selectedCase in the *selectedCaseValue* variable, then change it to a negative number by multiplying it -1 and saving the negative value in the array at the same location. This will indicate that it was already selected.
- Prompt user to begin selecting cases to <u>eliminate</u>.
    - Create a loop that continues perpetually until the game ends. Within the loop:
        - Call the *displayCases* method to print out all available cases. (See instructions for this method below.)
        - Prompt the player to select a case number.
        - Print the contents of the selected case (from the *cases* array)
        - Change the value corresponding to that case in the *cases* array to a negative number by multiplying its value by -1 and saving the negative number at the same array location.
        - Repeat until the player exits the game with ctrl-c. After each selection, the *displayCases* method will display an updated view of the available cases, with the case they just selected removed.

displayCases method

- Prints a header with the text “Available Cases”.
- Create a loop that loops through both arrays using the loop counter variable as the array’s index number. For each element, if the its value is less than 0 (negative), print 10 blank spaces. Otherwise, print the text [CASE *nn*] substituting the case number for *nn.*
- Maintain another counter variable with the number of items printed. After every sixth case (or blanks) is printed, print a CRLF and reset the counter to restart counting for the next row. This will print four rows with six cases (or blanks) in each row.

DealGame1 sample output:

```
$ java DealGame1
Welcome to Deal or No Deal!

Please select your case (1-24): 10
You now have case #10 and it's time to eliminate cases.


Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14] [CASE 15] [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]

Please pick a case to eliminate: 15
Case 15 contained $25,000

Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14]           [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]

Please pick a case to eliminate: 15
Case 7 contained $-25,000
```

Selecting the same case again confirms that the value changed to a negative #

```
Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14]           [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]

Please pick a case to eliminate: 7
Case 7 contained $100

Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
          [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14]           [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]

Please pick a case to eliminate: (Hit CTRL-C to exit)
```

DealGame2 [30 points]

Create a copy of DealGame1.java and name it DealGame2.java. You will create and use two new methods in this version: *shuffleCases* and *getOffer*. Make the following enhancements to *DealGame2:*

shuffleCases method (new)

- Using counter controlled repetition, loop through all the elements in the *cases* array. Within the loop:
  - Using the library class *Random*, create a Random object to generate a random number from 0-23. (Remember to import java.util.Random)
  - Save the contents of the case at the location of the loop's current counter variable (starting with 0) in a newly declared variable (variable1).
  - Save the contents (dollar amount) of the value in the random index location in another newly declared variable (variable2).
  - Swap the values by storing variable2 contents in the location of the loop's current counter variable and store variable1 in the location of the random number.
  - Continuing this loop for all 24 values will cause all values to be swapped once.
  - Wrap this loop in another outer loop which repeats the shuffle 100 times, for a fully shuffled array.

getOffer method (new)

- Using the contents of the *cases* array, loop through all array elements (0—24) and sum the values of all of the *cases* values *that have not been changed to a negative number.*
- Use a new int variable to count the number of values included in the sum.
- After summing up all the positive values, add the selectedCaseValue to the sum, and increment the case count one more time.
- When you exit the loop, divide the sum by the count, and store this average in a new int variable. (It will truncate the result due to division of two int values.)
- Return this average at the end of the method. This is the offer amount.

Changes to *main* method

- After the user selects a case to eliminate, and the case value is displayed, call the *getOffer* method and save the int value it returns in a new variable.
- Display the offer amount and ask the player "(D)eal or (N)o Deal?"
- Prompt the user to answer the question using a String to contain their answer.
- If they choose "D", display the original case value and notify them whether or not they made a good deal by comparing with the offer amount they accepted, then break out of the game loop to exit.
- If they choose "N", do nothing and repeat the game loop for the next selection.

DealGame2 sample output:

```
$ java DealGame2
Welcome to Deal or No Deal!

Please select your case (1-24): 10
You now have case #10 and it's time to eliminate cases.


Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14] [CASE 15] [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]

Please pick a case to eliminate: 3
Case 3 contained $75,000


The banker offers $ 129,212 for your case

(D)eal or (N)o Deal? N

Available cases:
----------------
[CASE  1] [CASE  2]           [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14] [CASE 15] [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]

Please pick a case to eliminate: 24
Case 24 contained $10,000


The banker offers $ 134,888 for your case

(D)eal or (N)o Deal? N

Available cases:
----------------
[CASE  1] [CASE  2]           [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9]           [CASE 11] [CASE 12]
[CASE 13] [CASE 14] [CASE 15] [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23]

Please pick a case to eliminate: 1
Case 1 contained $50,000


The banker offers $ 139,133 for your case

(D)eal or (N)o Deal? D
Congratulations on accepting the $ 139,133 offer!
Not a great deal.  Your case had $500,000.
```

DealGame3 [15 points]

Create a copy of DealGame2.java and name it DealGame3.java. There are no new methods in this version, but you will make changes in your *main* method to prevent user data entry errors.

Avoid entering duplicate values in the main method:

- To prevent selecting a case that has previously been selected, wrap the logic that prompts the user for a case to eliminate in a loop.
- After a case number is entered, verify that its corresponding value in the *selections* array is *false.* If *true,* print a message that a duplicate was entered and remain in the loop which will prompt the user to enter the value again. If *false*, break out of this loop to continue game play.

DealGame3 sample output:

```
$ java DealGame3
Welcome to Deal or No Deal!

Please select your case (1-24): 5
You now have case #5 and it's time to eliminate cases.

Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4]           [CASE  6]
[CASE  7] [CASE  8] [CASE  9] [CASE 10] [CASE 11] [CASE 12]
[CASE 13] [CASE 14] [CASE 15] [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]
Please pick a case to eliminate: 3
Case 3 contained $1

The banker offers $ 155,330 for your case

(D)eal or (N)o Deal? n

Available cases:
----------------
[CASE  1] [CASE  2]           [CASE  4]           [CASE  6]
[CASE  7] [CASE  8] [CASE  9] [CASE 10] [CASE 11] [CASE 12]
[CASE 13] [CASE 14] [CASE 15] [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]
Please pick a case to eliminate: 3
That case was already selected
Please pick a case to eliminate: 5
That case was already selected
Please pick a case to eliminate: 6
Case 6 contained $25


The banker offers $ 162,725 for your case

(D)eal or (N)o Deal? d
Congratulations on accepting the $ 162,725 offer!
You made a great deal!  Your case had $400.
```

DealGame4 [20 points]

Create a copy of DealGame3.java and name it DealGame4.java. This version will display a sorted list of all dollar amounts that have not yet been eliminated to the player. This list will contain fewer items with each turn as cases are eliminated. Note that the contents of the player's original case will always be included on this list because it was selected, not eliminated.

Create a new method called *showRemainingValues*. Create a new int array which is a complete copy of the cases array. We need to create of the array so we can sort it to display the remaining dollar values in numerical order, and we don't want to change the sort order of the original cases array. Note that all eliminated cases will be a negative number, and will sort to the front of the list. However, we want to keep the value of player's case on the remaining values list, so we first need to find it in the copied array, change it back to a positive number, then sort the new array a second time.

displayRemainingValues method (new):

- Copy the *cases* array and create a new array. (**hint**: java.util.Arrays copyOfRange method)
- Sort the newly copied array to prepare to do a search. (**hint**: java.util.Arrays *sort* method)
- Search the copied array to find the *selectedCaseValue*. Remember that this value was changed to a negative number, so you'll need to search for -1 * *selectedCaseValue*. (**hint:** java.util.Arrays binarySearch method will return the index number of the value you are searching for.
    - Once found, use the index number to change the value back to a positive number by multiplying it by -1.
- After changing the selectedCaseValue back to a positive number, sort the list again.
- Using the sorted copy of the array with the adjusted selectedCaseValue, create a loop to print all positive numbers in the array. (Just skip the negative numbers as you go through the loop.)
    - For this method, print 5 values per row, so in addition to the counter to manage looping through the copied array, you'll need another counter variable to keep track of how many values have been printed on the row. After it reaches 5, print a CRLF and reset that counter to count values on the next row.

NOTE: A portion of the solution is provided below to help you complete this (non-trivial) problem:

```
//create new array to contain remaining dollar values
int[] remainingValues = Arrays.copyOfRange(cases,0,cases.length);
Arrays.sort(remainingValues);
int selectedCaseIndex = Arrays.binarySearch(remainingValues, selectedCaseValue * -1);
remainingValues[selectedCaseIndex] = selectedCaseValue;
Arrays.sort(remainingValues);
```

main method changes:

- At the top of your game loop, before calling *displayCases*, add another statement to call the new method *displayRemainingValues.*

```
$ java DealGame4
Welcome to Deal or No Deal!

Please select your case (1-24): 15
You now have case #15 and it's time to eliminate cases.


Remaining Values:
-----------------
$         1 $         5 $        10 $        25 $        50
$        75 $       100 $       200 $       300 $       400
$       500 $     1,000 $     5,000 $    10,000 $    25,000
$    50,000 $    75,000 $   100,000 $   200,000 $   300,000
$   400,000 $   500,000 $   750,000 $ 1,000,000
Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
[CASE  7] [CASE  8] [CASE  9] [CASE 10] [CASE 11] [CASE 12]
[CASE 13] [CASE 14]           [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]
Please pick a case to eliminate: 8
Case 8 contained $1,000,000


The banker offers $ 109,893 for your case

(D)eal or (N)o Deal? n

Remaining Values:
-----------------
$         1 $         5 $        10 $        25 $        50
$        75 $       100 $       200 $       300 $       400
$       500 $     1,000 $     5,000 $    10,000 $    25,000
$    50,000 $    75,000 $   100,000 $   200,000 $   300,000
$   400,000 $   500,000 $   750,000
Available cases:
----------------
[CASE  1] [CASE  2] [CASE  3] [CASE  4] [CASE  5] [CASE  6]
[CASE  7]           [CASE  9] [CASE 10] [CASE 11] [CASE 12]
[CASE 13] [CASE 14]           [CASE 16] [CASE 17] [CASE 18]
[CASE 19] [CASE 20] [CASE 21] [CASE 22] [CASE 23] [CASE 24]
Please pick a case to eliminate: 10
Case 10 contained $10,000


The banker offers $ 114,650 for your case

(D)eal or (N)o Deal? d
Congratulations on accepting the $ 114,650 offer!
You made a great deal!  Your case had $1.
```

NOTES: Comment your program to explain your steps. Each program should compile without errors and should run to produce outputs described for each exercise. The following points will be discounted if the related element is missing or incorrect:

- Proper names for classes, variables, and methods [1 point each]
- No Comments [5 points]
- Program doesn't compile [5 points for each minor error up to 5 errors provided that after fixing the errors the program compiles. If the program does not compiler after the 5 errors are fixed, partial credit will be given not to exceed 50 points]
- Source code (java file) missing [15 points]
- Executable (class file) missing [15 points]
- Both java file and class file missing [100 points]
- Missing method where required [5 points each]