
Due Date:	By 11:55pm Wednesday March 22, 2017
Evaluation:	4% of final mark (see marking rubric at the end of handout)
Late Submission:	none accepted
Purpose:	The purpose of this assignment is to help you learn Java flow of control statements (nested loops), arrays of primitive types, arrays of strings, 2-dimensional arrays.
CEAB/CIPS Attributes:	Design/Problem analysis/Communication Skills

General Guidelines When Writing Programs:

Please refer to the handout of Assignment #1.

Question 1 - One dimensional array & while loops

There are 30 hockey teams in the NHL (National Hockey League). Some stores have vending machines that dispense miniature team hockey pucks for a loonie (\$1) each. When you put in a loonie you never know which puck you will get, any one of the 30 team pucks is as likely as any other to be dispensed by the machine. They are given out at random.


For this exercise pick 10 hockey teams.

Your job is to write a program to simulate the dispensing of NHL mini pucks until all 10 have been obtained.

Your program should proceed as follows:

1. Display a welcome message and ask the user to enter their name.
2. Store your 10 hockey team names in an array of String. Assign the team names in the declaration statement directly.
3. Your program should loop until all team pucks have been dispensed. Create an integer array of size 10, which will serve as a counter array to keep track of the number of each team puck dispensed by the vending machine. You will need to use the `Math.random()` function to determine which team puck was dispensed. The `Math.random()` method returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
4. Once you have accumulated all 10 pucks, display how many of each team puck you had to purchase, the total number of pucks and the cost in a personalized message.

Note:

1. Assume you have a perfect user who will follow instructions and enter a valid first name.
2. Figure 1 is a sample output screen which illustrates the expected behavior of your program.
3. Feel free to change the messages, prompt. You can pick any 10 hockey teams you like.
4. User input is marked by 

```

= 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 =
= 0
= 0      NHL Miniature Hockey Puck Vending Machine      0 =
= 0
= 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 = 0 =

Hello, what is your first name? Nancy
Welcome Nancy! Let's see how much money you will need to spend to
get all of the pucks.

Here is the breakdown of the pucks dispensed:
Montreal Canadians: 2
Boston Bruins: 4
Chicago BlackHawks: 2
Detroit Red Wings: 3
New York Rangers: 3
Buffalo Sabres: 1
Philadelphia Flyers: 1
Winnipeg Jets: 4
Vancouver Canucks: 5
Ottawa Senators: 8

Wow Nancy, you bought a total of 33 pucks at a total cost of $33 to
get a miniature puck of each team.
Enjoy them!

```

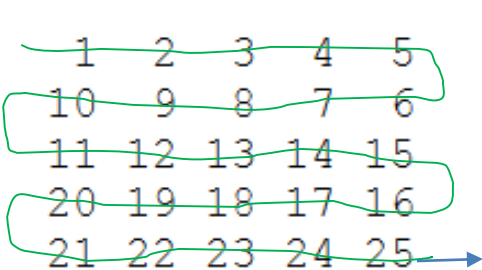
Figure 1- Sample output for Question 1

Question 2 – 2-dimensional arrays, while loops & do ... while loops

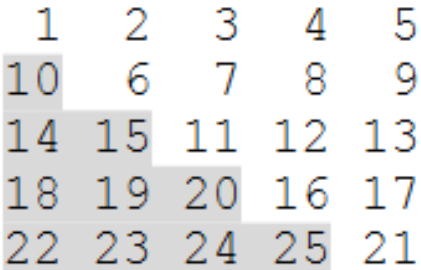
Write a Java program which fill a 2D-array square array as per a given pattern and display it.

Your program should proceed as follows:

1. Display a welcome message.
2. Prompt the user for an integer which is ≥ 3 . If the number entered is < 3 , keep prompting the user until they enter a number ≥ 3 .
3. Fill the array as per pattern 1 and display it.
4. Fill the same array as per pattern 2 and display it.
5. Display a closing message.



Pattern 1



Pattern 2 (Some slots are shaded to help you see the pattern.)

Following are a couple of screen captures to illustrate the expected behavior of your program.

```
[-----]
[   Array Pattern   ]
[-----]

How many rows/columns do you want your array to have? (Must be at least 3):
6
Pattern number 1:
  1  2  3  4  5  6
12 11 10  9  8  7
13 14 15 16 17 18
24 23 22 21 20 19
25 26 27 28 29 30
36 35 34 33 32 31

Pattern number 2:
  1  2  3  4  5  6
12  7  8  9 10 11
17 18 13 14 15 16
22 23 24 19 20 21
27 28 29 30 25 26
32 33 34 35 36 31

All done!!!!
```


```
[-----]
[   Array Pattern   ]
[-----]

How many rows/columns do you want your array to have? (Must be at least 3):
0
Let's try this again ....
How many rows/columns do you want your array to have? (Must be at least 3):
2
Let's try this again ....
How many rows/columns do you want your array to have? (Must be at least 3):
3
Pattern number 1:
  1  2  3
  6  5  4
  7  8  9

Pattern number 2:
  1  2  3
  6  4  5
  8  9  7

All done!!!!
```

Note:

- You can use static methods if you like.
- User input is marked with .
- Use *printf* to format the way the array is displayed.

Question 3 – Defining a class

“Combination locks found on padlocks, lockers, or safes may use a single dial which interacts with several parallel discs or *cams*. Customarily, a lock of this type is opened by rotating the dial clockwise to the first numeral, counter clockwise to the second, and so on in an alternating fashion until the last numeral is reached. The cams typically have an indentation or notch, and when the correct combination is entered, the notches align, allowing the latch to fit into them and open the lock. “

https://en.wikipedia.org/wiki/Combination_lock#Single-dial_locks



<http://shopping.com/products/combination-lock>

a) Create a class named **CombinationLock** that stores information about a 3-number dialing combination lock. It contain the following:

- **Four private instance variables** that store the 3 combination numbers required to open the lock and a Boolean variable that will indicate whether the lock is open or closed (true if open, false if closed).
- 2 constructors:
 - default constructor - sets the 3 combination numbers to zero and marks the lock as closed.
 - three parameter constructor - sets the 3 combination numbers to the three passed integers and marks the lock as closed.
- No accessor methods
- A Boolean mutator method called `setCombination`, which is a 3 parameter method and sets the 3 combination numbers to the 3 passed numbers. It can only change the combination if the lock is closed. If the lock is open it returns `false` as it could not change the combination, if the lock is closed it returns `true` after it has reset the combination.
- A public void method called `closeLock()` which set the status of the lock as closed.
- A public Boolean method `isOpen()` which returns `true` if the lock is open and false if it is closed.

- A public Boolean method called `openLock()` which is a 3 parameter method which checks if the supplied combination is the correct one. If it is correct it returns true, if it is incorrect it returns false.
- A public integer method `HowManyCorrect()` which is a 3 parameter method. It returns the count of correct numbers in the supplied combination.
- A `toString()` method to return the combination of the lock in the following format: `Clockwise num1 - Counter-Clockwise num2 - Clockwise num3` where `num1`, `num2` and `num3` are the values of the instance variables corresponding to the combination which will open the lock.
- An `equals()` method to test for equality of the combination of two objects of class `CombinationLock`. We don't care whether they are open or closed.

b) Write a driver to test your class from a) as follows:

- Display wWelcome message/banner
- Declare 2 CombinationLock objects** each using a different constructors (you decide on the info) and output the description of the 2 object. Methods to use: constructors, `toString()`
- Open one of the locks** : Try once with the incorrect combination, then try a second time with the correct combination. In both cases display a status message as to the success or failure to open the lock. Methods to use: `openLock()`
- Change the combination of one of the locks**: Test your mutator method by changing the combination of the lock you opened in the previous step. (Make sure both locks do not have the same combination.) **Remember you can only change the combination if the lock is closed.** Methods to use: `setCombination()`, `isOpen()`, `closeLock()`
- Open the other lock**: Try once with the incorrect combination and report back how many of the entered numbers are correct. Try again with the correct combination and display a congratulations message. Methods to use: `openLock()`, `howManyCorrect()`
- Check 2 locks for equality**: Test both locks for equality. Report the result and the open/close status of each lock. Change the combinations of one of the locks to be the same as the other. Test both locks for equality again report the result and the open/close status of each lock again. . Methods to use: `equals()`, `isOpen()`, `setCombination()` .
- Closing message**: Finish off with a closing message.

Following is a sample run to illustrate the expected behavior of your driver.

```
|-----|
| Locksmith Program |
|-----|
```

```
Combination for lock 1 is: Clockwise 0 - Counter-Clockwise 0 - Clockwise 0
Combination for lock 2 is: Clockwise 13 - Counter-Clockwise 7 - Clockwise 11
```

```
==> Attempting to open lock 1
Sorry the combination you supplied for lock 1 is incorrect
Congratulations you opened lock 1
```

```
===> Attempting to change the combination of lock 1
You can't change the combination if the lock is open. Close the lock and try
again
Congratulations you successfully changed the combination of lock 1 to
Clockwise 3 - Counter-Clockwise 7 - Clockwise 11
```

```
===> Attempting to open lock 2
2 of the numbers are correct. Try again.
Congratulations you opened lock 2
```

```
===> Testing the two locks for equality
Lock 1 combination is: Clockwise 3 - Counter-Clockwise 7 - Clockwise 11
Lock 2 combination is: Clockwise 13 - Counter-Clockwise 7 - Clockwise 11
Combination of both locks is the NOT same
Lock 1 is not open
Lock 2 is open
```

```
Lock 1 combination is: Clockwise 3 - Counter-Clockwise 7 - Clockwise 11
Lock 2 combination is: Clockwise 13 - Counter-Clockwise 7 - Clockwise 11
Combination of both locks is the same
Lock 1 is not open
Lock 2 is open
```

```
Hope you are comfortable with the manipulation of objects after this!!!
```

Submitting Assignment 3

- Zip the source code (the .java file only please) of this assignment.

- Naming convention for zip file: Create one zip file, containing the source files for your assignment using the following naming convention:
 - If the assignment is done by 1 student:
The zip file should be called *a#_studentID*, where # is the number of the assignment and *studentID* is your student ID number.
For example, for this assignment, student 123456 would submit a zip file named *a3_123456.zip*
 - If the assignment is done by 2 students:
The zip file should be called *a#_studentID*, where # is the number of the assignment *studentID1* and *studentID2* are your student ID numbers. For example, for this

assignment, student 123456 and 9876543 would submit a zip file named
a3_123456_9876543.zip

- For submission instructions please refer to the course web page.
- **Assignments not submitted to the correct location or not in the requested format will not be graded.**

Evaluation Criteria for Assignment 3 (20 points)

Source Code	
Programming Style (3 pts.)	
Comments-description of variables and constants-description of the program (authors, date, purpose)	0.5 pt.
Clear prompts to user & clear message with output	1 pt.
Use of significant names for identifiers	0.5 pt.
Indentation and readability	0.5 pt.
Welcome Banner/Closing message	0.5 pt.
Question 1 (5 pts.)	
Reading data and verifying input	1 pt.
Selection for Poem lines/variation in nouns-adjectives	2 pt.
Generate different versions of poem	1 pt.
Display poem properly formatted	1 pt.
Question 2 (7 pts.)	
Initialization and display of booking layout	1 pt.
Reading seat assignment and verifying data	2 pt.
Checking seat availability	1 pt.
Make reservation and display price	1 pt.
Update and display booking layout	1 pt.
Repeat for different booking sessions	1 pt.
Question 3 (5 pts.)	
Constructor-accessor-mutator	1.5 pt.
Comparison (=,<,>) , string	1 pt.
Other methods	0.5 pt.
Test code	2 pt.
TOTAL	20 pts.