Homework #15

More practice with creating & manipulating arrays of primitive type. Also more practice with program design.

1. Aim

Gain experience in creating and manipulating arrays. For this first array exercise, we will create arrays of primitive type, rather than class type, as they are slightly easier to deal with.

The second part of this lab leaves the design of the solution up to you. Rather than supplying you with a detailed problem description and how to break down the problem, you are given a more open-ended problem. Make sure you take the time do create a good design before you start coding – it will make your life so much easier. Use methods to help handle small tasks for you.

2. Files Needed

You will not be provided with any starter files for this homework. Create a project in Eclipse for this assignment. For each exercise below, create a Java class with the appropriate name. When you create each program, be sure to have Eclipse create the public static void main method for you. Be sure to include the standard header comments at the top of each file.

3. To be Handed In

The files **Mailboxes.java** and **MasterMind.java** should be sent to me when you have them completed. Be sure to name the files/classes as specified.

4. Exercises

Part I: The bored postman

Complete the following exercise by writing appropriate Java code in the file **Mailboxes.java.**

The night postman was bored one evening and he decided to experiment with the mailboxes in the mailroom. The mailboxes were numbered from zero to 200 (there were 201 total mailboxes). Initially all boxes were closed. Starting with mailbox #2, he opened the door of every even numbered mailbox. Next, beginning with mailbox #3 and going to every third mailbox, he opened its door if it was closed and closed it if it was open. Then he repeated this procedure with every fourth mailbox starting with mailbox #4; and then every fifth mailbox starting with mailbox #5; continuing in this manner all the way up to the changing every 200th mailbox starting with mailbox #200 (which of course just changes mailbox #200). In the end, he was surprised at the distribution of closed mailboxes.

Your job is to write a program to determine which mailboxes were closed at the end of the experiment. Write this program in the **Mailboxes.java** file. The program should ask the user for how many mailboxes are in the mailroom, and then perform the desired experiment for that many mailboxes. This will allow us to run the experiment for a small mailroom with only 201 mailboxes, or a large mailroom with thousands of mailboxes.

Recommendations/Hints:

1) Simply create an array of integers of the desired size. Each mailbox will be represented by the array element whose index matches the mailbox number; i.e., mailbox #0 will be represented by the array element at index 0.
2) Let the integer value zero represent a closed mailbox and the integer value one represent an open mailbox. Alternatively, you could use an array of booleans and let the values true & false represent open & closed.
3) After creating an array of the needed size, be sure to initialize the array so that all mailboxes start in the "closed" position.
4) You will need a double-nested loop (a loop inside a loop) to conduct the experiment.
5) After conducting the experiment, only print the numbers of the boxes that are in the closed position, starting at mailbox #0. Also print out the distance between the closed mailboxes. See the sample output below for the format of the output.
6) Note how the above splits the main part of your program into 3 sections: (a) create & initialize the array, (b) the actual experiment, and (c) printing of the results. You should write separate methods to tackle the different sections of this problem.

7) You are not required to ask the user if they want to run the program again, though you can do so if you want.

Example:

This is a sample execution to show you how **Mailboxes.java** should behave, it is run for only 5 mailboxes so that you can see the format of the output but without ruining your fun of determining the results of the experiment:

```
How many mailbox are in the mailroom?
5

When done, here are all the mailboxes that are closed:
Box #0;   distance from previous box: 0
Box #1;   distance from previous box: 1
Box #4;   distance from previous box: 3
```

Part II: MasterMind.java

Our project will be to write a program that plays a variation of the game of Master Mind. If you are not familiar with the game, visit the Wikipedia page (http://en.wikipedia.org/wiki/Mastermind_%28board_game%29).

Your task is to write a program that plays a game of Master Mind under the following rules:

1. We will use 4 integer numbers rather than 4 colored pegs. You will want to use an array of ints to hold these numbers.
2. Each number will be in the range 1 to 6.
3. Numbers may not be repeated. I.e., the hidden numbers can be {3, 2, 6, 4}, but they cannot be {2, 5, 1, 2} because the number 2 is repeated. This restriction makes it a bit easier on the person playing the game, but more importantly for us, it also simplifies the logic necessary to give proper feedback on a player's guess.
4. Whenever the user wants to play a game (or play another one), your program should create the array of 4 hidden numbers by generating random integers in the range 1 to 6, ensuring that there are no repeats.
5. Each time the player makes a guess, they will enter 4 integers (store them in another array). You will need to verify that the numbers they entered are in the expected range and that they are all unique. You then need to provide proper feedback on how many numbers were correct in both value & position, and how many number were correct in value but in the wrong position. See the sample execution below. [Not allowing duplicate numbers makes this task a bit easier.]
6. When the player finally wins, inform them how many guesses it took, and ask them if they want to play again. When the player is done playing games, report the number of games played, the total number of guesses made in all the games, the average number of guesses per game, and the number of guesses made in their best game.
7. You should write your program in such a way so that if someone wants to change it to have a different number of hidden numbers (say 5 or 6 rather than 4) then you would only have to change one line of code in your program. Similarly, if someone wanted to change the range of values for the numbers to something other than 1 to 6 (say 1 to 9), then you would again only have to change one line of code.
8. Be sure to think about your design and how to break the problem down into smaller, manageable pieces. Write separate methods to handle all the pieces. As an example, my solution has 5 methods and approximately 90-110 lines of Java code (not counting blank lines or lines containing only a comment or curly braces). None of my method has more than 25 lines of code. For this assignment, we are placing a limit of 25 lines of code per method (again, not counting blank lines or lines containing only a comment or curly braces).

Those are the guidelines for this project. As you see, I have not given you a high level design – that will be your job. You might want to consider the examples you have seen in prior programming assignments. Be sure to create small helper methods as needed. Your design will account for one sixth of your grade for this program. **Think** before you code!

Going above & beyond: If you want an additional challenge once you have completed the MasterMind game, consider removing the restriction that the hidden numbers and the user's guess numbers are unique. This makes generating the hidden numbers much easier, since you no longer have to prevent duplicates, but it makes the task of giving a proper response much more difficult. This is because if there are duplicate numbers in the guess, they cannot all be counted as a correct guess (whether correct position or not) unless they correspond to the same number of duplicate numbers in the hidden code. If you do this extra challenge, please be sure to make a note of it in your Oak submission so that the grader is aware of it. This work is worth up to 20% extra credit on this portion of the assignment.

Example:

This is a sample execution to show you how **MasterMind.java** should behave:

```
I am thinking of 4 numbers in the range 1 to 6.

Enter 4 numbers separated by blanks: 1 2 3 4
You had 0 correct numbers in the correct positions
And you had 2 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 4 6 7 5
You entered a value not in the range 1 to 6  ← Note: does not count as a guess
Try again.

Enter 4 numbers separated by blanks: 1 2 5 2
Your guess contained duplicate numbers (the number 2 appeared at least twice).
Try again.                                   ← Note: also does not count as a guess

Enter 4 numbers separated by blanks: 3 4 5 6
You had 1 correct numbers in the correct positions
And you had 2 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 3 1 6 5
You had 2 correct numbers in the correct positions
And you had 2 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 3 6 1 5
You had 1 correct numbers in the correct positions
And you had 3 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 6 1 3 5
You had 0 correct numbers in the correct positions
And you had 4 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 3 5 6 1
You had 4 correct numbers in the correct positions
And you had 0 correct numbers in the wrong positions

Congratulations, you solved the puzzle in 6 guesses.

Do you want to play again? (Y|N)
y
I am thinking of 4 numbers in the range 1 to 6.

Enter 4 numbers separated by blanks: 6 5 4 2
You had 1 correct numbers in the correct positions
And you had 3 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 6 4 2 5
You had 0 correct numbers in the correct positions
And you had 4 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 2 5 6 4
You had 4 correct numbers in the correct positions
And you had 0 correct numbers in the wrong positions

Congratulations, you solved the puzzle in 3 guesses.

Do you want to play again? (Y|N)
y
I am thinking of 4 numbers in the range 1 to 6.

Enter 4 numbers separated by blanks: 1 2 5 6
```

```
You had 0 correct numbers in the correct positions
And you had 3 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 3 6 2 5
You had 0 correct numbers in the correct positions
And you had 3 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 2 5 6 4
You had 2 correct numbers in the correct positions
And you had 0 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 2 5 1 3
You had 1 correct numbers in the correct positions
And you had 2 correct numbers in the wrong positions

Enter 4 numbers separated by blanks: 2 3 6 1
You had 4 correct numbers in the correct positions
And you had 0 correct numbers in the wrong positions

Congratulations, you solved the puzzle in 5 guesses.

Do you want to play again? (Y|N)
n

You played a total of 3 games.
You made 14 total guesses over those games.
That is an average of 4.666666666666667 guesses per game.
Your best game was 3 guesses.
```

5. Additional requirements

A. You must start your program with header comments that provide your name, VUnetID, email address, the date the program was last modified, an honor statement ("I have neither given nor received unauthorized help on this assignment"), and a short description of the program (see the examples distributed with homework #2).
B. Each method that you write (except main), should be preceded by a block of Javadoc comments which describe what the method does, what the parameters are, and what it returns. Any required preconditions should also be clearly stated.
C. For this assignment, we are placing a limit of 25 lines of code per method (again, not counting blank lines or lines containing only a comment or curly braces).
D. You should use a consistent programming style. This should include the following.
    a. Meaningful variable & method names
    b. Consistent indenting
    c. Use of "white-space" and blank lines to make the code more readable
    d. Use of comments to explain pieces of tricky code
    e. A descriptive JavaDoc comment before each method (other than main, which already has a short description of the program). Note that we will be using JavaDoc comments from here on out to document our methods.
    f. Lines that do not extend beyond column 100 (or column 80 is even better)

See the code examples in the class text for a good formatting style.

6. Submission for grading

Once you have completed the exercise, the files **Mailboxes.java** and **MasterMind.java** should be sent to me.

7. Grading

This lab is worth 45 points – 15 points for Part I and 30 points for Part II. Your grade will be based on whether your solution is correct or not, and on how closely you followed the directions above. Remember that programming style will now be a larger part of your grade. 5 of the 30 points for Part II will be based solely on the design of your MasterMind solution.