Arrays and 2 Dimensional Arrays

Due: no later than Friday, August 19, 2016

The purpose of this assignment is to design and implement algorithms involving arrays and 2D arrays.

Provided Files:

File	Description	Responsibility
ArrayProblems.java	Class that contains 6 methods to be completed by you and some simple tests in a main method. Add more tests to the main method of this class ***if you can***	You and Me. This is the file you will turn in.

Description: Complete the 6 methods in ArrayProblems.java. The methods are explained in further detail in the Tips section of this page. When completing the methods, you are encouraged to break the problem up into smaller problems and create helper methods of your own to solve these smaller problems.

VERY IMPORTANT: For this assignment the **only** methods and classes from the Java standard library you may use in your final solutions are:

- methods from the System class AND
- methods from the Math class AND
- Of course you can use the length field on native arrays (arr.length) and create and use other native arrays if you want to.

Note all the methods require that the parameter not be altered in any way. You can create local copies of the parameters and alter them if you want to.

ArrayProblems.java includes a main method with some tests for the methods. **Some of these tests may be incorrect. You are required to find and fix any incorrect tests. share information about other students about which tests are in error and how to correct them.** Your methods must pass the corrected tests. Add at least 4 more tests per method (24 additional tests total) to the main method of ArrayProblems.java.

Fill in the header for ArrayProblems.java.

Replace <NAME> with your name. Note, you are stating, on your honor, that you did the assignment on your own.

Checklist: Did you remember to:

1. work on the assignment by yourself or with one other person in the class?
2. fill in the header in your file ArrayProblems.java?
3. implement the six required methods?
4. ensure your program does not suffer a compile error or runtime error?
5. identify and correct any incorrect tests provided with the assignment?
6. ensure your program passes the tests in the main method of ArrayProblems.java?
7. add the required number of test cases to the main method of ArrayProblems.java?
8. turn in your source code in a file named ArrayProblems.java to the proper account

---

Tips:

1. Hamming Distance. "Hamming distance between two strings of equal length is the number of positions for which the corresponding symbols are different. Put another way, it measures the number of *substitutions* required to change one into the other, or the number of *errors* that transformed one string into the other." From the Wikipedia article on Hamming Distance. For this problem you will be working with arrays of ints instead of String objects. Complete the following method in ArrayProblems.java

```
/**
* Determine the Hamming distance between two arrays of ints.
* Neither the parameter aList or
* bList are altered as a result of this method.
* @param aList != null, aList.length == bList.length
* @param bList != null, bList.length == aList.length
* @return the Hamming Distance between the two arrays of ints.
*/
public static int hammingDistance(int[] aList, int[] bList){
```

For example given the array {1, 2, 3, 4, 5, 4, 3, 2, 1} and the array {1, 2, **8**, 4, 5, 4, 3, **5**, 1} the Hamming distance is 2.

2. Permutations. This method determines if one int array is a permutation of another int array.

```
/*
  pre: listA != null, listB != null
  post: return true if listB is a permutation of listA, false
otherwise.
  neither listA or listB are altered as a result of this method.
*/
public static boolean isPermutation(int[] listA, int[] listB)
```

"A permutation, also called an "arrangement number" or "order," is a rearrangement of the elements of an ordered list S into a one-to-one correspondence with S itself." [mathworld.wolfram.com] For example the list {1, 2} has the following permutations; {1, 2} and {2, 1}.

Note the elements of listA and listB are lists, not sets, so duplicate items could appear. So for example given the list {1, 2, 2} the unique permutations are {1, 2, 2}, {2, 1, 2}, and {2, 2, 1}. {2, 1} is not a permutation of {1, 2, 2}. Another example of lists that are not permutations of each other: {2, 1, 1} is not a permutation of {2, 2, 1}.

Hint: Do not try to solve the problem by taking one the arrays and generating all the permutations of that array and then check to see if the other array is one of those permutations. That is way too inefficient except for arrays with a very small number of items.

3. Smallest Difference. Given an array of ints, find the indices of the pair of integers that have the smallest absolute difference between them.

```
/** Find the two values in an array that are closest to
* each other. In other words find the two nearest
neighbors.
* The parameter nums is not altered as a result of
* this method.
* @param nums The array of ints to find the nearest
neighbors in.
* nums != null, nums.length >= 2
* @return Returns an array of length 2.
* The elements of the result are the indices of ints in
* nums that have the smallest distance (absolute value of
* difference) of any pair of ints in nums.
*
```

```
* If there is more than one pair of ints that meet this
* criteria returns the indices of the pair with the minimum
index.
*
* If there is more than one pair of ints with the minimum
* index, returns the indices of the pair with the smaller
* second index.
*
* The first element of the result is the smaller of the two
indices.
* For example given the array {5, 3, 21, 10, 12, 7} the
method
* would return {0, 1}.
*/
public static int[] nearestNeighbors(int[] nums){
```

For example, consider the following array: {19, 0, -5, 4, 7, 10}

Here are the distances (absolute value of the difference) between the various pairs of elements.

Pair	Difference	Pair	Difference	Pair	Difference	Pair	Difference	Pair	Difference
19, 0	19	19, -5	24	19, 4	15	19, 7	12	19, 10	9
0, -5	5	0, 4	4	0, 7	7	0, 10	10		
-5, 4	9	-5, 7	12	-5, 10	15				
4, 7	3	4, 10	6						
7,10	3								

Given the above values there are two pairs of ints that have a difference of 3. The pairs are 4 and 7, and 7 and 10. The method would return an array with the indices 3 and 4 which are the indices of the elements 4 and 7 in the array nums.

The hierarchy of importance when ties occurs is :

1. The actual difference between two elements (not their indices)
2. The pair with the lower first index (if the actual difference is the same between two pairs)
3. The pair with the lower second index (if the first indices are the same between two pairs)

4. Max Sum. Complete a method that determines which row or column in a 2d array of ints has the largest sum. Here is the method header.

```
/**
* determine which row or column in a matrix has the largest sum.
 <p>pre: mat != null, mat.length > 0,
  mat is a rectangular matrix, mat[0].length > 0

  <p>post: determine which row or column of ints has the maximum
sum in max.
If a row contains the maximum sum, return a string starting with
"R" and then the number of the row with no spaces. For example
"R0" or "R12". If a column contains the maximum sum, return a
string starting with "C" and then the number of the column with no
spaces. For example "C0" or "C12".
If there is more than one row or column with the maximum
sum return rows over columns first, then smaller indices over
larger indices.
Thus if rows 3, 5, and 12, and columns 0 and 2 all contained
the same maximum sum the method would return "R3".
*/
```

5. Most valuable plot of land. A 2d array of ints will be used to represent the value of each block in a city. The value could be negative indicating the block is a liability to own. Complete a method that finds the value of the most valuable contiguous sub rectangle in the city represented by the 2d array. The sub rectangle must be at least 1 by 1. (If all the values are negative "the most valuable" rectangle would be the negative value closest to 0.)

Consider the following example. The 2d array of ints has 6 rows and 5 columns per row, representing an area of the city. The cells with the white background represent the most valuable contiguous sub rectangle in the given array. (Value of 15.)

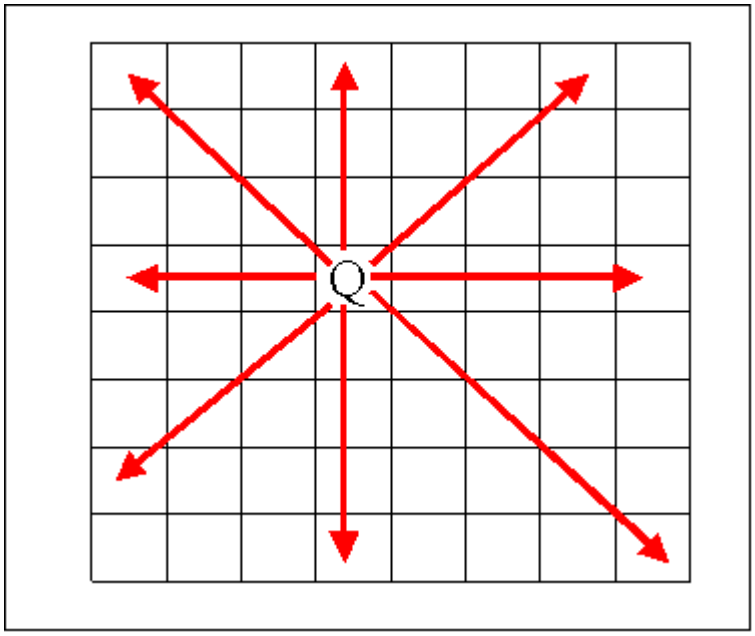
0	**-2**	**-7**	**0**	**-1**
9	**2**	**-6**	**2**	**0**
-4	**1**	**-4**	**1**	**0**
-1	**8**	**0**	**-2**	**1**
-10	**1**	**1**	**-5**	**6**
-15	**-1**	**1**	**5**	**-4**

Here is another example with the same 2D array with only one change. The value of the block at row 4, column 2 has been changed from 1 to 6. Given that configuration the most valuable contiguous sub rectangle in the given array has a value of 17 and is the cells with the white background.

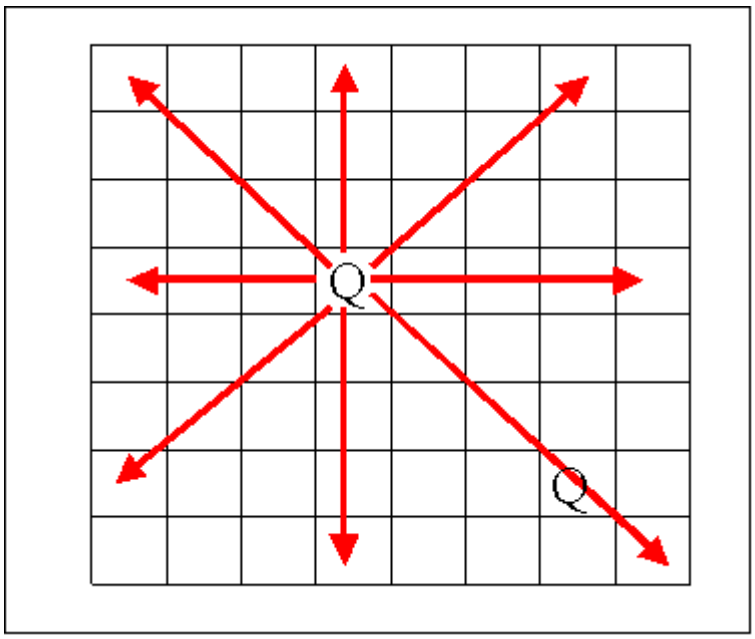
0	**-2**	**-7**	**0**	**-1**
9	**2**	**-6**	**2**	**0**
-4	**1**	**-4**	**1**	**0**
-1	**8**	**0**	**-2**	**1**
-10	**6**	**1**	**-5**	**6**
-15	**-1**	**1**	**5**	**-4**

6. Are the queens Safe? There is a chess and programming problem called the 8 queens problem. The goal is to place eight queens on a chess board so that none of them may attack any other queen. That is, no two queens are in the same row, column, or diagonal. In chess a queen may move any number of spaces straight up, down, left, right, or along any of the 4 diagonals. In the method you are completing the board is square (same number of rows as columns) but is not necessarily 8 by 8.

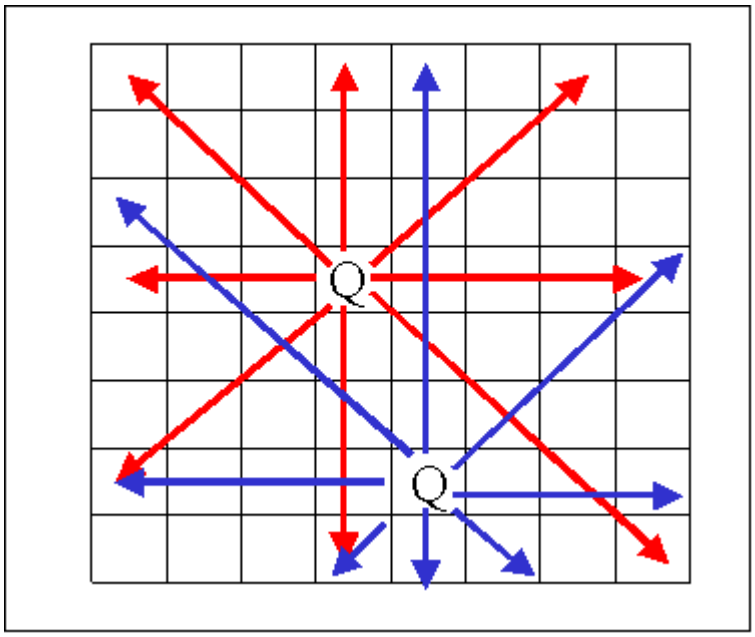
Consider the following board:



A queen's position is designated with the Q. The red arrows show the squares that queen can attack. Thus if there were a queen in any of those squares this would be an unsafe board. So the following set up is unsafe.



The following set up is safe, but the number of other safe squares is going down fast.



..

Note the example only shows two queens, but there is no precondition limiting the number of queens.

Complete a method that checks if a given board represents a safe placement of Queens.

```
/*
 pre: board != null, board.length > 0, board is a square
matrix. (In other words all rows in board have board.length
columns.), all elements of board == 'q' or '.'. 'q's
represent queens, '.'s represent open spaces.
 post: return true if the configuration of board is safe,
that is no queen can attack any other queen on the board.
Return false otherwise.
 The parameter board is not altered as a result of this
method.
*/
public static boolean queensAreSafe(char[][] board)
```

Design and style:

- Use assertions or if statements that throw exceptions to check for preconditions and post conditions on all public methods.
- Make instance variables within classes private.
- Do not use static data as a form of global variables.
- Break up long methods (over 25 lines of code) into smaller methods.
- Use consistent tabbing to make you code easier to understand and follow.
- Use a consistent brace style to make your code easier to understand and follow.
- Include white space to make your code easier to understand and follow
- Comment your code as necessary. (Complicated algorithms or sections of code should be commented. Comments should summarize what is going on, not restate the obvious.)

Documentation:

- Use /* ... */ or // style comments for implementation details. Explain your implementation so someone not familiar with your solution can understand it.
- Follow Sun's Coding Conventions.
- Use good variable names.