(Lab 6)

Containers: Arrays & Lists

[Due: October 26, 11:59 PM]

Submission Rules:

1. Submissions must be zipped into a **handin.zip** file. Each problem must be implemented in its own class file. Use the name of the problem as the class name.

2. You must use standard input and standard output for ALL your problems. It means that the input should be entered from the keyboard while the output will be displayed on the screen.

3. Your source code files should include a comment at the beginning including your name and that problem number/name.

4. The output of your solutions must be formatted exactly as the sample output to receive full credit for that submission.

5. Compile & test your solutions before submitting.

6. Each problem is worth up to 10 points total. The breakdown is as follows: 2 points for compiling, 3 points for correct output with sample inputs, 5 points for additional inputs.

7. You will get full marks if you can solve Problems **1,2,3,4 5,6**. Any score above 60 will be counted as bonus marks.

8. Submission:

- You have unlimited submission attempts until the deadline passes
- You'll receive your lab grade immediately after submitting
- You must submit a report (.pdf preferred) showing the result of your successful run for each of the solved problems. See below for reference:

```
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java ReverseTester1
[e, d, c, b, a]
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java ResizeTester1
[a, b, c, null, null, null]
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java AddTester1
[a, b, c, d, null, null]
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java ContainsTester1
true
false
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java MinMaxByIndexTester1
[0, 5]
[2, 3]
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java MinMaxByValueTester1
1
3
Correct!
```

```
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java CeasarCypherTester1
Uryy|-d|yq
Hello World
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java VectorUtilTester1
32.0
Correct!
PS C:\Users\Sharmin Aktar\OneDrive - University of New Orleans\Teaching\FALL_23\Lab06\Upload> java ConwayGameOfLifeTester1
[[false, false, false], [false, false, false], [false, false, false]]
Correct!
```

Problem 1: Array Util (10 points) Part 1: Reverse Array

(Data Structure algorithms) Arrays contain collections of related data values. An important component involving arrays is the reordering of its internal data values, a process commonly known as sorting. Sorting may be more formally defined as modifying the position of elements based on a set of rules or criteria. In this problem you must reverse the order of elements within in the array. This is a trivial form of sort as it reorders all elements based on their position in array and not the value.

Array Util Method API:

Modifier and Type	Method and Description
`static void`	**`reverse`**`(String[] array)` Reverses the elements within a String array

Facts

- Implement this method in the same **`ArrayUtil`** class as Problems 1,2,3,4 5,6
- No return is necessary because the array is passed into the method by reference
- Your **`ArrayUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`ArrayUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`ArrayUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Results ***(Not Printouts)***
`String[] arr = {"a","b","c","d","e"};` `ArrayUtil.reverse(arr);`	`{"e","d","c","b","a"}`

Problem 2: Array Util (10 points) Part 2: Array Resize

(Data Structure algorithms) **`ArrayList`** is a class in the **`java.util`** package that provides much more functionality than standard arrays. One powerful feature of **`ArrayList`** is that they can dynamically resize themselves, whereas a basic array has a fixed length determined during its initialization. **`ArrayList`** resize by creating a new Array twice the size of their original array and then copy their values to the new bigger array. Implement a **`resize`** method within your **`ArrayUtil`** class as specified in the API below.

Array Util Method API:

Modifier and Type	Method and Description
`static String[]`	**`resize`**`(String[] array)` Returns new array with the same elements as original but that's twice the length

Facts

- Implement this method in the same **`ArrayUtil`** class as Problems 1,2,3,4 5,6
- A return is required because a new array is created in memory
- Your **`ArrayUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`ArrayUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`ArrayUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***
`String[] arr = {"a","b","c"};` `arr = ArrayUtil.resize(arr);`	`{"a","b","c",null,null,null}`

Problem 3: Array Util (10 points) Part 3: Add Item

(Data Structure algorithms) **`ArrayList`** is a class in the **`java.util`** package that provides much more functionality than standard arrays. **`ArrayList`** has an **`add`** method which appends a new item to the end of its list. If the array is too small to add a new item, then it should first be resized, and then the new item should be added. Implement an **`add`** method within your **`ArrayUtil`** class as specified in the API below.

Array Util Method API:

Modifier and Type	Method and Description
`static String[]`	**`add`**`(String element, String[] array)` Returns an array with new element inserted at end, resize array if too small

Facts

- iterate array for a **`null`** reference and set item into that index.
- A return is required because a new array might need to be generated when adding a new item
- Implement this method in the same **`ArrayUtil`** class as Problems 1,2,3,4 5,6
- Your **`ArrayUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **ArrayUtil** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **ArrayUtil** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***
`String[] arr = {"a","b","c"};` `arr = ArrayUtil.add("d", arr);`	`{"a","b","c","d",null,null}`

Problem 4: Array Util (10 points) Part 4: Array Contains

(Data Structure algorithms) **ArrayList** is a class in the **java.util** package that provides much more functionality than standard arrays. A useful method in **ArrayList** is the **contains** method which reports true/false whether the list contains an element or not. Implement a similar **contains** method within your **ArrayUtil** class as specified in the API below.

Array Util Method API:

Modifier and Type	Method and Description
`static boolean`	**contains**`(String element, String[] array)` Returns true if the array contains the given String, otherwise false

Facts

- Java **String** class contains ***equals*** method that allows checks for String equality
  - https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
- Implement this method in the same **ArrayUtil** class as Problems 1,2,3,4 5,6
- Your **ArrayUtil** class implementation should ***not*** have a **main** method.
- **NO Scanner** for input & **NO System.out** for output!

Input

The **ArrayUtil** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`ArrayUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns *(Not Printouts)*

`String[] arr = {"a","b","c"};`
`ArrayUtil.contains("b",arr);`
`ArrayUtil.contains("d",arr);`
	`true`
	`false`

Problem 5: Array Util (10 points) Part 5: MinMax By Value

(Data Structure algorithms) Given an array of integer values, create methods that return the minimum integer value and the maximum integer value.

Array Util Method API:

Modifier and Type	Method and Description
`static int`	**`findMinValue`**`(int[] array)` Returns the min value from the array
`static int`	**`findMaxValue`**`(int[] array)` Returns the max value from the array

Facts

- Implement this method in the same **`ArrayUtil`** class as Problems 1,2,3,4 5,6
- Your **`ArrayUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`ArrayUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **ArrayUtil** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`int[] arr = {1,2,3,3,2,1};`
`ArrayUtil.findMinValue(arr);`
`ArrayUtil.findMaxValue(arr);`
	`1`
	`3`

Problem 6: Array Util (10 points) Part 6: MinMax By Index

(Data Structure algorithms) Given an array of integer values, create methods that return an array of the indexes where the minimum integer values occur and an array of the indexes where the maximum integer values occur. If a max or min value only occur once, then the array need only hold one index.

Array Util Method API:

Modifier and Type	Method and Description
`static int[]`	**`findMinIndex`**`(int[] array)` Prints the min value in the array
`static int[]`	**`findMaxIndex`**`(int[] array)` Prints the max value in the array

Facts

- Implement this method in the same **`ArrayUtil`** class as Problems 1,2,3,4 5,6
- Your **`ArrayUtil`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input

The **`ArrayUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`ArrayUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`int[] arr = {1,2,3,3,2,1};`
`ArrayUtil.findMinIndex(arr);`
`ArrayUtil.findMaxIndex(arr);`
	`{0,5}`
	`{2,3}`

Problem 7: Ceaser Cypher (10 points)

(Cyber Security) In cryptography, a shifting cipher, also known as Caesar's cipher, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each character in the message is replaced by a character some fixed number of positions up the alphabet. This a symmetric scheme as the same key is used for both encryption and decryption. This means the key shouldn't be made public.

Caesar Cypher Method API:

Modifier and Type	Method and Description
`static String`	**`encrypt`**`(String message, int key)` Returns encrypted message shifting characters up by key value
`static String`	**`decrypt`**`(String message, int key)` Returns decrypted message shifting characters down by key value

Facts

- Java **`String`** class contains ***`toCharArray`*** method that converts String into char array
  - https://docs.oracle.com/javase/10/docs/api/java/lang/String.html
- **`char`** data is also numerical data that support arithmetic operations
- After mathematical operations, You may cast **`int`** to **`char`** to convert number into ASCII
- Concatenation operations may convert **`char`** data into **`String`** data
- Your **`CeaserCypher`** class implementation should ***not*** have a **`main`** method.
- **NO** **`Scanner`** for input & **NO** **`System.out`** for output!

Input
The **`CeaserCypher`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output
The **`CeaserCypher`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***
`CeaserCypher.encrypt("Hello World",13);` `CeaserCypher.decrypt("Uryy|-d|\177yq",13);`	`"Uryy|-d|\177yq"` `"Hello World"`

Problem 8: Vector Util (10 points)
(Linear Algebra) Linear Algebra is a domain in mathematics that defines operations on entire fields of data, instead of individual numbers. These fields of data are called vectors or matrices, and are represented as an ordered set of numbers in the form of v1 = (x,y,z,...). For instance, an (x,y) coordinate is an example of a vector. Matrix operations have many applications including Scientific Computing, Data Science, Computer Graphics, Physics Simulations, Machine Learning, Image Processing, and many more.

A common operation is the dot product which defines how to multiply two vectors together. in the case of one dimensional vectors, i.e. a single sequence of numbers. (x,y,z,..) You simply multiply the values at each index together and then sum all of the resulting products together to derive a final result.

In other words,
Given a vector, v1 = (a,b,c) and vector, v2 = (x,y,z)
then v1 * v2 = ax + by + cz

So for example:
(1,2,3) * (4,5,6)
= 1*4 + 2*5 + 3*6
= 4 + 10 + 18
= 32

Vector Util Method API:

Modifier and Type	Method and Description
`static double`	**`dotProduct`**`(double[] vector1, double[] vector2)` Returns the dot product between matrix1 and matrix2

Facts

- Both vectors must have the same number of elements
- Your **`VectorUtil`** class implementation should ***not*** have a **`main`** method.
- **NO `Scanner`** for input & **NO `System.out`** for output!

Input

The **`VectorUtil`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`VectorUtil`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`double[] a = {1,2,3};`
`double[] b = {4,5,6};`
`VectorUtil.dotProduct(a,b);`
	`32.0`

Problem 9: Conway Game Of Life (10 points)

(Bioinformatics) . Simulation. The game is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. Game of Life is a 2d grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. The grid wraps around edges such that all cells have 8 neighbors. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbors dies, as if by underpopulation.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overpopulation.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Implement a method, that given the current grid of living/dead cells, determine the next generation of the grid using the rules from above.

Conway Game Of Life Method API:

Modifier and Type	Method and Description
`static boolean[][]`	**`update`**`(boolean[][] grid)` Returns new grid with the next lifecycle based on this current grid's state
`static int`	**`countNeighbors(`**`int i, int j, boolean[][] grid`**`)`** Returns the count of live cells adjacent to this cell's position (i,j)

Facts

- Each cell uses the 8 adjacencies from the given array to determine if its new state
- Grid wrapping logic can be achieved using the **`Math.floorMod`** method for indexing
- The **update** method used by the **countNeighbors** method to help decide each cell's state.
- Your **`ConwayGameOfLife`** class implementation should ***not*** have a **`main`** method.
- **NO `Scanner`** for input & **NO `System.out`** for output!

Input

The **`ConwayGameOfLife`** class will be accessed by an external Java Application within Autolab. This Java app will send data in as arguments into each of the methods parameters.

Output

The **`ConwayGameOfLife`** class should return the correct data calculations back to the invoking client code

Sample Method Calls	Sample Method Returns ***(Not Printouts)***

`boolean[][] grid =`
`{`
`  {true, false, false},`
`  {false, true, true },`
`  {false, true, true}`
`};`
 	
`ConwayGameOfLife.update(grid);`
	`[`
	`  [ false, false, false ],`
	`  [ false, false, false ],`
	`  [ false, false, false ]`
	`]`