CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 3

Methods, and Single-Dimensional and Two-Dimensional Arrays in Java

Due: Wednesday, October 19, 11:59 PM

Learning Goals:

- Write Java code that includes:
  - Methods
  - Single Dimensional Arrays
  - Two Dimensional Arrays
- Unit testing

NOTE: This assignment should be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Get Started with Starter Code (0 points)

1. Make sure there is no problem with your Java software development environment. If there is any, review Assignment 1 or come to the office/lab hours before you start Assignment 3.
2. First, navigate to the `cse8b` folder that you have created in Assignment 1 and create a new folder titled `assignment3`
3. **Download the starter code.**
   You can download the starter code from Piazza → Resources → Homework → `assignment3.zip` and **unzip** it.

a. Once you have downloaded the zip file, double click on it from the Finder (for Mac) or File Explorer (for Windows).
b. A folder should be created in the same location as the zip file. You can find the files with the starter code within that folder.

4. The starter code should only be two files called `StarbucksOrderQueue.java` and `MatrixTransform.java`. Place the starter code within the `assignment3` folder that you have just created.
5. Compile and run the starter code, and you should expect the following output:
   For `StarbucksOrderQueue.java`,

```
$ javac StarbucksOrderQueue.java
$ java StarbucksOrderQueue
Testing addOrder()
FAILED: unexpected output for addOrder()
ERROR: Failed test.
```

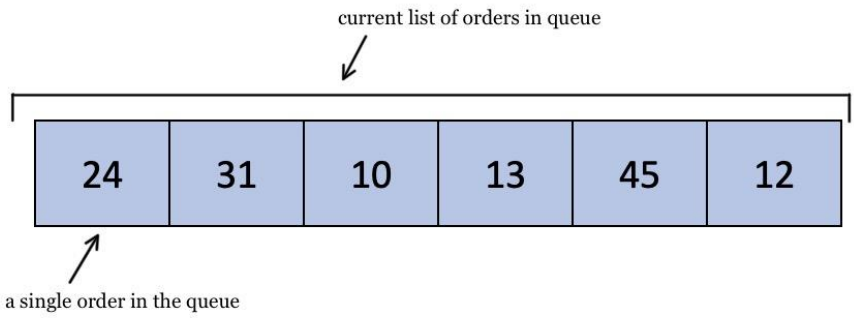
   For `MatrixTransform.java`,

```
$ javac MatrixTransform.java
$ java MatrixTransform
Testing mathTransform()
FAILED: unexpected output for mathTransform()
ERROR: Failed test.
```

Although we have not covered Java classes in lecture yet, you can think of Java classes as an isolated entity that contains methods. The starter code is set up so you need not know anything about the details of creating a class from scratch. Your job is to just fill in the starter code with methods that have been explained below. *(We have mentioned the places where you need to fill in the code)*

Part 1: Implement Starbucks Queue (46 points)

In `StarbucksOrderQueue.java` starter code, you are provided with a class named `StarbucksOrderQueue`. In the methods below, `queue` stores the order numbers that are remaining so the Starbucks staff can process each order, and `n` is an order number as shown below:



You need not concern yourself with what classes are. You will be creating and implementing the following methods inside this class:

- `addOrder(int n, int[] queue)`
- `removeOrder(int n, int[] queue)`
- `getOrderTime(int n, int[] queue)`

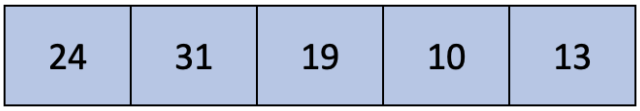
Each of these methods have been explained below.

1. **`addOrder(int n, int[] queue)` (14 points)**
   Let's say there is a new customer at Starbucks and they give their order. Their order will be assigned an order number and you will need to add this to the queue so the staff can process it. This method takes in a parameter `n` and places the order number stored in `n` at the back of the `queue`.
   - The order number *cannot be negative*, therefore you will be placing the absolute value of `n` at the back of the `queue`.
   - There cannot be two orders with the same order number. If there is already an order with the value of `n`, then return `queue` unmodified. *(How will you check if an element is present in an array or not?)*
   - If the order number has been successfully added to `queue`, then return the new queue in variable `newQueue`.
   - ***Hint:*** *What will be the length of the new array that you will need to add the new order?*

   Parameters: `int, int[]`
   Returns: `int[]`
   Description: Add new order to the Starbucks queue
   Example:
   Let's assume that the current `queue` looks like this:

   Now, if `n = -13` is passed to the method as a parameter, then `newQueue` would look like this:

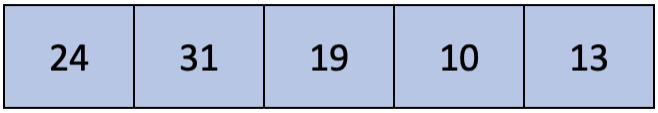
   Note: In the starter code, `newQueue` is initialized to an array of **one** element with default value. You will need to change the length of the array to an appropriate number while you are implementing this method.

2. **`removeOrder(int n, int[] queue)` (14 points)**

Let's say a customer decides to cancel their order, in which case you will need to remove their order number from the `queue`. This method takes in a parameter `n` and removes that order number from the `queue`.

- Think about how the positions of elements in the array are changing.
- If an order is not present in the queue, then return `queue` unmodified.
- After removing an order number, return the new queue in variable `newQueue`.
- ***Hint:*** *What will be the length of the new array after you remove an order?*

Parameters: `int n, int[]`
Returns: `int[]`
Description: Remove an order from the Starbucks queue
Example:
Let the current `queue` look like the below image and `n = 19`:



If you call this method with the same example as above, then `newQueue` will look like this:

Hint*: Here, the position of 24 and 31 doesn't change, but the position of 10 changes from 4th to 3rd and the position of 13 changes from 5th to 4th and so on.*

Note: In the starter code, `newQueue` is initialized to an array of **one** element with default value. You will need to change the length of the array to an appropriate number while you are implementing this method.

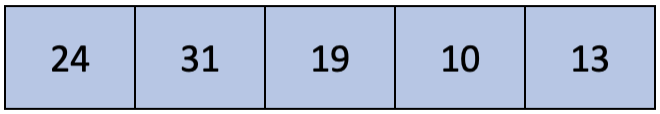
3. `getOrderTime(int n, int[] queue)` (12 points)

This method is used to estimate the time when the order will be ready. For the purpose of this assignment you will assume each order (i.e., each element in the `queue`) takes **4 minutes** to be prepared. Given the order number, this method computes the minutes until that order number in `queue` will be done being prepared, stores the resulting number of minutes in a variable named `result`, and **returns** it. Some things to consider:

- How is the position of the element in an array relative to the amount of time taken?
- If `queue` is empty, then return `0`.
- If an order is not present in `queue`, then return `-1`.
- In the below example, `19` is the third element, what is the *index* of `19` in `queue`? Can you use that value to compute the time?

Parameters: `int, int[]`

Returns: `int`
Description: Calculate the number of minutes taken for an order to be complete
Example:
Let the current `queue` look like the below image and `n = 19`:



Then, if you call this method, it will return the value 12. Since 19 is the third element in queue the time taken for it to be processed will be $4 * 3 = 12$ minutes.

Note: In the starter code, this method returns `0` by default. You will need to change that to an appropriate value while you are implementing this method..

Part 1.1: Test Correctness of Three Methods (6 points)

Similar to the previous assignment, you will be testing your code. **In this part of the assignment, you need to implement your own test cases in the method called `unitTests`.**

In the starter code, several test cases are already implemented for you. You can regard it as an example to implement other cases. The general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method should return `true` only when all the test cases are passed. Otherwise, it should return `false`. **To get full credit for this section, for each method, you should have at least four total test cases that cover different situations (including the ones we have provided).** In other words, you will need to create three more tests for `addOrder`, three more tests for `removeOrder`, and three more tests for `getOrderTime`.

Part 2: Implement Matrix Transformation (44 points)

A matrix is a rectangular 2D array of numbers. For this question, you will be performing matrix transformations to familiarize yourself with how 2D arrays are used in Java. In `MatrixTransform.java`, there is a class `MatrixTransform`. You will be creating and implementing the following methods inside this class:

- `mathTransform()`
- `dataTransform()`

Each of these methods have been explained below.

1. **`mathTransform(int[][] dataMatrix)` (20 points)**
   This method transforms the matrix based on the values of indices of each of its elements as explained below.

- Assume that `i` represents the row index and `j` represents the column index of the `dataMatrix`.
- If `i` and `j` are equal, you will compute the **square** of the value of the original element.
- If `i + j` is an odd number, you will compute the **negative of the product** of `(i + j)` and the original element.
- If `i + j` is an even number, you will compute the **absolute value of the product** of `(i - j)` and the original element.
- Return the transformed matrix in variable `newMatrix`.

Parameters: `int[][]`
Return: `int[][]`
Description: Transform matrix with math operations
Example:
Consider the following matrix:

	0	1	2
0	3	5	9
1	8	1	4
2	7	6	5

After transformation, `newMatrix` will have the following values.

	0	1	2
0	9	-5	18
1	-8	1	-12
2	14	-18	25

- Notice how the elements in `(0,0), (1,1), (2,2)` have been squared to `9, 1, 25` respectively. As described above, in these positions `i` is equal to `j`.
- Likewise, the element at index `(1,2)` is transformed as `-(1+2) * 4 = -12`. As described above, in this position the sum `(i+j) = (1+2) = 3` is odd.
- Similarly, the element at `(2,0)` is transformed as `abs((2-0) * 7) = 14`. As described above, in this position the sum `(i+j) = (2+0) = 2` is even.

Note: In the starter code, `newMatrix` is initialized to a 2D array with one row and one column. You will need to change the length of the rows and columns to an appropriate number while you are implementing this method.

2. **`dataTransform(int[][] dataMatrix)` (20 points)**

In this method, you will be rearranging the data in the `dataMatrix` by generating a transpose of the matrix **and** vertically flipping it.

- A transpose of a matrix is formed by interchanging rows into columns and columns into rows.
- A vertical flip of a matrix is its mirror image across the vertical axis. Take a look at the example below for better understanding.
- Similar to the previous function, you will store this result in a variable called `newMatrix` and return it.
- **Note**: Make sure to compute the transpose first, then flip the matrix. If you do it in the opposite order, the autograder will fail.

Parameters: `int[][]`
Return: `int[][]`
Description: Create a transpose-then-vertical-flip of a matrix
Example:
Consider the following matrix:

1	2
3	4
5	6

After transformation, the transpose of the matrix will look like this:

1	3	5
2	4	6

Now, if you vertically flip the matrix, `newMatrix` will look like this:

5	3	1
6	4	2

*(**Hint**: If `i` represents the row index and `j` represents the column index, notice how the element position `(i, j)` changes when transposing a matrix. Similarly, notice how `(i, j)` changes when vertically flipping the matrix.)*

Note: In the starter code, `newMatrix` is initialized to a 2D array with one row and one column. You will need to change the length of the rows and columns to an appropriate number while you are implementing this method.

Part 2.1: Test Correctness of Two Methods (4 points)

Similar to the previous part of the assignment, you will be testing your code. **In this part of the assignment, you need to implement your own test cases in the method called `unitTests`.**

In the starter code, several test cases are already implemented for you. You can regard it as an example to implement other cases. The general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method should return `true` only when all the test cases are passed. Otherwise, it should return `false`. **To get full credit for this section, for each method, you should have at least four total test cases that cover different situations (including the ones we have provided).** In other words, you will need to create three more tests for `mathTransform` and three more tests for `dataTransform`.

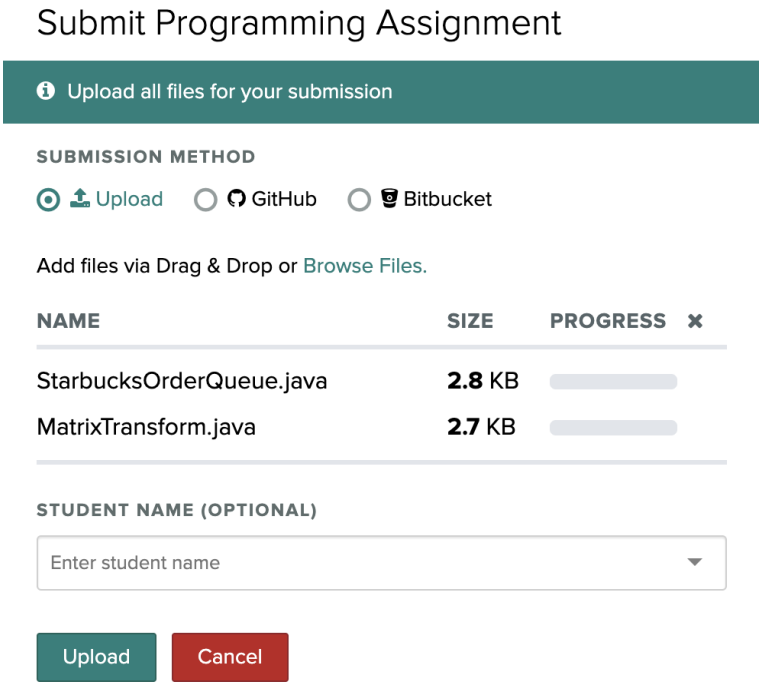
Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on Assignment 2.
2. Click the DRAG & DROP section and directly select the required files (`StarbucksOrderQueue.java` **and** `MatrixTransform.java`). Drag & drop is fine. **Please make sure you don't submit a zip. Make sure the filename is correct.**
3. **You can resubmit an unlimited number of times before the due date.** Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of basic tests. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

5. Your submission should look like the screenshot below. **If you have any questions, then feel free to post on Piazza!**


Submit Programming Assignment
Upload all files for your submission
SUBMISSION METHOD
Upload
GitHub
Bitbucket
Add files via Drag & Drop or Browse Files.
NAME
SIZE
PROGRESS
StarbucksOrderQueue.java
2.8 KB
MatrixTransform.java
2.7 KB
STUDENT NAME (OPTIONAL)
Enter student name
Upload
Cancel