Contents

Matrix Rotation

Value

Base problem

- Base implementation: 5 points
- Base unit tests: 5 points

Extra credit

- In-place implementation: 4 points
- Base unit tests on in-place implementation: 3 points
- Parameterized unit tests on base implementation: 3 points if in addition to base unit tests, 6 points if instead of base unit tests
- Parameterized unit tests on in-place implementation: 3 points if in addition to base unit tests on in-place implementation, 5 points if instead of base unit tests

Background

In many graphics processing and data analysis tasks, we need to rotate a matrix of data. This might be as visually intuitive as rotating an image 90° clockwise, or it might be more abstract, less directly connected to a visual result; the fundamental task is essentially the same, regardless.

Base problem

Your assignment is to write code to rotate a square matrix of integers (in Java terms, an array of arrays of integers, or `int[][]`, where the number of elements in each row is the same as the number of rows) 90° clockwise.

Examples

For example, assume we start with the matrix

$$\begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$$

Your code would produce the result

$$\begin{pmatrix} 4 & 1 \\ 3 & 2 \end{pmatrix}$$

Implementation details

We strongly recommend you read (carefully) **all** of the following, including the "Unit testing" section, before writing any code.

- Write a method with the following declaration:

  ```
  public static int[][] rotate(int[][] data)
  ```

  That is, your `rotate` method will take a parameter that is a 2-dimensional array of `int`, and return a new 2-dimensional array of `int`, with exactly the same dimensions. (In this case, we have the additional knowledge that the arrays will be square.)

  The body of this method must perform the processing described above, to construct and return a matrix that is rotated 90° clockwise from the original.

- Your method should be in a `public` class named `Matrices`, in the `edu.cnm.deepdive` package.

- Your code should be an an IntelliJ Java project (not Android), using either JDK 8 (1.8) or JDK 11.
- Your code might include one or more additional methods as helper methods. These should probably be written as `private static` methods in the `Matrices` class.

Base unit tests

Create a test class using JUnit5, with one or more test methods annotated with `@Test`, to test your implementation with the following test cases. (See "Incorporating JUnit5 into Your IntelliJ IDEA Project" for details on incorporating JUnit5 into your project.)

Case 1: 1 X 1 matrix

Input

$$\begin{pmatrix} -1 \end{pmatrix}$$

In Java terms, this is a two-dimensional array (an array of arrays), with 1 row and 1 column, i.e. :

```
int[][] data = {
  {-1}
};
```

Expected output

$$\begin{pmatrix} -1 \end{pmatrix}$$

(Rotating a 1 X 1 matrix must give us an array that is identical to the input.)

Case 2: 2 X 2 array

Input

$$\begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$$

In Java terms, this is a two-dimensional array (an array of arrays), with 2 rows and 2 columns, i.e.

```
int[][] data = {
  {1, 2},
  {4, 3}
};`
```

Expected output

$$\begin{pmatrix} 4 & 1 \\ 3 & 2 \end{pmatrix}$$

Case 3: 3 X 3 matrix

Input

$$\begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}$$

Expected output

$$\begin{pmatrix} 14 & 8 & 2 \\ 16 & 10 & 4 \\ 18 & 12 & 6 \end{pmatrix}$$

(In a square matrix with an odd number of rows, the central element has the same value after rotation.)

Hints

- **Remember**: In this type of problem, where you are asked to write a `static` method that performs an operation using some input data, the parameters of the method are the **only** inputs the method should use. **There is no need to use a `Scanner` to get data from the user.** Another way of thinking about is that your method will be a sort of "black box": the only connection it has to the world is what it gets in its parameters, and what it returns. (Of course, your code can use the Java standard library to perform some portion of the processing; it simply doesn't need—and shouldn't look for—additional data from any source other than its parameters.)
- The body of your method should probably start by declaring and allocating space for an output array, of the same size as the input array.

- The body of your method should end with a `return` statement that returns the output array.
- Think about iterating over a 2-dimensional array using nested loops:

  ```
  for (int i = 0; i < data.length; i++) {
    for (int j = 0; j < data.length; j++) {
      // TODO Most of the work happens here.
    }
  }
  ```

  In the above snippet, at the line where the `// TODO` comment is located, `i` is the current row of the input `data`, and `j` is the current column of row `i`.

  Now, consider this question: If the value of the current element is `data[i][j]`, what are the row & column indices (in terms of `i`, `j`, and `data.length`) where that value should be placed in the output array? Once you answer that question, how do you *put* that value in the desired row & column of the output array?

Extra credit

In the basic task, you probably performed rotation by allocating an output array, and copying the elements from the input array to the rotated positions in the output array. For extra credit, you must write a method that modifies the contents of the input array *in-place*.

In-place implementation details

- Write a method with the following declaration:

  ```
  public void rotateInPlace(int[][] data)
  ```

  This method must perform the same task as the original (at least conceptually), but instead of returning a new, rotated matrix (note that this method has the `void` return type; that is, it does not return a value), it must rotate the input matrix in-place.
- Your implementation **must not** simply invoke the `rotate` method you wrote for the basic task and copy the results over the contents of the input `data`. Instead, it should do the rotation processing directly on the `data` array, without allocating (directly or indirectly) a new array.

Unit testing

There are 3 extra credit opportunities involve unit testing.

Base unit tests on in-place implementation

- Use test methods annotated with `@Test` to test the in-place implementation, with the same test cases given in "Base unit tests" (above).

Parameterized unit tests on basic implementation

- Use the `@ParameterizedTest` annotation to write a single test method that will be invoked multiple times—once for each test case. (See "Parameterized Tests" in the *JUnit5 User Guide* for details on parameterized tests.)
- Write the test cases in a CSV file (using the `@CsvFileSource` annotation on your test method), or as a `Stream<Arguments>` returned by a provider method (using the `@MethodSource` annotation on your test method).

Parameterized unit tests on in-place implementation

- Use the `@ParameterizedTest` annotation to write a single test method that will be invoked multiple times—once for each test case.
- Write the test cases in a CSV file (using the `@CsvFileSource` annotation on your test method), or as a `Stream<Arguments>` returned by a provider method (using the `@MethodSource` annotation on your test method).