# CSCI 2134 Lab 7: Requirements Testing

Winter 2023

## Objective

In this lab, you will review a specification for a class and ensure that the unit tests match the requirements. Working by yourself, or with a partner, you will identify what tests are missing or not matching the specification and either add tests or note where the specification can be improved.

## Preparation

1. Ensure that you have your Integrated Development Editor (IDE) installed.
2. Ensure that you are able to write and run a JUnit test case in the IDE. You will not have the time to both debug your IDE environment and complete the lab task within the lab time. The lab time should be able to concentrate on the lab task.
3. Clone the Lab 7 repo: https://git.cs.dal.ca/courses/2023-fall/csci-2134/lab7/????.git where ???? is your CSID.
4. Review the provided code listed in the Resources section below by reading the code of the class you will be fortifying (`Matrix.java`) and the unit tests (`MatrixTest.java`). These files are located in the **`src`** and **`test`** directories of the cloned project.

## Resources

- Documentation at the end of this document
- Code base to be reviewed for compliance is in the **`src`** directory of the Lab 7 repository.
- Test code is in the **`test`** directory of the Lab 7 repository.

## Procedure

### Set-up

1. Decide if you wish to work by yourself or pair up with another student
2. Open the project you created in preparation for this lab
3. **If you are working in pairs**, be sure that both of your names are listed in the FIX LIST at the top of `MatrixTest.java`

### Lab Steps

1. Review the *Matrix* class specification at the end of this document and the code in `MatrixTest.java`.
   - Select a method or constructor
   - Compare the specification of the method/constructor with the unit tests for it.
   - Identify a gap where the tests don't cover the specification, or the specification does not match the tests.
   - If no gap is found, select another method/constructor.

2. If there are tests missing for a method/constructor, add a unit test to `MatrixTest.java` to fix the issue. If the specification is ambiguous or missing information, proceed directly to Step 4.
3. Rerun the unit tests to ensure you did not introduce any problems. Fix them, if you did. ☺
4. Note the issue in the FIX LIST at the top of `MatrixTest.java` (see item 0 in the FIX LIST of that test file, including current time).
5. **Commit and push back to the repository.**
6. Repeat until all issues have been reviewed and fixed.

## Analysis and Reporting

1. Categorize (in the FIX LIST) all issues you find as either (a) testing issue, (b) specification issue, or (c) other. If the issue is (c) be sure to explain what the issue is.
2. **Commit and push the project.**

## Grading

The lab will be marked out of 4 points:

| Task | 2 Points | 1 Point | 0 Points |
|---|---|---|---|
| **Requirements Testing** | At least four (4) issues have been identified and fixed. | At least two (2) issues have been identified and fixed. | Less than two (2) issues identified and fixed. |
| **Analysis and Reporting** | At least four (4) issues are properly categorized as testing issue, requirements issue, or other. | At least one (2) issues are properly categorized as testing issue, requirements issue, or other. | Less than two (2) issues are properly categorized and fixed. |

# *Matrix* Class Specification

The `Matrix` class represents a matrix whose entries store values of type double. The class should provide the following public interface:

## Constructors

| Constructor | Description |
|---|---|
| `Matrix(int m, int n)` | Creates a matrix of size m × n and initializes all entries to 0. |
| `Matrix(Matrix mtx)` | Creates a copy of matrix mtx. |

## Methods

| Method | Description |
|---|---|
| `boolean equals(Matrix a)` | Return true if this matrix equals the one passed to this method. |
| `Matrix negate()` | Negate all entries in this matrix and return this matrix; |
| `Matrix add(Matrix b)` | Add matrix **b** to **this** matrix, and returns it. |
| `Matrix add(Matrix b, Matrix res)` | Add matrix **b** to **this** matrix, store the result in matrix **res**, and return it. |
| `Matrix multiplyWithScalar(double s)` | Multiply scalar **s** with matrix **this** and return it. |
| `Matrix multiplyWithScalar(double s, Matrix res)` | Multiply scalar **s** with matrix **this,** store the result in matrix **res**, and return it. |
| `Matrix multiplyWithMatrix(Matrix b, Matrix res)` | Multiply **this** matrix with matrix **b,** store the result in matrix **res** and return it. |
| `double getElem(int i, int j)` | Takes two integers, **i** and **j**, and return element E[i,j] of the matrix. |
| `void setElem(int i, int j, double v)` | Takes two integers, **i** and **j**, and value **v**, and sets element E[i,j] to value **v**. |
| `int getHeight()` | Returns the height of **this** matrix. |
| `int getWidth()` | Returns the width of **this** matrix. |

## Exceptions

| Exception | Description |
|---|---|
| `DimensionMismatchException` | Raised when the matrix dimensions do not match for a given operation. |
| `NullMatrixException` | Raised if a null matrix is passed. |
| `IndexOutOfBoundsException` | Raised if the indices for specified element are negative of exceed matrix dimensions. |

## Notes

- For the `add()` method, both parameters, **b** and **res**, must be of the same size as **this** matrix. If the sizes differ, or a **null** matrix is passed, an exception will be raised.
- For the `multiplyWithScalar()` method, parameter res must be of the same size as **this** matrix. If the sizes differ, or a **null** matrix is passed, an exception is raised.
- For the `multiplyWithMatrix()` method, both parameters, **b** and **res**, must be of appropriate size. I.e., the height of **b** must be equal to the width of **this** matrix, height of **res** must be equal to the height of **this** matrix, and the width of **res** must be equal to the width of **b**. If the sizes are not appropriate, or a null matrix is passed, an exception is raised. If the **res** parameter is the same as the **b** parameter or **this** matrix, the result of the multiplication is undefined.

# A Primer on Matrices

A matrix of size *m* × *n* is a 2D grid of values, typically decimal numbers, with *m* rows and *n* columns. Element *E[i,j]* of a matrix, refers to the value in row *i* and column *j*. Element *E[1,1]* is in the top left corner of the matrix and element *E[m,n]* is the bottom right corner of the matrix. A matrix can be negated, added with another matrix, multiplied by a scalar, or multiplied by another matrix.

Two matrices are equal if their elements-wise are the same. A matrix is negated by negating all the elements. E.g.,

$$\begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix} = -\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Two matrices can be added only if they have the same size. Addition is performed element-wise, meaning that if *C = A + B*, where *C*, *B*, and *A*, are matrices, then element *C[i, j] = A[i, j] + B[i, j]* for all elements of *C*. E.g.,

$$\begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Any matrix can be multiplied by a scalar, which is typically a decimal value. Scalar multiplication is performed element-wise, meaning that if *C = sA*, where s is a scalar value and A is a matrix, then *C[i,j] = s × A[i,j]*. E.g.,

$$\begin{bmatrix} -3 & -6 \\ -9 & -12 \end{bmatrix} = -3\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Two matrices can be multiplied only if the width of the first matrix is equal to the height of the second matrix. If *C = AB*, where *A* is a matrix of size *m×n*, *B* is a matrix of size *n×p*, and *C* is a matrix of size *m×p*, then *C[i, j] = A[i, 1] × B[1, j] + A[i, 2] × B[2, j] + A[i, 3] × B[3, j] + . . . + A[i, n] × B[n, j]* E.g.,

$$\begin{bmatrix} 9 & 12 & 15 \\ 19 & 26 & 33 \\ 29 & 40 & 51 \\ 39 & 54 & 69 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$