CS102 – Algorithms and Programming II
Programming Assignment 1
Spring 2026

LAB OBJECTIVES:

- The review of classes, objects, inheritance, and polymorphism.

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn1_YourSurname_YourName.zip
- Replace the variables “Sec1”, “YourSurname”, and “YourName” with your actual section, surname, and name.
- Upload the above zip file to Moodle by the deadline (if not, significant points will be taken off). You will have the opportunity to update and improve your solution by consulting with the TAs and tutors during the lab. You have to make the preliminary submission before the lab day. After the TA checks your work in the lab, you will make your final submission. Even if your code does not change in between these two versions, you should make two submissions for each assignment.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Vectors and Matrices

In this lab assignment, you will implement a console-based Java linear algebra calculator for performing operations on vectors and matrices. The user will enter an initial mathematical object (a Vector or Matrix) and then sequentially apply operations to it, such as addition or multiplication. You will define a shared interface named Algebraic for common algebraic operations and implement Vector and Matrix classes that implement this interface.

Algebraic Interface

The Algebraic interface defines a common set of algebraic operations that may be performed on mathematical objects in this lab. Any class that implements this interface represents an object that can participate in basic algebraic computations.

The interface declares the following methods:

Algebraic negate(): Returns the negation (additive inverse) of this object.
Algebraic add(Algebraic other): Returns the result of adding this object to other, or null if the operation is not defined or the objects are incompatible.
Algebraic subtract(Algebraic other): Returns the result of subtracting other from this object, or null if the operation is not defined or the objects are incompatible.
Algebraic multiply(Algebraic other): Returns the result of multiplying this object by the other object. Depending on the types, this represents either the dot product (for vectors) or matrix multiplication (for matrices). Returns null if the operation is undefined or if the object dimensions are incompatible.

Not all operations are valid for all combinations of algebraic objects. Implementing classes are responsible for determining whether an operation is mathematically valid and for returning null when it is not.

Vector Class

You will implement a `Vector` class that represents a mathematical column vector whose elements are of type `float`. The class must include a constructor that initializes the vector from a float array. The class will implement the `Algebraic` interface and support the following operations:

- **Vector(float[] vec):** Constructs a new `Vector` from a given array of floats. The array is copied, so modifying the original array does not affect the vector.
- **negate():** Returns a new `Vector` with all elements negated.
- **add(Algebraic other):** Returns a new `Vector` whose elements are the sum of this vector and other. Returns `null` if the vectors have different lengths or other object is not a vector.
- **subtract(Algebraic other):** Returns a new Vector whose elements are the difference between this vector and other. Returns `null` if the vectors have different lengths or other object is not a vector.
- **multiply(Algebraic other):** Returns a new one-dimensional `Vector` containing the dot product of this vector and other. Returns null if the vectors have different lengths or if the other object is not a Vector.
- **crossproduct(Vector other):** Returns a new `Vector` representing the cross product of this vector and other. Only defined for 3-dimensional vectors; returns `null` otherwise.
- **equals(Object other):** Returns true if other is a `Vector` of the same length with all corresponding elements equal within a tolerance of 10^{-6}.
- **toString():** Returns a nicely formatted string representing the vector in square brackets, with each value shown to two decimal places.

All operations return their results without modifying the original vector. You can use the following code to test your `Vector` class:

```
final Vector v1 = new Vector(new float[]{3.f, 2.f, 5.f});
final Vector v2 = new Vector(new float[]{1.32f, 3.15f, 1.5f});
final Vector v3 = new Vector(new float[]{1.32f, 3.15f, 1.5f, 32.f});

System.out.printf("v1 \n%s\n\n", v1);
System.out.printf("v2 \n%s\n\n", v2);

System.out.println();
System.out.printf("-v1 \n%s\n\n", v1.negate());
System.out.printf("v1 + v2 \n%s\n\n", v1.add(v2));
System.out.printf("v2 + v1 \n%s\n\n", v2.add(v1));
System.out.printf("v1 - v2 \n%s\n\n", v1.subtract(v2));
System.out.printf("v2 - v1 \n%s\n\n", v2.subtract(v1));

System.out.println();
System.out.printf("v1 == null         => %s\n", v1.equals(null));
System.out.printf("v1 == v3           => %s\n", v1.equals(v3));
System.out.printf("v1 == -v1          => %s\n", v1.equals(v1.negate()));
System.out.printf("v1 == -(-v1)       => %s\n",
v1.equals(v1.negate().negate()));
System.out.printf("v1 + v2 == v2 + v1 => %s\n",
(v1.add(v2)).equals(v2.add(v1)));

System.out.println();
System.out.printf("v1 * v2 = %s\n", v1.multiply(v2));
System.out.printf("v1 * v3 = %s\n", v1.multiply(v3));
System.out.printf("v1 x v3 = %s\n", v1.crossproduct(v3));
System.out.printf("v1 x v2 \n%s\n", v1.crossproduct(v2));
```

Output:

```
v1
|3.00|
|2.00|
|5.00|

v2
|1.32|
|3.15|
|1.50|
```

```
-v1
|-3.00|
|-2.00|
|-5.00|

v1 + v2
|4.32|
|5.15|
|6.50|

v2 + v1
|4.32|
|5.15|
|6.50|

v1 - v2
| 1.68|
|-1.15|
| 3.50|

v2 - v1
|-1.68|
| 1.15|
|-3.50|


v1 == null         => false
v1 == v3           => false
v1 == -v1          => false
v1 == -(-v1)       => true
v1 + v2 == v2 + v1 => true

v1 * v2 = |17.76|
v1 * v3 = null
v1 x v3 = null
v1 x v2
|-12.75|
|  2.10|
|  6.81|
```

Matrix Class

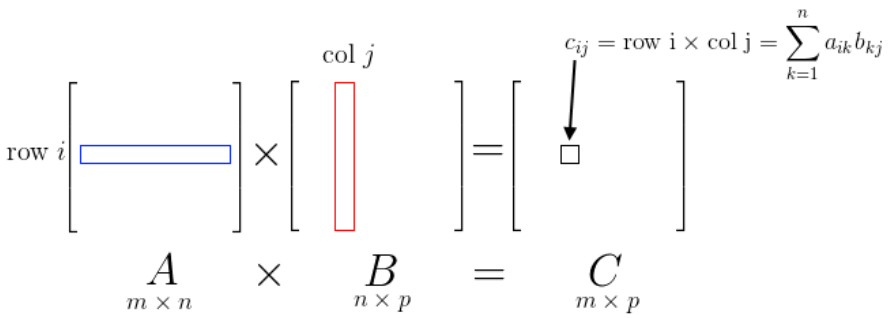
You will implement a `Matrix` class that represents a mathematical matrix whose elements are of type float. The class will implement the `Algebraic` interface. A matrix is stored internally as a two-dimensional array of `floats`, where each row represents a row of the matrix.

The Matrix class supports the following operations:

- **Matrix(float[][] mat):** Constructs a new Matrix from a given two-dimensional float array. The input array is copied.
- **negate()**: Similar to the `Vector` class, this method returns a new Matrix with all elements negated. The original matrix must not be modified. Returns `null` if the matrix is invalid.
- **add(Algebraic other):** Returns the elementwise sum of this matrix and other. The matrices must have the same dimensions. Returns `null` if other is not a `Matrix` or if the matrices have different dimensions.
- **subtract(Algebraic other):** Returns the elementwise difference between this matrix and other. The matrices must have the same dimensions. Returns `null` if other is not a `Matrix` or if the matrices have different dimensions.
- **multiply(Algebraic other):** Returns the product of this matrix and the other object.
    - If other is a `Matrix`: Performs matrix multiplication (see definition below) and returns a new `Matrix`.
    - If other is a `Vector`: Performs matrix-vector multiplication (applying the matrix transformation to the vector) and returns a new `Vector`.
    - Returns `null` if the dimensions are incompatible for the specific operation.
- **determinant()**: Returns the determinant of the matrix. This operation is only defined for 2x2 or 3x3 square matrices. Returns a 1D `Vector` containing the result. If the matrix is not 2x2 or 3x3, return `null`.
- **equals(Object other):** Compares this matrix with another object for equality. Two matrices are equal if they have the same dimensions and all of their corresponding elements are equal within a tolerance of 10^{-6}.
- **toString():** Returns a formatted string representation of the matrix.

Matrix multiplication is defined as follows: **C = A x B**. Let's say **A** is of size **m x n** and **B** is of size **n x p**. Then, the resulting matrix **C** is of size **m x p**. Two matrices can be multiplied if the number of columns of the first matrix is equal to the number of rows in the second matrix. Otherwise, the matrices cannot be multiplied. The entries of the resulting matrix **C** are calculated as follows:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$$



As with the `Vector` class, all matrix operations return their results without modifying the original matrix. The following code can be used to test your `Matrix` class:

```
final Matrix m1 = new Matrix(new float[][]{{1, 3.2f, 0}, {0, 2, 1}, {0, 2.16f,
1}});
final Matrix m2 = new Matrix(new float[][]{{0, 1.f, 0}, {0, 1, 0}, {3.02f, 0, 1}});
final Matrix m3 = new Matrix(new float[][]{{1.f, 3.f}, {1.f, 4.f}, {5.12f,
2.31f}});
final Vector v = new Vector(new float[]{1, 2, 3});
System.out.printf("m1 \n%s\n\n", m1);
System.out.printf("m2 \n%s\n\n", m2);
System.out.printf("m3 \n%s\n\n", m3);
System.out.printf("v \n%s\n\n", v);

// Arithmetic operations
System.out.println();
System.out.printf("-m1\n%s\n\n", m1.negate());
System.out.printf("m1 + m2 \n%s\n\n", m1.add(m2));
System.out.printf("m2 + m3 %s\n\n", m2.add(m3));
System.out.printf("m1 - m2 \n%s\n\n", m1.subtract(m2));
System.out.printf("m1 * m2 \n%s\n\n", m1.multiply(m2));
System.out.printf("m2 * m1 \n%s\n\n", m2.multiply(m1));
System.out.printf("m1 * m3 \n%s\n\n", m1.multiply(m3));
System.out.printf("m3 * m1 \n%s\n\n", m3.multiply(m1));

// Matrix-Vector operations
System.out.println();
System.out.printf("m1 + v \n%s\n\n", m1.add(v));
System.out.printf("v * m1 \n%s\n\n", v.multiply(m1));
System.out.printf("m3 * v \n%s\n\n", m3.multiply(v));
System.out.printf("m1 * v \n%s\n\n", m1.multiply(v));

// Determinant
System.out.println();
```

```
System.out.printf("|m1| = %s\n", m1.determinant());
System.out.printf("|m3| = %s\n", m3.determinant());

// Equality
System.out.println();
System.out.printf("m1 == null => %s\n", m1.equals(null));
System.out.printf("m1 == m2 => %s\n", m1.equals(m2));
System.out.printf("m1 == m3 => %s\n", m1.equals(m3));
System.out.printf("m1 == -(-m1) => %s\n", m1.equals(m1.negate().negate()));

final Algebraic m13 = m1.multiply(m3);
final Algebraic m23 = m2.multiply(m3);
final Algebraic m123 = m1.add(m2).multiply(m3);
System.out.printf("(m1 + m2) * m3 == (m1 * m3) + (m2 * m3) => %s\n",
    m123.equals(m13.add(m23)));
```

Output:

```
m1
|1.00 3.20 0.00|
|0.00 2.00 1.00|
|0.00 2.16 1.00|

m2
|0.00 1.00 0.00|
|0.00 1.00 0.00|
|3.02 0.00 1.00|

m3
|1.00 3.00|
|1.00 4.00|
|5.12 2.31|

v
|1.00|
|2.00|
|3.00|


-m1
|-1.00 -3.20 -0.00|
|-0.00 -2.00 -1.00|
|-0.00 -2.16 -1.00|

m1 + m2
|1.00 4.20 0.00|
|0.00 3.00 1.00|
```

```
|3.02 2.16 2.00|

m2 + m3 null

m1 - m2
| 1.00  2.20  0.00|
| 0.00  1.00  1.00|
|-3.02  2.16  0.00|

m1 * m2
|0.00 4.20 0.00|
|3.02 2.00 1.00|
|3.02 2.16 1.00|

m2 * m1
| 0.00  2.00  1.00|
| 0.00  2.00  1.00|
| 3.02 11.82  1.00|

m1 * m3
| 4.20 15.80|
| 7.12 10.31|
| 7.28 10.95|

m3 * m1
null


m1 + v
null

v * m1
null

m3 * v
null

m1 * v
|7.40|
|7.00|
|7.32|


|m1| = |-0.16|
|m3| = null
```

```
m1 == null => false
m1 == m2 => false
m1 == m3 => false
m1 == -(-m1) => true
(m1 + m2) * m3 == (m1 * m3) + (m2 * m3) => true
```

LTMatrix Class

Derive a new `LTMatrix`, which is a square, lower triangular matrix, by extending the `Matrix` class using **inheritance**. It has zero entries over the diagonal. An example 5 x 5 lower triangular matrix (LTM) is given below:

$$LTM = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 2 & 7 & 0 & 0 & 0 \\ 4 & 6 & 4 & 0 & 0 \\ 1 & 9 & 1 & 8 & 0 \\ 5 & 6 & 3 & 4 & 2 \end{bmatrix}$$

The constructor **LTMatrix(float[][] mat)** constructs a new lower triangular matrix from the given two-dimensional float array. The input array must represent a square matrix with all entries above the main diagonal equal to zero. If the input array does not represent a valid lower triangular matrix, print an appropriate error message and construct a 0x0 matrix.

Since an `LTMatrix` has zero entries above the main diagonal, its matrix operations can be implemented more efficiently. You must avoid iterating over the zero elements by limiting your loops to the lower triangular region. Override the following `LTMatrix` methods and implement them efficiently:

- **negate():** Returns a new `LTMatrix` with all lower triangular entries negated. This method should be implemented efficiently by processing only the lower triangular entries.
- **add(Algebraic other):** Returns a new `Matrix` whose entries are the elementwise sum of this matrix and other. If other is an `LTMatrix`, this method should perform the addition efficiently and return a new `LTMatrix`; otherwise, it should perform regular matrix addition and return a new `Matrix`. Similar to matrix addition, this method should return `null` if the matrices have different dimensions or other is not a `Matrix`.
- **subtract(Algebraic other):** Returns a new Matrix whose entries are the elementwise difference of this matrix and other. If other is an `LTMatrix`, this method should perform the subtraction efficiently and return a new `LTMatrix`; otherwise, it should perform regular matrix subtraction and return a new `Matrix`. Similar to matrix subtraction, this

method should return `null` if the matrices have different dimensions or other is not a `Matrix`.

- **multiply(Algebraic other):** Returns the product of this matrix and the other object.
    - If other is a `LTMatrix`: Performs efficient matrix multiplication and returns a new `LTMatrix`.
    - If other is a `Matrix`: Performs matrix multiplication and returns a new `Matrix`.
    - If other is a `Vector`: Performs matrix-vector multiplication (applying the matrix transformation to the vector) and returns a new `Vector`.
    - Returns `null` if the dimensions are incompatible for the specific operation.
- **determinant():** Returns the determinant of the matrix.
- **equals(Object other):** Compares this matrix with another object for equality. Similar to the `Matrix` class, two matrices are considered equal if they have the same dimensions and all corresponding elements are equal within a tolerance of 10^{-6}. If the other is an `LTMatrix`, only the lower triangular entries need to be compared. If the other is a `Matrix`, all matrix entries must be compared.

Linear Algebra Calculator

Implement a console-based Linear Algebra Calculator using the `Vector`, `Matrix`, and `LTMatrix` classes developed in this assignment.

At the start of the program, the user should be prompted to enter an initial `Algebraic` object, which may be a `Vector`, a `Matrix`, or an `LTMatrix`. The program should then repeatedly display a menu of supported operations and prompt the user to choose an operation to perform on the current object.

If the selected operation requires additional input (e.g., another vector, matrix, or lower-triangular matrix), the program should prompt the user to enter the required data. After the operation is performed, the resulting `Algebraic` object should become the new current object. This process should continue until the user selects the exit option. The program should handle invalid operations gracefully by displaying appropriate error messages.

The menu options are as follows:

1. Negate
2. Add
3. Subtract
4. Multiply
5. Cross Product or Determinant
6. Compare for Equality
7. Exit

Note: Your implementation should reuse code as much as possible. Avoid duplication between `Vector`, `Matrix`, and `LTMatrix` operations; use polymorphism where appropriate. You are encouraged to add extra helper methods, getters, or setters as needed.

Example runs of the program are given below:

Example Run 1:

```
Enter a vector or matrix:
Enter number of rows and columns (n x m): 1 3
Enter vector elements separated by spaces: 3.32 2.45 1.23
|3.32|
|2.45|
|1.23|

Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 1
  |3.32|   |-3.32|
- |2.45| = |-2.45|
  |1.23|   |-1.23|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 2
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 2 3
Enter matrix elements separated by spaces:
2.35 0.21 0
2.0  2.13 5.63
Invalid operation
```

```
Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 2
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 1 3
Enter vector elements separated by spaces: 2.35 0.21 0

|-3.32|   |2.35|   |-0.97|
|-2.45| + |0.21| = |-2.24|
|-1.23|   |0.00|   |-1.23|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 1
  |-0.97|   |0.97|
- |-2.24| = |2.24|
  |-1.23|   |1.23|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 5
Enter the second vector
Enter number of rows and columns (n x m): 1 3
Enter vector elements separated by spaces: 2 1 4
```

```
|0.97|   |2.00|   | 7.73|
|2.24| x |1.00| = |-1.42|
|1.23|   |4.00|   |-3.51|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 7
Exiting...
```

Example Run 2:

```
Enter a vector or matrix:
Enter number of rows and columns (n x m): 3 4
Enter matrix elements separated by spaces:
3.32 2.45 1.23
2.35 0.21 0
2.0  2.13 5.63
3.2 3 0
|3.32 2.45 1.23 2.35|
|0.21 0.00 2.00 2.13|
|5.63 3.20 3.00 0.00|

Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 3
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 3 1
Enter matrix elements separated by spaces:
3.24
3.432
1.23
Invalid operation
```

```
Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 3
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 3 4
Enter matrix elements separated by spaces:
23.12 64.2 53.2
2.23 67.2 41.32
3.54 84.2 87.1
52 42 16

|3.32 2.45 1.23 2.35|   |23.12 64.20 53.20  2.23|   |-19.80 -61.75 -51.97   0.12|
|0.21 0.00 2.00 2.13| - |67.20 41.32  3.54 84.20| = |-66.99 -41.32  -1.54 -82.07|
|5.63 3.20 3.00 0.00|   |87.10 52.00 42.00 16.00|   |-81.47 -48.80 -39.00 -16.00|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 1
  |-19.80 -61.75 -51.97   0.12|   |19.80 61.75 51.97 -0.12|
- |-66.99 -41.32  -1.54 -82.07| = |66.99 41.32  1.54 82.07|
  |-81.47 -48.80 -39.00 -16.00|   |81.47 48.80 39.00 16.00|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 4
```

```
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 4 3
Enter matrix elements separated by spaces:
0 1 3 2
1 3 2 1
3 5 2 0

|19.80 61.75 51.97 -0.12|   |0.00 1.00 3.00|   |226.84 133.28 400.56|
|66.99 41.32  1.54 82.07| * |2.00 1.00 3.00| = |496.07 273.99 329.55|
|81.47 48.80 39.00 16.00|   |2.00 1.00 3.00|   |255.60 201.27 507.81|
                            |5.00 2.00 0.00|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 5
|6109048.00|

Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 6
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 1 2
Enter vector elements separated by spaces: 6109048.00 0
|6109048.00| == |6109048.00| ==> false
                |      0.00|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
```

```
5: Cross Product
6: Compare
7: Exit
Enter your choice: 6
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 1 1
Enter vector elements separated by spaces: 6109048
|6109048.00| == |6109048.00| ==> true


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Cross Product
6: Compare
7: Exit
Enter your choice: 7
Exiting...
```

Example Run 3:

```
Enter a vector or matrix:
Enter number of rows and columns (n x m): 2 2
Enter matrix elements separated by spaces:
2 0
3 1
Constructed the LTMatrix
|2.00 0.00|
|3.00 1.00|

Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 2
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 2 2
Enter matrix elements separated by spaces:
1 0
0 0
```

```
Constructed the LTMatrix

|2.00 0.00| + |1.00 0.00| = |3.00 0.00|
|3.00 1.00|   |0.00 0.00|   |3.00 1.00|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 2
Enter the second vector or matrix:
Enter number of rows and columns (n x m): 2 2
Enter matrix elements separated by spaces:
1 3
5 7

|3.00 0.00| + |1.00 3.00| = |4.00 3.00|
|3.00 1.00|   |5.00 7.00|   |8.00 8.00|


Select an operation:
1: Negate
2: Add
3: Subtract
4: Multiply
5: Determinant
6: Compare
7: Exit
Enter your choice: 7
Exiting...
```

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- Implementation of the `Vector` and `Matrix`.

<u>Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.</u>