Matrices (Assignment 3)

[30 Marks, D/L: 28/10/21]

There are ~15 types of Matrices:

1. Rectangular Matrix
2. Row Matrix
3. Column Matrix
4. Square Matrix
5. Symmetric Matrix
6. Skew-symmetric Matrix
7. Upper-triangular Matrix
8. Lower-triangular Matrix
9. Singular Matrix
10. Diagonal Matrix
11. Scalar Matrix
12. Identity Matrix
13. Singleton Matrix
14. Ones Matrix
15. Null Matrix

While above **matrix-types** have certain **properties/methods** that make them **separate** from others, they also have certain **properties/methods** that make them **similar** to others. For example, both square and rectangular matrices can be transposed, but you can compute determinants for only square matrices. Likewise, not all square matrices are singular. Singular matrices have a special property/method that makes them separate from the rest. However, that doesn't mean they are not square matrices. These special properties/methods can make a program dealing with matrices **efficient** through OOP principles.

Some hints on **efficiency**:

- When a matrix is already identified as singular, you cannot use it as a divisor. So, throw an error immediately.
- There is no point in trying to multiply a matrix with a null matrix: you can simply make all its elements zero.
- Determinants of a diagonal matrices can be very easily computed.
- You can store diagonal & null/ones matrices in a compressed manner.

Write a program to deal with all the types of matrices, strictly adhering to principles of **OOP**. Your program should **efficiently** do the following tasks:

1. Take matrices as input and label them with appropriate matrix-types.
2. Create matrices of requested matrix-types and label them with appropriate matrix-types.
3. Change the elements of a matrix as long as the fixed matrix-type labels remain valid.
4. Display all the matrix-type labels of a requested matrix.

5. Perform addition, subtraction, multiplication & division.
6. Perform element-wise operations.
7. Transpose matrices.
8. Inverse matrices.
9. Compute means: row-wise mean, column-wise mean, mean of all the elements.
10. Compute determinants.
11. Use singleton matrices as scalars, if requested.
12. Compute $A+A^T$ for a matrix A.
13. Compute Eigen vectors and values.
14. Solve sets of linear equations using matrices.
15. Retrieve all the existing matrices (entered or created) having requested matrix-type labels.

CHALLENGE: A matrix can be saved in memory as an object of only one class (matrix-type).

HINT: All the appropriate matrix-type labels are available (from your inheritance tree) before the instantiation/initialization of a matrix (object). Considering that the to-be-created object needs to satisfy all of them, choose the optimal one that gives you the best memory efficiency. Use temporary variables wherever necessary.

NOTE:

- A matrix must have an Id for identification/retrieval purposes.
- Matrix size can vary from **1x1 to 3x3**.
- All the implementations must be done from **scratch**.
- You must report how principles of **OOP** helped you program this assignment **efficiently**.
- Each of the above tasks carry **2 marks**: 1[successful-run] +1[efficiency/OOP].