**OOP Ezejiofor Emmanuel: Documentation,assignment 1.**

**Ezejiofor Emmanuel 1. assignment/1.task 15th March 2024**

**NEPTUN CODE: H2NS25 Group: 5**

**Task**

Implement the chessboard matrix type which contains integers. In these matrices, every second entry is zero. The entries that can be nonzero are located like the same colored squares on a chessboard, with indices (1, 1), (1, 3), (1, 5), ..., (2, 2), (2, 4), .... The zero entries are on the indices (1, 2), (1, 4), .., (2, 1), (2, 3), ... Store only the entries that can be nonzero in row-major order in a sequence. Don't store the zero entries. Implement as methods: getting the entry located at index (i, j), adding and multiplying two matrices, and printing the matrix (in a shape of m by n).

**Chessboard matrix type**

ChessBMatrix (n) = { a ∈ ℤ n×n ⎪ ∀i,j∈[1..n]: i+j mod 2 = 0 → a[i,j]=0 }

**Operations**

1. *Getting the entry at index (i, j)*

*Gets the element at the given index from the matrix.*

**Formally:** A =( a : ChessBMatrix (n), i : ℤ, j : ℤ, e : ℤ )

Pre = (a = a` ∧ i=i’ ∧ j=j’ ∧ i,j∈[1..n])

Post = (Pre ∧ e=a[i,j])

This Operation will output the non-zero value only if sum of i and j is even, otherwise the output is zero.

1. *Addition of two matrices*

*Adds two same-size matrices.*

**Formally:** A = (a : ChessBMatrix (n), b : ChessBMatrix (n), c : ChessBMatrix (a) )

Pre = ( a = a` ∧ b = b`)

Post = ( Pre ∧ ∀i,j∈[1..n]: c[i,j] += b[i,j] )

This Operation will sum two same size matrices and generate a new matrix with a result.

1. *Multiplication of two matrices*

*Multiplies two same-size matrices.*

**Formally:** A = a : ChessBMatrix (n), b ChessBMatrix (n), c : ChessBMatrix (n)

Pre = ( a = a` ∧ b = b`)

Post = ( Pre ∧ ∀i,j∈[1..n]: c[i,j] = Σk=1..n a[i,k] \* b[k,j])

This Operation will multiply two same size matrices and generate a new matrix with a result.

1. ***Printing the matrix (in a shape of m by n)***

*ToString method provides the string of a matrix in a shape of m by n.*

**Formally:** A = a : ChessBMatrix (n), s : S

Pre = ( a = a` )

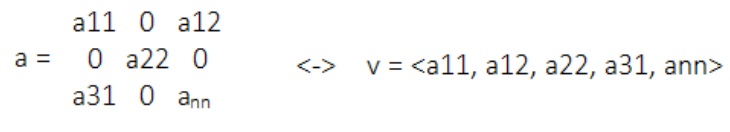
Post = ∀i,j∈[1..n] : s = ⊕ a[i,j] + “newLine”

where newLine = “\n”

This operation will go through all the values and concatenate them to the result string. Once a row ends, a new line is added to represent the rows and columns.

**Representation**

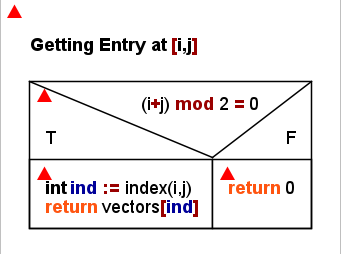
The chess board matrix is represented by a vector of integer values.



**Implementation**

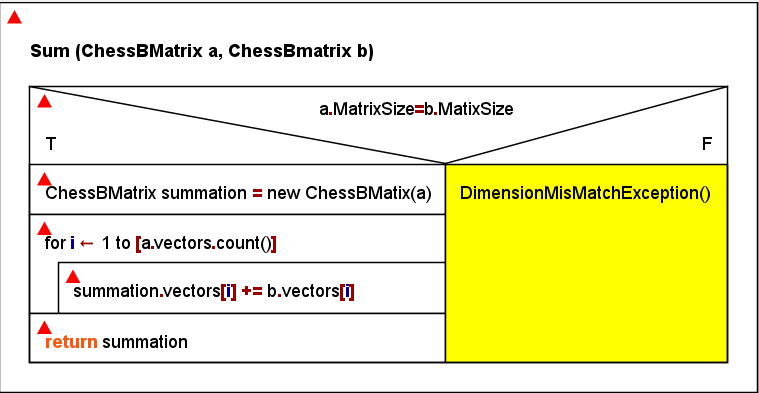
1. *Getting the entry at index (i, j)*

Getting the entry of the ith column and jth row (i,j∈[1..n]) e := a[i,j] where the matrix is represented by vectors, 1≤i≤n, and n stands for the size of the matrix can be implemented as:



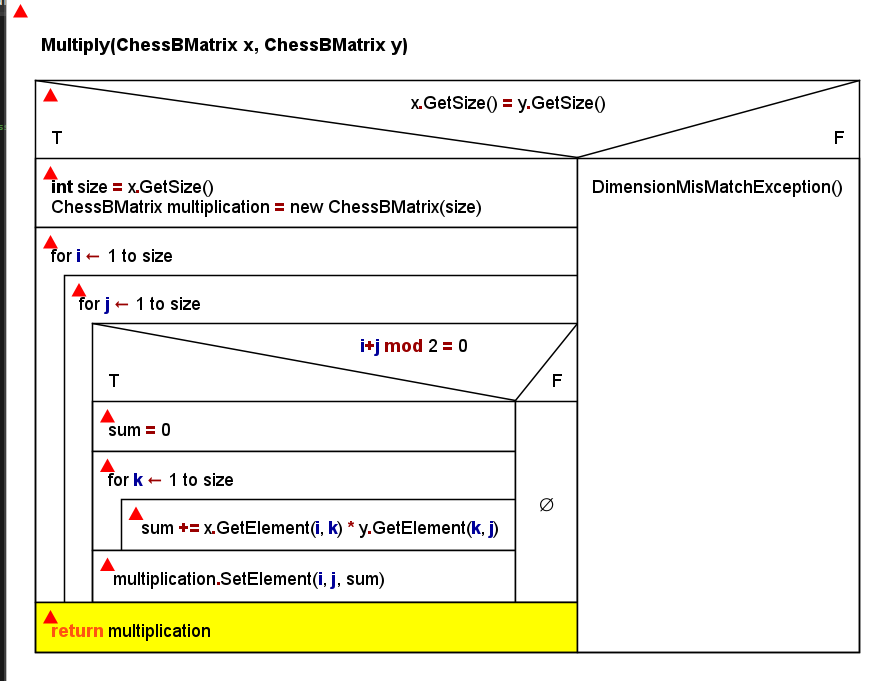
1. *Addition of two matrices*

The sum of matrices a and b (represented by lists a and b) goes to matrix c (represented by list summation), where all the lists have to have the same size.



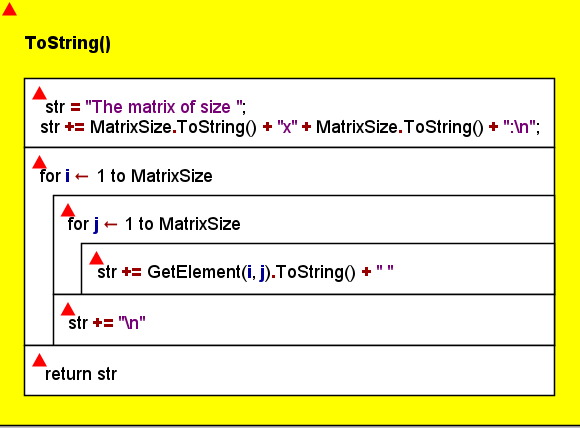
1. *Multiplication of two matrices*

The product of matrices a and b (represented by lists x and y) goes to matrix c (represented by list multiplication), where all the lists have to have the same size.

****

1. *Printing the matrix (in a shape of m by n)*

The method will return the string in the format of m by n with the proper paddings.

****

**Testing**

**Testing the operations (black box testing)**

1. Constructor:
2. Empty constructor. => Size of a matrix is 3. There are 5 values from 1 to 5 in the list of the matrix.
3. Constructor with an even integer as an argument. 6=> Size of the matrix is 6. Number of values is counted with a formula for even size matrices which is 18 in the current case
4. Constructor with an odd integer as an argument. 5 => Size of the matrix is 5. Number of values is counted with a formula for odd size matrices which is 13 in the current case.
5. Constructor with a list of values (number of which is enough for a full matrix). { 5, 10, 15, 20, 25 } => Size of a matrix is 3. Number of values is 5
6. Constructor with a list of values (number of which is not enough for a full matrix). { 5, 10, 15, } => Exception InvalidVectorException because there are not enough values in the list to create a full matrix.
7. Copy constructor:

Constructor with another matrix as an argument. new ChessBMatrix () => Size of the matrix is 3. Number of values is 5. If we change the entry in the matrix-argument, it won’t change in the new matrix because it’s not a reference but a copy.

3) Get element:

a) Matrix with empty constructor:

• GetElement(0, 0) – very first value => 1.

• GetElement(2, 2) – very last value not equal to => 2.

• GetElement(1, 2) – zero value cell => 0.

• SetElement(1, 1, 40); GetElement(1, 1) – after interaction with SetElement() => 40.

• GetElement(7, 7) – outside of the matrix, indices exceed the actual size => Exception InvalidIndexException.

4) Set element:

a) Matrix with empty constructor:

• SetElement(0, 0, 70); GetElement(0, 0) – very first value => 70.

• SetElement(0, 2, 60); GetElement(0, 2) – very last value => 60.

• SetElement(7, 7, 2) – outside of the matrix, indices exceed the actual size => Exception InvalidIndexException.

• SetElement(0, 1, 2) – zero value cell => Exception ZeroCellException.

5) Sum of matrices:

a) Sum of matrix with empty constructor and matrix with 3 as a parameter in the constructor:

• GetElement(0, 0) – 1 + 1 => 2.

• GetElement(1, 1) – 3 + 1 => 4.

• GetElement(0, 1) – 0 + 0 => 0. (zero entry)

b) Sum of matrix with empty constructor which in this case is 3 and matrix with 4 as a parameter in the constructor => Exception DimensionMismatchException.

6) Product of matrices:

a) Product of matrix with empty constructor and matrix with 3 as a parameter in the constructor:

• GetElement(0, 0) – 1\*1 + 0\*0 + 1\*2 => 3.

• GetElement(1, 1) – 0\*0 + 3\*1 + 0\*0 => 3.

• GetElement(0, 1) – 0\*1 + 3\*0 + 0\*1 => 0.

b) Product of matrix with empty constructor and matrix with 4 as a parameter in the constructor => Exception DimensionMismatchException.

**Testing based on the code (white box testing)**

1. Creating an extreme-size matrix (-1, 0, 1, 1000). It can not be a negative value or less than one but can be from 1 to 1000.

2. Generating and catching exceptions.