Name: Harley Phung
Class: CSDS 293
Assignment 11

Designing Sparse Matrix

1. Requirements

- Build extensions of Ring, support SparseMatrix (mostly zero matrix / large size identity Matrix)
- Coexist with MatrixMap. Omits zero from MatrixMap so that matrix converts few entries
- SparseMatrix in the same package as Matrixmap. MatrixMap can be converted to SparseMatrix and vice versa.
- Square MatrixMap and SparseMatrix form a ring
- Design:
    - Create design document that describes architectural decisions
    - Sketches and Diagrams. Contains classes or interfaces
    - Describe usages of those classes / interfaces
    - Include approach for error handling

2. Diagram

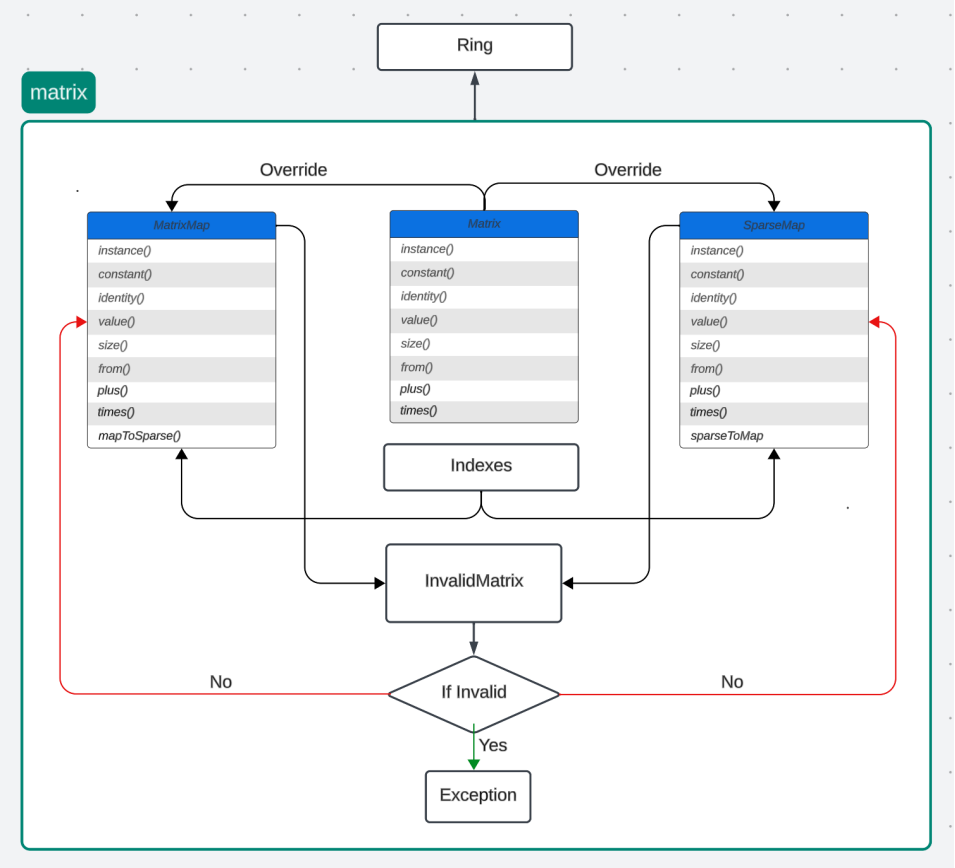

Figure 1: Design Diagram

3. Class Design

a. *Ring*:

- Reused from previous Assignments.
- Contains IntegerRing, BigIntegerRing, DoubleRing, PolynomialRings, Polynomial

b. *matrix:*

- This is the package, containing Indexes, MatrixMap, Matrix, SparseMatrix

i. *MatrixMap*:

- Reused from Assignment 5 (Definitely not from Assignment 10).
- MatrixMap overrides methods from interface Matrix. Contains instance(), constant(), identity(), value(), size(), from(), plus(), times(), mapToSparse().
- MatrixMap is immutable (need to consider clone() for each method, all fields are final)
- Make sure MatrixMap works well with Error Handling.

ii. *Matrix:*

- Interface. This interface lists the methods used in SparseMatrix and MatrixMap.
- Contains instance(), constant(), identity(), value(), size(), from(), plus(), times().

iii. *SparseMatrix:*

- Public class. Defines the sparse matrix that mostly contains zero or large size identity matrix. Overrides methods from interface Matrix. Contains instance(), constant(), identity(), value(), size(), from(), plus(), times(), sparseToMap().
- SparseMatrix is immutable (need to consider clone() for each method, all fields are final)
- Make sure SparseMatrix works well with Error Handling.

iv. *Indexes:*

- Reused from Assignment 5.

c. Notes for Immutable Class (MatrixMap, SparseMatrix)

- No methods that modifies object's state
- Class cannot be extended
- All fields are private final
- Ensure exclusive access to any mutable components

4. Method Design

a. Class MatrixMap

- *Method: mapToSparse().*
  - Name: mapToSparse(MatrixMap matrix):
  - Input: MatrixMap
  - Output: SparseMatrix
  - Pseudo Code:

```
mapToSparse(MatrixMap matrix):
        checkException
        Rows <- matrix.row()
        Col <- matrix.col()
        Indexes <- new array if Indexes
        For i to rows-1:
                For j to cols-1:
                        Value <- get value from map at matrix map of index (i,j)
                        If value not 0:
                                Add (value) to indexes
                        end condition
        End loop
        Return new SparseMatrix(indexes)
End function
```

- Method: times().
  - Name: times()
  - Input: SparseMatrix, Ring
  - Output: MatrixMap
  - Pseudo code:

```
times(SparseMatrix sparse, Ring ring):
        checkException
        Rows <- matrix.row()
        Col <- matrix.col()
        Indexes <- new array if Indexes
        For i to rows-1:
                For j to cols-1:
                        if (value of sparse = null) store 0
                        else continue like times(MatrixMap, Ring)
                End loop
        Return new MatrixMap(indexes)
End function
```

- Method: plus()
    - Name: plus()
    - Input: SparseMatrix, BinaryOperator
    - Output: MatrixMap
    - Pseudo code:

```
plus(SparseMatrix sparse, BinaryOperator plus):
    Rows <- matrix.row()
    Col <- matrix.col()
    Indexes <- new array if Indexes
    For i to rows-1:
        For j to cols-1:
            if (value of sparse = null) store value of current matrix map's
                                                                    index
            else continue like times(MatrixMap, Ring)
        End loop
    Return new MatrixMap(indexes)
End function
```

b. Class SparseMatrix

- Method: sparseToMatrix()
    - Name: sparseToMap(SparseMatrix sparse)
    - Input: SparseMatrix
    - Output: MatrixMap
    - Pseudo Code:

```
sparseToMap (SparseMatrix sparse):
    checkException
    Rows <- sparse.row()
    Col <- sparse.col()
    Indexes <- new array if Indexes
    For i to rows-1: //1
        For j to cols-1: //2
            Value <- get value from map at sparse of index (i,j)
            If value not null: //3
                Add (value) to indexes
            Else add(0) to indexes
            end condition
    End loop
    Return new MatrixMap(indexes)
End function
```

- Method: times().
    - Name: times()
    - Input: MatrixMap, Ring

- Output: SparseMatrix
- Pseudo code:

```
times(MatrixMap matrix, Ring ring):
        checkException
        Rows <- matrix.row()
        Col <- matrix.col()
        Indexes <- new array if Indexes
        For i to rows-1:
                For j to cols-1:
                        if (value of sparse = null) store 0
                        else continue
                End loop
        Return new SparseMatrix(indexes)
End function
```

- Method: plus()
  - Name: plus()
  - Input: MatrixMap, BinaryOperator
  - Output: SparseMatrix
  - Pseudo code:

```
plus(MatrixMap matrix, BinaryOperator plus):
        Rows <- matrix.row()
        Col <- matrix.col()
        Indexes <- new array if Indexes
        For i to rows-1:
                For j to cols-1:
                        if (value of sparse = null) store value of current matrix map's
                                                                                index
                        else continue
                End loop
        Return new SparseMatrix(indexes)
End function
```

5. Error Handling

a. mapToSparse() condition:

Identifier	Goal	Note	Condition
CC1	Code Coverage		If matrix not null
CC2	Code Coverage		If matrix is exception
BR1	Branch Coverage	1	In range i = 0 to rows - 1
BR2	Branch Coverage	2	In range j = 0 to cols - 1
BR3	Branch Coverage	2, 3 true	In range j = 0 to cols -1 and value not 0 then add to index
BR4	Branch Coverage	2,3 false	In range j = 0 to cols - 1 and value is 0 then continue loop
BR5	Branch Coverage	1	Out of range i < 0 and i >= rows - 1. End loop
BR6	Branch Coverage	2	Out of range j < 0 and j >= cols - 1. Continue 1.
B1	Boundary	1,2	0 < i < rows, 0 < j < cols
B2	Boundary	1,2	0 < i < rows, j < 0 || j >= cols
B3	Boundary	1	i < 0, i >= rows
BD1	Bad Data		Matrix has exception (null)

b. sparseToMatrix() condition:

Identifier	Goal	Note	Condition
CC1	Code Coverage		If sparse not null
CC2	Code Coverage		If sparse is exception
BR1	Branch Coverage	1	In range i = 0 to rows - 1
BR2	Branch Coverage	2	In range j = 0 to cols - 1
BR3	Branch Coverage	2, 3 true	In range j = 0 to cols -1 and value not 0 then add to index
BR4	Branch Coverage	2,3 false	In range j = 0 to cols - 1 and value is 0 then continue loop
BR5	Branch Coverage	1	Out of range i < 0 and i >= rows - 1. End loop
BR6	Branch Coverage	2	Out of range j < 0 and j >= cols - 1. Continue 1.
B1	Boundary	1,2	0 < i < rows, 0 < j < cols
B2	Boundary	1,2	0 < i < rows, j < 0 or j >= cols
B3	Boundary	1	i < 0, i >= rows
BD1	Bad Data		sparse has exception (null)