CALIFORNIA STATE UNIVERSITY,
NORTHRIDGE

DATA STRUCTURES AND PROGRAM DESIGN

COMP 182

Lab 1 Report

Written by
Kyle NGUYEN

Professor
Mohammed
ABDELRAHIM

September 11, 2017

1 Introduction

The project consists of two classes, a Main class and a Matrix class. The program will prompt the user to enter the dimensions for the first Matrix and then randomly populate this matrix with the dimensions entered by the user. It will then prompt the user for the dimensions of the second matrix. Once the dimensions of both matrices are finalized the user is then asked to pick an operation ranging from:

1. Add
2. Subtract
3. Multiply/Dot Product
4. Hadamard Product
5. Clone
6. Equal
7. Exit

The user will select one of the operations and the program will determine if the selected operation is possible due to the ranging dimensions of the matrices. The program will keep prompting which operation to do until the user ends the program.

The Main Methods will make instances of the Matrix class and prompt the user to input the dimensions, *row(s) and column(s)* for both matrix instances. Once the dimensions are finalized, there will be a set of operations that the user can select from, depending on which operation the user enters it will perform the assigned operation, if the dimensions are valid for the given operation.

The Matrix class will have three different fields, a double integer array to represent the location of each element in the matrix and two integer values to represent the dimensions. The Matrix class will have numerous methods to perform the desired operation on the instances created in the Main Method.

2 Class Matrix

2.1 Fields:

Starting off with Class Matrix are the three fields. There will need to be at least three fields, the row(s) of the matrix, the column(s) of the matrix, and the elements of the matrix. They're declared as integers because the dimensions are only positive integers and the elements are only positive integer locations. They are declared private since I do not want anything modifying the values of these fields.

```
1   private int mRows; // Rows
2   private int mCols; // Columns
3   private int[][] mElements; // Row x Column Int Array
```

2.2 Constructors:

There will be three constructors for this class:

1. A default constructor which initializes the matrix without any input arguments which will be initialized to a 3x3 matrix and randomly populated. *Lines (5-10)*

2. A second constructor that takes in two input arguments and creates a matrix with dimensions particular to the input and populates it randomly.*Lines (13-18)*

3. A third constructor that takes in an array as an input argument and creates an equivalent matrix instance with that particular array.*Lines (21-24)*

```
1   /**************************************
2    * CONSTRUCTORS *
3    **************************************/
4   // Default Constructor:
5   public Matrix() {
6      this.mRows = 3;
7      this.mCols = 3;
8      this.mElements = new int[mRows][mCols];
```

```
9       this.populate(-100, 100);
10   }
11
12   // Initialization Constructor given size
13   public Matrix(int rows, int cols) {
14      this.mRows = rows;
15      this.mCols = cols;
16      this.mElements = new int[mRows][mCols];
17      this.populate(-100, 100);
18   }
19
20   // Initialize Array Constructor
21   public Matrix(int[][] matrixArray) {
22      this.mRows = matrixArray.length;
23      this.mCols = matrixArray[0].length;
24      this.mElements = matrixArray.clone();
25
26   }
```

2.3 Getter Methods:

The getter methods are used to return the matrix's row or column to the caller.

```
1    /**************************************
2     * GET METHODS FOR ROW / COLUMN *
3     **************************************/
4    public int getRows() {
5       return this.mRows;
6    }
7
8    public int getCols() {
9       return this.mCols;
10   }
```

2.4 Clone Method:

The clone method returns an exact copy of the matrix of type Matrix.

```
/*****************************************
 * Matrix CLONE *
 *****************************************/
public Matrix clone() {
   return new Matrix(this.mElements.clone());
}
```

2.5 Populate Method:

The populate method iterates through the double array randomly populating the elements with the maximum possible number as well as the minimum.

```
/*****************************************
 * MATRIX POPULATE *
 *****************************************/
public void populate(int minValue, int maxValue) {
   Random genNums = new Random();

   for (int i = 0; i < mRows; i += 1) {
      for (int j = 0; j < mCols; j += 1) {
         this.mElements[i][j] = genNums.nextInt((maxValue -
             minValue) + 1) + minValue;
      }
   }
}
```

2.6 Addition, Subtraction, Hadamard Product:

These operations are identical as they iterate through the double array performing the specified operation. The calling instance's is added to the corresponding argument's elements of type matrix which result in another double integer array. The array is then passed into the matrix constructor which creates an equivalent matrix with that particular array. The only difference between these operation is the actual operation ((+, -, *) on *Line (9)*

```
/**************************************
 * MATRIX HADAMARD PRODUCT *
 **************************************/
public Matrix had(Matrix otherMatrix) {
   int[][] resultMatrixArray = new int[mRows][mCols];

   for (int i = 0; i < mRows; i += 1) {
      for (int j = 0; j < mCols; j += 1)
         resultMatrixArray[i][j] = this.mElements[i][j] *
             otherMatrix.mElements[i][j];
   }

   return new Matrix(resultMatrixArray);
}
```

2.7 Multiplication/Dot Product:

Of the operations this operation is not the same as addition, subtraction, or the Hadamard Product. Where in those operations the dimensions of the matrix stays the same, the dimensions for the Dot Product change. The new dimensions of the matrix are the row's from the calling instance and the column's of the argument's matrix. The operation iterates through the double array multiplying the row's of the instance's matrix to the column's of the argument's matrix. It requires three nested loops, the first two are the same as the other operations where it iterates through the rows, then the columns, but for multiplication it needs a third loop to control the index of which individual cell is being multiplied and added up. The array is then passed into the matrix constructor which accepts a double integer array and creates an equivalent matrix with that particular array.

```
/**************************************
 * MATRIX DOT PRODUCT *
 **************************************/
public Matrix mul(Matrix otherMatrix) {
   int[][] resultMatrixArray = new int[mRows][otherMatrix.mCols];
       // Rows of MatrixA, Cols of Matrix B

   for (int i = 0; i < mRows; i += 1)
```

```
8           for (int j = 0; j < otherMatrix.mCols; j += 1)
9              for (int k = 0; k < this.mCols; k += 1) {
10                resultMatrixArray[i][j] += this.mElements[i][k] *
                      otherMatrix.mElements[k][j];
11             }
12
13       return new Matrix(resultMatrixArray);
14    }
```

2.8 Equals Method:

The equals method returns a boolean whether the two matrices' values are equal. Using the .deepEquals compares two arrays of single dimensional or multidimensional arrays are equal or not where as the .equals method only works for single dimensional arrays. For this case the .deepEquals was the obvious choice.

```
1     public boolean isEqual(Matrix otherMatrix) {
2        if (Arrays.deepEquals(this.mElements, otherMatrix.mElements))
             { // Returns true if elements of the array are truly equal
3           return true;
4        }
5
6        else
7           return false;
8     }
```

2.9 toString Method:

The toString method converts the called matrix instance to a String in order for the Main Method to display it. Using a for each loop, the first for each loop iterates over the double array and stores it in a single dimensional array. Since it is multidimensional, you have to iterate over again to actually get the elements, using another for each loop it iterates over the single dimensional array and stores it in an integer since the array is an array of integers, iterating over it again gives you the actual elements of the multidimensional array. The results of each row are then placed into

a string and once the row is completed a newline is placed for the second row and so forth.

```
/**************************************
 * MATRIX TO STRING *
 **************************************/
public String toString() {
   String displayMatrixArray = "";
   for (int[] i : this.mElements) { // For each int [][] Matrix, iterate to get int [] i
      displayMatrixArray += "[";

      for (int j : i) { // For each int [], iterate to get int elements
         displayMatrixArray += String.format(" % 6d", j); // Do something with elements
      }

      displayMatrixArray += "] \n"; // New line for next row
   }

   return displayMatrixArray;
}
```

3 Main Class

3.1 Fields:

The fields consist of a scanner class scan which is used for user input throughout the entirety of the program.

```
private static Scanner scan = new Scanner(System.in);
```

3.2 Main Method:

The main method has two Strings which are the names of the matrix that will be used when prompted for user input. Two instances are made

of the Matrix Class and are the dimensions are set through a method which prompts the user for the dimensions and returns a Matrix instance which is assigned to either Matrix A or Matrix B. Matrix C is simply just a clone of Matrix A. The next task is to prompt the user for which operation they would like to do. Once the user decides to end the program by entering the operation to end the program the scanner is finally closed and the program ends.

```
1   public static void main(String[] args) {
2      String A = "A"; // Matrix A identifier
3      String B = "B"; // Matrix B identifier
4
5      // Method to get dimensions for MatrixA and Matrix B
6      Matrix myMatrixA = getDimensions(A);
7      Matrix myMatrixB = getDimensions(B);
8
9      // Matrix C is a clone of Matrix A
10     Matrix myMatrixC = myMatrixA.clone();
11
12     // What operation?
13     operation(myMatrixA.clone(), myMatrixB.clone(),
           myMatrixC.clone());
14
15     scan.close(); // Close scanner
16     System.exit(0);
17  }
```

3.3 getDimensions Method:

The getDimensions method prompts the user for input for the matrix specified by the argument passed in when the method was called. It takes in input for the rows of the matrix and one of four things can happen.

1. The user did not enter anything. *Lines (9-12)*
2. The user entered something that was not an integer. *Lines (66-68)*
3. The user entered an integer, but the integer was not a valid integer. *Lines (20-22)*

4. The user entered a valid integer.*Lines (24-70)*

```
1        boolean bInputValid = true; // Flag for valid input
2
3        // Get row(s) for Matrix
4        System.out.print("Enter the # of row(s) for Matrix " +
             matrixIdentifier + ": ");
5        do {
6           String sRowsMatrix = scan.nextLine();
7
8           // Did you enter anything?
9           if (sRowsMatrix.isEmpty()) {
10             System.out.println("You didn't type anything!\r\n");
11             System.out.print("Enter a valid # of row(s) for Matrix "
                   + matrixIdentifier + ": ");
12          }
13
14          // Is it a number?
15          else if (isNumeric(sRowsMatrix)) {
16             {
17                nRowsMatrix = Integer.parseInt(sRowsMatrix);
18
19                // Is it positive?
20                if (nRowsMatrix <= 0) {
21                   System.out.print("\r\nEnter a positive # for the
                         row(s) of Matrix " + matrixIdentifier + ": ");
22                }
23                // Yes! It's a valid input, get column(s)
24                else {
25
26                   // Get column(s) for Matrix
27                   System.out.print("\r\nEnter the # of column(s) for
                         Matrix " + matrixIdentifier + ": ");
28                   do {
29                      String sColsMatrix = scan.nextLine();
30
31                      // Did you enter anything?
32                      if (sColsMatrix.isEmpty()) {
33                         System.out.println("You didn't type
                               anything!\r\n");
```

```
34                      System.out.print("Enter a valid # of column(s)
                            for Matrix " + matrixIdentifier + ": ");
35                   }
36
37                   // Is it a number?
38                   else if (isNumeric(sColsMatrix)) {
39                      {
40                         nColsMatrix = Integer.parseInt(sColsMatrix);
41
42                         // Is it positive?
43                         if (nColsMatrix <= 0) {
44                            System.out.print("\r\nEnter a positive #
                                  for the column(s) of Matrix " +
                                  matrixIdentifier + ": ");
45                         }
46
47                         // Yes! End the while loop and get
                               dimensions.
48                         else {
49                            System.out.println();
50                            bInputValid = false;
51
52                         }
53                      }
54                   }
55
56                   // Invalid input
57                   else {
58                      System.out.print("\r\nEnter a valid # of
                            column(s) for Matrix " + matrixIdentifier
                            + ": ");
59                   }
60
61                } while (bInputValid);
62             }
63          }
64       }
65       // Invalid Input
66       else {
67          System.out.print("\r\nEnter a valid # of row(s) for
```

```
                Matrix " + matrixIdentifier + ": ");
68          }
69
70      } while (bInputValid);
```

This do-while loop will loop forever until the condition is met and the condition is only met until the user enters a valid number for the row(s) and column(s) for the specified matrix.

To test whether the input is a valid integer another method is called isNumeric and returns a boolean if the input is an integer or not.

```
1   /**************************************
2    * IS IT A NUMBER *
3    **************************************/
4   public static boolean isNumeric(String input) {
5      try {
6         Integer.parseInt(input);
7
8         return true;
9
10     }
11
12     catch (Exception e) {
13        return false;
14     }
15  }
```

When the input testing is completed the input was assigned to the integer values for the row(s) and column(s), which were passed as the arguments for the initialization constructor for the desired row(s) and column(s). The matrix is then returned back to the caller of type Matrix.

```
1       // Matrix constructor (rows, cols)
2       Matrix myMatrix = new Matrix(nRowsMatrix, nColsMatrix);
3       return myMatrix;
```

3.4 whatOperation Method:

The whatOperation method is a while loop that will loop until the condition where the user enters the *"Exit"* operation and sets the flag to false. This method will determine what operation the user entered whether it was valid or not and prompt them endlessly until the user exits. In this method it also tests whether or not the operation are valid on the matrices inputted at the start.

The method starts off with some the arguments passed being assigned to variables of type Matrix. Since these arguments were clones of the original, it is same to do operations on them without modifying the original value. There are integers which are assigned the row(s) and column(s) of matrices A and B which is later used in determining whether the operation is valid or not.

The matrices then are converted to variables of type String and the whatOperation method really begins. //

```
1       // Everything is a clone now, it's safe to perform operations
            without modifying data.
2       Matrix myMatrixA = A;
3       Matrix myMatrixB = B;
4       Matrix myMatrixC = C;
5
6       int nArow = myMatrixA.getRows();
7       int nAcol = myMatrixA.getCols();
8
9       int nBrow = myMatrixB.getRows();
10      int nBcol = myMatrixB.getCols();
11
12      // Using Class Matrix toString method
13      String sMatrixA = myMatrixA.toString();
14      String sMatrixB = myMatrixB.toString();
15      String sMatrixC = myMatrixC.toString();
```

The operation starts off by prompting the user with the operations available which was earlier stated in the introduction. Another method which validates which input was chosen is then called and returns an in-

teger of which operation was pressed when the input was valid. It has the same input filtering as it does when obtaining the dimensions for the matrices. *Lines (15, 21, 25, 36)*

```
1   /**************************************
2    * WHAT OPERATION DID YOU PICK? *
3    **************************************/
4   public static int whichOperation() {
5      int whatOp = 0;
6
7      boolean bInputValid = true;
8
9      // Get operation
10     System.out.print("What operation do you want to do? ");
11     do {
12        String sInputOp = scan.nextLine();
13
14        // Did you enter anything?
15        if (sInputOp.isEmpty()) {
16           System.out.println("You didn't type anything!\r\n");
17           System.out.print("Enter a valid operation from 1 through
                 7: ");
18        }
19
20        // Is it a number?
21        else if (isNumeric(sInputOp)) {
22           whatOp = Integer.parseInt(sInputOp);
23
24           // Is it within the range (1 - 7)?
25           if (whatOp > 7 || whatOp < 1) {
26              System.out.print("\r\nEnter an operation ONLY from 1
                    through 7: ");
27           }
28
29           // Yes, it's a valid operation! End while loop and do
                 "operation".
30           else {
31              System.out.println();
32              bInputValid = false;
```

```
33              }
34           }
35           // Invalid input
36           else {
37              System.out.print("\r\nEnter a valid operation from 1
                    through 7: ");
38           }
39        } while (bInputValid);
40
41        return whatOp;
42     }
```

Before the results of a valid operation, the program prints out Matrices A and B for reference and displays the result below the operation performed on these matrices. It prints out the string which converted the variable of type Matrix earlier in the whatOperation method and displays the String representation.

```
1     /**************************************
2      * PRINT OUT MATRIX A AND MATRIX B *
3      **************************************/
4     public static void printMatrixAandB(String matrixA, String
          matrixB) {
5        String sMatrixA = matrixA;
6        String sMatrixB = matrixB;
7
8        System.out.println("Matrix A:");
9        System.out.println(sMatrixA);
10
11       System.out.println("Matrix B:");
12       System.out.println(sMatrixB);
13    }
```

With similar input filtering the method only returns an integer from 1 to 7. The operation is then put through a switch which then performs the operation. Before actually performing the operation the case actually checks whether or not the dimensions are valid before it tries to do the specified operation. If it is valid it will continue, if it isn't it will prompt the user that there were invalid dimensions for the given operation. If the operation is valid it will perform the operation and print the String repre-

sentation of operations on the matrices.

```
        int whatOp = whichOperation();

        // Which operation from (1 - 7)?
        switch (whatOp) {
        /*************************************
         * MATRIX ADDITION *
         *************************************/
        case 1: {
           // Valid dimensions?
           if (nArow != nBrow || nAcol != nBcol) {
              System.out.println("Invalid dimensions for Addition.");
              System.out.println();
              break;
           }

           // Yes! Perform addition.
           else {
              printMatrixAandB(sMatrixA, sMatrixB);

              String sMResultAdd =
                  myMatrixA.add(myMatrixB).toString();

              System.out.println("Addition: Matrix A + Matrix B");
              System.out.println(sMResultAdd);
              break;
           }
        }
```

```
        /*************************************
         * MATRIX SUBTRACTION *
         *************************************/
        case 2: {
           // Valid dimensions?
           if (nArow != nBrow || nAcol != nBcol) {
              System.out.println("Invalid dimensions for
                  Subtraction.");
              System.out.println();
```

```
            break;
         }

         // Yes! Perform subtraction.
         else {
            printMatrixAandB(sMatrixA, sMatrixB);

            String sMResultSub =
                myMatrixA.sub(myMatrixB).toString();

            System.out.println("Subtraction: Matrix A - Matrix B");
            System.out.println(sMResultSub);
            break;
         }
      }
```

```
      /**************************************
       * MATRIX DOT PRODUCT *
       **************************************/
      case 3: {
         // Valid dimensions?
         if (nAcol != nBrow) {
            System.out.println("Invalid dimensions for
                Multiplication (Dot Product).");
            System.out.println();
            break;
         }

         // Yes! Perform dot product(multiplication).
         else {
            printMatrixAandB(sMatrixA, sMatrixB);

            String sMResultMul =
                myMatrixA.mul(myMatrixB).toString();

            System.out.println("Multiplication (Dot Product):
                Matrix A * Matrix B");
            System.out.println(sMResultMul);
            break;
```

```
            }
        }
```

```
        /****************************************
         * MATRIX HADAMARD PRODUCT *
         ****************************************/
        case 4: {
            // Valid dimensions?
            if (nArow != nBrow || nAcol != nBcol) {
                System.out.println("Invalid dimensions for Hadamard
                    Product.");
                System.out.println();
                break;
            }

            // Yes! Perform Hadamard product.
            else {
                printMatrixAandB(sMatrixA, sMatrixB);

                String sMResultHad =
                    myMatrixA.had(myMatrixB).toString();

                System.out.println("Hadamard Product: Matrix A o
                    Matrix B:");
                System.out.println(sMResultHad);
                break;
            }
        }
```

I added a couple of more operations such as the clone and equals operation to the list as a part of another challenge I wanted to give myself as it asks the user for the clone operation which matrix do they want to clone. Again with the similar input filtering it would only work if the user entered the name for either Matrix A or B, but these identifiers can be changed in the Main Method. The equals operation determines whether or not the matrix chosen is equal to or not equal to Matrix C.

```
        /****************************************
```

```
2            * CLONE OF MATRIX A/B *
3            ***************************************/
4           case 5: {
5              boolean bInputValid = true;
6
7              System.out.print("Which Matrix do you want to clone? A or
                   B? ");
8
9              do {
10                String sAorB = scan.nextLine();
11
12                // Did you enter anything?
13                if (sAorB.isEmpty()) {
14                   System.out.println("You didn't type anything!\r\n");
15                   System.out.print("Please choose either Matrix A or
                        Matrix B. ");
16                }
17
18                // Is it equal to "a" or "A"? Yes? Make clone of
                      MatrixA then end while loop!
19                else if (sAorB.equals("a") || sAorB.equals("A")) {
20                   System.out.println();
21                   printMatrixAandB(sMatrixA, sMatrixB);
22
23                   System.out.println("Clone of Matrix A:");
24                   String sMclone = myMatrixA.toString();
25                   System.out.println(sMclone);
26                   bInputValid = false;
27                }
28
29                // Is it equal to "b" or "B"? Yes? Make clone of
                      MatrixB then end while loop!
30                else if (sAorB.equals("b") || sAorB.equals("B")) {
31                   printMatrixAandB(sMatrixA, sMatrixB);
32
33                   System.out.println("Clone of Matrix B:");
34                   String sMclone = myMatrixB.toString();
35                   System.out.println(sMclone);
36                   bInputValid = false;
37                }
```

```
38
39              // Invalid input
40              else {
41                 System.out.println("Invalid Matrix option!\r\n");
42                 System.out.print("Please choose either Matrix A or
                       Matrix B. ");
43              }
44           } while (bInputValid);
45
46           break;
47        }
```

```
1         /****************************************
2          * MATRIX A/B EQUAL TO MATRIX C*
3          ****************************************/
4         case 6: {
5            boolean bInputValid = true;
6
7            System.out.print("Which Matrix do you want to compare? A
                 or B? ");
8            do {
9               String sAorB = scan.nextLine();
10
11              if (sAorB.isEmpty()) {
12
13                 // Did you enter anything?
14                 System.out.println("You didn't type anything!\r\n");
15                 System.out.print("Please choose either Matrix A or
                       Matrix B. ");
16              }
17
18              // Is it equal to "a" or "A"? Yes? Do comparison, then
                    end while loop!
19              else if (sAorB.equals("a") || sAorB.equals("A")) {
20                 System.out.println();
21                 printMatrixAandB(sMatrixA, sMatrixB);
22
23                 System.out.println("Matrix C:");
24                 System.out.println(sMatrixC);
```

```
25
26                  // Matrix Class method isEqual
27                  boolean equal = myMatrixA.isEqual(myMatrixC);
28                  if (equal) {
29                     System.out.println("Matrix A is EQUAL to Matrix
                           C!");
30                  } else {
31                     System.out.println("Matrix A is NOT EQUAL to
                           Matrix C!");
32                  }
33
34                  bInputValid = false;
35               }
36
37               // Is it equal to "b" or "B"? Yes? Do comparison, then
                     end while loop!
38               else if (sAorB.equals("b") || sAorB.equals("B")) {
39                  printMatrixAandB(sMatrixA, sMatrixB);
40
41                  System.out.println("Matrix C:");
42                  System.out.println(sMatrixC);
43
44                  // Class Matrix method isEqual
45                  boolean equal = myMatrixB.isEqual(myMatrixC);
46                  if (equal) {
47                     System.out.println("Matrix B is EQUAL to Matrix
                           C!");
48                  } else {
49                     System.out.println("Matrix B is NOT EQUAL to
                           Matrix C!");
50                  }
51
52                  bInputValid = false;
53               }
54
55               // Invalid input
56               else {
57                  System.out.println("Invalid Matrix option!\r\n");
58                  System.out.print("Please choose either Matrix A or
                        Matrix B. ");
```

```
59              }
60           } while (bInputValid);
61
62           break;
63        }
```

Since Matrix C was a clone of Matrix A, the program would yield A is equal to C, but it was not hard coded in there as shown above and actually tests whether they are equal or not. The last part to the switch statement is the *"Exit"* and *"Default"* cases. The exit statement which simply ends the loop and returns back to the Main Method, where as the default case should always be there in case of program errors. The program should not reach the default case, but if it ever does it can be easily fixed if need be.

```
1        /*****************************************
2         * PROGRAM ENDING *
3         *****************************************/
4        case 7: {
5           System.out.print("Program Ended . . . ");
6           notEnding = false; // End while loop
7           break;
8        }
9
10       /*****************************************
11        * SOMETHING WRONG.. *
12        *****************************************/
13       default: {
14          System.out.print("Something Wrong . . . "); // Who knows
                what happened if it got here
15          notEnding = false; // End while loop
16          break;
```

4 Conclusion

The program is working as expected as it prompts the user for input for the dimensions of Matrix A and subsequently asking the dimensions Matrix B. It will then prompt the user a list of operations they can choose from, and if they operations are valid and the dimensions are valid they

will perform the given operation, otherwise it will print out an error message. This will loop continuously until the user chooses the *"Exit"* operation or if an unforeseen error occurred where none of the cases would match up and land in the default case. The program ends there once the user *"Exits"*.