CPSC 331 - Assignment 1
Student: Marc-Andre Fichtel
Due: June 2, 11pm

Methodology

Assignment 1's program is divided into three subprograms: **SP1**, **SP2**, and **SP3**. In my implementation, these three subprograms were constructed as three methods. There is also a main method, which executes the subprograms in order, and catches any thrown exceptions. Lastly, three important matrix variables are declared globally after the class definition. Since the main method is in the same class as the subprograms and global variables, both subprograms and variables were declared as static. The global variables include three 2D integer arrays (i.e. matrices; two will be defined by the user, the third is the result of the matrix multiplication of the first two). While it is possible to declare *matrixC* in the main method only, it was declared globally to keep all matrix variables in the same place.

SP1: The "*GetMatrixInput*" method takes no parameters. It prompts the user for all required inputs, assigns them to their respective global variables, and checks their validity. Exceptions are thrown, if any input is not an integer, if the given two matrices' dimensions are outside the range stated in the assignment (i.e. between 1 and 5 for both rows and columns on either matrix), or if the given matrices' dimensions are incompatible for multiplication (i.e. if *matrixA*'s number of columns do not match *matrixB's* number of rows). Since inputs are directly assigned to the global variables, the method does not need to return anything.

SP2: The "*MultiplyMatrices*" method takes two parameters: The two 2D integer arrays defined by the user in **SP1**, *matrixA* and *matrixB*. It computes the resulting matrix of multiplying the given two matrices, in order AxB. In the first step, *matrixC* is initialized with the number of rows from *matrixA* and the number of columns from *matrixB*. No exceptions are expected to occur here. The method returns the result of multiplying *matrixA* with *matrixB*.

SP3: The "*DisplayMatrices*" method takes three arguments: All three 2D integer array matrices, including the two user defined ones, *matrixA* and *matrixB*, as well as the result of multiplying these two, *matrixC*. It prints those matrices to the console in a square matrix format. To do this, two nested for loops are used to iterate over each element in each matrix, which is printed to the console, with newline characters at the end of each row. No exceptions are expected to occur here.

The program's main method call all subprograms in succession, and surrounds them with a try/catch block to detect any thrown exceptions. Should an exception be thrown, then it will be printed to the console for the user to see. Concluding the program, the input Scanner is closed.

Proof of Correctness

Intermediate assertion I: Declare three global 2D integer arrays.

→ {P_0} int[][] *matrixA*; int[][] *matrixB*; int[][] *matrixC*; {Q_0}

SP1:

- Precondition **P1** - Inputs include:
    - *matrixA and matrixB*: Globally declared uninitialized static 2D integer arrays
    - User integer input for number of dimensions of *matrixA* and *matrixB*, each between 1 and 5, where *matrixA*'s number of columns is equal to *matrixB*'s number of rows.
    - User integer input for every entry of *matrixA* and *matrixB*.
- Postcondition **Q1**:
    - *matrixA*: A globally declared initialized static 2D integer array with number of rows *m* and number of columns *n,* where $0 < m, n < 6$.
    - *matrixB*: A globally declared initialized static 2D integer array with number of rows *n* and number of columns *k,* where $0 < n, k < 6$.
        - SP1 was successfully executed

Intermediate assertion I: Global 2D integer arrays *matrixA* and *matrixB* declared, user gives integer input for matrix dimensions.

→ {**P1**} Scanner input = new Scanner (System.in); int matrixARows = input.nextInt(); int matrixACols = input.nextInt(); int matrixBRows = input.nextInt(); int matrixBCols = input.nextInt(); {R_1}

Intermediate assertion II: user gives integer input between 1 and 5.

→ {R_2} if (matrixARows < 1 || matrixARows > 5 || matrixACols < 1 || matrixACols > 5 || matrixBRows < 1 || matrixBRows > 5 || matrixBCols < 1 || matrixBCols > 5) {throw new Exception ("Incorrect dimension values entered.");} {R_3}

Intermediate assertion III: *matrixACols* equals *matrixBRows*.

→ {R_3} if (matrixACols == matrixBRows) {matrixA = new int[matrixARows][matrixACols]; matrixB = new int[matrixBRows][matrixBCols];} else {throw new Exception ("Matrix dimensions are incompatible for multiplication.");} {R_4}

Intermediate assertion IV: User gives integer input

→ {R_4} for (int i = 0; i < matrixARows; i++) {for (int j = 0; j < matrixACols; j++) {matrixA[i][j] = input.nextInt();}} for (int i = 0; i < matrixBRows; i++) {for (int j = 0; j < matrixBCols; j++) {matrixB[i][j] = input.nextInt();}} input.close(); {**Q1**}

→ Proof: The loop invariants (matrix dimension integers) are bound to values of at least 1 or at most 5 by *intermediate assertion III*, and each loop variant starts at 0. Therefore, each loop will execute between 1 and 5 times. Since the four for loops consist of two pairs of nested loops, there will be a minimum of 1*1 + 1*1 = 2, and a maximum of 5*5 + 5*5 = 50 executions, so in any case the loops will terminate.

- Precondition **P2** - Inputs include:
    - *matrixA and matrixB*: Globally declared uninitialized static 2D integer arrays
    - User integer input for number of dimensions of *matrixA* and *matrixB*, where any one of them is smaller than 1 or greater than 5, and *matrixA*'s number of columns is equal to *matrixB*'s number of rows.
    - User integer input for every entry of *matrixA* and *matrixB*.
- Postcondition **Q2:**
    - An exception (Dimensions out of range) is thrown.
        - **SP1** was unsuccessful (an error occurred)

Intermediate assertion I: Global 2D integer arrays *matrixA* and *matrixB* declared, user gives integer input for matrix dimensions.

→ {**P2**} Scanner input = new Scanner (System.in); int matrixARows = input.nextInt(); int matrixACols = input.nextInt(); int matrixBRows = input.nextInt(); int matrixBCols = input.nextInt(); {$\mathbf{R_1}$}

Intermediate assertion II: user gives integer input smaller than 1 or greater than 5.

→ {$\mathbf{R_2}$} if (matrixARows < 1 || matrixARows > 5 || matrixACols < 1 || matrixACols > 5 || matrixBRows < 1 || matrixBRows > 5 || matrixBCols < 1 || matrixBCols > 5) {throw new Exception ("Incorrect dimension values entered.");} {**Q2**}

- Precondition **P3** - Inputs include:
    - *matrixA and matrixB*: Globally declared uninitialized static 2D integer arrays
    - User integer input for number of dimensions of *matrixA* and *matrixB*, each between 1 and 5, where *matrixA*'s number of columns is not equal to *matrixB*'s number of rows.
    - User integer input for every entry of *matrixA* and *matrixB*.
- Postcondition **Q3:**
    - An exception (Incompatible matrix dimensions) is thrown.
        - **SP1** was unsuccessful (an error occurred)

Intermediate assertion I: Global 2D integer arrays *matrixA* and *matrixB* declared, user gives integer input for matrix dimensions.

→ {**P3**} Scanner input = new Scanner (System.in); int matrixARows = input.nextInt(); int matrixACols = input.nextInt(); int matrixBRows = input.nextInt(); int matrixBCols = input.nextInt(); {$\mathbf{R_1}$}

Intermediate assertion II: user gives integer input between 1 and 5.

→ {$\mathbf{R_2}$} if (matrixARows < 1 || matrixARows > 5 || matrixACols < 1 || matrixACols > 5 || matrixBRows < 1 || matrixBRows > 5 || matrixBCols < 1 || matrixBCols > 5) {throw new Exception ("Incorrect dimension values entered.");} {$\mathbf{R_3}$}

Intermediate assertion III: *matrixACols* does not equal *matrixBRows*.

→ {$\mathbf{R_3}$} if (matrixACols == matrixBRows) {matrixA = new int[matrixARows][matrixACols]; matrixB = new int[matrixBRows][matrixBCols];} else {throw new Exception ("Matrix dimensions are incompatible for multiplication.");} {**Q3**}

- Precondition **P4** - Inputs include:
    - *matrixA and matrixB*: Globally declared uninitialized static 2D integer arrays
    - User input of any type other than integer
- Postcondition **Q4:**
    - An exception (Input mismatch) is thrown.
        - **SP1** was unsuccessful (an error occurred)

Intermediate assertion I: Global 2D integer arrays *matrixA* and *matrixB* declared, user gives non-integer input for matrix dimensions.

→ {**P4**} Scanner input = new Scanner (System.in); int matrixARows = input.nextInt(); int matrixACols = input.nextInt(); int matrixBRows = input.nextInt(); int matrixBCols = input.nextInt(); {**Q4**}

SP2:

- Precondition **P5**:
    - *matrixA*: A globally declared initialized static 2D integer array with number of rows *m* and number of columns *n.*
    - *matrixB*: A globally declared initialized static 2D integer array with number of rows *n* and number of columns *k*.
    - *m, n, k* > 0.
- Postcondition **Q5**:
    - *matrixC*: A globally declared initialized static 2D integer array with number of rows *m* and number of columns *k*.

Intermediate assertion I: Two 2D integers arrays *matrixA* and *matrixB are given*, where *matrixA*'s number of columns matches *matrixB*'s number of rows.

→ {**P5**} int[][] result = new int[matrixA.length][matrixB[0].length]; for (int i = 0; i < matrixA.length; i++) {for (int j = 0; j < matrixB[0].length; j++) {for (int k = 0; k < matrixA[0].length; k++) {result[i][j] += matrixA[i][k] * matrixB[k][j];}}} {**Q5**}

→ Proof: *matrixA* is given with dimensions m*n, and *matrixB* is given with dimensions n*k, so the matrices are compatible and the result is initialized with dimensions m*k. There are thus three *loop invariants*: The number m of rows in *matrixA*, the number k of columns in *matrixB*, and the number n of columns in *matrixA* and rows in *matrixB*. The three nested for loops iterate over each of these with *loop variants* starting at 0. The number of calculations is thus m*n*k, which, since m,n,k > 0, is at least 1*1*1 = 1 calculation. While this subprogram does not require an upper limit, m,n,k are essentially bound to values between 1 and 5 in this program, so the maximum number of calculations here is 5*5*5 = 125, so in any case, the loops will terminate.

SP3:

- Precondition **P6**:
    - *matrixA*: A given 2D integer array
    - *matrixB*: A given 2D integer array
    - *matrixC*: A given 2D integer array

- Postcondition **Q6**:
    - *matrixA, matrixB,* and *matrixC* have been printed to the console in square matrix format.

Intermediate assertion I: A given 2D integer array matrixA exists.

→ {**P6**} System.out.println("\nFirst matrix: "); for (int i = 0; i < matrixA.length; i++) { for (int j = 0; j < matrixA[0].length; j++) {System.out.print(matrixA[i][j] + "\t");System.out.print("\n"); }} {**R₁**}

Intermediate assertion I: A given 2D integer array matrixB exists.

→ {**R₁**} System.out.println("\nSecond matrix: "); for (int i = 0; i < matrixB.length; i++) { for (int j = 0; j < matrixB[0].length; j++) {System.out.print(matrixB[i][j] + "\t");System.out.print("\n"); }} {**R₂**}

Intermediate assertion I: A given 2D integer array matrixC exists.

→ {**R₂**} System.out.println("\nThird matrix: "); for (int i = 0; i < matrixC.length; i++) { for (int j = 0; j < matrixC[0].length; j++) {System.out.print(matrixC[i][j] + "\t");System.out.print("\n"); }} {**Q6**}

Analysis (Worst Case Cost)

- 3 global variables declarations → 3 charged units.
- SP1:
    - Assignment statements, user input prompts, print statements, control statements, exception statements → 29 charged units.
    - 2*5*5*3 for two pairs of nested for loops with a print statement, an assignment statement, and a user input prompt each loop → 150 charged units.
- SP2:
    - Assignment statement, return statement, array access method calls → 4 charged units.
    - 5*5*5*2 for three nested for loops with an assignment statement and an arithmetic addition each loop → 250 charged units.
- SP3:
    - Print statements → 3 charged units.
    - 3*5*5*1+3*5*1 for three pairs of nested for loops with a print statement each loop, and a print statement each outer loop → 90 charged units.
- Main:
    - 3 method calls, an assignment statement, and a print statement → 5 charged units.

Worst Case Cost: 3+29+150+4+250+3+90+5 = 534.

Note: If this algorithm wasn't limited to a maximum matrix size of 5x5, the worst case cost would be m*n*k.