1. Section 9 Exercises

- Exercise 9-1: The matrix m is an array of 4 arrays, each with 6 members. If m is regarded as a 2-dimensional array, then
  - m[1][2] is 75
  - m[2][5] is 88
  - m[4][1] is a run-time error
  - m[3] references { 58, 72, 66, 57, 76, 73 }
Exercise 9-2 - Max value in a matrix

GIVENS: 
m (a matrix) 
nRows (the number of rows in m) 
nCols (the number of columns in m)

INTERMEDIATES: 
row (index of current row) 
rol (index of current column)

RESULT: 
max (the maximum value in the matrix)

HEADER: 
max ← findMatrixMax(m, nRows, nCols)

BODY:

Alternative:

max ← maxInArray(m[0], nCols)

row ← 1
row < nRows ?
true
false
row ← row + 1
rowMax ← maxInArray(m[row], nCols)
rowMax > max ?
true
false
max ← rowMax
row ← row + 1
Program Memory

Exercise 9-3 - Diagonal-check algorithm

GIVENS:
- m (a matrix)
- nRows (the number of rows in m)
- nCols (the number of columns in m)

INTERMEDIATES:
- row (index of current row)
- col (index of current column)

RESULT:
- isDiagonal (TRUE if matrix is diagonal, false otherwise)

HEADER:
- isDiagonal ← checkDiag( m, nRows, nCols )

BODY:

Efficient Version

...
// Note: Integer.MIN_VALUE is the most // negative allowable integer for a Java // int, and can be used for -∞.

public static int matrixMax (int[][] m, int nRows, int nCols) {
    int max = Integer.MIN_VALUE; // m[0][0] is an alternate choice
    int max; // if the matrix m is not empty
    int row; // INTERMEDIATE
    int col; // INTERMEDIATE
    for (row = 0; row < nRows; row = row + 1) {
        for (col = 0; col < nCols; col = col + 1) {
            if (m[row][col] > max) {
                max = m[row][col];
            } else {
                /* do nothing */;
            }
        }
    }
    return max;
}
public static int readMatrix(int nRows, int nCols) {
    int row, col;
    int nRows, nCols;
    int[][] m;
    System.out.println("Enter number of rows and number of columns: ");
    nRows = ITI1120.readInt();
    nCols = ITI1120.readInt();
    m = new int[nRows][nCols];
    for (row = 0; row < nRows; row = row+1) {
        System.out.println("Enter the values for row " + row);
        for (col=0; col < nCols; col = col+1) {
            m[row][col] = ITI1120.readInt();
        }
    }
Exercise 9-6 - Find Cheap Direct Flights

GIVENS:
- home (the number of the city you live)
- cost (reference to the cost matrix)
- d (the amount you afford)
- n (the total number of cities)

RESULTS:
- cities (reference to an array of cities which you can visit)

INTERMEDIATE:
- aCity (the city we are currently checking)
- numCities (the number of cities to which you can go)

HEADER

(numCities, cities) ← cheapDirectFlights (home, cost, d, n)

Incomplete…
We must also put the list of cities in an array of the right size.
public static int[] cheapDirectFlights(int home, int[][] cost, int d, int n )
{
    int[] cities; // RESULT: an array of cities we can afford to visit
    // INTERMEDIATES:
    int aCity; // The city currently checking
    int numCities; // Number of cities we can visit
    int[] tempCities; // Temporary array for cities
    // BODY
    tempCities = new int[n-1];
    numCities = 0;
    for (aCity = 0; aCity < n ; aCity = aCity + 1 )
    {
        if(( aCity != home ) &&
            (cost[home][aCity ] <= d ) )
        {
            tempCities[numCities] = aCity;
            numCities = numCities + 1;
        }
        else
        {
            /* do nothing */ ;
        }
    }
    // At this point we have to get around Java's inability to return more than one value.
    // Create a new array of the correct length. The caller can obtain the number of cities by the
    // length of the array.
    cities = new int[numCities];
    for ( aCity = 0; aCity < numCities; aCity = aCity + 1 )
    {
        cities[aCity] = tempCities[aCity];
    }
    // Now return the array of cities with correct length
    return cities;
}
Exercise 9-7 - Delete a Row

GIVENS: M (a square matrix)
N (number of rows and columns in M)
R (row number to be removed)

RESULTS: (none)

MODIFIED: M (the original matrix with row R removed, and all rows moved up by one)

INTERMEDIATES: Index (index of row being moved)

HEADER: DeleteRow(M, N, R)

BODY:

```java
public static void deleteRow(int[][] m, int n, int r) {
    int index; // INTERMEDIATE
    for (index = r + 1; index < n; index = index + 1) {
        moveUp(m, n, index);
    }
    putRowToZero(m, n);
}
```

private static void moveUp(int[][] m, int n, int r) {
    int index; // INTERMEDIATE
    for (index = 0; index < n; index = index + 1) {
        m[r - 1][index] = m[r][index];
    }
}

This approach is necessary for moving columns. To move a row, we could use (without a loop, see alternative deleteRow algorithm). Careful, the above moves references, not the actual rows.
Exercise 9-7 - Delete a Row

GIVENS:  
- \( m \) (a square matrix)  
- \( n \) (number of rows and columns in \( m \))  
- \( r \) (row number to be removed)

RESULTS:  
- (none)

MODIFIEDS:  
- \( m \) (the original matrix with row \( r \) removed, and all rows moved up by one)

INTERMEDIATES:  
- \( \text{index} \) (index of row being moved)

HEADER:  
- `deleteRow(m, n, r)` (Alternative algorithm using the array of references)

BODY:

```java
public static void deleteRow(int[][] m, int n, int r) {
    int index; // INTERMEDIATE
    for (index = r + 1; index < n; index = index + 1) {
        m[index-1] = m[index];
    }
    putRowToZero(m, n);
}
```

This approach cannot be used with columns. Why?

Write 0's into the last row of the matrix \( m \) of size \( n \)

PutRowToZero(m, n)

\( m[\text{index}-1] \leftarrow m[\text{index}] \)
\( \text{index} \leftarrow \text{index} + 1 \)
\( \text{index} < n ? \)
\( \text{false} \)
\( \text{index} \leftarrow r+1 \)
\( \text{true} \)
Exercise 9-7 (continued) - Delete a Row

GIVENS: \( m \) (a square matrix)  
\( n \) (size of \( m \))

RESULTS: (none)

MODIFIED: \( m \) (last row put to 0)

INTERMEDIATE: index (index of the column)

HEADER: putRowToZero(m, n)

```java
private static void putRowToZero(int[][] m, int n) {
    int index; // INTERMEDIATE
    for (index = 0; index < n; index = index + 1) {
        m[n - 1][index] = 0;
    }
}
```