2D Arrays Test 2

1. Which of the following correctly instantiates a two-dimensional array of strings with dimensions of 3 rows and 2 columns?
2. String[][] arr = new Array[2][3];
3. String[][] arr = new Array[3][2];
4. String[][] arr = new String[2][3];
5. String[][] arr = new String[3][2];
6. Assume a two-dimensional array named mat has been instantiated. Refer to the following code segment.

for (int r = 0; r < /\* code1 \*/ ; r++)

{

for (int c = 0; c < /\* code2 \*/ ; c++)

{  
 System.out.print(mat[r][c] + " ");  
 }

}

Which of the following replaces /\* code1 \*/ and /\* code2 \*/ so that the code above displays the entire contents of mat regardless of its dimensions.

/\* code1 \*/ /\* code2 \*/

1. mat.length[r] mat[c].length
2. mat.length mat[r].length
3. mat.length mat.length
4. mat.length r

**Questions 3-4 refer to the following array declaration.**

Assume a two-dimensional array named mat has been instantiated and has been assigned the following integer values.

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **4** | **5** | **6** |
| **7** | **8** | **9** |

1. What is output by the following code segment?

for (int [] row : mat)  
{  
 for (int n : row)  
 {  
 if(n % 2 == 1)  
 System.out.print(n + " ");  
 }  
}

1. 1 2 3 4 5 6 7 8 9
2. 2 4 6 8
3. 1 3 5 7 9
4. 0 0 0 0 0 0 0 0 0
5. What value is stored in the variable num after the following code has been executed?

int num = 0;  
  
 for (int r = 2; r >= 0; r--)  
 {  
 num += mat[mat.length – r - 1][r];

}

1. 6
2. 12
3. 15
4. 45
5. Consider the following method.

public void mystery()  
 {  
 for (int row = 0; row < mat.length - 1; row++)  
 {  
 for (int col = 0; col < mat[0].length; col++)  
 {  
 mat[row][col] = mat[row + 1][col];  
 }  
 }  
 }

Assume that mat has been instantiated and contains the following values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 1 | 3 | 4 | 2 |
| 1 | 8 | 7 | 5 | 3 |
| 7 | 4 | 6 | 9 | 2 |
| 3 | 8 | 1 | 2 | 4 |
| 5 | 6 | 7 | 0 | 3 |

What values does mat contain after a call to mystery?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 8 | 7 | 5 | 3 |
| 7 | 4 | 6 | 9 | 2 |
| 3 | 8 | 1 | 2 | 4 |
| 5 | 6 | 7 | 0 | 3 |
| 5 | 6 | 7 | 0 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 6 | 7 | 0 | 3 |
| 3 | 8 | 1 | 2 | 4 |
| 7 | 4 | 6 | 9 | 2 |
| 1 | 8 | 7 | 5 | 3 |
| 4 | 1 | 3 | 4 | 2 |

1. B)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 8 | 7 | 5 | 3 |
| 1 | 8 | 7 | 5 | 3 |
| 1 | 8 | 7 | 5 | 3 |
| 1 | 8 | 7 | 5 | 3 |
| 1 | 8 | 7 | 5 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 8 | 7 | 5 | 3 |
| 7 | 4 | 6 | 9 | 2 |
| 3 | 8 | 1 | 2 | 4 |
| 5 | 6 | 7 | 0 | 3 |
| 4 | 1 | 3 | 4 | 2 |

C) D)

1. Consider the following method.

public void bam()  
 {  
 for (int[] row : mat)

{

for(int col = 0; col < row.length; col++)

{

int temp = row[col];

row[col] = row[row.length - 1 - col];

row[row.length - 1 - col] = temp;

}

}  
 }

Assume that mat has been instantiated and contains the following values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |

What values does mat contain after a call to bam?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 4 | 3 | 2 | 1 |
| 5 | 4 | 3 | 2 | 1 |
| 5 | 4 | 3 | 2 | 1 |
| 5 | 4 | 3 | 2 | 1 |
| 5 | 4 | 3 | 2 | 1 |

A) B)

C) D)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 5 | 5 | 5 | 5 |
| 4 | 4 | 4 | 4 | 4 |
| 3 | 3 | 3 | 3 | 3 |
| 2 | 2 | 2 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 |

**Free Response**

In this question you will write the code for three methods in a MagicSquare class.

An *n x n magic square* is a square array of n2 distinct integers arranged such that *n* numbers along any row, column, major diagonal, or minor diagonal have the same sum. Shown below are two magic squares.

|  |  |  |
| --- | --- | --- |
| 8 | 1 | 6 |
| 3 | 5 | 7 |
| 4 | 9 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 16 | 2 | 3 | 13 |
| 5 | 11 | 10 | 8 |
| 9 | 7 | 6 | 12 |
| 4 | 14 | 15 | 1 |

The MagicSquare class has two private instance variables, a two-dimensional array mat, and a final variable SIZE. In the matrices shown above, the one on the left has SIZE 3, while the one on the right has SIZE 4.

The MagicSquare class in shown below.

public class MagicSquare

{

private int[][] mat;

/\*\*

\* Postcondition: mat has been instantiated

\* mat.length > 0, mat[0].length > 0

\*/

public MagicSquare(int[][] matrix)

{ /\* implementation not shown \*/ }

/\*\*

\* Precondition: 0 <= row < mat.length

\* @param row a row in the matrix

\* @return the sum of elements in row

\*/

public int sumRow(int row)

{ /\* implemented in part(a) \*/ }

/\*\*

\* Precondition: 0 <= col < mat[0].length

\* @param col a column in the matrix

\* @return the sum of elements in col

\*/

public int sumCol(int col)

{ /\* implementation not shown \*/ }

/\*\*

\* @return the sum of elements in the major diagonal

\*/

public int sumMajorDiag()

{ /\* implementation not shown \*/ }

/\*\*

\* @return the sum of elements in the minor diagonal

\*/

public int sumMinorDiag()

{ /\* implemented in part(b) \*/ }

/\*\*

\* Precondition: mat is a square matrix of integers,

\* which may or may not be a magic square.

\* @return true if mat is a magic square, false otherwise

\*/

public boolean isMagic()

{ /\* implemented in part(c) \*/ }

}

1. Write the MagicSquare method sumRow. This method should return the sum of elements in row, its parameter. Your code should work for any square matrix, irrespective of whether it’s a magic square or not.

Complete method sumRow below.

/\*\*

\* Precondition: 0 <= row < mat.length

\* @param row a row in the matrix

\* @return the sum of elements in row

\*/

public int sumRow(int row)

1. Write the MagicSquare method sumMinorDiag. This method should return the sum of the elements in the minor diagonal. The minor diagonal extends from the top right corner of the matrix to the lower left corner, as shown in the two matrices below.

|  |  |  |  |
| --- | --- | --- | --- |
| 16 | 2 | 3 | 13 |
| 5 | 11 | 10 | 8 |
| 9 | 7 | 6 | 12 |
| 4 | 14 | 15 | 1 |

|  |  |  |
| --- | --- | --- |
| 8 | 1 | 6 |
| 3 | 5 | 7 |
| 4 | 9 | 2 |

Your code should work for any square matrix, irrespective of whether it’s a magic square or not.

Complete method sumMinorDiag below.

/\*\*

\* @return the sum of elements in the minor diagonal

\*/

public int sumMinorDiag()

1. Write the MagicSquare method isMagic. This method should return true if mat is a magic square, false otherwise. In writing isMagic you may use any methods in the MagicSquare class, and assume they work as specified.

Information repeated for your convenience

public class MagicSquare

private int[][] mat  
public int sumRow(int row)  
public int sumCol(int col)  
public int sumMajorDialog()  
public int sumMinorDialog()

/\*\*

\* Precondition: mat is a square matrix of integers,

\* which may or may not be a magic square.

\* @return true if mat is a magic square, false otherwise

\*/

public boolean isMagic()