edhesive

Unit 8 Exam – Alternative Version – Solutions

1. Consider the following method:

```
public static int mystery(int[] nums)
{
  int total = 0;
  for (int k = 0; k < nums.length / 3; k++)
  {
    total = total + nums[k];
  }
  return total;
}
```

Assume that the array test has been declared and initialized as follows.

```
int[] test = {1, 3, 2, 5, 8, 7, 0, 9, 2, 4};
```

What value will be returned as a result of the call `mystery(test)`?

a. 4 ➜ This is incorrect. The elements in the array with indices less than one-third the length of the array (using integer floor division) will be summed. The number of integers summed is equal to one-third the length of the array (rounded down to the nearest integer), as the first element in the array has an index of 0. This total is therefore too low.

b. 6 ➜ This is correct. The length of the array test, is 10. Therefore when the procedure is called, with this array as the argument for the parameter nums, the value of nums.length/3 will be equal to 10/3. Using integer division this evaluates to 3, therefore the for loop runs when k = 0, k = 1 and k = 2 (i.e the values of k less than 3). This means the elements in the array at these indices will be added to total. Since the first element in the array has an index of 0, the numbers summed are 1, 3 and 2, giving a total of 6.

c. 11 ➜ This is incorrect. The elements in the array with indices less than one-third the length of the array (using integer floor division) will be summed. The number of integers summed is equal to one-third the length of the array (rounded down to the nearest integer), as the first element in the array has an index of 0. This total is therefore too high.

d. 13 ➜ This is incorrect. The elements in the array with indices less than one-third the length of the array (using integer floor division) will be summed. The number of integers summed is equal to one-third the length of the array (rounded down to the nearest integer), as the first element in the array has an index of 0. This total is therefore too high.

e. 26 ➜ This is incorrect. Not every element in the array will be summed, as the value of the index variable k is only increased until the condition k < nums.length / 3 is satisfied.

2. Consider the following code segment.

```
int[][] matrix = new int[7][15];
```

Which of the following correctly gives the number of rows in the two-dimensional array matrix?

a. `matrix[0]` ➜ This gives the contents of the first row as a 1-d array.
b. `matrix.length` ➜ Correct. This gives the number of rows in the array.
c. `matrix[matrix.length - 1]` ➜ This gives the contents of the last row as a 1-d array.
d. `matrix[0].length` ➜ This gives the number of columns in the array
e. `matrix.length[0]` ➜ This will cause an error.

3. Consider the following declaration for a two-dimensional array.

```
int[][] grid = new int[5][3];
int c = 0;
for (int i = 0; i < grid.length; i++)
{
  for (int j = 0; j < grid[i].length; j++)
  {
    grid[i][j] = c;
    c++;
  }
}
```

What element is displayed when the following line of code is executed?

```
System.out.println(grid[2][1]);
```

a. 4 ➜ This is the element in row 1, column 0.
b. 5 ➜ This is the element in row 1, column 1.
c. 7 ➜ This is the element in row 2, column 1. The variable c begins as 0, (placed at grid[0][0]), and is increased by 1 as each row is filled left to right and top to bottom. The first element of row 1 is therefore 4, then row 2 begins with 7. This means grid[2][1] has a value of 7.
d. 8 ➜ This is the element in row 2, column 2.
e. 10 ➜ This is the element in row 3, column 0.

4. Consider the following method intended to swap the first and last rows in a two-dimensional array:

```
public static void swapRow(int[][] arr)
{
  /* missing code */
}
```

Which of the following correctly replaces `/* missing code */`?

```
a. for (int k = 0; k < arr[0].length; k++)
   {
     int last = arr.length - 1;
     arr[0][k] = arr[last][k];
     arr[last][k] = arr[0][k];
   }
```

This will place the values from the last row into the first row but will not place the original values from the first row into the last row.

```
b. for (int k = 0; k < arr[0].length; k++)
   {
     int last = arr.length;
     arr[0][k] = arr[last][k];
     arr[last][k] = arr[0][k];
   }
```

This will cause an ArrayIndexOutOfBoundsException to be thrown.

```
c. for (int k = 0; k < arr[0].length; k++)
   {
     int last = arr.length - 1;
     int temp = arr[0][k];
     arr[0][k] = arr[last][k];
     arr[last][k] = temp;
   }
```

This will work correctly. The indices of the elements for one row of the array are iterated through, and each time the element at that index in row 0 is swapped with the element at the same index in row a.length - 1 (the last row). This is done by means of a temporary variable to facilitate the swap.

```
d. for (int k = 0; k < arr[0].length; k++)
   {
     int last = arr.length;
     int temp = arr[0][k];
     arr[0][k] = arr[last][k];
     arr[last][k] = temp;
   }
```

This will cause an ArrayIndexOutOfBoundsException to be thrown.

e. None of the above.

This is incorrect; choice c is correct.

5. Consider the following method declaration.

```
public static void increment(int[][] a)
{
  for (int r = 0; r < a.length; r++)
  {
    for (int c = 0; c < a[0].length; c++)
    {
      if (a[r][c] >= 0)
      {
        a[r][c]--;
      }
      else
      {
        a[r][c]++;
      }
    }
  }
}
```

Assume the 2D array, `matrix`, has been initialized to the following values:

```
4 6 -15
5 11 21
-11 -10 3
4 -10 -18
-21 14 -23
```

What is the value of `matrix` after the method call `increment(matrix)`?

a.
```
4  6  -15
5 11 21
-11 -10 3
4 -10 -18
-21 14 -23
```
➜ An array is a class data type, so matrix will be modified by the method call.

b.
```
3 5 -15
4 10 20
-11 -10 2
3 -10 -18
-21 13 -23
```
➜ Both positive and negative values in matrix are changed.

c.
```
3 5 -16
4 10 20
-12 -11 2
3 -11 -19
-22 13 -24
```
➜ The method increases rather than decreases each negative value by one.

d.
```
4 6 -14
5 11 21
-10 -9 3
4 -9 -17
-20 14 -22
```
➜ Both positive and negative values in matrix are changed by the method.

e.
```
3 5 -14
4 10 20
-10 -9 2
3 -9 -17
-20 13 -22
```
➜ The method decreases each positive value by one, and increases each negative value by one.

6. The following code is meant to find the smallest value in an array.

```
double[][] list = /* Initialization not Shown */
double m = /* Initialization not Shown */
for (int i = 0; i < list.length; i++)
{
  for (int j = 0; j < list[i].length; j++)
  {
    if (list[i][j] < m)
    {
      m = list[i][j];
    }
  }
}
System.out.println(m);
```

What could m be set to in order for the code to work as intended?

a. 100000 ➜ If all elements are larger than 100000 this will not work as the method will return 100000.

b. 0 ➜ If all elements are larger than 0 this will not work as the method will return 0.

c. -100000 ➜ If all elements are larger than -100000 this will not work as the method will return -100000.

d. Double.MIN_VALUE ➜ If all elements are larger than Double.MIN_VALUE (which is 2^-1074) this will not work as the method will return this value.

e. Double.MAX_VALUE ➜ This is the largest possible value for a double, so every element in the list will be smaller than or equal to this. Therefore the value m will definitely be set to the minimum element in the array when it is encountered.

7. Consider the following code.

```
int[][] matrix = new int[4][5];
```

Suppose we want to initialize matrix to the following rows and columns.

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
```

Which of the options below correctly initializes matrix?

I.
```
for (int i = 0; i < matrix.length; i++)
{
  for (int j = 0; j < matrix[i].length; j++)
  {
    matrix[i][j] = j;
  }
}
```

II.
```
for (int i = 0; i < matrix.length; i++)
{
  for (int j = 0; j < matrix[i].length; j++)
  {
    matrix[i][j] = i;
  }
}
```

III.
```
for (int i = 0; i < matrix.length; i++)
{
  for (int j = 0; j < matrix[i].length * 2; j += 2)
  {
    matrix[i][j] = j;
  }
}
```

a. I only ➜ Incorrect, this will output 0 1 2 3 4 for all rows
b. II only ➜ Correct, this code sets row one to 0 0 0 0 0, and then each row after to a value one higher, 1 1 1 1 1, 2 2 2 2 2, etc.
c. III only ➜ Incorrect, This sets the values in the first row to 0 2 4 6 8 and each row after to the same thing
d. I and II only ➜ I is incorrect.
e. I, II and III ➜ II is the only correct choice.

8. Consider the following code:

```
int[][] grid = /* code not shown */;
```

Which of the following could be used to calculate how many cells are in the array?

a. `grid.length[0] * grid[0].length` ➜ This multiplies the number of columns by itself.
b. `grid[0].length * grid.length` ➜ This multiplies the number of columns by the number of rows, giving the total number of cells.
c. `grid[0].length` ➜ This is the number of columns in the array.
d. `grid.length` ➜ This is the number of rows in the array.
e. `grid.length * grid.length` ➜ This multiplies the number of rows by itself.

9. Consider the following method that is intended to return true if all the Strings in the ArrayList start with an uppercase letter:

```
public static boolean capitalized(String[][] a)
{
  /* Missing Code */
}
```

Which of the following could replace `/* Missing Code */` so that the method works as intended? (You may assume that all the Strings in 2-D array contain only letters.)

I.

```
for (String[] s : a)
{
  for (String st : s)
  {
    if (!st.toUpperCase().substring(0, 1).equals(st.substring(0, 1)))
    {
      return true;
    }
  }
}
return false;
```

II.

```
for (String[] s : a)
{
  for (String st : s)
  {
    if (!st.toUpperCase().substring(0, 1).equals(st.substring(0, 1)))
    {
      return false;
    }
  }
}
return true;
```

III.

```
for (String[] s : a)
{
  for (String st : s)
  {
    if (st.toUpperCase().substring(0, 1).equals(st.substring(0, 1)))
    {
      return false;
    }
  }
}
return true;
```

a. I only ➜ Incorrect; see explanation for choice b.

b. II only ➜ I does the opposite of what is intended, returning true if any string starts with a lower-case letter. III returns true only if every single string starts with a lower-case letter. Only II returns true when every single string starts with an upper-case letter, and false otherwise.

c. III only ➜ Incorrect; see explanation for choice b.
d. I and III only ➜ Incorrect; see explanation for choice b.
e. II and III only ➜ Incorrect; see explanation for choice b.

10. What does the following segment of code do?

```
int[][] a = /* initialization not shown */;
int sum = 0;
for (int i = 0; i < a.length; i++)
{
  for (int j = 0; j < a[0].length; j++)
  {
    if (i % 2 == 1)
    {
      sum += a[i][j];
    }
  }
}
```

a. It finds the sum of every other element in the array. ➜ i is the index of the rows, not the columns.
b. It finds the sum of the elements in the odd rows in the array. **➜ The variable i gives the row numbers of the array. Each element is only summed if i % 2 == 1 is true: in other words the row number is odd.**
c. It finds the sum of the odd elements in the array. ➜ i, tested for divisibility by 2, is an index, not an element.
d. It finds the sum of all elements in the array. ➜ Only elements for which the condition i % 2 == 1 is true are summed.
e. It finds the sum of the elements in the odd columns in the array. ➜ i is the index of the rows, not the columns.

11. Which option best describes what the following algorithm does?

```
int a [][] = /* initialization not shown */;
int j = 1;
for (int i = 0; i < a[0].length; i++)
{
  int temp = a[j + 1][i];
  a[j + 1][i] = a[j][i];
  a[j][i] = temp;
}
```

a. Swaps columns 1 and 2. ➜ This is incorrect.
b. Swaps columns 2 and 3. ➜ This is incorrect.
c. Swaps rows 1 and 2 **➜ The values j and j+1 are used as constant row-numbers in the statements accessing the array. Since j is set to 1, this means the values in the rows with indices 1 and 2 are swapped.**
d. Swaps rows 2 and 3. ➜ This is incorrect.
e. Initializes the values in the ➜ The array is initialized at the start of the code segment. The values in

array. the array are then modified.

12. You need a method to find the minimum value in every row of an array. Which of the following methods works as intended?

I.
```
public static int[] findMinList(int[][] a)
{
  int[] temp = new int[a.length];
  for (int i = 0; i < a.length; i++)
  {
    int min = a[i][0];
    for (int j = 0; j < a[0].length; j++)
    {
      if (a[i][j] < min)
      {
        min = a[0][j];
      }
    }
    temp[i] = min;
  }
  return temp;
}
```

II.
```
public static int[] findMinList(int[][] a)
{
  int[] temp = new int[a.length];
  for (int i = 0; i < a.length; i++)
  {
    int min = a[i][0];
    for (int j = 0; j < a[0].length; j++)
    {
      if (a[i][j] < min)
      {
        min = a[i][j];
      }
    }
    temp[i] = min;
  }
  return temp;
}
```

III.
```
public static int[] findMinList(int[][] a)
{
  int[] temp = new int[a.length];
  for (int i = 0; i < a.length; i++)
  {
    int min = a[i][0];
    for (int j = 0; j < a[0].length; j++)
    {
      if (a[i][j] < a[0][j])
      {
        min = a[i][j];
      }
    }
    temp[i] = min;
  }
  return temp;
}
```

a. I only ➜ This is incorrect; see explanation for choice b.

b. II only ➜ I adds elements from row 0 only to the returned array. III compares elements in a row to the first element of that row, and not the minimum currently found. Only II goes through the elements of each row, updating a min value each time a smaller element is found and adds this to the array once the end of the row is reached.

c. III only ➜ This is incorrect; see explanation for choice b.

d. II and III only ➜ This is incorrect; see explanation for choice b.

e. I, II and III ➜ This is incorrect; see explanation for choice b.

13. Consider the following code segment.

```
int[][] mat = new int[3][5];
for (int j = 0; j < mat.length; j++)
{
  for (int k = 0; k < mat[0].length; k++)
  {
    mat[j][k] = (k + j) * 2;
  }
}
```

What are the contents of `mat` after the code segment has been executed?

a. {{0, 2, 4},
{2, 4, 6},
{4, 6, 8},
{6, 8, 10},
{8, 10, 12}}
This is incorrect. The matrix mat is initialized with 3 rows and 5 columns (3 arrays of int values, with 5 int values in each one), rather than 5 rows and 3 columns.

b. {{2, 4, 6},
{4, 6, 8},
{6, 8, 10},
{8, 10, 12},
{10, 12, 14}}
This is incorrect. The matrix mat is initialized with 3 rows and 5 columns (3 arrays of int values, with 5 int values in each one), rather than 5 rows and 3 columns.

c. {{0, 2, 4, 6, 8},
{0, 2, 4, 6, 8},
{0, 2, 4, 6, 8}}
This is incorrect. See explanation for choice d.

**d. {{0, 2, 4, 6, 8},
{2, 4, 6, 8, 10},
{4, 6, 8, 10, 12}}
This is correct. The matrix mat is initialized with 3 rows and 5 columns (3 arrays of int values, with 5 int values in each one). The outer loop uses the variable j iterates through each row of mat, and the inner loop uses the variable k to iterate through each column within each row of mat. Each element of mat is given the value (k + j) * 2: therefore the digit in each place is the sum of the row and column times 2.**

e. {{4, 6, 8, 10, 12},
{6, 8, 10, 12, 14},
{8, 10, 12, 14, 16}}
This is incorrect. See explanation for choice d.

14. Consider the following method

```
public static int[][] operation(int[][] mat, int c)
{
  int[][] result = new int[mat.length][mat[0].length];
  for (int j = 0; j < mat.length; j++)
  {
    for (int k = 0; k < mat[j].length; k++)
    {
      if (k >= c && j >= c)
      {
        result[j][k] = 0;
      }
      else
      {
        result[j][k] = mat[j][k];
      }
    }
  }
  return result;
}
```

The following code segment appears in another method in the same class.

```
int[][] m = {{1, 2, 4, 2},
             {3, 3, 5, 1},
             {2, 1, 3, 1},
             {1, 3, 2, 4}};

int[][] grid = operation(m, 2);
```

Which of the following represents the contents of `grid` as a result of executing the code segment?

a. {{1, 2, 4, 2},
{3, 3, 5, 1},
{2, 1, 3, 1},
{1, 3, 2, 4}}

This is incorrect. In the operation method, not every element of the result matrix is set equal to the corresponding element in the argument mat.

b. {{1, 2, 4, 2},
{3, 3, 5, 1},
{2, 1, 3, 1},
{1, 3, 2, 0}}

This is incorrect. Remember that in the if, it is checking both j and k to be greater than or equal to 2.

**c. {{1, 2, 4, 2},
{3, 3, 5, 1},
{2, 1, 0, 0},
{1, 3, 0, 0}}**

This is correct. The values that are in a row and column that are both greater than or equal to 2 are changed to 0.

d. {{0, 0, 0, 0},
{0, 0, 0, 0},
{0, 0, 3, 1},
{0, 0, 2, 4}}

This is incorrect. This does the opposite of what it should do, changing the values of elements less than

e. {{0, 0, 0, 0},
{0, 0, 0, 0},
{0, 0, 0, 0},
{0, 0, 0, 0}}

This is incorrect. Only certain values are changed to 0s.

row and column 2.

15. Consider the following definition

```
String[][] letters = {{"c", "a", "t"},
                      {"d", "o", "g"}};
```

Which of the following code segments produces the output ”tacgod”?

a.
```
for (String l : letters)
{
  System.out.print(l);
}
```
This is incorrect. Elements of the letters array are themselves arrays of the String[] type. Therefore attempting to iterate through them using a String variable will cause an error.

b.
```
for (String[] row : letters)
{
 for (String l : row)
 {
   System.out.print(l);
 }
}
```
This is incorrect. This segment iterates through each row of letters in forward order, and for each row through the String objects in that row in order. This means the segment prints “catdog”.

c.
```
for (String[] row : letters)
{
 for (int k = row.length - 1; k >= 0; k--)
 {
   System.out.print(row[k]);
 }
}
```
This is correct. The outer for loop iterates through the arrays of Strings contained in letters in forward order. Within each of these arrays the inner loop iterates backwards through the Strings, printing each one. This results in the segment printing “tacgod”.

d.
```
for (int j = letters.length - 1; j >= 0; j--)
{
  for (int k = letters[j].length - 1; k >= 0; k--)
  {
    System.out.print(letters[j][k]);
  }
}
```
This is incorrect. This segment iterates through each row of letters in a backwards order from last to first, and for each row through the String objects in that row in backwards order too. This means the segment prints “godtac”.

e.
```
for (int j = 0; j < letters.length; j++)
{
  for (int k = 0; k < letters[j].length; k++)
  {
    System.out.print(letters[j][k]);
  }
}
```

This is incorrect. This segment iterates through each row of letters in forward order, and for each row through the String objects in that row in order. This means the segment prints “catdog”.

16. Consider the following method.

```
public String mystery(String s1, String s2)
{
  String output = "";
  int len;

  if (s1.length() < s2.length())
  {
    len = s1.length();
  }
  else
  {
    len = s2.length();
  }

  for (int k = 0; k < len; k++)
  {
    output += s1.substring(k, k+1);
    output += s2.substring(k, k+1);
  }

  return output;
}
```

What is returned as a result of the call `mystery(“Sally”, “Joe”)`?

a.	**SJaole**	**➜ This is correct. The if/else structure sets the value of len to whichever is the shorter of the lengths of the two strings. As the string Sally has length 5, and Joe has length 3, len is set to 3. The command s1.substring(k, k+1) returns the substring starting between letter k (inclusive) and letter k+1 (exclusive) of s1. This means in effect it returns the single letter at index k. The for loop therefore adds a letter of s1, then a letter of s2 to the output string each time it runs. As it first runs with a value of k = 0, and stops when it reaches a value of 3, it will add three letters from each string in total, taking from each in turn. Therefore for inputs of “Sally” and “Joe” the method will return SJaole.**
b.	JSoael	➜ This is incorrect. Each time the for loop iterates, the letter from s1 (Sally) is added to output before the letter from s2 (Joe), This means that the string should begin with a letter from Sally, and end with a letter from Joe.
c.	SJaolel	➜ This is incorrect, the number of letters from each of the two strings s1 and s2 added to output will be the same, as each run of the for loop adds a letter from each.
d.	SallyJoe	➜ This is incorrect. Each time the for loop runs it adds a single letter from each of the strings s1 and s2 in turn to output. This means the letters from each word will alternate in the final string rather than one string being followed by the other.
e.	Nothing is returned because an IndexOutOfBounds Exception is thrown.	➜ This is incorrect. The value of k is only increased up to a maximum value of len - 1 while the for loop is running: once it has a value of len, the loop exits. As len is set to the minimum of the two string lengths by the if/else structure, and the command s2.substring(k, k+1) only accesses characters of the string which are strictly less than

k+1, the program will not attempt to access a character of either string which does not exist.

17. Consider the following code segment.

```
int[][] mat = new int[4][4];
int fill = 0;

for (int[] row : mat)
{
  for (int k = 0; k < row.length; k++)
  {
    row[k] = fill;
    fill++;
  }
}
System.out.println(mat[1][2]);
```

What is printed as a result of executing the code segment?

a. 1 ➜ This is incorrect. This will be the value of the element mat[0][1].
b. 2 ➜ This is incorrect. This will be the value of the element mat[0][2].
c. 5 ➜ This is incorrect. This will be the value of the element mat[1][1].
d. 6 ➜ This is correct. The array mat contains 3 arrays with indices 0, 1 and 2, each of which contains 3 ints with indices 0, 1 and 2. The effect of the code is to select each of the arrays in mat (these are given the variable row), and fill each element of each of these with an increasing integer, fill. In this way, the first array in mat, mat[0], is set to {0, 1, 2}, the second, mat[1], is set to {3, 4, 5}, and the third mat[2], is set to {6, 7, 8}. Therefore at the end the expression mat[1][2] returns the third element of the second array in mat, which is 5.
e. 10 ➜ This is incorrect. This will be the value of the element mat[2][2].

18. Consider the following method.

```
public static int operation(int[][] mat)
{
  int currentVal = mat[0][0];
  int result = 0;

  for (int j = 0; j < mat.length; j++);
  {
    for (int k = 0; k < mat[j].length; k++)
    {
      /* missing code */
    }
  }
  return result;
}
```

Which of the following should replace `/* missing code */` so that the method returns the index of the row which contains the largest value in the two-dimensional array?

a.
```
if (mat[j][k] > currentVal)
{
  currentVal = mat[j][k];
  result = j;
}
```
This is correct. The value j iterates through the indices for each row element in the two-dimensional array and the value k iterates through the indices for each element in a row array. Therefore the value of result should be updated to the value of j when a new maximum element is found. The most recently found maximum element is stored in the currentVal variable, therefore this should be updated to the value mat[j][k] when this is larger than its current value.

b.
```
if (mat[j][k] > currentVal)
{
  currentVal = mat[j][k];
  result = k;
}
```
This is incorrect. The value j iterates through the indices for each row element in the two-dimensional array and the value k iterates through the indices for each element in a row array. Therefore if this code is used the column index will be returned rather than the row index.

c.
```
if (mat[j][k] > currentVal)
{
  currentVal = j;
  result = mat[j][k];
}
```
This is incorrect. The value returned by the method is result. Therefore this should be set to one of the index variables, and not the value of an element in the two-dimensional array.

d.
```
if (mat[j][k] < currentVal)
{
  currentVal = mat[j][k];
  result = j;
}
```
This is incorrect. Since it is an index of the maximum value that is sought, not the minimum, the values of the variables should be updated when a new element is found which is larger than the current maximum, rather than one which is larger.

e.
```
if (mat[j][k] > currentVal)
{
  currentVal = k;
  result = mat[j][k];
}
```
This is incorrect. The value returned by the method is result. Therefore this should be set to one of the index variables, and not the value of an element in the two-dimensional array.

19. Consider the following code segment.

```
int[][] mat = new int[4][6];
for (int r = 0; r < mat.length - 1; r++)
{
  for (int c = 0; c < mat[0].length - 1; c++)
  {
    if(c < 3 - r || c > 3 + r)
    {
      mat[r][c] = 1;
    }
    else
    {
      mat[r][c] = 0;
    }
  }
}
```

Which of the following represents mat after this code segment is executed?

a.

1	1	1	1	1	1
1	1	0	0	1	1
1	0	0	0	0	1
0	0	0	0	0	0

This is incorrect. Consider the first iteration of the outer for loop, when r = 0. The statement c < 3 - r || c > 3 + r is not satisfied when c = 3. Therefore not every element of the first row of the 2-dimensional array will be set to 1: at least one will be set to 0.

b.

1	1	1	1	1	1
1	1	1	0	1	1
1	1	0	0	0	1
0	0	0	0	0	0

Incorrect; see explanation for choice a.

c.

1	1	0	1	1	1
1	0	0	0	1	1
0	0	0	0	0	1
0	0	0	0	0	0

This is incorrect. Consider the first iteration of the outer for loop, when r = 0. The statement c < 3 - r || c > 3 + r is satisfied when c is 2 as 2 < 3. Therefore the element in the third column of the first row will be 1, not 0.

d.

1	**1**	**1**	**0**	**1**	**1**
1	**1**	**0**	**0**	**0**	**1**
1	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**0**	**0**	**0**

This is correct. On the first iteration of the outer for loop, when r = 0 the statement c < 3 - r || c > 3 + r is satisfied for every value of c except when c is 3. Therefore the 4th element of the first row is the only 0, all others are 1s. For each subsequent iteration of the for loop, there will be one fewer value of c which satisfies c < 3 - r, and one fewer value of c which satisfies c > 3 + r. Therefore the subsequent rows will contain 2 more 0s, equally expanding to the left and right until the edge of the matrix is reached.

e.

1	1	0	0	1	1
1	0	0	0	0	1
0	0	0	0	0	0
0	0	0	0	0	0

This is incorrect. Consider the first iteration of the outer for loop, when r = 0. The statement c < 3 - r || c > 3 + r is not satisfied when c = 3. Therefore not every element of the first row of the 2-dimensional array will be set to 1: at least one will be set to 0.

20. Consider the following method, which is intended to return an array which contains the minimum elements in each of the rows of a 2-dimensional array.

```
/** @param  mat a 2-dimensional array
 *  @return an array which contains the minimum elements of each row
 * in mat.
 */
public double[] minRows(double[][] mat)
{
  double[] mins = new double[mat.length];
  for (int k = 0; k < mat.length; k++)
  {
    double localMin = mat[k][0];
    for (double num : mat[k])
    {
      /* missing code */
    }
    mins[k] = localMin;
  }
  return mins;
}
```

Which of the following could be used to replace `/* missing code */` so that `minRows` will work as intended?

```
a. if (num < localMin)
   {
     localMin = mat[k][num];
   }
```

This is incorrect. A statement in this code attempts to access mat[k][num], however num is not an int index variable, it is a double from the array mat[k], therefore this code will cause an error.

```
b. if (num < localMin)
   {
     localMin = num;
   }
```

This is correct. The for loop sets the value of num to each double in the array mat[k] in turn. This is equivalent to setting it to each variable in a row in turn. Therefore each successive value of num should be compared to the current minimum value, localMin and this should be updated to num if this is smaller.

```
c. if (mat[num] < localMin)
   {
     localMin = mat[num];
   }
```

This is incorrect. A statement in this code attempts to set the double localMin to a value of mat[num], however num is not an int index variable, it is a double from the array mat[k], and even were it a valid index, the result would be another array, as mat is an array containing arrays.

```
d. if (mat[k][num] < localMin)
   {
     localMin = num;
   }
```

See explanation for choice a.

```
e. if (mat[k][num] < mins[k])
   {
     localMin = mat[k][num];
   }
```

See explanation for choice a.