03 Java Creating Using Arrays

Problema 1

Consider the following code:

```
public class ArrayTest {

    static int[][] table = new int[2][3];

    public static void init() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to initialize

            }
        }
    }

    public static void multiply() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to multiply

            }
        }
    }
}
```

Which of the following options can be used in the code above so that the init method initializes each table element to the sum of its row and column number and the multiply method just multiplies the element value by 2?

table[x, y] = x+y;
and
table[x, y] = table[x, y]*2;

table[x][y] = x+y;
and
table[x][y] = table[x][y]*2;

table[[x] [y]] = x+y;
and
table[[x] [y]] = table[[x] [y]]*2;

table(x, y) = x+y;
and
table(x, y) = table(x, y)*2;

La respuesta es [0][1].
This code snippet illustrates correct syntax for accessing array elements in a multi dimensional array. All other options are syntactically incorrect and will not compile.

Prolema 2

What would be the result of compiling and running the following program?

```
class SomeClass{
   public static void main(String args[]){
      int size = 10;
      int[] arr = new int[size];
      for (int i = 0 ; i < size ; ++i) System.out.println(arr[i]);
   }
}
```

You have to select one option:

- The code will fail to compile, because the int[] array declaration is incorrect.
- The program will compile, but will throw an `IndexArrayOutOfBoundsException` when run.
- The program will compile and run without error, and will print nothing.
- The program will compile and run without error and will print null ten times.
    - Here, all the array elements are initialized to have 0.
- The program will compile and run without error and will print 0 ten times.
    - It correctly will declare and initialize an array of length 10 containing int values initialized to have 0.

Explanation:
// Hace un array de int de dimension 10, sin valor inicial por lo que al ser primitivo es 0 en los 10 espacios del array, iterara sólo '0' 10 veces.

Elements of Arrays of primitive types are initialized to their default value (i.e. 0 for integral types, 0.0 for float/double and false for boolean)
Elements of Arrays of non-primitive types are initialized to null.

Problema 3

Which of the following option(s) correctly declare(s)
a variable that can directly reference an array of 10 ints?

You have to select 2 options:

- int[ ] iA
- int[10] iA
    - Size of the array is NEVER specified on the Left Hand Side.
- int iA[ ]
- Object[ ] iA
    - Here, iA is an array of Objects. It cannot hold an array of integers.
- Object[10] iA
    - Size of the array is NEVER specified on the LHS.

Explanation:
Note that an array of integers IS an Object :
Object obj = new int[]{ 1, 2, 3 }; // is valid.
But it is not an array of objects.
Object[] o = new int[10]; // is not valid.

Difference between the placement of square brackets:
int[] i, j; //here i and j are both array of integers.
int i[], j; //here only i is an array of integers. j is just an integer.

Problema 4

Given:

```
String[][] matrix = new String[2][2];
matrix[0][0] = "petrol";
matrix[1][0] = "diesel";
matrix[0][1] = "manual";
matrix[1][1] = "auto";
```

Which option will print petrol:diesel:manual:auto:?

`for(String[] row : matrix)`
`  for (String spec : row)`
`  System.out.print(spec+":");`
 	
This will print petrol:manual:diesel:auto:.
	`for(int i = 0; i<2; i++) {`
	`  for(int j = 0; j<2; j++){`
	
	`System.out.print(matrix[i][j]+":");`
	`  }`
	`}`
	This will print petrol:manual:diesel:auto:.

`for(int i = 0; i<2; i++) {`
`  for(int j = 0; j<2; j++){`
 	
`System.out.print(matrix[j][i]+":");`
`  }`
`}`
	`System.out.println("");`
	`for(int i = 1; i<2; i++) {`
	`  for(int j = 1; j<2; j++){`
	
	`System.out.print(matrix[j][i]+":");`
	`  }`
	`}`

`System.out.println("");`
`for(int i = 1; i<2; i++) {`
`  for(int j = 1; j<2; j++){`
 	
`System.out.print(matrix[i][j]+":");`
`  }`
`}`
This will print just one element auto.
	`for(int i = 2; i>0; i--) {`
	`  for(int j = 0; j<2; j++){`
	
	`System.out.print(matrix[j][i]+":");`
	`  }`
	`}`
	It will throw an ArrayIndexOutOfBoundsException because when i is 2, matrix[0][2] will be trying to access the third element of the array referred to by matrix[0], which is out of bounds.

Explanation:
Array indexing starts with 0. Thus, the given code really creates the following array structure:

```
matrix ==>  { matrix[0] , matrix[1] }
matrix[0] ==>  { matrix[0][0],  matrix[0][1]} ==> { "petrol", "manual"}
matrix[1] ==>  { matrix[1][0],  matrix[1][1]} ==> { "diesel", "auto"}
```

// Fijate que te ayuda la declaracion, la j se cambia con i y debe empezar en '0' para mostrar

Problema 5

Which of the following are valid code fragments:

You have to select 2 options:

- `new Object[]{ "aaa", new Object(), new ArrayList(), 10};`
  - 10 is a primitive and not an Object but due to auto-boxing it will be converted into an Integer object and that object will then be stored into the array of Objects.

- `new Object[]{ "aaa", new Object(), new ArrayList(), {} };`
  - {} is not a valid way to create an Object here. However, it is valid while creating an array. For example, the following are valid: String[] sa = { }; //assigns a valid String[] object of length 0 to sa Object arr[][] = new Object[][] {new String[5], {} }; //assigns a valid Object[] object of length 0 to arr[1].

- `new Object[]{ "aaa", new Object(), new ArrayList(), new String[]{""} };`
  - Every array is an Object so new String[]{""} is also an Object and can be placed in an array of objects.

- `new Object[1]{ new Object() };`
  - You can't specify array length if you are initializing it at the same place.

Explanation:
1. An array of objects can store Objects of any class.
2. Primitives (i.e. int, byte, char, short, boolean, long, double, and float) are NOT objects.
3. An array (of primitives as well as of objects) is an Object.

Problema 6

What will be the result of attempting to compile and run the following class?

```
public class TestClass{
    public static void main(String args[ ] ){
       int i = 1;
       int[] iArr = {1};
       incr(i) ;
       incr(iArr) ;
       System.out.println( "i = " + i + "  iArr[0] = " + iArr [ 0 ] ) ;
    }
    public static void incr(int   n ) { n++ ; }
    public static void incr(int[ ] n ) { n [ 0 ]++ ; }
}
```

You have to select 1 option:

- The code will print i = 1 iArr[0] = 1
- The code will print i = 1 iArr[0] = 2
- The code will print i = 2 iArr[0] = 1
- The code will print i = 2 iArr[0] = 2
- The code will not compile. //The code will not compile.

Explanation:
Arrays are proper objects (i.e. iArr instanceof Object returns true) and Object references are passed by value (so effectively, it seems as though objects are being passed by reference). So the value of reference of iArr is passed to the method incr(int[] i); This method changes the actual value of the int element at 0.

Video: https://youtu.be/b35eNdLEsZw

Problema 7

Consider the following code:

```
//INSERT CODE HERE
a[0][0] = 1;
a[0][1] = 2;


a[1][0] = 3;
a[1][1] = 4;
a[1][2] = 5;
a[1][3] = 6;
```

What can be inserted independently in the above code so that it will compile and run without any error or exception?

```
int[][] a = new int[2][];
```

This will instantiate only the first dimension of the array. The elements in the second dimension will be null. In other words, a will be instantiated to two elements but a[0] and a[1] will be null and so a[0][0] (and access to all other such ints) will throw a NullPointerException.

```
int[][] a = new int[2][4];
```

This is correct because it will instantiate both the dimensions of the array. i.e. a will be initialized with 2 references to int arrays a[0] and a[1]. Further, the arrays pointed to by a[0] and a[1] will also be initialized with size 4.

```
int[][] a = new int[4][2];
```

This will initialize a to an array of size 4 and each element of this array will be initialized to an int array of size 2. Therefore, a[0][2], a[0][3], a[1][2], and a[1][3], will cause an ArrayIndexOutOfBoundsException to be thrown.

```
int[][] a = new int[2][];
a[0] = new int[2];
a[1] = new int[4];
```

Observe that this creates a jagged array. i.e. the elements in the second dimension of a are not of same length. The first element in the second dimension is only of length 2 while the second element is of length 4. Since the given code doesn't need a[0][2] and a[0][3], it is ok.

```
int[][] a = new int[4][];
a[0] = new int[2];
a[1] = new int[2];
```

In this case, a[1][2] and a[1][3] will cause an ArrayIndexOutOfBoundsException to be thrown.

Problema 8

Consider the following program...

```
class ArrayTest{
   public static void main(String[] args){
      int ia[][] = { {1, 2}, null };
      for (int i = 0; i < 2; i++)
         for (int j = 0; j < 2; j++)
            System.out.println(ia[i][j]);
   }
}
```

Which of the following statements are true?
You have to select 1 option:

- It will not compile.
- It will throw an `ArrayIndexOutOfBoundsException` at Runtime.
- It will throw a `NullPointerException` at Runtime.
- It will compile and run without throwing any exceptions.
- None of the above.

Explanation:
It will throw a NullPointerException for ia[1][0] because ia[1] is null. Note that null is not same as having less number of elements in an array than expected. If you try to access ia[2][0], it would have thrown ArrayIndexOutOfBoundsException because the length of ia is only 2 and so ia[2] tries to access an element out of that range. ia[2] is not null, it simply does not exist.

Problema 9

The following class will print 'index = 2' when compiled and run.

```
class Test{
   public static int[ ] getArray() {  return null;  }
   public static void main(String[] args){
      int index = 1;
      try{
         getArray()[index=2]++;
      }
      catch (Exception e){  }  //empty catch
      System.out.println("index = " + index);
   }
}
```

Ya have to select 1 option:

- True
- False

Explanation:
If the array reference expression produces null instead of a reference to an array, then a NullPointerException is thrown at runtime, but only after all parts of the array reference expression have been evaluated and only if these evaluations completed normally.

This means, first index = 2 will be executed, which assigns 2 to index. After that null[2] is executed, which throws a NullPointerException. But this exception is caught by the catch block, which prints nothing. So it seems like NullPointerException is not thrown but it actually is.

In other words, the embedded assignment of 2 to index occurs before the check for array reference produced by getArray().

In an array access, the expression to the left of the brackets appears to be fully evaluated before any part of the expression within the brackets is evaluated. Note that if evaluation of the expression to the left of the brackets completes abruptly, no part of the expression within the brackets will appear to have been evaluated.

Problema 10

Which of these array declarations and instantiations are legal?
You had to select 4 options

- `int[ ] a[ ] = new int [5][4] ;`
  - This will create an array of length 5. Each element of this array will be an array of 4 ints.

- `int a[ ][ ] = new int [5][4] ;`
  - This will create an array of length 5. Each element of this array will be an array of 4 ints.

- `int a[ ][ ] = new int [ ][4] ;`
  - The statement int[ ][4] will not compile, because the dimensions must be created from left to right.

- `int[ ] a[ ] = new int[4][ ] ;`
  - This will create an array of length 4. Each element of this array will be null. But you can assign an array of ints of any length to any of the elements. For example:

```
a[0] = new int[10];//valid
a[1] = new int[4];//valid
a[2] = new int[]; //invalid because you must specify the length
a[3] = new Object[] //invalid because a[3] can only refer to an
array of ints.
```

  - This shows that while creating a one dimensional array, the length must be specified but while creating multidimensional arrays, the length of the last dimension can be left unspecified. Further, the length of multiple higher dimensions after the first one can also be left unspecified if none of the dimensions are specified after it. So for example,

```
a[][][][] = new int[4][3][3][5];
```

    is same as

```
a[][][][] = new int[4][][][];
```

    (Note that the first dimension must be specified.)

  - Thus, multidimensional arrays do not have to be symmetrical.

- `int[ ][ ] a = new int[5][4] ;`
  - This will create an array of length 5. Each element of this array will be an array of 4 ints.

Explanation
The [] notation can be placed both before and after the variable name in an array declaration.

```
int[] ia, ba;  // here ia and ba both are int arrays.
int ia[], ba; //here only 'ia' is int array and ba is an int.
```

Multidimensional arrays are created by creating arrays that can contain references to other arrays .

Problema 11

Consider the following array definitions:

```
int[] array1, array2[];
int[][] array3;
int[] array4[], array5[];
```

Which of the following are valid statements?

You had to select 3 options

- `array2 = array3;`
- `array2 = array4;`
- `array1 = array2;`
- `array4 = array1;`
- `array5 = array3;`

Explanation:
// para mi lo fácil es decir que solo esas opciones señalan un array de 2 dimensiones.y esto es porqueya sea que desde el inicio se dijo array de 2 dimensiones como en 3, o array[] de array[] como en el caso de 2, 4 y 5. El único array de una dimensión de int es 1.

There is a subtle difference between: `int[] i;` and `int i[];` although in both the cases, i is an array of integers. The difference is if you declare multiple variables in the same statement such as: `int[] i, j;` and `int i[], j;`, j is not of the same type in the two cases.
In the first case, j is an array of integers while in the second case, j is just an integer.

Therefore, in this question:
array1 is an array of int array2, array3, array4, and array5 are arrays of int arrays
Therefore, option 1, 2 and 5 are valid.

Problema 12

What will the following program print?

```
class Test{
   public static void main(String[] args){
      int i = 4;
      int ia[][][] = new int[i][i = 3][i];
      System.out.println( ia.length + ", " + ia[0].length+", "+
ia[0][0].length);
   }
}
```

You had to select 1 option

- It will not compile.
- 3, 4, 3
- 3, 3, 3
- 4, 3, 4
- 4, 3, 3

Explanation:
// En mi idea se asigna y una vez asignado el siguiente ya tiene esa nueva asignación de 3

In an array creation expression, there may be one or more dimension expressions, each within brackets. Each dimension expression is fully evaluated before any part of any dimension expression to its right. The first dimension is calculated as 4 before the second dimension expression sets 'i' to 3.
Note that if evaluation of a dimension expression completes abruptly, no part of any dimension expression to its right will appear to have been evaluated.

Problema 13

Which of the options will create and initialize a matrix of ints as shown below?

	1		
	1		
	1		
	1		
	1		

You had to select 1 option

```
int[][] matrix = new int[5][4];
for(int i=1;i<=5; i++) matrix[i][1]  = 1;
```

```
int[][] matrix = new int[5][4];
for(int i=0;i<5; i++) matrix[i][1]  = 1;
```

```
int[][] matrix = new int[4][5];
for(int i=1;i<=5; i++) matrix[i][1]  = 1;
```

```
int[][] matrix = new int[5][4];
for(int i=1;i<=5; i++) matrix[i][2]  = 1;
```

Explanation:
// Mi explicacion fácil es ver bien que aunque las opciones 1 y 3 esten bien en las dimensiones, comienzan con i=1 por lo que nunca pondrá el valor en `matrix[i][1] = 1` en la primera fila.
// Practicamente son 2 pasos y se explican adelante: ver dimension y comienzo de for

You can visualize the given table in two ways - 1. As a table with 5 rows and 4 columns, where the second column of each row has a 1. In this case, you can define your matrix as new int[5][4]. Thus, the elements that you need to populate are: `[0][1]`, `[1][1]`, `[2][1]`, `[3][1]`, and `[4][1]`. This can be easily done using the for loop:

```
for(int i=0; i<5;i++) matrix[i][1] = 1;
```

Remember that array indexing starts with 0. Therefore the addresses of the rows are from 0 to 4 and the address of the second column is 1. 2. As a transposed table with 4 rows and 5 columns, where all the columns of the second row has a 1. In this case, you can define your matrix as `new int[4][5]`. Thus, the elements that you need to populate are: `[1][0]`, `[1][1]`, `[1][2]`, `[1][3]`, and `[1][4]`. This can be easily done using the for loop:

```
for(int i=0; i<5;i++) matrix[1][i] = 1;
```

Remember that array indexing starts with 0. Therefore the address of second row is 1 and the addresses of the columns are from 0 to 4.

Problema 14

Consider the following class…

```
class Test{
   public static void main(String[ ] args){
      int[] a = { 1, 2, 3, 4 };
      int[] b = { 2, 3, 1, 0 };
      System.out.println( a [ (a = b)[3] ] );
   }
}
```

What will it print when compiled and run ?
You had to select 1 option

- It will not compile.
- It will throw ArrayIndexOutOfBoundsException when run.
- It will print 1.
- It will print 3.
- It will print 4

Explanation:
// A mis palabras, se asigna en a[3] el valor de b[3] el cual es cero, por lo que lo que se imprime es
// a[0] que en este caso es 1

In an array access, the expression to the left of the brackets appears to be fully evaluated before any part of the expression within the brackets is evaluated.
In the expression `a[(a=b)[3]]`, the expression a is fully evaluated before the expression `(a=b)[3];` this means that the original value of a is fetched and remembered while the expression `(a=b)[3]` is evaluated. This array referenced by the original value of a is then subscripted by a value that is element 3 of another array (possibly the same array) that was referenced by b and is now also referenced by a.
So, it is actually `a[0] = 1`.
Note that if evaluation of the expression to the left of the brackets completes abruptly, no part of the expression within the brackets will appear to have been evaluated.

Problema 15

What will the following code snippet print?

```
int index = 1;
String[] strArr = new String[5];
String   myStr  = strArr[index];
System.out.println(myStr);
```

You had to select 1 option

- nothing
- null
- It will throw ArrayIndexOutOfBounds at runtime.
- It will print some junk value.
- None of the above.

Explanation:
// Se Inicia un array de String con 5, y mandas una String en el index 1, pero inicializado es ‘null’

When you create an array of Objects (here, Strings) all the elements are initialized to null. So in the line 3, null is assigned to myStr.
Note that. empty string is "" (`String str = "";`) and is not same as null.

Problema 16

What will the following program print?

```
public class TestClass{
  public static void main(String[] args){
    String str = "111";
    boolean[] bA = new boolean[1];
    if( bA[0] ) str = "222";
    System.out.println(str);
  }
}
```

You had to select 1 option

- 111
- 222
- It will not compile as bA[0] is uninitialized.
- It will throw an exception at runtime.
- None of the above.

Explanation:
// Lo que ayuda mucho es que `boolean[] bA =new boolean[1]` es como si inicializaras un boolean primitivo y este comienza con `'false'`, no para por el if y sol imprimirá el valor original del 'String str'

All the arrays are initialized to contain the default values of their type. This means,
`int[] iA = new int[10];` will contain 10 integers with a value of 0.
`Object[] oA = new Object[10];` will contain 10 object references pointing to null.
`boolean[] bA = new boolean[10];` will contain 10 booleans of value false.

So, as `bA[0]` is `false`, the if condition fails and str remains `111`.

Problema 17

Given the following code :

```
public class TestClass {

    int[][] matrix = new int[2][3];

    int a[] = {1, 2, 3};
    int b[] = {4, 5, 6};

    public int compute(int x, int y){
        //1 : Insert Line of Code here
    }

    public void loadMatrix(){
        for(int x=0; x<matrix.length; x++){
            for(int y=0; y<matrix[x].length; y++){
                //2: Insert Line of Code here
            }
        }
    }
}
```

What can be inserted at //1 and //2?
You had to select 1 option

- `return a(x)*b(y);` and `matrix(x, y) = compute(x, y);`
  - (and) are used to call a method on an object. To access array elements, you need to use [and].
- `return a[x]*b[y];` and `matrix[x, y] = compute(x, y);`
- `return a[x]*b[y];` and `matrix[x][y] = compute(x, y);` `//Es el que vi correcto declarado`
- `return a(x)*b(y);` and `matrix(x)(y) = compute(x, y);`
  - a(x), b(y), and matrix(x)(y) are invalid because a, b, and matrix are not methods.
- `return a[x]*b[y];` and `matrix[[x][y]] = compute(x, y);`
  - [[x][y]] is invalid syntax.

The correct syntax to access any element within an array is to use the square brackets - [ ]. Thus, to access the first element in an array, you would use array[0].

Problema 18

Given:

```
public class FunWithArgs {
    public static void main(String[][] args) {
        System.out.println(args[0][1]);
    }
    public static void main(String[] args) {
        FunWithArgs fwa = new FunWithArgs();
        String[][] newargs = {args};
        fwa.main(newargs);
    }
}
```

The above program is compiled with the command line:
`javac FunWithArgs.java`
and then run with:
`java FunWithArgs a b c`

What will be the output?
You had to select 1 option

- It will not compile.
- It will throw ArrayIndexOutOfBoundsException at run time.
- It will print b
- It will print null

Explanation.
// Pensé que no leeria un array de entrada por ser de 1 dimensión pero tanto para imprimir en el método main y en el principal puede ser accedido. Eso quiere decir que args[0][1] es el primer array (y unico) y en el segundo valor, que en este caso es 'b'

There is no problem with the code. The main method is just overloaded. When it is run, the main method with String[] will be called. This method then calls the main with String[][] with an argument { {"a", "b", "c"} } Thus, args[0][1] refers to "b", which is what is printed.

Problema 19

What will be the result of trying to compile and execute the following program?

```
public class TestClass{
   public static void main(String args[] ){
      int i = 0 ;
      int[] iA = {10, 20} ;
      iA[i] = i = 30 ;
      System.out.println(""+ iA[ 0 ] + " " + iA[ 1 ] + "  "+i) ;
    }
}
```

You had to select 1 option

- It will throw ArrayIndexOutOfBoundsException at Runtime.
- Compile time Error.
- It will print 10 20 30
- It will print 30 20 30
- It will print 0 20 30

Explanation:
// De alguna manera ya lo pude ver más rápido, iA[i] en el momento inicial es como si tuviera el valor iA[0], y lo que hace es asignar a iA[0] y a `i` el valor de 30, por lo que el print ya es fácil de ver.

The statement `iA[i] = i = 30 ;` will be processed as follows:
`iA[i] = i = 30; => iA[0] = i = 30 ;  =>  i = 30; iA[0] = i ; =>  iA[0] = 30 ;`

Here is what JLS says on this:
1 Evaluate Left-Hand Operand First
2 Evaluate Operands before Operation
3 Evaluation Respects Parentheses and Precedence
4 Argument Lists are Evaluated Left-to-Right

For Arrays: First, the dimension expressions are evaluated, left-to-right. If any of the expression evaluations completes abruptly, the expressions to the right of it are not evaluated.

Problema 20

Consider the following code:

```
public class ArrayTest {

    static int[][] table = new int[2][3];

    public static void init() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to initialize

            }
        }
    }

    public static void multiply() {
        for (int x = 0; x < table.length; x++) {
            for (int y = 0; y < table[x].length; y++) {

                //insert code to multiply

            }
        }
    }
}
```

Which of the following options can be used in the code above so that the init method initializes each table element to the sum of its row and column number and the multiply method just multiplies the element value by 2?

You had to select 1 option

- `table[x, y] = x+y;` and `table[x, y] = table[x, y]*2;`
- `table[x][y] = x+y;` and `table[x][y] = table[x][y]*2;`
  - This code snippet illustrates correct syntax for accessing array elements in a multi dimensional array. All other options are syntactically incorrect and will not compile.
- `table[[x] [y]] = x+y;` and `table[[x] [y]] = table[[x] [y]]*2;`
- `table(x, y) = x+y;` and `table(x, y) = table(x, y)*2;`

// Es el que está bien escroto