**Emre KARAGÖZ**

# ASSIGNMENT 4
# CREATING AND USING JAVA ARRAYS
# (HW 4)

**Q1-) Option B**

```
public static void main(String... args)
```

This line uses a syntax called varargs (variable arguments). "Ellipsis (…)" is used for a varargs method parameter. **Option B** is the answer.

**Q2-) Option B**

The "…" syntax tells the Java compiler that the method can be called with zero or more arguments. As a result, "`f`" variable is implicitly declared as array of type "`Frisbee`". Thus, inside the method, "`f`" variable is accessed using the array syntax. We can use "`f[0]`" to get the first element from the varargs parameter. **Option B** is the answer.

**Q3-) Option D**

Both "`int[] lowercase`" and `Integer[] uppercase` are arrays. In Java, all arrays are heap objects /references. Namely, none of them is primitive. **Option D** is the answer.

**Q4-) Option C**

While declaring an array, we can type the "[]" before or after the name, and adding a space is optional. "`double[] tiger;`" and "`double bear[];`" are legal declarations but "`[]double lion;`" is illegal. Typing "[]" before variable type is not allowed. **Option C** is the answer.

**Q5-) Option C**

A String varargs parameter can take either a String array or String type variables but "`public void printStormNames(String[] names)`" method can only take String array parameter. "`printStormNames("Cindy");`" method call does not compile. Because, the parameter `"Cindy"` is a String, not an array. **Option C** is the answer.

**Q6-) Option A**

In Java, "`length`" variable is used to determine the number of elements in an array. It's not a method, it is a variable. **Option A** is the answer.

**Q7-) Option C**

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

We can use these signatures to create an empty two-dimensional array with dimensions 2×2. **Option C** is one of these and it is the answer.

**Q8-) Option B**

`days.length=7,` the code compiles without an issue and prints seven lines. **Option B** is the answer.

**Q9-) Option B**

`Arrays.binarySearch()` and `Arrays.sort()` methods are methods to do searching and sorting respectively on arrays. **Option B** is the answer.

**Q10-) Option B**

`String[] nums` is a String array so, "`Arrays.sort()`"method sorts elements of this array in alphabetic order. While doing this, short strings sort before longer strings. So, [1, 10, 9] is the correct order. **Option B** is the answer.

**Q11-) Option B**

"`length`" attribute is the number of elements in array. "`Index 0`" is the first, "`Index (length-1)`" is the last element of array. We can reach them by using "`trains[0]` and `trains[trains.length - 1]`". **Option B** is the answer.

**Q12-) Option C**

The most common ways to create an array looks like these:

```
int[] numbers1 = new int[3];
int[] numbers2 = new int[] {1, 2, 3};
```

Another way to create an array is to specify all the elements it should start out with:

```
int[] numbers2 = {1, 2, 3};
```

This approach is called an anonymous array. It is anonymous because you don't specify the type and size. Finally, you can type the "`[]`" before or after the name, and adding a space is optional.

- `String lion [] = new String[] {"lion"};`
- `String tiger [] = new String[1] {"tiger"};`
- `String bear [] = new String[] {};`
- `String ohMy [] = new String[0] {};`

Respect to this information, the first and third statements above are legal declarations. **Option C** is the answer.

**Q13-) Option B**

If no elements are being provided when creating the arrays, the size must be specified after the array type. Only the second declaration below follows this rule. **Option B** is the answer.

- `float[] lion = new float[];`
- **`float[] tiger = new float[1];`**
- `float[] bear = new[] float;`
- `float[] ohMy = new[1] float;`

**Q14-) Option C**

| Scenario | Result |
|---|---|
| Target element found in sorted array | Index of match |
| Target element not found in sorted array | Negative value showing one smaller than the negative of index, where a match needs to be inserted to preserve sorted order |
| Unsorted array | A surprise—this result isn't predictable |

`binarySearch()` rules are showed above. A sorted array is required to get an accurate result while using `binarySearch()`. **Option C** is the answer.

**Q15-) Option A**

An array uses a zero index to reference the first element so **Option D** is true. Besides, an array is allowed to contain duplicate values and understands the concept of ordered elements. This makes both **Option B** and **Option C** are also true statments about arrays. Yet, an array does not expand automatically when it is full. Size of arrays are not dynamic. **Option A** is false and it is the answer.

**Q16-) Option C**

```
String[][] matrix = new String[1][2];
matrix[1][0] = "Is all around you "; //m3
matrix[1][1] = "Why oh why didn't I take the BLUE pill?"; m4
```

This lines try to reference the second element of the outer array of this 2D array but, there is no second element. Size of this 2D array is 1x2. Because of this firstly, m3 couses an `ArrayIndexOutOfBoundsException.` If we remove this line, m4 couses this exception. **Option C** is the answer.

**Q17-) Option B**

```
String[] os = new String[] { "Mac", "Linux", "Windows" };
```

"`Arrays.sort(os)`" method sorts elements of this array in alphabetic order as follows:

```
{ "Linux", "Mac", "Windows" };
```

"`Arrays.binarySearch(os, "Mac")`" returns the index of "`Mac`" which is "`1`". **Option B** is the answer.

**Q18-) Option A**

```
char[][] ticTacToe = new char[3,3];
```

This is not a legal declaration, this is the first line that prevents the code from compiling. So, **Option A** is the answer. Even if we correct this illegal declaration as follow:

`char[][] ticTacToe = new char[3,3];` the code still not compiled. Because the following lines: "`ticTacToe[1][3] = 'X';`", `ticTacToe[3][1] = 'X';` cause `ArrayIndexOutOfBoundsException.`

**Q19-) Option B**

Firstly, an array is created and it is the first object of that created. After that 2 Integer objects are created and assigned to first 2 indexes of the array. As a result, we have 3 objects here. **Option B** is the answer.

**Q20-) Option B**

```
"[][] String alpha;"
"[] String beta;"
```

These are illegal declarations. "[]" is not allowed to before the variable type.

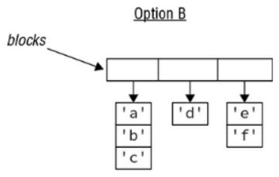
```
"String[][] gamma;"
"String[] delta[];"
"String epsilon[][];"
```

These are legal declarations. "[]" is allowed to before or after the variable type. **Option B** is the answer.

**Q21-) Option B**

```
char[][] blocks = new char[][] { { 'a', 'b', 'c' }, { 'd' }, { 'e', 'f' } };
```

`blocks` is an a multidimensional array that points three different arrays of different lengths.



**Option B** is the answer.

**Q22-) Option D**

```
public static void addStationName(String[] names){
     names[names.length] = "Times Square";
}
```

As explained in Q11, "`length`" attribute is the number of elements in array. "`Index 0`" is the first, "`Index (length-1)`" is the last element of array. There is not an element such as `names[names.length]`. Because of this, the code throws an `ArrayIndexOutOfBoundsException`. **Option D** is the answer.

**Q23-) Option C**

The code does not compile. "`days`" is a String array, not an array list. To get number of elements of an array, we should use length attribute. size() method can be used for array lists.

**Q24-) Option C**

```
boolean[][][] bools, moreBools;
```

Both `"bools"` and `"moreBools"` are 3D arrays here. Becuse, braces were declared before the variable names. **Option C** is the answer.

**Q25-) Option C**

```
String[] strings = new String[2];
System.out.println(strings);
```

`[Ljava.lang.String;@74a14482` is the possible output for this code. `"[L"` means it is an array, `"java.lang.String`" is the reference type, and `"160bc7c0"` is the hash code. So, **Option C** is the answer. To print elements of `"strings"` array, `"Arrays.toString`" may be used. If we use this, we get `"["null", "null"]` as the output.

**Q26-) Option B**

```
char[][] ticTacToe = new char[3][3]; // r1
ticTacToe[1][3] = 'X'; // r2
ticTacToe[2][2]='X';
ticTacToe[3][1] = 'X';
System.out.println(ticTacToe.length + " in a row!"); // r3
```

Line r2, prevents from this code from compiling. Because there is not an element of this array such as `ticTacToe[1][3].`r3 couses an `ArrayIndexOutOfBoundsException`. **Option B** is the answer.

**Q27-) Option D**

varargs can only be used as a method parameter.

`String... copy = original;` this is not a legal variable declaration and it prevents from this code from compiling. **Option D** is the answer.

**Q28-) Option D**

`"game"` is a two-dimensional `"int array"` of size 6×6. After the creation of this `"int[]"`, "6" is assigned to `game[3][3]`. This is legal. After that, `"game"` casts to an `Object[]` and it is legal too. Yet, line 8 causes a `"ArrayStoreException"` at runtime. Because, `"X"` is a "String" and, we should not assign a "String" to `Object[] obj`, `int[]` is correct type for `"obj"`. **Option D** is the answer.

**Q29-) Option C**

"Arrays.binarySearch" searches for the index of "RedHat". Although it is not in the list the search can determine that is should be inserted at element 2. According to the rule, if we negate this index and subtract 1, we get the result what Java gives us. By doing this, we get (-2-1) "-3" as the result. **Option C** is the answer.

**Q30-) Option B**

```
java FirstName Wolfie
```

"FirstName" class is run with the argument "Wolfie". This code prints the first element of "names" (names[0]). "Wolfie" is the output. **Option B** is the answer.

**Q31-) Option C**

```
java Count 1 2
```

"Count" class is run with the argument "1", "2". This code prints the number of elements of target array. It has 2 elements and **Option C** is the answer.

**Q32-) Option B**

"EchoFirst" class is called with 2 arguments which are "seed" and "flower". "seed" is assigned to "String one" After that elements of array are sorted in alphabetical order as follows:

```
{"flower", "seed"}
```

"Arrays.binarySearch" searches for the index of "seed" and returns 1. **Option B** is the answer.

**Q33-) Option D**

**Option A:** `int[][] nums = new int[2][1];`
**Option B:** `int[] nums[] = new int[2][1];`
**Option C:** `int[] nums[] = new int[][] { { 0 }, { 0 } };`

These options create a two-dimensional array of size 2×1. Option C also specifies the elements.

**Option D:** `int[] nums[] = new int[][] { { 0, 0 } };`

This option creates a two-dimensional array of size 1×2. It is different than others and, it is the answer.

**Q34-) Option C**

Both **options A** and **B** are illegal because, arrays are not indexed by using Strings. Indexing is started with zero. We can access `"x"` with `dimensions[2][2]`. **Option C** is the answer.

**Q35-) Option D**

Since arrays are indexed start with "0", days[0] is the first element of this array. When the list time the for loop runs, it attemps to access an element called "days[7]". Yet, this element does not exist. "days[6]" the last element of this array.The code compiles but throws an `ArrayIndexOutOfBoundsException at runtime.` **Option D** is the answer.

**Q36-) Option C**

```
java FirstName Wolfie
```

`"FirstName"` is called with one argument which is `"Wolfie".` It is the first and only element of "names". There is not a second element such as names[1]. The code throws an `ArrayIndexOutOfBoundsException` exception. **Option C** is the answer.

**Q37-) Option D**

The code compiles without an issue, **Option D** is the answer. A 2D array is created and, 'X' is assigned to three elements of the array. `"ticTacToe.length"` is "3" because, this array has "3" elements. So, the output is `" 3 in a row!".`

**Q38-) Option D**

The code does not compile because of `"lenghth()".` `"length"` is variable, not a method. To compile this code, parentheses after the `"length"` should be removed. **Option D** is the answer.

**Q39-) Option B**

`"boolean[][] bools[], moreBools;"`
This line creates two arrays. One of them is "`bools`" which is a 3D array and, the other one is `"moreBools" which is 2D.`**Option B** is the answer.

**Q40-) Option B**

`java counting.Binary`
"`args`" is an empty array because, no argument is passed. Sorting an empty array does not cause an error and, it returns an empty array again. **Option B** is the answer.
**Q41-) Option D**

`Arrays.binarySearch` searches for the index of "Linux". Yet, a sorted array is required to get accurate result from `Arrays.binarySearch`. Since "`os`" is not a sorted array, the output is not defined. **Option D** is the answer.

**Q42-) Option B**

The code does not compile because, we are not allowed to assign a String to int array. Line 8 attempts to this. **Option B** is the answer.

**Q43-) Option A**

This is 2D array has 2 elements. So, "`listing.length`" is 2. The first element of it has only 1 element. "`listing[0].length`" is 1. **Option A** is the answer.

**Q44-) Option C**

```
java FirstName
```

"`FirstName`" class is called with no arguments. So, "`names`" is an empty array and, it has no elements. Attempting to access to "`name[1]`" throws `ArrayIndexOutOfBoundsException`. Because "`name[1]`" does not exist. **Option C** is the answer.

**Q45-) Option A**

Arrays are indexed started with zero. "`days`" has 7 elements and, the for loop prints all elements of "`days`" array except the first element. So, it prints 6 lines. **Option A** is the answer.

**Q46-) Option B**

```
java Count "1 2"
```

"`Count`" class is run with the argument "`1 2`". This is only one argument because double quotes are used to declare it. This code prints the number of elements of target array and, it has 1 element and **Option B** is the answer.

**Q47-) Option A**

```
String[] os = new String[] { "Linux", "Mac", "Windows" };
```

A sorted array is required to get an accurate result by using `Arrays.binarySearch()` and, `"os"` is already sorted. We do not need to use `"Arrays.sort()"`. `Arrays.binarySearch()` searches for the index of "Linux" and returns "0". **Option A** is the answer.

**Q48-) Option A**

- You can always change a method signature from `call(String[] arg)` to `call(String... arg)` without causing a compiler error in the calling code.

We can do this because, from within a method, an array parameter and a varargs parameter are acted the same.

- You can always change a method signature from `call(String... arg)` to `call(String[] arg)` without causing a compiler error in the existing code.

We cannot do this because; even if both types can receive an array, only a varargs parameter is allowed to automatically turn individual parameters into an array.
**Option A** is the answer.

**Q49-) Option B**

**Option A:** `int[][][][] nums1a, nums1b;`
**Option C:** `int[][] nums3a[][], nums3b[][];`
**Option D:** `int[] nums4a[][][], numbs4b[][][];`

These options create two 4D arrays.

**Option B:** `int[][][] nums2a[], nums2b;`

But this option creates a 4D array and a 3D array. It is the answer.

**Q50-) Option C**

```
java unix.EchoFirst seed flower
```

The code does not compile because, "`Arrays.binarySearch()`" searches for the index of desired element and returns an "int" value. We cannot assign an "int" to a "String". This prevents the code from compiling. **Option C** is the answer.