Bünyamin AKTAŞ
08/05/2020

ASSIGNMENT – 4

Q1 **The answer is B.**

Varargs is a special array parameter declaration only used in method parameters. It can be any type of data, for example, *int.* As an example they can be used as below:

public static void main(String… args)

Varargs can only be used once in a method and they must be at the end of parameter list. *(int … nums, int num2)* is not a valid usage, for example. It is because varargs can take any number of individual parameters and transform them to an array. Also an array can be passed for the vararg parameter.

(..) and (---) have no usage in Java which makes **options A** and **D** are incorrect. (--) is a unary operator which decrements variables by 1. So the **option C** is also not correct.

Option B is the only correct option.

Q2 **The answer is B.**

As explained above varargs are treated such an array. So they can be indexed as to they are arrays. Since arrays are indexed zero based the first element of vararg f can be accessed via *f[0]*, which makes **option B true.**

The word *f* here is a reference variable which points to the array object of *Frisbee.* So if we use *f* we would be trying to assign a reference variable, which points to an array object, to a Frisbee reference variable which would cause compiler error. **The option A** is not true.

f[1] points to the second element of the array *f.* So **the option C** is wrong.

Q3 **The answer is D.**

Arrays are actually objects that they can be cast to *Object* class type. So none of the arrays are not primitives.

The first one, *int[] lowercase,* indicates an array object consisted of primitive data type values of *int.* But this does not make the object primitive. So the answer is **D.**

Q4 **The answer is C.**

There are two types of array declaration in Java:

double[] tiger;
double tiger[];

Spaces are optional that they can be placed before brackets, [].

Among these examples *double[] tiger* and *double bear[]* are the correct ones. *[]double lion* is not correct because putting brackets before array declaration is not valid. **The answer is C, two.**

Q5 **The answer is C.**

As also explained in **Q1,** for a vararg parameter an array or a number of elements can be passed to the method as long as those elements are of type of the vararg parameter. So **the option A** is correct because Java will take *"Arlene"* and put it into an array object of size 1. **The option B** is also correct since it already passes an array.

Method calls in **options C** and **D** are different than the **options A** and **B.** These call the method *printStormNames(String[] names)* which takes an array as a parameter, not a vararg. So that an array must be passed to this method. **Option D** passes an array so that will compile. But **option C** tries to pass a String value which is not valid because it is not an array.

Q6 **The answer is A.**

A – *length* attribute is used to determine the number of elements of arrays.

B – *length()* method is used to determine the size of a String variable, not of an array.

C – There is no attribute called *size* for arrays or objects.

D – *size()* method are used on collections such as *ArrayList*, to determine the size.

Q7 **The answer is C.**

Arrays are actually objects as explained before. And also we know that arrays can hold objects, which brings us to the conclusion that arrays can hold arrays. Arrays holding arrays is available in Java, called *multidimensional arrays.* Multidimensional arrays can be created as below:

int [][] nums; *// 2D array*
int [] nums []; *// 2D array but not recommended because of less readability.*
int [][] nums = new int[3][2]; *// 2D array which all sizes determined.*
int [][] nums = new int[4][]; *// 2D array, the size of the first dimension is determined. The others can be determined separately.*
nums[0][1] = new int[5];
nums[0][2] = new int[3]; which also means that multidimensional arrays with different sizes are available in Java.

int [][] nums = new int{{1,2},{3},[5,6,7}}; // 2D array with different sizes.

Up until now, it is understood that **the options A, B** and **D** are not valid. **The option C is the answer.**

Q8 **The answer is B.**

First statement declares and initializes a String array with seven elements which are the days of a week. Second statement is a *for loop* which visits every element of the array *days* and prints them respectively. For that there are seven days, the array *days* also has seven elements so that **seven lines will be printed.**

The code compiles because there is no syntax error. The code will not throw an exception at runtime because there is no trying to access an element of an object or of an array and also that the single array is a fixed array that cannot be changed at runtime causing an exception.

Q9 **The answer is B.**

Java allows searching an element in sorted arrays. If the array to be searched is not sorted then the result of searching is unpredictable.

Java also allows sorting elements of an array. These two things can be done via two methods residing in *Arrays* class that needs to be imported in order to be used. *Arrays* class can be imported with *java.util.Arrays* or *java.util.**, which imports all classes of *java.util* library.

Assuming we have an array called *numbers*, we can sort this class by the statement,

Arrays.sort(numbers);

We can search an element, for example 2, in the array with the statement,

Arrays.binarySearch(numbers,2);

linearSort() and *search()* methods do not exist. So **the only option that contains the required methods is B.**

Q10 **The answer is B.**

The array *nums* is consisted of type String elements. So when it is passed into the method *Arrays.sort()* then it will be sorted according to String sorting rules which are actually alphabetical sorting. In alphabetical sorting, numbers come before letters and uppercases come before lowercases.

Here, since 1 comes before 9, the method will sort the elements beginning with “1”. So “1” and “10” will be put first. Then “9” will come. The method will become [1, 10, 9].

The other method, *Arrays.toString()*, is a method created to print arrays nicely. If we try to print an array object using its reference name then the code prints the hash code of the array object, not its elements.

So finally, the array will be printed in its last shape which is [1, 10, 9].

Q11 **The answer is B.**

Arrays are indexed zero based which means the first element of an array is accessed in the index 0. So first element is referenced by *trains[0]*.

And again since arrays are zero based, if an array has 10 elements then the indexes are, *0,1,2,3,4,5,6,7,8,9.* So the last element is at index *9.* But the length of the array is 10. So the last element of the array *trains* can be referenced by *trains[trains.length – 1]*. So **the answer is B.**

Q12 **The answer is C.**

As already explained in **Q7,** an array can be initialized by stating its size or expressing explicitly its elements. But it cannot be initialized both stating its size and its elements. For example, this type of declare is not valid:

int nums[] = new int[2] {2,3};

So the statements of *tiger* and *ohMy* is not valid. First statement is valid as it does not state any size but states single element. Third statement is also valid because it does not state any size but states no element in brackets which means an empty array. So **the answer is two.**

Q13 **The answer is B.**

If the array's size is not stated then its elements must be stated. In the first line, none of them are stated which makes this statement not valid. Second statement is valid since it states the size of the array as 1.

The last two statements are not valid because the brackets in the initialization part must be at the end, not between *new* keyword and the data type name.

So only **one statement** is valid which makes **the option B is true.**

Q14 **The answer is C.**

As it is explained in **Q9,** an array must be sorted in order to get an accurate result of searching an element. That must be so because the *binarySearch()* method performs binary search. In binary search, the middle element of the array is checked and if that element is not equal to the searched value then two values are compared. If the middle element is bigger than the searched value, then the middle element of the left part of the middle element of the original array is checked. If the

middle element is less than the searched value, then the middle element of the right part of the middle element of the original array is checked. This process continues recursively. So in order to get an accurate result the array must be sorted. **The option C is correct.**

Q15 **The answer is A.**

A – An array has a fixed size that never changes.

B and C – An array can contain duplicate values because it is an ordered list.

D – Arrays are indexed zero based as stated before, so that the first element of an array is referenced by zero index.

Q16 **The answer is C.**

Arrays are indexed zero based which makes the first element referenced by zero index. In this question, the array *matrix* is a two-dimensional array which the first array is of size 1 and the inner arrays (which, in this case, is one array) are of size 2. So the first index can only be accessed via [0]. So the lines *m3* and *m4* are trying to access to second element of the array, which does not exist. So these two lines are capable of causing *ArrayIndexOutOfBoundsException.* But since this exception is thrown out at runtime Java will throw out when it encountered the first line which is trying to access to a non-existing element, which is the *line m3.* **So the answer is C.**

Q17 **The answer is B.**

Since the array is of type String, it will be sorted alphabetically which will be *Linux, Mac, Windows.*

So *Mac* is now at the second place which is indexed as 1 in arrays. **The output will be 1.**

Q18 **The answer is A.**

Line r1 – Initilialization of the array is wrong because the brackets cannot be this way, [3,3]. It must be two bracket couples which contains *3* in them such as [3][3]. **So this is the first line causing the program not compiling.**

Line r2 – If the previous line was fixed then the array must have 3 element slots, each of which has an array of size 3. As arrays are indexed zero based, the last elements can be accessed via *ticTacToe[2][2].* If we use [3] in any brackets we would be trying to access to the fourth element which does not exist. So if the line r1 was fixed then this line would also prevent the code from running without exception error.

Line before the line r3 – This line is not marked but this line suffers from the same mistake as **the line r2.**

Line r3 – This line does not cause any error.

Q19 **The answer is B.**

Arrays are objects which means that the first statement creates an array object which consists of Integer variables. In the next statements of initializing of the elements of the array *lotto* new objects are created by *new* keyword. **So in the final place, there are three different objects.**

Q20 **The answer is B.**

The valid array declarations are as below, as also stated in **Q7**:

int [][] nums; *// 2D array*
int [] nums []; *// 2D array but not recommended because of less readability.*
int [][] nums = new int[3][2]; *// 2D array which all sizes determined.*
int [][] nums = new int[4][]; *// 2D array, the size of the first dimension is determined. The others can be determined separately.*
int [][] nums = new int{{1,2},{3},[5,6,7}}; *// 2D array with different sizes.*

As a result brackets cannot be put before the statement so the first two statements are not valid. **The last three statements are valid.**

Q21 **The answer is B.**

The statement creates a two-dimensional array with different sizes. This array has 3 elements which first one of them is of size 3, the second one is size of 1 and the last one is size of 2.

Options A and **B** represents a two-dimensional array of size 3x3 which has 9 slots at total. So these graphics are wrong.

Option D represent a 2D array which has 3 elements, each of which has an array of size 3 which is not correct because our array *blocks* has different sized arrays.

Option B is the only one representing 3 arrays in the array *blocks*, that is of correct sizes.

Q22 **The answer is D.**

The method tries to assign a new value to the n. element of an n-size array, which is not possible because if an array is of size n, the last element of that array is at the index n-1, which also falsifies **the option B**. So the code compiles but throws an exception at runtime. **The answer is D.**

Arrays are fixed sized ordered lists that cannot be added new elements which makes **the option A** wrong.

Q23 **The answer is C.**

This code snippet tries to invoke a method called *size()* on the array object *days*. Arrays do not have any method called *size()*. This method exists for collections such as ArrayList, to get the size of the collection. **So this code does not compile.**

If the *for-loop* statement is fixed such as using *.length*, which is the attribute that returns the size of an array, instead of *.size()*, then it would print seven lines.

Q24 **The answer is C.**

In this statement two array objects are declared with three-dimension since three brackets are put before the reference names. If the brackets were after a reference name the dimensions could have to be different. **This way the answer is C.**

Q25 **The answer is C.**

The word *strings* here is a reference name for the *strings* array object. So if we attempt to print *strings* then it will print the object type (L, indicates array), the reference type (java.lang.String) and the hash code. **The answer is C.**

If *Arrays.toString(strings)* was used instead of *strings*, then it would print *[null, null]*. There is no way of getting printed [,] because even if the array was zero-sized then output would be [].

Q26 **The answer is B.**

First line r1 is correct so this line does not prevent this code from running or compiling.

Second line r2 attempt to reach an element that does not exist. This 2D array has 3 elements, each of which has an array of size 3. So the last elements are index 2 since arrays are zero based. This statement does compile but throws *ArrayIndexOutOfBoundsException*at runtime. **This is the first line preventing the code from running well.**

The line before the line r3 also attempt to reach a non-existing element as *the line r2* does. So even if *the line r2* was fixed, the code still wouldn't run well.

The line r3 has no mistakes, so does not prevent the code from compiling or running without error.

Q27 **The answer is D.**

As it is explained in **Q1**, varargs can only be used in method signatures, in parameter lists. In this code, it is attempted to declare a vararg inside of the method which is not allowed. So the code would not compile.

If we attempt to fix the mistake by changing vararg to a normal array doing *String[] copy* then the code either throws *ArrayIndexOutOfBoundsException* at runtime since no arguments passed to the program. *java Copier* only runs the program, not passes any arguments. So even in this case none of the other options are true.

Q28 **The answer is D.**

game is a static two-dimensional array. Line 7 casts *game* to a wider type *Object.* This is legal because *Object* can hold all objects and game[][] is an object, too. But this is dangerous. This *obj[]* array holds one-dimensional arrays. However, since Object is the superclass of all objects, Java assumes that it can hold all object type values and sees Line 8 as legal, which assigns a String value to the int[].

In Java, when you assign a reference name of an array object to a different array reference name, both reference names refer to the same array object which means if you perform an action with/on one of the reference names, it will affect the other since the both reference names refer to the same object, the action is performed on that object.

So here, when Line 8 assigns the String value to int[] in *obj[]*, it also happens to assign this value to *game[]* which is not legal because *game* is of type int[][].

Because Java sees Line 8 legal, the code will compile. But at runtime, the code will throw an *ArrayStoreException* since it expects an int[] value to be assigned at Line 8. **The answer is D.**

Q29 **The answer is C.**

The second statement sorts the array alphabetically which results in [Linux, Mac, Windows].

The third statement attempt to print the result of searching a String value in this sorted array. The value searched is “RedHat” which actually doesn’t exist. When *binarySearch()* method can’t find a value in a sorted array then it finds the proper index number to put the value in the array preserving the sorted order. It negates that index number and subtract 1 from it. Then returns the final value.

Here “RedHat” must be placed after “Mac” and before “Windows” to preserve the sorted order which indicates the index number of possible “RedHat” value as 2. Then we should negate this number and subtract 1 which makes it, **-2 – 1 = -3. The answer is C.**

Q30 **The answer is B.**

Since one argument, *Wolfie*, is passed to the program, to the main() method with the command *java FirstName Wolfie*, Java will create single sized an array, named *names*, which has “Wolfie” as the element.

So the *System.out.println()* method will print that single element to the screen as Wolfie. **The answer is B.**

The code does not throw an ArrayIndexOutOfBoundsException because the code attempts to reach the first element of the array which contains already 1 element. The code does not also throw a NullPointerException because the array *names[0]* cannot be null. Either it exists or not. If it exists then its value is printed. If not, then ArrayIndexOutOfBoundsException is thrown at runtime.

Q31 **The answer is C.**

The code doesn't have any syntax error so it does compile. Since the program takes 2 arguments from command line the parameter array, *target[]*, of the *main()* method is created as two sized. So the *length* attribute of this array, *target*, will return 2.

Q32 **The answer is B.**

This program takes two arguments from command line. The first argument, which is placed in *args* array, is assigned to String variable *one*. Then the *args* array is sorted. After this operation the value of variable *one*, which is *seed*, is searched in sorted array and the result is assigned to an *int* variable named *result*.

Since the alphabetical order of *args* array is [flower, seed], the value of variable *one* is in second index which is 1. **So the result will be 1.**

The code compiles and runs without throwing an exception because there is no attempt to reach a non-existent index.

Q33 **The answer is D.**

A – This statement produces a 2D array with two elements, each of which has 1 element.

B – This statement also produces a 2D array with two elements, each of which has 1 element, although using different style of declaration.

C – This statement also produces a 2D array with two elements, each of which has 1 element, explicitly stating the values of elements.

D – This statement produces a 2D array with one element which has two elements, differently than the previous three declarations.

Q34 **The answer is C.**

The reference name *dimensions* references a 2D array with three elements, each of which has different sizes. But the requested value "z" is in the third slot of the second dimension array which is in the third slot of first dimension array.

Since the arrays are zero based, we can reach this value as below:

dimensions[2][2]

Arrays are indexed using numbers, not strings. So **the options A** and **B** are incorrect.

Q35 **The answer is D.**

The code compiles because there is no syntax error. But in *for-loop*, index variable *i*, is started from 1 which indicates the second element in an array. And the value of *I* is incremented until the length of the array which will cause the program throw an ArrayIndexOutOfBoundsException at runtime, since the array indexes are zero based, after printing days starting from the second element, which is "Monday" until the last one. At the end there will be six lines printed but since before terminating the program there is an exception, **the answer is D.**

Q36 **The answer is C.**

java FirstName Wolfie command passes only one argument to the program, there will be one sized array. But this code attempts to print the second element of that array which does not exist. So that the program does compile but throws an ArrayIndexOutOfBoundsException at runtime.

The code does not throw a NullPointerException because the array *names[1]* cannot be null. Either it exists or not. If it exists then its value is printed. If not, then ArrayIndexOutOfBoundsException is thrown at runtime as it will in this code.

Q37 **The answer is D.**

The code does not have any syntax error, as in **Q18,** or does not attempt to reach a non-existent element, as in both **Q18** and **Q26.** So the code compiles and runs without error. The answer is D.

Q38 **The answer is D.**

java Count 1 2 passes two arguments to class *Count* so the array of the *main()* method will be at length 2. But the code tries to call a method *length()* rather than the attribute *length*. This method does not exist for arrays so the code does not compile.

Q39 **The answer is B.**

Since there are two bracket couples after *boolean* keyword the next reference names will be allowed multiple dimension arrays beginning from two. *bools* reference name has an additional [] so that this reference name will refer to a 3D array. But *moreBools* does not have any additional brackets so this array will be two-dimensional.

Q40 **The answer is B.**

java counting.Binary does not pass any argument to the program. The command contains *java* which is needed to run a Java program from command line and the package name *counting* with a dot which is needed to go in the package directory, and the class name *Binary.*

When no argument passed to a Java program, Java creates an array with 0 size. So in this program, the array *args* will be created empty. *Arrays.sort()* method actually does not do anything since the array is empty. *Arrays.toString()* method will print an empty brackets **[].**

Q41 **The answer is D.**

Arrays.binarySearch() method requires a sorted array. The array *os* is not a sorted array so that the result of the search is unpredictable. So the answer is D.

If the array was sorted, then the result would be 0 since the alphabetical order the array would be [Linux, Mac, Windows].

Q42 **The answer is B.**

game is a static two-dimensional array of type int[][]. Line 7 casts *game* to a wider type *Object.* This is legal because *Object* can hold all objects and game[][] is an object, too. This is dangerous as explained in **Q28** but since this variable is not used it is not a concern in this code.

Line 8 attempts to assign a String value to int. This is not a legal assignment so that **the code does not compile.**

Q43 **The answer is A.**

When *length* attribute is used with an array, then it returns the length of the array of the first dimension. That means if we need to require of the length of an array which is held inside of an another array, then we need to state the index number of the array.

In this code, a multi-dimensional array *listing* is created with different sized arrays. *System.out.println()* method firstly tries to print the length of the first dimension of the array *listing and* later tries to print the first array held in the *listing*, which has 1 element.

So the output will be the lengths of the array *listing* and of the first array held in *listing.* That is **2 1.**

Q44 **The answer is C.**

java FirstName command does not pass any argument to the program. So the array is initialized as an empty array, which means zero sized. But the code attempts to print the first element of

the array which does not exist since the array is empty. This will cause the program throw ArrayIndexOutOfBoundsException at runtime.

Q45 **The answer is A.**

The declaration and initialization of the array *days* are correct. The *for-loop* statement starts the index counter variable *I* from 1 and increments it until it reaches to the last element's index number which equals to the ((length of the array) – 1). So the code prints days starting from "Monday" until the "Saturday". The number of lines printed is 6.

Q46 **The answer is B.**

java Count "1 2" passes 1 argument to the program since the characters between quotes refers to one single String value. So the array of the *main()* method will be initialized with size one. So the *target.length* will return 1 as the length of the array is 1.

Q47 **The answer is A.**

Arrays.binarySearch() method requires a sorted array and the array *os* is initialized in a sorted way. So the code compiles and finds the requested value accurately.

Since "Linux" is the first element, the method will return 0 as the arrays' indexes are zero based.

Q48 **The answer is A.**

I – Since varargs can take normal arrays as argument, changing a signature of a method from *call(String[] arg)* would not cause an error. This method could be called from somewhere being passed an array. When the change occurs, then the array will be passing to this vararg parameter, which is valid. So this change does not cause a compiler error.

II – As already explained in **Q1,** varargs can take multiple individual arguments and tranform them into an array. In some place this method could be invoked by being given individual arguments for the vararg parameter. In that case changed signature *call(String[] arg)* could not recognize these arguments since this signature only accepts an array. So in this case the code does not compile.

As a result **only the statement I is correct.**

Q49 **The answer is B.**

Valid array declarations are as below:

int [][] nums; *// 2D array*

int [] nums []; *// 2D array but not recommended because of less readability.*

int [][] nums = new int[3][2]; *// 2D array which all sizes determined.*

int [][] nums = new int[4][]; *// 2D array, the size of the first dimension is determined. The others can be determined separately.*

int [][] nums = new int{{1,2},{3},[5,6,7}}; *// 2D array with different sizes.*

So either the brackets come before the reference name or the other or both, it is valid. If both, then a *total-number-of-brackets*-dimensional array.

In **options A, C** and **D,** all arrays are created as four-dimensional arrays. But in **the option B,** while *nums2a* is created as four-dimensional array, *nums2b* is created as three-dimensional array. So **the answer is B.**

Q50 **The answer is C.**

java unix.EchoFirst seed flower passes 2 arguments to the program, *seed* and *flower.* These values are put in the array *args* as [seed, flower]. *Arrays.sort()* method sorts this array alphabetically and makes it [flower, seed].

Arrays.binarySearch() will search for the value of the first element of the array *args. Arrays.binarySearch()* method returns an *int* value. But in this code, this return value is attempted to be assigned to a String variable *result,* which will cause a compiler error. **The code does not compile.**

The Zen of Python

The Zen of Python is a set of principles for writing computer programs, that influence Python programming language.

1) Beautiful is better than ugly.
2) Explicit is better than implicit.
3) Simple is better than complex.
4) Complex is better than complicated.
5) Flat is better than nested.
6) Sparse is better than dense.
7) Readability counts.
8) Special cases aren't special enough to break the rules.
9) Although practicality beats purity.
10) Errors should never pass silently.
11) Unless explicitly silenced.
12) In the face of ambiguity, refuse the temptation to guess.
13) There should be one-- and preferably only one --obvious way to do it.
14) Although that way may not be obvious at first unless you're Dutch.
15) Now is better than never.
16) Although never is often better than *right* now.
17) If the implementation is hard to explain, it's a bad idea.

18) If the implementation is easy to explain, it may be a good idea.
19) Namespaces are one honking great idea -- let's do more of those!