Homework 4

Q1)(B)

A variable-length argument is specified by three periods(…).

In JDK 5, Java has included a feature that simplifies the creation of methods that need to take a variable number of arguments. This feature is called varargs and it is short-form for variable-length arguments. A method that takes a variable number of arguments is a varargs method.

Prior to JDK 5, variable-length arguments could be handled two ways. One using overloaded method(one for each) and another put the arguments into an array, and then pass this array to the method. Both of them are potentially error-prone and require more code. The varargs feature offers a simpler, better option. It is still true that multiple arguments must be passed in an array, but the varargs feature automates and hides the process.

```
public static String format(String pattern,
                                Object... arguments);
```

The three periods after the final parameter's type indicate that the final argument may be passed as an array *or* as a sequence of arguments.

Note: A method can have variable length parameters with other parameters too, but one should ensure that there exists only one varargs parameter that should be written last in the parameter list of the method declaration.

```
    int nums(int a, float b, double … c)
```

Q2)(B)

If the last formal parameter is a variable arity parameter of type **T**, it is considered to define a formal parameter of type **T[]**.

The **...** syntax tells the Java compiler that the method can be called with zero or more arguments. As a result, nums variable is implicitly declared as an array of type **int[ ]**. Thus, inside the method, nums variable is accessed using the array syntax.

Q3)(D)

Although int is a primitive type int[] and Integer[] are both reference types, because arrays are objects that stores a collection of values. The fact that an array itself is an object is often overlooked. An array is an object itself; it stores references to the data it stores.
Arrays can store two types of data:

- A collection of primitive data types
- A collection of objects

Q4)(C)

Square brackets can not be placed before the type name of the array.
An array declaration includes the array type and array variable. The type of objects that an array can store depends on its type. An array type is followed by one or more empty pairs of square brackets []. Square brackets can follow either the variable name or its type. In the case of multidimensional arrays, it can follow both of them. Here are some valid array declaration examples:

- `int[] anArray;`
- `int intArray[];`
- `String[] strArray;`
- `int[] multiArray[];`
- `int[][] multiArr;`
- `int multiArr[][];`

Note: In an array declaration, placing the square brackets next to the type(as in `int[]` or `int[][]`) is preferred because it makes the code easier to read by showing the array types in use.

Q5)(C)

If the last formal parameter is a variable arity parameter of type T, it is considered to define a formal parameter of type T[]. varags is essentially turning given variable number of arguments to an array so giving it and array or variable number of arguments is same. varargs can take individual actual parameters or directly an array. From within the method varargs is actually an array. Even arguments are given as individual parameters, it is still true that multiple arguments must be passed in an array, but the varargs feature automates and hides the process. So, basically **`T`**… is actually a syntactic sugar for **`T[].`** Varargs doesn't make anything new possible (by definition of syntactic sugar), but it reduces verboseness and makes some constructs much more agreeable to implement.

Q6)(A)

The array's length is available as a final instance variable **length**. **length()** method is mainly used for size of string related objects. The **size()** method of List interface in Java is used to get the number of elements in this list. That is, this method returns the count of elements present in this list container.

Q7)(C)

When you allocate memory for an array, you should specify its dimensions, such as the number of elements the array should store. Note that the size of an array can't expand or reduce once it is allocated. Here are a few examples:

- `int intArray[] = new int[2];`
- `String[] strArray = new String[4];`
- `int[] multiArr[] = new int[2][3];`

Because an array is an object, it's allocated using the keyword new, followed by the typeof value that it stores, and then its size. The code won't compile if you don't specify the size of the array or if you place the array size on the left of the = sign, as follows:

- `intArray = new int[];`
- `intArray[2] = new int;`

The size of the array must evaluate to an int value. You can't create an array with its size specified as a floating-point number. The following line of code won't compile:

- `int[] intArray = new int[2.4];`

Q8)(B)

The given code uses for loop to iterate through the array and print out its elements. Array has the length 7 including weekdays. For loop starts with 0 and runs until `i < array.length`, using `<` relational operator here does not cause a problem,because array indexes starts at 0.

Q9)(B)

The java.util.Arrays.binarySearch(Object[] a, Object key) method searches the specified array for the specified object using the binary search algorithm.The array be sorted into ascending order according to the natural ordering of its elements prior to making this call. If it is not sorted, the results are undefined.

Java *utils.Arrays.sort method* provides us with a quick and simple way to sort an array of primitives or objects that implement the *Comparable* interface in ascending order. When sorting primitives, the *Arrays.sort* method uses a Dual-Pivot implementation of QuickSort. However, when sorting objects an iterative implementation of MergeSort is used.

Q10)(B)

Because of the elements of the array are Strings, sort method sorts elements in alphabetical order. 1 comes before 10 and 10 comes before 9. Although “1” and “10” share same character which is “1”, because “1” is shorter than “10”, it comes before “10”.

Q11)(B)

Array indexes start with 0, that is to say the first element of the array is at the index 0. Public instance variable **length** gives us the length of the array but because arrays indexes start with 0 we need to subtract the length with 1 to get to the last element of the array.

Q12)(C)

We can combine of all the three steps of array declaration, allocation and initialization into one step. as follows:

- `int intArray[] = {0, 1};`
- `String[] strArray = {"Summer", "Winter"};`
- `int multiArray[][] = { {0, 1}, {3, 4, 5} };`
-

All the previous steps of array declaration, allocation, and initialization can be combined in the following way, as well:

- `int intArray2[] = new int[]{0, 1};`
- `String[] strArray2 = new String[]{"Summer", "Winter"};`
- `int multiArray2[][] = new int[][]{ {0, 1}, {3, 4, 5}};`

Unlike the first approach, the preceding code uses the keyword **new** to initialize and array. If we try to specify the size of an array with the preceding approach, the code won't compile. When we combine an array declaration, allocation, and initialization in a single step, we can’t specify the size of the array. The size of the array is calculated by the number of values that are assigned to the array.

Q13)(B)

Because an array is an object, it’s allocated using the keyword `new`, followed by the type of value that it stores, and then its size. THe code won’t compile if we don’t specify the size of the array or if we place the array size just after the `new` keyword, as follows:

- `float[] ohMy = new[1] float;`

Q14)(C)

By definition binary search is a search algorithm that finds the position of a target value within a sorted array. Binary search works by assuming the middle of the array contains the median value in the array. If it is not sorted, this assumption does not make sense, since the median can be anywhere and cutting the array in half could mean that we cut off the number we were searching for.

Q15)(A)

The size of an array can't expand or reduce once it it allocated. Arrays are allocated as a single chunk of memory. Growing an array is problematic because the only way to do it properly is to grow it at the end. For a growth of size N there must be at least N free bytes at the end of the array before the next allocated address. Supporting this type of allocation necessitates that allocations be spread across the virtual address space. This both removes the benefits of having memory allocations closer to each other and serves to increase fragmentation.

Q16)(C)

The code first creates a two-dimensional array and then initializes its elements one by one. On line three and four it tries to access second element of the first dimension while the length of the first dimension is 1, so the code throws and indexOutOfBounds exception at runtime.

Q17)(B)

Code declares and initializes a string array and prepares it to binary search by sorting it with the sort method. In the sorted array "Mac" element is at index 1, which is being the second element, is printed to screen.

Q18)(A)

Line r1 is the first line to prevent this code from compiling. Invalid syntax is used to define the size of the array. Multi-dimensional arrays' sizes should be defined in square brackets, one couple for each size (`[3][2]`).

We should also note that, even with the syntax corrected, code would throw `indexOutOfBounds` at runtime, because two lines tries to access elements where there is none, that is to say out of bounds of the array.

Q19)(B)

First and array is created with length 4. Note that because it's type is an reference type, at the time of the allocation references in the array gets initialized with default value of `null`, which is to say they point to nowhere. After that two Integer wrapper objects created and assigned as first and second element of the object. Array itself is an object, plus two wrapper objects makes the total count of the objects created 3.

Q20)(B)

An array declaration includes the array type and array variable. An array type is followed by one or more empty pairs of square brackets `[]`. To declare an array, specify its type, followed by the name of the array variable. Here's An example of declaring arrays of int and String values:

- `int intArray[];`
- `String[] strArray;`
- `int[] multiArray[];`
- `int[] multiArr[];`
- `int[][] multiArr;`
- `int[] anArr;`
- `int anArr[];`
- `int multiArr[][];`

The square brackets can follow the array type or its name, but can't come before the array type.

Q21)(B)

An array of primitives stores a collection of values that constitute the primitive values themselves. (With primitives, there are no objects to reference.) An array of objects stores a collection of values, which are in fact heap-memory addresses or pointers.The addresses point to (reference) the object instances that your array is said to store,which means that object arrays store references (to objects) and primitive arrays store primitive values.

In java, we can define one-dimensional and multidimensional arrays. A one-dimensional array is an object that refers to a collection of scalar values. A two-dimensional (or more) array is referred to as a multidimensional array. A two-dimensional array refers to a collection of objects in which each of the objects is one-dimensional array. Similarly, a three-dimensional array refers to a collection of two-dimensional arrays, and so on.

Note that multi-dimensional arrays may or may not contain the same number of elements in each row or column.

Q22)(D)

`names.length` gives the length of the array. Because array indexes start with 0, last element in the array is `names.length - 1`. Because we are assigning and String literal to a index that is out of bounds of the array, code will throw the exception `ArrayIndexOutOfBounds`.

Q23)(C)

`size()` is a method specified in java.util.Collection interface, which is then inherited by every data structure such as (ArrayList, LinkedList etc.) in the standard library. `length` is a field on any array that stores the size of the array.

Collection interface is the the root interface in the collection hierarchy . A collection represents a group of objects, known as its elements. All general-purpose Collection implementation classes typically implement Collection indirectly through one of its subinterfaces.

Q24)(D)

We can put the brackets after the type, in this case when declaring more than one array with one statement, brackets applies to all variables.

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

We can also place brackets after the variable names an have variables that have different dimensions. For example:

- `boolean bools2[][][], moreBools2[][];`

Q25)(C)

All Java objects have a `toString()` method, which is invoked when you try and print the object. This method is defined in the Object class (the superclass of all Java objects). The `Object.toString()` method returns a fairly ugly looking string, composed of the name of the class, an @ symbol and the hashcode of the object in hexadecimal.

A result such as `com.foo.MyType@2f92e0f4` can therefore be explained as:

- `com.foo.MyType` - the name of the class, the class is `String` in the package `com.foo`
- `@` - joins the string together
- `2f92e0f4` - The hashcode of the object

The name of array classes look a little different, which is explained well in the Javadocs for `Class.getName()`. For instance, `[Ljava.lang.String` means:

- `[` - an single-dimensional array (as opposed to [[or [[[etc.)
- `L` - the array contains a class or interface
- `java.lang.String` - the type of objects in the array

Q26)(B)

Indexes starts with 0 and since the `ticTacToe` array is a length 3 array with both dimensions, maximum valid index is 2 which is the last element in the array if used in the second dimension. Code throws ArrayIndexOutOfBounds exception at runtime because on line two it tries to assign to an index which is out of bounds for the `ticTacToe` array.

Q27)(D)

The code does not compile. While varargs parameter is just a syntactic sugar and arguments are packed as an array inside method, it can only be used as method parameter. Inside the method such syntax is invalid.

Q28)(B)

The code does not compile. First we assign and integer to a cell, which is valid. Then we assign the array reference of type `int[][]` to an array reference of type `Object[]`. This is okay since arrays themself are objects. Now, `obj` array holds references to **arrays of `int[]`**, but we are assigning and string where we should assign and array of integer `int[]`. SO the compiler gives the error `ArrayStoreException`.

Q29)(C)

`binarySearch` method returns index of the search key, if it is contained in the array; otherwise, **(-(insertion point) - 1)**. The insertion point is defined as the point at which the key would be inserted into the array: the index of the first element greater than the key, or a.length if all elements in the array are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

Q30)(B)

`varargs` is just a syntactic sugar, inside method they are packed into an array. `FirstName` in the command line command `java FirstName Wolfie` is just the class name. `Wolfie` is the program argument and since it is the first and only argument we can access it in the main method with the `0` index of the array.

Q31)(C)

`length` is an instance variable in array objects storing the size of the array. When run as `java Count 1 2`, `System.out.println(args.length)` would print “2”. `Count` is the class name and `1` and `2` is the parameters given. So the total count of the array’s elements are 2.

Q32)(B)

The class is called with 2 arguments, “seed” and “flower” respectively. `unix.EchoFirst` means `EchoFirst` class in the unix package. Then the code sorts the array readying it for binary search for the first element in the array which is “seed” parameter. After the sort “seed” becomes the second element in the array, residing at index 1. Result is assigned to an integer and then printed to screen.

Q33)(D)

`int[][] nums = new int[2][1]` and `int[] nums[] = new int[2][1]` just showing that the brackets can both follow type and variable name. `int[] nums[] = new int[][] { { 0 }, { 0 } }` is also a 2x1 array declared, allocated and also initialized in the same statement. However `int[] nums[] = new int[][] { { 0, 0 } }` declares, allocates and initializes a 1x2 array on the contrary.

Q34)(C)

Element “z” resides as the third element of the third array in the array dimensions. Accessed correctly with `dimensions[2][2].` Arrays are indexed with numbers. When we want to access an element of an array we use indexes, (array[0] for example). `dimensions["three"][2]` and `dimensions["three"][3]` both uses Strings to access first dimension elements of the array where it should had been used numbers. `dimensions[3][3]` have the corrects syntax but while having a correct syntax it tries to access elements where there is none actually.

Q35)(D)

The code compiles but throws ArrayIndexOutOfBounds exception at runtime. Because `<=` operator used inside for expression, last iteration runs with `i = 7` and then used as index to access an element, which is out of bounds of the array for that index.

Q36)(C)

The code throws an ArrayIndexOutOfBoundsException when run as `java FirstName Wolfie`. FirstName is the class name so we are given only one parameter to the program but we are trying to access the second element in the array.

Q37)(D)

- `char[][] ticTacToe = new char[3][3]; // r1`

r1 line is a valid declaration and allocation for a 3x3 array.

- `ticTacToe[0][0] = 'X'; // r2`

r2 line is also a valid line, assigning a char primitive to an element in the 3x3 array.

- `System.out.println(ticTacToe.length + " in a row!");  // r3`

This line just prints the length of the array concatenating it with a string literal. Code compiles and runs without any issue printing "`3 in a row!`".

Q38)(D)

The code does not compile because the `length()` is a method defined on String objects which gives the length of the string. We need to use the instance variable `length` instead of invoking an undefined method `length()` to return the size of the array.

Q39)(B)

Two pairs of brackets follow the type. `boolean[][]` applies both of the variables. One pair of brackets that follow the variable `bool` only applies to that variable, making it effectively three-dimensional array. `moreBool` on the other hand does not have extra brackets that follow variable so it is only affected by `boolean[][]` making it a two-dimensional array.

Q40)(B)

`Arrays.toString()` returns a string representation of the contents of the specified array. The string representation consists of a list of the array's elements, enclosed in square brackets (`[]`). Adjacent elements are separated by the characters ",“ (a comma followed by a space). Returns "null" if a is null.

Since no arguments are given java creates an empty array and sorting an empty array is valid, the code would print `[]`.

Q41)(D)

We get the correct result with this example but this behavior is not guaranteed the binary search runs with the assumption that the array is already sorted, so with an unsorted array the result is undefined.

Q42)(B)

The code does not compile because line 8 (`game[3][3] = "X";`) is trying to assign a string to an array element, where the array type is `int[][].` We should note that even if the type mismatch corrected, we would get an runtime exception because of line “`game[3][3] = 6;`”. The game variable is not initialized with an array it just got the default value of null. We would get `NullPointerException`, trying to access a variable which is `null`.

Q43)(A)

Multidimensional arrays may or may not contain the same number of elements in each row or column. `listing` variable is declared, allocated and initialized on the same statement.

- `String[][] listing = new String[][] { { "Book" }, { "Game", "29.99" } };`

Listing is a two-dimensional array, having varied size arrays in the second dimension.

- `System.out.println(listing.length + " " + listing[0].length);`

`listing` array has the size 2, first array in the `listing` array has the size one ({ "Book" }).

Q44)(C)

When run as `java FirstName` the code would throw an `ArrayIndexOutOfBounds` exception at runtime. “`FirstName`” is the class name, we are giving no parameter to the class so `names` is an empty argument.

Q45)(A)

Starting from index `1`, until the last iteration with `i = 6` which corresponds to the index of the last element in the array. First element which has the index 0 will not get printed while all the other elements will be printed.

Q46)(B)

We are actually giving just one parameter as an argument. When run as `java Count “1 2”` the code outputs `1`. `Count` is the class name and `“1 2”` is the one and only argument because double quotes are used. `“1 2”` argument is of type String and has three characters including the space character.

Q47)(A)

The binary search algorithm requires the array at hand to be sorted. We do not have to sort the given array `{ "Linux", "Mac", "Windows" }` with `Arrays.sort()` method because the given array is already sorted. `Arrays.binarySearch(os, "Linux")` searches for the key which is “`Linux`” and return the index of the key that is `0`.

Q48)(A)

The three periods after the final parameter's type indicate that the final argument may be passed as an array or as a sequence of arguments. We can’t change the method signature from `call(String… args)` to `call(String[] args)` because varargs accepts individual parameters and also an array but `call(String[] args)` just accepts and array.

Q49)(B)

- `int[][][] nums2a[], nums2b;`

In this line `int[][][]` is applied to both but `nums2a` variable has another dimension added because of the following square brackets. While `nums2a` is a four-dimensional array, `nums2b` is an three-dimensional array in this case. They can point to an array that is different from the others.

Q50)(C)

The code does not compile because we are trying to assign ant int primitive to a String object. We should also note that this code is meaningless aside from the type mismatch compilation error because we are using the first element of the already sorted array and using it as the key in binary search. The result is always going to be 0. Also this code is prune to `ArrayIndexOutOfBoundsException`, given there are no arguments.