**Arrays:**

* **An area of memory on the heap with space for designated number of elements.**
* **Ordered list that can contain duplicate**
* **Two arrays with same content are not equal because array equals() method only tests for reference not equivalence.**
* **Arrays class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists. The methods in this class all throw a NullPointerException, if the specified array reference is null, except where noted.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Parameters | Return Type | Description | Examples |
| asList | T | List<T> | Returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to the array.) This method acts as bridge between array-based and collection-based APIs, in combination with Collection.toArray(). The returned list is serializable and implements RandomAccess.  This method also provides a convenient way to create a fixed-size list initialized to contain several elements. | List<String> stooges = Arrays.asList("Larry", "Moe", "Curly"); |
| binarySearch | Array, key | Int | Searches the specified array of type for the specified value using the binary search algorithm. The array must be sorted (as by the [sort(Any Array[])](https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#sort-byte:A-) method) prior to making this call. If it is not sorted, the results are undefined. If the array contains multiple elements with the specified value, there is no guarantee which one will be found. |  |
| binarySearch | Array, fromIndex,toIndex, key | Int | Same as above |  |
| Equals | Array, Array | Boolean | Returns true if the two specified arrays of booleans are *equal* to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. In other words, two arrays are equal if they contain the same elements in the same order. Also, two array references are considered equal if both are null | deepEquals(Object[],Object[]) is used for comparing multi -dimensional arrays. |
| Sort | Array | Void | Sorts the specified array into ascending numerical order.  **Does not work with Multidimensional arrays** | String[][] strings = {{**"A"**,**"Z"**},{**"C"**,**"D"**,**"S"**},{**"L"**}};  Arrays.sort(strings) throws a class cast exception. |
| Sort | Array,fromIndex, toIndex | Void | Same as above |  |
|  |  |  |  |  |

**Different ways of declaring and initializing an array:**

*This is the most common way of declaration, note left hand side*

*Is declaration and this can use various combinations like*

*String stringArray[], String [] stringArray, String[] stringArray*

*The right hand side is the instantiation, note the [] is required if*

*you are using the new String, you can’t just say new String{“A”,”B”,”C”}*

**String[] stringArray = new String[]{"A", "B", "C"};**

*You can directly instantiate an array with the {} syntax as well, just*

*Remember not to use the new String verbiage on the right-hand side.*

**String [] stringArrayCopy = {"A", "B", "C"};**

*You can also instantiate an empty array with just the size. Below command will create an array with size two and default value 0 filled in the array.*

**int** intArray[] = **new int**[2];

You can declare and instantiate an array in two separate lines like below

**char** []charArray ;  
charArray = **new char**[2];

However while doing so, if you want to initialize the array in a separate line, you will need the new char[] syntax . In other words, you cant do this

**char** []charArray ;

charArray = {**'X'**,**'Y'**}; //!! COMPILE ERROR Array initializer is not allowed

The correct way is

charArray = new char[] {‘X’,’Y’};

You can initialize an empty array with size 0

**int** array[] = **new int**[0]; // This is valid

But if you try to add an element you get array index bound error

**Invalid Declaration of array**

**int** singleArray[] = **new int** []; // Array Initializer expected. The compiler wants you to specify the size of the array.

**int** singleArray = **new int** [3]; // Required Array but found Integer

**int** array[][] = **new int**[][]; // array initializer expected

For a Multidimensional array, you can’t do this

**int**[] multiArrInt[] = **new int**[1][];  
multiArrInt[0][0] = 1;

The compiler does allow you to instantiate a multidimensional array if you just declare the size of the first dimension of the array, but while using the array (just the first dimension, you get a null pointer.

You can however do this

**int**[] multiArrInt[] = **new int**[1][];  
multiArrInt[0] = **new int**[1];  
multiArrInt[0][0] = 1;

Notice in the second line, we have explicitly instantiated the first dimension of the array with size 1. We need to instantiate the second dimension as well if we want to use the second dimension of the array.

But if you have instantiated all the dimensions of the array properly during the instantiation phase, then you can use the array dimensions without worrying about null pointer errors

**int**[] multiArrInt1[] = **new int**[1][0];  
multiArrInt[0][0] = 1;

Varargs are not allowed to be declared as reference variables. They are only allowed as method parameters.

**These are valid declaration for “for” and “if” statements .**

**for**(**int** x = 0; x <=10;x++);  
 System.***out***.println(**"Hello"**); // This prints out “Hello” once

**int** x = 0;  
 **if**(x != 0);  
 System.***out***.println(**"Hello"**); // This prints out “Hello” once

**Switch Statements:**

* Switch allows int, byte, short, char, Enum, String, Byte, Short, Character and Integer.
* Empty Switch statements are ok but break just by itself is not OK.
* Continue statement is not allowed inside a switch statement.
* Curly braces are required for a switch
* Switch cases do not allow duplicates
* They only allow compile time constants or literals

**If- else Statements**

* This declaration is OK

If(y < 10) System.out.println(“OK”);

Else (“OK”);

Notice the semi colon in the end of If, which is ok.

* Below declaration is ok   
  **if** (x <=6)  
   **if**(y > 0)  
   **if**(x > y)  
   out +=**"1"**;  
   **else** out +=**"2"**;  
   **else** out +=**"3"**;  
  **else** out += **"4"**;
* Below declaration is not OK

**if**(x = 0) **return** 0;

Notice how (x=0) is assignment and not an expression, if x had been a Boolean then this would be valid (x = true)

* If does not allow continue but allows labeled break.
* This is a valid declaration

**int** x = 0;  
 **if**(x != 0);  
 System.***out***.println(**"Hello"**); //

**Basically, the if(x !=0); get’s executed as any other java statement, the loop for this clause does not hold because it has a termination character “;” .**

* Continue can be used with an If else only if the If-Else is inside a loop like :while , do-while and for

**While loop**

* The variable you are using with for a while loop, needs to be initialized, and there needs to be termination condition inside the loop otherwise there’s a chance of having an infinite loop.
* If the condition inside a while loop is false to begin with, then the code does not compile with an error: unreachable statement

Ex: while(false) {} //!! Compile error

**For Loop:**

* For(;;) // Valid
* Int x = 0;  
   for(long y = 0, x = 4;;) not valid because x is an int but in the initialization part, we are using it as a long.
* Int x = 0; long y = 4;

For(x = 1,y = 2;;) This is valid because the declaration of the long and int are done in the previous line of the loop.

* This is a valid for loop

@Test  
**public void** testWierdLookingForLoop() {  
  
 */\*Example below demonstrates that it is valid to have  
 ;; as a for loop syntax , this creates an infinite loop\*/*

**int** count = 0;  
 **int** x = 0;  
 String output =**""**;  
 **for** (doStuff(),System.***out***.println(); count < 10 && doStuff(); output += count,System.***out***.println(),x =21,count++,doStuff());  
 *assertEquals*(output,**"0123456789"**);  
  
}  
**public boolean** doStuff(){  
 **return true**;  
}

* */\*This would generate an infinite loop because the expression inside the loop increments i but   
   then assignes i to the old value 0. \*/*.   
  */\* for(int i = 0;i < 10;)  
   i = i++;\*/*