Array Assignment

Q.1 An array in programming is a data structure that stores a fixed-size, sequential collection of elements of the *same data type*. Think of it like a row of boxes, where each box can hold one value, and all the boxes are arranged next to each other in a specific order.

Here's a breakdown of the key characteristics of arrays:

- **Fixed Size:** Once you create an array, its size (the number of elements it can hold) is fixed. You cannot dynamically increase or decrease the size of an array after it's created (in standard arrays; some languages have dynamic array implementations).
- **Homogeneous Data Type:** All elements in an array must be of the same data type (e.g., all integers, all strings, all floating-point numbers).
- **Sequential Storage:** Elements are stored in contiguous memory locations, meaning they are placed one after the other in memory. This allows for efficient access to elements using their index.
- **Index-Based Access:** Each element in an array is accessed using its index (position). Indices typically start from 0. So, the first element is at index 0, the second element is at index 1, and so on.

Example (Conceptual):

An array named `numbers` that can store 5 integers:

numbers: | 10 | 25 | 5 | 18 | 32 |

Indices: 0 | 1 | 2 | 3 | 4

Q.2 In Java, you can create an array in two main ways:

1. Declaration and Creation Separately

- Declaration

  ` int[] numbers; // Declares an array variable named 'numbers' that can hold integers ```

  Creation (Instantiation):

  numbers = new int[5]; // Creates an array of 5 integers. All elements are initialized to 0 by default.

2.Combined Declaration and Creation
int[] ages = new int[10]; // Declares and creates an array of 10 integers

Example:
public class ArrayExample {

```
    public static void main(String[] args) {
        // Separate declaration and creation
        int[] numbers;
        numbers = new int[5];

        // Combined declaration and creation
        String[] names = new String[3];

        // Initialization with values (during creation)
        int[] scores = {85, 92, 78, 95, 88};
    }
}
```

Q.3 No, you cannot directly change the size of a standard Java array at runtime. Once you create an array with a specific size using `new int[size]`, that size is fixed for the lifetime of that array object.

If you need a data structure that can grow or shrink dynamically, you should use one of the Java Collections Framework classes, such as `ArrayList`, `LinkedList`, or `Vector`.

Why Arrays are Fixed-Size:

Arrays in Java (and many other languages) are stored as contiguous blocks of memory. This contiguous storage is essential for efficient access to elements using their index (e.g., `array[index]`). If you were to resize an array, it might require moving a large chunk of data to a new memory location to accommodate the new size, which is inefficient.

Q.4 In Java, you can declare an array variable *without* specifying its size initially, but you must specify the size before you can actually use the array (i.e., access or modify its elements).

1. Declaration without Size

int[] numbers; // Declares an array variable 'numbers' that can hold integers

At this point, `numbers` is just a reference variable. It doesn't point to any actual array object in memory. It's like having a label that could be attached to a box, but you haven't created the box yet.

2. Creation (with Size):

You must then create the array using the `new` keyword and specify its size

numbers = new int[5]; // Creates an array of 5 integers

Now, `numbers` refers to an actual array object in memory that can hold 5 integers.

Combining Declaration and Creation (Common Practice):

The most common way to declare and create an array is to do it in a single line:

int[] values = new int[10]; // Declares and creates an array of 10 integers

Initializing with Values (Alternative to Specifying Size):

There's one special case where you don't explicitly specify the size using `new int[size]`: when you initialize the array with values directly during declaration:

int[] data = {1, 2, 3, 4, 5}; // Creates an array of size 5 and initializes it with the given values

In this case, the size of the array is determined automatically by the number of values you provide within the curly braces `{}`.

Q.5 When you create an array in Java using the `new` keyword, the elements of the array are automatically initialized to default values. These default values depend on the data type of the array:

- **Numeric Types:**
    - `int`, `byte`, `short`, `long`: `0`
    - `float`, `double`: `0.0` or `0.0d`
- **Boolean Type:**
    - `boolean`: `false`
- **Character Type:**
    - `char`: `\u0000` (null character)

- **Reference Types (Objects):**
    - Any object type (e.g., `String`, custom classes): `null`

Q.6 In Java, a 1D array is a linear collection of elements of the same data type stored in contiguous memory locations.

Example:

int[] numbers = {10, 20, 30, 40, 50};

- **`int[]`:** This declares an array named `numbers` that can hold integer values.
- **`{10, 20, 30, 40, 50}`:** This initializes the array with the given values.

Key Characteristics:

- **Fixed Size:** Once created, the size of an array is fixed. You cannot change the number of elements it can hold.
- **Homogeneous Data Type:** All elements in the array must be of the same data type.
- **Index-Based Access:** Each element is accessed using its index (position), starting from 0. For example, `numbers[0]` refers to the first element (10).

Accessing Elements:

int firstElement = numbers[0]; // Accesses the first element (10)

int thirdElement = numbers[2]; // Accesses the third element (30)

1D arrays are a fundamental data structure in programming, used to store and organize collections of data efficiently.

Q.7 public class TwoDArrayExample {

public static void main(String[] args) {

// Declare and initialize a 2D array (3 rows, 4 columns)

int[][] matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12}

};

```
        // Accessing elements
        System.out.println("Element at [0][0]: " + matrix[0][0]); // Output: 1
        System.out.println("Element at [1][2]: " + matrix[1][2]); // Output: 7
        System.out.println("Element at [2][3]: " + matrix[2][3]); // Output: 12

        // Modifying elements
        matrix[1][1] = 60;
        System.out.println("Modified element at [1][1]: " + matrix[1][1]); // Output: 60

        // Getting the number of rows
        int numRows = matrix.length;
        System.out.println("Number of rows: " + numRows); // Output: 3

        // Getting the number of columns (for the first row, but all rows should have the same
number of columns in a standard 2D array)
        int numCols = matrix[0].length;
        System.out.println("Number of columns: " + numCols); // Output: 4

        // Iterating through the 2D array (nested loops)
        System.out.println("Matrix elements:");
        for (int i = 0; i < numRows; i++) { // Outer loop for rows
            for (int j = 0; j < numCols; j++) { // Inner loop for columns
                System.out.print(matrix[i][j] + "\t"); // Print with tab spacing
            }
```

```
            System.out.println(); // New line after each row
        }

        // Another way to declare and initialize a 2D array
        int[][] anotherMatrix = new int[2][3]; // 2 rows, 3 columns
        anotherMatrix[0][0] = 100;
        anotherMatrix[0][1] = 200;
        anotherMatrix[0][2] = 300;
        anotherMatrix[1][0] = 400;
        anotherMatrix[1][1] = 500;
        anotherMatrix[1][2] = 600;

        System.out.println("\nAnother Matrix elements:");
        for (int i = 0; i < anotherMatrix.length; i++) {
            for (int j = 0; j < anotherMatrix[0].length; j++) {
                System.out.print(anotherMatrix[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```