Array (Module 21)
Assignment

Q.1 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.2 No, you cannot pass a negative number as the size of an array in Java . Doing so will result in a `NegativeArraySizeException` at runtime.

Explanation:

- Array Size and Memory Allocation: When you create an array using `new int[size]`, the Java Virtual Machine (JVM) needs to allocate a contiguous block of memory to store the array elements. The `size` parameter specifies how many elements the array should hold, and therefore, how much memory needs to be reserved.

- Negative Size Makes No Sense: A negative size for an array doesn't make logical sense in terms of memory allocation. You can't allocate a negative amount of memory. It's like asking for -5 boxes to store items; it's not a meaningful request.

- `NegativeArraySizeException`: If you attempt to create an array with a negative size, the JVM will throw a `java.lang.NegativeArraySizeException`. This is a runtime

exception, meaning it occurs during program execution, not at compile time.

Q.3 In Java, arrays are stored in the heap memory.

- **Heap:**
    - The heap is a region of memory used for dynamic memory allocation.
    -
    - When you create an array using `new int[size]` (or for any other data type), the JVM allocates a contiguous block of memory on the heap to store the array elements.
    - Objects in Java are also stored on the heap.

Q.4 While arrays are a fundamental and useful data structure, they do have some limitations:

1. Fixed Size:
    - This is the most significant disadvantage. Once you declare an array with a specific size, you cannot change its size during runtime.
    - If you need to store more elements than the declared size, you have to create a new, larger array and copy all the elements from the old array to the new one. This is inefficient in terms of both time and memory.
2. Homogeneous Data Type:
    - Arrays can only store elements of the same data type. You cannot have an array that stores both integers and strings, for example.
    - This lack of flexibility can sometimes make it necessary to use more complex data structures when dealing with heterogeneous data.
3. Inefficient Insertions and Deletions (in the middle):
    - Inserting or deleting an element in the middle of an array requires shifting all subsequent elements to make space or

close the gap. This can be time-consuming, especially for large arrays.

4. Memory Wastage (Potential):
    - If you declare an array with a large size but only use a small portion of it, you are wasting memory. The unused portion of the array still occupies memory space.

Q.5 An anonymous array in Java is an array that is created and used without explicitly assigning it to a named variable. It's essentially a one-time-use array.

Key Characteristics:

- No Name: As the name suggests, it doesn't have a name or identifier.
- One-Time Use: It's typically used for a single purpose, such as passing an array directly as an argument to a method.
- Creation and Initialization in One Step: It's created and initialized at the same time using the `new` keyword followed by the data type and the array initializer (curly braces `{}`).

Syntax:

```
new dataType[] {value1, value2, value3, ...};
```

```
public class AnonymousArrayExample {
```

Example:

```
    // Method that takes an integer array as an argument
    static void printArray(int[] arr) {
        System.out.print("[");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i]);
```

```
            if (i < arr.length - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    public static void main(String[] args) {
        // Passing an anonymous array to the printArray method
        printArray(new int[] {10, 20, 30, 40, 50}); // Output: [10,
20, 30, 40, 50]

        // Another example
        int sum = 0;
        for (int num : new int[] {2, 4, 6, 8}) {
            sum += num;
        }
        System.out.println("Sum: " + sum); // Output: Sum: 20

        //Anonymous String Array
        for(String str : new String[] {"Hello","World"}){
            System.out.print(str + " ");//Output: Hello World
        }
```

```
        System.out.println();
    }
}
```

Q.6 Here are the different ways to traverse an array in Java:

1. Using a `for` loop:

- This is the most common and straightforward way to traverse an array.
- You iterate through the array using a loop variable that starts from 0 and increments until it reaches the length of the array.

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


for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
// Output: 10 20 30 40 50
```

2. Using an enhanced `for` loop (for-each loop):

- This is a more concise way to iterate through an array when you only need to access the elements and don't need the index.

```
for (int number : numbers) {
    System.out.print(number + " ");
}
// Output: 10 20 30 40 50
```

3. Using a `while` loop:

- This is another way to traverse an array, although less common than `for` loops.

```
int i = 0;

while (i < numbers.length) {

    System.out.print(numbers[i] + " ");

    i++;

}

// Output: 10 20 30 40 50
```

4. Using a `do-while` loop:

- Similar to the `while` loop, but the loop body executes at least once before the condition is checked.

```
int i = 0;

do {

    System.out.print(numbers[i] + " ");

    i++;

} while (i < numbers.length);

// Output: 10 20 30 40 50
```

Q.7 The difference between `length` and `length()` is :

- **`length` (without parentheses) is a *property* or *attribute* used to get the size of an *array*.**
- **`length()` (with parentheses) is a *method* used to get the number of characters in a *String*.**

Examples:

1. Using `length` with Arrays:

```
public class ArrayLengthExample {

    public static void main(String[] args) {

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

        int arrayLength = numbers.length; // Accessing the length
property

        System.out.println("Array length: " + arrayLength); // Output:
5



        String[] names = {"Alice", "Bob", "Charlie"};

        int namesLength = names.length;

        System.out.println("Names array length: " + namesLength); //
Output: 3

    }

}
```

2. Using `length()` with Strings:

```
public class StringLengthExample {

    public static void main(String[] args) {

        String message = "Hello, World!";

        int stringLength = message.length(); // Calling the length()
method

        System.out.println("String length: " + stringLength); //
Output: 13



        String emptyString = "";
```

```
        int emptyLength = emptyString.length();

        System.out.println("Empty string length: " + emptyLength); //
Output: 0

    }

}
```