

```
/*
Problem:
Given an array of integers, return the second largest UNIQUE number.
If it doesn't exist, return -1.
```

Approach:

- Traverse the array once.
- Track the largest and second largest unique numbers.
- Make sure duplicates don't affect the unique comparison.
- Time Complexity: O(n), Space Complexity: O(1)

Sample Input: [3, 5, 2, 5, 6, 6, 1]

Sample Output: 5

Sample Input: [7, 7, 7]

Sample Output: -1

*/

```
public class SecondLargestUnique {
```

```
    public static int findSecondLargestUnique(int[] arr) {
        if (arr == null || arr.length == 0) return -1;

        Integer largest = null;
        Integer secondLargest = null;

        // Using a set just to ensure uniqueness of values we have seen
        java.util.HashSet<Integer> seen = new java.util.HashSet<>();

        // First pass: collect all unique numbers
        for (int num : arr) {
            seen.add(num);
        }

        // Second pass: find largest and second largest among unique numbers
        for (int num : seen) {
            if (largest == null || num > largest) {
                secondLargest = largest;
                largest = num;
            } else if ((secondLargest == null || num > secondLargest) && num != largest) {
                secondLargest = num;
            }
        }

        return secondLargest == null ? -1 : secondLargest;
    }

    public static void main(String[] args) {
        int[] input1 = {3, 5, 2, 5, 6, 6, 1};
        int[] input2 = {7, 7, 7};

        System.out.println("Output 1: " + findSecondLargestUnique(input1)); // Expected: 5
        System.out.println("Output 2: " + findSecondLargestUnique(input2)); // Expected: -1
    }
}
```