Summary

Time Complexity: O(n) where n is the number of elements in the array, since we traverse through the array once and another time through the keys which are also O(n).
Space complexity: O(n) since we may store up to n entries in the hash table.

Problem Pattern

1. You need to find a Superstar whose integer doesn't repeat, from the input. This suggests that you would need to traverse the input and keep track of the frequency of each integer representing Superstars.
2. There can be multiple Superstars who have come alone, but you need to return the first such Superstar. This suggests that you will have to keep track of the index of integers as well.
3. How can you perform both these tasks efficiently? Think about which data structure will help you do this. A Hashmap.
4. How will you store 3 things - the integer, its frequency and its index at the same time in a hashmap? If one of these is the key, how can you combine the other 2 to store them as value corresponding to the key? Consider creating your own structure or class.

Problem Approach

1. Create a class 'pair', which stores 2 values - frequency and index.
2. Create a hashmap that stores the integer as key and a pair corresponding to each integer as value.
3. Traverse the array and insert elements and their pairs in the hash table. Now we only need to traverse keys in the hash table and keep track of the smallest index with frequency 1, to find the first non repeating element.

Problem Pseudocode

```
class Pair {

    int freq;
    int index;
}

void firstNonRepeating(int arr[], int n) {

        Map<Integer, Pair> m

        for (i = 0 to n) {

            if (m.containsKey(arr[i]))

                Pair p = m.get(arr[i])

                p.freq += 1

                m.put(arr[i], p)

            else

                Pair p = new Pair()

                p.index = i

                p.freq = 1

                m.put(arr[i], p)

        }



        int min_index = Integer.MAX_VALUE

        int first_unique = -1

        for (int key : m.keySet()) {

            if (m.get(key).freq == 1 && m.get(key).index < min_index) {

                min_index =  m.get(key).index

                first_unique = key

            }

        }

        print(first_unique)

}
```

Alternate Approaches

1. A Simple Solution is to use two loops. The outer loop picks elements one by one and the inner loop checks if the element is present more than once or not.
2. Traverse array and insert elements and their counts in a hash table. Traverse array again and print the first element with count equals to 1. This is similar to the above approach and is done without creating a pair class. However if there are many duplicates in the array, traversing the array again is inefficient.