* Majority element :-

→ Any element occurring more than $n/2$ times.

→ Majority element will always exists in the given array.

* Example :-

Input : nums = [3, 2, 3]

Output : 3

Input : nums = [2, 2, 1, 1, 1, 2, 2]

Output : 2

* Solution :-

* Approach 1 :- Using a hashmap

```
Hashmap <Integer, Integer> countMap = new Hashmap <>();
int majorityElement = arr[0];
for (int num : arr) {
    countMap.put (num , countMap.getOrDefault (num, 0) + 1);
    if (countMap.get (num) > n/2) {
        majorityElement = num;
        break;
    }
}

return majorityElement;
```

Time complexity :- O (n)

Space complexity :- O (n)

* Approach 2 :-

→ If we take one occurrence of the majority element & one occurrence of the non-majority element & cancel both of them with each other, after all the possible cancellations we will still be left with majority elements.

[15 , 15 , 3 , 15 , 3 , 1 , 3 , 15 , 3 , 3 , 15 , 15 , 15 , 15 , 15]

$\uparrow$

currentMajority = ~~15~~ ~~3~~ 15

currentMajorityFreq = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ ~~1~~ ~~2~~ 3

Time complexity :- O (n)

Space complexity :- O (1)