**[Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array/)**

**Problem:**

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in*nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

* Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
* Return k.

**Custom Judge:**

The judge will test your solution with the following code:

int[] nums = [...]; // Input array

int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;

for (int i = 0; i < k; i++) {

assert nums[i] == expectedNums[i];

}

If all assertions pass, then your solution will be **accepted**.

**Example 1:**

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

**Output:** 2, nums = [1,2,\_]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Example 2:**

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]

**Output:** 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

**Explanation:** Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Constraints:**

* 1 <= nums.length <= 3 \* 104
* -100 <= nums[i] <= 100
* nums is sorted in **non-decreasing** order.

**Solution:**

package com.leetcode;  
  
public class RemoveDuplicatesFromSortedArray {  
 public int removeDuplicates(int[] nums) {  
 // Handle edge case: if the input array is null  
 if (nums == null) {  
 throw new IllegalArgumentException("Input array cannot be null");  
 }  
  
 // Handle edge case: if the input array is empty  
 if (nums.length == 0) {  
 return 0;  
 }  
  
 // Initialize the pointer for the unique elements  
 int j = 1;  
  
 // Iterate through the array starting from the second element  
 for (int i = 1; i < nums.length; i++) {  
 // If the current element is not equal to the previous one,  
 // it means it is a unique element  
 if (nums[i] != nums[i - 1]) {  
 nums[j] = nums[i];  
 j++;  
 }  
 }  
 // Return the number of unique elements  
 return j;  
 }  
}