Q1: Find kth permutation :
Ans:

```
import java.util.ArrayList;

import java.util.List;


public class KthPermutation {


   public static String getPermutation(int n, int k) {

      List<Integer> numbers = new ArrayList<>();

      int[] factorial = new int[n + 1];

      factorial[0] = 1;


      for (int i = 1; i <= n; i++) {

         numbers.add(i);

         factorial[i] = factorial[i - 1] * i;

      }


      StringBuilder result = new StringBuilder();


      // Adjust k to be 0-based

      k--;


      for (int i = 1; i <= n; i++) {

         int index = k / factorial[n - i];

         result.append(numbers.get(index));

         numbers.remove(index);

         k -= index * factorial[n - i];

      }
```

```
        return result.toString();
    }

    public static void main(String[] args) {
        int n = 3;
        int k = 3;

        String permutation = getPermutation(n, k);
        System.out.println("The " + k + "th permutation for n=" + n + " is: " + permutation);
    }
}
```

Q.2: Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space

Ans:

```
public class SingleNumber {

    public static int singleNumber(int[] nums) {
        int result = 0;

        // Use XOR to find the single number
        for (int num : nums) {
            result ^= num;
        }
```

```
        return result;
    }

    public static void main(String[] args) {
        int[] nums = {2, 2, 1};
        int single = singleNumber(nums);

        System.out.println("The single number is: " + single);
    }
}
```