1.Program for Basic Calculator

```
public class Calculator {

    private int index=0; // Index used to traverse the input string

    // Recursive method to evaluate arithmetic expressions
    private int calc(String s) {
        if(s==null)
        {
            return -1;
        }
        int result = 0, num = 0, sign = 1;
           while (index < s.length()) {
            char c = s.charAt(index++); // Get the current character

            if (c >= '0' && c <= '9') {
                num = num * 10 + c - '0'; // Accumulate digits to form the number
            } else if (c == '(') {
                num = calc(s); // Recursive call to solve sub-expression inside
parentheses
            } else if (c == ')') {
                return result + sign * num; // Return result with scaled number when
closing parenthesis is encountered
            } else if (c == '+' || c == '-') {
                result =result+ sign * num; // Update intermediate result with the
scaled number
                num = 0;
                if (c == '-') {
                    sign = -1;
                } else {
                    sign = 1;
                }
            }
        }

 return result + sign * num; //Final result is the sum of intermediate result and
scaled number
    }

    public static void main(String[] args) {
        Calculator c = new Calculator();

        String expression ="1 + (2 - 5) + 4";
        int result = c.calc(expression);

        System.out.println("Expression = " + expression);
        System.out.println("Result = " + result);
    }
}
```

OUTPUT

Expression = 1 + (2 - 5) + 4

Result = 2

2.Program for Next Permutation

```java
public class NextPermutation {

    public static void nextPermutation(int[] nums) {
        int n = nums.length;
        int i = n - 2;

        // Find the first element (from the right) that is smaller than the element
next to it
        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }

        if (i >= 0) {
            int j = n - 1;
            // Find the first element (from the right) that is greater than the
above-found element
            while (nums[j] <= nums[i]) {
                j--;
            }
            // Swap these two elements to get the next permutation
            swap(nums, i, j);
        }

        // Reverse the subarray after index i to get the smallest lexicographically
next permutation
        reverse(nums, i + 1, n - 1);
    }

    public static void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    public static void reverse(int[] nums, int start, int end) {
        while (start < end) {
            swap(nums, start, end);
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        int[] nums = {1,2,3,7};
        System.out.println("Number Permutation " + Arrays.toString(nums));

        nextPermutation(nums);

        System.out.println("Next Permutation: " + Arrays.toString(nums));
    }
}
```

OUTPUT

Number Permutation [1, 2, 3, 7]

Next Permutation: [1, 2, 7, 3]