Solution 14-11-23

2666. Allow One Function Call

```
import java.util.function.Supplier;

public class OnceCaller {
    private boolean called = false;
    private Object result;

    public Supplier<Object> once(Supplier<Object> fn) {
        return () -> {
            if (!called) {
                called = true;
                result = fn.get();
                return result;
            } else {
                return null; // or throw an exception, depending on your requirements
            }
        };
    }

    public static void main(String[] args) {
        // Example usage:
        OnceCaller onceCaller = new OnceCaller();

        // Original function
        Supplier<Object> originalFunction = () -> {
            System.out.println("Executing the original function");
            return "Result from the original function";
        };
```

```
        // Wrap the original function using once()
        Supplier<Object> wrappedFunction = onceCaller.once(originalFunction);

        // Call the wrapped function multiple times
        System.out.println(wrappedFunction.get()); // Should print the result
        System.out.println(wrappedFunction.get()); // Should print null or throw an exception
    }
}
```

Output:-

```
java -cp /tmp/V0vcLwLrKG OnceCaller
Executing the original function
Result from the original function
Null
```

60. Permutation Sequence

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class PermutationSequence {
    public static String getPermutation(int n, int k) {
        List<Integer> numbers = new ArrayList<>();
        int[] factorial = new int[n + 1];
        StringBuilder result = new StringBuilder();

        // Create a list of numbers and calculate factorials
        for (int i = 1; i <= n; i++) {
            numbers.add(i);
        }
        calculateFactorials(factorial, n);
```

```
    // Adjust k to 0-based index
    k--;

    for (int i = 1; i <= n; i++) {
        int index = k / factorial[n - i];
        result.append(numbers.remove(index));
        k -= index * factorial[n - i];
    }

    return result.toString();
}

private static void calculateFactorials(int[] factorial, int n) {
    factorial[0] = 1;
    for (int i = 1; i <= n; i++) {
        factorial[i] = i * factorial[i - 1];
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the length of the sequence (n): ");
    int n = scanner.nextInt();

    System.out.print("Enter the desired permutation (k): ");
    int k = scanner.nextInt();

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

```
        scanner.close();
    }
}
```

Output:-

```
java -cp /tmp/V0vcLwLrKG PermutationSequence
Enter the length of the sequence (n): 3
Enter the desired permutation (k): 3
The kth permutation is: 213
```