KMP

```java
package com.wipro;

public class Main {
    // JAVA program for implementation of KMP pattern searching algorithm

    // Method to search for the pattern in the given text using KMP algorithm
    public void KMPSearch(String pat, String txt) {
        int M = pat.length();
        int N = txt.length();

        // Create lps[] that will hold the longest prefix suffix values for pattern
        int lps[] = new int[M];
        int j = 0; // Index for pat[]

        // Preprocess the pattern (calculate lps[] array)
        computeLPSArray(pat, M, lps);

        int i = 0; // Index for txt[]
        while ((N - i) >= (M - j)) {
            if (pat.charAt(j) == txt.charAt(i)) {
                j++;
                i++;
            }
            if (j == M) {
                System.out.println("Found pattern at index " + (i - j));
                j = lps[j - 1];
            } else if (i < N && pat.charAt(j) != txt.charAt(i)) {
                // Mismatch after j matches
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    i++;
                }
            }
        }
    }

    // Method to preprocess the pattern and create lps[] array
    void computeLPSArray(String pat, int M, int lps[]) {
        // Length of the previous longest prefix suffix
        int len = 0;
        int i = 1;
        lps[0] = 0; // lps[0] is always 0

        // Loop calculates lps[i] for i = 1 to M-1
        while (i < M) {
            if (pat.charAt(i) == pat.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            } else {
                if (len != 0) {
                    len = lps[len - 1];
                } else {
                    lps[i] = len;
                    i++;
```

```
                }
            }
        }
    }

    // Driver code
    public static void main(String[] args) {
        String txt = "ABABDABACDABABCABAB";
        String pat = "ABABCABAB";
        new Main().KMPSearch(pat, txt);
    }
}
```