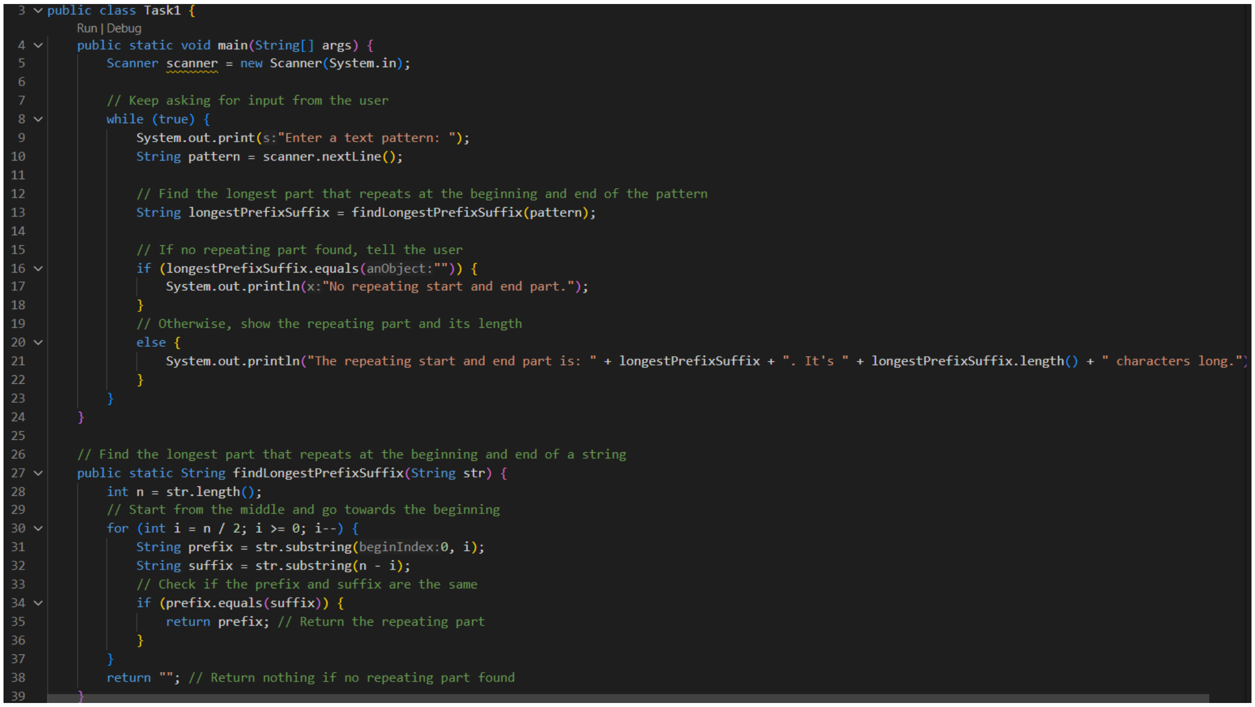
Name: IBRAHIM ALSHAYEA; ID: 202176470, DATE: 5/4/2024

TASK1:

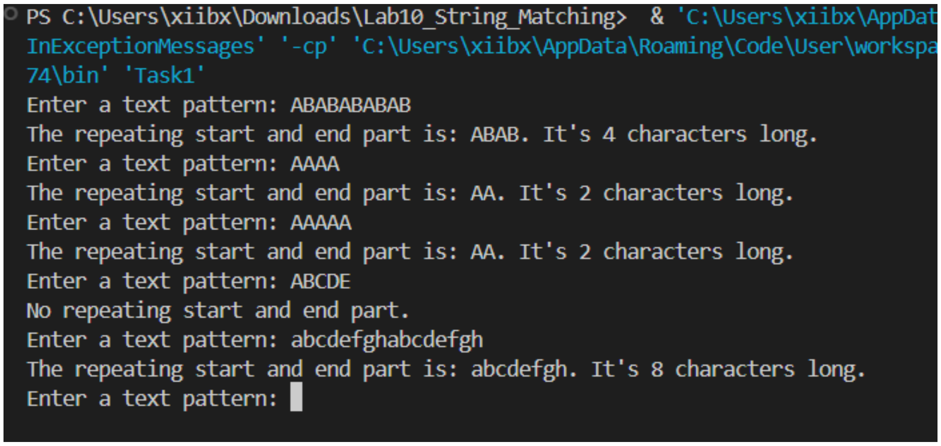
Task1class asks for a text pattern. It then finds the longest part at the beginning that's the same as the end of the pattern, without any overlap. After that, it shows this part and how long it is. If there's no such part, it tells you.

Task1 Code:



```java
public class Task1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Keep asking for input from the user
        while (true) {
            System.out.print(s:"Enter a text pattern: ");
            String pattern = scanner.nextLine();

            // Find the longest part that repeats at the beginning and end of the pattern
            String longestPrefixSuffix = findLongestPrefixSuffix(pattern);

            // If no repeating part found, tell the user
            if (longestPrefixSuffix.equals(anObject:"")) {
                System.out.println(x:"No repeating start and end part.");
            }
            // Otherwise, show the repeating part and its length
            else {
                System.out.println("The repeating start and end part is: " + longestPrefixSuffix + ". It's " + longestPrefixSuffix.length() + " characters long.")
            }
        }
    }

    // Find the longest part that repeats at the beginning and end of a string
    public static String findLongestPrefixSuffix(String str) {
        int n = str.length();
        // Start from the middle and go towards the beginning
        for (int i = n / 2; i >= 0; i--) {
            String prefix = str.substring(beginIndex:0, i);
            String suffix = str.substring(n - i);
            // Check if the prefix and suffix are the same
            if (prefix.equals(suffix)) {
                return prefix; // Return the repeating part
            }
        }
        return ""; // Return nothing if no repeating part found
    }
```

Task1 Output:

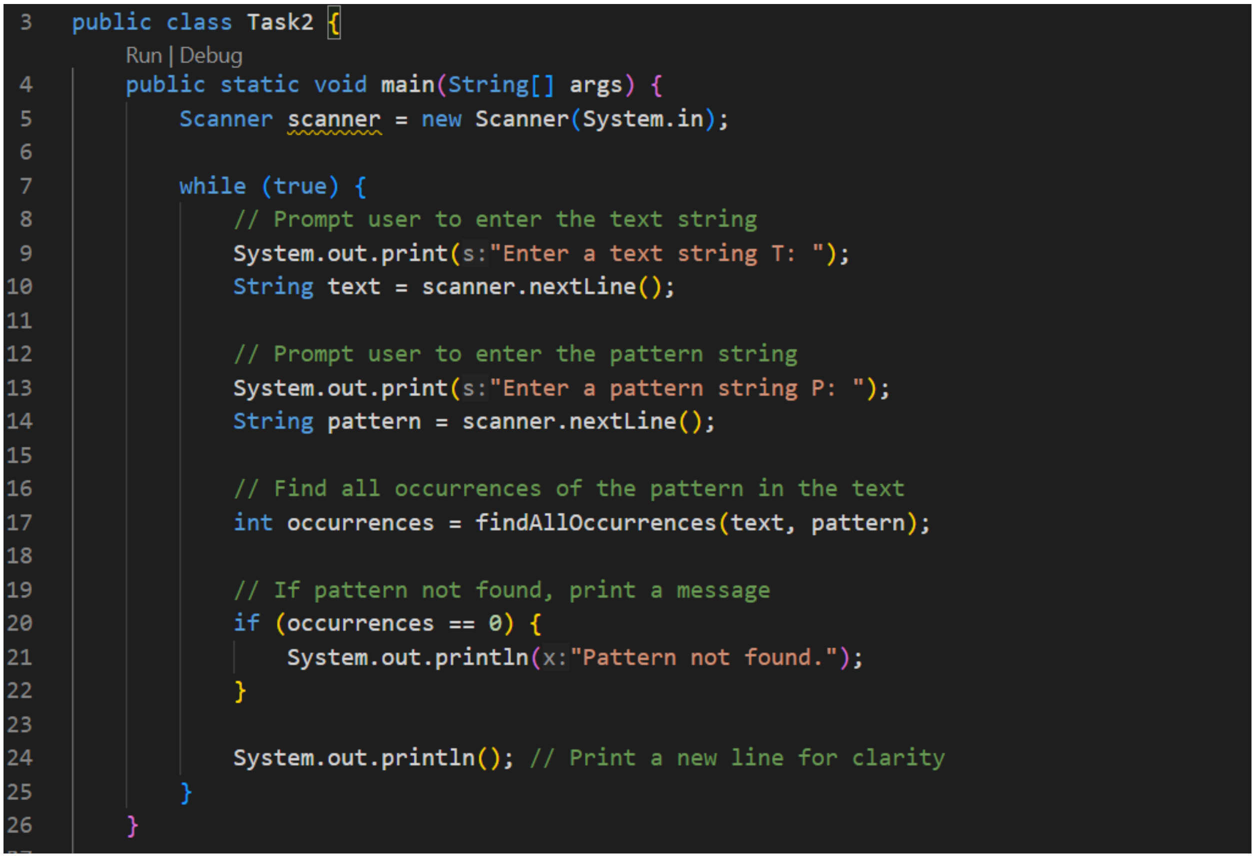


```
PS C:\Users\xiibx\Downloads\Lab10_String_Matching>  & 'C:\Users\xiibx\AppDat
InExceptionMessages' '-cp' 'C:\Users\xiibx\AppData\Roaming\Code\User\workspa
74\bin' 'Task1'
Enter a text pattern: ABABABABAB
The repeating start and end part is: ABAB. It's 4 characters long.
Enter a text pattern: AAAA
The repeating start and end part is: AA. It's 2 characters long.
Enter a text pattern: AAAAA
The repeating start and end part is: AA. It's 2 characters long.
Enter a text pattern: ABCDE
No repeating start and end part.
Enter a text pattern: abcdefghabcdefgh
The repeating start and end part is: abcdefgh. It's 8 characters long.
Enter a text pattern: 
```

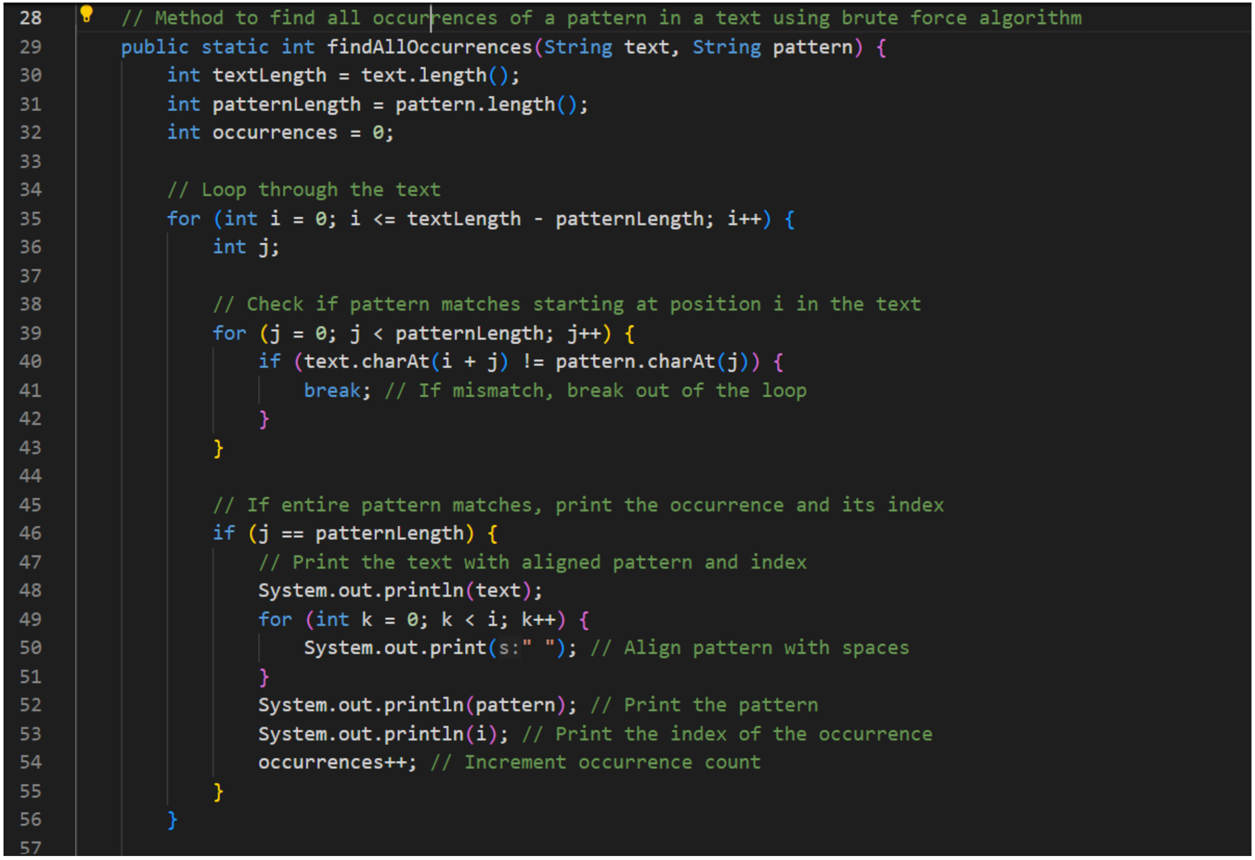
Task2:

Task2 class prompts users to enter a text string and a pattern string. Then, it finds and displays all occurrences of the pattern within the text, using the brute force algorithm. If the pattern is not found, it informs the user.

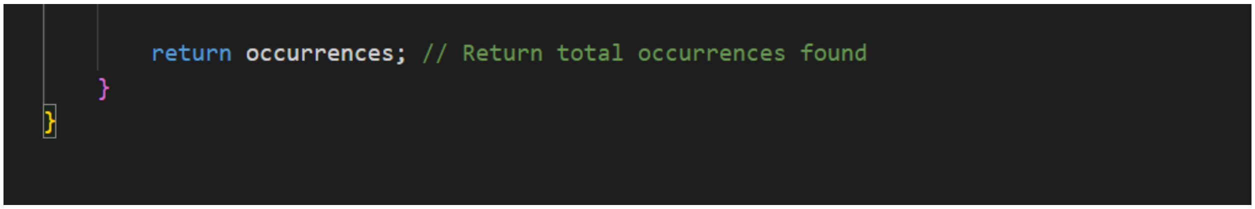
Task2 Code:



```java
public class Task2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (true) {
            // Prompt user to enter the text string
            System.out.print(s:"Enter a text string T: ");
            String text = scanner.nextLine();

            // Prompt user to enter the pattern string
            System.out.print(s:"Enter a pattern string P: ");
            String pattern = scanner.nextLine();

            // Find all occurrences of the pattern in the text
            int occurrences = findAllOccurrences(text, pattern);

            // If pattern not found, print a message
            if (occurrences == 0) {
                System.out.println(x:"Pattern not found.");
            }

            System.out.println(); // Print a new line for clarity
        }
    }
```

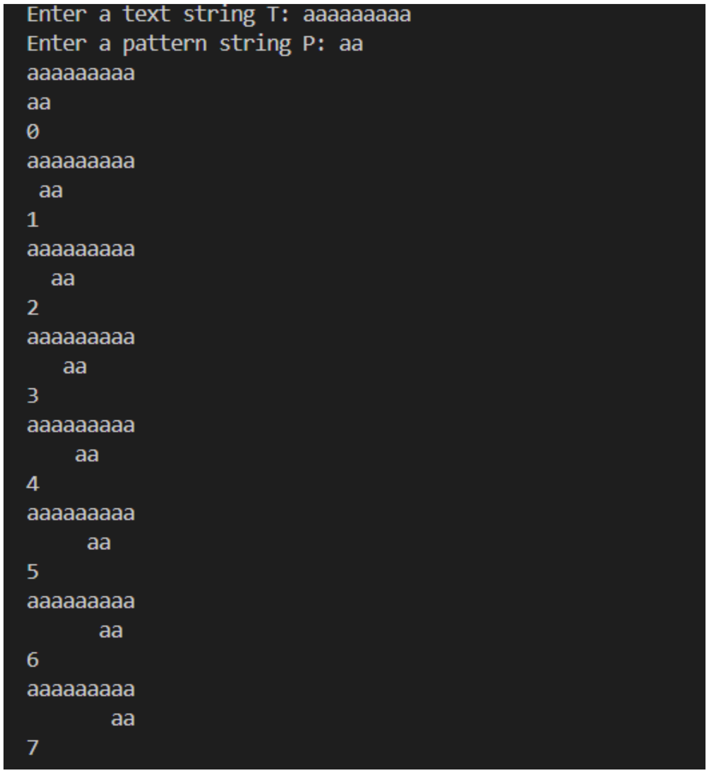


```java
    // Method to find all occurrences of a pattern in a text using brute force algorithm
    public static int findAllOccurrences(String text, String pattern) {
        int textLength = text.length();
        int patternLength = pattern.length();
        int occurrences = 0;

        // Loop through the text
        for (int i = 0; i <= textLength - patternLength; i++) {
            int j;

            // Check if pattern matches starting at position i in the text
            for (j = 0; j < patternLength; j++) {
                if (text.charAt(i + j) != pattern.charAt(j)) {
                    break; // If mismatch, break out of the loop
                }
            }

            // If entire pattern matches, print the occurrence and its index
            if (j == patternLength) {
                // Print the text with aligned pattern and index
                System.out.println(text);
                for (int k = 0; k < i; k++) {
                    System.out.print(s:" "); // Align pattern with spaces
                }
                System.out.println(pattern); // Print the pattern
                System.out.println(i); // Print the index of the occurrence
                occurrences++; // Increment occurrence count
            }
        }

```

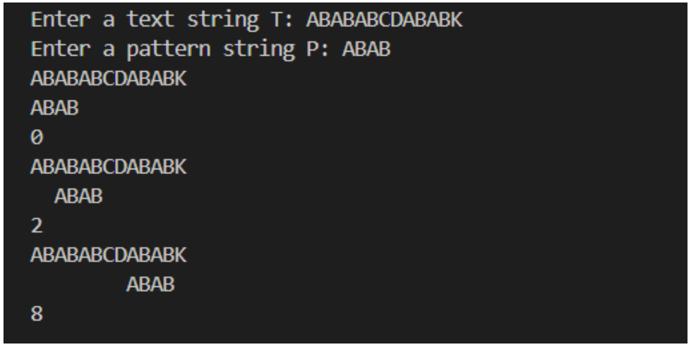


```
        return occurrences; // Return total occurrences found
    }
}
```

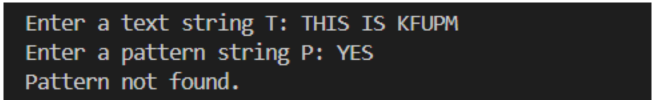
Task2 Output:



```
Enter a text string T: aaaaaaaaa
Enter a pattern string P: aa
aaaaaaaaa
aa
0
aaaaaaaaa
 aa
1
aaaaaaaaa
  aa
2
aaaaaaaaa
   aa
3
aaaaaaaaa
    aa
4
aaaaaaaaa
     aa
5
aaaaaaaaa
      aa
6
aaaaaaaaa
       aa
7
```



```
Enter a text string T: ABABABCDABABK
Enter a pattern string P: ABAB
ABABABCDABABK
ABAB
0
ABABABCDABABK
  ABAB
2
ABABABCDABABK
        ABAB
8
```

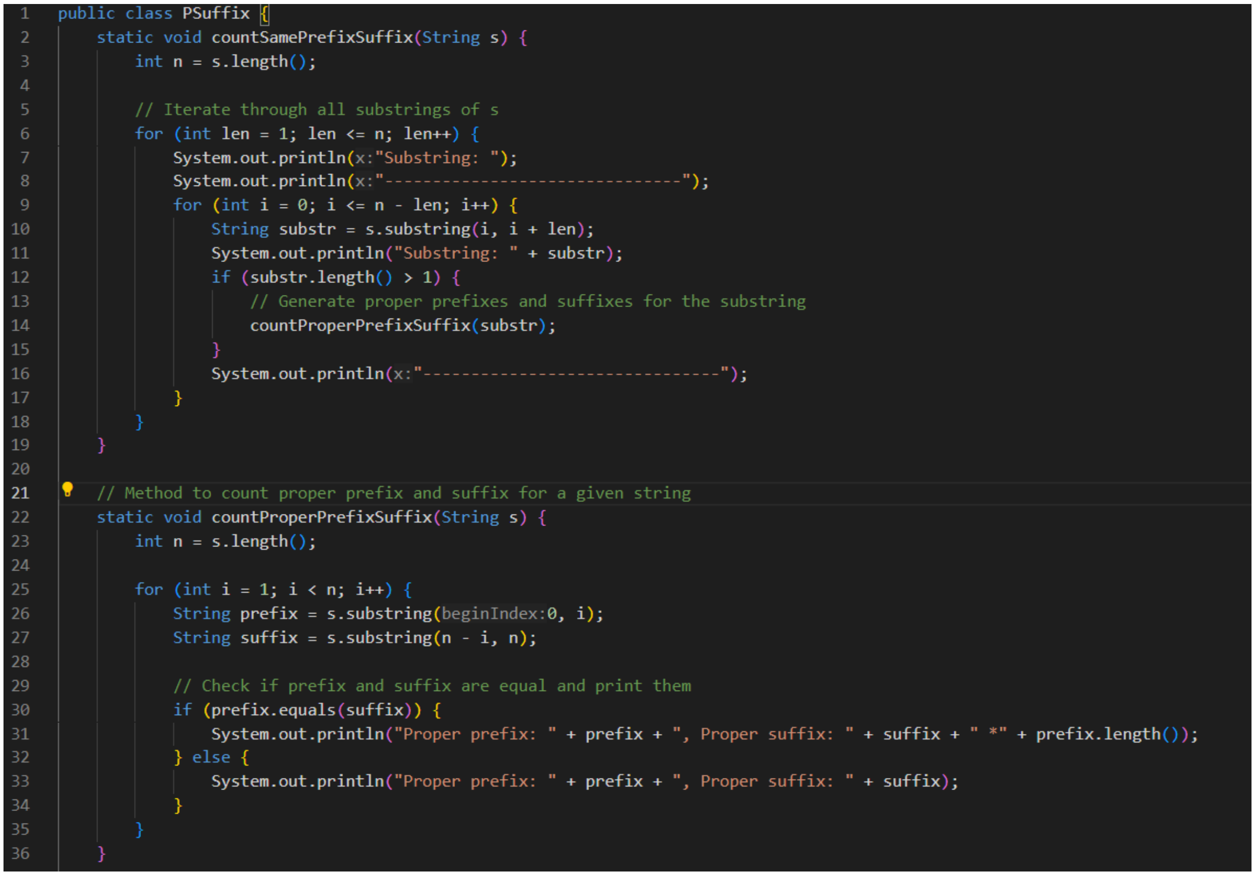


```
Enter a text string T: THIS IS KFUPM
Enter a pattern string P: YES
Pattern not found.
```

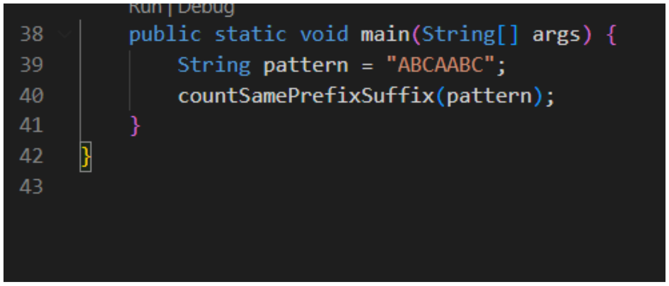
Task3a:

PSuffix class analyzes a given string to identify and print all substrings along with their proper prefixes and suffixes. It iterates through each substring of the input string, prints the substring, and then calculates and prints the proper prefixes and suffixes for each substring. If a substring has a proper prefix that matches its suffix, it indicates the length of the matching prefix. It demonstrates this functionality using the pattern "ABCAABC".

Task3a Code:

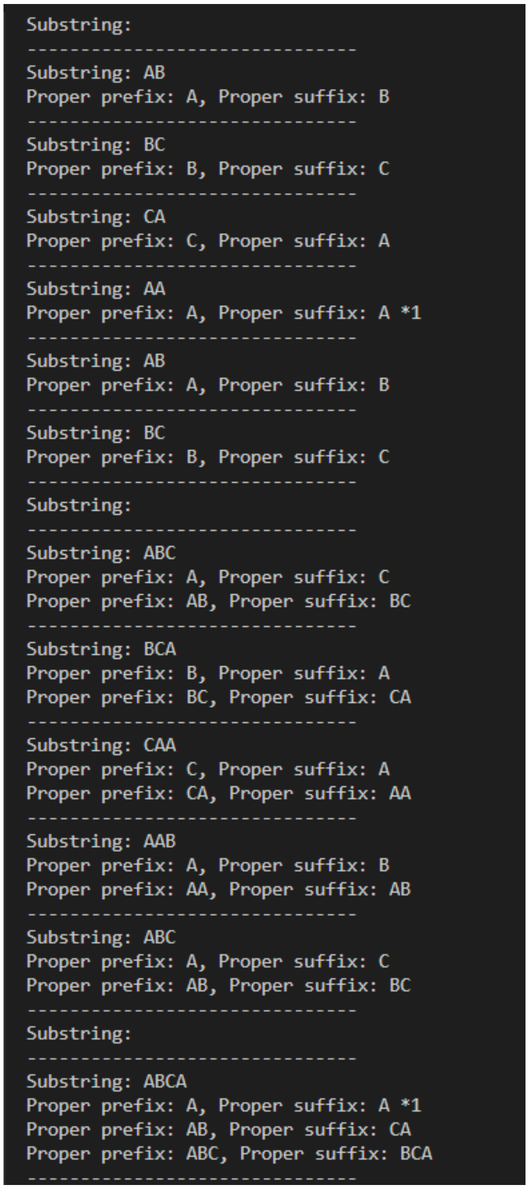


```java
public class PSuffix {
    static void countSamePrefixSuffix(String s) {
        int n = s.length();

        // Iterate through all substrings of s
        for (int len = 1; len <= n; len++) {
            System.out.println(x:"Substring: ");
            System.out.println(x:"--------------------------------");
            for (int i = 0; i <= n - len; i++) {
                String substr = s.substring(i, i + len);
                System.out.println("Substring: " + substr);
                if (substr.length() > 1) {
                    // Generate proper prefixes and suffixes for the substring
                    countProperPrefixSuffix(substr);
                }
                System.out.println(x:"--------------------------------");
            }
        }
    }

    // Method to count proper prefix and suffix for a given string
    static void countProperPrefixSuffix(String s) {
        int n = s.length();

        for (int i = 1; i < n; i++) {
            String prefix = s.substring(beginIndex:0, i);
            String suffix = s.substring(n - i, n);

            // Check if prefix and suffix are equal and print them
            if (prefix.equals(suffix)) {
                System.out.println("Proper prefix: " + prefix + ", Proper suffix: " + suffix + " *" + prefix.length());
            } else {
                System.out.println("Proper prefix: " + prefix + ", Proper suffix: " + suffix);
            }
        }
    }
```



```java
    public static void main(String[] args) {
        String pattern = "ABCAABC";
        countSamePrefixSuffix(pattern);
    }
}
```

Task3a sample output:



```
Substring: 
-------------------------------
Substring: AB
Proper prefix: A, Proper suffix: B
-------------------------------
Substring: BC
Proper prefix: B, Proper suffix: C
-------------------------------
Substring: CA
Proper prefix: C, Proper suffix: A
-------------------------------
Substring: AA
Proper prefix: A, Proper suffix: A *1
-------------------------------
Substring: AB
Proper prefix: A, Proper suffix: B
-------------------------------
Substring: BC
Proper prefix: B, Proper suffix: C
-------------------------------
Substring: 
-------------------------------
Substring: ABC
Proper prefix: A, Proper suffix: C
Proper prefix: AB, Proper suffix: BC
-------------------------------
Substring: BCA
Proper prefix: B, Proper suffix: A
Proper prefix: BC, Proper suffix: CA
-------------------------------
Substring: CAA
Proper prefix: C, Proper suffix: A
Proper prefix: CA, Proper suffix: AA
-------------------------------
Substring: AAB
Proper prefix: A, Proper suffix: B
Proper prefix: AA, Proper suffix: AB
-------------------------------
Substring: ABC
Proper prefix: A, Proper suffix: C
Proper prefix: AB, Proper suffix: BC
-------------------------------
Substring: 
-------------------------------
Substring: ABCA
Proper prefix: A, Proper suffix: A *1
Proper prefix: AB, Proper suffix: CA
Proper prefix: ABC, Proper suffix: BCA
-------------------------------
```

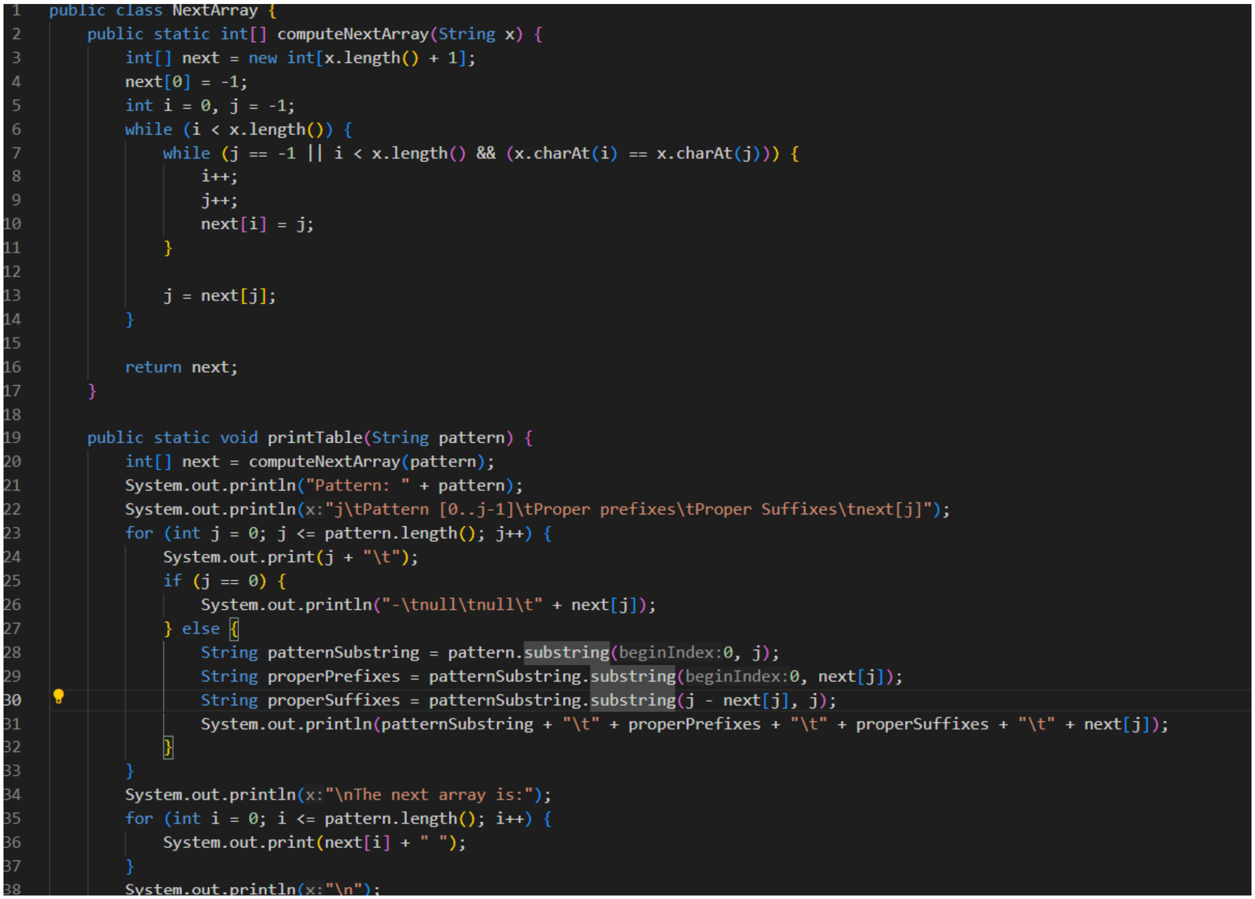
Task3b:

I solveld the tabels manualy, same as the output.

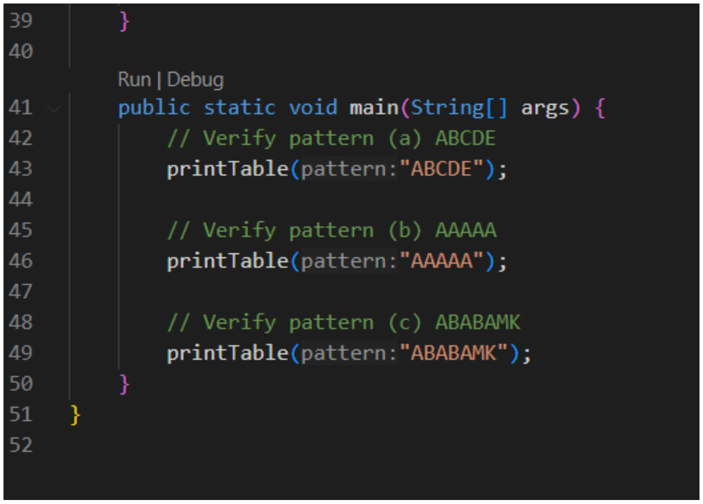
Task3b code explenation:

The NextArray class find the next array for a given pattern. Then, there's printTable, which uses computeNextArray to display a table showing how this array is calculated. The main method tries out computeNextArray with three different patterns: "ABCDE", "AAAAA", and "ABABAMK", printing their tables and resulting next arrays.

Task3b Code:

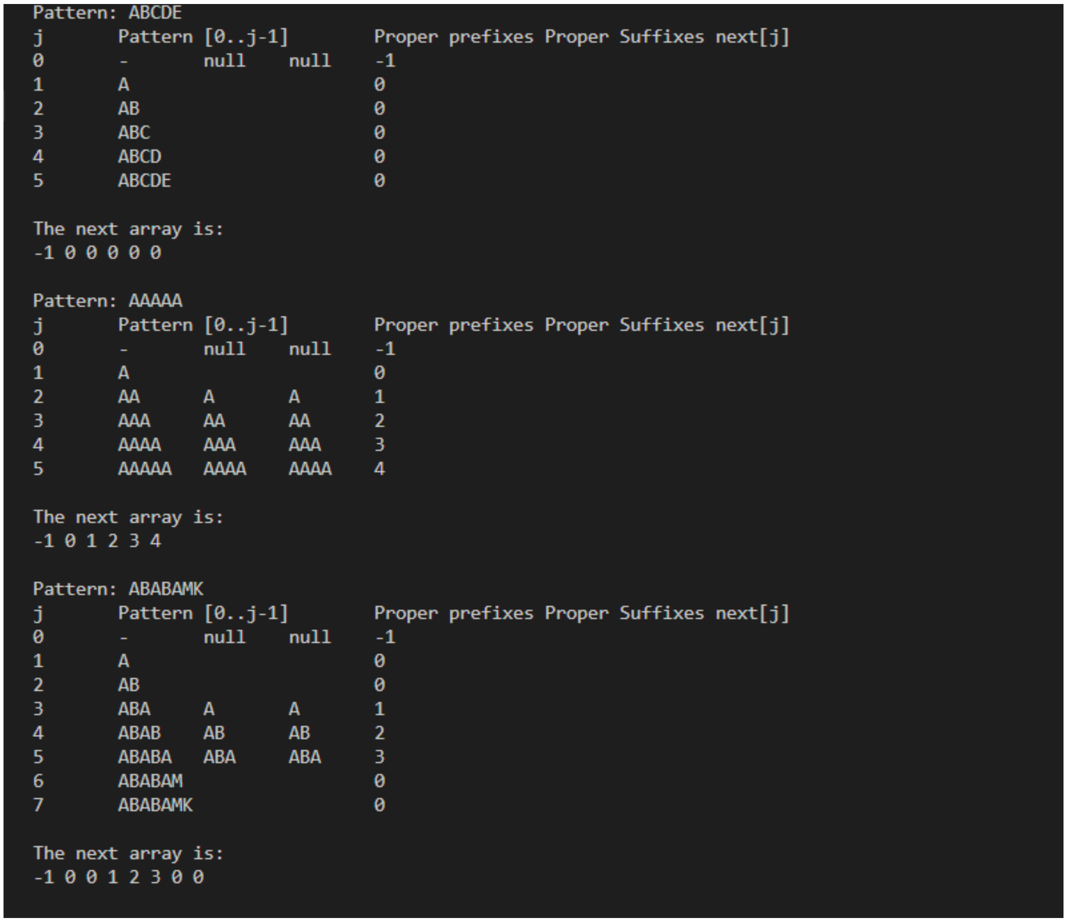


```java
public class NextArray {
    public static int[] computeNextArray(String x) {
        int[] next = new int[x.length() + 1];
        next[0] = -1;
        int i = 0, j = -1;
        while (i < x.length()) {
            while (j == -1 || i < x.length() && (x.charAt(i) == x.charAt(j))) {
                i++;
                j++;
                next[i] = j;
            }

            j = next[j];
        }

        return next;
    }

    public static void printTable(String pattern) {
        int[] next = computeNextArray(pattern);
        System.out.println("Pattern: " + pattern);
        System.out.println(x:"j\tPattern [0..j-1]\tProper prefixes\tProper Suffixes\tnext[j]");
        for (int j = 0; j <= pattern.length(); j++) {
            System.out.print(j + "\t");
            if (j == 0) {
                System.out.println("-\tnull\tnull\t" + next[j]);
            } else {
                String patternSubstring = pattern.substring(beginIndex:0, j);
                String properPrefixes = patternSubstring.substring(beginIndex:0, next[j]);
                String properSuffixes = patternSubstring.substring(j - next[j], j);
                System.out.println(patternSubstring + "\t" + properPrefixes + "\t" + properSuffixes + "\t" + next[j]);
            }
        }
        System.out.println(x:"\nThe next array is:");
        for (int i = 0; i <= pattern.length(); i++) {
            System.out.print(next[i] + " ");
        }
        System.out.println(x:"\n");
```



```java
    }

    Run | Debug
    public static void main(String[] args) {
        // Verify pattern (a) ABCDE
        printTable(pattern:"ABCDE");

        // Verify pattern (b) AAAAA
        printTable(pattern:"AAAAA");

        // Verify pattern (c) ABABAMK
        printTable(pattern:"ABABAMK");
    }
}
```

Task3b Output:

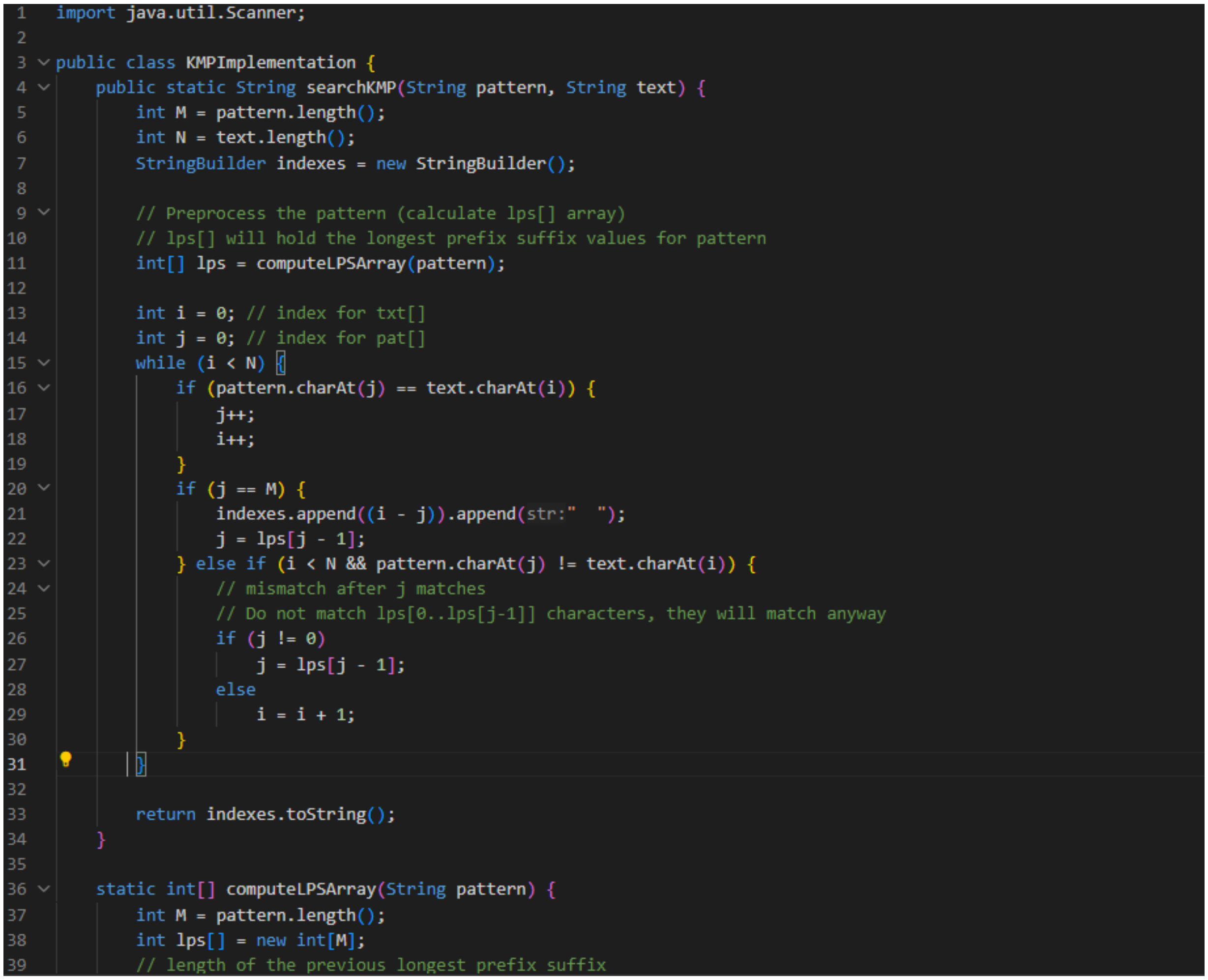


```
Pattern: ABCDE
j       Pattern [0..j-1]        Proper prefixes Proper Suffixes next[j]
0       -       null    null    -1
1       A                       0
2       AB                      0
3       ABC                     0
4       ABCD                    0
5       ABCDE                   0

The next array is:
-1 0 0 0 0 0

Pattern: AAAAA
j       Pattern [0..j-1]        Proper prefixes Proper Suffixes next[j]
0       -       null    null    -1
1       A                       0
2       AA      A       A       1
3       AAA     AA      AA      2
4       AAAA    AAA     AAA     3
5       AAAAA   AAAA    AAAA    4

The next array is:
-1 0 1 2 3 4

Pattern: ABABAMK
j       Pattern [0..j-1]        Proper prefixes Proper Suffixes next[j]
0       -       null    null    -1
1       A                       0
2       AB                      0
3       ABA     A       A       1
4       ABAB    AB      AB      2
5       ABABA   ABA     ABA     3
6       ABABAM                  0
7       ABABAMK                 0

The next array is:
-1 0 0 1 2 3 0 0
```

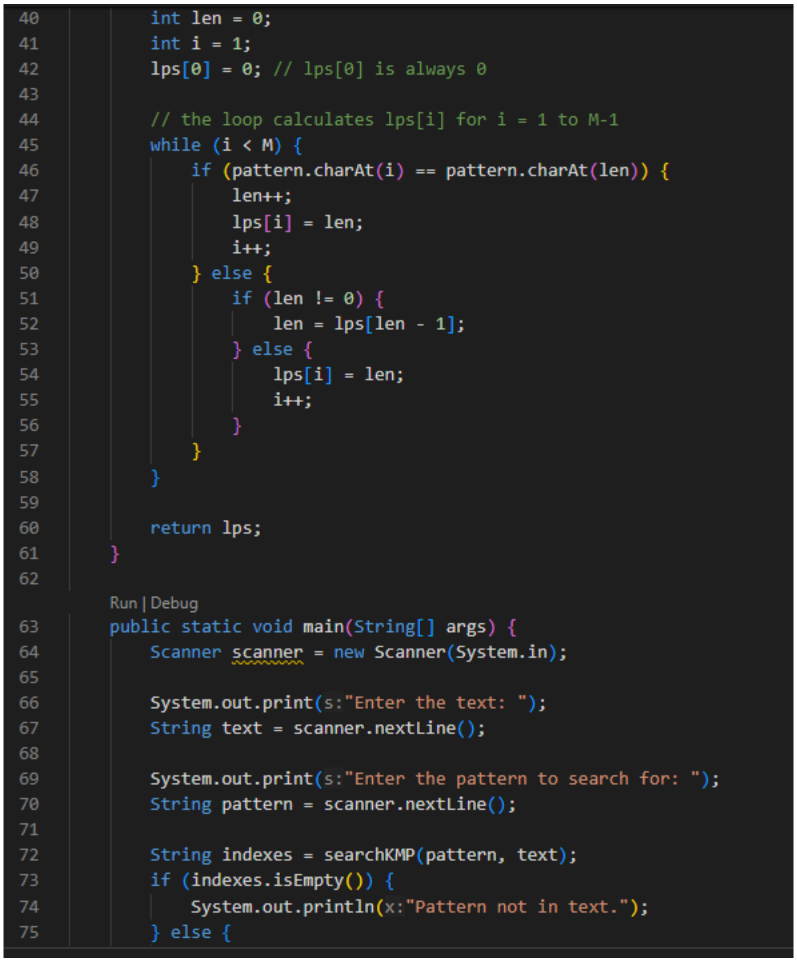
Task4:

In the KMPImplementation class, I simplified the code by using a StringBuilder to gather indexes of pattern occurrences. Additionally, I adjusted the computeLPSArray method to directly return an array of integers representing the longest prefix suffix values for the pattern.

The code:

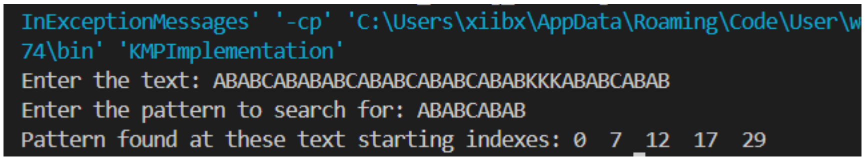


```java
import java.util.Scanner;

public class KMPImplementation {
    public static String searchKMP(String pattern, String text) {
        int M = pattern.length();
        int N = text.length();
        StringBuilder indexes = new StringBuilder();

        // Preprocess the pattern (calculate lps[] array)
        // lps[] will hold the longest prefix suffix values for pattern
        int[] lps = computeLPSArray(pattern);

        int i = 0; // index for txt[]
        int j = 0; // index for pat[]
        while (i < N) {
            if (pattern.charAt(j) == text.charAt(i)) {
                j++;
                i++;
            }
            if (j == M) {
                indexes.append((i - j)).append(str:"  ");
                j = lps[j - 1];
            } else if (i < N && pattern.charAt(j) != text.charAt(i)) {
                // mismatch after j matches
                // Do not match lps[0..lps[j-1]] characters, they will match anyway
                if (j != 0)
                    j = lps[j - 1];
                else
                    i = i + 1;
            }
        }

        return indexes.toString();
    }

    static int[] computeLPSArray(String pattern) {
        int M = pattern.length();
        int lps[] = new int[M];
        // length of the previous longest prefix suffix
```

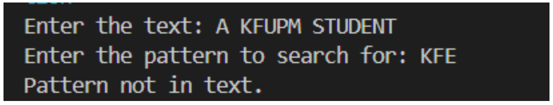


```
        int len = 0;
        int i = 1;
        lps[0] = 0; // lps[0] is always 0

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

        return lps;
    }

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

        System.out.print(s:"Enter the text: ");
        String text = scanner.nextLine();

        System.out.print(s:"Enter the pattern to search for: ");
        String pattern = scanner.nextLine();

        String indexes = searchKMP(pattern, text);
        if (indexes.isEmpty()) {
            System.out.println(x:"Pattern not in text.");
        } else {
```

Task4 output:



```
InExceptionMessages' '-cp' 'C:\Users\xiibx\AppData\Roaming\Code\User\w
74\bin' 'KMPImplementation'
Enter the text: ABABCABABABCABABCABABCABABKKKABABCABAB
Enter the pattern to search for: ABABCABAB
Pattern found at these text starting indexes: 0  7  12  17  29
```



```
Enter the text: A KFUPM STUDENT
Enter the pattern to search for: KFE
Pattern not in text.
```