KMP.java

Below is the syntax highlighted version of KMP.java from §5.3 Substring Search.

```java
/******************************************************************************
 *  Compilation:  javac KMP.java
 *  Execution:    java KMP pattern text
 *  Dependencies: StdOut.java
 *
 *  Reads in two strings, the pattern and the input text, and
 *  searches for the pattern in the input text using the
 *  KMP algorithm.
 *
 *  % java KMP abracadabra abacadabrabracabracadabrabrabracad
 *  text:    abacadabrabracabracadabrabrabracad
 *  pattern:               abracadabra
 *
 *  % java KMP rab abacadabrabracabracadabrabrabracad
 *  text:    abacadabrabracabracadabrabrabracad
 *  pattern:         rab
 *
 *  % java KMP bcara abacadabrabracabracadabrabrabracad
 *  text:    abacadabrabracabracadabrabrabracad
 *  pattern:                                   bcara
 *
 *  % java KMP rabrabracad abacadabrabracabracadabrabrabracad
 *  text:    abacadabrabracabracadabrabrabracad
 *  pattern:                        rabrabracad
 *
 *  % java KMP abacad abacadabrabracabracadabrabrabracad
 *  text:    abacadabrabracabracadabrabrabracad
 *  pattern: abacad
 *
 ******************************************************************************/

/**
 *  The <tt>KMP</tt> class finds the first occurrence of a pattern string
 *  in a text string.
 *  <p>
 *  This implementation uses a version of the Knuth-Morris-Pratt substring search
 *  algorithm. The version takes time as space proportional to
 *  <em>N</em> + <em>M R</em> in the worst case, where <em>N</em> is the length
 *  of the text string, <em>M</em> is the length of the pattern, and <em>R</em>
 *  is the alphabet size.
 *  <p>
 *  For additional documentation,
 *  see <a href="http://algs4.cs.princeton.edu/53substring">Section 5.3</a> of
 *  <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 */
public class KMP {
    private final int R;       // the radix
    private int[][] dfa;       // the KMP automoton

    private char[] pattern;    // either the character array for the pattern
    private String pat;        // or the pattern string

    /**
     * Preprocesses the pattern string.
     *
     * @param pat the pattern string
     */
    public KMP(String pat) {
        this.R = 256;
        this.pat = pat;

        // build DFA from pattern
        int M = pat.length();
        dfa = new int[R][M];
        dfa[pat.charAt(0)][0] = 1;
        for (int X = 0, j = 1; j < M; j++) {
            for (int c = 0; c < R; c++)
                dfa[c][j] = dfa[c][X];     // Copy mismatch cases.
            dfa[pat.charAt(j)][j] = j+1;   // Set match case.
            X = dfa[pat.charAt(j)][X];     // Update restart state.
        }
    }

    /**
     * Preprocesses the pattern string.
     *
     * @param pattern the pattern string
     * @param R the alphabet size
     */
    public KMP(char[] pattern, int R) {
        this.R = R;
        this.pattern = new char[pattern.length];
        for (int j = 0; j < pattern.length; j++)
            this.pattern[j] = pattern[j];

        // build DFA from pattern
        int M = pattern.length;
        dfa = new int[R][M];
        dfa[pattern[0]][0] = 1;
        for (int X = 0, j = 1; j < M; j++) {
            for (int c = 0; c < R; c++)
                dfa[c][j] = dfa[c][X];     // Copy mismatch cases.
            dfa[pattern[j]][j] = j+1;      // Set match case.
            X = dfa[pattern[j]][X];        // Update restart state.
        }
    }

    /**
     * Returns the index of the first occurrrence of the pattern string
     * in the text string.
     *
     * @param  txt the text string
     * @return the index of the first occurrence of the pattern string
     *         in the text string; N if no such match
     */
    public int search(String txt) {

        // simulate operation of DFA on text
        int M = pat.length();
        int N = txt.length();
        int i, j;
        for (i = 0, j = 0; i < N && j < M; i++) {
            j = dfa[txt.charAt(i)][j];
        }
        if (j == M) return i - M;    // found
        return N;                    // not found
    }

    /**
     * Returns the index of the first occurrrence of the pattern string
     * in the text string.
     *
     * @param  text the text string
     * @return the index of the first occurrence of the pattern string
     *         in the text string; N if no such match
     */
    public int search(char[] text) {

        // simulate operation of DFA on text
        int M = pattern.length;
        int N = text.length;
        int i, j;
        for (i = 0, j = 0; i < N && j < M; i++) {
            j = dfa[text[i]][j];
        }
        if (j == M) return i - M;    // found
        return N;                    // not found
    }


    /**
     * Takes a pattern string and an input string as command-line arguments;
     * searches for the pattern string in the text string; and prints
     * the first occurrence of the pattern string in the text string.
     */
    public static void main(String[] args) {
        String pat = args[0];
        String txt = args[1];
        char[] pattern = pat.toCharArray();
        char[] text    = txt.toCharArray();

        KMP kmp1 = new KMP(pat);
        int offset1 = kmp1.search(txt);

        KMP kmp2 = new KMP(pattern, 256);
        int offset2 = kmp2.search(text);

        // print results
        StdOut.println("text:    " + txt);

        StdOut.print("pattern: ");
        for (int i = 0; i < offset1; i++)
            StdOut.print(" ");
        StdOut.println(pat);

        StdOut.print("pattern: ");
        for (int i = 0; i < offset2; i++)
            StdOut.print(" ");
        StdOut.println(pat);
    }
}
```