REGULAR EXPRESSION

|  |  |  |
| --- | --- | --- |
| No. | Method | Description |
| 1 | boolean matches() | test whether the regular expression matches the pattern. |
| 2 | boolean find() | finds the next expression that matches the pattern. |
| 3 | boolean find(int start) | finds the next expression that matches the pattern from the given start number. |
| 4 | String group() | returns the matched subsequence. |
| 5 | int start() | returns the starting index of the matched subsequence. |
| 6 | int end() | returns the ending index of the matched subsequence. |
| 7 | int groupCount() | returns the total number of the matched subsequence. |

|  |  |  |
| --- | --- | --- |
| No. | Method | Description |
| 1 | static Pattern compile(String regex) | compiles the given regex and returns the instance of the Pattern. |
| 2 | Matcher matcher(CharSequence input) | creates a matcher that matches the given input with the pattern. |
| 3 | static boolean matches(String regex, CharSequence input) | It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern. |
| 4 | String[] split(CharSequence input) | splits the given input string around matches of given pattern. |
| 5 | String pattern() | returns the regex pattern. |

Pattern p = Pattern.compile("s."); //. represents single character

Matcher m = p.matcher("as");

boolean b = m.matches();

System.out.println(b);

System.out.println(c);

}

}

import java.util.regex.\*;

public class RegexExample2{

public static void main(String args[]){ //dot represents single character

System.out.println(Pattern.matches(".s", "as")); //true(2nd char is s)

System.out.println(Pattern.matches(".s", "mk")); //false(2nd char is not s)

System.out.println(Pattern.matches(".s", "mst")); //false(has more than 2 char)

System.out.println(Pattern.matches(".s", "amms")); //false(has more than 2 char)

System.out.println(Pattern.matches("..s", "mas")); //true(3rd char is s)

}}

|  |  |  |
| --- | --- | --- |
| No. | Character Class | Description |
| 1 | [abc] | a, b, or c (simple class) |
| 2 | [^abc] | Any character except a, b, or c (negation) |
| 3 | [a-zA-Z] | a through z or A through Z, inclusive (range) |
| 4 | [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| 5 | [a-z&&[def]] | d, e, or f (intersection) |
| 6 | [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| 7 | [a-z&&[^m-p]] | a through z, and not m through p: [a-lq-z](subtraction) |

## regular Expression Syntax

|  |  |
| --- | --- |
| **Subexpression** | **Matches** |
| ^ | Matches the beginning of the line. |
| $ | Matches the end of the line. |
| . | Matches any single character except newline. Using **m** option allows it to match the newline as well. |
| [...] | Matches any single character in brackets. |
| [^...] | Matches any single character not in brackets. |
| \A | Beginning of the entire string. |
| \z | End of the entire string. |
| \Z | End of the entire string except allowable final line terminator. |
| re\* | Matches 0 or more occurrences of the preceding expression. |
| re+ | Matches 1 or more of the previous thing. |
| re? | Matches 0 or 1 occurrence of the preceding expression. |
| re{ n} | Matches exactly n number of occurrences of the preceding expression. |
| re{ n,} | Matches n or more occurrences of the preceding expression. |
| re{ n, m} | Matches at least n and at most m occurrences of the preceding expression. |
| a| b | Matches either a or b. |