Java Regex

**Java Regex**  is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations.

It is widely used to define constraint on strings such as password and email validation.

Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class

## Matcher class

It implements **MatchResult** interface. It is a *regex engine* i.e. used to perform match operations on a character sequence.

## Pattern class

It is the *compiled version of a regular expression*. It is used to define a pattern for the regex engine.

Example:

**import** java.util.regex.\*;

**public** **class** RegexExample1{

**public** **static** **void** main(String args[]){

//1st way

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

Matcher m = p.matcher("3Edge");

**boolean** b = m.matches();

//2nd way

**boolean** b2=Pattern.compile("..d..").matcher("3Edge").matches();

//3rd way

**boolean** b3 = Pattern.matches("..d..", "3Edge");

System.out.println(b+" "+b2+" "+b3);

}}

## Character Classes

|  |  |  |
| --- | --- | --- |
| **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) |

**Quantifiers**

|  |  |
| --- | --- |
| **Regex** | **Description** |
| X? | X occurs once or not at all |
| X+ | X occurs once or more times |
| X\* | X occurs zero or more times |
| X{n} | X occurs n times only |
| X{n,} | X occurs n or more times |
| X{y,z} | X occurs at least y times but less than z times |

**Metacharacters**

|  |  |
| --- | --- |
| **Regex** | **Description** |
| . | Any character (may or may not match terminator) |
| \d | Any digits, short of [0-9] |
| \D | Any non-digit, short for [^0-9] |
| \s | Any whitespace character, short for [\t\n\x0B\f\r] |
| \S | Any non-whitespace character, short for [^\s] |
| \w | Any word character, short for [a-zA-Z\_0-9] |
| \W | Any non-word character, short for [^\w] |

**Hints:**

## simple class:

The most basic form of a character class is to simply place a set of characters side-by-side within square brackets. For example, the regular expression [bcr]at will match the words "bat", "cat", or "rat" because it defines a character class (accepting either "b", "c", or "r") as its first character.

**Ranges:**

Sometimes you'll want to define a character class that includes a range of values, such as the letters "a through h" or the numbers "1 through 5". To specify a range, simply insert the "-" metacharacter between the first and last character to be matched, such as [1-5] or [a-h]. You can also place different ranges beside each other within the class to further expand the match possibilities. For example, [a-zA-Z] will match any letter of the alphabet: a to z (lowercase) or A to Z (uppercase).

**Unions:**

unions to create a single character class comprised of two or more separate character classes. To create a union, simply nest one class inside the other, such as [0-4[6-8]]. This particular union creates a single character class that matches the numbers 0, 1, 2, 3, 4, 6, 7, and 8.

**Intersections:**

To create a single character class matching only the characters common to all of its nested classes, use &&, as in [0-9&&[345]]. This particular intersection creates a single character class matching only the numbers common to both character classes: 3, 4, and 5.

**Subtraction:**

subtraction to negate one or more nested character classes, such as [0-9&&[^345]]. This example creates a single character class that matches everything from 0 to 9, except the numbers 3, 4, and 5.

**Boundary Matchers**

|  |  |
| --- | --- |
| **Boundary Construct** | **Description** |
| ^ | The beginning of a line |
| $ | The end of a line |

Enter your regex: ^dog$

Enter input string to search: dog

Found

Enter your regex: ^dog$

Enter input string to search: dog

Not Found

Enter your regex: \s\*dog$

Enter input string to search: dog

Found

Enter your regex: ^dog\w\*

Enter input string to search: dogblahblah

Found

## Java Regex Finder

**import** java.io.Console;

**import** java.util.regex.Pattern;

**import** java.util.regex.Matcher;

**public** **class** RegexExample8{

**public** **static** **void** main(String[] args){

        Console console = System.console();

**if** (console == **null**) {

            System.err.println("No console.");

            System.exit(1);

        }

**while** (**true**) {

           Pattern pattern = Pattern.compile(console.readLine("Enter your regex: "));

           Matcher matcher = pattern.matcher(console.readLine("Enter input string to search: "));

**boolean** found = **false**;

**while** (matcher.find()) {

                System.out.println("I found the text "+matcher.group()+" starting at index "+

                 matcher.start()+" and ending at index "+matcher.end());

                found = **true**;

            }

**if**(!found){

                System.out.println("No match found.");

            }

        }

    }

}