Regular Expression in Java also called as Regex.

The regular expression in java defines a pattern for a String. Regular Expression can be used to search, edit or manipulate text. A regular expression is not language specific but they differ slightly for each language. Regular Expression in Java is most similar to Perl. Java Regex classes are present in java.util.regex package that contains three classes:

1. **Pattern**: Pattern object is the compiled version of the regular expression. Pattern class doesn’t have any public constructor and we use it’s public static method compile to create the pattern object by passing regular expression argument.
2. **Matcher**: Matcher is the java regex engine object that matches the input String pattern with the pattern object created. Matcher class doesn’t have any public constructor and we get a Matcher object using pattern object matcher method that takes the input String as argument. We then use matches method that returns boolean result based on input String matches the regex pattern or not.
3. **PatternSyntaxException**: PatternSyntaxException is thrown if the regular expression syntax is not correct.

/\*\*

\* Below is bad regular expression

\*

\* Exception in thread "main" java.util.regex.PatternSyntaxException: Dangling

\* meta character '\*' near index 0 \*

\*

\* pattern = Pattern.compile("\*yy\*");

\*

\* Note: X\* means X occurs zero or more times but it’s not mentioned correctly

\*

\*/

**Refer demo1() method**

Since java regular expression revolves around String, String class has been extended in Java 1.4 to provide a matches method that does regex pattern matching. Internally it uses Pattern and Matcher java regex classes to do the processing but obviously it reduces the code lines. Pattern class also contains matches method that takes regex and input String as argument and return boolean result after matching them. So below code works fine for matching input String with a regular expression in Java.

**Refer demo2() method**

Rule:

*If your requirement is just to check if the input String matches with the pattern, you should save time and lines of code by using simple String matches method.*

*You should use Pattern and Matches classes only when you need to manipulate the input String or you need to reuse the pattern.*

Note that the pattern defined by regex is applied on the String from left to right and once a source character is used in a match, it can’t be reused. For example, regex “121” will match “31212142121” only twice as “\_121\_\_\_\_121”.

**Regular Expression in Java - common matching symbols**

| **Regular Expression** | **Description** | **Example** |
| --- | --- | --- |
| **.** | Matches any single character | (“…”, “a%”) – true(“…”, “.a”) – true (“…”, “a”) – false |
| **^aaa** | Matches aaa regex at the beginning of the line | (“^a.c.”, “abcd”) – true (“^a”, “ac”) – false |
| **aaa$** | Matches regex aaa at the end of the line | (“…cd$”, “abcd”) – true(“a$”, “a”) – true (“a$”, “aca”) – false |
| **[abc]** | Can match any of the letter a, b or c. [] are known as character classes. | (“^[abc]d.”, “ad9”) – true(“[ab].d$”, “bad”) – true (“[ab]x”, “cx”) – false |
| **[abc][12]** | Can match a, b or c followed by 1 or 2 | (“[ab][12].”, “a2#”) – true(“[ab]…[12]”, “acd2”) – true (“[ab][12]”, “c2”) – false |
| **[^abc]** | When ^ is the first character in [], it negates the pattern, matches anything except a, b or c | (“[^ab][^12].”, “c3#”) – true(“[^ab]…[^12]”, “xcd3”) – true (“[^ab][^12]”, “c2”) – false |
| **[a-e1-8]** | Matches ranges between a to e or 1 to 8 | (“[a-e1-3].”, “d#”) – true(“[a-e1-3]”, “2”) – true (“[a-e1-3]”, “f2”) – false |
| **xx** | yy | Matches regex xx or yy |

**Java Regex Metacharacters**

We have some meta characters in Java regex, it’s like shortcodes for common matching patterns.

| **Regular Expression** | Description |
| --- | --- |
| **\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] |
| **\b** | A word boundary |
| **\B** | A non word boundary |

There are two ways to use metacharacters as ordinary characters in regular expressions.

1. Precede the metacharacter with a backslash (\).
2. Keep metacharcter within \Q (which starts the quote) and \E (which ends it).

**Regular Expression in Java – Quantifiers**

Java Regex Quantifiers specify the number of occurrences of a character to match against.

| **Regular Expression** | Description |
| --- | --- |
| **x?** | x occurs once or not at all |
| **X\*** | X occurs zero or more times |
| **X+** | X occurs one or more times |
| **X{n}** | X occurs exactly n times |
| **X{n,}** | X occurs n or more times |
| **X{n,m}** | X occurs at least n times but not more than m times |

Some important methods of Pattern and Matcher classes.

1. We can create a Pattern object with flags. For example Pattern.CASE\_INSENSITIVE enables case insensitive matching.
2. Pattern class also provides split(String) method that is similar to String class split() method.
3. Pattern class toString() method returns the regular expression String from which this pattern was compiled.
4. Matcher classes have start() and end() index methods that show precisely where the match was found in the input string.
5. Matcher class also provides String manipulation methods replaceAll(String replacement) and replaceFirst(String replacement).

**Refer demo3() method**