**Regular Expressions**

* A sequence of characters that define a search pattern
  + This pattern is used to find or match strings
* Applied for validation of forms.

**What are Regular Expressions?**

**A *Regular Expression* is a sequence of characters that constructs a search pattern. When you search for data in a text, we can use this search pattern to describe what we are looking for.**

**^[a-z0-9\_-]{3-15}$**

**^ 🡪 Start of the line**

**[a-z0-9\_-] 🡪 letters, numbers, underscores, hyphens**

**{3-15} 🡪 3 to 15 characters long**

**$ 🡪 End of the line**

**What is Java Regex?**

The Java Regex is an API which is used to define a patten for searching or manipulating *Strings.* It is widely used to define the constraint on Strings such as password and email validation.

* **Matcher Class**

This class is used to perform the match operations on a character sequence.

Comprises of various methods.

|  |  |
| --- | --- |
| boolean matches() | Tests if the given regular expression matches or not |
| boolean find() | Used to find the next expression that matches the pattern |
| boolean find(int start) | Searches the next expression from the given start number |
| string group() | Used to return the matched sequence |
| int start() | Returns the starting index |
| int end() | Returns the ending index |
| int groupcount() | Returns the total number of the matched sequence |

* **Pattern Class**

A compiled version of regular expression which is used to define the pattern of a regex engine.

**Various methods**

|  |  |
| --- | --- |
| Static Pattern compile(String regex) | It compiles the given regex and returns the instance of a pattern |
| Matcher matcher(charSequence input) | Used to create a matcher that matches the given input with the pattern |
| Static boolean matches(String regex) | Used to split the given String around matches of a given pattern |
| String pattern() | Helps to return the regex pattern |
| Int end() | Returns the ending index |

**Character Class**

|  |  |
| --- | --- |
| [abc] | a,b or c [A simple class] |
| [^abc] | Any Class except a, b or c[negation] |
| [a-zA-Z] | **A through Z** or **a through z** |
| [a-d[m-p]] | A through d or m through p |
| [a-z&&[def]] | d,e, or f(Intersection) |
| [a-z&&[^bc]] | A through except b or c(Substraction) |
| [a-z&&[^m-p]] | A through z andnot m through p (Substraction) |

**Regex Quantifiers**

Quantifiers specify the number of occurrences of a character.

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

**Regex Metacharacters**

The regular expression metacharacters work as shortcuts.

|  |  |
| --- | --- |
| . | Can be any character |
| \d | Represents any digit |
| \D | Represents any non digit |
| \s | Represents any white space |
| \S | Non white space character |
| \w | Can be a word character |
| \W | Any non-word character, short for [^\w] |
| \b | Represents a word boundary |
| B | Represents a non word boundary |

Links –

1. [**https://www.javatpoint.com/java-regex**](https://www.javatpoint.com/java-regex)
2. [**https://www.youtube.com/watch?v=f0lZbeueVzU&t=523s**](https://www.youtube.com/watch?v=f0lZbeueVzU&t=523s)

**Email Validator** -[**https://mkyong.com/regular-expressions/how-to-validate-email-address-with-regular-expression/**](https://mkyong.com/regular-expressions/how-to-validate-email-address-with-regular-expression/)