Regular Expressions in Java

Difficulty Level : Medium Last Updated : 09 Feb, 2022

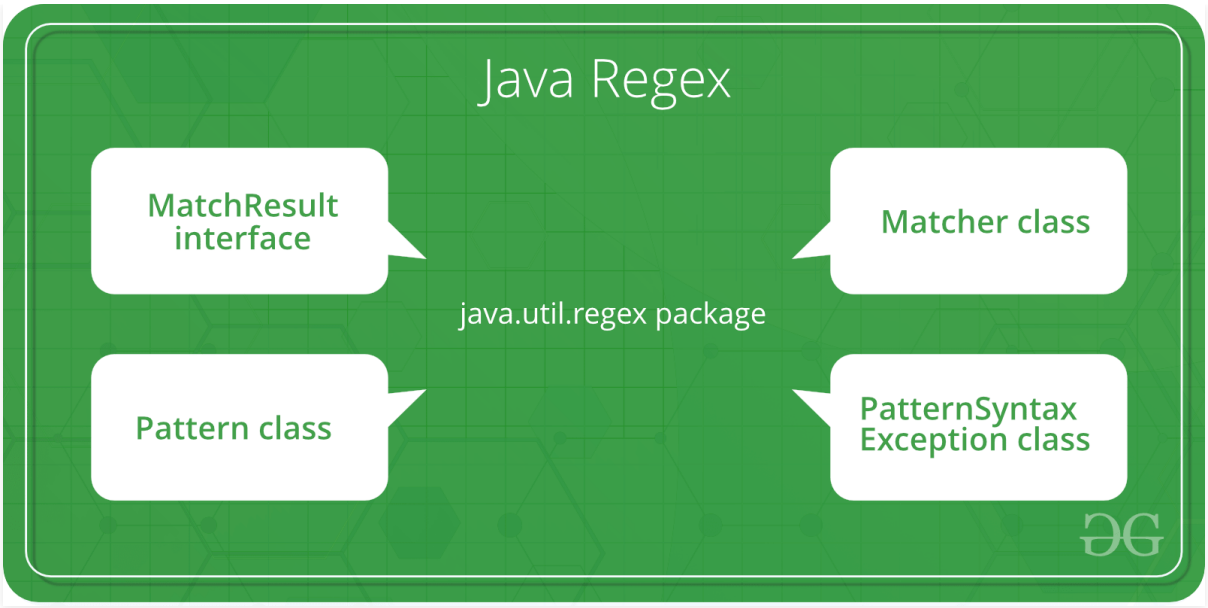
Regular Expressions or Regex (in short) in Java is an API for defining String patterns that can be used for searching, manipulating, and editing a string in Java. Email validation and passwords are a few areas of strings where Regex is widely used to define the constraints. Regular Expressions are provided under **java.util.regex** package. This consists of **3 classes and 1 interface**. The **java.util.regex** package primarily consists of the following three classes as depicted below in tabular format as follows:

S. No.	Class/Interface	Description
1.	Pattern Class	Used for defining patterns
2.	Matcher Class	Used for performing match operations on text using patterns
3.	PatternSyntaxException Class	Used for indicating syntax error in a regular expression pattern
4.	MatchResult Interface	Used for representing the result of a match operation

Regex in java provides us with 3 classes and 1 interface listed below as follows:

1. Pattern Class
2. Matcher Class
3. PatternSyntaxException Class
4. MatchResult Interface

More understanding can be interpreted from the image provided below as follows:



Class 1: Pattern Class

This class is a compilation of regular expressions that can be used to define various types of patterns, providing no public constructors. This can be created by invoking the compile() method which accepts a regular expression as the first argument, thus returns a pattern after execution.

S. No.	Method	Description
1.	compile(String regex)	It is used to compile the given regular expression into a pattern.
2.	compile(String regex, int flags)	It is used to compile the given regular expression into a pattern with the given flags.
3.	flags()	It is used to return this pattern's match flags.
4.	matcher(CharSequence input)	It is used to create a matcher that will match the given input against this pattern.
5.	matches(String regex, CharSequence input)	It is used to compile the given regular expression and attempts to match the given input against it.
6.	pattern()	It is used to return the regular expression from which this pattern was compiled.

S. No.	Method	Description
7.	quote(String s)	It is used to return a literal pattern String for the specified String.
8.	split(CharSequence input)	It is used to split the given input sequence around matches of this pattern.
9.	split(CharSequence input, int limit)	It is used to split the given input sequence around matches of this pattern. The limit parameter controls the number of times the pattern is applied.
10.	toString()	It is used to return the string representation of this pattern.

Example: Pattern class

```
// Java Program Demonstrating Working of matches() Method
// Pattern class

// Importing Pattern class from java.util.regex package
import java.util.regex.Pattern;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Following line prints "true" because the whole
        // text "geeksforgeeks" matches pattern
        // "geeksforge*ks"
        System.out.println(Pattern.matches(
            "geeksforge*ks", "geeksforgeeks"));

        // Following line prints "false" because the whole
        // text "geeksfor" doesn't match pattern "g*geeks*"
        System.out.println(
            Pattern.matches("g*geeks*", "geeksfor"));
    }
}
```

Output

```
true
false
```

Class 2: Matcher class

This object is used to perform match operations for an input string in java, thus interpreting the previously explained patterns. This too defines no public constructors. This can be implemented by invoking a matcher() on any pattern object.

S. No.	Method	Description
1.	find()	It is mainly used for searching multiple occurrences of the regular expressions in the text.
2.	find(int start)	It is used for searching occurrences of the regular expressions in the text starting from the given index.
3.	start()	It is used for getting the start index of a match that is being found using find() method.
4.	end()	It is used for getting the end index of a match that is being found using find() method. It returns the index of the character next to the last matching character.
5.	groupCount()	It is used to find the total number of the matched subsequence.
6.	group()	It is used to find the matched subsequence.
7.	matches()	It is used to test whether the regular expression matches the pattern.

Note: *T Pattern.matches() checks if the whole text matches with a pattern or not. Ot her methods (demonstrated below) are mainly used to find multiple occurrences of patterns in the text.*

Let us do discuss a few sample programs as we did for Pattern class. Here we will be discussing a few java programs that demonstrate the workings of compile(), find(), start(), end(), and split() in order to get a better understanding of the Matcher class.

Example 1: Pattern searching

```
// Java program to demonstrate working of
// String matching in Java

// Importing Matcher and Pattern class
import java.util.regex.Matcher;
import java.util.regex.Pattern;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Create a pattern to be searched
        // Custom pattern
        Pattern pattern = Pattern.compile("geeks");

        // Search above pattern in "geeksforgeeks.org"
        Matcher m = pattern.matcher("geeksforgeeks.org");

        // Finding string using find() method
        while (m.find())

            // Print starting and ending indexes
            // of the pattern in the text
            // using this functionality of this class
            System.out.println("Pattern found from "
                               + m.start() + " to "
                               + (m.end() - 1));
    }
}
```

Output

```
Pattern found from 0 to 4
Pattern found from 8 to 12
```

Example 2: Simple regular expression searching

```
// Java program to demonstrate working of
// String matching in Java

// Importing Matcher and Pattern class
// from java.util package
import java.util.regex.Matcher;
import java.util.regex.Pattern;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Creating a pattern to be searched
        // Custom pattern to be searched
        Pattern pattern = Pattern.compile("ge*");

        // Searching for the above pattern in
        // "geeksforgeeks.org"
        Matcher m = pattern.matcher("geeksforgeeks.org");

        // Checking whether the pattern is there or not
        // using find() method
        while (m.find())

            // Print starting and ending indexes of the
            // pattern in text using method functionality of
            // this class
            System.out.println("Pattern found from "
                               + m.start() + " to "
                               + (m.end() - 1));
    }
}
```

Output

```
Pattern found from 0 to 2
Pattern found from 8 to 10
Pattern found from 16 to 16
```

Example 3: Case Insensitive pattern searching

```
// Java Program Demonstrating Working of String matching

// Importing Matcher class and Pattern classes
// from java.util.regex package
import java.util.regex.Matcher;
import java.util.regex.Pattern;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Creating a pattern to be searched
        Pattern pattern = Pattern.compile(
            "ge*", Pattern.CASE_INSENSITIVE);

        // Searching above pattern in "geeksforgeeks.org"
        Matcher m = pattern.matcher("GeeksforGeeks.org");

        // Find th above string using find() method
        while (m.find())

            // Printing the starting and ending indexes of
            // the pattern in text using class method
            // functionalities
            System.out.println("Pattern found from "
                               + m.start() + " to "
                               + (m.end() - 1));
    }
}
```

Output

```
Pattern found from 0 to 2
Pattern found from 8 to 10
Pattern found from 16 to 16
```

Example 4: split() method to split a text based on a delimiter pattern.

The string split() method breaks a given string around matches of the given regular expression. There exist two variations of this method so do go through it prior to moving onto the implementation of this method.

Illustration:

```
Input  --> String: 016-78967
Output -->  Regex: {"016", "78967"}
```

```
// Java program Illustrating Working of split() Method
// by Splitting a text by a given pattern

// Importing Matcher and Pattern classes from
// java.util.regex package
import java.util.regex.Matcher;
import java.util.regex.Pattern;

// Main class
class GFG {

    // Main driver method
    public static void main(String args[])
    {

        // Custom string
        String text = "geeks1for2geeks3";

        // Specifies the string pattern
        // which is to be searched
        String delimiter = "\\d";
        Pattern pattern = Pattern.compile(
            delimiter, Pattern.CASE_INSENSITIVE);

        // Used to perform case insensitive search of the
        // string
        String[] result = pattern.split(text);

        // Iterating using for each loop
        for (String temp : result)
```

```
            System.out.println(temp);
        }
    }
```

Output

```
geeks
for
geeks
```

Now we are done with discussing both the classes. Now we will be introducing you to two new concepts that make absolute clear Also there is an exception class associated with

Class 3: PatternSyntaxException Class

This is an object of Regex which is used to indicate a syntax error in a regular expression pattern and is an unchecked exception. Following are the methods been there up in the PatternSyntaxException class as provided below in tabular format as follows.

S. No.	Method	Description
1.	getDescription()	It is used to retrieve the description of the error.
2.	getIndex()	It is used to retrieve the error index.
3.	getMessage()	It is used to return a multi-line string containing the description of the syntax error and its index, the erroneous regular-expression pattern, and a visual indication of the error index within the pattern.
4.	getPattern()	It is used to retrieve the erroneous regular-expression pattern.

Interface 1: MatchResult Interface

This interface is used to determine the result of a match operation for a regular expression. It must be noted that although the match boundaries, groups, and group boundaries can be seen, the modification is not allowed through a MatchResult. Following

are the methods been there up here in this interface as provided below in tabular format as follows:

S. No.	Method	Description
1.	end()	It is used to return the offset after the last character is matched.
2.	end(int group)	It is used to return the offset after the last character of the subsequence captured by the given group during this match.
3.	group()	It is used to return the input subsequence matched by the previous match.
4.	group(int group)	It is used to return the input subsequence captured by the given group during the previous match operation.
5.	groupCount()	It is used to return the number of capturing groups in this match result's pattern.
6.	start()	It is used to return the start index of the match.
7.	start(int group)	It is used to return the start index of the subsequence captured by the given group during this match.

Lastly, let us do discuss some of the important observations as retrieved from the above article

1. We create a pattern object by calling Pattern.compile(), there is no constructor. compile() is a static method in Pattern class.
2. Like above, we create a Matcher object using matcher() on objects of Pattern class.
3. Pattern.matches() is also a static method that is used to check if given text as a whole matches pattern or not.
4. find() is used to find multiple occurrences of patterns in the text.
5. We can split a text based on a delimiter pattern using the split() method

This article is contributed by **Akash Ojha**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.orgerror-index. See your article appearing on the GeeksforGeeks

main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.