The java.util.regex.Pattern class, or the Pattern class for short, is the main access point of the Java regular expression API. Whenever you need to work with regular expressions, you start with the Pattern class.

Here is a list of the Pattern methods covered in this text:

* [**Pattern.matches()**](http://tutorials.jenkov.com/java-regex/pattern.html#0)
* [**Pattern.compile()**](http://tutorials.jenkov.com/java-regex/pattern.html#1)
* [**Pattern.matcher()**](http://tutorials.jenkov.com/java-regex/pattern.html#2)
* [**Pattern.split()**](http://tutorials.jenkov.com/java-regex/pattern.html#3)
* [**Pattern.pattern()**](http://tutorials.jenkov.com/java-regex/pattern.html#4)

**Pattern.matches()**

The fastest way to check if a regular expression pattern matches a text is to use the static Pattern.matches() method. Here is an example:

String text =

"This is the text to be searched " +

"for occurrences of the pattern.";

String pattern = ".\*is.\*";

boolean matches = Pattern.matches(pattern, text);

System.out.println("matches = " + matches);

This example searches the string in the text variables for an occurrence of the word "is", allowing zero or more characters to be present before and after the word (the two .\* parts of the pattern).

The Pattern.matches() method is fine if you just need to check a pattern against a text a single time, and the default settings of the Pattern class are appropriate.

If you need to match for multiple occurrences, and even access the various matches, or just need non-default settings, you need to compile a Pattern instance using the Pattern.compile() method.

**Pattern.compile()**

If you need to match a text against a regular expression pattern more than one time, you need to create a Pattern instance using the Pattern.compile() method. Here is an example of how to do that:

String text =

"This is the text to be searched " +

"for occurrences of the http:// pattern.";

String patternString = ".\*http://.\*";

Pattern pattern = Pattern.compile(patternString);

You can also compile a Pattern using special flags. Here is how to compile a Pattern with flags:

Pattern pattern = Pattern.compile(patternString, Pattern.CASE\_INSENSITIVE);

The Pattern class contains a list of flags (int constants) that you can use to make the Pattern matching behave in certain ways. The flag used above makes the pattern matching ignore the case of the text when matching.

**Pattern.matcher()**

Once you have obtained a Pattern instance, you can use that to obtain a Matcher instance. The Matcher instance is used to find matches of the pattern in texts. Here is how to create a Matcher instance from a Pattern instance:

Matcher matcher = pattern.matcher(text);

The Matcher class has a matches() method that tests whether the pattern matches the text. Here is a full example of how to use the Matcher:

String text =

"This is the text to be searched " +

"for occurrences of the http:// pattern.";

String patternString = ".\*http://.\*";

Pattern pattern = Pattern.compile(patternString, Pattern.CASE\_INSENSITIVE);

**Matcher matcher = pattern.matcher(text);**

boolean matches = matcher.matches();

System.out.println("matches = " + matches);

The Matcher will be covered in more detail in the text on [**Java regex Matcher**](http://tutorials.jenkov.com/java-regex/matcher.html).

**Pattern.split()**

The split() method in the Pattern class can split a text into an array of String's, using the regular expression (the pattern) as delimiter. Here is an example:

String text = "A sep Text sep With sep Many sep Separators";

String patternString = "sep";

Pattern pattern = Pattern.compile(patternString);

String[] split = pattern.split(text);

System.out.println("split.length = " + split.length);

for(String element : split){

System.out.println("element = " + element);

}

This example will split the text in the text variable into 5 separate strings. Each of these strings are included in the returned String array.

**Pattern.pattern()**

The pattern() method of the Pattern class simply returns the pattern that the Pattern instance was compiled from. Here is an example:

String patternString = "sep";

Pattern pattern = Pattern.compile(patternString);

String pattern2 = pattern.pattern();

In this example the pattern2 variable will contain the value sep, which was the value the Pattern instance was compiled from.