

Java Regular Expressions

[< Previous](#)[Next >](#)

What is a Regular Expression?

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Java does not have a built-in Regular Expression class, but we can import the `java.util.regex` package to work with regular expressions. The package includes the following classes:

- `Pattern` Class - Defines a pattern (to be used in a search)
- `Matcher` Class - Used to search for the pattern
- `PatternSyntaxException` Class - Indicates syntax error in a regular expression pattern

Example

Find out if there are any occurrences of the word "w3schools" in a sentence:

☐ Dark mode



```
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {
        Pattern pattern = Pattern.compile("w3schools", Pattern.CASE_I
        Matcher matcher = pattern.matcher("Visit W3Schools!");
        boolean matchFound = matcher.find();
        if(matchFound) {
            System.out.println("Match found");
        } else {
            System.out.println("Match not found");
        }
    }
}
// Outputs Match found
```

Try it Yourself »

Example Explained

In this example, The word "w3schools" is being searched for in a sentence.

First, the pattern is created using the `Pattern.compile()` method. The first parameter indicates which pattern is being searched for and the second parameter has a flag to indicates that the search should be case-insensitive. The second parameter is optional.

The `matcher()` method is used to search for the pattern in a string. It returns a `Matcher` object which contains information about the search that was performed.

The `find()` method returns true if the pattern was found in the string and false if it was not found.

☐ Dark mode



Flags

Flags in the `compile()` method change how the search is performed.

Here are a few of them:

- `Pattern.CASE_INSENSITIVE` - The case of letters will be ignored when performing a search.
- `Pattern.LITERAL` - Special characters in the pattern will not have any special meaning and will be treated as ordinary characters when performing a search.
- `Pattern.UNICODE_CASE` - Use it together with the `CASE_INSENSITIVE` flag to also ignore the case of letters outside of the English alphabet

Regular Expression Patterns

The first parameter of the `Pattern.compile()` method is the pattern. It describes what is being searched for.

Brackets are used to find a range of characters:

Expression	Description
<code>[abc]</code>	Find one character from the options between the brackets
<code>[^abc]</code>	Find one character NOT between the brackets
<code>[0-9]</code>	Find one character from the range 0 to 9

Metacharacters

Metacharacters are characters with a special meaning:

Metacharacter	Description

☐ Dark mode



.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
\$	Finds a match at the end of the string as in: World\$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers

Quantifiers define quantities:

Quantifier	Description
n^+	Matches any string that contains at least one n
n^*	Matches any string that contains zero or more occurrences of n
$n?$	Matches any string that contains zero or one occurrences of n
$n\{x\}$	Matches any string that contains a sequence of X n 's
$n\{x,y\}$	Matches any string that contains a sequence of X to Y n 's
$n\{x,\}$	Matches any string that contains a sequence of at least X n 's

☐ Dark mode