March 8, 2017

# Java Regular Expressions Cheat Sheet (Regex Java)

DEVELOPER PRODUCTIVITY | JAVA APPLICATION DEVELOPMENT

In this article we explore Java regular expressions, including how they're used, best practices, and shortcuts to help you use them. Then, at the end of the article, we provide a Java RegEx cheat sheet PDF that gives you all RegEx shortcuts on one page. But first, let's start with the basics.

# What Is a Java Regular Expression?

> A Java regular expression, or Java regex, is a sequence of characters that specifies a pattern which can be searched for in a text. A regex defines a set of strings, usually united for a given purpose.

Suppose you need a way to formalize and refer to all the strings that make up the format of an email address. Since there are a near infinite number of possible email addresses, it'd be hard to enumerate them all.

However, as we know an email address has a specific structure, and we can encode that using the regex syntax. A Java regex processor translates a regular expression into an internal representation which can be executed and matched against the text being searched. It will tell you whether a string is in the set of strings defined by a pattern or find a substring that belongs in that set.

# Useful Java Classes & Methods

Most languages have a regular expressions implementation either baked in or provided by a library. Java is no exception. Below are the classes you have to know in order to be effective using Java Regex.

## RegEx Pattern Methods

Pattern (https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html) is a compiled representation of a regular expression in Java. Below is the list of the most frequently used methods in the Pattern class API.

**Java RegEx Pattern Methods**

| RegEx Pattern Method | Description |
| --- | --- |
| Pattern compile(String regex) | Compiles the given regular expression into a pattern. |

| | |
|---|---|
| Pattern compile(String regex, int flags) | Compiles the given regular expression into a pattern with the given flags. |
| boolean matches(String regex) | Returns whether or not this string matches the given regular expression. |
| String[] split(CharSequence input) | Splits the given input sequence around matches of this pattern. |
| String quote(String s) | Returns a literal pattern String for the specified String s. |
| Predicate asPredicate() | Creates a predicate which can be used to match a string. |

## RegEx Matcher Methods for Java Pattern Matching

A matcher (https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html) is the engine that performs Java pattern matching operations on a character sequence by interpreting a Pattern. Below is the list of the most frequently used methods in the Matcher class API:

**Java RegEx Matcher Methods**

| RegEx Matcher Method | Description |
|---|---|
| boolean matches() | Attempts to match the entire region against the pattern. |
| boolean find() | Attempts to find the next subsequence of the input that matches the pattern. |
| int start() | Returns the start index of the last match. |

By compiling a pattern and obtaining a matcher for it, you can match many texts for the pattern efficiently. So if you expect to process lots of texts, compile a matcher, cache it and use it repeatedly.

# Java RegEx Syntax

Let's move on now to the syntax for Java RegEx. The `Pattern.compile` method takes a String, which is the RegEx that defines a set of matching strings. Naturally, it has to have a tricky syntax, otherwise a single string defining the pattern can only represent itself. A regular character in the RegEx Java syntax matches that character in the text. If you'll

create a Pattern with `Pattern.compile("a")` it will only match only the String "a". There is also an escape character, which is the backslash "\". It is used to distinguish when the pattern contains an instruction in the syntax or a character.

### Java RegEx Escape Example

Let's look at an example as to why we need an escape character. Imagine "[" has a special meaning in the regular expression syntax (it has). How can you determine if "[" is a command to the matching engine or a pattern containing only the bracket? You cannot, so to specify the characters that are also the commands in the syntax you need to escape them. It means "\[" is a pattern for the string "[", and "[" is part of a command. What about trying to match a backslash? You need to escape it too, so be prepared to see something like "\\\\" in the RegEx code.

## Character Classes in Java Regular Expressions

On top of specifying the expressions that contain individual characters only, you can define the whole classes of characters. Think of them as sets, if a character in some text belongs to the character class, it is matched. Here is a table with the most used character classes in Java RegEx.

**Java RegEx Character Classes**

| Character Class | Description |
|---|---|
| [abc] | simple, matches a or b, or c |
| [\^abc] | negation, matches everything except a, b, or c |
| [a-c] | range, matches a or b, or c |
| [a-c[f-h]] | union, matches a, b, c, f, g, h |
| [a-c&&[b-c]] | intersection, matches b or c |
| [a-c&&[\^b-c]] | subtraction, matches only a |

## Predefined Character Classes in Java RegEx

For your convenience, there are some useful classes defined already. For example, digits are a perfect example of a useful character class. For example a 5 digit number could be coded into a pattern as "[0-9][0-9][0-9][0-9][0-9]", but it's quite ugly. So there's a shorthand for that: "\d". Here are the other classes you need to know, starting with the regex for any character:

**Predefined Java RegEx Character Classes**

| Character Class | Description |
|---|---|
| . | Any character |
| \d | A digit: [0-9] |
| \D | A non-digit: [\^0-9] |
| \s | A whitespace character: [ \t\n\x0B\f\r] |
| \S | A non-whitespace character: [\^\s] |
| \w | A word character: [a-zA-Z_0-9] |
| \W | A non-word character: [\^\w] |

Note that the letter specifying a predefined character class is typically lowercase, the uppercase version tends to mean the negation of the class. Also, note that a dot "." is a character class, which contains all the characters. Particularly useful, but remember to escape it when you need to match the actual dot character.

# Java RegEx Boundary Matchers

Next, there's syntax to specify the position of the matched sequence in the original text you're searching. If you only need to filter out the strings that start with an email address or something, this is extremely useful.

**Java RegEx Boundary Matchers**

| Boundary Matcher | strong>Description |
|---|---|
| ^ | The beginning of a line |
| $ | The end of a line |
| \b | A word boundary |
| \B | A non-word boundary |

| | |
|---|---|
| \A | The beginning of the input |
| \G | The end of the previous match |
| \Z | The end of the input but for the final terminator, if any. |
| \z | The end of the input |

A noteworthy combination of the boundary matchers is the "^pattern$" which will only match the text if it is the full pattern.

# Java RegEx Logical Operations

Now we're getting into more advanced territory. If a pattern is more than a single character long, it will match a longer string too. In general "XY" in the RegEx Java syntax matches X followed by Y. However, there's also an OR operation, denoted by the post "|". The "X|Y" RegEx means it is either X or Y. This is a very powerful feature; you can combine the character classes or sequences of characters (include them in brackets).

# RegEx Java Quantifiers

On top of everything, you can say how many times the sequence of characters can be repeated for the match. The RegEx "1" only matches the input "1", but if we need to match a string of any length consisting of the character "1" you need to use one of the following quantifiers.

**Java RegEx Quantifiers**

| Quantifier | Description |
|---|---|
| * | Matches zero or more occurrences. |
| + | Matches one or more occurrences. |
| ? | Matches zero or one occurrence. |

# Regular Expression Groups and Backreferences

A group is a captured subsequence of characters which may be used later in the expression with a backreference. We've mentioned already that if you enclose a group of characters in parentheses, you can apply quantifiers or logical or to the whole group. What is even more awesome is that you can refer to the actual characters in the text matched by the group, later. Here's how you do it:

**Java RegEx Groups and Backreferences**

| Groups and Backreferences | Description |
|---|---|
| (...) | Matches zero or more occurrences. |
| \N | Matches one or more occurrences. |
| (\d\d) | Matches zero or one occurrence. |
| (\d\d)/\1 | Two digits repeated twice, \1 - refers to the matched group |

# Regular Expressions Pattern Flags

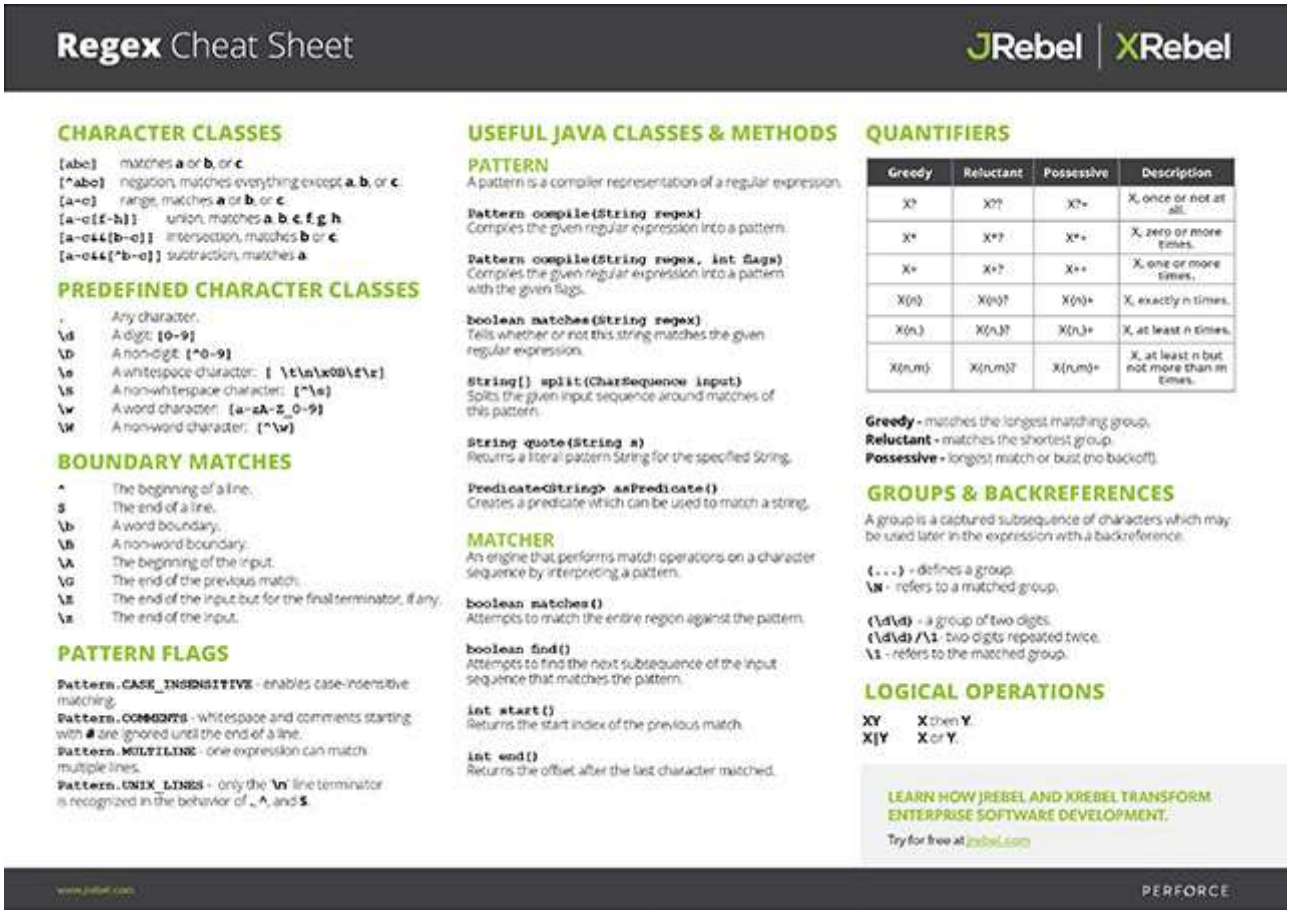
Remember when we talked about the useful API for the regular expressions in Java, there was a method to compile a pattern that took the flags. These will control how the pattern behaves. Here are some flags that can be useful here and there.

**Java RegEx Pattern Methods**

| Pattern Flags | Description |
|---|---|
| Pattern.CASE_INSENSITIVE | Enables case-insensitive matching. |
| Pattern.COMMENTS | Whitespace and comments starting with # are ignored until the end of a line. |
| Pattern.MULTILINE | One expression can match multiple lines. |
| Pattern.DOTALL | The expression "." matches any character, including a line terminator. |
| Pattern.UNIX_LINES | Only the '\n' line terminator is recognized in the behavior of ., ^, and $. |

# Download the Regular Expressions Cheat Sheet PDF

In our Regular Expressions Cheat Sheet, we include essential Java classes and methods, RegEx syntax, character classes, boundary matchers, logical operations, quantifiers, groups, backreferences, and pattern flags. You can download the Java RegEx Cheat Sheet, below.




(https://www.jrebel.com/system/files/regular-expressions-cheat-sheet.pdf)

**DOWNLOAD THE CHEAT SHEET (HTTPS://WWW.JREBEL.COM/SYSTEM/FILES/REGULAR-EXPRESSIONS-CHEAT-SHEET.PDF)**

# Additional Resources

SEND FEEDBACK

Last year we explored some of the topics that are universally used in software development and have quite a library of useful Java cheat sheets (https://www.jrebel.com/resources/java-resources) to please your sight and remind of the commands and options developers often forget and google:

1. Java 8 Best Practices (/blog/java-8-cheat-sheet)
2. Java 8 Streams (/resources/java-8-streams-cheat-sheet)
3. Git Commands (/blog/git-cheat-sheet)
4. Docker Commands (/blog/docker-commands-cheat-sheet)
5. Java Collections (/blog/java-collections-cheat-sheet)
6. SQL (/blog/sql-cheat-sheet)
7. JUnit (/resources/junit-cheat-sheet)
8. Spring Framework Annotations (/resources/spring-framework-annotations-cheat-sheet)
9. Java Generics (/resources/java-generics-cheat-sheet)
10. JVM Options (/resources/jvm-options-cheat-sheet)

## Save Development Time With JRebel

Regular expressions in Java make the coding process faster and less tedious for developers. JRebel provides an additional layer of efficiency by eliminating the costly downtime associated with code updates. Depending on your Java application framework, you can save hours off every coding day.

**TRY JREBEL FREE (/PRODUCTS/FREE-TRIAL)**

Papers & Videos (/Resources/Papers-And-Videos)
Events & Webinars (/Resources/Events)
Recorded Webinars (/Resources/Recorded-Webinars)
Blog (/Blog)
Video Tutorials (/Support/Video-Tutorials)
ROI Calculator (/Calculate-Your-Roi-With-Jrebel)

## INTEGRATIONS

Eclipse (/Jrebel-And-Xrebel-Eclipse-Plugins)
IntelliJ (/Jrebel-And-Xrebel-Intellij-Idea-Plugins)

## SUPPORT

JRebel Documentation (/Products/Jrebel/Learn)
XRebel Documentation (/Products/Xrebel/Learn)
FAQs (/Jrebel/Learn/Faq)

## DOWNLOADS

JRebel Download (/Products/Jrebel/Download)
XRebel Download (/Products/Xrebel/Download)
Rebel Licenses On-Premise Download (/Products/Licenses/On-Premise)
Eclipse Plugin (/Jrebel-And-Xrebel-Eclipse-Plugins)
IntelliJ Plugin (/Jrebel-And-Xrebel-Intellij-Idea-Plugins)

## HUBS

New Features In Java (/Resources/New-Features-Java)
Exploring Java Microservices (/Resources/Exploring-Java-Microservices)
Java Resources For Developers (/Resources/Java-Resources)
Java Technology Overview (/Resources/Java-Technology-Overview)

JRebel by Perforce (https://www.perforce.com)

Terms of Use (/legal/terms-of-use) | Privacy Policy (/legal/privacy-policy) | Data Processing Policy (/legal/data-processing-policy) | Sitemap (/sitemap)

SEND FEEDBACK