Java Regular Expressions

<https://docs.oracle.com/javase/tutorial/essential/regex/>

Java Regex Tutorial

<http://www.vogella.com/tutorials/JavaRegularExpressions/article.html>

Guide to Regular Expressions in Java (Part 1)

<http://ocpsoft.org/opensource/guide-to-regular-expressions-in-java-part-1/>

Java Regular Expression Tutorial with Examples

<http://www.journaldev.com/634/java-regular-expression-tutorial-with-examples>

## Further Reading

[[](http://www.regular-expressions.info/javabook.html)](http://www.regular-expressions.info/javabook.html)If you'd like a more detailed overview of all the functionality offered by the java.util.regex package, you may want to get yourself a copy of "Java Regular Expressions" written by Mehran Habibi and published by Apress. While the [regular expressions tutorial](http://www.regular-expressions.info/tutorial.html) on this website is more complete and more detailed, Mr. Habibi's book contains a more slow-paced regular expressions tutorial. The book definitely provides more information and examples on the java.util.regex package than this single web page can provide.

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** JavaRegularExpressions {

**public** **static** **void** main( String args[] ){

String text =

"This is the text to be searched " +

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

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

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

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

}

}

".\*http://.\*";

searches for occurrences of the string http://

## Matcher class

It implements **MatchResult** interface. It is a *regex engine* i.e. used to perform match operations on a character sequence.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | boolean matches() | test whether the regular expression matches the pattern. |
| 2 | boolean find() | finds the next expression that matches the pattern. |
| 3 | boolean find(int start) | finds the next expression that matches the pattern from the given start number. |

## Pattern class

It is the *compiled version of a regular expression*. It is used to define a pattern for the regex engine.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | static Pattern compile(String regex) | compiles the given regex and return the instance of pattern. |
| 2 | Matcher matcher(CharSequence input) | creates a matcher that matches the given input with pattern. |
| 3 | static boolean matches(String regex, CharSequence input) | It works as the combination of compile and matcher methods. It compiles the regular expression and matches the given input with the pattern. |
| 4 | String[] split(CharSequence input) | splits the given input string around matches of given pattern. |
| 5 | String pattern() |  |

## Regular Expression . Example

The . (dot) represents a single character.

1. import java.util.regex.\*;
2. class RegexExample2{
3. public static void main(String args[]){
4. System.out.println(Pattern.matches(".s", "as"));//true (2nd char is s)
5. System.out.println(Pattern.matches(".s", "mk"));//false (2nd char is not s)
6. System.out.println(Pattern.matches(".s", "mst"));//false (has more than 2 char)
7. System.out.println(Pattern.matches(".s", "amms"));//false (has more than 2 char)
8. System.out.println(Pattern.matches("..s", "mas"));//true (3rd char is s)
9. }}

More Java Regular Expressions

<http://www.javatpoint.com/java-regex>

**java Cohesion & Coupling**

Benefits of Higher Cohesion:

* Highly cohesive classes are much easier to maintain and less frequently changed.
* Such classes are more usable than others as they are designed with a well-focused purpose.

**Cohesion means that the whole of a class sticks together (well, roughly). A class should be responsible for itself, should do one thing and as far as possible do everything for that one thing. A Car class should remember its make, color, speed. It is responsible for changing speed; the speedUp() and slowDown() methods should be in the Car class; no other class should make your Car go faster or slower.**

**Coupling means that one class gets at the implementation of another class.**

**Driver papa = new Driver();**

**Car ford = new Car("Ford", "red");**

**. . . .**

**public class Driver**

**{**

**Car myCar;**

**. . .**

**public void goFaster(int speed)**

**{**

**myCar.speed += speed;**

**}**

**. . .**

**}**

**The Car class has allowed access to its speed field and the Driver class changes its value directly. This means other classes gain access to the implementation of the Car class; any changes to that implementation will "break" the Driver class. This is "tight coupling" and tight coupling is A Bad Thing, because any changes to one class can mean that other classes would have to be altered too.**

**To avoid tight coupling**

**•All classes should have as small a public interface as possible.**

**•All non-constant fields should have private access.**

**•Any alterations to the values of fields should be via method calls.**

**coupling = is the how much your class knows about other class.**

**Let us say that you have one friend and you know a lot about this friend as: What is your friend’s bank balance?**

**This you should not know.**

**The same happens with the class, when one knows a lot about other.**

**cohesion = is when you have one class focus, one objective.**

**for example:**

**class AddCustomers{}**

**This is one class that has high cohesion… one focus**

**one class with low cohesion is when it does several things for example:**

**- add customers**

**- print list of the customers**

**- connect to database**

**all in one unique class this is one class with low cohesion.**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**What is the difference between cohesion and coupling in java?**

**cohesion literally means that how much sticky/strong/purposeful an object is**

**coupling means how much two objects are dependent on each other.**

**For a better Java App design,**

**the classes should be highly cohesive and loosely coupled.**

**If two classes are highly coupled then if one is changed, other also needs a change. To avoid high coupling, many design patterns have been written like Business Delegate, Session Facade etc.**

**If the purpose of a class is clear and is handling only one purpose then that class is said to be highly cohesive.**

**e.g. the ArrayList class implements the operations of ArrayList only and does not operations for Map or Set. ArrayList class is highly cohesive in that sense.**

**coupling and cohesion, have to do with the quality of an OO design. In general, good OO design calls for loose coupling and shuns tight coupling, and good OO design calls for high cohesion, and shuns low cohesion. As with most OO design discussions, the goals for an application are**

**• Ease of creation**

**• Ease of maintenance**

**• Ease of enhancement**

example of low cohesion:

class Calculator

{

public static void main(String args[])

{

//calculating sum here

result = a + b;

//calculating difference here

result = a - b;

//same for multiplication and division

}

}

example of high cohesion:

class Calculator

{

public static void main(String args[])

{

Calculator myObj = new Calculator();

System.out.println(myObj.SumOfTwoNumbers(5,7));

}

public int SumOfTwoNumbers(int a, int b)

{

return (a+b);

}

//similarly for other operations

}

**API**

[**http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm**](http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing1.htm)