Cairo University
Faculty of Computers & Artificial Intelligence
Theory of Computations

Lab#3

Regex In Java

Uses of Regular Expressions (Pattern Matching)

• Validation/ Verification (e.g. validating input, validating that a string contains a valid email address, etc).
• Searching (e.g. looking for patterns, searching logs, extracting information, etc.).

Symbols

Regular Expressions Basics

- **Expressing Choice**
  - Alteration (OR operator : '|')

a|b	Matches either a or b

  - Grouping

a(b|c)	Matches either ab or ac

  - Character Classes

[abc]	Simple class	a, b, or c
[^abc]	Negation	Any character except a, b, or c
[a-zA-Z]	Range	a through z or A through Z, inclusive
[a-d[m-p]]	Union	a through d, or m through p: [a-dm-p]
[a-z&&[def]]	Intersection	d, e, or f
[a-z&&[^bc]]	Subtraction	a through z, except for b and c: [ad-z]
[a-z&&[^m-p]]	Subtraction	a through z, and not m through p: [a-lq-z]

  - Character Classes (predefined)

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

- **Arbitrary Characters:** The dot '.' matches any character except for '\n'.

".*?"	strings surrounded by double-quotation marks

- **Repetitions**
  - Quantifiers

?	Zero or One
*	Zero or More
+	One or More

  - Expressing Limits

{n}	Exactly n items
{n,m}	Min n & Max m
{n,}	Minimum n items

- **Special Characters**
  - '[', '{', '(', ')', '?', '+', '*', '.', '^', '$', '|', '\' are all special characters
  - Backslash can be used for escape:
    "\[", "\{", "\(", "\)", "\?", "\+", "*", "\.", "\^", "\$", "\|", "\\"
  - Also used for ASCII special characters:
    "\n", "\r", ...

Boundaries & Anchors

^	The beginning of a line
$	The end of a line

Examples:

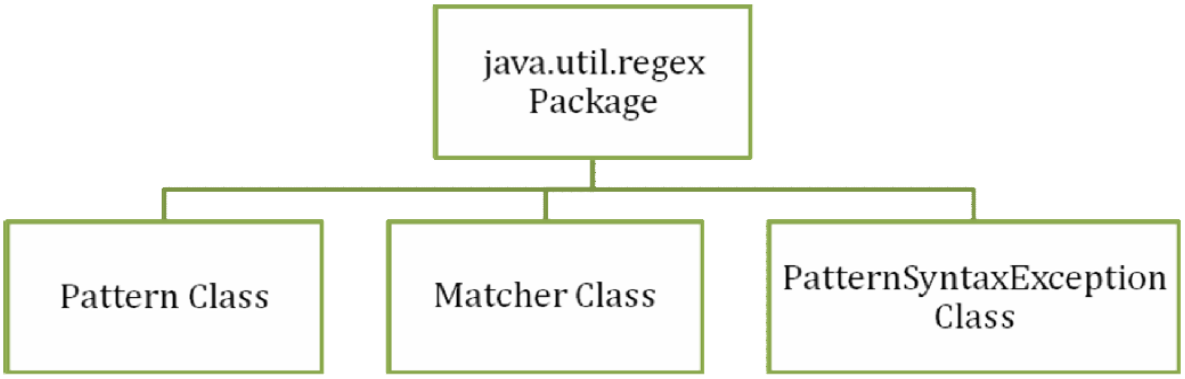
1. Develop a regular expression that can match any number of a's and ends with one b.
2. Develop a regular expression that can match any floating point number
3. Sentence starts with "The" and ends with "Spain" ["The rain in Spain"]
4. HTML headers <H 1 >, …., <H 6 >

Solutions

1. a*b
2. [-+]?[0-9]+\.[0-9]+
3. "^The.*Spain$"
4. <H[123456]> ,OR <H[1-6]> OR <[Hh][1-6]>

The Java RegEx API

Java RegEx API consists of 3 classes as follows:



Steps to use the regex package:

1. Write your regex as a string.
2. Compile your regex string and get a reference to a Pattern object using the Pattern class.
3. The Pattern object is retrieved via the Pattern class's static compile method (i.e. you cannot instantiate a Pattern object using new. Once you have a Pattern object you can use it to get a reference to a Matcher object.
    - The Pattern class's *matcher()* method to obtain a Matcher object.
4. The Matcher object is where the resulting matches are held.
    - Use the Matcher class's *find()* method to get at any matches. This method will parse just enough of our target string to make a match, at which point it will return true.
    - You can retrieve the entire matched substring using the Matcher class's *group()* method.
    - You can use the Matcher's *start()* and *end()* methods to find out where the matched substrings occurred in the target string.

Problem:

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

// Compile the regex.
     String regex = "a*b";
     Pattern pattern = Pattern.compile(regex);

// Create the 'target' string we wish to interrogate
String targetString = "aaaaabaacaab";

// Get a Matcher based on the target string
     Matcher matcher = pattern.matcher(targetString);

// Find all the matches
     while (matcher.find())
     {
        System.out.println("Found a match: " +
     matcher.group());
        System.out.println("Start position: " +
     matcher.start());
        System.out.println("End position: " +
     matcher.end());
     }
```

Output will be:

```
Found a match: aaaaab
Start position: 0
End position: 6
Found a match: aab
Start position: 9
End position: 12
```

More on Pattern Class:

You can use the following flag as a second argument to its compile() method.
- Pattern.CASE_INSENSITIVE: to tell the regex engine to match ASCII characters regardless of case.

In the last example you can edit these 2 lines:

```
Pattern pattern = Pattern.compile(regex,Pattern.CASE_INSENSITIVE);
String targetString = "aaaaabaacaabAAAAAb";
```

Here it will match with **AAAAAb** as it will ignore the case sensitive.

Output will be:

```
Found a match: aaaaab
Start position: 0
End position: 6
Found a match: aab
Start position: 9
End position: 12
Found a match: AAAAAb
Start position: 12
End position: 18
```