A Quick Introduction to Regular Expressions in Java

Lecture 5a

1

Readings

- SUN regexps tutorial

http://java.sun.com/docs/books/tutorial/extra/regex/index.html

- Java.util.regex API

http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html

2

Regular Expressions

- Regular expressions (regex's) are sets of symbols and syntactic elements used to match patterns of text.

3

Basic Syntax

Char	Usage	Example
.	Matches any single character	.at = cat, bat, rat, 1at…
*	Matches zero or more occurrences of the single preceding character	.*at = everything that ends with at 0*123 = 123, 0123, 00123…
[...] [^...]	Matches any single character of the ones contained Matches any single character except for the ones contained	[cbr]at = cat, bat, rat. [^bc]at = rat, sat…, *but not* bat, cat. <[^>]*> = <…anything…>
^	Beginning of line	^a = line starts with a
$	End of line	^$ = blank line (starts with the end of line)
\	Escapes following special character: . \ / & [ ] * + -> \. \\ \/ \& \[\] * \+	[cbr]at\. = matches cat., bat. and rat. only
...	…	

4

Matches

- Input string consumed from left to right
- Match ranges: inclusive of the beginning index and exclusive of the end index
- Example:

Current REGEX is: foo
Current INPUT is: foofoofoo
I found the text "foo" starting at index 0 and ending at index 3.
I found the text "foo" starting at index 3 and ending at index 6.
I found the text "foo" starting at index 6 and ending at index 9.

5

Character Classes

[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z, or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z] (subtraction)

6

Predefined Character Classes

.	Any character (may or may not match line terminators)
\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]

7

Quantifiers

Greedy	Reluctant	Possessive	Meaning
X?	X??	X?+	X, once or not at all
X*	X*?	X*+	X, zero or more times
X+	X+?	X++	X, one or more times
X{n}	X{n}?	X{n}+	X, exactly n times
X{n,}	X{n,}?	X{n,}+	X, at least n times
X{n,m}	X{n,m}?	X{n,m}+	X, at least n but not more than m times

8

Quantifier Types

- **Greedy**: first, the quantified portion of the expression eats the whole input string and tries for a match. If it fails, the matcher backs off the input string by one character and tries again, until a match is found.
- **Reluctant**: starts to match at the beginning of the input string. Then, iteratively eats another character until the whole input string is eaten.
- **Possessive**: try to match only once on the whole input stream.

9

Example

- **Greedy**:

Current REGEX is: .*foo
Current INPUT is: xfooxxxxxxfoo
I found the text "xfooxxxxxxfoo" starting at index 0 and ending at index 13.

- **Reluctant**:

Current REGEX is: .*?foo
Current INPUT is: xfooxxxxxxfoo
I found the text "xfoo" starting at index 0 and ending at index 4.
I found the text "xxxxxxfoo" starting at index 4 and ending at index 13.

- **Possessive**

Current REGEX is: .*+foo
Current INPUT is: xfooxxxxxxfoo
No match found.

10

Groups

- With parentheses, we can create groups to apply quantifiers to several characters: “(abc)+”
- Also useful for parsing results (see last slide)
- Groups are numbered by counting their opening parentheses from left to right
- Example: groups in “((A)(B(C)))”
  1. ((A)(B(C)))
  2. (A)
  3. (B(C))
  4. (C)

11

Boundary matchers

^	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

Current REGEX is: \bdog\b
Current INPUT is: The doggie plays in the yard.
No match found.

12

RegExps in Java

- Two important classes:
  - **java.util.regex.Pattern** -- a compiled representation of a regular expression
  - **java.util.regex.Matcher** -- an engine that performs match operations by interpreting a Pattern
- Example

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

- ! To produce a slash in a Java String: "//"

13

Example

```
import java.util.regex.*;
public class RegEx{
   public static void main( String args[] ){
      String amounts = "$1.57 $316.15 $19.30 $0.30 $0.00 $41.10 $5.1 $.5";
      Pattern strMatch = Pattern.compile( "\\$(\\d+)\\.(\\d\\d)" );
         Matcher m = strMatch.matcher( amounts );
         while ( m.find() ){
            System.out.println( "$" + ( Integer.parseInt( m.group(1) ) + 5 )
              + "." + m.group(2) );
         }
   }
}
```

=> Adds 5$ to every amount except the last two

14