A Quick Introduction to Regular Expressions in Java

Lecture 10a

RegEx

Readings

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

• Java.util.regex API
  http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/package-summary.html
Regular Expressions

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

Motivating Example

• “I want to find all book titles that contain the word JDBC”
  – Go through all strings and search for “JDBC”
  – In file system we would do: `ls *JDBC*`
## Basic Syntax

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

## 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.
Character Classes

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[abc]</td>
<td>a, b, or c (simple class)</td>
</tr>
<tr>
<td>^abc</td>
<td>Any character except a, b, or c (negation)</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>a through z, or A through Z, inclusive (range)</td>
</tr>
<tr>
<td>[a-d[m-p]]</td>
<td>a through d, or m through p: [a-dm–p] (union)</td>
</tr>
<tr>
<td>[a-z&amp;&amp;[def]]</td>
<td>d, e, or f (intersection)</td>
</tr>
<tr>
<td>[a-z&amp;&amp;[^bc]]</td>
<td>a through z, except for b and c: [ad–z] (subtraction)</td>
</tr>
<tr>
<td>[a-z&amp;&amp;[^m–p]]</td>
<td>a through z, and not m through p: [a–lq–z] (subtraction)</td>
</tr>
</tbody>
</table>

Predefined Character Classes

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Any character (may or may not match line terminators)</td>
</tr>
<tr>
<td>\d</td>
<td>A digit: [0–9]</td>
</tr>
<tr>
<td>\D</td>
<td>A non-digit: [^0–9]</td>
</tr>
<tr>
<td>\s</td>
<td>A whitespace character: [ \t\n\x0B\f\r]</td>
</tr>
<tr>
<td>\S</td>
<td>A non-whitespace character: [^\s]</td>
</tr>
<tr>
<td>\w</td>
<td>A word character: [a-zA-Z_{0–9}]</td>
</tr>
<tr>
<td>\W</td>
<td>A non-word character: [^\w]</td>
</tr>
</tbody>
</table>
### Quantifiers

<table>
<thead>
<tr>
<th>Greedy</th>
<th>Reluctant</th>
<th>Possessive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X?</td>
<td>X??</td>
<td>X?+</td>
<td>X, once or not at all</td>
</tr>
<tr>
<td>X*</td>
<td>X*?</td>
<td>X*+</td>
<td>X, zero or more times</td>
</tr>
<tr>
<td>X+</td>
<td>X+?</td>
<td>X++</td>
<td>X, one or more times</td>
</tr>
<tr>
<td>X{n}</td>
<td>X{n}?</td>
<td>X{n}+</td>
<td>X, exactly n times</td>
</tr>
<tr>
<td>X{n,}</td>
<td>X{n,}?</td>
<td>X{n,}+</td>
<td>X, at least n times</td>
</tr>
<tr>
<td>X{n,m}</td>
<td>X{n,m}?</td>
<td>X{n,m}+</td>
<td>X, at least n but not more than m times</td>
</tr>
</tbody>
</table>

### Quantifier Types

- **Greedy**: first, the quantified portion of the expression reads in 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. *(opposite of greedy)*
- **Possessive**: try to match only once on the whole input stream.
Example

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

- **Reluctant:**
  Current REGEX is: .*?foo
  Current INPUT is: xfooxxxxxxxxfoo
  I found the text "xfoo" starting at index 0 and ending at index 4.
  I found the text "xxxxxfoo" starting at index 4 and ending at index 13.

- **Possessive**
  Current REGEX is: .*+foo
  Current INPUT is: xfooxxxxxxxxfoo
  No match found.

Groups

- With parentheses, we can create groups to apply quantifiers to several characters: “(abc)⁺”
  - Treat multiple characters as a unit
- 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)
Boundary matchers

Search at particular location in the string

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>The beginning of a line</td>
</tr>
<tr>
<td>$</td>
<td>The end of a line</td>
</tr>
<tr>
<td>\b</td>
<td>A word boundary</td>
</tr>
<tr>
<td>\B</td>
<td>A non-word boundary</td>
</tr>
<tr>
<td>\A</td>
<td>The beginning of the input</td>
</tr>
<tr>
<td>\G</td>
<td>The end of the previous match</td>
</tr>
<tr>
<td>\Z</td>
<td>The end of the input but for the final terminator, if any</td>
</tr>
<tr>
<td>\z</td>
<td>The end of the input</td>
</tr>
</tbody>
</table>

Examples

Current REGEX is: ^dog$ // beginning line, end line
Current INPUT is: dog
I found the text "dog" starting at index 0 and ending at index 3.

Current REGEX is: ^dog$
Current INPUT is: dog
No match found.

Current REGEX is: \s*dog$ // white spaces
Current INPUT is: dog
I found the text "dog" starting at index 0 and ending at index 15.

Current REGEX is: ^dog\w* // word char.
Current INPUT is: dogblahblah
I found the text "dogblahblah" starting at index 0 and ending at index 11.
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
  ```java
  Pattern p = Pattern.compile("a*b");
  Matcher m = p.matcher("aaaaab");
  boolean b = m.matches();
  ```
- ! To produce a slash in a Java String: “/”

Simple Example

```java
import java.util.regex.*;

public final class MatcherTest {

  private static final String REGEX = "\\bdog\\b";
  private static final String INPUT = "dog dog dog doggie dogg";

  public static void main(String[] argv) {
    Pattern p = Pattern.compile(REGEX);
    Matcher m = p.matcher(INPUT); // get a matcher object
    int count = 0;
    while(m.find()) {
      count++;
      System.out.println("Match number "+count);
      System.out.println("start(): "+m.start());
      System.out.println("end(): "+m.end());
    }
  }
}
```
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

//Checks for email addresses starting with inappropriate symbols like dots or @ signs.
Pattern p = Pattern.compile("\\.\\|\\@");
Matcher m = p.matcher(input);
if (m.find()){
    System.err.println("Email addresses don't start" + " with dots or @ signs.");
    //Checks for email addresses that start with www. and prints a message if it does.
    p = Pattern.compile("www\\.");
    m = p.matcher(input);
    if (m.find()){
        System.out.println("Email addresses don't start" + " with "www\.", only web pages do.");
    }
    p = Pattern.compile("[^A-Za-z0-9\\.\\@\-_--#]+");
    m = p.matcher(input);
    StringBuffer sb = new StringBuffer();
    boolean result = m.find();
    boolean deletedIllegalChars = false;
    while(result){
        deletedIllegalChars = true;
        m.appendReplacement(sb , "");
        result = m.find();
    }
    // Add the last segment of input to the new String
    m.appendTail(sb);
    input = sb.toString();
    if (deletedIllegalChars) {
        System.out.println("It contained incorrect characters" + " , such as spaces or commas.");
    }
}
Summary

• Regular expressions are a powerful way to search for characters in strings
• Can be used in several different programming languages (e.g. Perl)
• Generally applicable