CS601: Principles of Software Development

Regular Expressions.

Olga A. Karpenko

Parts of this presentation is based on the materials of Prof. Engle.

Announcements

- Lab 4 is out, due on Wednesday
- Code camps today:

  3:30-4:30 and 4:30-5:30 (in HR411)
- No instructor's office hours today
  - Bhargavi has office hours 2:15-3:30
  - Go to the tutors or post on Piazza
- Midterm on Friday, in class

Regular Expression

- A “pattern” that can be applied to text

```
Ex: word.replaceAll("[^a-zA-Z0-9]", "");
```

- Useful for manipulating text
  - Check that data is in a certain format
  - Validate Input
  - Extract data

Example

- "[a-z]+" will match a sequence of one or more lowercase letters
  - [a-z] means any character from a through z
  - + means "once or more"

Some simple patterns

.	matches any character
abc	exactly this sequence of three letters
[abc]	any *one* of the letters a, b, or c
[^abc]	any *one* character *except* a, b, c
[a-z]	any *one* character from a through z
[a-zA-Z0-9]	any *one* letter or digit

Some predefined character classes

\d	a digit: [0-9]
\D	a non-digit: [^0-9]
\s	a whitespace character
\S	a non-whitespace character: [^\s]
\w	a word character: [a-zA-Z_0-9]
\W	a non-word character: [^\w]

Sequences and alternatives

- One pattern followed by another -> the two patterns must match consecutively
  - Ex: [A-Za-z]+[0-9] will match one or more letters immediately followed by one digit
- The vertical bar, |, is used to separate alternatives
  - Ex: the pattern abc|xyz will match either abc or xyz

Greedy Quantifiers

- Assume *X* represents some pattern

X?	optional, *X* occurs once or not at all
*X**	*X* occurs zero or more times
X+	*X* occurs one or more times

Greedy Quantifiers

- Assume *X* represents some pattern

X{*n*}	*X* occurs exactly *n* times
X{*n*,}	*X* occurs *n* or more times
X{*n*,*m*}	*X* occurs at least *n* but not more than *m* times

Examples

- Username regular expression
  - Can contain lowercase letters, digits, underscore , hyphen
  - Length at least 3 characters and maximum 15 characters

Examples

- Username regular expression
  - Can contain lowercase letters, digits, underscore , hyphen
  - Length at least 3 characters and maximum 15 characters

[a-z0-9_-]{3,15}

Examples

- Email regular expression
  - Letters, digits, underscore, followed by @
  - After @ - letters and numbers, then dot, then at least two letters

http://www.mkyong.com/regular-expressions/10-java-regular-expression-examples-you-should-know/

Examples

- Email regular expression
  - Letters, digits, underscore, followed by @
  - After @ - letters and numbers, then dot, then at least two letters

```
[_A-Za-z0-9]+@[A-Za-z0-9]+(\\.[A-Za-z]{2,})
```

http://www.mkyong.com/regular-expressions/10-java-regular-expression-examples-you-should-know/

Example

- InputValidationExample.java
- Uses matches() method in class String
  - Ok, if using the pattern only once
  - If using the pattern again, should use Pattern/ Matcher

Pattern and Matcher

- A Pattern:
  - a compiled representation of a regular expression
- A Matcher:
  - interprets the pattern & matches it against text

http://www.tutorialspoint.com/java/java_regular_expressions.htm

Matching a Pattern in Java

- Import `java.util.regex;`
- Compile the pattern

```
Pattern p = Pattern.compile("[a-z]+");
```

- Create a matcher for a specific piece of text :

```
Matcher m = p.matcher("Now is the time");
```

- Use it to match the pattern
  - different options

Applying the Pattern

- Pattern: "[a-z]+"
- Text: "Now is the time"

  3 ways to apply this pattern:

  1. To the entire string: it fails to match
  2. To the beginning of the string: it fails to match
  3. "Somewhere" in the string: it will succeed and match ow
     - If applied repeatedly, it will find: is, the, time

Regular Expressions in Java

1. `m.matches()`

   true if matches the entire text string

2. `m.lookingAt()`

   true if matches at the beginning of the text string

3. `m.find()`

   true if the pattern matches any part of the text string

   - Called again -> will start from where the last match was found

After a Successful Match

- `m.start()` index of the first character matched
- `m.end()` index of the last character matched, *plus one*

If a Match is Unsuccessful

- `m.start()` and `m.end()` will throw an `IllegalStateException`

A Complete Example

```
import java.util.regex.*;
public class RegexTest {
  public static void main(String args[]) {
    String pattern = "[a-z]+";
    String text = "Now is the time";
    Pattern p = Pattern.compile(pattern);
    Matcher m = p.matcher(text);
    while (m.find()) {
      String s = text.substring(m.start(), m.end());
      System.out.print(s + "*");
    }
  }
}
```

Output: ow*is*the*time*

Additional methods

- `m.replaceFirst(replacement)`

  Returns a new String where the first substring matched by the pattern has been replaced by replacement

- `m.replaceAll(replacement)`

  Returns a new String where every substring matched by the pattern has been replaced by replacement

Additional methods

- `m.reset(newText)`

  Resets this matcher and gives it new text to examine

  - Text: A String, a StringBuffer, or a CharBuffer

- m.group()

  For capturing groups

Capturing groups in Java

- If m is a matcher that has just performed a successful match:
  - `m.group(`*n*`)`

    Returns the String matched by capturing group *n*

  - `m.group()`

    Returns the String matched by the entire pattern

Example use of capturing groups

- Suppose word holds a word in English
- Move all the consonants at the beginning of word (if any) to the end of the word
  - Ex: string becomes ingstr

```
Pattern p = Pattern.compile("([^aeiou]*)(.*)");
Matcher m = p.matcher(word);
if (m.matches()) {
   System.out.println(m.group(2) + m.group(1));
}
```

- (.*) - "all the rest of the characters"

Groups

- Numbered by counting their opening parentheses from left to right
- Example: ((A)(B(C))), four groups:

```
((A)(B(C)))
(A)
(B(C))
(C)
```

http://docs.oracle.com/javase/tutorial/essential/regex/groups.html