CS601: Principles of Software Development

More on Regular Expressions.

Olga A. Karpenko

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

Announcements

- Lab 4 due tonight
- Midterm exam on Friday
  - Groups are on the exam
  - Greedy/reluctant/possessive quantifiers
- Lab 3 Part 1 has been graded
  - If you have no grade, you need to see me

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

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

Types of quantifiers

- A greedy quantifier will match as much as it can, and back off if it needs to
- A reluctant quantifier will match as little as possible, then take more if it needs to
  - You make a quantifier reluctant by appending a ?
    Examples: *X**? *X*{*n*,}?
- A possessive quantifier will match as much as it can, and never let go
  - You make a quantifier possessive by appending a +:
    *X**+ *X*++ *X*{*n*,}+

Quantifier examples

- Text “aardvark”
- Using the pattern a*ardvark
  - (a* is greedy):
  - The a* will first match aa -> ardvark won’t match
  - The a* then “backs off” -> matches only a single a
    - that allows the rest of the pattern (ardvark) to succeed

Quantifier examples

- Text “aardvark”
- Using the pattern a*?ardvark
  - (a*? is reluctant):
  - a*? will first match zero characters -> ardvark won’t match
  - a*? then extends and matches the first a
    - the rest of the pattern (ardvark) matches

Quantifier examples

- Text “aardvark”
- Using the pattern a*+ardvark
  - (a*+ is possessive):
  - a*+ will match the aa,
  - Will not back off -> ardvark never matches
  - The pattern match fails

Example: what will be printed?

```
String str = "Hello: This is a Test:";

Pattern p1 = Pattern.compile("(.*):");
Pattern p2 = Pattern.compile("(.*?):");

Matcher m1 = p1.matcher(str);
if (m1.find()) {
        System.out.println(m1.group(1));
}

Matcher m2 = p2.matcher(str);
if (m2.find()) {
        System.out.println(m2.group(1));
}
```

http://stackoverflow.com/questions/12248793/pattern-compile

Example

- See `MultipleGroupsExample.java`, `GroupsExample.java`

Case Insensitive

- i flag
- Non-capturing group

  `(?i)[a-z]+`

- Alternative:

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

Spaces

- Spaces are significant!
- A space stands for a *space*
  - Space in a pattern = a space in the text string
  - Do not put spaces in a regular expression to make it look better

Thinking in regular expressions

- Regular expressions are *not* easy to use at first
  - A bunch of punctuation, not words
  - The individual pieces are not hard, but
  - Takes practice to learn to put them together correctly
- Make string manipulation easy, very powerful

Escaping metacharacters

- A lot of special characters used in regex:
  - parentheses, brackets, stars, plus signs, etc.
  - called metacharacters

- Ex: Search for an “a” followed by a “star”
  - "a*"; doesn’t work; that means “zero or more a letters”
  - "a*" works

Practice regex

- Circle strings that match the following regular expression:

(very)+(nice)?(kind|generous) person

A. very nice person
B. nice kind person
C. very very nice generous person
D. very very very kind person

Practice regex

- Consider the following regex describing a substring:
  ([A-Z])([\\d]{2,4})
- Write a code snippet that changes the input string so that *for each substring*, digits come before letters
- Example: "A25 K150 Z228 D4679 J67"
  "25A 150K 228Z 4679D 67J"
- Hints:
  - compile a Pattern p,
  - create a Matcher m for the pattern and a given text,
  - Call find() in a loop to find all matches
  - Use m.group(1) and m.group(2) to refer to different parts of the string that was matched to group 1 and group 2 in the regex
- See RegexPracticeExercise.java